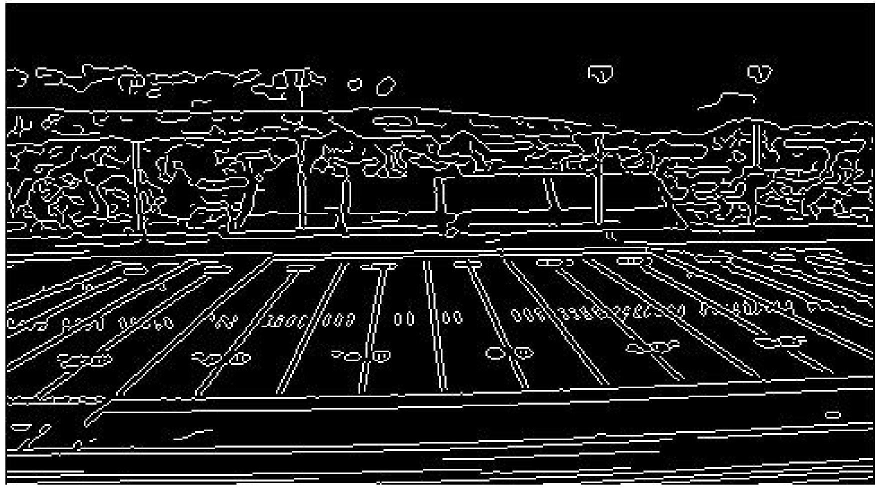
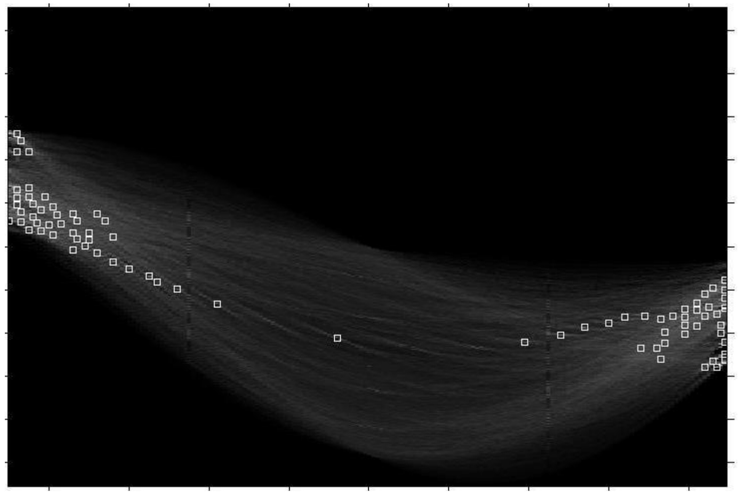
Previously at MP…

- Edge detection (Canny)



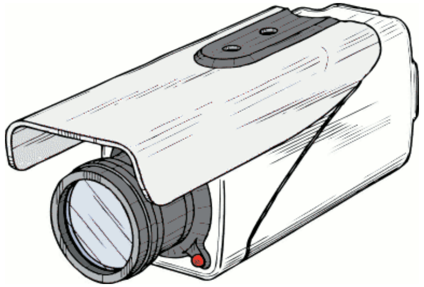
- Fitting parametric models of shapes by voting (Hough transform)
  - Lines
  - Circles
  - General shapes

Univerza v Ljubljani

Machine perception
Fitting parametric models

Matej Kristan



Laboratorij za Umetne Vizualne Spoznavne Sisteme,
Fakulteta za računalništvo in informatiko,
Univerza v Ljubljani

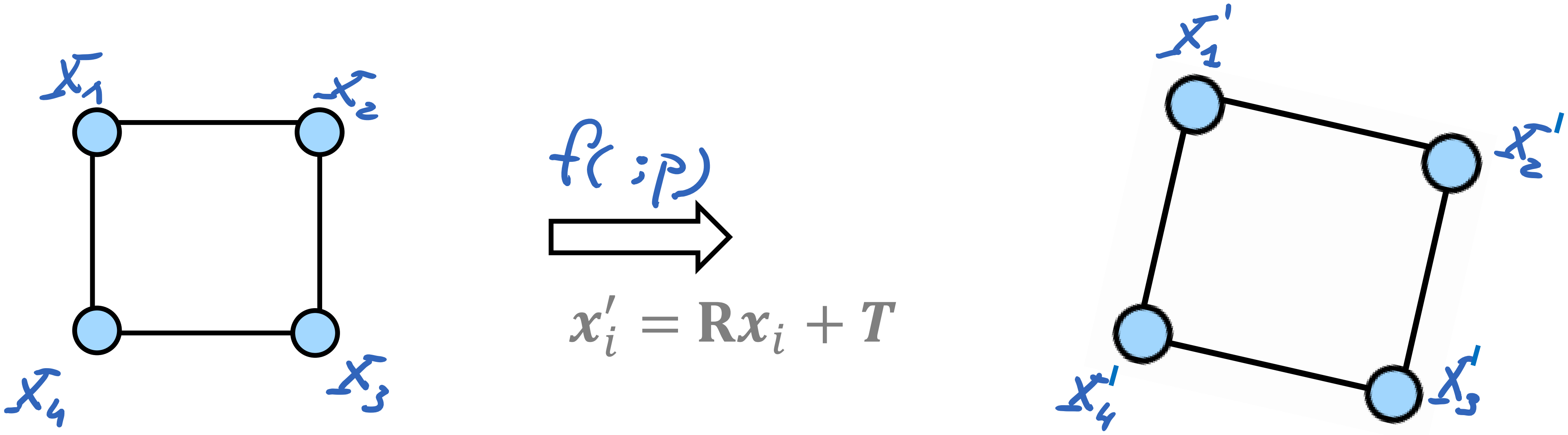
Parametric models: Forward application

- Transformation parameterized by (many) parameters

$$\mathbf{x}_i' = f(\mathbf{x}_i; \mathbf{p})$$

- Example: transform $\boldsymbol{x}_i$ into $\boldsymbol{x}_i'$ by a function $f(\boldsymbol{x}; \boldsymbol{p})$

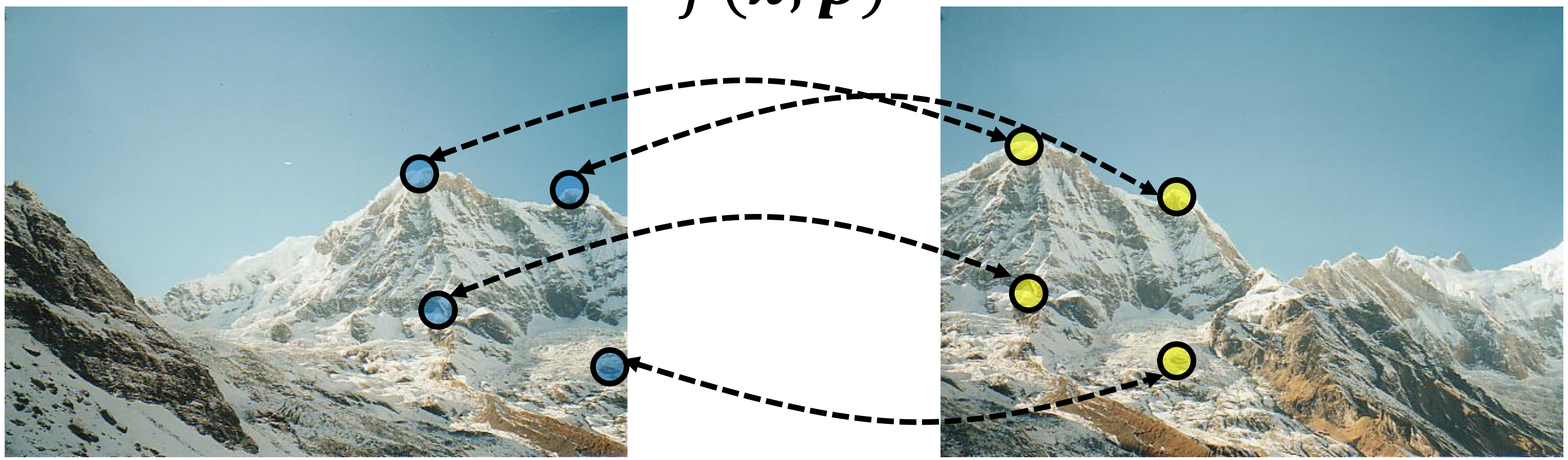
$$x_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \in \mathbb{R}^2 \qquad p = [p_1, p_2, p_3, \cdots, p_M] \in \mathbb{R}^M$$



Parametric models: Use cases

- Inverse problem: ``*Given a set of correspondences, what are the parameters of the transformation?*''

$f(\boldsymbol{x}; \boldsymbol{p})$



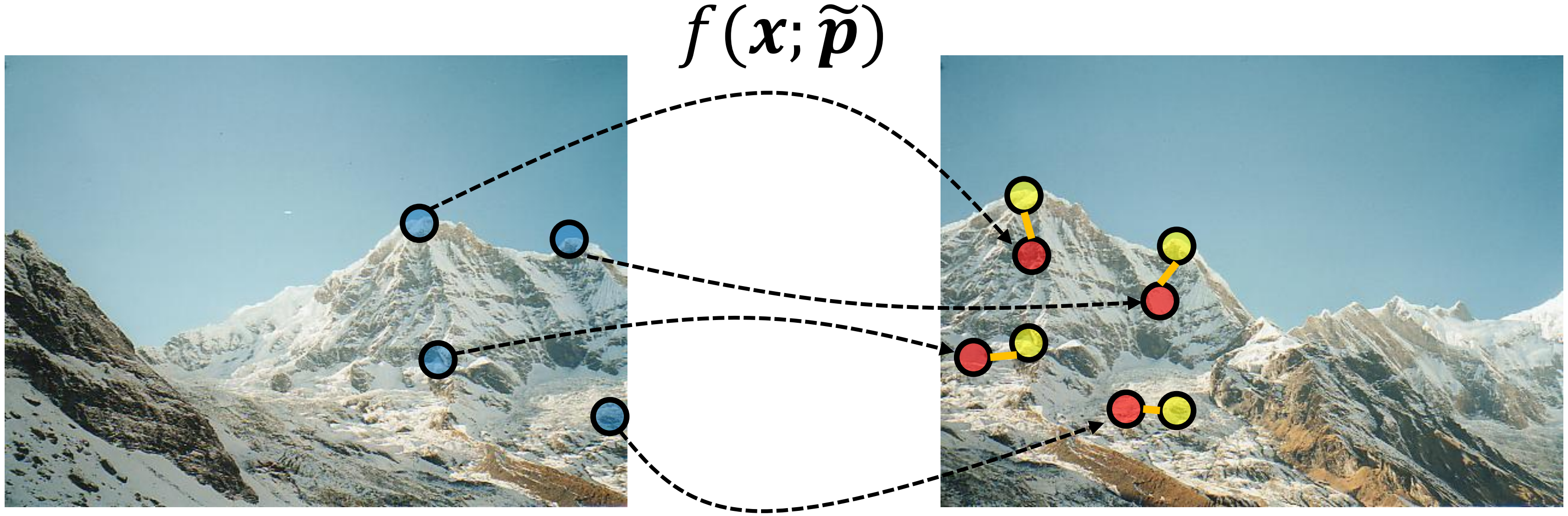
Source keypoint

Destination keypoint

- Assuming the transformation can be well approximated by $f(\boldsymbol{x}; \boldsymbol{p})$, what are the best parameter values for $\boldsymbol{p}$?

Parametric models: Use cases

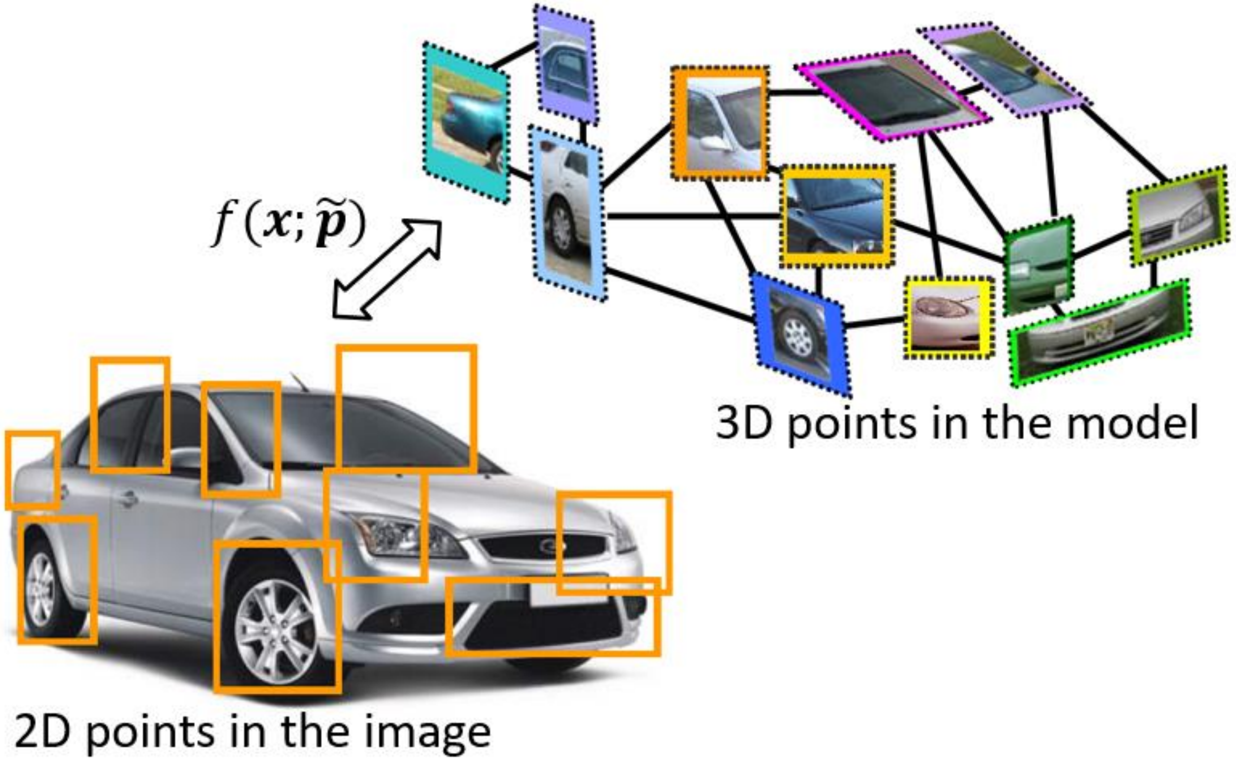
- Best parameter values: *those that minimize the projection error*



Stitched images:
Coordinates of all pixels in the left-hand image transformed by $f(x;\tilde{p})$

Parametric models: Use cases

3D pose estimation problem



Find rotation+translation of the model in 3D, such that the 2D projections of the 3D model parts into the camera, match the observed image of a car.

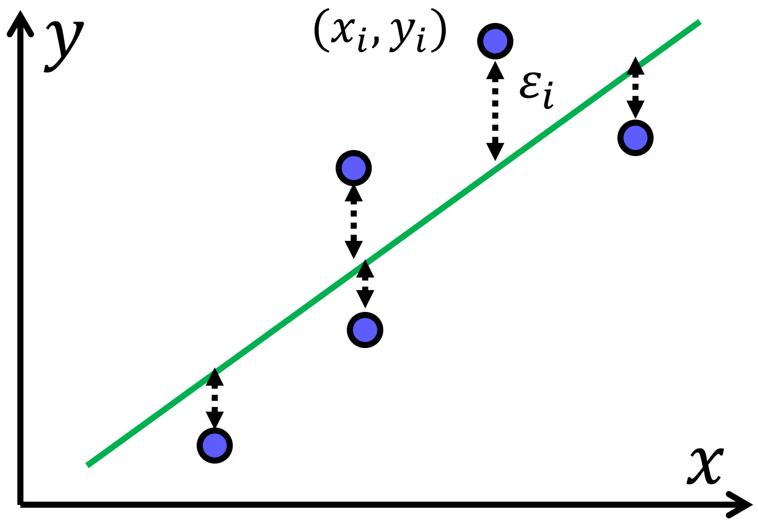
3D pose estimation in action

TLD3.0 - 3D Tracking of Rigid Objects (ICCV 2017 demo)

https://www.youtube.com/watch?v=i3cg8spZCrY

Least squares: Line fitting

Problem formulation



- Data: $\{(x_1, y_1), \dots, (x_N, y_N)\}$
- Line equation:
  $y = f(x; \boldsymbol{p}) = xp_1 + p_2$
- Parameters:
  $\boldsymbol{p} = [p_1, p_2]^T$
- Projection error at i-th correspondence:
  $\varepsilon_i = f(x_i; \boldsymbol{p}) - y_i$
- The cost function (goodness of fit): $E(\mathbf{p}) = \sum_{i=1}^{N} {\varepsilon_i}^2$
- Best parameters: $\tilde{\mathbf{p}} = \arg\min_{\mathbf{p}} E(\mathbf{p})$

A 1D minimization

Least squares: Line fitting

Strategy:

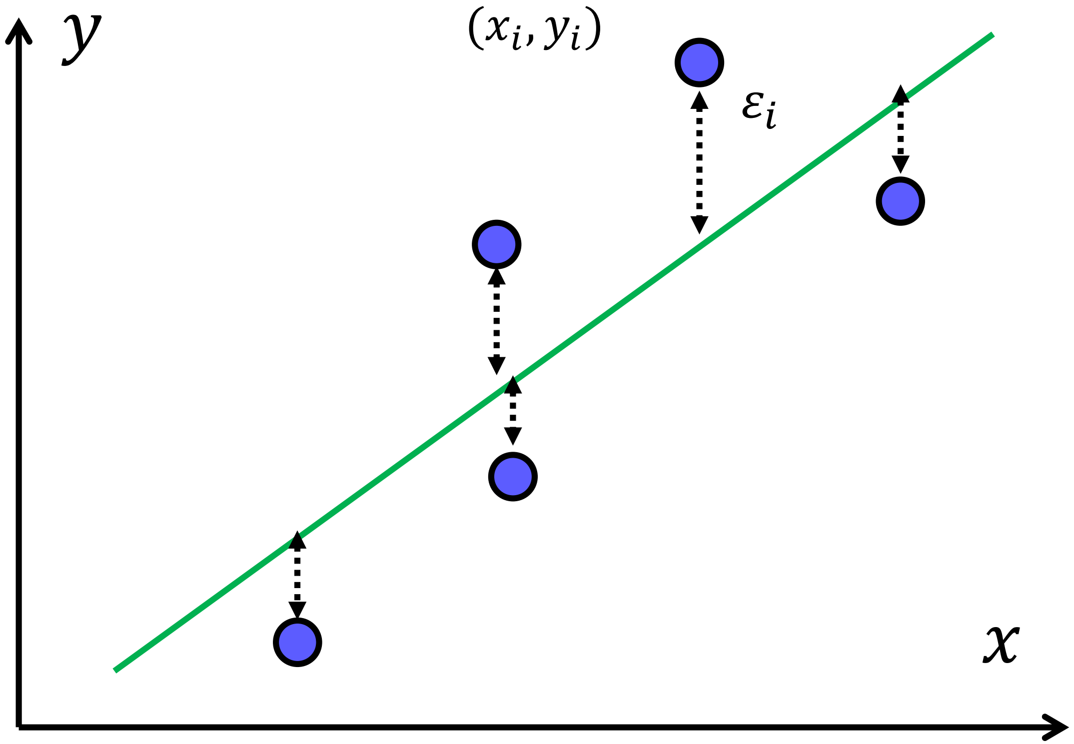
1. Rewrite the cost function $E(\boldsymbol{p})$ into a vector-matrix form
2. Take derivative w.r.t. $\boldsymbol{p}$, set to zero, solve for $\boldsymbol{p}$.

$$f(x;\boldsymbol{p}) = xp_1 + p_2$$

$$\boldsymbol{p} = [p_1, p_2]^T$$

$$\varepsilon_i = f(x_i;\boldsymbol{p}) - y_i$$

$$E(\mathbf{p}) = \sum_{i=1}^{N} {\varepsilon_i}^2$$



Least squares: Line fitting

Strategy:

- Rewrite the cost function $E(\boldsymbol{p})$ into a vector-matrix form
- Take derivative w.r.t. $\boldsymbol{p}$, set to zero, solve for $\boldsymbol{p}$.

$$E(\mathbf{p}) = \sum_{i=1}^{N} \left(y_i - [x_i, 1] \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \right)^2 = \left\| - \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} + \begin{bmatrix} x_1, 1 \\ \vdots \\ x_N, 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \right\|^2 = \|-\mathbf{b} + \mathbf{A}\mathbf{p}\|^2$$

Normal equation:

$$\frac{dE(\mathbf{p})}{d\mathbf{p}} = 2\mathbf{A}^T\mathbf{A}\mathbf{p} - 2\mathbf{A}^T\mathbf{b} \equiv \mathbf{0}$$

Solution:

$$\mathbf{p} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} = \mathbf{A}^\dagger\mathbf{b}$$

Pseudoinverse:

$$\mathbf{A}^\dagger = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$$

$$\mathbf{A} \overset{\mathrm{SVD}}{=} \mathbf{U}\mathbf{S}\mathbf{V}^T$$

$$\mathbf{A}^\dagger = \mathbf{V}\mathbf{S}^{-1}\mathbf{U}^T$$

A cookbook for normal equations:

1. Define the set of corresponding points
   $\{\boldsymbol{x}_i\}_{i=1:N}$, $\{\boldsymbol{x}'_i\}_{i=1:N}$
2. Define the linear transformation
   $f(\boldsymbol{x};\boldsymbol{p})\colon \boldsymbol{x} \to \boldsymbol{x}'$
3. Define the per-point error and stack all errors into a single vector $\boldsymbol{\varepsilon}$:

$$E(\mathbf{p}) = \sum_{i=1}^{N} {\varepsilon_i}^2$$

$$\boldsymbol{\varepsilon}_i = f(\boldsymbol{x}_i;\boldsymbol{p}) - \boldsymbol{x}_i{}'$$

$$\varepsilon = \left[\varepsilon_1^T, \ldots, \varepsilon_i^T, \ldots, \varepsilon_N^T\right]^T$$

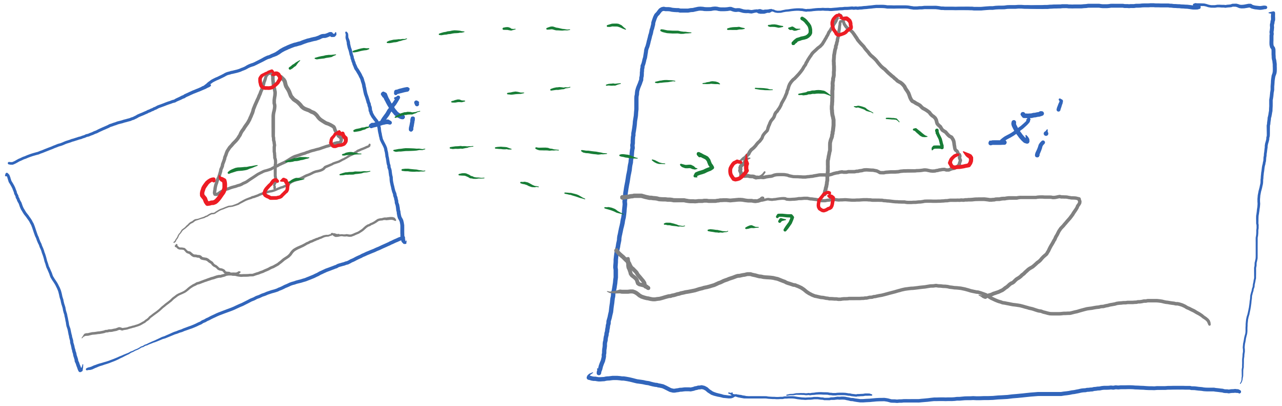
4. Rewrite the error into a form $\boldsymbol{\varepsilon} = \boldsymbol{A}\boldsymbol{p} - \boldsymbol{b}$
5. Solve by pseudoinverse: $\boldsymbol{p} = \boldsymbol{A}^{\dagger}\mathbf{b}$

Note: point errors $\boldsymbol{\varepsilon}_i$ are of same dimensionality as the points $\boldsymbol{x}'$.

Matlab: p = A \ b

Least squares: A simple image alignment

- Task: Align two images based on correspondences



- Assume a similarity transform (scale, rotation, translation)

$$\boldsymbol{x}' = f(\boldsymbol{x}; \boldsymbol{p})$$

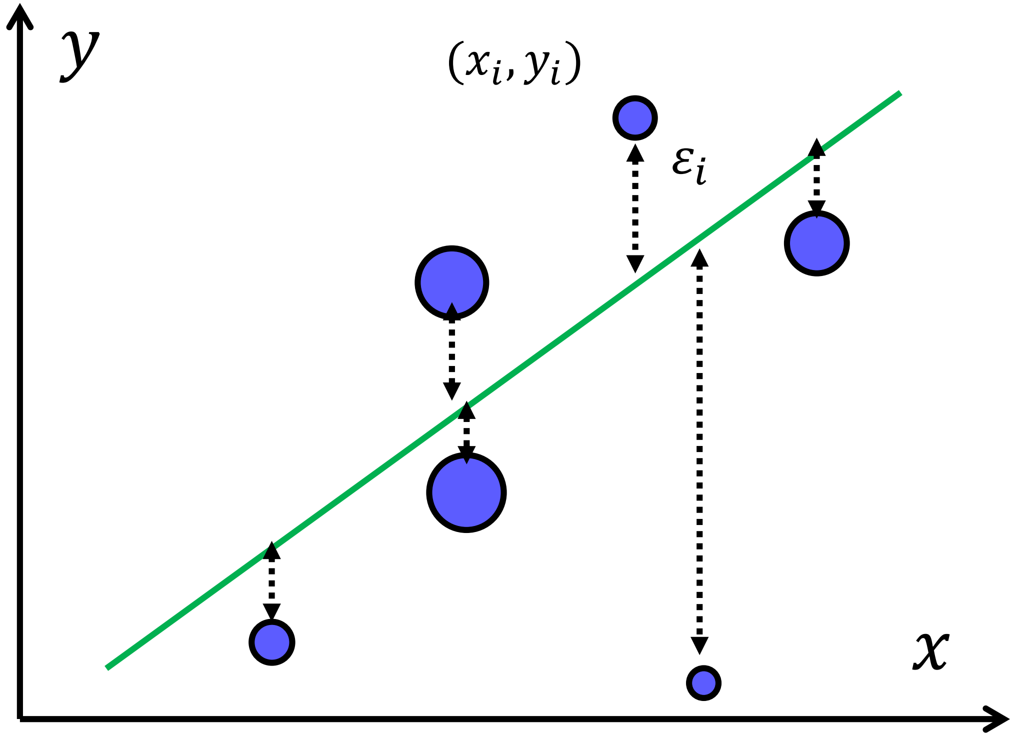
- The similarity transform is parameterized by (See Szeliski, Section 2.1.2):

$$x_i' = \begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} p_1 x_i - p_2 y_i + p_3 \\ p_2 x_i + p_1 y_i + p_4 \end{bmatrix} \; ; \; p = [p_1 \; p_2 \; p_3 \; p_4]^T$$

Weighted least squares: Line fitting

Problem formulation

- Data: $\{(x_1, y_1), \ldots, (x_N, y_N)\}$
- All points are *not* equally accurately measured!
- Weight at each point: w_i
- Projection error at i-th correspondence:
  $\varepsilon_i = f(x_i; \boldsymbol{p}) - y_i$
- A weighted cost: $E(\mathbf{p}) = \sum_{i=1}^{N} w_i {\varepsilon_i}^2$
- Best parameters: $\tilde{\mathbf{p}} = \arg\min_{\mathbf{p}} E(\mathbf{p})$



Weighted least squares: Line fitting

Strategy:

- Rewrite the cost function $E(\boldsymbol{p})$ into a vector-matrix form
- Take derivative w.r.t. $\boldsymbol{p}$, set to zero, solve for $\boldsymbol{p}$.

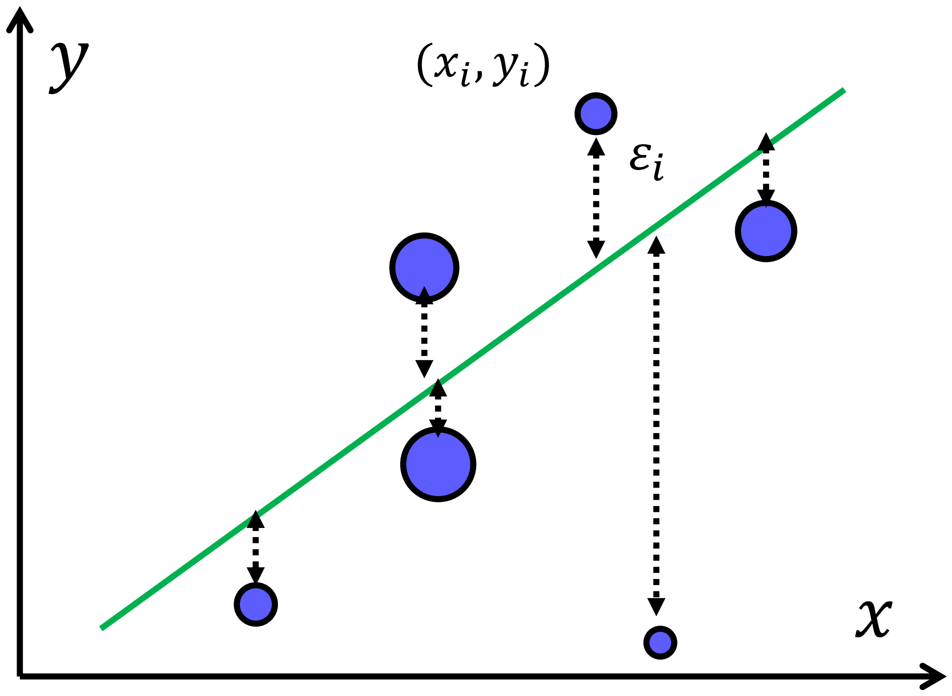
$$f(x;\boldsymbol{p}) = xp_1 + p_2$$

$$\boldsymbol{p} = [p_1, p_2]^T$$

$$\varepsilon_i = f(x_i;\boldsymbol{p}) - y_i$$

$$E(\mathbf{p}) = \sum_{i=1}^{N} w_i {\varepsilon_i}^2$$

$$\tilde{\mathbf{p}} = \arg\min_{\mathbf{p}} E(\mathbf{p})$$



Weighted least squares: Line fitting

Strategy:

- Rewrite the cost function $E(\boldsymbol{p})$ into a vector-matrix form
- Take derivative w.r.t. $\boldsymbol{p}$, set to zero, solve for $\boldsymbol{p}$.

$$E(\mathbf{p}) = \sum_{i=1}^{N} w_i \left(y_i - [x_i, 1] \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \right)^2$$

$$E(\mathbf{p}) = [\varepsilon_1, ..., \varepsilon_N] \begin{bmatrix} w_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & w_N \end{bmatrix} \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_N \end{bmatrix} = \varepsilon^T \mathbf{W} \varepsilon$$

$$\frac{dE(\mathbf{p})}{d\mathbf{p}} = 2\mathbf{A}^T\mathbf{W}\mathbf{A}\mathbf{p} - 2\mathbf{A}^T\mathbf{W}\mathbf{b} \equiv \mathbf{0}$$ ⟵ Normal equation

$$\mathbf{p} = (\mathbf{A}^T\mathbf{W}\mathbf{A})^{-1}\mathbf{A}^T\mathbf{W}\mathbf{b}$$

A cookbook for weighted least squares:

1. Define a weighted set of corresponding points
   $\{\boldsymbol{x}_i\}_{i=1:N}$, $\{\boldsymbol{x}'_i\}_{i=1:N}$, $\{w_i\}_{i=1:N}$
2. Define the linear transformation
   $f(\boldsymbol{x};\boldsymbol{p})\colon \boldsymbol{x} \to \boldsymbol{x}'$
3. Rewrite the error into a form $\boldsymbol{\varepsilon} = \boldsymbol{A}\boldsymbol{p} - \boldsymbol{b}$
4. Create a weight matrix $\boldsymbol{W}$ as
   $W = diag([\boldsymbol{w}_1^T, \ldots, \boldsymbol{w}_N^T])$
   with $\boldsymbol{w}_i^T = w_i[1, .., 1]_{1\times d}$
5. Solve by : $\mathbf{p} = (\mathbf{A}^T\mathbf{W}\mathbf{A})^{-1}\mathbf{A}^T\mathbf{W}\mathbf{b}$

Note: $\boldsymbol{x}' \in \mathbb{R}^d, w \in \mathbb{R}^1$

Note: think about why are $\boldsymbol{w}_i^T$ vectors of same dimensionality as the points $\boldsymbol{x}'$.

To practice: solve the "sailboat" example

NOTE

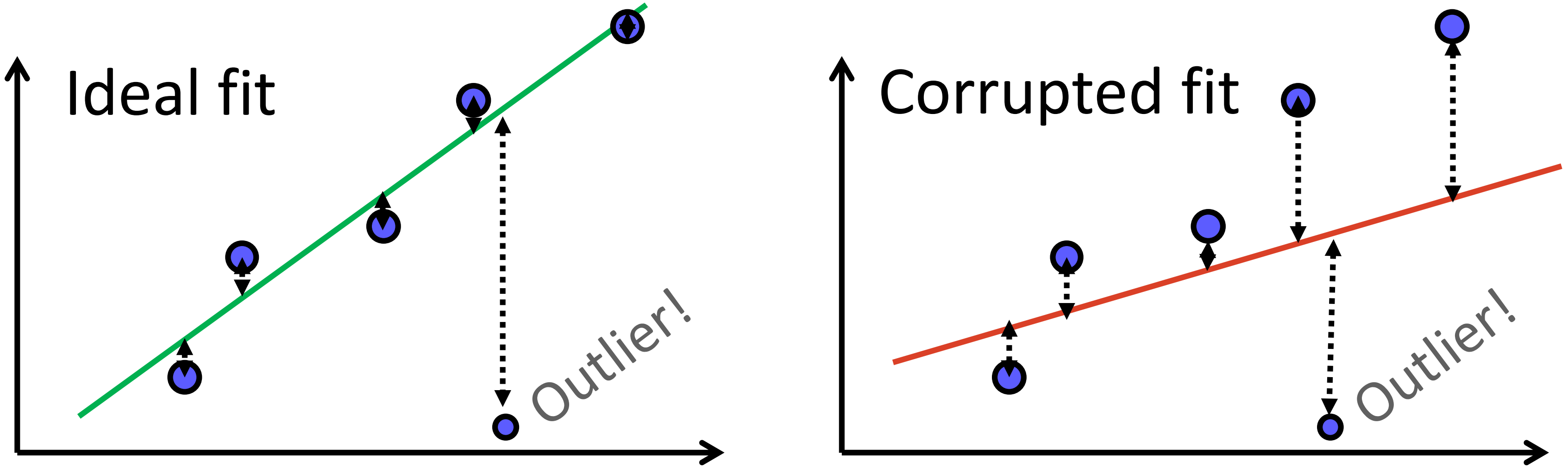
- Weighted least squares can be used for NONLINEAR/ROBUST least-squares problems as well!

- Robust least squares, for example can be implemented by iterative algorithm that applies a weighted least squares solver

- See the slides on e-classroom if you're interested

Robust least squares

- Quadratic cost function behaves poorly with outliers:



- To see where the problem lies, we will have to rewrite our cost function into a general form.
- The cost can be generally written as: $E(\mathbf{p}) = \sum_{i=1}^{N} h(\varepsilon_i)$
- For ordinary least squares we had: $h(\boldsymbol{\varepsilon}_i) = ||\boldsymbol{\varepsilon}_i||^2$

Not covered in 2021/22 lectures – Consider this as additional material

Robust least squares

- For a cost function with robust error function $h(\varepsilon_i)$

$$E(\mathbf{p}) = \sum_{i=1}^{N} h(\varepsilon_i)$$

- It is possible to find an equivalent weighted L_2 cost

$$E_W(\mathbf{p}) = \sum_{i=1}^{N} w(\varepsilon_i)||\varepsilon_i||^2$$

with $w = \frac{h'(\varepsilon)}{\varepsilon}$ and $h'(\varepsilon) = \frac{\partial h(\varepsilon)}{\partial \varepsilon}$.

- Problems:
  1. Weights depend on the errors incurred by the optimal parameters of our model
  2. But the *parameters are unknown* and so are the weights.
- Solution: Can apply an iterative approach

  that will converge as long as $h\left(\sqrt{|\epsilon|}\right)$ is concave[1].

Not covered in 2021/22 lectures – Consider this as additional material

[1]Aftab, K. and Hartley, R., Convergence of Iteratively Re-weighted Least Squares to Robust M-estimators, WACV 2015

R. Hartley, Robust Optimization Techniques in Computer Vision, Session 3,ECCV2014 tutorials

Iterative reweighted least squares

1. Set all the weights to $w_i^{t-1} = 1$.
2. Solve for $\boldsymbol{p}^t$ by the weighted least squares problem.
3. Using the estimated parameters $\boldsymbol{p}^t$ re-calculate per-point projection errors $\boldsymbol{\varepsilon}_i^t$.
4. Using the projection errors re-calculate new weights w_i^t from:

$$w = \frac{h'(\varepsilon)}{\varepsilon} \qquad h'(\varepsilon) = \frac{\partial h(\varepsilon)}{\partial \varepsilon}$$

5. Go back to step 2 and continue until the change in parameters is negligibly small (convergence).

Not covered in 2021/22 lectures – Consider this as additional material

Note: $(\cdot)^t$ indicates a step of iteration in the iterative reweighted least squares.

For an instructive discussion on parameters of the Huber cost function from data, please see:
J. Fox, Robust Regression--Appendix to An R and S-PLUS Companion to Applied Regression, 2002, "1.1 Objective Functions".

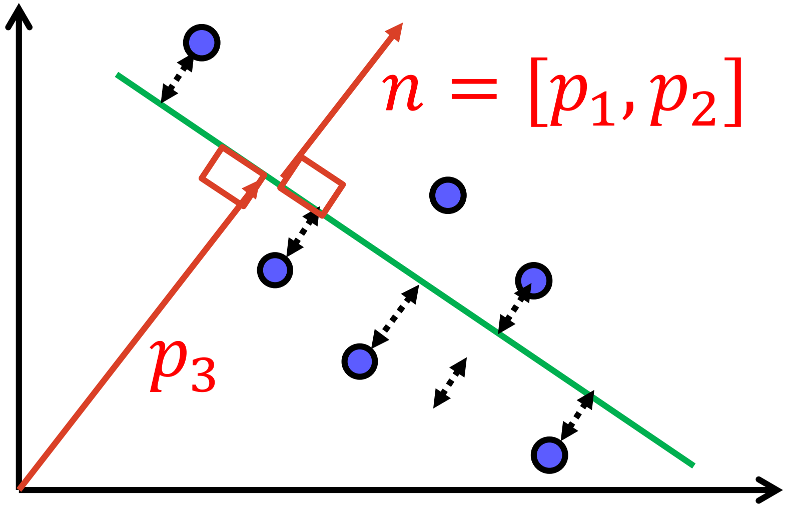
Constrained least squares

- Often we will seek parameters $\boldsymbol{p}$ that satisfy constraints.
- Reconsider line-fitting example, but this time we'll minimize *perpendicular* distances!

$$E(\mathbf{p}) = \sum_{i=1}^{N} ||\varepsilon_i||^2$$

- Re-parameterize:
  $\boldsymbol{p} = [p_1, p_2, p_3]^T$
- Distance of a point to line:
  $||\boldsymbol{\varepsilon_i}||^2 = (x_i p_1 + y_i p_2 - p_3)^2$
- Let's minimize:

$$E(\mathbf{p}) = \sum_{i=1}^{N} ||\varepsilon_i||^2$$



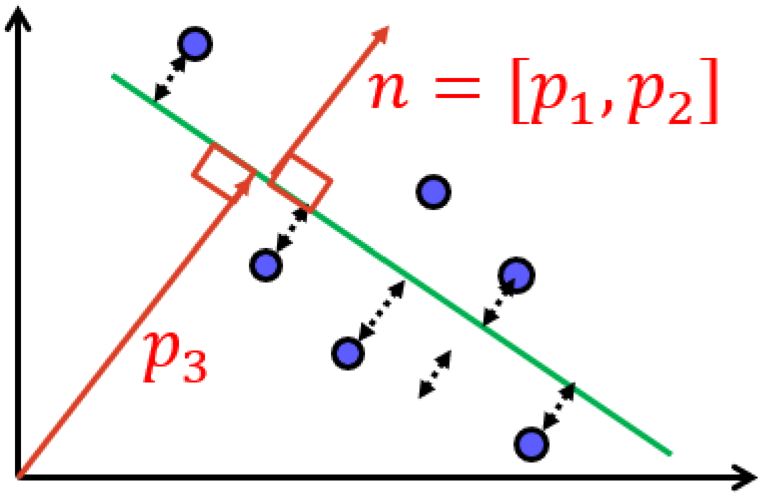
Constrained least squares

- Distance of a point to line:

$$||\boldsymbol{\varepsilon_i}||^2 = (x_i p_1 + y_i p_2 - p_3)^2$$

- Let's minimize:

$$E(\mathbf{p}) = \sum_{i=1}^{N} ||\varepsilon_i||^2$$



Constrained least squares

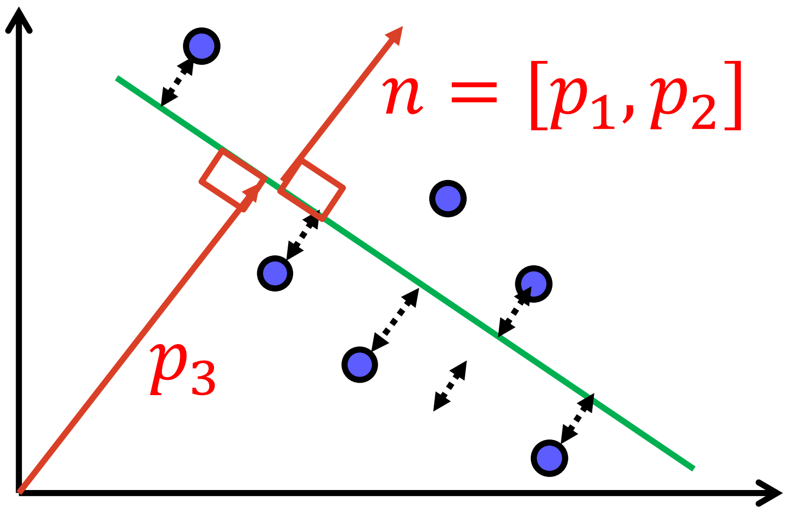
- The solution: $\frac{dE(\mathbf{p})}{d\mathbf{p}} = 2\mathbf{A}^T\mathbf{A}\mathbf{p} \equiv \mathbf{0}$
- Trivial solution: $\boldsymbol{p} = \mathbf{0}$

- A nontrivial solution is obtained by constraint $\left||\boldsymbol{p}|\right|^2 = 1$

$$\boldsymbol{p} = [p_1, p_2, p_3]^T$$

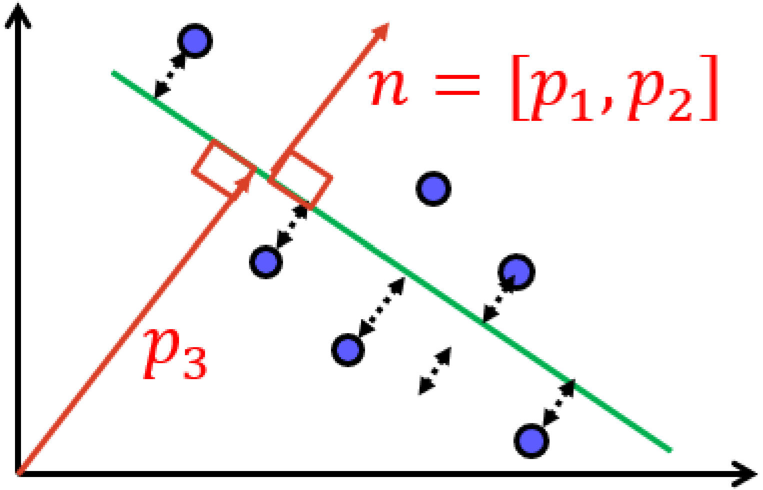
$$||\boldsymbol{\varepsilon_i}||^2 = (x_i p_1 + y_i p_2 - p_3)^2$$

$$E(\mathbf{p}) = \sum_{i=1}^{N} \left||\varepsilon_i|\right|^2$$



Back to line fitting example…

- Distance of a point to line:
  $$||\boldsymbol{\varepsilon_i}||^2 = (x_i p_1 + y_i p_2 - p_3)^2$$
- Let's minimize:
  $$E(\mathbf{p}) = \sum_{i=1}^{N} ||\varepsilon_i||^2$$
  $$\mathbf{A}^T\mathbf{A}\mathbf{p} = \lambda\mathbf{p}$$



Constrained least squares

> In case you are not confident with Lagrange multipliers, see this excellent tutorial!

- The solution: $\frac{dE(\mathbf{p})}{d\mathbf{p}} = 2\mathbf{A}^T\mathbf{A}\mathbf{p} = \mathbf{0}$
- Trivial solution: $\boldsymbol{p} = \mathbf{0}$
- A nontrivial solution is obtained by constraint $||\boldsymbol{p}||^2 = 1$
- Taking the derivative of a Langrangian and setting to 0:

$$\mathbf{A}^T\mathbf{A}\mathbf{p} = \lambda\mathbf{p} \quad \longleftarrow \text{Homogenous equation!}$$

- The solution is the eigenvector of $(\boldsymbol{A}^T\boldsymbol{A})$ corresponding to the smallest eigenvalue.
- Actually, it can be shown that this is also the eigenvector corresponding to the smallest eigenvalue of $\boldsymbol{A}$. (see notes on "*Avoid computing A^TA*")

Recognizing the hammer for your nail!

- Problems that can be written as systems of equations (*normal equations*):
  $\boldsymbol{Ap} = \boldsymbol{b}$
  (if you have weights on equations, then $\mathbf{W}\boldsymbol{Ap} = \mathbf{W}\boldsymbol{b}$)
  can be solved by ordinary LS or IRWLS

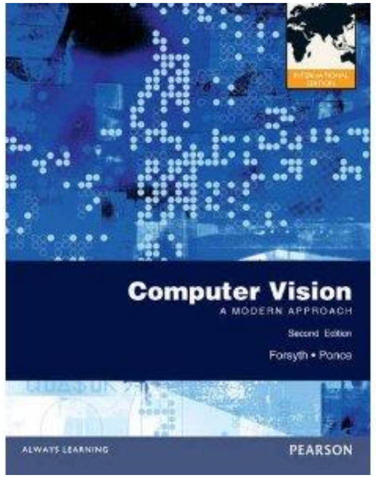
  Matlab: `p = A \ b ;`

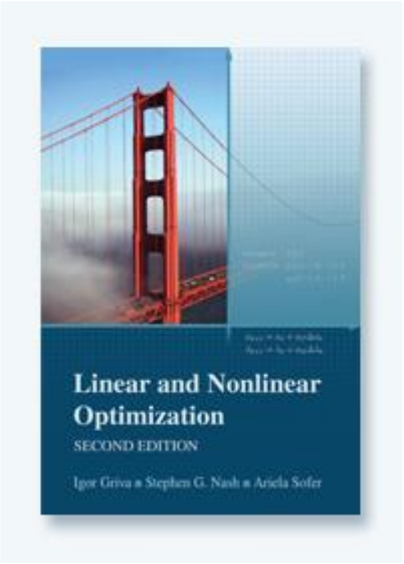
- Problems that result in a homogenous system:
  $\boldsymbol{Ap} = \mathbf{0}$
  can be solved by putting the constraint $||\boldsymbol{p}||^2 = 1$, the solution is the eigenvector corresponding to the smallest eigenvalue.
  (If required, rescale the solution for $\boldsymbol{p}$)

  Matlab: `[U,S,V] =svd(A) ; p = V(:,end) ;`

For nonlinear cost functions

- Often nonlinear error functions are used, which cannot be minimized analytically in a closed form.





- Popular approaches:
  - Gradient descend
  - Newton‘s method
  - Gauss-Newton method
  - Levenberg-Marquardt
  - Alternate direction method of multipliers (ADMM) [!very powerful & simple]
- More about these:
- Fua and Lepetit: Computer Vision Fundamentals: Robust Non-Linear Least-Squares and their Applications
- Griva et al., Linear and Nonlinear Optimization (See appendix on Matrix Algebra)
- The Matrix Coockbook (List of common vector/matrix solutions)
- Forsyth, Ponce, „Computer Vision – A modern approach“, (Appendix in *2nd ed.*)

Need to deal even better with outliers

- Large disagreements in only a few points (outliers) cause failure of the least-squares-based methods.
- The detection, localization and recognition in CV have to operate in significantly noisy data.
- In some cases >½ data is expected to be outliers.
- Standard methods for robust estimation can rarely deal with such a large proportion of outliers.

RANSAC

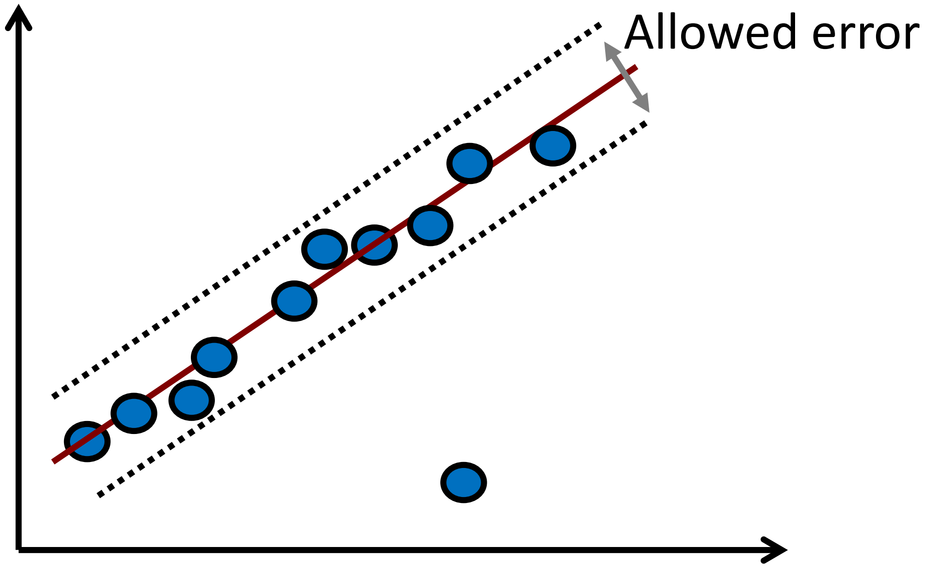
- The RANSAC algorithm (random sample consesus).

- Very popular due to its generality and simplicity.
- Can deal with large portions of outliers.
- Published in 1981 (Fischler in Bolles)
- One of the most cited papers in Computer Vision
- Many improvements proposed since!

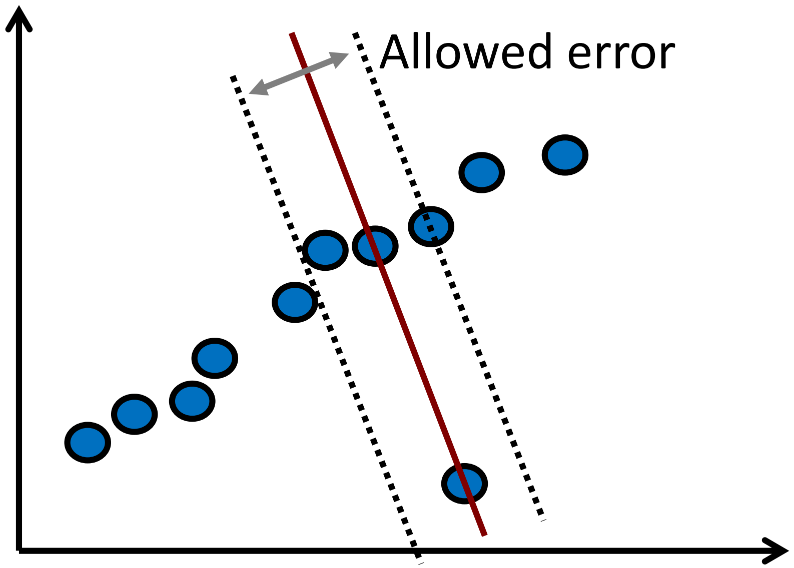
M. A. Fischler, R. C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Comm. of the ACM, Vol 24, pp. 381-395, 1981.

RANSAC: Intuition by line fitting

- A good estimate of our model should have a strong support in data:
  "recognize a good model when you see it"



10 point support this line!

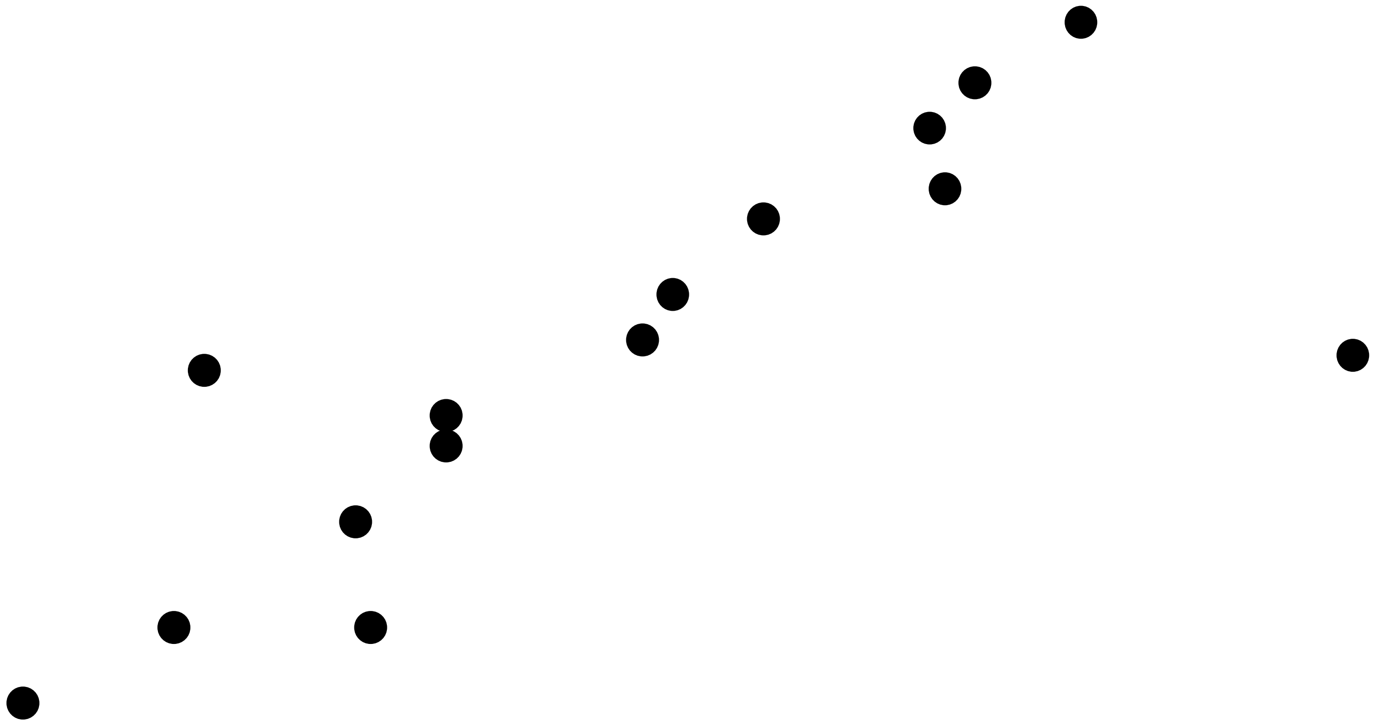


4 point support this line!

- How to find a model with a strong support?
- By randomly sampling potential models.

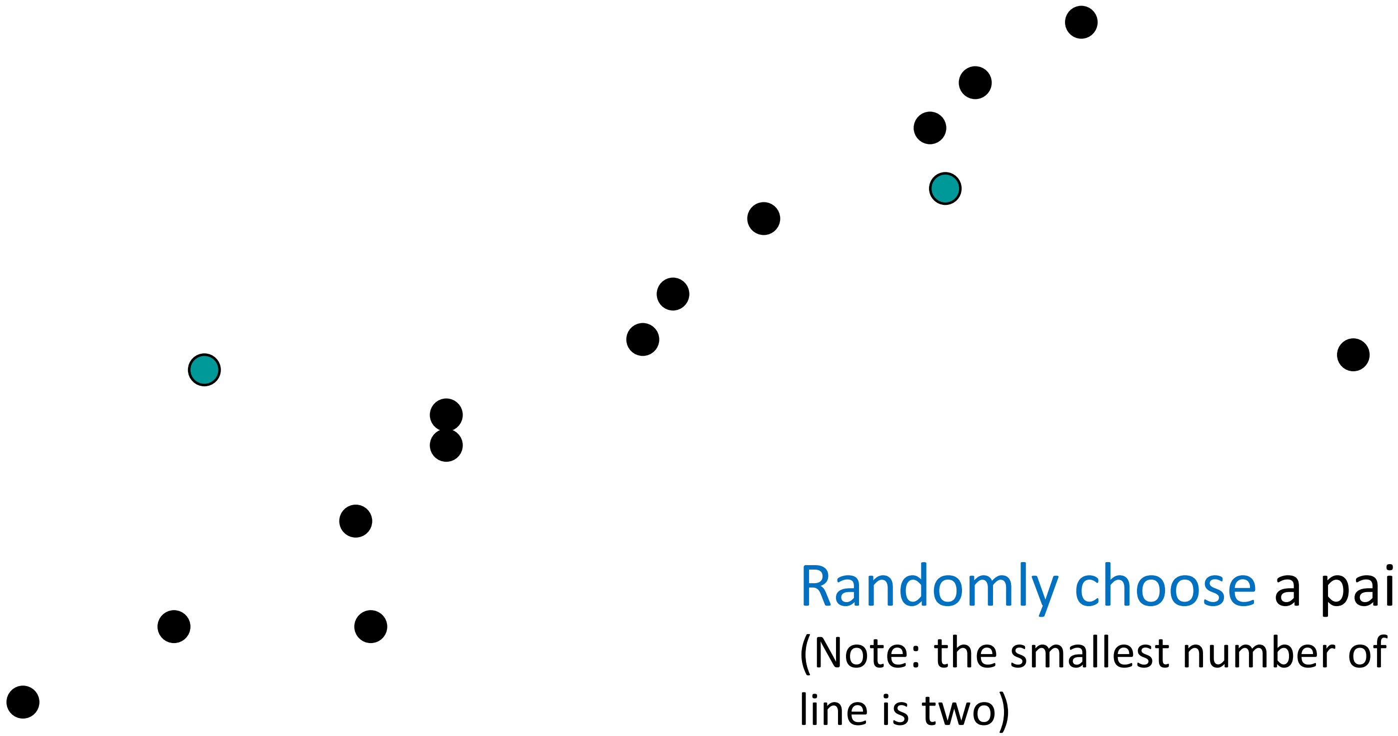
RANSAC: Intuition by line fitting

- Task: Robustly estimate the most likely line



RANSAC: Intuition by line fitting

- Task: Robustly estimate the most likely line

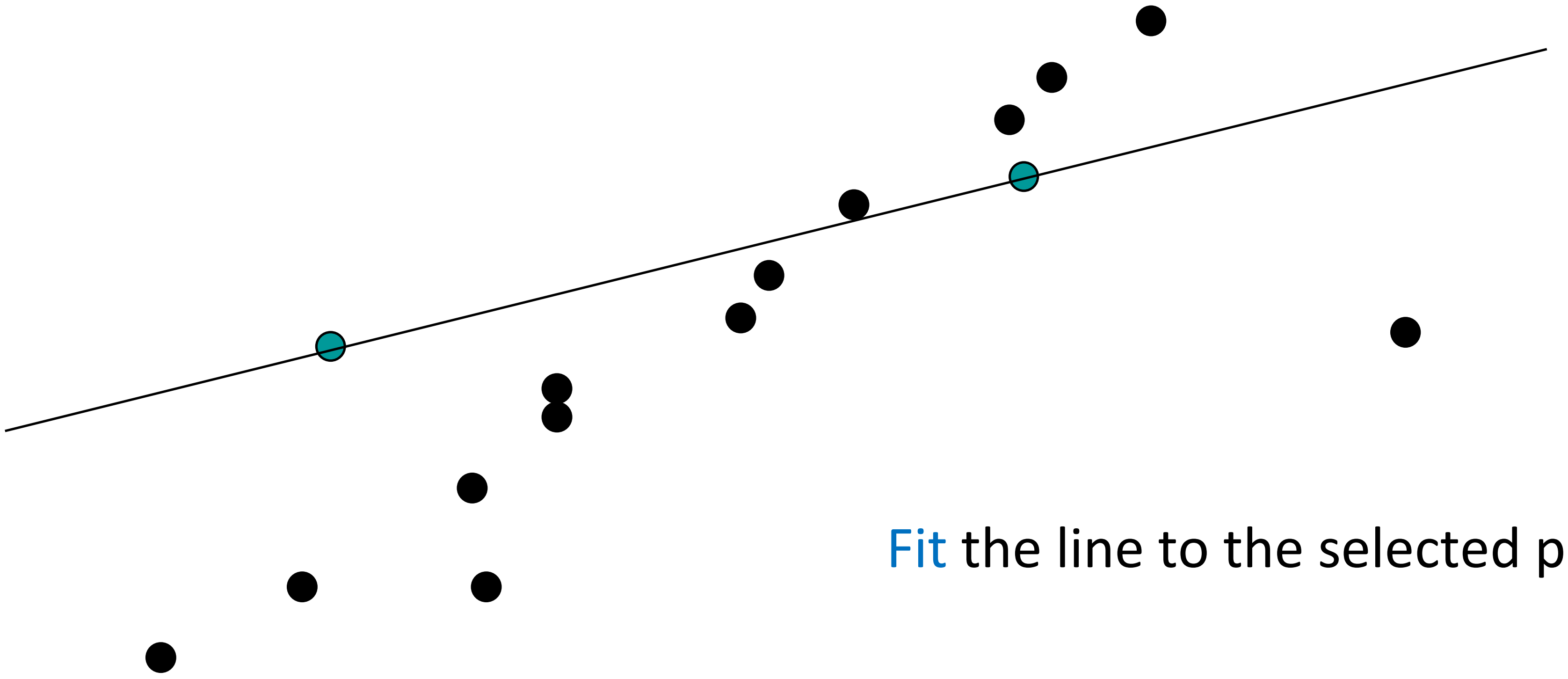


Randomly choose a pair of points

(Note: the smallest number of points to fit a line is two)

RANSAC: Intuition by line fitting

- Task: Robustly estimate the most likely line

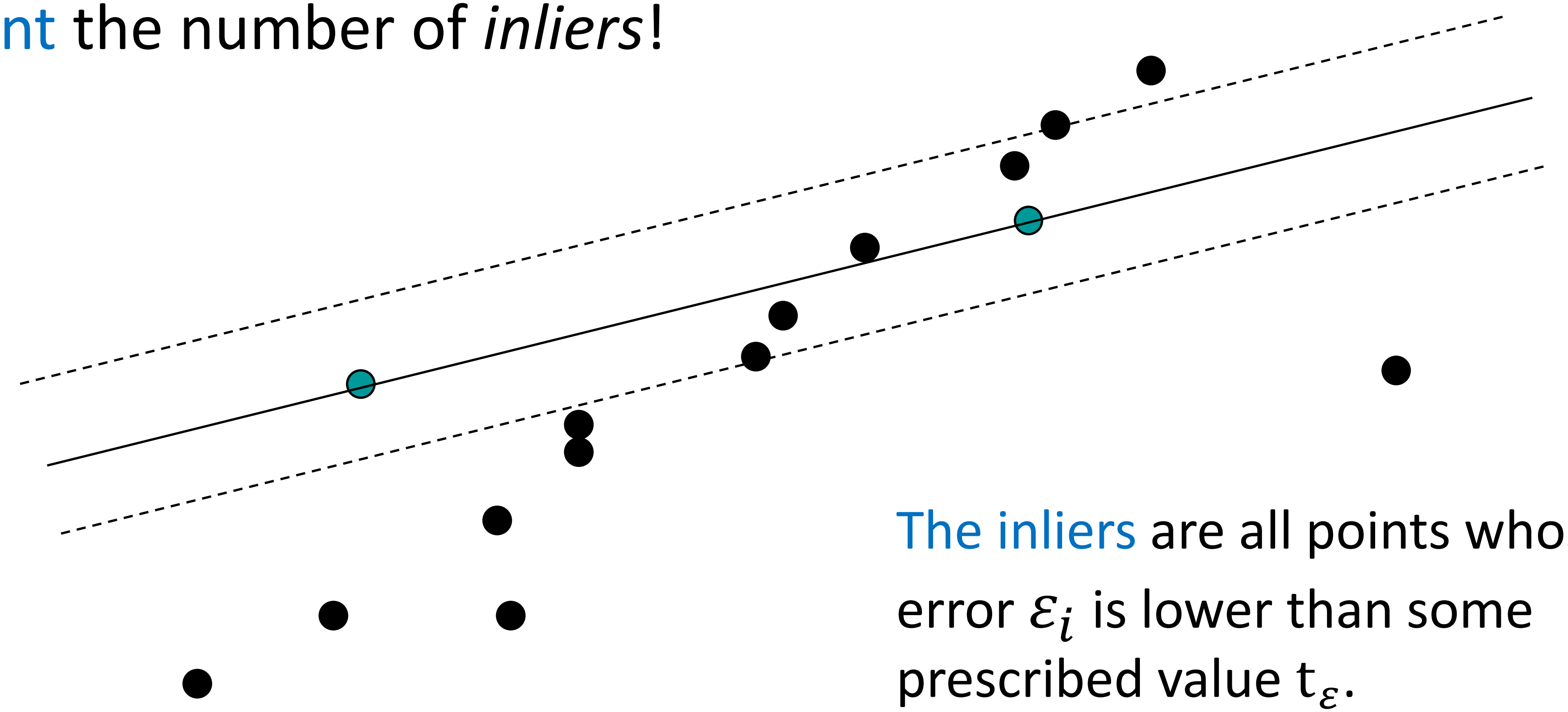


Fit the line to the selected points.

RANSAC: Intuition by line fitting

- Task: Robustly estimate the most likely line

Count the number of *inliers*!

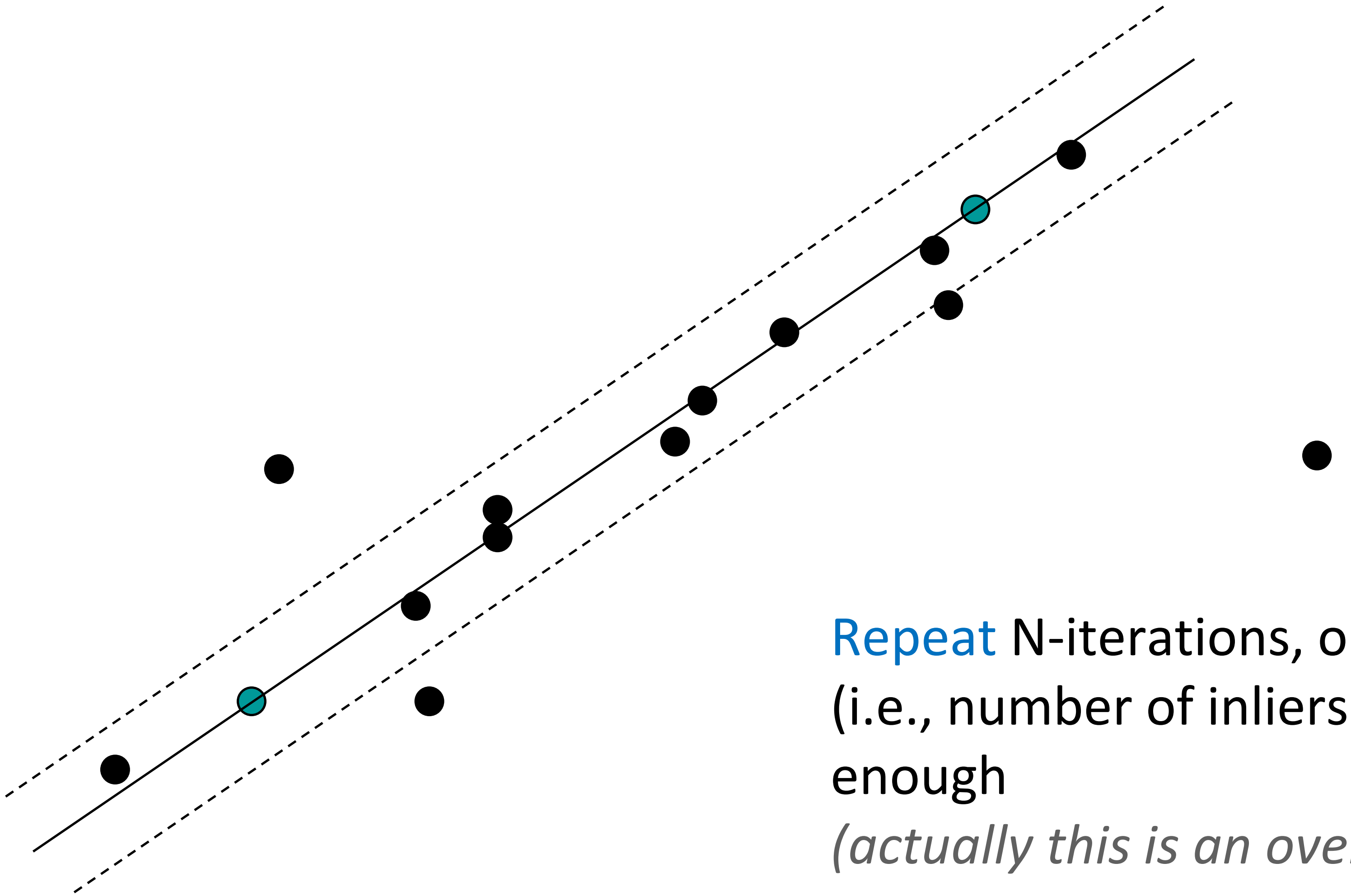


The inliers are all points whose error ε_i is lower than some prescribed value t_ε.

$$\varepsilon_i = |f(x_i; \boldsymbol{p}) - y_i|$$

RANSAC: Intuition by line fitting

- Task: Robustly estimate the most likely line

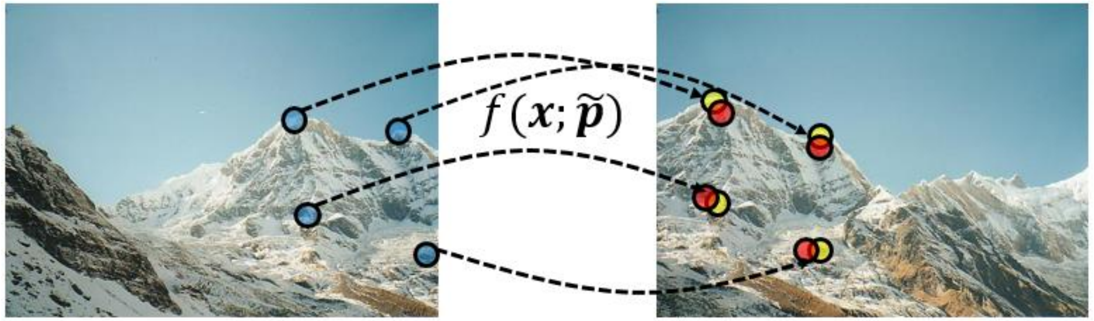


Repeat N-iterations, or, until the support (i.e., number of inliers) becomes strong enough

(actually this is an oversimplification).

Previously at MP…

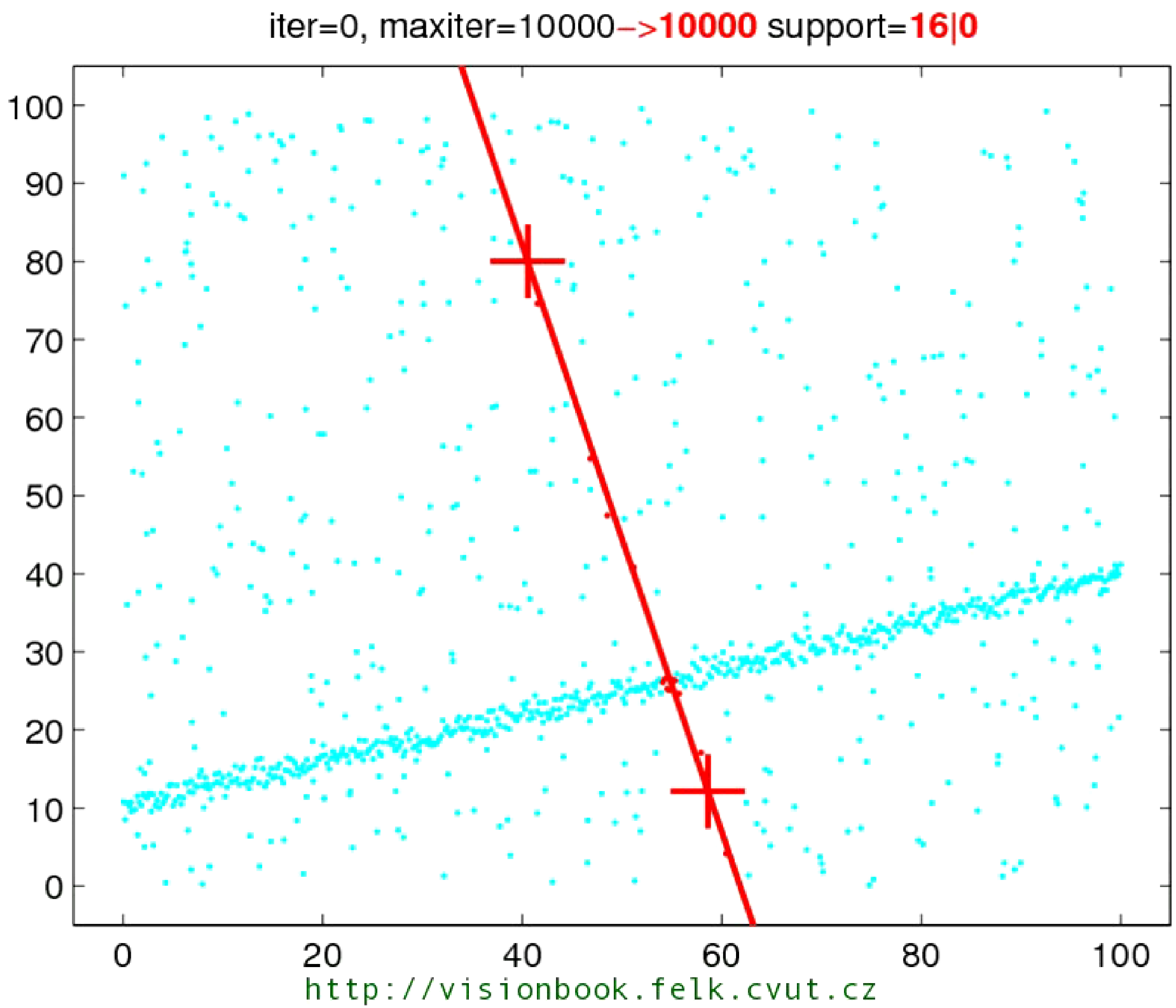
- Least squares parameters estimation



- Ordinary + Weighted (+ Robust) + Constrained Least squares
- Normal equations: $\boldsymbol{Ax} = \boldsymbol{b}$ → pseudoinverse
- Homogeneous system: $\boldsymbol{Ax} = \boldsymbol{0}$ → eigenvectors

RANSAC: line fitting

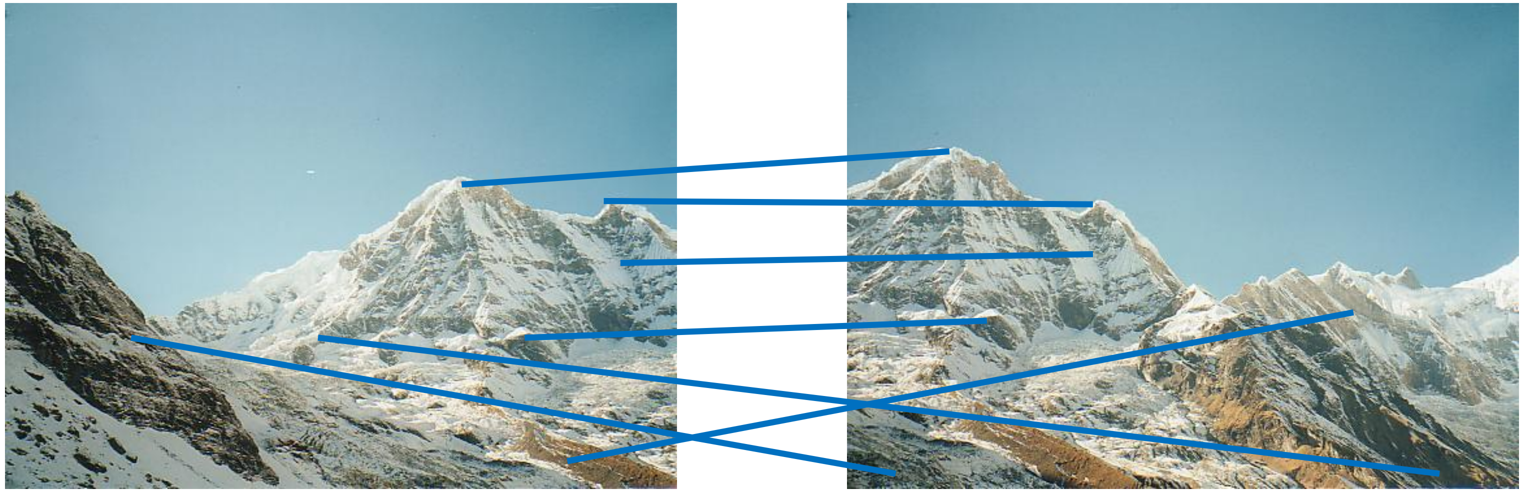
- Another example



A general setting

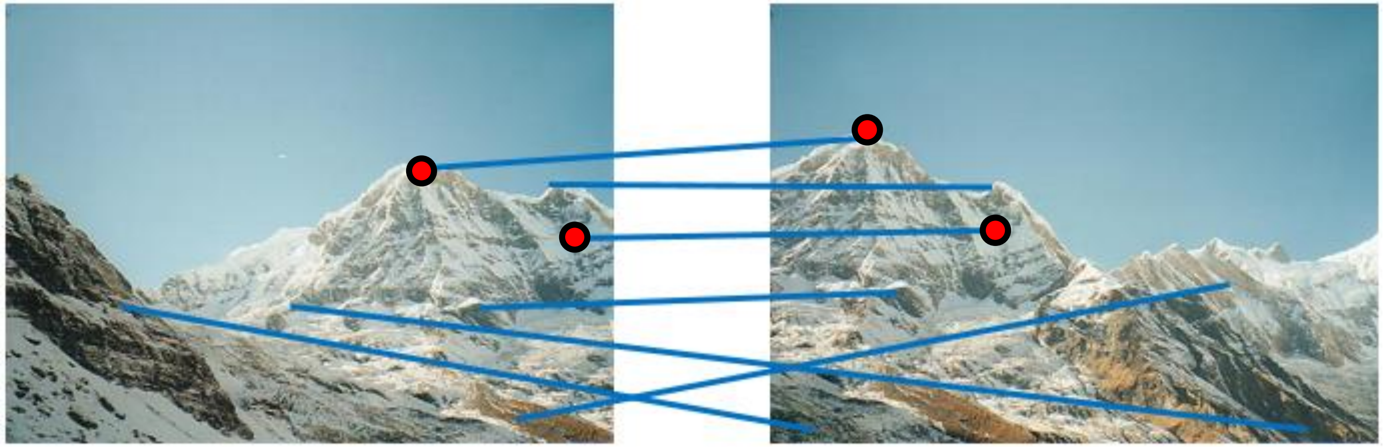
1. Define the set of "potentially" corresponding points:
   $\{\boldsymbol{x}_i\}_{i=1:N}$, $\{\boldsymbol{x}'_i\}_{i=1:N}$
2. Define the transformation model: $f(\boldsymbol{x}; \boldsymbol{p}): \boldsymbol{x} \rightarrow \boldsymbol{x}'$

   In this example, let $f(\boldsymbol{x}; \boldsymbol{p})$ be a simple translation + scaling.



Important: Some correspondences are correct and some are NOT!

A simple RANSAC loop



$\{\boldsymbol{x}_i\}_{i=1:N}$, $\{\boldsymbol{x}'_i\}_{i=1:N}$

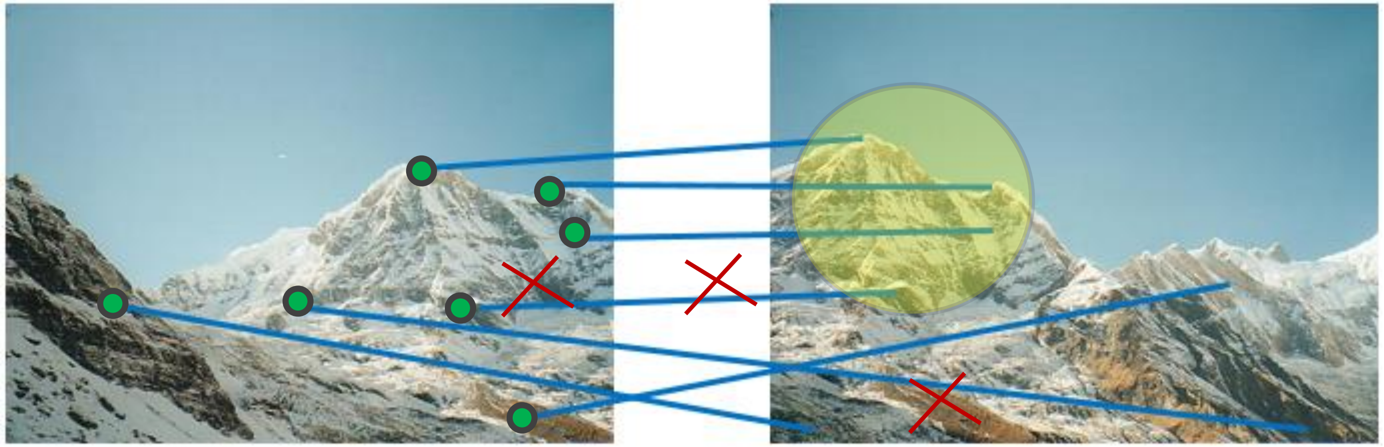
$f(\boldsymbol{x};\boldsymbol{p}): \boldsymbol{x} \rightarrow \boldsymbol{x}'$

In this example, let $f(\boldsymbol{x};\boldsymbol{p})$ be a simple translation + scaling.

1. Randomly select the smallest group of correspondences, from which we can estimate the parameters of our model.

2. Fit the parametric model $\tilde{p}$ to the selected correspondences (e.g., by LS).

A simple RANSAC loop



$\{\boldsymbol{x}_i\}_{i=1:N}$, $\{\boldsymbol{x}'_i\}_{i=1:N}$

$f(\boldsymbol{x};\boldsymbol{p}): \boldsymbol{x} \rightarrow \boldsymbol{x}'$

In this example, let $f(\boldsymbol{x};\boldsymbol{p})$ be a simple translation + scaling.

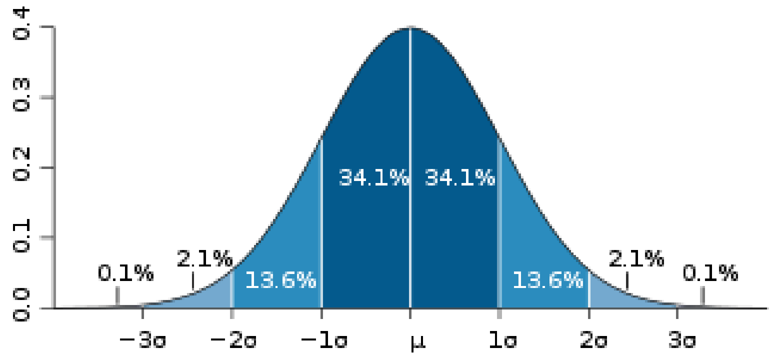
1. Randomly select the smallest group of correspondences, from which we can estimate the parameters of our model.
2. Fit the parametric model $\tilde{p}$ to the selected correspondences (e.g., by LS).
3. Project all other points and count how many of all correspondences are in agreement with the fitted model – number of inliers.

- Remember the model parameters $\tilde{p}_{opt}$ that maximize the number of inliers.

The choice of parameters

- How many correspondences $''s''$ are required?
  - Typically the smallest number that allows estimating the model parameters, i.e., as many as the model parameters.
- Threshold distance t for identifying the inliers
  - Choose t, such, that the probability that an inlier falls below the threshold is equal to p_w. For example (p_w=0.95)
  - Assuming a Gaussian noise on the measurements.

    The noise standard dev. σ: t=2σ
- Number of sampling iterations N
  - Chose N such, that the probability p of drawing a sample with all inliers at least once is high enough.



The choice of parameters: N

- Setting the number of sampling iterations *N:*
  - Assume we know the proportion *e* of outliers (probability of selecting an outlier at random).
  - Choose *N* such, that the probability of drawing a sample set with all inliers at least once in N draws is *p*,(e.g., *p*=0.99).
  - Derive the probability of drawing a bad sample in N trials, $1 - \mathrm{p} = {p_{bad}}^{N}$, and expose N
    - Probability of choosing a single inlier: 1 - *e*
    - Probability of an all-inlier sample:
      → *s-times sample an inlier*: $(1 - e)^s$
    - Probability, of a bad sample:
      → at least one of *s not an inlier*: $[1 - (1 - e)^s]$
    - Probability of always drawing a bad sample in N trials: $(1 - (1 - e)^s)^N$

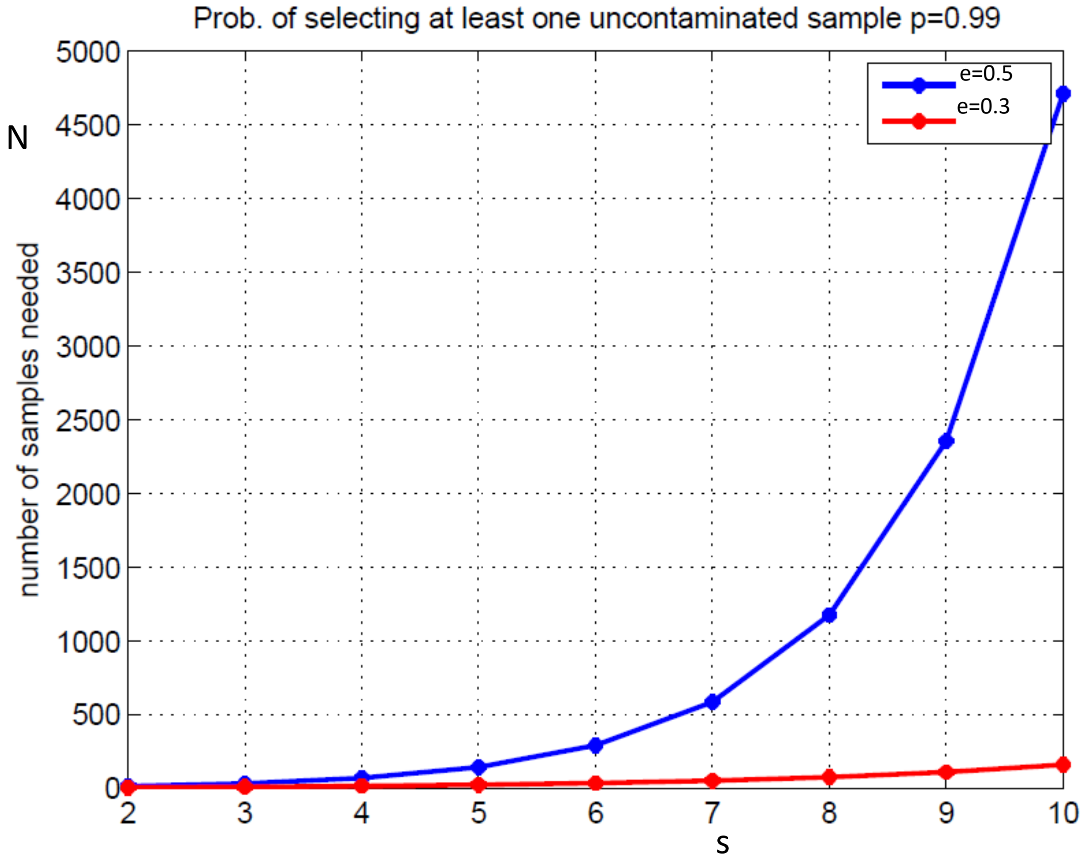
$$1 - p = (1 - (1 - e)^s)^N \quad \Rightarrow \quad N = \frac{\log(1-p)}{\log(1-(1-e)^s)}$$

The choice of parameters: N

Number of iterations N required to sample an inlying model with s parameters at least once with probability p if the proportion of outliers is e:

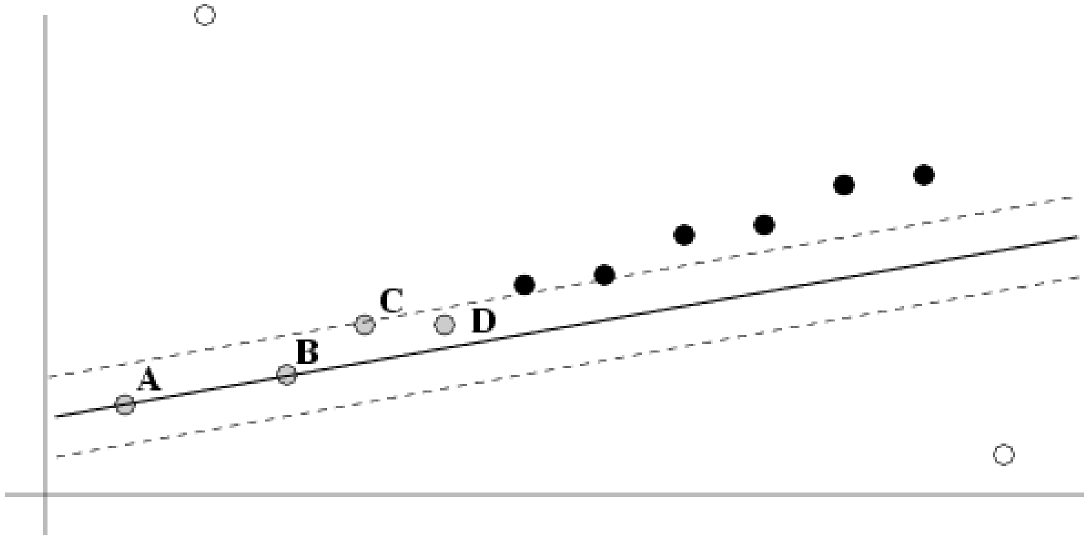
	portion of outliers: e						
s	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

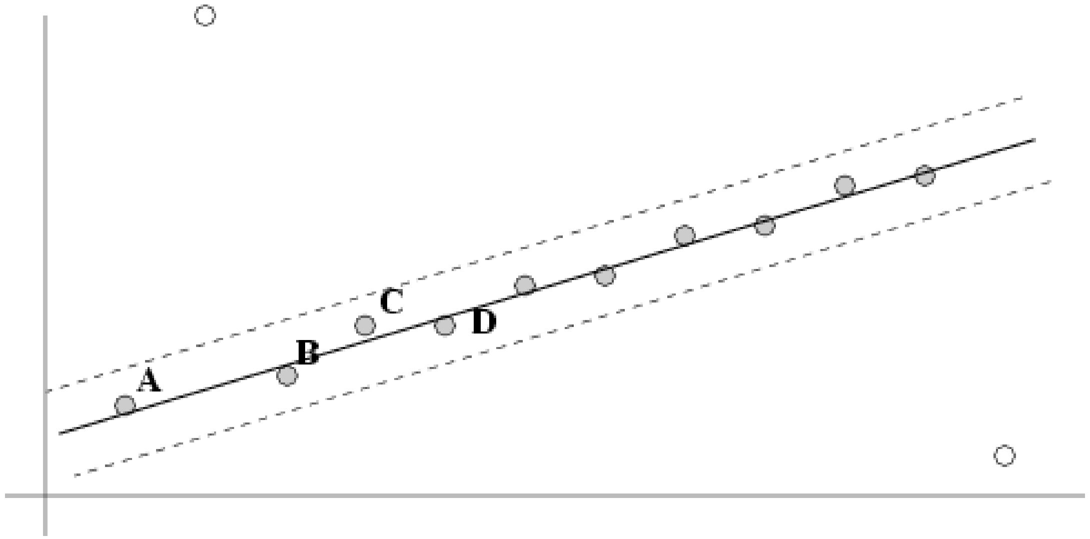
Tabulated values of N for $p = 0.99$



After RANSAC: Refit by LS

- RANSAC splits the data into inliers and outliers, and calculates the model parameters using a minimal number of correspondences.
- Improve the model parameters by applying least squares to the inliers.





Beyond the simple RANSAC

- A great deal of research was invested by many researchers into improving RANSAC
  - Finding the right solution faster & with better resiliency to outliers

- Further reading:
  - PROSAC (state of the art, better chooses the order of samples)
  - MAGSAC++ (current state of the art – top performance on benchmarks)
  - Excellent tutorial in recent RANSAC developments and toolboxes: RANSAC in 2020: A CVPR Tutorial, CVPR 2020 (Video presentations available!)

RANSAC: Summary

- Pros
  - Very simple and general
  - Applicable to many real-life problems
  - Often used in practice
- Cons
  - Requires setting some parameters (modern methods make it simpler)
  - Potentially many iterations required to find the optimum.
  - Fails at very small number of inliers.
  - In some cases more accurate procedures, that do not require brute-force sampling, can be found.

Fitting: Challenges

- If we know the inliers how to estimate the parameters?
  - Least squares
- What if our data includes outliers?
  - Robust least squares, RANSAC
- What if we have multiple instances of our model (e.g., multiple lines)?
  - Apply voting: sequential RANSAC, Hough transform
- What if we have multiple models (e.g., unknown degree of a polynomial)?
  - Apply model selection (e.g., MDL, BIC, AIC)
- Complicated nonparametric models
  - Generalized Hough (GHT)
  - Iterative Closest Point, (ICP) == iterative local least squares

Further reading

- A simple and interesting way to iteratively fit a complicated model to data:
  Iterative Closest Point method
  Matlab implementation: ICP

- A very nice and accessible tutorial on nonlinear optimization in computer vision: http://cvlabwww.epfl.ch/~fua/courses/lsq/Intro.htm

References

- R. Szeliski, Computer Vision: Algorithms and Applications, 2010
- David A. Forsyth, Jean Ponce, Computer Vision: A Modern Approach (2nd Edition), (*second edition!*)
    - See appendix on Normal equations and Homogeneous systems
- Igor Griva, Stephen G. Nash, Ariela Sofer , Linear and Nonlinear Optimization
    - See appendix on Matrix Algebra
- The Matrix Cookbook
    - List of common vector/matrix solutions