

# Sensor-enabled Intelligent Environments

Nico Blodow, Mihai Dolha, Zoltan-Csaba Marton,  
Dejan Pangercic@IAS

21. Mai 2010





# Today's Roadmap

---

- ▶ Spatial Decompositions
- ▶ Least Squares Fitting
- ▶ Normal Estimation - PCA
- ▶ Segmentation - Region Growing
- ▶ Feature Estimation - \*PFH
- ▶ Registration - ICP
- ▶ Model Fitting
- ▶ Robust Estimation - RANSAC & Co.
- ▶ HW and Q&A



# Regular Voxel Grid

---

Basically an array in  $n$  dimensions

Advantages:

- ▶ trivial implementation
- ▶ fast lookups of cells: ( $O(1)$ )

Disadvantages:

- ▶ also empty space will have grid cells created
- ▶ somewhat inefficient nearest neighborhood searches depending on grid size



# Octree

---

3D equivalent of a simple binary tree  
bounding box gets divided in 8 children  
every node thus holds 8 pointers to the child nodes (or NULL)

Advantages:

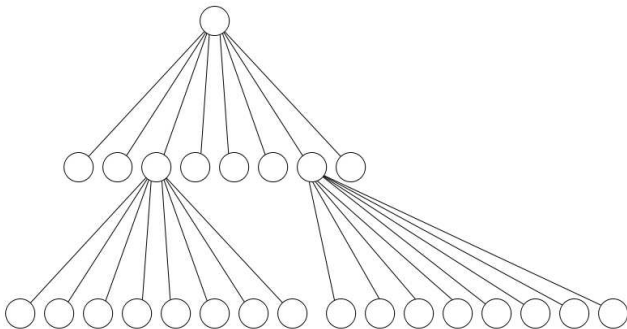
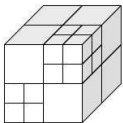
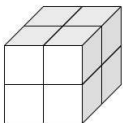
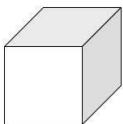
- ▶ for empty space, branches of the tree can be pruned → space efficient
- ▶ still relatively simple implementation
- ▶ fast cell lookup:  $O(\log(n))$

Disadvantages:

- ▶ none, but kd-trees are better for nearest neighbor searches



# Octree

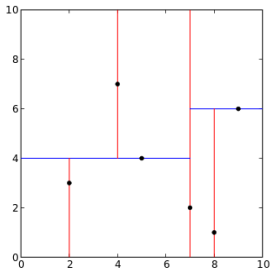


images from <http://en.wikipedia.org/wiki/Octree>

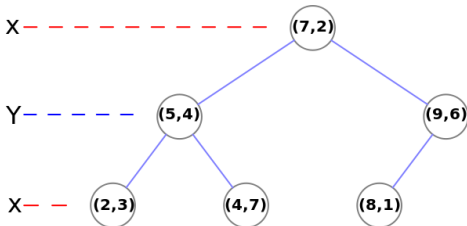


# kd-tree

Instead of splitting a cell into 8 equal children, split in one dimension only (i.e. two children), and iterate through the dimensions:

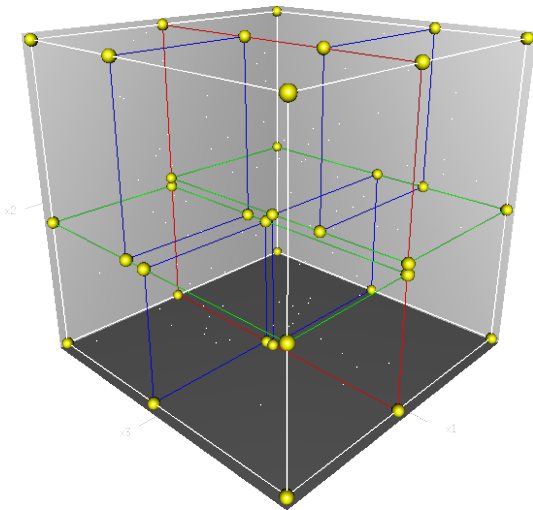


images from <http://en.wikipedia.org/wiki/Kd-tree>





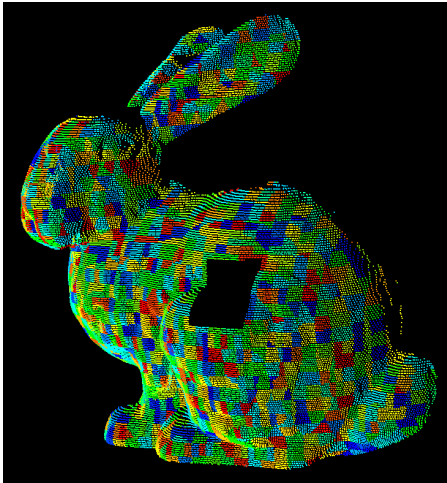
# kd-tree



images from <http://en.wikipedia.org/wiki/Kd-tree>



# kd-tree





# Eigenvectors, Eigenvalues

---

Given a set of 3D points, we want to find out the orientation and smoothness of the surface they lie on. Using (**smallest-**)PCA:

- ▶ Centralize points (subtract centroid)
- ▶ Compute covariance matrix
- ▶ Get eigenvalues and eigenvectors from a decomposition (SVD)
- ▶ Eigenvector corresponding to smallest eigenvalue is the normal of the best fitting plane in the least square sense
- ▶ Eigenvalues are **proportional** to variations along vectors
- ▶ Proportion of smallest eigenvalue with the others is a measure of flatness



# Local Coordinate System

---

- ▶ Usually it is done using for a neighborhood, with results saved for the query point  $q$
- ▶ Points can be weighed by distance from  $q$  ( $e^{-d^2/g^2}$ ) for a more local approximation
- ▶ Iterative Weighted Least Squares approximates tangent plane
- ▶ IWLS: weighting, least squares plane fitting (PCA), projecting  $q$  on plane and repeat until it converges
- ▶ Eigenvectors with  $q$  will form a local coordinate system (dot product of axis with  $(p - q)$  gives local coordinate)



# Linear Systems of Equations

---

Given a set of points, find a function that best approximates them (MLS)

- ▶ Height function in a local coordinate system:  $z=f(x,y)$
- ▶ Every real function can be approximated with a polynomial of enough degree (see Taylor series)
- ▶ We fix the order ( $3^{rd}$  order bivariate polynomial has 2 directions of curvedness) and compute coefficients
- ▶ Each point gives a relation between the powers of the (local) coordinates (matrix  $A$ ) and the height (vector  $b$ )  
 $\rightarrow A * c = b$ , where  $c$  is the list of unknown coefficients
- ▶ Weighting is possible here as well  
 $\rightarrow A * W * A' * c = A * W * b$ , where  $W$  contains the weights for each point on the diagonal
- ▶ Surface normals can be computed from partial derivatives



# Zoli

---

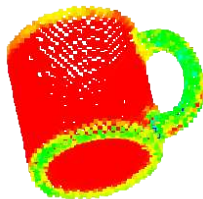
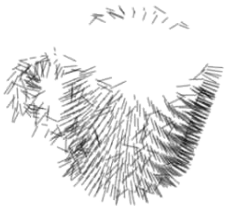


# Normal Estimation

## Normal and Curvature Estimation

$$C_j = \sum_{i=1}^k (p_i - \bar{p}_j)^T \cdot (p_i - \bar{p}_j), \quad \bar{p} = \frac{1}{k} \cdot \sum_{i=1}^k p_i$$

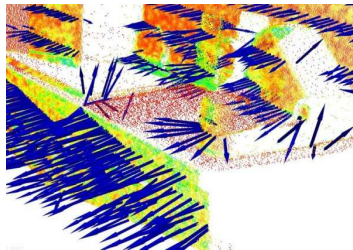
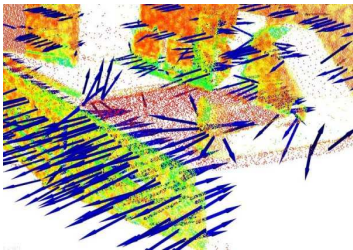
$$\sigma_p = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2}$$





# Normal Estimation

## Normal Consistency



**left:** after normal estimation, normal vectors are generally randomly oriented

**right:** propagating normal orientation consistency (flip normals to view point or more complex methods)



# Region Growing

## Basic Algorithm

init list of regions:  $R = \{\}$ , counter  $i = 0$

while there are points left:

- ▶  $p_{s,i} := \dots$  // *select a seed point*
- ▶  $R_i := \{p_{s,i}\}$ , pivot  $c = 0$
- ▶ while ( $c < \|R_i\|$ ):
  - ▶  $N := \{q_j \mid \text{dist}(q_j, R_i[c]) < d_{\text{thresh}}\}$  // *select neighborhood*
  - ▶ for all  $q_j \in N$ :
    - ▶ if  $q_j$  satisfies some selection criterion:
      - ▶  $R_i \leftarrow R_i \cup \{q_j\}$
  - ▶  $c++$
- ▶  $i++$
- ▶  $R \leftarrow R \cup R_i$



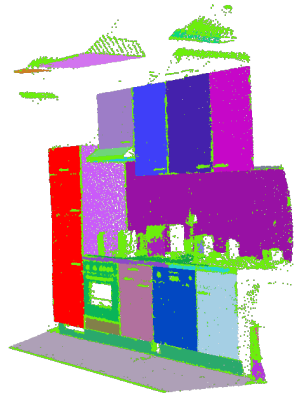
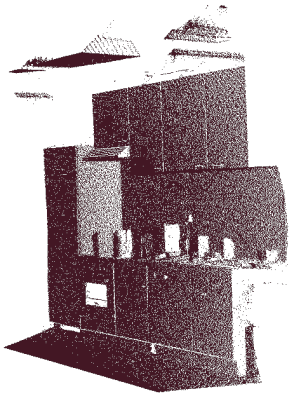
# Region Growing

---

- ▶ seed point selection:  
e.g. point with lowest curvature (flattest point)
- ▶ point selection criterion:  
e.g. curvature below certain threshold  
this threshold could be chosen according to the curvature distribution within the remaining point cloud
- ▶ neighborhood selection: kd\_trees



# Region Growing

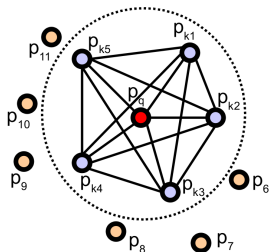




# Feature Estimation

## Feature Histograms

- ▶ For every point pair  $\langle (p_s, n_s); (p_t, n_t) \rangle$ , let  
 $u = n_s$ ,  $v = (p_t - p_s) \times u$ ,  $w = u \times v$



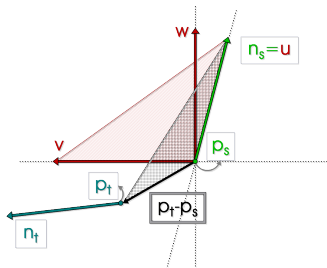
$$\left. \begin{aligned}
 f_0 &= \langle v, n_j \rangle \\
 f_1 &= \langle u, p_j - p_i \rangle / \|p_j - p_i\| \\
 f_2 &= \|p_j - p_i\| \\
 f_3 &= \text{atan}(\langle w, n_j \rangle, \langle u, n_j \rangle)
 \end{aligned} \right\} i_{hist} = \sum_{x=0}^{x \leq 3} \left[ \frac{f_x \cdot d}{f_{x_{max}} - f_{x_{min}}} \right] \cdot d^x$$



# Feature Estimation

## Feature Histograms

- For every point pair  $\langle (p_s, n_s); (p_t, n_t) \rangle$ , let  
 $u = n_s$ ,  $v = (p_t - p_s) \times u$ ,  $w = u \times v$



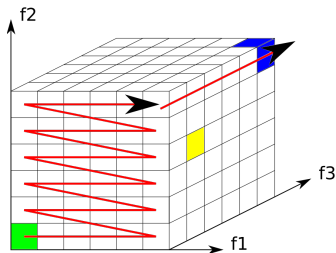
$$\left. \begin{aligned} f_0 &= \langle v, n_j \rangle \\ f_1 &= \langle u, p_j - p_i \rangle / \|p_j - p_i\| \\ f_2 &= \|p_j - p_i\| \\ f_3 &= \text{atan}(\langle w, n_j \rangle, \langle u, n_j \rangle) \end{aligned} \right\} i_{hist} = \sum_{x=0}^{x \leq 3} \left[ \frac{f_x \cdot d}{f_{x_{max}} - f_{x_{min}}} \right] \cdot d^x$$



# Feature Estimation

## Feature Histograms

- For every point pair  $\langle (p_s, n_s); (p_t, n_t) \rangle$ , let  
 $u = n_s$ ,  $v = (p_t - p_s) \times u$ ,  $w = u \times v$

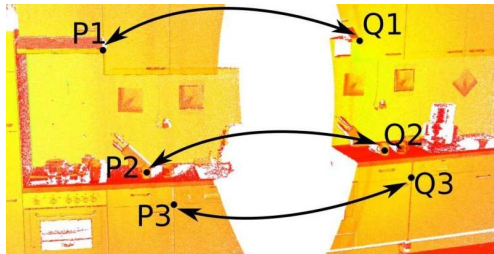


$$\left. \begin{aligned} f_0 &= \langle v, n_j \rangle \\ f_1 &= \langle u, p_j - p_i \rangle / \|p_j - p_i\| \\ f_2 &= \|p_j - p_i\| \\ f_3 &= \text{atan}(\langle w, n_j \rangle, \langle u, n_j \rangle) \end{aligned} \right\} i_{hist} = \sum_{x=0}^{x \leq 3} \left[ \frac{f_x \cdot d}{f_{x_{max}} - f_{x_{min}}} \right] \cdot d^x$$

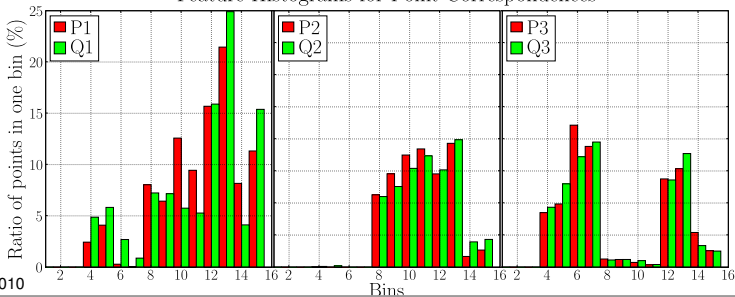


# Feature Estimation

## Example Histograms



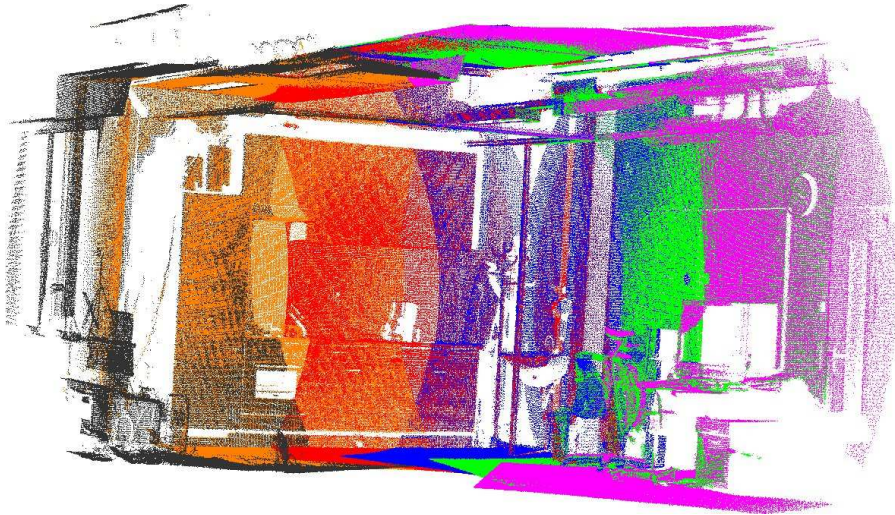
Feature Histograms for Point Correspondences





# Feature Estimation

## Registration



Merging multiple views (>10 million points)



# Iterative Closest Point

---

Given two datasets, find the best alignment.

- ▶ For each point (or a subset) from one dataset find the best matching point from the other (best matching can be interpreted as closeness in Euclidean and/or feature space)
- ▶ Compute transformation between them that minimizes the error (either least square error using SVD or some non-linear optimization like LM)
- ▶ Transform the point cloud and repeat until convergences

Gets stuck in local minima, so a good initial "guess" of correspondences or alignment is needed.



# Hough Space and Parameter Clusterization

Given a model to be searched for in the data, estimate possible model parameters from each point/tuple (or subset) and vote for most probable parameters.

- ▶ Hough Transform has a discretization of the parameter space and counts votes
- ▶ For big parameter spaces Randomized Hough Transform can be used, but it still has the problem of discretization
- ▶ Parameter clustering is the continuous case, where each parameter tuple is treated as a point, and the searched values are computed from the clustering
- ▶ Densities in the continuous case are taking the place of votes
- ▶ Different clustering methods can be employed to find the best model (same approaches as for classification)



# Random Sample Consensus

---

Robust estimator by checking the correctness of models found using random samples

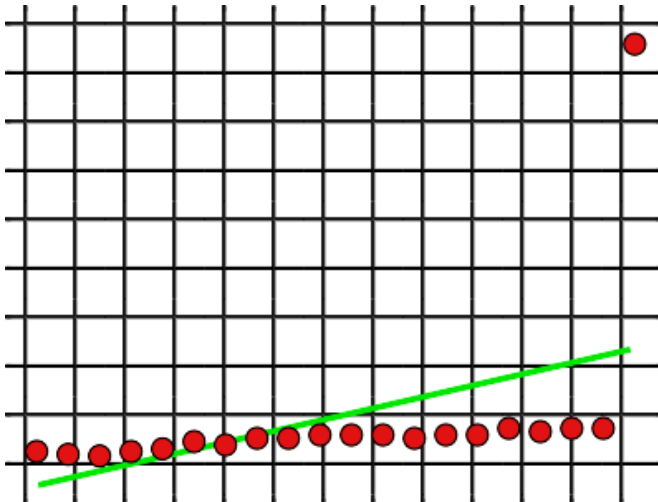
- ▶ Pick random samples
- ▶ Compute model using the samples
- ▶ Check model against data
- ▶ Store it if this is the best fit so far
- ▶ (Re-)estimate the number of trials needed for finding the best model (with a user defined precision)
- ▶ Stop if number of needed iterations reached

Different scoring functions can be used and other optimizations, but the key problem remains how to compare which model fits best to the data (as in the previous methods as well).



# Outliers

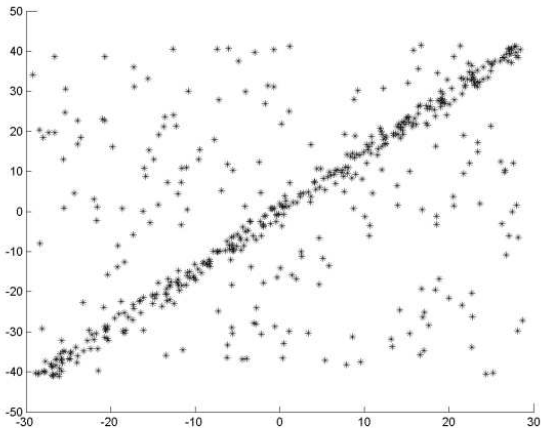
## Effect of outliers on linear mean square estimation



images from <http://de.wikipedia.org/wiki/RANSAC-Algorithmus>



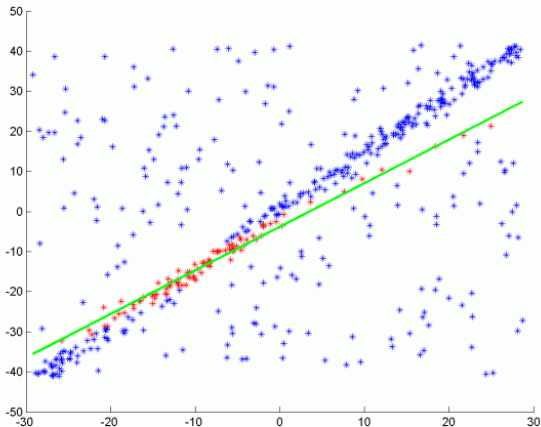
# Example data set



images from <http://de.wikipedia.org/wiki/RANSAC-Algorithmus>



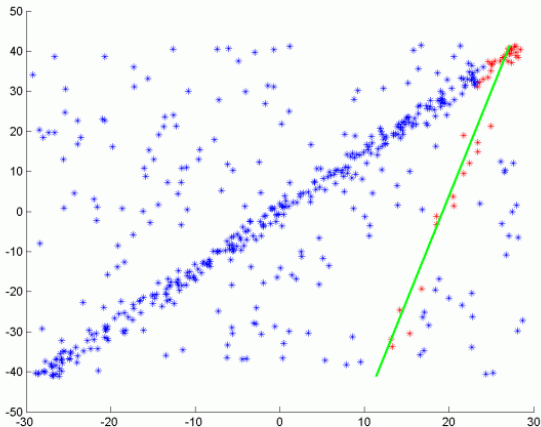
# Animation of RANSAC iterations



images from <http://de.wikipedia.org/wiki/RANSAC-Algorithmus>



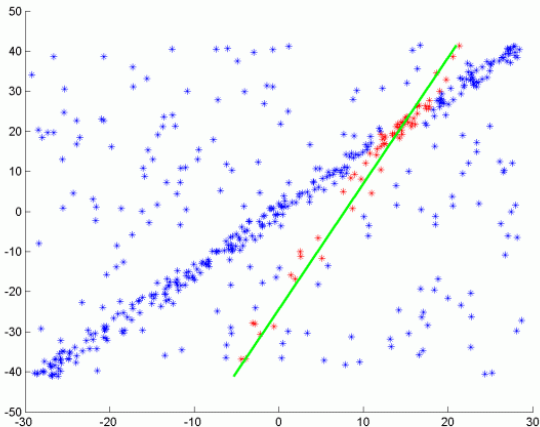
# Animation of RANSAC iterations



images from <http://de.wikipedia.org/wiki/RANSAC-Algorithmus>



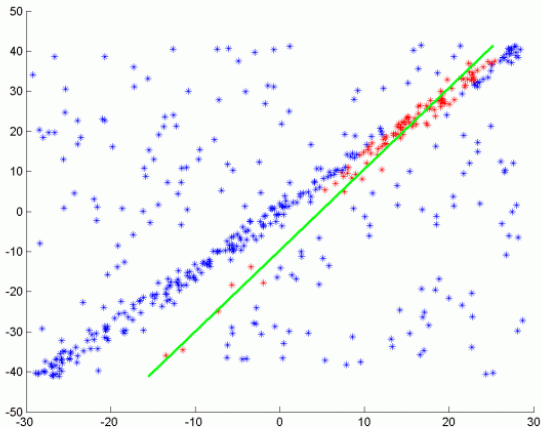
# Animation of RANSAC iterations



images from <http://de.wikipedia.org/wiki/RANSAC-Algorithmus>



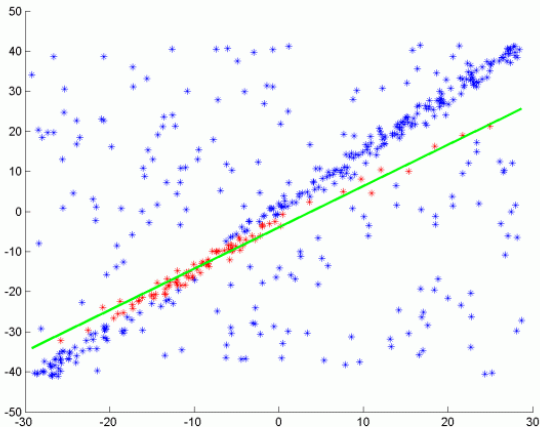
# Animation of RANSAC iterations



images from <http://de.wikipedia.org/wiki/RANSAC-Algorithmus>



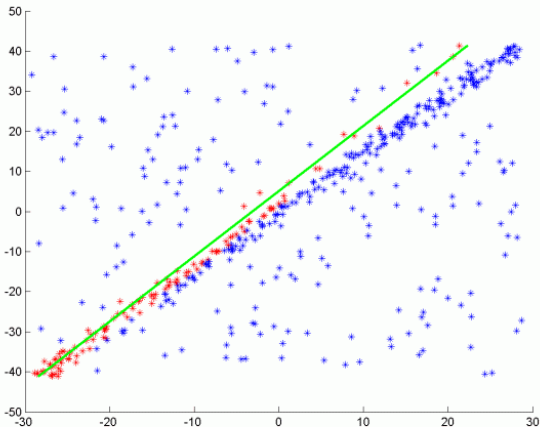
# Animation of RANSAC iterations



images from <http://de.wikipedia.org/wiki/RANSAC-Algorithmus>



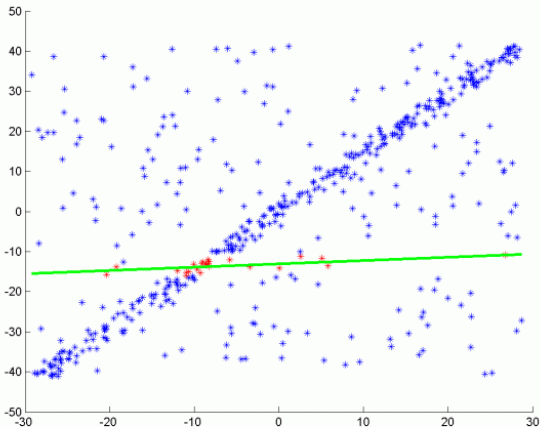
# Animation of RANSAC iterations



images from <http://de.wikipedia.org/wiki/RANSAC-Algorithmus>



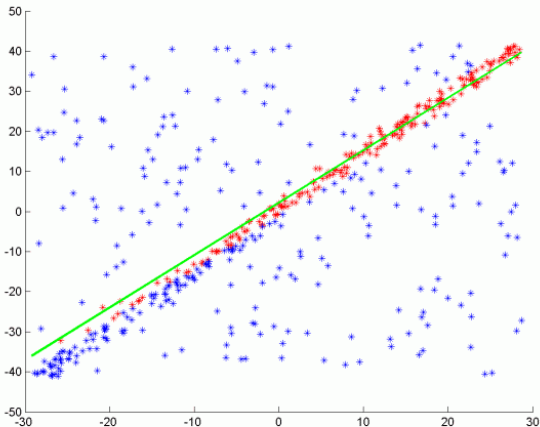
# Animation of RANSAC iterations



images from <http://de.wikipedia.org/wiki/RANSAC-Algorithmus>



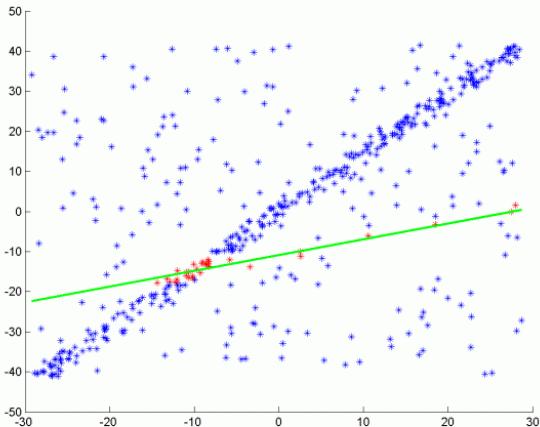
# Animation of RANSAC iterations



images from <http://de.wikipedia.org/wiki/RANSAC-Algorithmus>



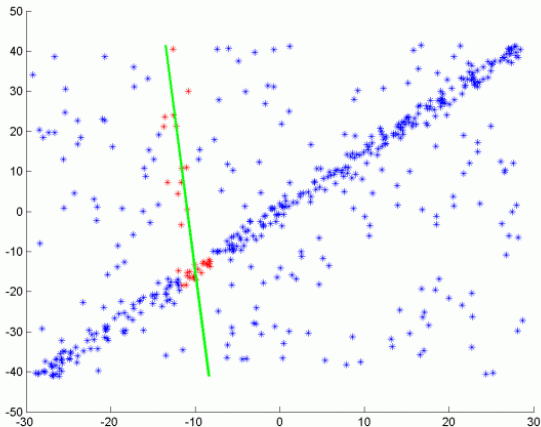
# Animation of RANSAC iterations



images from <http://de.wikipedia.org/wiki/RANSAC-Algorithmus>



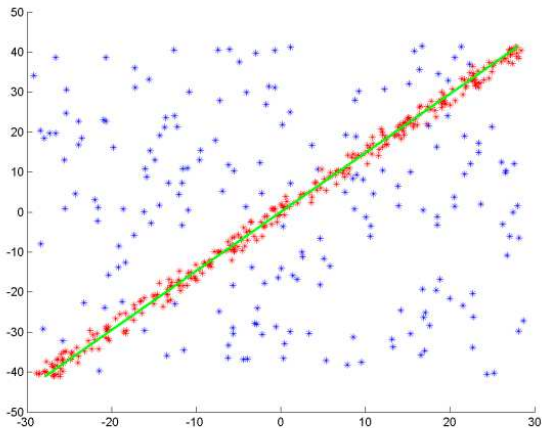
# Animation of RANSAC iterations



images from <http://de.wikipedia.org/wiki/RANSAC-Algorithmus>



# Result - Best Line

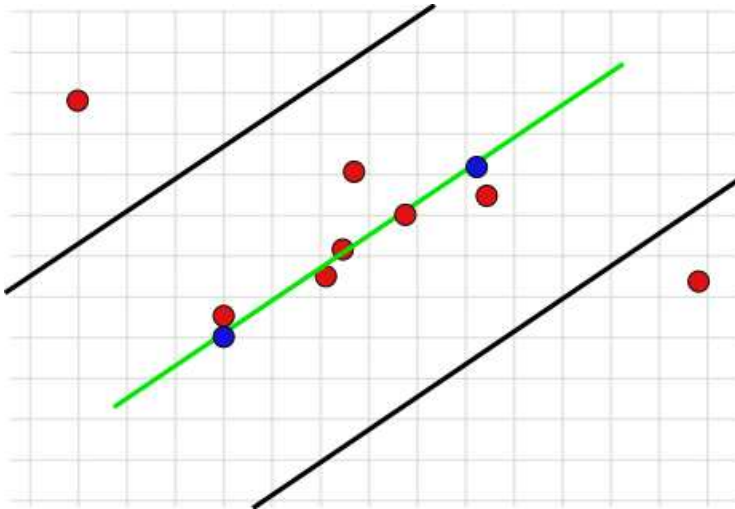


images from <http://de.wikipedia.org/wiki/RANSAC-Algorithmus>



# Careful Selection of distance threshold

Threshold might seem good

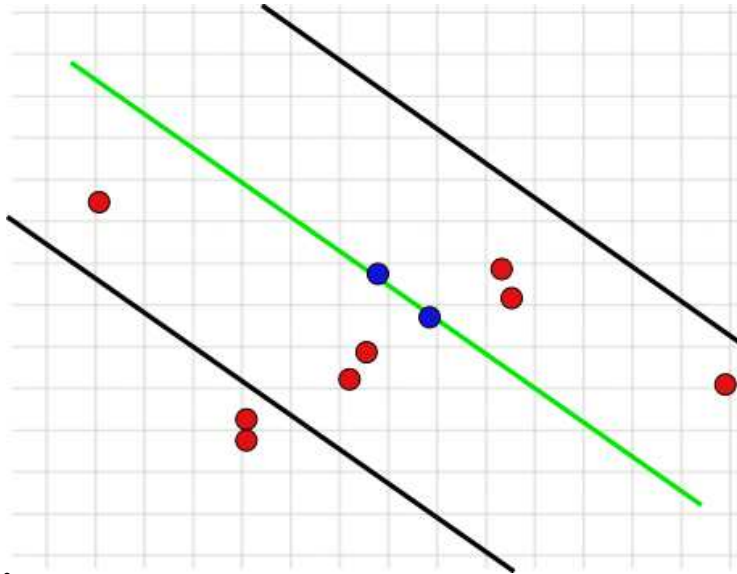


images from <http://de.wikipedia.org/wiki/RANSAC-Algorithmus>



# Careful Selection of distance threshold

Threshold is in fact too high



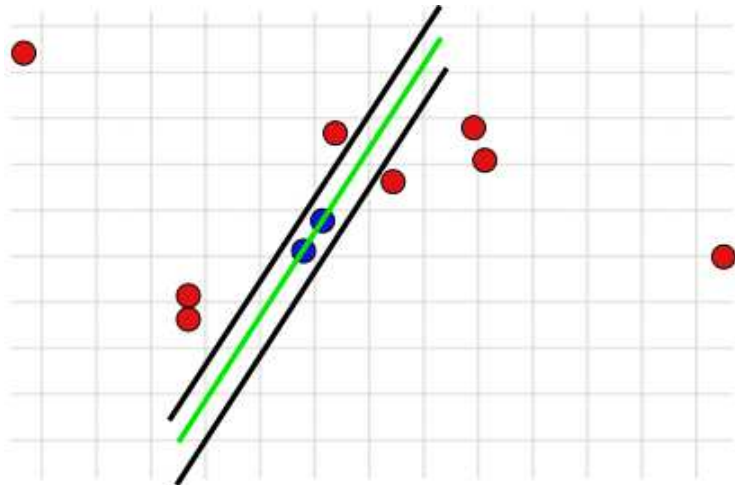
21. Mai 2010





# Careful Selection of distance threshold

Threshold is too low



images from <http://de.wikipedia.org/wiki/RANSAC-Algorithmus>



# Homework

---

1. create a 2D dataset with 500 points random and 500 points on a arbitrary line (plus small amount of gaussian noise)
2. range on x and y dimensions: [0.0-1.0]
3. visualize the hough space for a line fitting problem for this dataset  
(PPM file format might be helpful...)
4. write a RANSAC estimator for the line equation and run it on the same dataset  
(wikipedia is your friend...)
5. compare the line parameter from the maximum vote from Hough space with those from RANSAC and from the original line equation

## Questions?