Computer Vision (7CCSMCVI / 6CCS3COV)

Recap

- **Image formation**
- **Low-level vision**
- **Mid-level vision**
  - grouping and segmentation of image elements
  - Biological
    - bottom-up influences (Gestalt cues)
    - top-down influences (knowledge, expectation, etc.)
  - Artificial
    - thresholding, region-based, clustering, fitting
  - Multi-View Vision
    - correspondence problem ←Today
      - stereo
      - video
- **High-level vision**
  - object recognition

Multiple Images

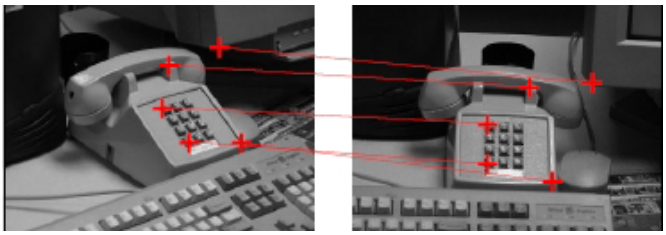
Arise due to:

- multiple cameras (stereo)
  - two, or more, images taken simultaneously by different cameras
- multiple times (video)
  - two, or more, images taken at different times by one camera
- object recognition
  - current image and memorised “training” image(s)

Correspondence Problem

Finding matching image elements across views is fundamental to solving many problems in vision:

- stereo depth recovery
  - finding corresponding points in two images taken by different cameras enables recovery of 3D information
- motion tracking
  - finding corresponding points in two images taken at different times enables estimation of camera and/or object motion
- object recognition
  - finding corresponding points in training and test images enable object recognition



Correspondence Problem

For each selected element in one image, find the corresponding element in the other image.

This is a search problem.

Three main design decisions for correspondence algorithms:

- which elements to match, e.g. intensities, edges, other features.
- how to search for matching elements.
- how to compare elements to confirm/reject match.

Basic requirements to be able to solve the correspondence problem:

1. Most scene points visible in both images
2. Corresponding image regions appear “similar”

Correspondence Problem: problems

Occlusions

some elements may not have a corresponding element due to self-occlusion, occlusion by other object(s), or no longer being “in shot”

False matches

there may be several similar elements, only one of which can be the “true match”

Changing element characteristics

feature values of corresponding elements may differ due to, e.g.:

- change in lighting direction (change in intensity)
- viewpoint differences (change in size and shape)

Large search space

each element in one image has many possible matches in other image.

Methods often employ additional assumptions to constrain search space and resolve ambiguities.

Correspondence Problem: solutions

Algorithms to solve the correspondence problem fall into two classes:

Correlation-based methods

Attempt to establish a correspondence by matching image intensities – usually over a window of pixels in each image

- start from raw image intensity values
- match image windows
- compare them using a similarity measure for intensity values

Feature-based methods

Attempt to establish a correspondence by matching sparse sets of image features

- start from image features extracted by preprocessing
- match image features
- compare them using distance between feature descriptors

Correlation-Based Methods

Matching based on correlating pixel values within image regions.

For each region I_1 in image one

For each region I_2 of same size in search area of other image

Compute similarity between I_1 and I_2.

Repeat for all regions in image two within a search area.

Corresponding point is centre of region giving highest similarity.

Repeat for all regions in image one for which correspondence is required.

Correlation-Based Methods

Need to decide on:

1. Size of correlation window – success depends on window exhibiting a distinctive structure that occurs infrequently in the search region of the other image
   - too small a window
     - may not capture enough image structure to be distinctive, and
     - may be too noise sensitive resulting in *many false matches*
   - too large a window
     - makes matching less sensitive to noise (desired) but also
     - decreases precision (blurs correspondence map)
2. Size of search area – full correlation of all pixels is computationally expensive. Therefore usually constrained either:
   - arbitrarily to be around pixel location from which I_1 taken, or by
   - explicit knowledge of task (e.g. using epipolar geometry in stereo correspondence problem [next lecture]).
3. Method used to measure similarity.

Similarity Measures

We can **maximise** the following measures:

Cross-correlation: $\sum_{i,j} I_1(i,j) I_2(i,j)$

If I_1 and I_2 are considered to be vectors in feature space, then the cross-correlation is the dot-product of these vectors $I_1 . I_2 = \|I_1\| \, \|I_2\| \cos\theta$

Normalised cross-correlation: $$\frac{\sum_{i,j} I_1(i,j) I_2(i,j)}{\sqrt{\sum_{i,j} I_1(i,j)^2} \sqrt{\sum_{i,j} I_2(i,j)^2}} = \frac{I_1 . I_2}{\|I_1\| \, \|I_2\|} = \cos\theta$$

Correlation coefficient: $$\frac{\sum_{i,j} \left(I_1(i,j) - \bar{I}_1\right)\left(I_2(i,j) - \bar{I}_2\right)}{\sqrt{\sum_{i,j} \left(I_1(i,j) - \bar{I}_1\right)^2} \sqrt{\sum_{i,j} \left(I_2(i,j) - \bar{I}_2\right)^2}}$$

equals normalised cross-correlation if means are zero

Similarity Measures

We can **minimise** the following measures:

Sum of Squared Differences (SSD): $\sum_{i,j} \left(I_1(i,j) - I_2(i,j)\right)^2$

Euclidean distance: $\sqrt{SSD} = \sqrt{\sum_{i,j} \left(I_1(i,j) - I_2(i,j)\right)^2}$

Sum of Absolute Differences (SAD): $\sum_{i,j} |I_1(i,j) - I_2(i,j)|$

In practice, SAD is often used as it is simple and the increased computational costs of using other measures is usually not justified in terms of improved performance.

Correlation-Based Methods: performance

Advantages

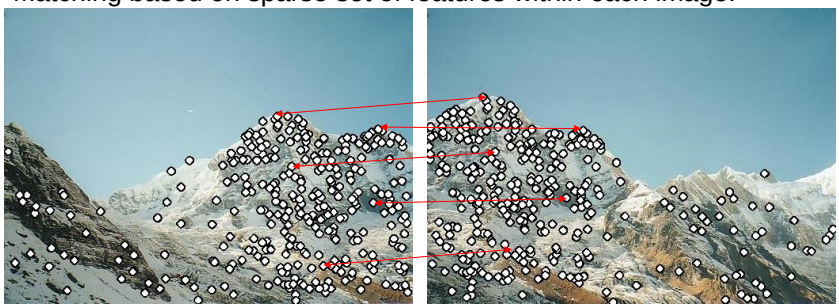
- Easy to implement
- Provides a dense correspondence map (i.e. we calculate correspondence at all points, not just a few)

Disadvantages

- Computationally expensive
- Needs 'busy' images (lots of distinct intensity patterns) to work well
  - regions that are constant or repetitive give many false matches
- Doesn't work well when viewpoints are very different, due to
  - change in illumination
    - changes image intensity values
  - foreshortening
    - changes size/shape/appearance of corresponding regions.

Feature-Based Methods

Matching based on sparse set of features within each image.



Detect interest points in both images

Find corresponding pairs of points:

for each interest point in image one
  for each interest point in search area of other image
    compute similarity between features
  repeat for all interest points in image two within a search area.
  select the interest point in image two that maximizes the similarity measure
repeat for each interest point in image one

Feature-Based Methods: performance

Advantages

- Relatively insensitive to illumination and size changes
- Less computationally expensive than correlation-based methods (only need to match selected locations rather than every pixel)

Disadvantages

- Provides sparse correspondence map
  - if correspondence required at intermediate locations this needs to be interpolated
  - sufficient for many applications
- Only suitable when “good” interest points can be extracted from the scene
  - doesn’t perform well with textured/random regions
  - choice of interest points is important

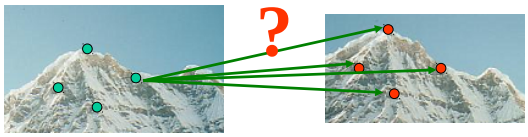
Feature-Based Methods: interest points

1. Need to detect the *same* point *independently* in both images

We need a repeatable detector

Interest points should be invariant to image scaling, rotation, translation (caused by changing 3D camera viewpoint) and to change in illumination

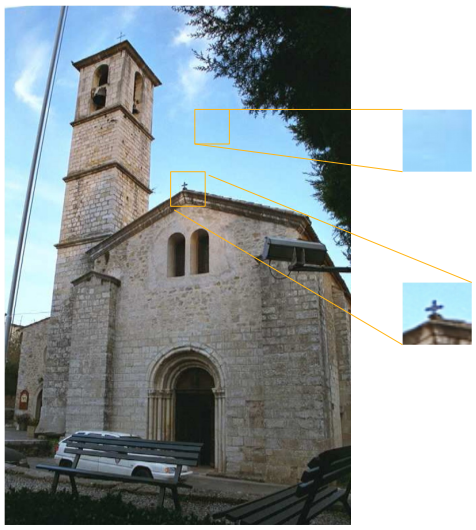
2. Need to correctly recognize corresponding points in both images



We need a distinctive descriptor

Feature descriptors should be sufficiently complex so that corresponding points can be correctly matched with high probability.

Feature-Based Methods: interest points



Hard to localize
Low information content
BAD interest point

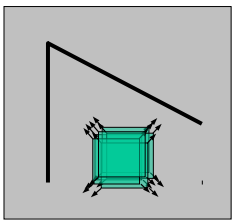
Easy to localize
High information content
GOOD interest point

Interest points: corners

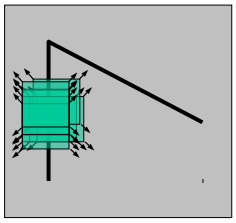
Corners provide repeatable points that are well suited for matching.

A corner = a point where two edges meet

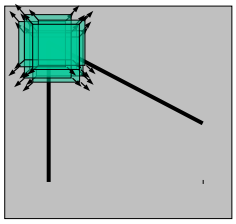
Hence, in the region around a corner, the intensity gradient is high in two directions



"flat" region: no change in all directions



"edge": no change along the edge direction



"corner": significant change in all directions

Corner Detection

Compute intensity gradients (I_x and I_y) in x and y directions by convolving image with derivative of Gaussian masks

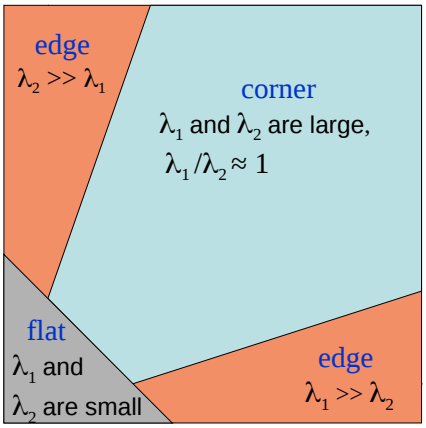
Sum gradients over a small neighbourhood (Gaussian window) around each pixel to generate Hessian matrix H

$$H = \begin{pmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{pmatrix}$$

Find eigenvalues, λ_1 and λ_2, of H.

Eigenvalues correspond to maximum slope of intensity gradient at two orthogonal directions.

Corner is where $\min(\lambda_1, \lambda_2) >$ threshold



Harris corner detector

Avoids calculating eigenvalues. Defines a measure, R, as:

$$R = det(H) - k(trace(H))^2$$

$$R = \left[\sum I_x^2 \sum I_y^2 - \left(\sum I_x I_y\right)^2\right] - k\left[\sum I_x^2 + \sum I_y^2\right]^2$$

(k = 0.04-0.06 found to work empirically)

R is large for a corner

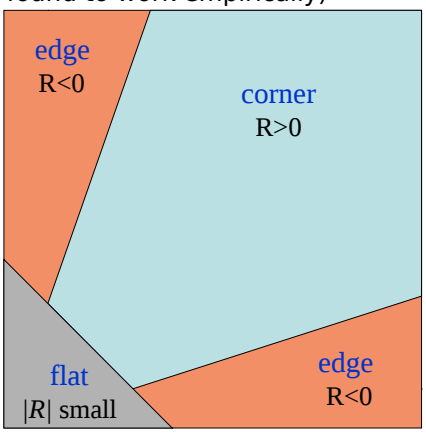
R is negative with large magnitude for an edge

|*R*| is small for a flat region

Hence,

Corner is where *R* > threshold

Take the points of local maxima of *R* as interest points



Harris corner detector

Take the points of local maxima of *R* as interest points.

Use non-maximum suppression:

For each pixel, if it has a neighbour with a larger R value, set its value to 0.

1	2	2	2
0	1	4	3
0	2	2	2
0	1	1	0

→

0	0	0	0
0	0	4	0
0	0	0	0
0	0	0	0

Non-maximum suppression is commonly used in computer vision, not just for interest point detection

Harris corner detector: algorithm

1. Compute x and y derivatives of image

$$I_x = G_\sigma^x * I \quad I_y = G_\sigma^y * I$$

2. Compute products of derivatives at every pixel

$$I_{x2} = I_x.I_x \quad I_{y2} = I_y.I_y \quad I_{xy} = I_x.I_y$$

3. Compute the sums of the products of derivatives at each pixel

$$S_{x2} = G_{\sigma\prime} * I_{x2} \quad S_{y2} = G_{\sigma\prime} * I_{y2} \quad S_{xy} = G_{\sigma\prime} * I_{xy}$$

4. Define at each pixel (x, y) the matrix

$$H(x,y) = \begin{bmatrix} S_{x2}(x,y) & S_{xy}(x,y) \\ S_{xy}(x,y) & S_{y2}(x,y) \end{bmatrix}$$

5. Compute the response of the detector at each pixel

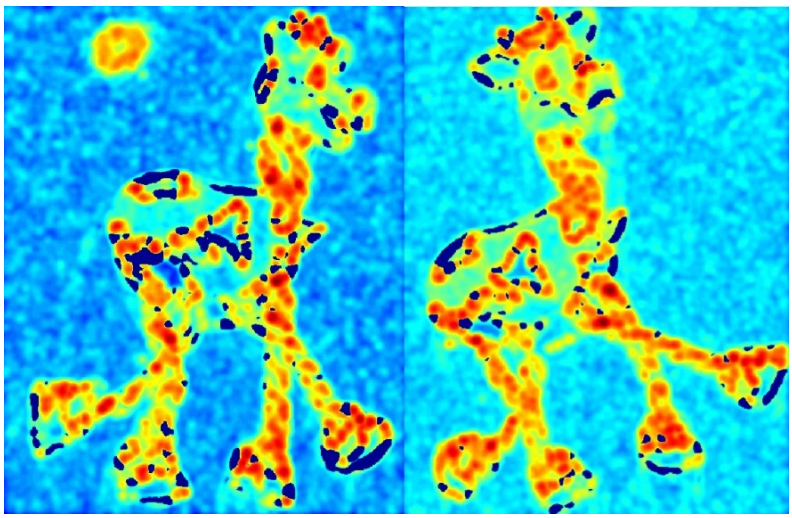
$$R = Det(H) - k(Trace(H))^2$$

6. Threshold on value of R. Compute nonmax suppression.

Harris corner detector: example

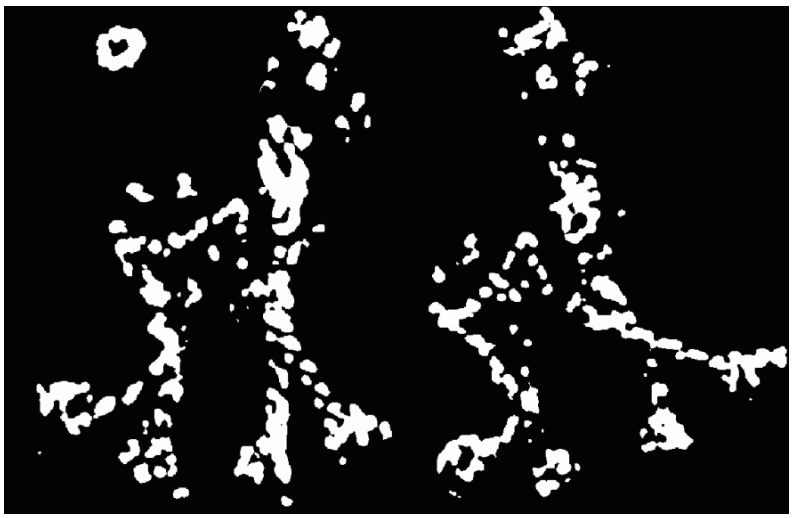
Two images

Harris corner detector: example



Corner response R
red=+ve, black=-ve

Harris corner detector: example



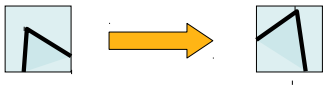
Thresholded R

Harris corner detector: example

Local maxima of R

Interest points: corners

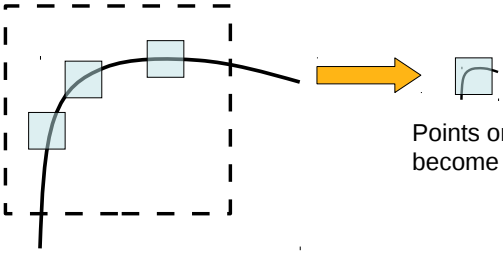
Harris corner detector is translation and rotation invariant



Eigenvectors rotate, but eigenvalues (and R values) remain the same.

Harris corner detector is partly invariant to changes in illumination and to changes in viewpoint.

Harris corner detector is not scale invariant



Points originally classified as edges become corners at coarser scale.

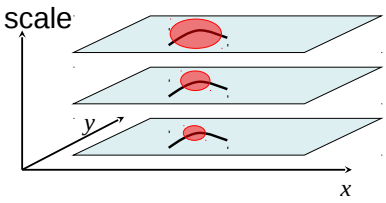
Interest points: scale invariant

To overcome sensitivity to scale, we can perform corner detection across a range of scales using an image pyramid.

Harris-Laplacian

Find local maximum of:

- Harris corner detector in space and scale

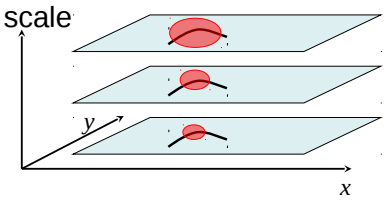


An alternative algorithm for scale invariant interest point detection is the Scale Invariant Feature Transform (SIFT).

SIFT

Find local maximum of:

- Difference of Gaussians in space and scale



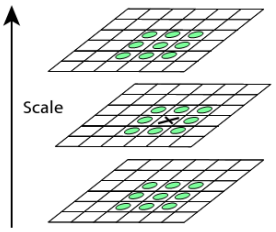
Interest points: scale invariant

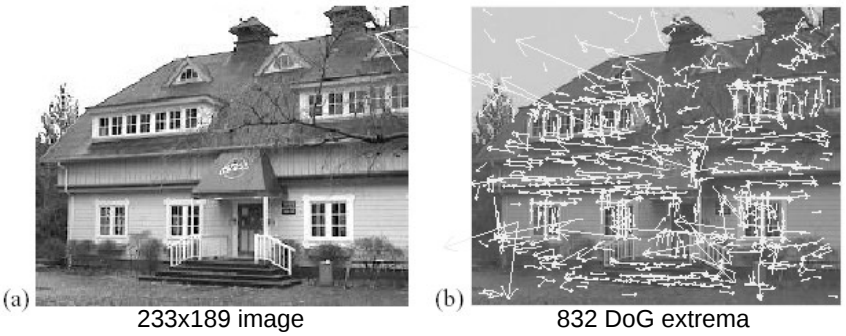
Both the Harris-Laplacian and SIFT algorithms search for the maxima of suitable functions (interest point detectors) in scale and in space

This will enable us to find the same interest points independently in two images which differ in scale

SIFT: interest point detection

- Convolve image with Difference of Gaussians (DoG) mask
- Repeat for different image resolutions (i.e. create a Laplacian image pyramid)
- Detect maxima and minima of difference-of-Gaussian across scale space (x selected if larger or smaller than all 26 neighbours in 3x3x3 neighbourhood)





233x189 image

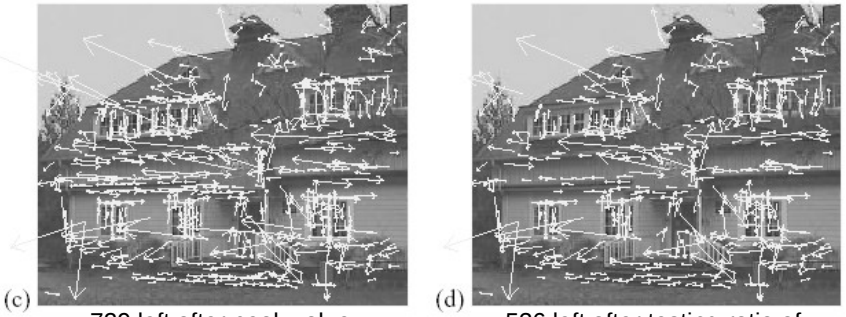
832 DoG extrema

SIFT: interest point detection

- Keep points with high contrast.
- Keep points with sufficient structure: approach similar to Harris corner detector but using ratio of Trace and Determinant of Hessian matrix.

$$\frac{(trace(H))^2}{det(H)} < \frac{(r+1)^2}{r}$$

(r = 10 found to work empirically)

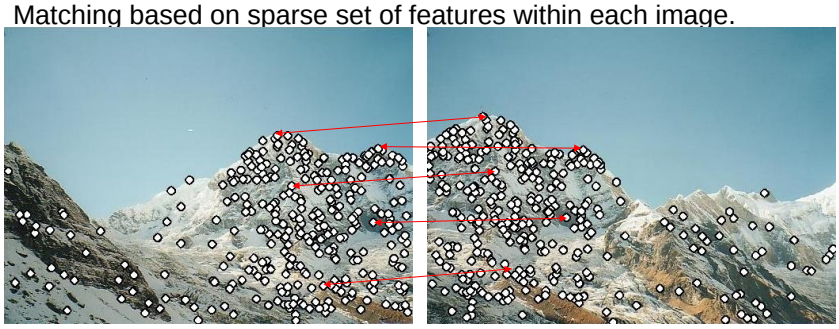


729 left after peak value threshold

536 left after testing ratio of principle curvatures

Feature-Based Methods

Matching based on sparse set of features within each image.



Detect interest points in both images

Harris corner detector, or

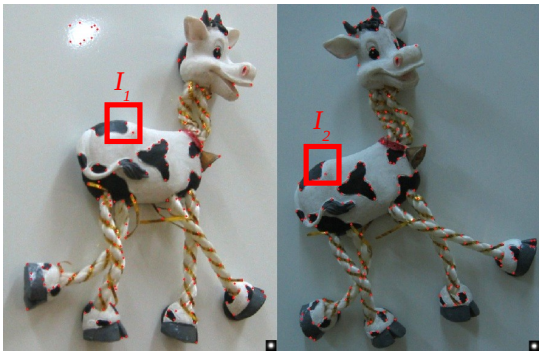
SIFT detector

Find corresponding pairs of points

Requires a measure of similarity between points.

Need a "descriptor" (a list of features) for each interest point.

Harris: feature descriptor and matching



Descriptor: a small window around the interest point (i.e. a set of pixel intensity values).

Similarity measure: Euclidean distance, SSD, SAD, etc.

Robust to translation, but not to rotation, scale, changes in viewpoint or illumination.

SIFT: feature descriptor

Descriptor: the SIFT algorithm specifies a method for deriving a set of features for each interest point.

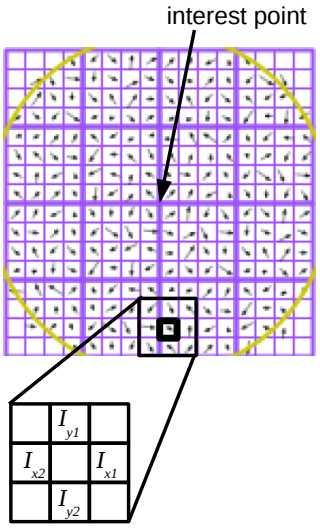
Step 1: Calculate the orientation and magnitude of the intensity gradient at all pixels surrounding the interest point.

This is done using the Gaussian smoothed image at the scale where the interest point was found.

Magnitude and orientation approximated using pixel differences.

$$mag = \sqrt{(I_{x1} - I_{x2})^2 + (I_{y1} - I_{y2})^2}$$

$$ori = \tan^{-1}\left(\frac{I_{y1} - I_{y2}}{I_{x1} - I_{x2}}\right)$$



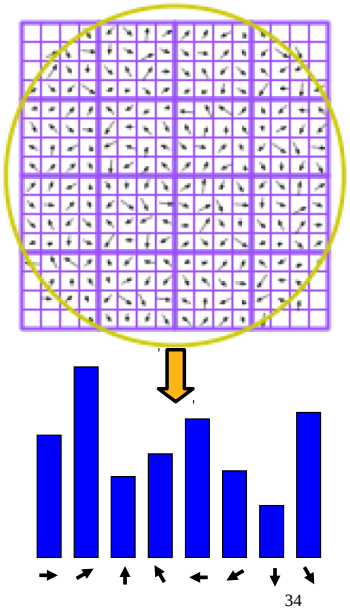
SIFT: feature descriptor

Step 2: Create a histogram of all the orientations around the interest point.

Each sample added to the histogram is weighted by its gradient magnitude and a Gaussian centred on the interest point.

Find dominant orientation, by finding peak in histogram.

Rotate all orientations so that dominant orientation points up.



SIFT: feature descriptor

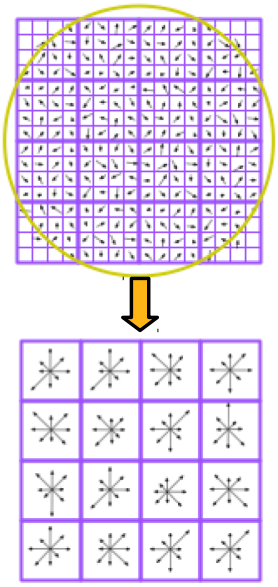
Step 3: Create separate histograms of all the orientations in 4x4 sub-windows around the interest point.

Each sample added to each histogram is weighted by its gradient magnitude and a Gaussian centred on the interest point.

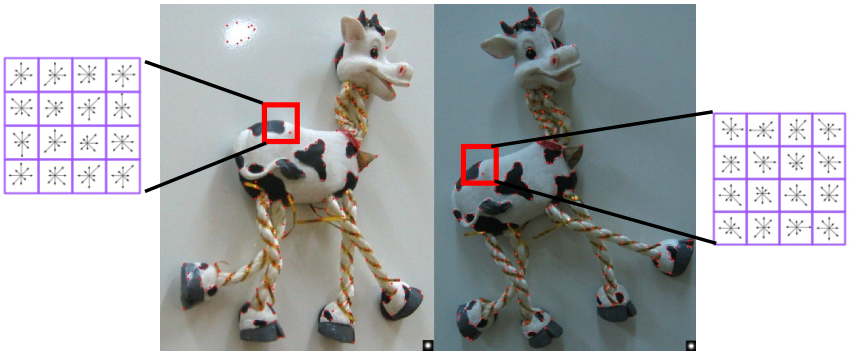
Use 8 orientation bins in each histogram.

The descriptor for each interest point is therefore a 4x4x8 = 128 element vector.

This vector is normalised to unit length.



SIFT: feature descriptor and matching



Descriptor: 128 element vector of intensity gradient orientations around the interest point.

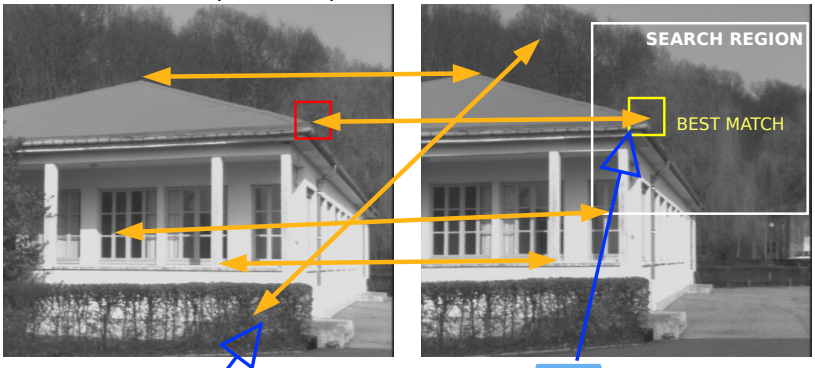
Similarity measure: Euclidean distance between vectors.

Robust to translation, rotation, scale, changes in viewpoint and illumination.

Matching: dealing with outliers

Whether a correlation-based method or a feature-based method is used, search is required to find points that are most similar.

These “most similar” points are *putative matches*

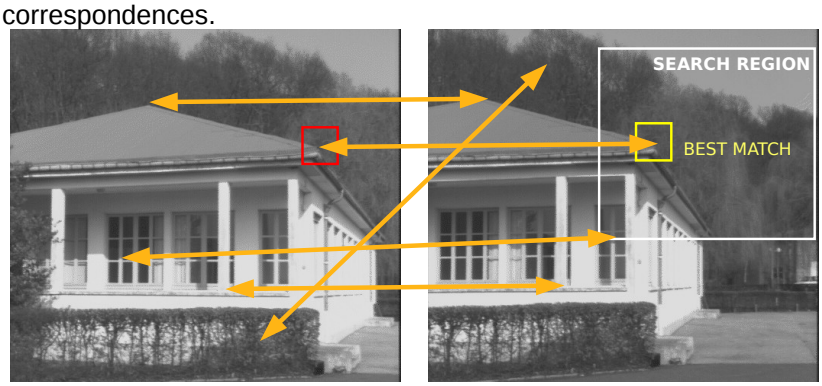


Some of these putative matches may be correct (“inliers”) but others may be wrong (“outliers”).

How do we find the true correspondence between images despite these matching errors?

Matching: dealing with outliers

Need to estimate transformation between images despite erroneous correspondences.



1. (Extract features – if using feature-based method)
2. Compute putative matches
3. Find most likely transformation (i.e. the one with the most inliers and fewest outliers)

   use the **RANSAC** (= RANdom SAmpling & Consensus) algorithm

RANSAC: algorithm

Objective:

Robust fit of model to data set which contains outliers

Requirements:

1. Data consists of inliers and outliers
2. A parameterized model explains the inliers

Procedure:

1. Randomly choose a minimal subset (a *sample*) of data points necessary to fit the model
2. Fit the model to this subset of data
3. Test all the other data points to determine if they are consistent with the fitted model (i.e. if they lie within a distance *t* of the model's prediction)
4. Count the number of inliers (the *consensus* set). Size of consensus set is model's *support*
5. Repeat from step 1 for N trials

After N trials select the model parameters with the highest support and re-estimate the model using all the points in this subset.

RANSAC: simple correspondence example

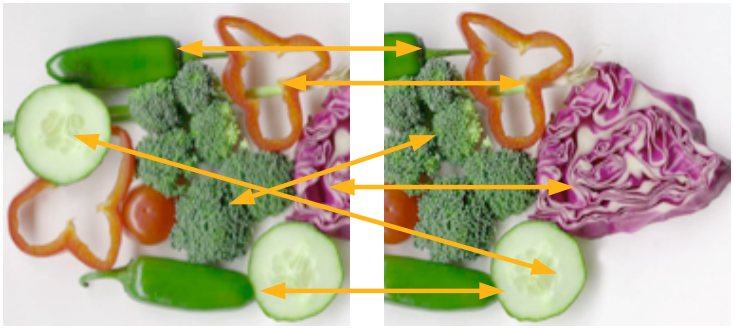
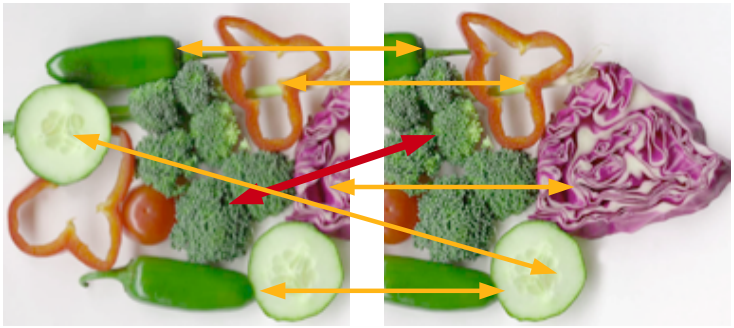


Image shows a set of putative matches between points in two images

Assume the two images are related by a pure translation.
i.e. the model we wish to fit is a translation by Δx and Δy.

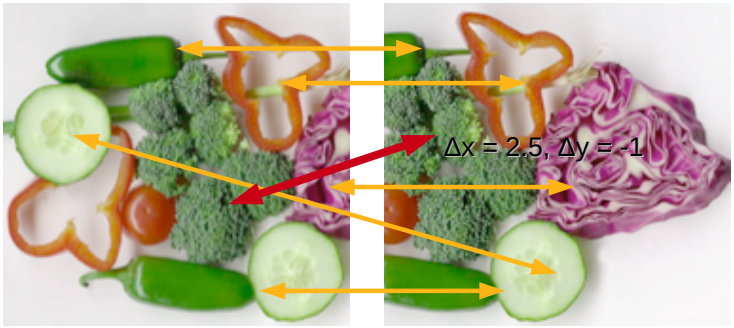
One putative match is sufficient to define Δx and Δy.

RANSAC: simple correspondence example



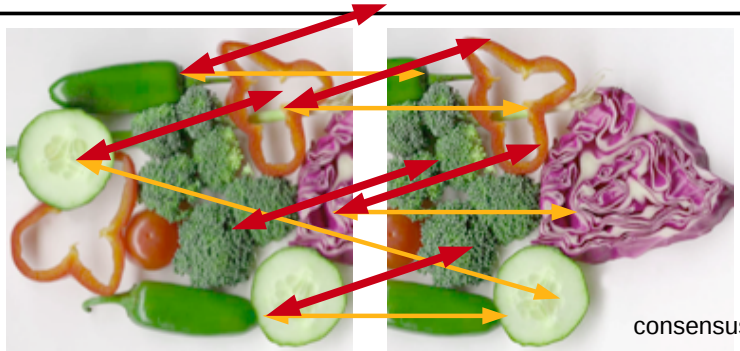
1. Randomly choose a minimal subset (a *sample*) of data points necessary to fit the model
2. Fit the model to this subset of data
3. Test all the other data points to determine if they are consistent with the fitted model (i.e. if they lie within a distance t of the model's prediction).
4. Count the number of inliers (the *consensus* set). Size of consensus set is model's *support*
5. Repeat from step 1 for N trials

RANSAC: simple correspondence example



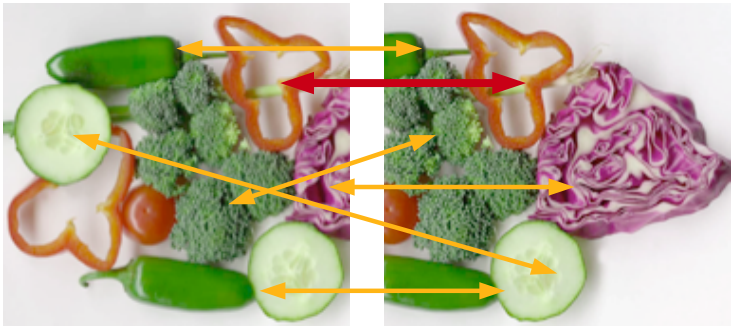
1. Randomly choose a minimal subset (a *sample*) of data points necessary to fit the model
2. Fit the model to this subset of data
3. Test all the other data points to determine if they are consistent with the fitted model (i.e. if they lie within a distance t of the model's prediction).
4. Count the number of inliers (the *consensus* set). Size of consensus set is model's *support*
5. Repeat from step 1 for N trials

RANSAC: simple correspondence example



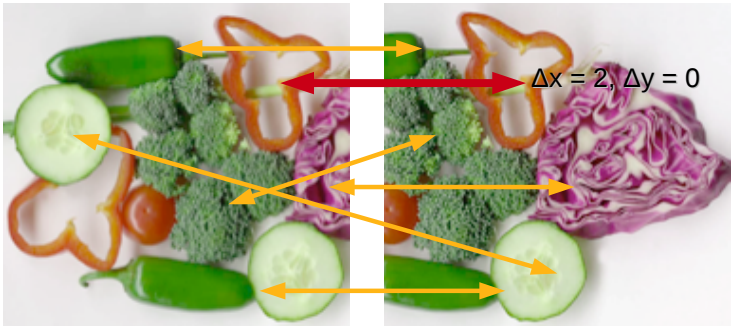
1. Randomly choose a minimal subset (a *sample*) of data points necessary to fit the model
2. Fit the model to this subset of data
3. Test all the other data points to determine if they are consistent with the fitted model (i.e. if they lie within a distance t of the model's prediction).
4. Count the number of inliers (the *consensus* set). Size of consensus set is model's *support*
5. Repeat from step 1 for N trials

RANSAC: simple correspondence example



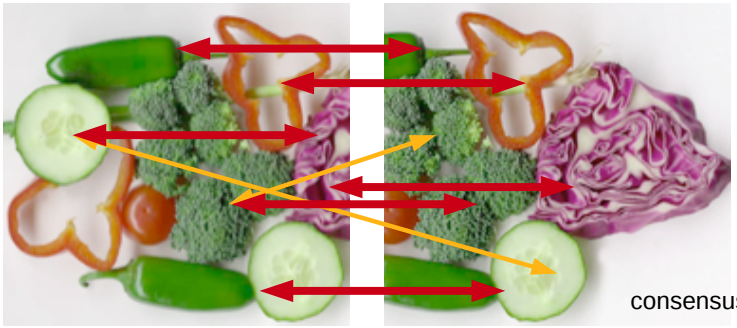
1. Randomly choose a minimal subset (a *sample*) of data points necessary to fit the model
2. Fit the model to this subset of data
3. Test all the other data points to determine if they are consistent with the fitted model (i.e. if they lie within a distance t of the model's prediction).
4. Count the number of inliers (the *consensus* set). Size of consensus set is model's *support*
5. Repeat from step 1 for N trials

RANSAC: simple correspondence example



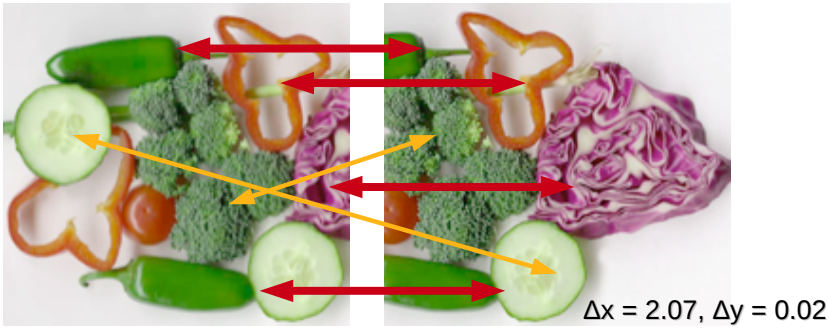
1. Randomly choose a minimal subset (a *sample*) of data points necessary to fit the model
2. Fit the model to this subset of data
3. Test all the other data points to determine if they are consistent with the fitted model (i.e. if they lie within a distance t of the model's prediction).
4. Count the number of inliers (the *consensus* set). Size of consensus set is model's *support*
5. Repeat from step 1 for N trials

RANSAC: simple correspondence example



1. Randomly choose a minimal subset (a *sample*) of data points necessary to fit the model
2. Fit the model to this subset of data
3. Test all the other data points to determine if they are consistent with the fitted model (i.e. if they lie within a distance t of the model's prediction).
4. Count the number of inliers (the *consensus* set). Size of consensus set is model's *support*
5. Repeat from step 1 for N trials

RANSAC: simple correspondence example



After N trials select the model parameters with the highest support and re-estimate the model using all the points in this subset.

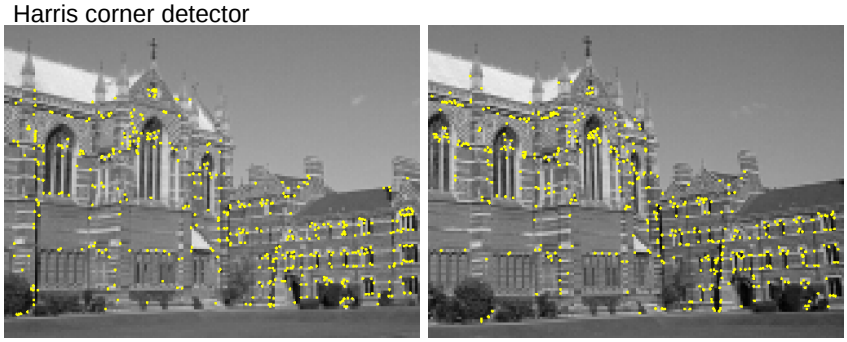
RANSAC: real correspondence example

Generally, the correspondence between views will be more complex than a pure translation.

Translation and rotation of the camera results in more complex transformations between images.

We can still estimate the parameters of this transformation by sampling more pairs of points (e.g. 4 pairs of putative matches)

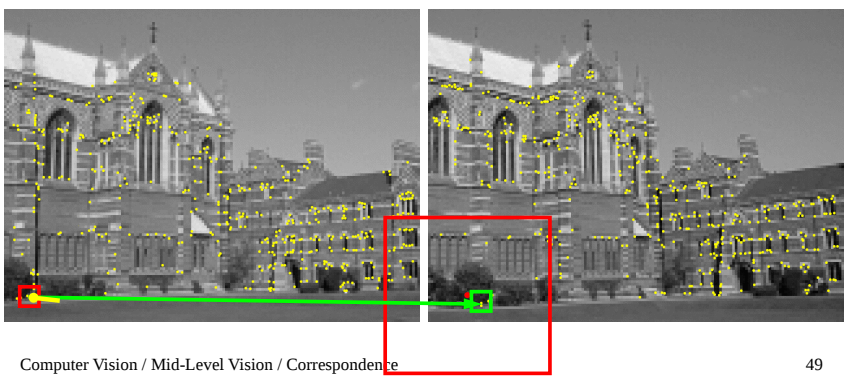
In this example approx 500 interest points have been extracted with the Harris corner detector



RANSAC: real correspondence example

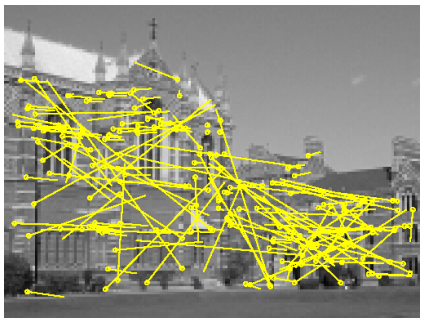
For each interest point the best match has been found within a square search window (here 300 pixels) using SSD

These putative matches are shown using a line pointing from the interest point in the left image to the pixel location of the corresponding point in the right image



RANSAC: real correspondence example

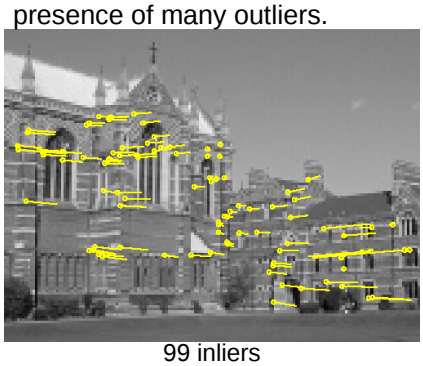
This results in 188 initial matches (which exceed some similarity threshold)



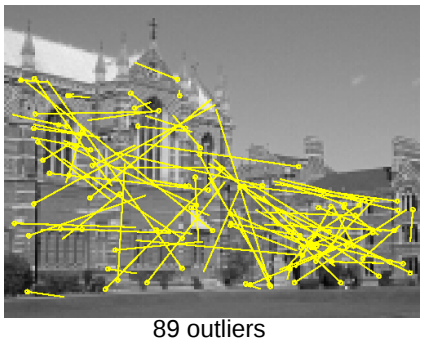
RANSAC: real correspondence example

Applying RANSAC to determine the transformation between the camera locations, results in a model that is consistent with 99 matches and inconsistent with 89 matches.

Note, RANSAC allows correspondence to be found even in the presence of many outliers.



99 inliers



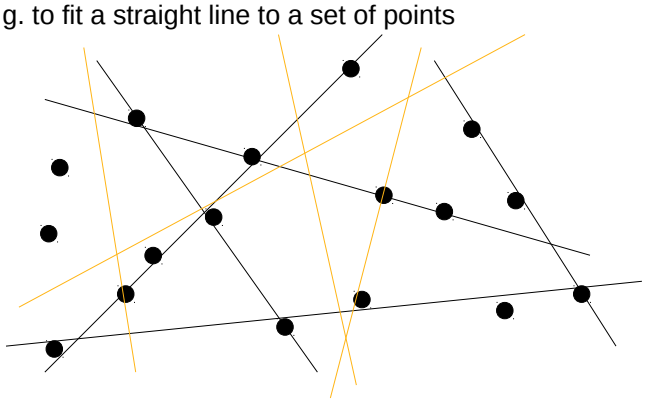
89 outliers

RANSAC for fitting

Recall, fitting algorithms (used for segmentation):

A class of methods that try to use a mathematical model to represent a set of tokens.

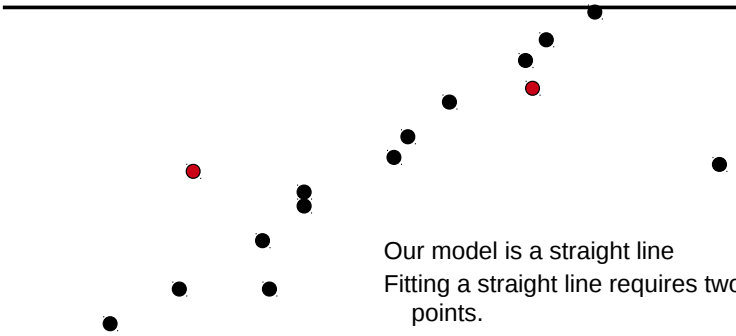
e.g. to fit a straight line to a set of points



One algorithm for fitting a model to data is the Hough Transform

RANSAC can also be used

RANSAC: line fitting example



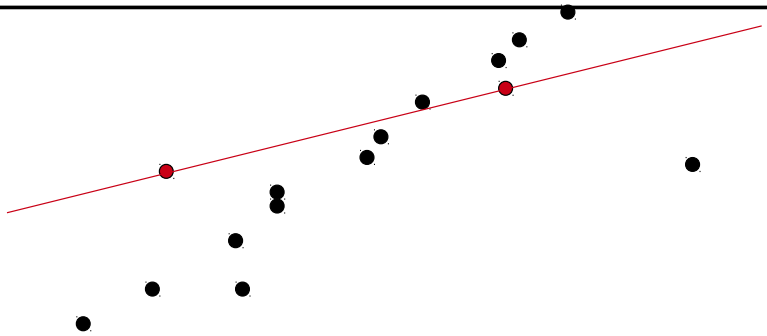
Our model is a straight line

Fitting a straight line requires two points.

Hence, our sample size is two.

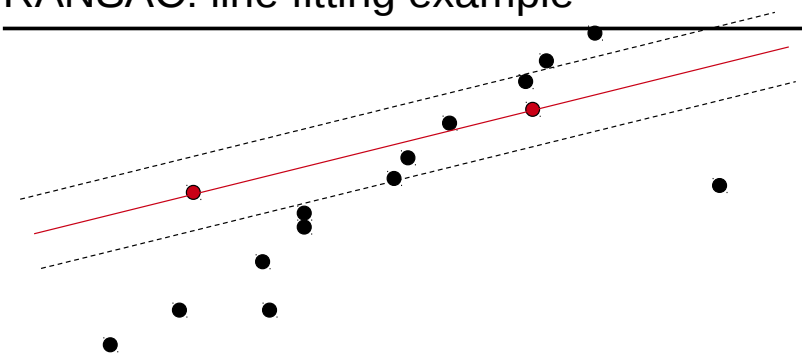
1. Randomly choose a minimal subset (a *sample*) of data points necessary to fit the model
2. Fit the model to this subset of data
3. Test all the other data points to determine if they are consistent with the fitted model (i.e. if they lie within a distance t of the model's prediction).
4. Count the number of inliers (the *consensus* set). Size of consensus set is model's *support*
5. Repeat from step 1 for N trials

RANSAC: line fitting example



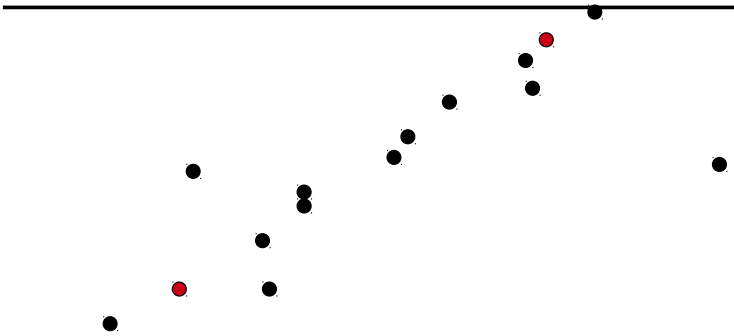
1. Randomly choose a minimal subset (a *sample*) of data points necessary to fit the model
2. Fit the model to this subset of data
3. Test all the other data points to determine if they are consistent with the fitted model (i.e. if they lie within a distance t of the model's prediction).
4. Count the number of inliers (the *consensus* set). Size of consensus set is model's *support*
5. Repeat from step 1 for N trials

RANSAC: line fitting example



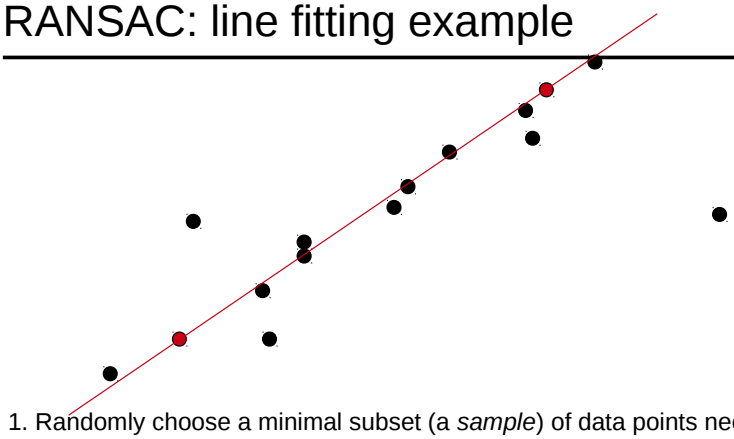
1. Randomly choose a minimal subset (a *sample*) of data points necessary to fit the model
2. Fit the model to this subset of data
3. Test all the other data points to determine if they are consistent with the fitted model (i.e. if they lie within a distance t of the model's prediction).
4. Count the number of inliers (the *consensus* set). Size of consensus set is model's *support*
5. Repeat from step 1 for N trials

RANSAC: line fitting example



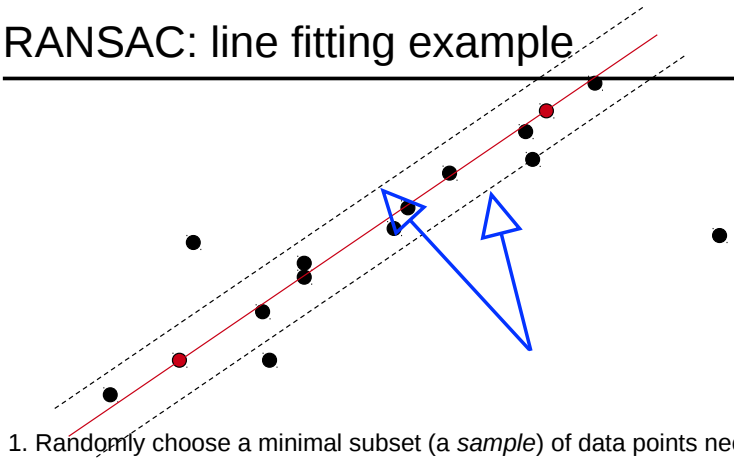
1. Randomly choose a minimal subset (a *sample*) of data points necessary to fit the model
2. Fit the model to this subset of data
3. Test all the other data points to determine if they are consistent with the fitted model (i.e. if they lie within a distance t of the model's prediction).
4. Count the number of inliers (the *consensus* set). Size of consensus set is model's *support*
5. Repeat from step 1 for N trials

RANSAC: line fitting example



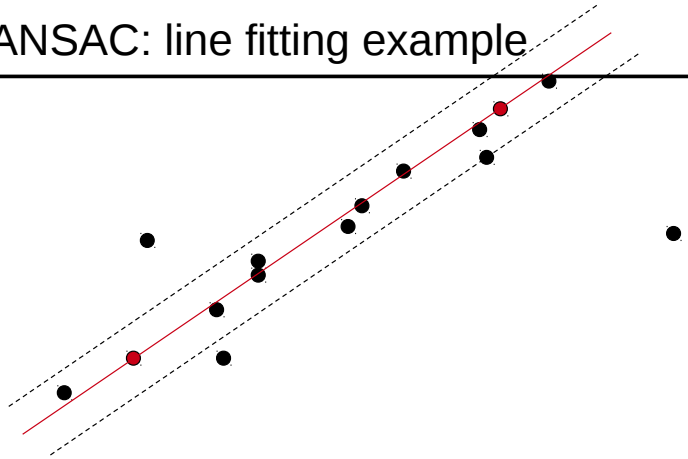
1. Randomly choose a minimal subset (a *sample*) of data points necessary to fit the model
2. Fit the model to this subset of data
3. Test all the other data points to determine if they are consistent with the fitted model (i.e. if they lie within a distance t of the model's prediction).
4. Count the number of inliers (the *consensus* set). Size of consensus set is model's *support*
5. Repeat from step 1 for N trials

RANSAC: line fitting example



1. Randomly choose a minimal subset (a *sample*) of data points necessary to fit the model
2. Fit the model to this subset of data
3. Test all the other data points to determine if they are consistent with the fitted model (i.e. if they lie within a distance t of the model's prediction).
4. Count the number of inliers (the *consensus* set). Size of consensus set is model's *support*
5. Repeat from step 1 for N trials

RANSAC: line fitting example



After N trials select the model parameters with the highest support and re-estimate the model using all the points in this subset.

RANSAC: pros and cons

Advantages:

- Simple and effective
- General method suited for a wide range of model fitting problems
  - » e.g. for segmentation by model fitting
  - » e.g. for finding camera transformation given stereo views
  - » e.g. for finding object trajectory given video

Disadvantages:

- Sometimes very many iterations are required if percentage of outliers is high.
- Lots of parameters to tune

Summary

Correspondence Problem

Finding matching image elements across images

- general problem arising in:
  - stereo (multiple cameras)
  - video (multiple times)
  - object recognition (comparing images)
- similar to grouping:
  - grouping is looking for similar elements in a single image
  - correspondence is looking for the same elements in multiple images

Summary

Solving the Correspondence Problem

1. Which image locations to match
   a. all locations (correlation-based method)
   b. selected interest points (feature-based methods: Harris, SIFT)

2. What properties to match
   a. image intensities (correlation-based method, Harris)
   b. a descriptor of image properties (SIFT)

3. Where to look for matches
   a. exhaustive search across entire image
   b. restricted search (constrained by task knowledge)

4. How to evaluate matches
   a. similarity (correlation, normalised correlation, correlation coefficient)
   b. differences (SSD, Euclidean distance, SAD)

5. How to find true correspondence (eliminate false matches)
   RANSAC