Image Mosaicing 2

COSC342

Lecture 7
21 March 2017

Mosaicing So Far

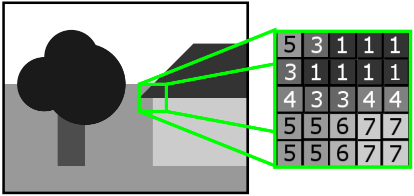
- Mosaicing is defined by a homography, $\mathbf{p}' = \mathrm{H}\mathbf{p}$
  - H is a 3×3 matrix, defined up to a scale
  - Can compute a homography from 4 corresponding points
- Features are points which can be accurately located in images
  - Corners – points with high gradient in all directions
  - Blobs – have a location and a characteristic scale
- Given features in two images, how do we find a correspondence?

Feature Descriptors

- Features are matched on the basis of some descriptor
- This is a list of numbers, represented as a vector
  - Typically this is a high-dimensional vector
  - SIFT descriptors, for example, have 128-dimensions
- The distance between matching vectors should be small
- The distance should be low regardless of changes in the image
  - Translation and rotation in the image plane
  - Changes in viewing direction
  - Changes in scale
  - Changes in lighting and brightness

A Simple Feature Descriptor

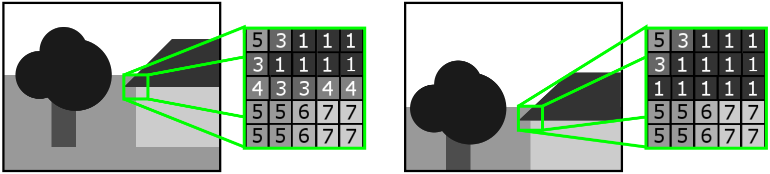
- We could use the pixel values in a window around the feature
  - This is easy to compute, and works well in some cases
  - For simplicity we'll use greyscale images
  - Generalises easily to colour images
- If we take a $n \times n$ window, we get a vector of n^2 values
- We can compare them with the usual (Euclidean) vector distance



$$(5, 3, 1, 1, 1, 3, 1, 1, 1, 1, 4, 3, 3, 4, 4, 5, 5, 6, 7, 7, 5, 5, 6, 7, 7)$$

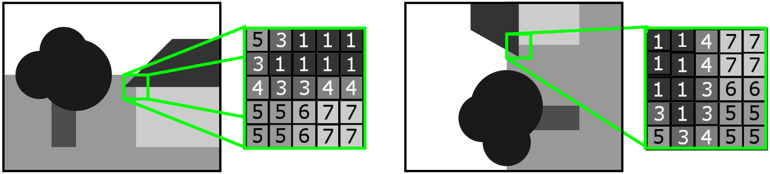
Feature Invariance

- Translation



$$|(0,0,0,0,0,0,0,0,0,0,3,2,2,3,3,0,0,0,0,0,0,0,0,0,0)| = \sqrt{35}$$

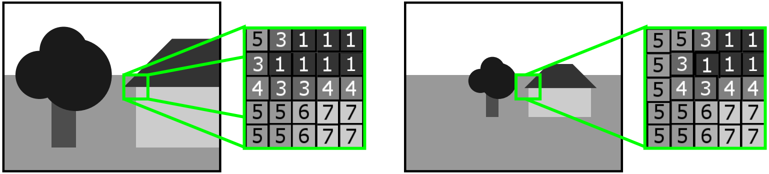
- Rotation



$$|(4,2,-3,-6,-6,-2,0,-3,-6,-6,\ldots,0,2,2,2,2)| = \sqrt{260}$$

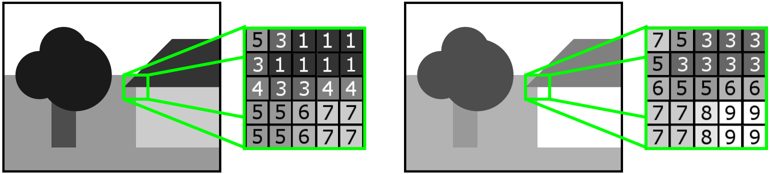
Feature Invariance

- Scale



$$|(0, -2, -2, 0, 0, -2, -2, 0, 0, 0, -1, -1, 0, 0 \ldots, 0, 0)| = \sqrt{18}$$

- Brightness changes



$$|(-2, -2, -2, -2, -2, -2, -2, -2, -2, -2 \ldots, -2, -2)| = \sqrt{100}$$

SIFT Features

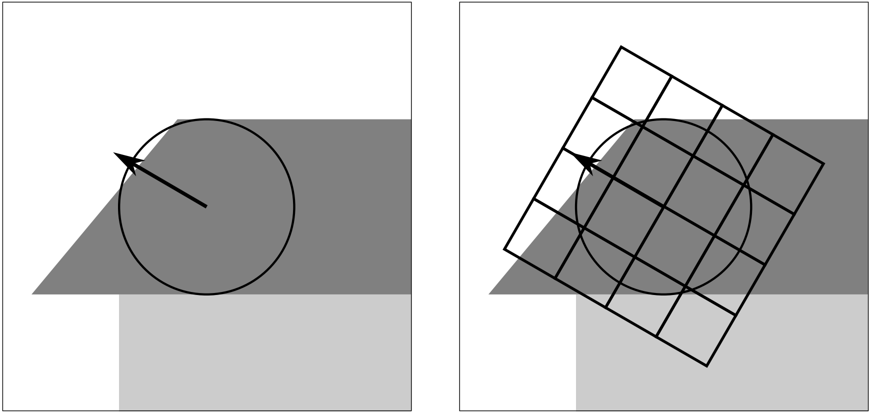
- In 1999 David Lowe proposed an invariant feature detector[1]
- Translation invariance is easy, as we've seen
- Scale invariance comes from using blob features
  - Descriptor is computed from a window around the feature
  - The size of the blob determines the size of the window
- Brightness invariance comes from using image gradients
  - The relative brightness of pixels is fairly constant
  - Gradients do not change much under moderate intensity change
- Rotation invariance comes from finding a dominant gradient direction
  - The window is oriented to the dominant gradient

[1] D. G. Lowe, *Object recognition from local scale-invariant features*, ICCV 1999

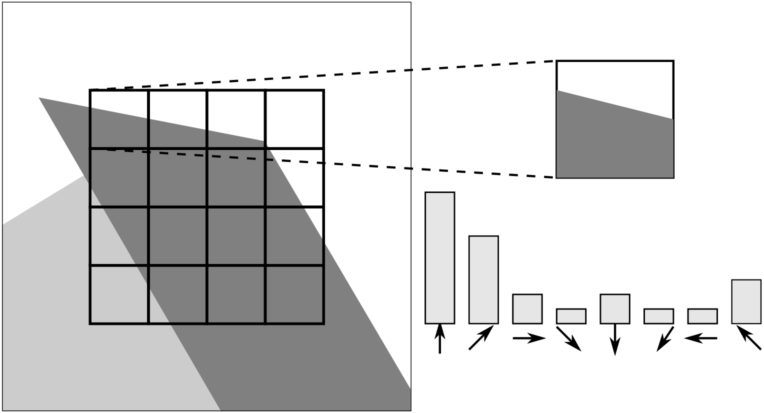
SIFT Features

- Blob features are detected and their scale determined
- A histogram of gradients around the blob are computed
- Peak(s) in the histogram determine the orientation
- A square region is used to compute the descriptor
  - The size of the square comes from the size of the blob
  - The square is aligned to the feature's orientation
- This region is divided into a 4×4 grid of squares
- In each sub-region a gradient histogram is made with 8 bins
- This gives $4 \times 4 \times 8 = 128$ values, which is the descriptor

SIFT Features



SIFT Features

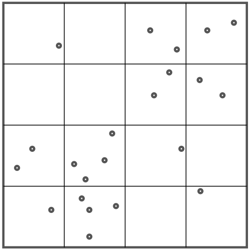


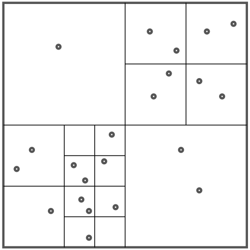
Matching Features

- The final descriptor is 128 values, usually bytes
  - Finding the distance between two descriptors takes 256 operations
  - OK to compute squared difference (no square root needed)
- If we find 10,000 features in each image
  - Matching one feature takes $\sim 2,500,000$ operations
  - Matching all features takes $\sim 25,000,000,000$ operations
- This is often too expensive, so approximate methods are used

Space Subdivision and Approximate Neighbours

- Split space into smaller regions
- 2D examples easier to draw...
- Uniform subdivision
  - Division into regular grid
  - Look for neighbours in the same cell as the point we are matching
- Quadtrees, octrees, etc.
  - Recursively split in half
  - Stop splitting when only a few elements in a cell
  - 2D gives a *quadtree*
    3D gives an *octree*





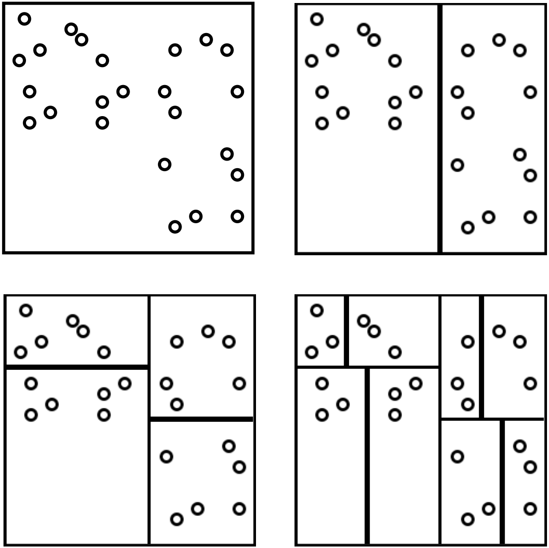
Space Subdivision

- This gets difficult in high dimensions
- Consider uniform subdivision with 8 divisions along each axis
  - In 2D this is $8 \times 8 = 64$ cells
  - In 3D we get $8 \times 8 \times 8 = 512$ cells
  - In nD we get 8^n cells, and $8^{128} \approx 3.9 \times 10^{115}$
- Even if we just have 2 divisions (such as one layer of a generalised quad-/oct-tree), we have $2^{128} \approx 3.4 \times 10^{38}$ cells
- So we can't split along all axes

k-d Trees

- A common solution is the use of k-d trees
- Choose one axis and split the data along it
  - Could choose the axis with the greatest spread
  - Could use the first axis, or a random one
  - Try to split the data roughly in half
- Then take each half and split along another axis
  - The axis could be chosen as above
  - Try to split each cell's data in half
- And repeat until cells have only a few items in them

k-d Trees in 2D



k-d Trees and Feature Matching

- Put all the features in one image into a k-d Tree
- Given a feature from the other image:
  - Find which cell in the k-d Tree it lies in
  - Compute the distance to all features in that cell
  - The nearest one is probably the best match
- For a tree with n layers and 10,000 features this requires:
  - n comparisons to find the appropriate cell
  - $256\frac{10{,}000}{O(2^n)}$ operations in the distance computations
  - If $n = 10$, then $\frac{10{,}000}{O(2^n)} \approx 10$
- This doesn't always find the best match – why not?

Matching SIFT features

- Even if we use brute-force matching most SIFT matches are wrong
  - A lot of blob features don't have much texture detail
  - A lot of scenes have repeating features
  - This leads to ambiguous matches
  - SIFT is often the best we have [2]
- With *k*-d Trees this gets a little worse, but not much
- Solution: Find the two best matches to check for ambiguity
  - Can use other methods to reject unreliable matches[3]
- Only keep matches if the best distance is much lower than the second
- This makes things better, but still some wrong matches
- Need robust methods (next lecture)

[2] N. Kahn, B. McCane, S. Mills *Better than SIFT?*, MVA 26(6), 2015

[3] S. Mills, Relative Orientation and Scale for Improved Feature Matching, ICIP, 2013

Coming up...

- Tutorial this week
  - 2D Transforms again
- Next lecture
  - Robust homography estimation
  - Mosaicing in OpenCV