INF 5300 Repetition

28.05.14

Anne Solberg

Snakes

The energy function

$$E_{snake} = \int_{s=0}^{1} E_{\text{int}}(v(s)) + E_{image}(v(s)) + E_{con}(v(s))ds$$

Internal deformation energy of the snake itself. How it can bend and stretch.

A term that relates to gray levels in the image, e.g. attracts the snake to points with high gradient magnitude.

Constraints on the shape of the snake. Enchourages the contour to be smooth. (Often omitted)

The minimum values is found by derivation:

$$\frac{dE_{snake}}{dv} = 0$$

The internal deformation term

$$E_{\text{int}} = \alpha(s)\left|\frac{dv(s)}{ds}\right|^2 + \beta(s)\left|\frac{d^2v(s)}{ds^2}\right|^2$$

First derivative
Measures how stretched the contour is.
Keyword: point spacing.
Imposes tension.
The curve should be short if possible.
Physical analogy: v acts like a membrane.

Second derivative
Measures the curvature or bending energy.
Keyword: point variation.
Imposes rigidity.
Changes in direction should be smooth.
Physical analogy: v acts like a thin plate.

α and β are penalty parameters that control the weight of the two terms.
Low α values: the snake can stretch much.
Low β values: the snake can have high curvature.

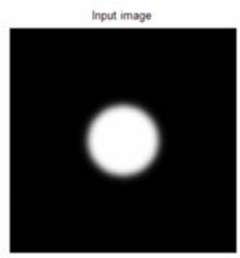
A simple image term

$$E_{image} = \int_0^1 P(v(s))ds$$

- A common way of defining P(x,y) is:

$$P(x,y) = -c\left|\nabla(G_\sigma * I(x,y))\right|$$

- c is a constant, ∇ is a gradient operator, G_σ is a Gaussian filter, and I(x,y) the input image. Note the minus sign as the gradient is high for edges.



The energy function

- Simple snake with only two terms (no termination energy):

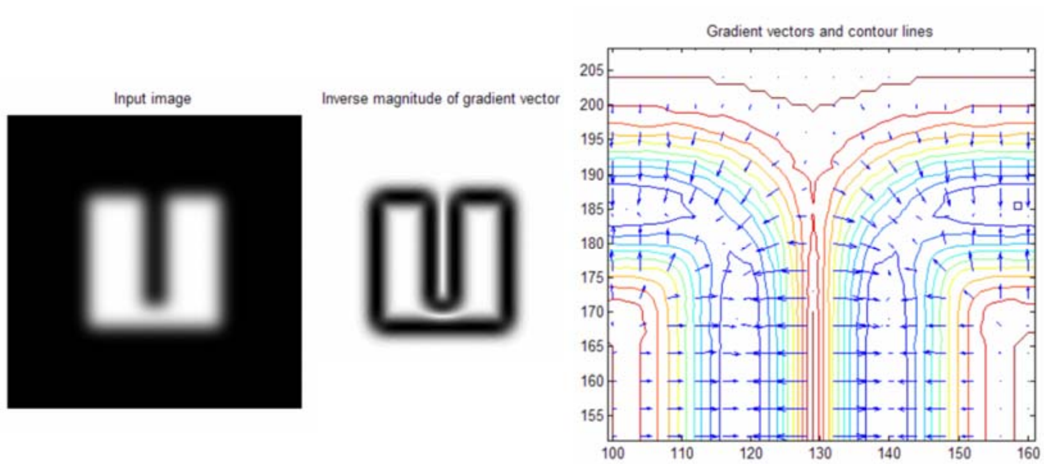
$$E_{snake}(s) = E_{\text{int}}(v_s) + E_{image}(v_s)$$
$$= \alpha\left|\frac{dv_s}{ds}\right|^2 + \beta\left|\frac{d^2v_s}{ds^2}\right|^2 + \gamma E_{edge}$$

- We need to approximate both the first derivative and the second derivative of v_s, and specify how E_{edge} will be computed.
- How should the snake iterate from its initial position?

How do we implement this?

- The energy function involves finding the new location of S new coordinates (x_s,y_s), $0 \leq s \leq 1$ for one iteration.
- Which algorithm can we use to find the new coordinate locations?
  1. Greedy algorithm
     - Simple, suboptimal, easier to understand
  2. Complete Kass algorithm
     - Optimizes all points on the countour simultaneously by solving a set of differential equations.
- These two algorithms will now be presented.

Capture range problems



Capture range problems

$$\mu\nabla^2 u = 0$$
$$\mu\nabla^2 v = 0$$

- The first term is Lagrange's equation which appear in often in physics, e.g. in heat flow or fluid flow.
- Imaging the a set of heaters is initialized at certain boundary conditions. As time evolves, the heat will redistribute/diffuse until we reach an equilibrium.
- In our setting, the gradient term act as the starting conditions.
- As the differential equation iterate, the gradient will diffuse gradually to other parts of the image in a smooth manner.

Capture range problems

- This equation has a similar solution to the original differential equation.
- We treat u and v as functions of time and solve the equations iteratively.
  - Comparable to how we iteratively computed $x^{<i+1>}$,$y^{<i+1>}$ from $x^{<i>}$,$y^{<i>}$
- The solution is obviously a numerical one, we use two sets of iterations, one for u and one for v.
- After we have computed v(x,y), we replace E_{ext} (the edge magnitude term) by v(x,y)
- So an interative algorithm is first used to compute v(x,y)

Computing **v**(x,y) continued..

- Select a time step Δt and a pixel spacing Δx and Δy for the iterations.
- Approximate the partial derivatives as

$$u_t = \frac{1}{\Delta t}\left(u_{i,j}^{n+1} - u_{i,j}^{n}\right)$$

$$v_t = \frac{1}{\Delta t}\left(v_{i,j}^{n+1} - v_{i,j}^{n}\right)$$

$$\nabla^2 u = \frac{1}{\Delta x \Delta y}\left(u_{i+1,j} + u_{i,j+1} + u_{i-1,j} + u_{i,j-1} - 4u_{i,j}\right) \text{A Laplacian approximation}$$

$$\nabla^2 v = \frac{1}{\Delta x \Delta y}\left(v_{i+1,j} + v_{i,j+1} + v_{i-1,j} + v_{i,j-1} - 4v_{i,j}\right)$$

- Then the iterative equations are:

$$u_{i,j}^{n+1} = \left(1 - b_{i,j}\Delta t\right)u_{i,j}^{n} + r\left(u_{i+1,j} + u_{i,j+1} + u_{i-1,j} + u_{i,j-1} - 4u_{i,j}\right) + c_{i,j}^{1}\Delta t$$

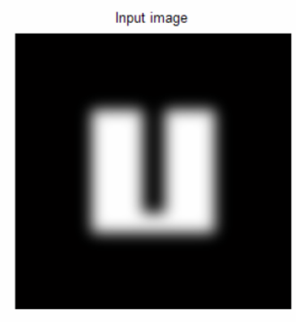
$$v_{i,j}^{n+1} = \left(1 - b_{i,j}\Delta t\right)v_{i,j}^{n} + r\left(v_{i+1,j} + v_{i,j+1} + v_{i-1,j} + v_{i,j-1} - 4v_{i,j}\right) + c_{i,j}^{2}\Delta t$$

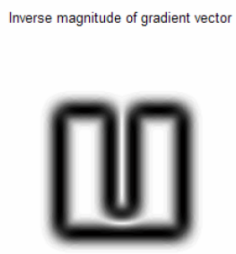
$$r = \frac{\mu \Delta t}{\Delta x \Delta y}$$

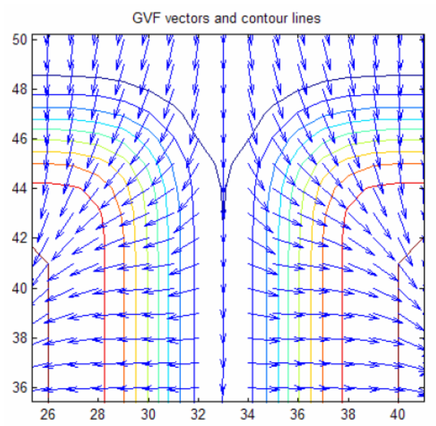
To get convergence we must have

$$\Delta t \le \frac{\Delta x \Delta y}{4\mu}$$

Capture range problems







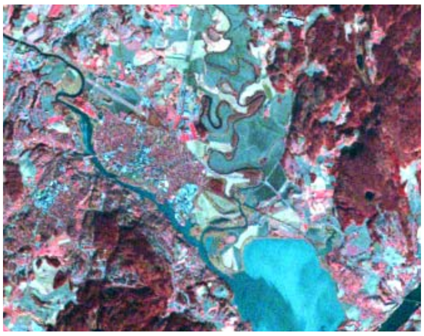
INF 5300 - 5.2.2014
Energy functions for segmentation/classification

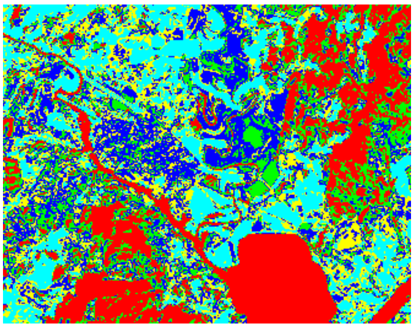
Anne Schistad Solberg

- Bayesian spatial models for classification
- Markov random field models for spatial context
- Other segmentation techniques:
  - EM-clustering
  - Mean shift segmentation
  - Graph-based segmentation (briefly)

Background – contextual classification

- An image normally contains areas of similar class
  - neighboring pixels tend to be similar.
- Classified images based on a non-contextual model often contain isolated misclassified pixels (or small regions).
- How can we get rid of this?
  - Majority filtering in a local neighborhood
  - Remove small regions by region area
  - Bayesian models for the joint distribution of pixel labels in a neighborhood.
- How do we know if the small regions are correct or not?
  - Look at the data, integrate spatial models in the classifier.





A Bayesian model for ALL pixels in the image

$Y = \{y_1,...,y_N\}$ Image of feature vectors to classify
$X = \{x_1,...x_N\}$ Class labels of pixels

- Classification consists choosing the class that maximizes the posterior probabilities for **ALL** pixels in the image

$$P(X \mid Y) = \frac{P(Y \mid X)P(X)}{\sum_{\text{all classes}} \mathrm{P(Y \mid X)P(X)}}$$

- Maximizing P(X|Y) with respect to $x_1,.....x_N$ is equivalent to maximizing P(Y|X)P(X) since the denominator does not depend on the classes $x_1,.....x_N$.
- Note: we are now maximizing the class labels of ALL the pixels in the image simultaneously.
- This is a problem involving finding N class labels simuntaneously.
- P(X) is the prior model for the scene. It can be simple prior probabilities, or a model for the spatial relation between class labels in the scene.

Back to the initial model...

$Y = \{y_1,...,y_N\}$ Image of feature vectors to classify
$X = \{x_1,...x_N\}$ Class labels of pixels
Task: find the optimal estimate **x**' of the true labels **x*** for all pixels in the image

- Classification consists choosing the class labels **x**' that maximizes the posterior probabilities

$$P(\mathbf{X} = \mathbf{x} \mid \mathbf{Y} = \mathbf{y}) = \frac{P(\mathbf{Y} = \mathbf{y} \mid \mathbf{X} = \mathbf{x})P(\mathbf{X} = \mathbf{x})}{\sum_{\text{all classes}} \mathrm{P}(\mathbf{Y} = \mathbf{y} \mid \mathbf{X} = \mathbf{x})\mathrm{P}(\mathbf{X} = \mathbf{x})}$$

- We assume that the observed random variables are conditionally independent:

$$P(\mathbf{Y}=\mathbf{y} \mid \mathbf{X}=\mathbf{x}) = \prod_{i=1}^{M} P(Y_i = y_i \mid X_i = x_i)$$

- We use a Markov field to model the spatial interaction between the classes (the term P(**X**=**x**)).

$$P(\mathbf{X}=\mathbf{x}) = \mathrm{e}^{-\mathrm{U}(\mathbf{x})/\mathrm{Z}}$$

$$U(\mathbf{x}) = \sum_{c \in Q} V_c(\mathbf{x})$$

$$V_c(\mathbf{x}) = \beta I(x_i, x_k)$$

- Rewrite $P(Y_i=y_i|X_i=x_i)$ as

$$P(\mathbf{Y}=\mathbf{y} \mid \mathbf{X}=\mathbf{x}) = \frac{1}{Z_1} e^{-Udata(Y|X)}$$

$$Udata(Y \mid X) = \sum_{i=1}^{M} -\log P(Y_i = y_i \mid X_i = x_i)$$

- Then, $P(\mathbf{X}=\mathbf{x} \mid \mathbf{Y}=\mathbf{y}) = \frac{1}{Z_2} e^{-Udata(Y|X)} e^{-U(X)}$

- Maximizing this is equivalent to minimizing

$$U_{data}(Y \mid X) + U(X)$$

Udata(X|C)

- Any kind of probability-based classifier can be used, for example a Gaussian classifier with a k classes, d-dimensional feature vector, mean μ_k and covariance matrix Σ_k:

$$Udata(x_i \mid c_i) = -\frac{d}{2}\log(2\pi) - \frac{1}{2}\log\left(\left|\Sigma_k\right|\right) - \frac{1}{2}x_i^T\Sigma_k^{-1}x_i + \mu_k^T\Sigma_k^{-1}x_i - \frac{1}{2}\mu_k^T\Sigma_k^{-1}\mu_k$$

$$\propto -\frac{1}{2}x_i^T\Sigma_k^{-1}x_i + \mu_k^T\Sigma_k^{-1}x_i - \frac{1}{2}\mu_k^T\Sigma_k^{-1}\mu_k - \frac{1}{2}\log\left(\left|\Sigma_k\right|\right)$$

Finding the labels of ALL pixels in the image

- We still have to find an algorithm to find an estimate **x’** for all pixels.
- Alternative optimization algorithms are:
  - Simulated annealing (SA)
    - Can find a global optimum
    - Is very computationally heavy
  - Iterated Conditional Modes (ICM)
    - A computationally attractive alternative
    - Is only an approximation to the MAP estimate
  - Maximizing the Posterior Marginals (MPM)
- We will only study the ICM algorithm, which converges only to a local minima and is theoretically suboptimal, but computationally feasible.

ICM in detail

Initilalize x_t, t=1,...N as the non-contextual classification by finding the class which maximize $P(Y_t=y_t|X_t=x_t)$, assign it to classified_image(i,j)

```
For iteration k=1:maxit do
    For i=i:N,j=1:N (all pixels) do
        minimum_energy=High_number;
        For class s=1:S do
            Udata = -log (P(Y_t=y_t|X_t=x_t))
            Ucontxt=0;
            nof_similar_neighbors=0;
            for neighb=1:nof_neighbors
                if (classified_image(neighb)=s) //neighbor and s of same class
                    ++nof_similar_neighbors;
            Ucontxt = -beta*nof_similar_neighbors;
            energy = Udata + Ucontxt;
            if (energy < minimum_energy)
                minimum_energy = energy;
                bestclass = s;
        new_classified_image(i,j) = bestclass;
        if (new_classified_image(i,j)!=classified_image(i,j))
            ++nof_pixels_changed;
    if nof_pixels_changed<min-limit
        break;
```


Segmentation methods covered

- Watershed segmentation (INF 4300)
- Split-and-merge/region growing (INF 4300)
- K-means clustering (INF 4300)
  - We extend this to mixtures of Gaussian now
- Mean shift segmentation
- Graph-cut algorithms

Segmentation methods covered

- Watershed segmentation (INF 4300)
- Split-and-merge/region growing (INF 4300)
- K-means clustering (INF 4300)
  - We extend this to mixtures of Gaussian now
- Mean shift segmentation
- Graph-cut algorithms

Clustering by mixtures of Gaussians

- Euclidean distance can be replaced by Mahalanobis distance from point x_i to cluster center k:

$$d(x_i, \mu_k, \Sigma_k) = (x_i - \mu_k)^T \Sigma_k^{-1} (x_i - \mu_k)$$

- We could just modify the K-means algorithm to use this measure after the first iteration.
- Mixtures of Gaussian considers that samples can be softly assigned to several nearby cluster centers:

$$p(x \mid \pi_k, \mu_k, \Sigma_k) = \sum_k \pi_k \frac{1}{|\Sigma_k|} e^{-d(x, \mu_k, \Sigma_k)}$$

- π_k is the mixing coefficient for cluster with mean μ_k and covariance Σ_k.

The EM-algoritm for clustering

- The EM-algoritm iteratively estimate the mixture parameters:

1. Expectation step (E-step): compute

$$z_{ik} = \frac{1}{Z_i} \pi_k \frac{1}{|\Sigma_k|} e^{-d(x,\mu_k,\Sigma_k)} \text{ with } \sum_k z_{ik} = 1$$

An estimate of the probability that xi belongs to the kth Gaussian

2. Maximation stage (M-step): update

$$\mu_k = \frac{1}{N_k} \sum_i z_{ik} x_i$$

$$\Sigma_k = \frac{1}{N_k} \sum_i z_{ik} (x_i - \mu_k)(x_i - \mu_k)^T$$

$$\pi_k = \frac{N_k}{N}$$

Mean shift clustering/segmentation algorithm

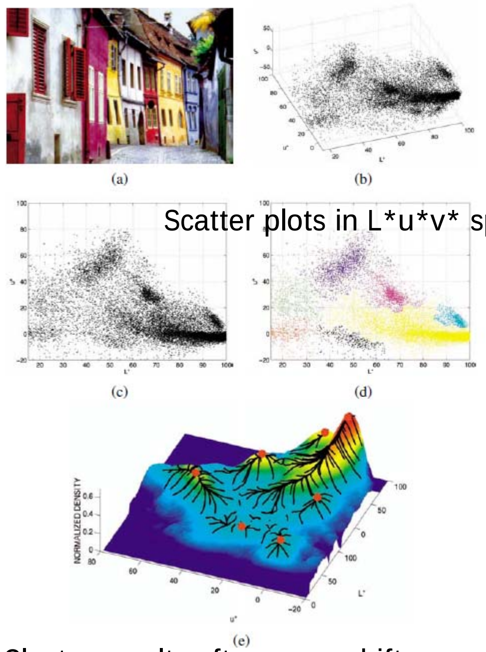
- K-means and mixtures of Gaussian are based on a parametric probability function.
- An alternative is to use a non-parametric smooth function that fits the data.
- The mean shift algoritms efficiently finds peaks in a distribution without estimating the entire distribution.
- It can be seen as the «inverse» of the watershed algorithm, which clims downhill.

The mean shift - background

- To estimate a density function for the scatter plots, we could use a Parzen window estimator, which smooths the data by convolving it with a kernel k() of width h:

$$f(x) = \sum_i K(x - x_i) = \sum_i k\left(\frac{\|x - x_i\|^2}{h^2}\right)$$

- When we have computed f(x), we could find peaks by gradient descent.
- Drawback: does not work well with sparse data points.
- Solution: finding the peaks WITHOUT estimating the entire distribution.



Scatter plots in L*u*v* space

Cluster results after mean shift clustering, peaks marked in red

Mean shift segmentation

- Multiple restart gradient descent algorithm: start at many points y_k and take a step up-hill from these point.
- The gradient of f is (g(r)=-k'(r)):

$$\nabla f(x) = \sum_i (x_i - x) G(x - x_i) = \sum_i (x_i - x) g\left(\frac{\|x - x_i\|^2}{h^2}\right)$$

- This can be written as

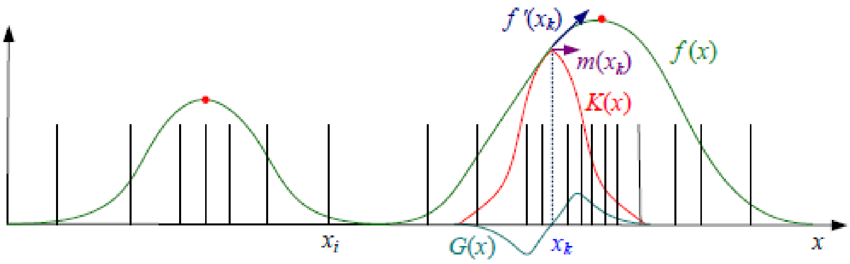
$$\nabla f(x) = \left[\sum_i G(x - x_i)\right] m(x)$$

$$m(x) = \frac{\sum_i x_i G(x - x_i)}{\sum_i G(x - xi)} - x$$

- The current estimate of y_k is replaced with its locally weighted mean:

$$y_{k+1} = y_k + m(y_k) = \frac{\sum_i x_i G(y_k - x_i)}{\sum_i G(y_k - x_i)}$$

Illustration of mean shift



- The kernel K is convolved with the image.
- The derivative of the kernel is computed by convolving the image with the derivative of the kernel
- The mean shift change m(x) is found from the derivative f'(x)

- Simple but slow algorithm: start a separate mean shift estimate y at every input point x, and iteration until only small changes.
- Faster: start at random points.
- Including location information:
  - Add the coordiates x_s= (x,y)in the kernel:

$$K(x_j) = k\left(\frac{\|x_r\|^2}{h_r^2}\right) k\left(\frac{\|x_s\|^2}{h_s^2}\right)$$

  - x_r is the spectral feature vector and h_r and h_s the bandwidth in the spectral and spatial domain.
  - The effect of this is that the algoritm step will take both spectral and spatial information and e.g. use larger steps in space between pixels with similar color.

INF 5300 - 26.2.2014
Detecting good features for tracking
Anne Schistad Solberg

- Finding the correspondence between two images
  - **What are good features to match?**
    - Points?
    - Edges?
    - Lines?

What type of features are good?

•Point-like features?

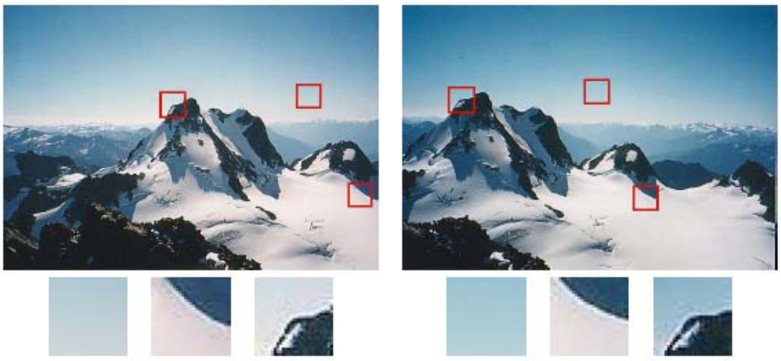
•Region-based features?

•Edge-based features?

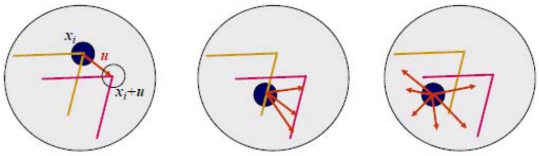
•Line-based features?

Feature detection

- Goal: search the image for locations that are likely to be easy to match in a different image.



- What characterizes the regions? How unique is a location?
  - Texture?
  - Homogeneity?
  - Contrast?
  - Variance?



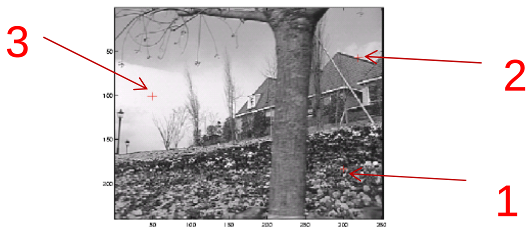
Feature detection

- A simple matching criterion: summed squared difference:

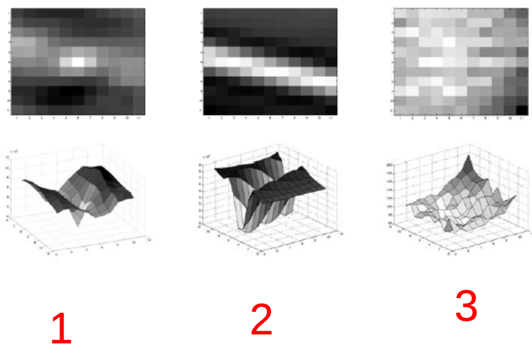
$$E_{WSSD}(u) = \sum_i w(x_i)[I_1(x_i + u) - I_0(x_i)]^2$$

- I_0 and I_1 are the two images, **u**=(u,v) the displacement vector, and w(x) a spatially varying weight function.
- Check how stable a given location is (with a position change Δu) in the first image by computing the auto-correlation function:

$$E_{AC}(\Delta u) = \sum_i w(x_i)[I_0(x_i + \Delta u) - I_0(x_i)]^2$$

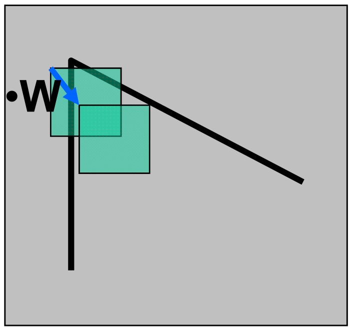


(a)



Feature detection: the math

- Consider shifting the window **W** by (u,v)
  - how do the pixels in **W** change?
  - Do a Taylor series expansion of the autocorrelation to allow fast computation:



$$E_{AC}(\Delta u) = \sum_i w(x_i)[I_0(x_i + \Delta u) - I_0(x_i)]^2$$
$$\approx \sum_i w(x_i)[I_0(x_i) + \nabla I_0(x_i)\Delta u - I_0(x_i)]^2$$
$$= \sum_i w(x_i)[\nabla I_0(x_i)\Delta u]^2$$
$$= \Delta u^T \mathbf{A} \Delta u,$$

where $\nabla I_0(x_i) = \left(\frac{\partial I_0}{\partial x}, \frac{\partial I_0}{\partial y}\right)(x_i)$ is the image gradient at x_i.

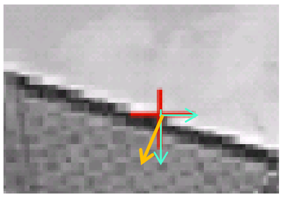
•Compute the gradients robustly using a Derivative of Gaussian filter

- The autocorrelation matrix A is:

$$A = w * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Feature detection: the math

- The matrix A carries information about the uncertainty of the location of a patch.
- A is called a tensor matrix and is formed by outer products of the gradients, convolved with a weighting function w to get a pixel-based uncertainty estimate.
- Eigenvector decomposition of A gives two eigenvalues, λ_0 and λ_1.
- The smallest eigenvalue carries information about the uncertainty.



•High gradient in the direction of maximal change

• If there is one dominant direction, we are quite certain about the direction estimate, and λ_{min} will be much smaller than λ_{max}.

•A high value of λ_{min} means that the gradient changes much in both directions, so this can be a good keypoint.

Feature detection: Harris corner detector

- Harris and Stephens (1988) proposed an alternative criterion computed from A (α=0.06 is often used):

$$\det(A) - \alpha \operatorname{trace}(A)^2 = \lambda_{\max}\lambda_{\min} - \alpha(\lambda_{\max} + \lambda_{\min})^2$$

- Other alternatives are e.g. the harmonic mean:

$$\frac{\det A}{\operatorname{trace}(A)} = \frac{\lambda_{\max}\lambda_{\min}}{\lambda_{\max} + \lambda_{\min}}$$

- The difference between these criteria is how the eigenvalues are blended together.

Feature detection algorithm

1. Compute the gradients I_x and I_y , and I_{xy} using a robust Derivative-of-Gaussian kernel (hint: convolve a Sobel x and y with a Gaussian).
2. Convolve these gradient images with a larger Gaussian to further robustify.
3. Create the matrix A from the robustified gradients from 2.
4. Compute either the smallest eigenvalue or the Harris corner detector measure from A.
5. Find local maxima above a certain threshold and report them as detected feature point locations.
6. Adaptive non-maximal suppression (ANMS) is often used to improve the distribution of feature points across the image.

How do we get rotation invariance?

- Option 1: use rotation-invariant feature descriptors.
- Option 2: estimate the locally dominant orientation and create a rotated patch to compute features from.

How do we get scale invariance?

- These operators look at a fine scale, but we might need to match features at a broader scale.
- Solution 1:
  – Create a image pyramid and compute features at each level in the pyramid.
    - At which level in the pyramid should we do the matching on? Different scales might have different characteristic features.
- Solution 2:
  - Extract features that are stable both in location AND scale.
  - SIFT features (Lowe 2004) is the most popular approach of such features.

Scale-invariant features (SIFT)

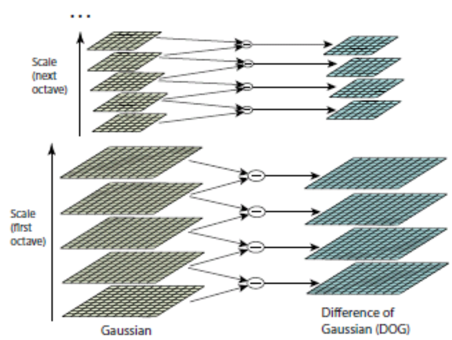
- See Distinctive Image Features from Scale-Invariant Keypoints by D. Lowe, International Journal of Computer Vision, 20,2,pp.91-110, 2004.
- Invariant to scale and rotation, and robust to many affine transforms.
- Main components:
  1. Scale-space extrema detection – search over all scales and locations.
  2. Keypoint localization – including determining the best scale.
  3. Orientation assignment – find dominant directions.
  4. Keypoint descriptor - local image gradients at the selected scale, transformed relative to local orientation.

SIFT: 1. Scale-space extrema

- A Gaussian filter is applied at different scales $L(x,y,\sigma) = G(x,y,\sigma) * I(x,y,\sigma)$.
- Compute keypoints in scale space by difference-of-Gaussian, where the difference is between two nearby scales separated by a constant k:

$$D(x,y,\sigma) = (G(x,y,k\sigma) - G(x,y,\sigma)) * I(x,y)$$

- This is an efficient approximation of a Laplacian of Gaussian, normalized to scale σ (see Lowe 2004).
- Detecting extrama in scale is based on sampling different scales.
- Extrema in space are also detected.



SIFT 1: extrema detection

- Consider a Taylor series expansion of the scale-space function D(x,y,σ) around sample point x

$$D(x) = D + \frac{\partial D^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x$$

- The location of the extreme is found by take the derivative of D(x) and setting it to zero:

$$\hat{x} = -\frac{\partial^2 D^{-1}}{\partial x^2} \frac{\partial D}{\partial x}$$

- It is computed by differences of neighboring sample points, yielding a 3x3 linear system.
- The value of D at the extreme point is useful for suppressing extrema with low contrast, |D|<0.03 are suppressed.

$$D(\hat{x}) = D + \frac{1}{2} \frac{\partial D^T}{\partial x} \hat{x}$$

SIFT 1: eliminating edge response

- Since points on an edge are not very stable, so such points need to be eliminated.
- This is done using the curvature, computed from the Hessian matrix of D.

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

- The eigenvalues of H are proportation to principal curvatures of D. As with Harris, consider the ratio between the eigenvalues α and β. They found a good criteria to be to only keep the points where

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r}$$

- r=10 is often used.

SIFT 2: computing orientation

- To normalize for the orientation of the keypoints, we need to estimate the orientation.
- They used simple pixel differences to do this (L is a Gaussian smoothed image):

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$
$$\theta(x,y) = \tan^{-1}\big((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y))\big)$$

- Then, they computed orientation histograms with 36 bins covering the 360 degrees of possible orientations.
- In this histogram, the highest peak, and other peaks with hight 80% of max are found. If a localization has multiple peaks, it can have more than 1 orientation.

Feature descriptors

- Which features should we extract from the key points?
- These features will later be used for **matching** to establish the motion between two images.
- How is a good match computed (more in chapter 8)?
  - Sum of squared differences in a region?
  - Correlation?
- The local appearance of a feature will often change in orientation and scale (this should be utilized e.g. by extracting the local scale and orientation and then use this scale (or a coarser one) in the matching).

SIFT: feature extraction stage

- Select the level of the Gaussian pyramid where the keypoints were identified.
- Compute the gradient at each point in a 16x16 window around each keypoint. Weight the gradient values by a Gaussian function.
- Threshold gradient magnitude to throw out weak edges.
- Form a gradient orientation histogram for each 4x4 quadrant using 8 directional bins (using trilinear interpolation of the gradient magnitude to 2x2x2 bins).
- This results in 128 (16*8) non-negative values which are the raw SIFT-features.
- Further normalize the vector.

Feature matching

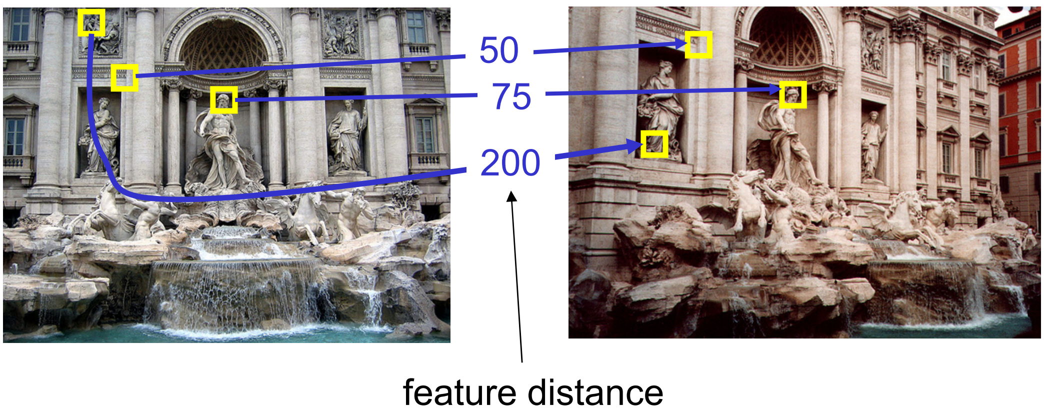
- Matching is divided into:
  - Define a matching strategy to compute the correspondence between two images.
  - Using efficient algorithms and data structures for fast matching (we will not go into details on this).
- Matching can be used in different settings:
  - Compute the correspondende between two partly overlapping images (= stitching).
    - Most key points are likely to find a match in the two images.
  - Match an object from a training data set with an unknown scene (e.g. for object detection).
    - Finding a match might be unlikely

Computing the match

- Assume that the features are normalize so we can measure distances using Euclidean distance.
- We have a list of keypoints features from the two images. Given a keypoint in image A, compute the similarity (=distance) between this point and all keypoints in image B.
- Set a threshold to the maximum allowed distance and compute matches according to this.
- Quantify the accuracy of matching in terms of:
  - TP: true positive: number of correct matches
  - FN: false negative: matches that were not correctly detected.
  - FP: false positive: proposed matches that are incorrect.
  - TN: true negative: non-matches that were correctly rejected.

Evaluating the results

How can we measure the performance of a feature matcher?



Performance ratios

- True positive rate (TPR)
  - TPR = TP/(TP+FN)
- False positive rate (FPR)
  - FPR = FP/(FP+TN)
- Positive predictive value (PPV)
  - PPV = TP/(TP+FP)
- Accuracy (ACC)
  - ACC = (TP+TN)/(TP+FN+FP+TN)

- Challenge: accuracy depends on the threshold for a correct match!

INF 5300 - 02.04.2014
Feature-based alignment

Anne Schistad Solberg

- Finding the alignment between features from different images
- Geometrical transforms – short repetition
- RANSAC algorithm for robust transform computation

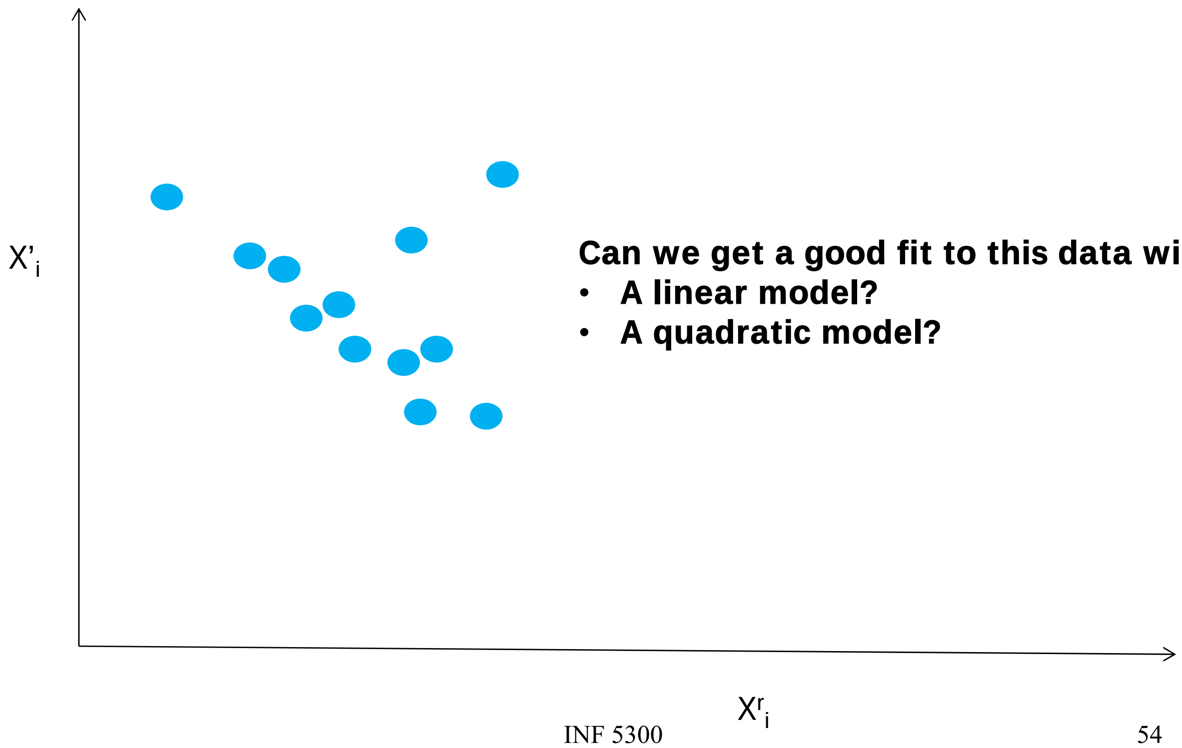
INF 2310 - coregistration III

- The root mean square error is used to evaluate how good a match is
- Given M point pairs $(x_i,y_i),(x_i^r,y_i^r)$ (r is the reference image)
- Assume that the transform gives estimated coordinates in the reference image as (x'_i,y'_i)
- $(x_i,y_i) \dashrightarrow (x'_i,y'_i)$
- The number of point pairs is M >>3 for affine transforms og M>>6 for quadratic
- The coefficients in the transform are computed as the values that minimize the square error between the true coordinates
- (x_i^r,y_i^r) and the transformed coordinates (x_i',y_i')

$$J = \sum_{i=1}^{M} (x_i' - x_i^r)^2 + (y_i' - y_i^r)^2$$

- Simple linear algebra is used to find the solution to this problem.

A data example
Estimated vs. true coordinates



Can we get a good fit to this data with
- **A linear model?**
- **A quadratic model?**

Robustness of matching

Introducing a robust matching algorithm

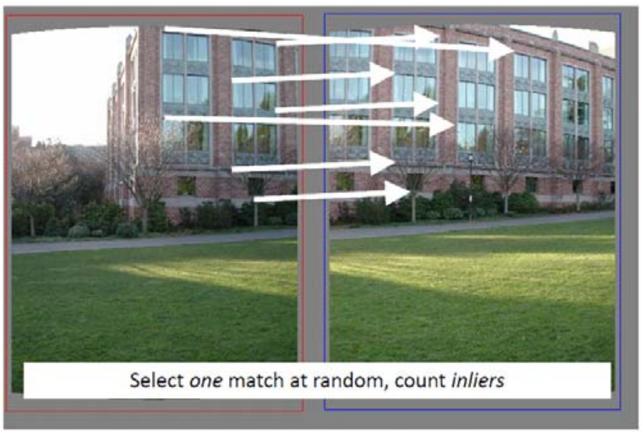
- The detected features are not perfect, there may be outliers where the match is NOT good.
- If we want to fit a line:
  - Count the number of points that agree with the line.
    - Agree means that the distance between the location of the estimated and the true coordinates is very small.
    - Points which fulfill this criterion are called inliers.
    - Other points are called outliers.
  - For all possible lines, select the one with the larges number of inliers.

How do we find the best line?

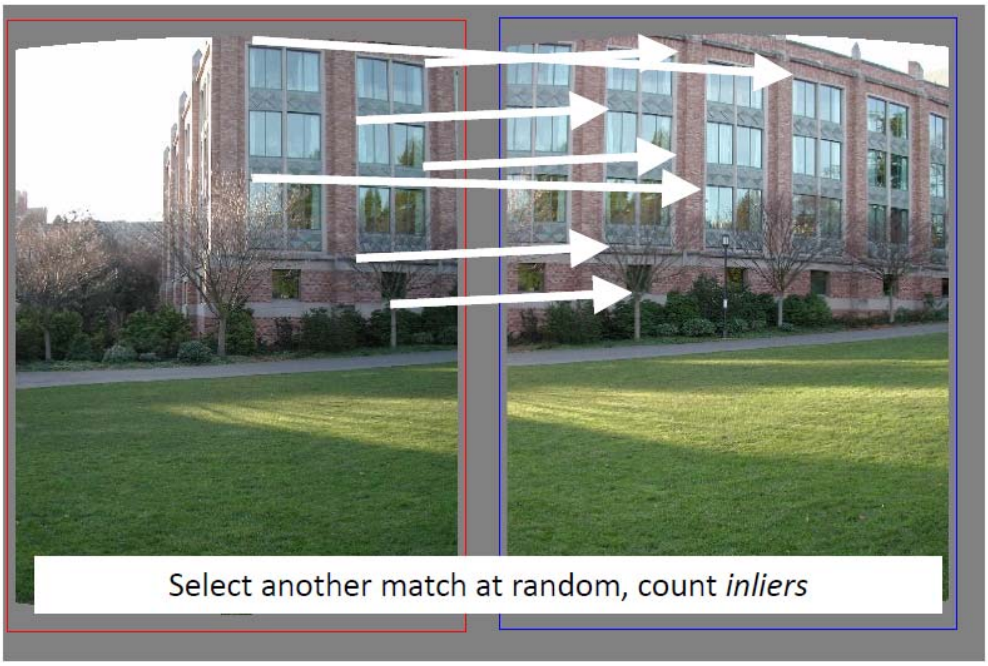
- Unlike least-squares, there is no simple closed-form solution.
- Trial-and-test:
  - Try out many lines, keep the best one

RANdom Sample Consensus

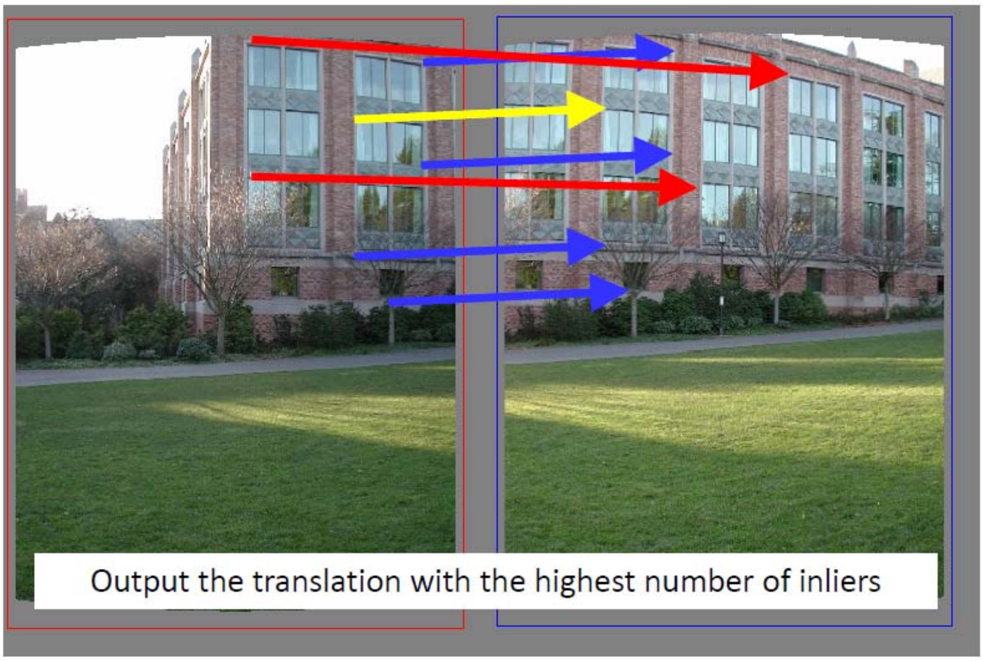
- In this example: Linear model, two points needed to get a fit.
- Select two points at random, compute the transform coefficients.
- Try this model for all other samples and count the number of inliers among the other samples.



<u>RAN</u>dom **<u>S</u>**ample **<u>C</u>**onsensus

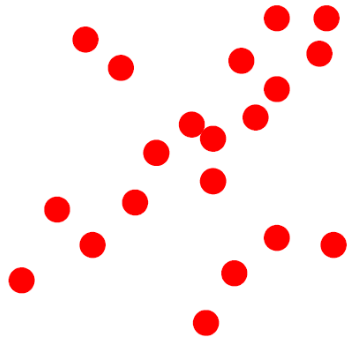


<u>RAN</u>dom **<u>S</u>**ample **<u>C</u>**onsensus



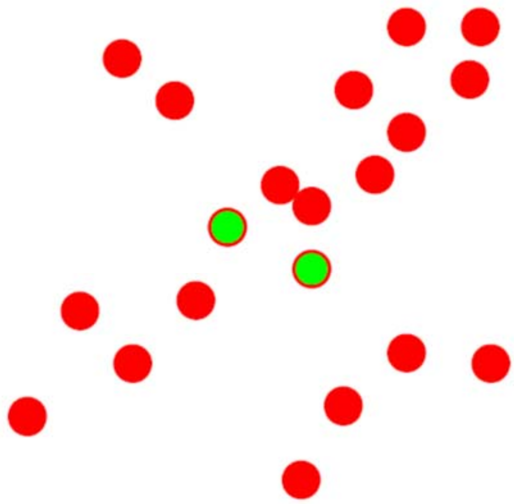
RANSAC

- RANdom Sample Consensus (Fischler and Bolles, 1981)
- Algorithm:
  1. Sample (randomly) exactly the number of points needed to fit the model.
  2. Solve for the model parameters based on the samples.
  3. Score by the fraction of inliers within a preset threshold.
- Repeat 1-3 until the best model is found with high confidence.



RANSAC

Line fitting example



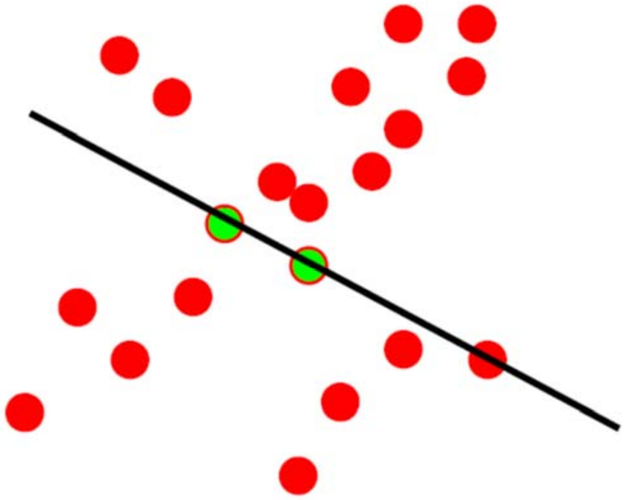
Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

RANSAC

Line fitting example



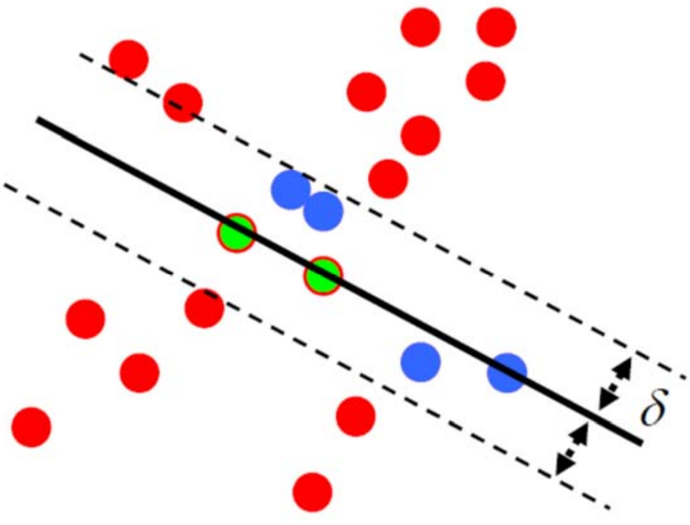
Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

RANSAC

Line fitting example



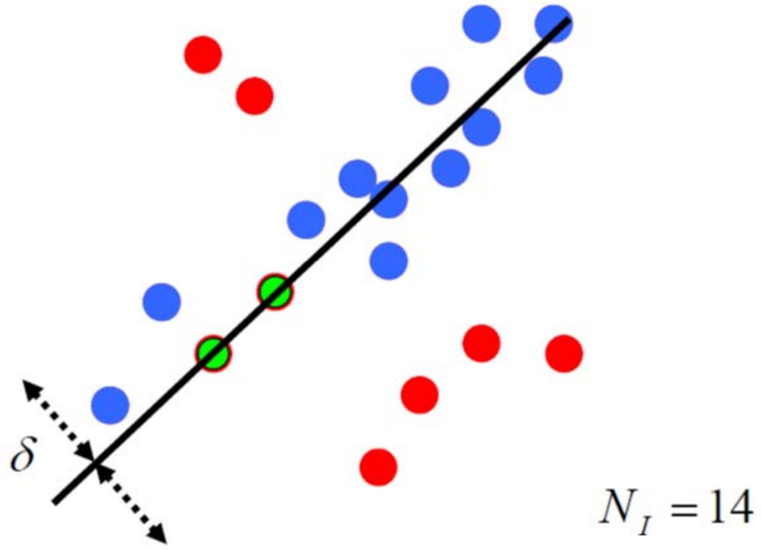
$N_I = 6$

Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

RANSAC



Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

RANSAC algorithm

General version:

1. Randomly choose s samples
   s=minimum sample size that let you fit a model
2. Fit a model (e.g. line) to those samples
3. Count the number of inliers that approximately fit the model.
4. Repeat N times
5. Choose the model that has the largest set of inliers, and fit this model to all inliers using e.g. least squares.
   – When we have the best set of points, refine the model using all inliers.

RANSAC conclusions

- Good:
  - Robust to outliers (can handle up to 50% outliers)
  - Applicapable to a larger number of parameters than Hough transform/parameters are easier to choose.
- Bad:
  - Computational time grows quickly with fraction of outliers and number of parameters.
  - Not good for getting multiple fits.
- Common applications:
  - Robust linear regression (and similar)
  - Computing the transform behind image stitching (called homography)
  - Image registration/Estimating the fundamental matrix relating two views.

INF 5300 - 09.04.2014
Dense motion and flow

Anne Schistad Solberg

- Motion perception
- Motion visualization
- Image similarity measures
- Motion estimation
- Optical flow algorithm

- Slide credits: Several slides adapted from R. Szeliski CSE 576.

This lecture: dense motion

- Motion vectors are now estimated from every point an a image sequence.
- Motion maps are created, and each pixel can have a different motion vector.
- Some regularization of the motion vectors is done to get smooth estimates.
  - No restriction that all pixels move in the same average direction.
- Video normally has high frame rate:
  - Small motion between one fram and the next frame

The Brightness Constraint

- Brightness Constancy Equation/Find similar patches in two images:

$$J(x,y) \approx \boxed{I(x+u(x,y), y+v(x,y))}$$

Or, equivalently, minimize :

$$E(u,v) = (J(x,y) - I(x+u, y+v))^2$$

Linearizing (assuming small *(u,v)*) using Taylor series expansion:

$$J(x,y) \approx \boxed{I(x,y) + I_x(x,y) \cdot u(x,y) + I_y(x,y) \cdot v(x,y)}$$

I_x and I_y are the horisontal and vertical image gradients

Patch Translation [Lucas-Kanade]

Assume a single velocity for all pixels within an image patch

$$E(u,v) = \sum_{x,y \in \Omega} \left(I_x(x,y)u + I_y(x,y)v + I_t \right)^2$$

Minimizing

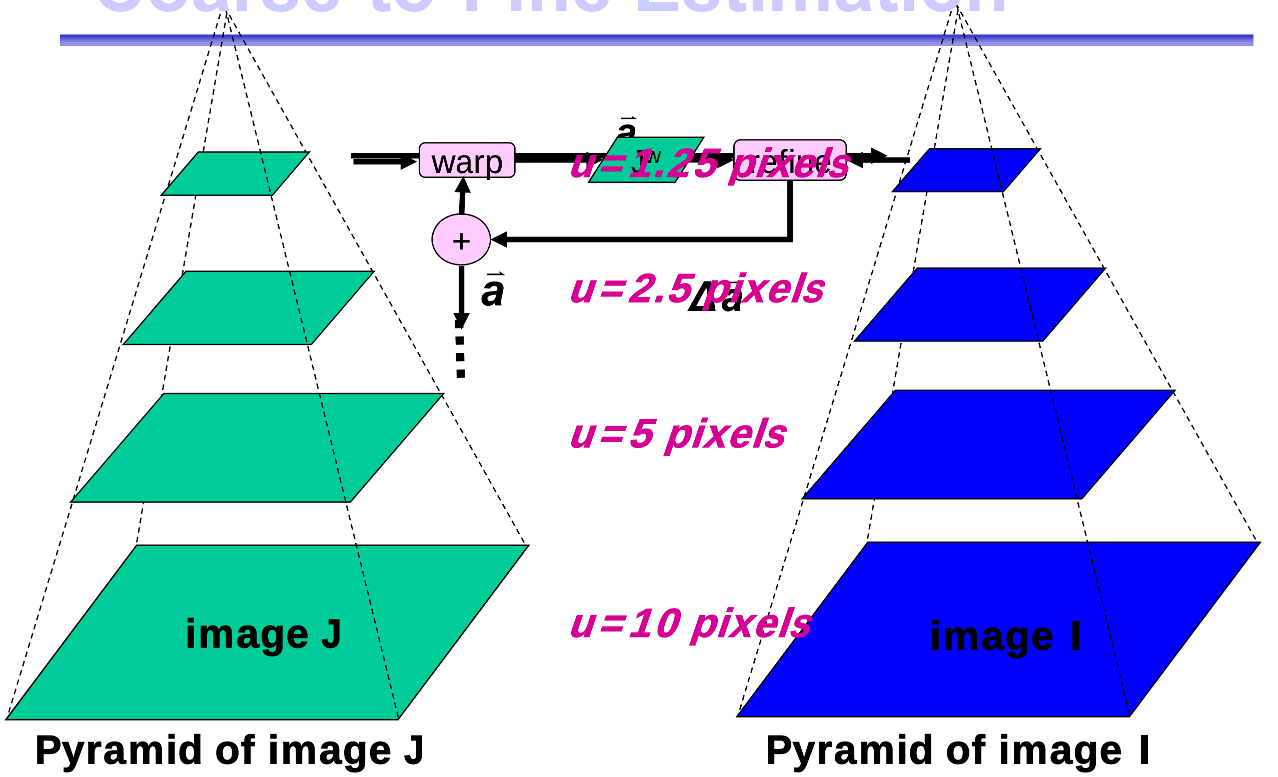
$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = -\begin{pmatrix} \sum I_x I_t \\ \sum I_y I_t \end{pmatrix}$$

Balance spatial gradients by temporal gradients and the shift in u

$$\left(\sum \nabla I \nabla I^T \right) \vec{U} = -\sum \nabla I I_t$$

LHS: sum of the 2x2 outer product of the gradient vector (gradient tensor)

Coarse-to-Fine Estimation



Szeliski

INF 5300 – Support Vector Machine Classifiers (SVM)

Anne Solberg (anne@ifi.uio.no)

Introduction:

Linear classifiers for two-class problems

The kernel trick – from linear to a high-dimensional generalization

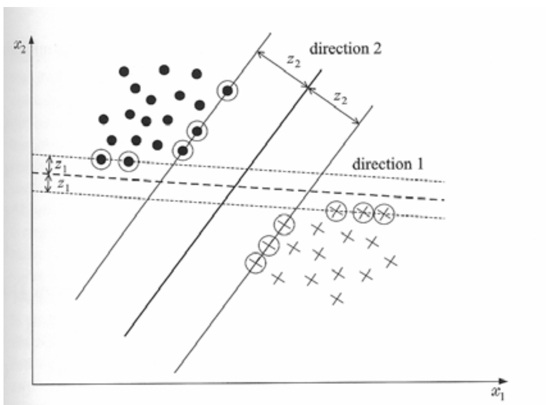
Generation from 2 to M classes

Practical issues

Hyperplanes and margins

- A hyperplane is defined by its direction (w) and exact position (w_0).
- Remember that w is orthogonal to the hyperplane
- If both classes are equally probable, the **distance from the hyperplane to the closest points** in both classes should be equal. This is called the margin.
- The margin for direction 1 is $2z_1$, and for direction 2 it is $2z_2$.
- The distance from a point to a hyperplane is

$$z = \frac{|g(x)|}{\|w\|}$$



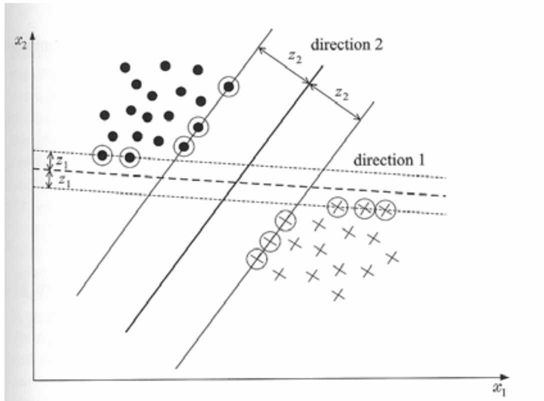
Hyperplanes and margins

- We can scale w and w_0 such that g(x) will be equal to 1 at the closest points in the two classes. This is equivalent to:

1. Have a margin of $\frac{1}{\|w\|}+\frac{1}{\|w\|}=\frac{2}{\|w\|}$
2. Require that

$$w^T x + w_0 \geq 1, \quad \forall x \in \omega_1$$
$$w^T x + w_0 \leq -1, \quad \forall x \in \omega_2$$

- **Goal: find w and w_0**



The optimization problem with margins

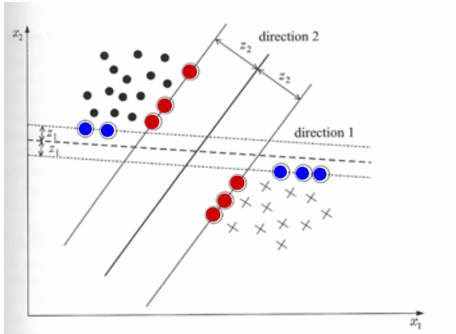
- The class indicator for pattern i, y_i, is defined as 1 if y_i belongs to class ω_1 and -1 if it belongs to ω_2.
- The best hyperplane with margin can be found by solving the optimization problem with respect to w and w_0 :

$$\text{minimize} \quad J(w)=\frac{1}{2}\|w\|^2$$
$$\text{subject to} \quad y_i(w^T x_i + w_0) \geq 1, \quad i=1,2,...N$$

- Checkpoint: do you understand this formulation?
- How is this criterion related to maximizing the margin?

- The feature vectors x_i with a corresponding $\lambda_i>0$ are called the support vectors for the problem.
- The classifier defined by this hyperplane is called a Support Vector Machine.
- Depending on y_i (+1 or -1), the support vectors will thus lie on either of the two hyperplanes

  $w^T x+w_0=\pm 1$
- The support vectors are the points in the training set that are closest to the decision hyperplane.
- The optimization has a unique solution, only one hyperplane satisfies the conditions.



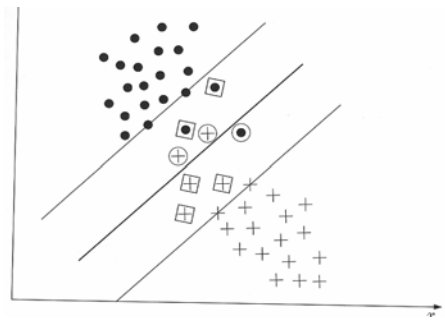
The support vectors for hyperplane 1 are the blue circles.

The support vectors for hyperplane 2 are the red circles.

The nonseparable case

- If the two classes are nonseparable, a hyperplane satisfying the conditions $w^T x-w_0=\pm 1$ cannot be found.
- The feature vectors in the training set are now either:

1. Vectors that fall outside the band and are correctly classified.
2. Vectors that are inside the band and are correctly classified. They satisfy $0\le y_i(w^T x+w_0)<1$ □
3. Vectors that are misclassified – expressed as $y_i(w^T x+w_0)<0$ ○



□ Correctly classified

○ Erroneously classified

Cost function – nonseparable case

- The cost function to minimize is now

$$J(w, w_0, \xi) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} I(\xi_i)$$

$$\text{where} \quad I(\xi_i) = \begin{cases} 1 & \xi_i > 0 \\ 0 & \xi_i = 0 \end{cases}$$

and ξ is the vector of parameters ξ_i.

- **C is a parameter that controls how much misclassified training samples is weighted.**
- We skip the mathematics and present the alternative dual formulation:

$$\max_{\lambda}\left(\sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i,j}\lambda_i\lambda_j y_i y_j x_i^T x_j\right)$$

$$\text{subject to} \quad \sum_{i=1}^{N}\lambda_i y_i = 0 \quad \text{and} \quad 0 \le \lambda_i \le C \quad \forall i$$

- All points between the two hyperplanes ($\xi_i>0$) can be shown to have $\lambda_i=C$.

SVMs: The nonlinear case

- We have now found a classifier that is not defined in terms of the class centres or the distributions, **but in terms of patterns close to the borders between classes, the support vectors.**
- It gives us a solution in terms of a hyperplane. This hyperplane can be expressed as a inner product between the training samples:

$$\max_{\lambda}\left(\sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i,j}\lambda_i\lambda_j y_i y_j x_i^T x_j\right)$$

$$\text{subject to} \quad \sum_{i=1}^{N}\lambda_i y_i = 0 \quad \text{and} \quad 0 \le \lambda_i \le C \quad \forall i$$

- The training samples are l-dimensional vectors.
- What if the classes overlap in l-dimensional space:
  - Can we find a mapping to a higher dimensional space, and use the SVM framework in this higher dimensional space?

- Assume that there exist a mapping from l-dimensional feature space to a k-dimensional space **(k>l)** :

$$x \in R^l \to y \in R^k$$

- Even if the feature vectors are not linearly separable in the input space, they might be separable in a higher dimensional space.
- Classification of a new pattern x is to be *computed by computing the sign of*

$$g(x) = w^T x + w_0$$
$$= \sum_{i=1}^{N_s} \lambda_i y_i x_i^T x_i + w_0$$

- In k-dimensional space, this involves the inner product between two k-dimensional vectors.
- Can it really help to go to a higher dimensional space?

A useful trick: Mercer's theorem – finding a mapping to the high-dimensional space using a kernel

What we need from all this math is just that the inner product can be computed using the kernel K(x,z). Someone has also identified some useful kernels.

Assume that ϕ is a mapping:

$$x \to \phi(x) \in H$$

where H is an Euclidean space.

The inner product has an equivalent representation

$$\sum_r \phi_r(x)\phi_r(z) = K(x,z)$$

where $\phi_r(x)$ is the r-component of the mapping $\phi(x)$ of x, and K(x,z) is a symmetric function satisfying

$$\int K(x,z)g(x)g(z)dxdz \geq 0$$

for any g(x), $x \in R^l$ such that

$$\int g(x)^2 dx < +\infty$$

K(x,z) defines a inner product. K(x,z) is called a kernel.

Once a kernel has been defined, a mapping to the higher dimensional space is defined.

Useful kernels for classification

- Polynomial kernels

$$K(x,z) = \left(x^T z - 1\right)^q, \quad q > 0$$

- Radial basis function kernels (most commonly used)

$$K(x,z) = \exp\left(-\frac{\|x-z\|^2}{\sigma^2}\right)$$

- Hyperbolic tangent kernels (often with β=2 and γ=1)

$$K(x,z) = \tanh\left(\beta x^T \ z + \gamma\right)$$

- The most common type of kernel is the radial basis function. It has an extra parameter σ that must be tuned.

How to use a SVM classifier

- Find a library with all the necessary SVM-functions ☺
  - For example libSVM http://www.csie.ntu.edu.tw/~cjlin/libsvm/
- Read the introductory guide http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf
- Use a radial basis function kernel.
- Scale the data to the range [-1,1] (will not be dominated with features with large values).
- Find the optimal values of C and σ by performing a grid search on selected values and using a validation data set.
- Train the classifier using the best value from the grid search.
- Test using a separate test set.

How to do a grid search

- Use n-fold cross valiation (e.g. 10-fold cross-validation).
  - 10-fold: divide the training data into 10 subsets of equal size. Train on 9 subsets and test on the last subset. Repeat this procedure 10 times.
- Grid search: try pairs of (C,σ). Select the pair that gets the best classification performance on average over all the n validation test subsets.
- Use the following values of C and σ:
    - $C = 2^{-5}, 2^{-3}, ..., 2^{15}$
    - $\sigma = 2^{-15}, 2^{-13}, ..., 2^{3}$