Technischen Universität München
Winter Semester 2018/2019

TRACKING and DETECTION in COMPUTER VISION

Non-linear optimization and robust estimation for tracking

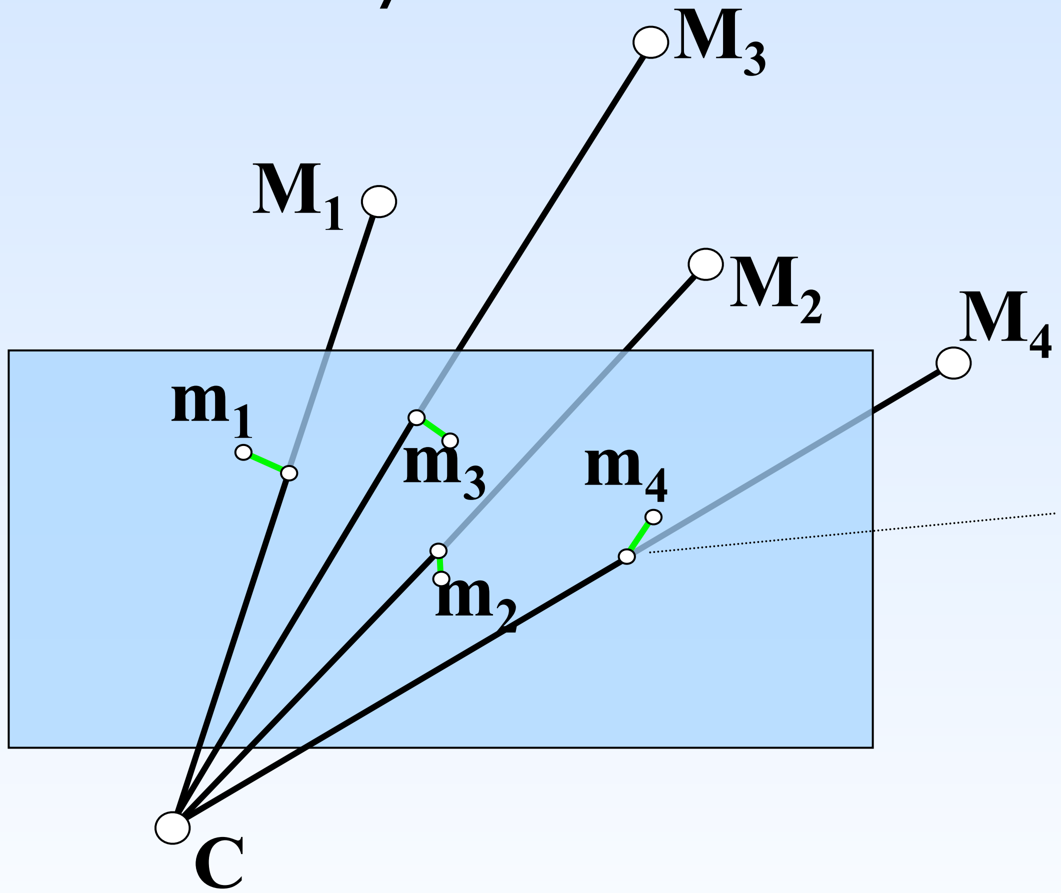
Slobodan Ilić

Slides adapted from V.Lepetit

Minimization of the Reprojection Error

$$\min_{\mathbf{R},\mathbf{T}} \sum_i \left\| \mathrm{Proj}_{\mathbf{A},\mathbf{R},\mathbf{T}}(\mathbf{M}_i) - \mathbf{m}_i \right\|^2$$

- Minimization of a physical, meaningful error (reprojection error, in pixels);
- No restriction on the number of correspondences;
- Can be very accurate.

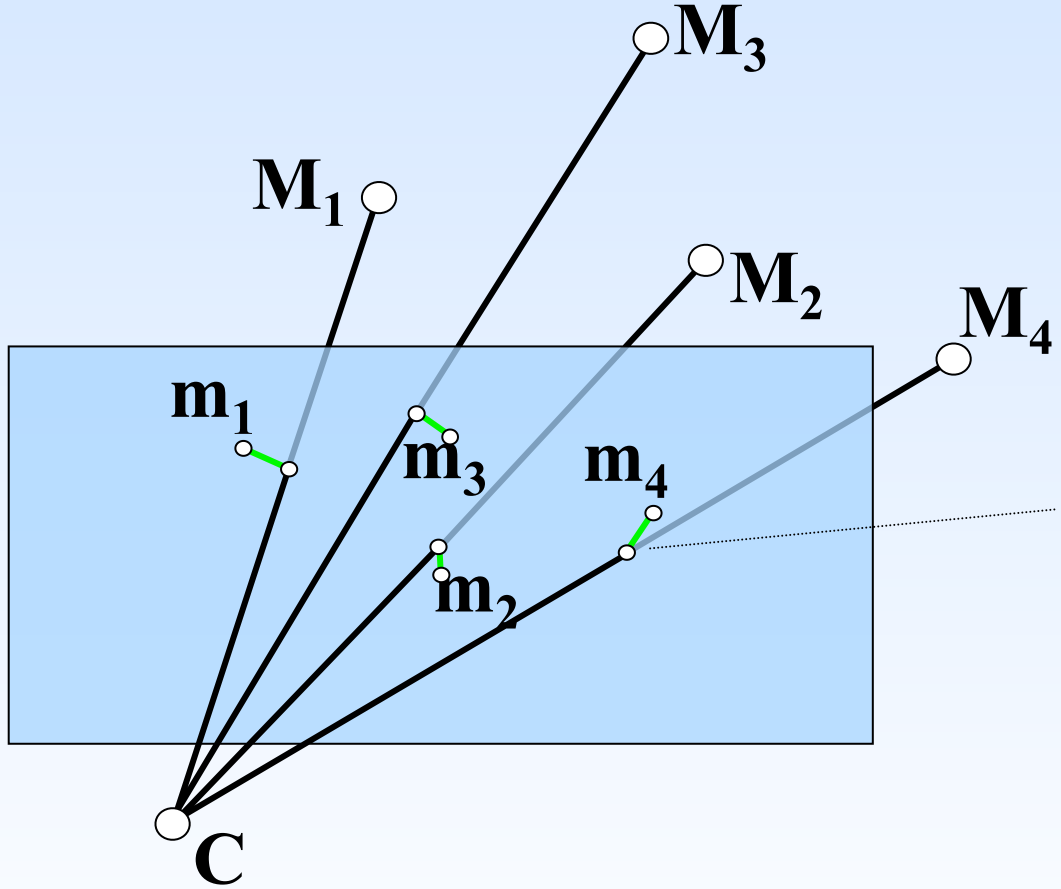


$$\mathrm{Proj}_{\mathbf{A},\mathbf{R},\mathbf{T}}(\mathbf{M}_4) \cong \mathbf{A}[\mathbf{R}|\mathbf{T}]\mathbf{M}_4$$

Minimization of the Reprojection Error

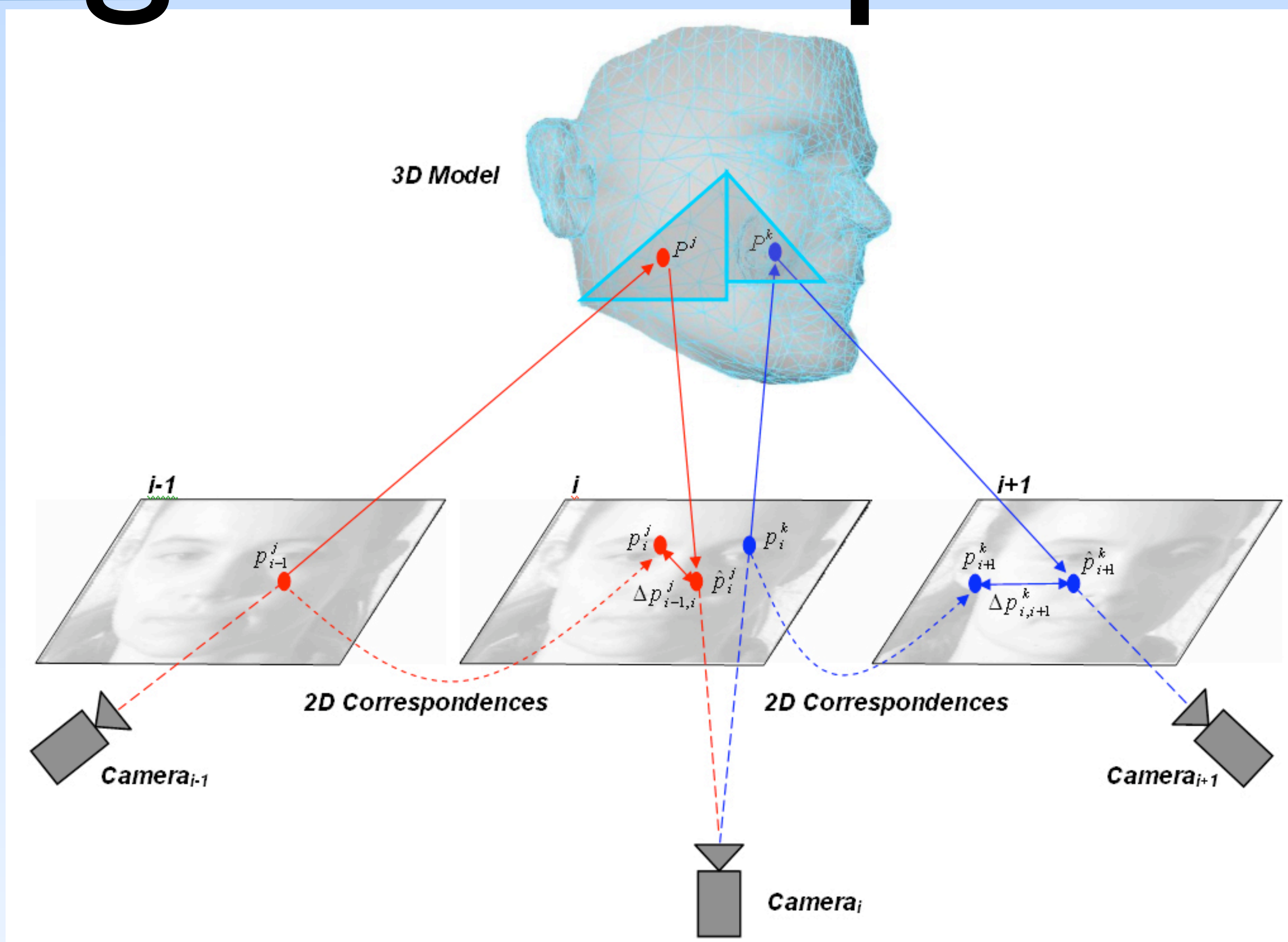
$$\min_{\mathbf{R},\mathbf{T}} \sum_i \left\| \mathrm{Proj}_{\mathbf{A},\mathbf{R},\mathbf{T}}(\mathbf{M}_i) - \mathbf{m}_i \right\|^2$$

- Non-linear least-squares minimization;
- Requires an iterative numerical optimization ➔ Requires an initialization.



$$\mathrm{Proj}_{\mathbf{A},\mathbf{R},\mathbf{T}}(\mathbf{M}_4) \cong \mathbf{A}[\mathbf{R}|\mathbf{T}]\mathbf{M}_4$$

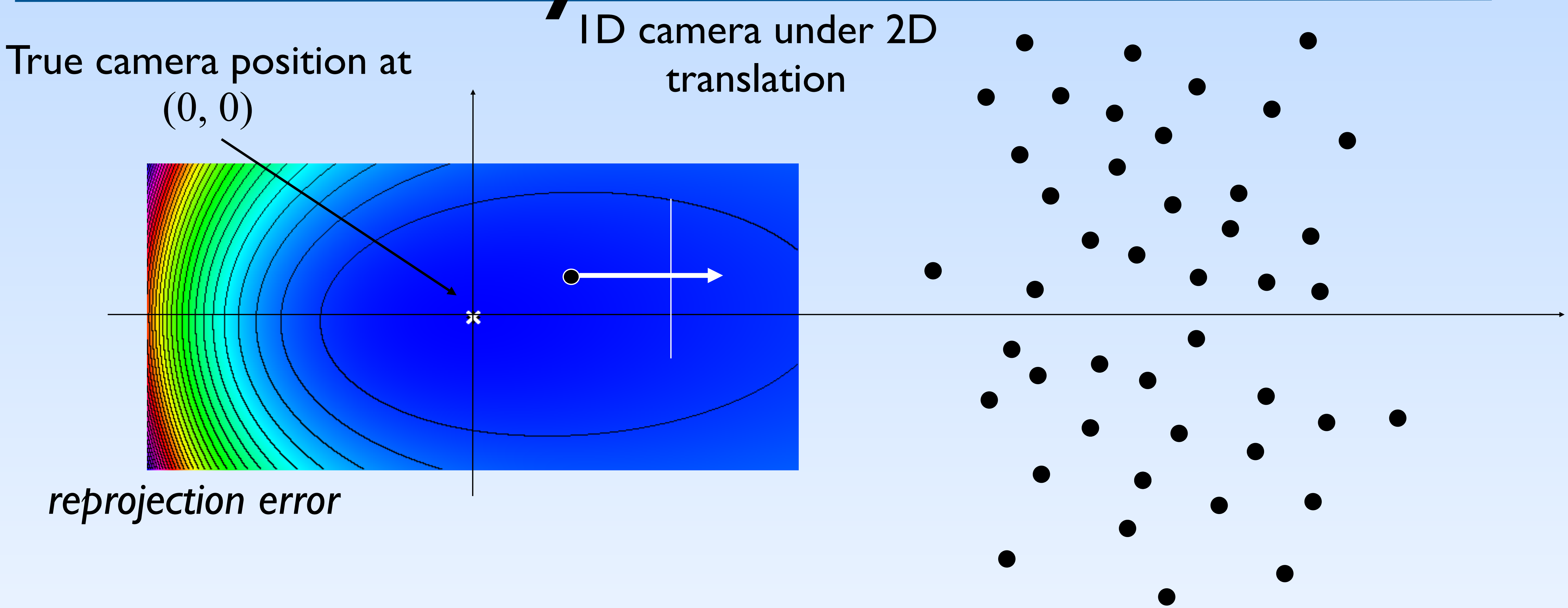
Objective function handling image correspondences



$$\min_{R,T} = \sum_{k=1}^{N} w_i^k \|\hat{\mathbf{p}}_{i+1}^k - \mathbf{p}_{i+1}^k\| = \sum_{k=1}^{N} w_i^k \|\psi(\mathbf{p}_i^k, \mathbf{\Theta}) - \mathbf{p}_{i+1}^k\|$$

$\psi(\mathbf{p}_i^k, \mathbf{\Theta})$ - transfer function of a back projection

Toy Problem

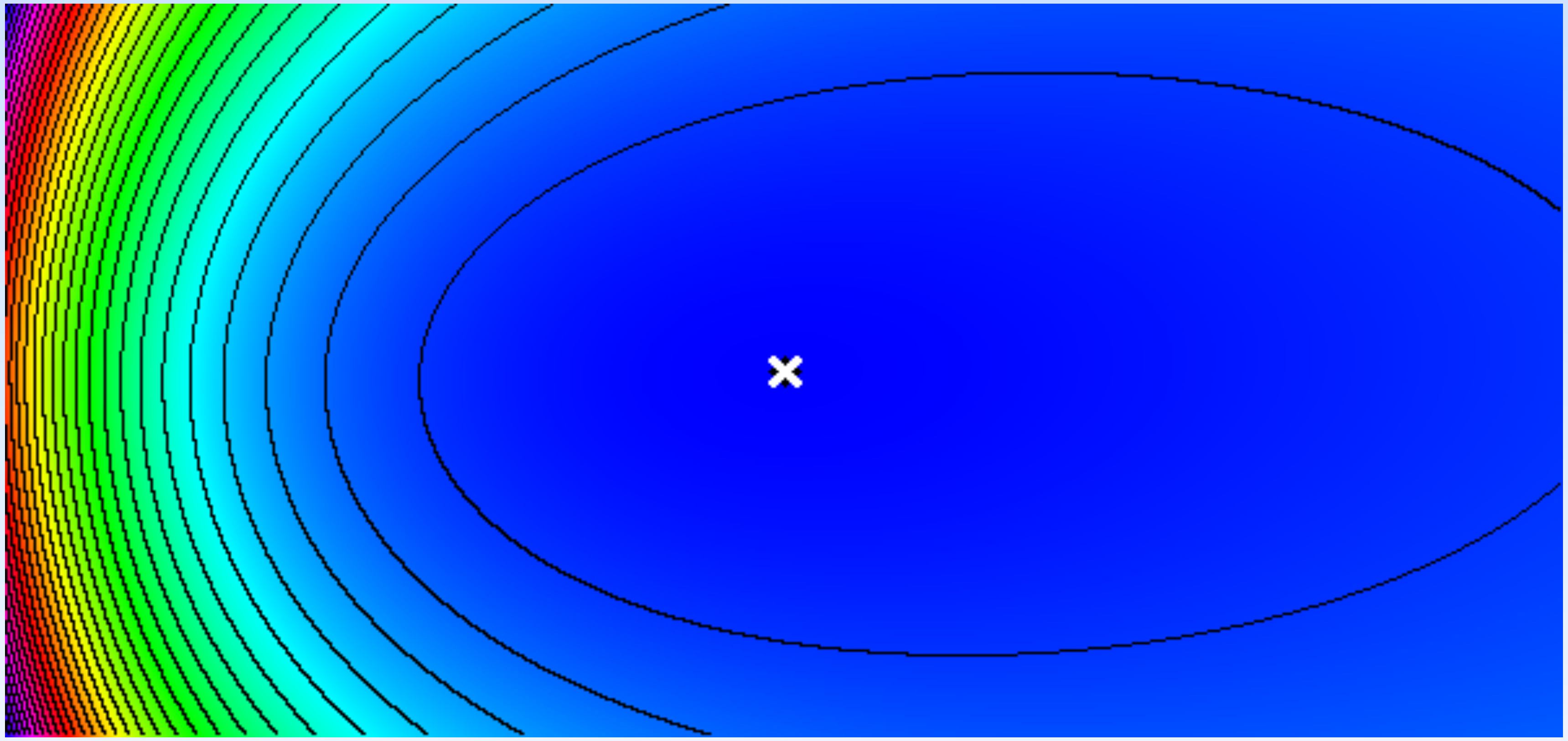


100 "3D points" taken at randomly in [400;1000]x[-500;+500]

Gaussian Noise on the Projections

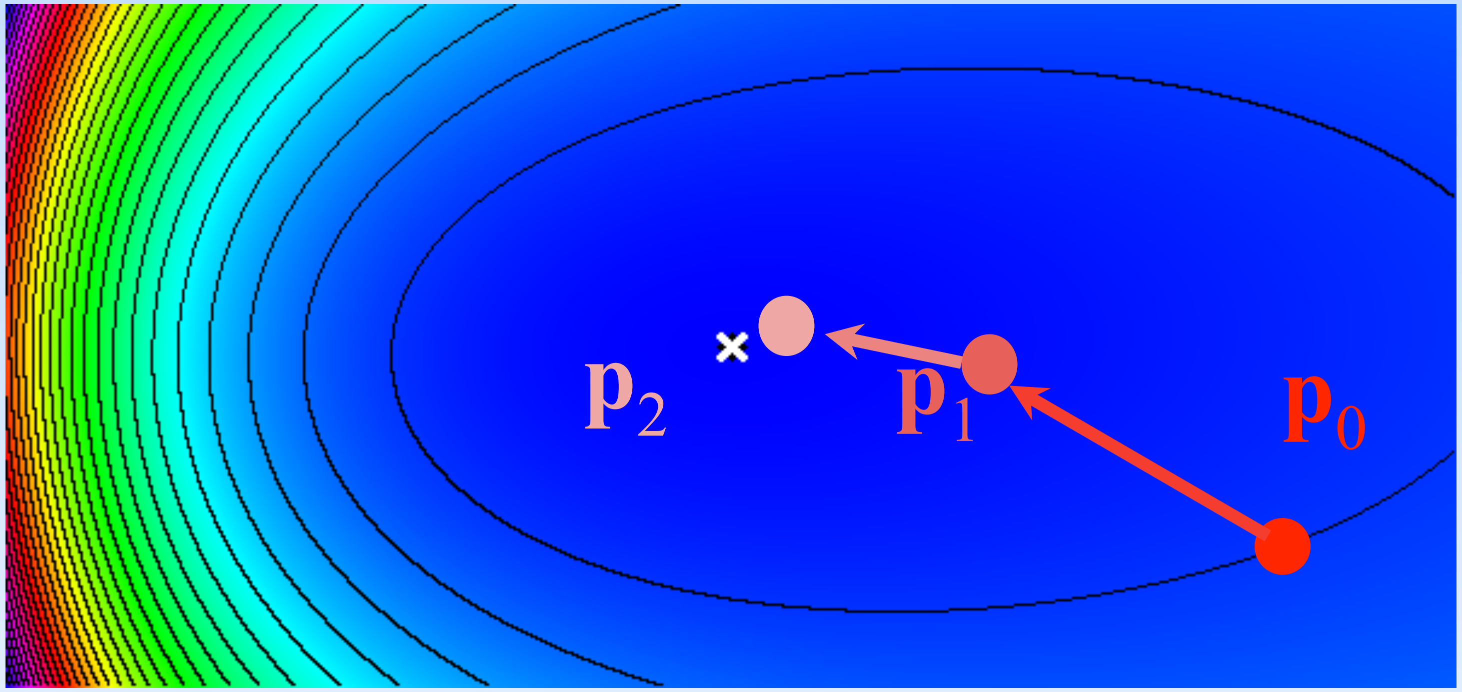
White cross: true camera position;

Black cross: global minimum of the objective function.



In case of Gaussian noise on projections, the global minimum of the objective function is very close(almost identical) to the true camera pose.

Numerical Optimization



Start from an initial guess $\mathbf{p}_0$:

$\mathbf{p}_0$ can be taken randomly but should be as close as possible to the global minimum:

- pose computed at time t-1;
- pose predicted from pose computed at time t-1 and a motion model;
- ...

Numerical Optimization

General methods:

- Gradient descent / Steepest Descent;
- Conjugate Gradient;
- ...

Non-linear Least-squares optimization:

- Gauss-Newton;
- Levenberg-Marquardt;
- ...

Numerical Optimization

We want to find $\mathbf{p}$ that minimizes:

$$
\begin{aligned}
E(\mathbf{p}) &= \sum_i \left\| \mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_i) - \mathbf{m}_i \right\|^2 \\
&= \left\| f(\mathbf{p}) - \mathbf{b} \right\|^2
\end{aligned}
$$

where

$$
f(\mathbf{p}) = \begin{bmatrix} u\left(\mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_1)\right) \\ v\left(\mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_1)\right) \\ \vdots \end{bmatrix} \quad b = \begin{bmatrix} u(\mathbf{m}_1) \\ v(\mathbf{m}_1) \\ \vdots \end{bmatrix}
$$

- $\mathbf{p}$ is a vector of parameters that define the camera pose (translation vector + parameters of the rotation matrix);
- $\mathbf{b}$ is a vector made of the measurements (here the $\mathbf{m}_i$);
- f is the function that relates the camera pose to these measurements.

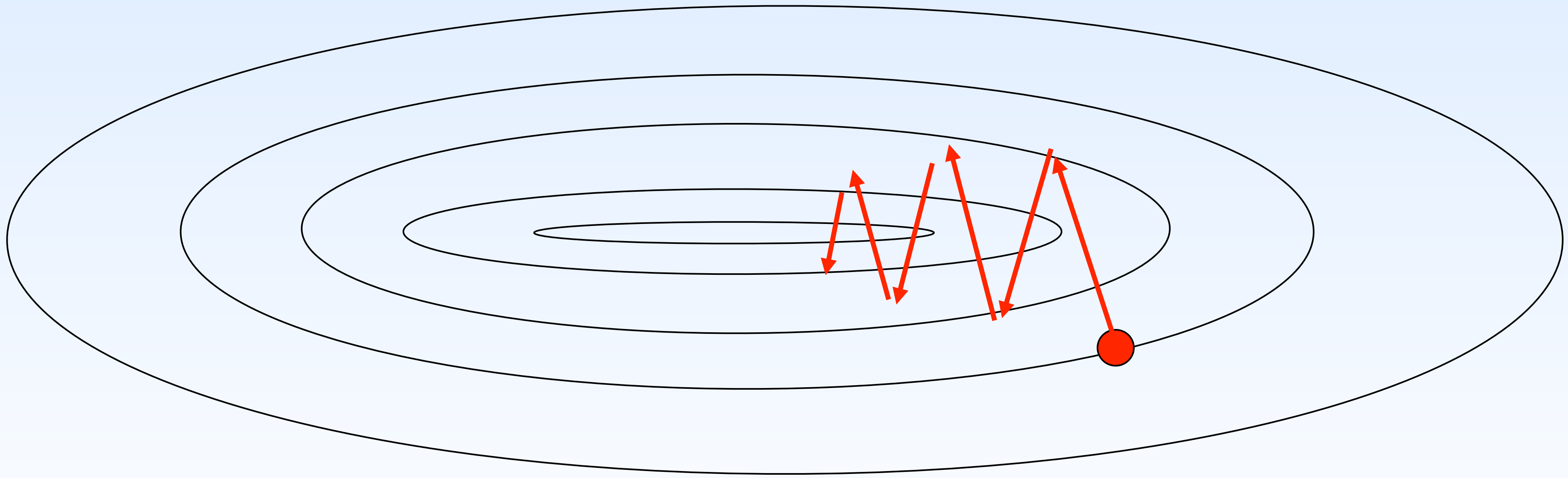
Gradient descent / Steepest Descent

$$\mathbf{p}_{i+1} = \mathbf{p}_i - \lambda \nabla E(\mathbf{p}_i)$$

$$E(\mathbf{p}_i) = \left\| f(\mathbf{p}_i) - \mathbf{b} \right\|^2 = \left(f(\mathbf{p}_i) - \mathbf{b} \right)^{\mathsf{T}} \left(f(\mathbf{p}_i) - \mathbf{b} \right)$$

$\rightarrow \nabla E(\mathbf{p}_i) = 2\mathbf{J}\left(f(\mathbf{p}_i) - \mathbf{b} \right)$ with $\mathbf{J}$ the Jacobian matrix of f, computed at $\mathbf{p}_i$

Weaknesses:
- How to choose λ ?
- Needs a lot of iterations in long and narrow valleys:



The Gauss-Newton and the Levenberg-Marquardt alg.

$$E(\mathbf{p}) = \left\| f(\mathbf{p}) - \mathbf{b} \right\|^2$$

If the function f is linear ie $f(\mathbf{p}) = \mathbf{Ap}$, $\mathbf{p}$ can be estimated as:

$$\mathbf{p} = \mathbf{A}^{+}\mathbf{b}$$

where $\mathbf{A}^{+}$ is the pseudo-inverse of $\mathbf{A}$: $\mathbf{A}^{+} = (\mathbf{A}^{T}\mathbf{A})^{-1}\mathbf{A}^{T}$

Non-Linear Least-Squares: The Gauss-Newton

Iteration steps:

$$\mathbf{p}_{i+1} = \mathbf{p}_i + \mathbf{\Delta}_i$$

$\mathbf{\Delta}_i$ is chosen to minimize the residual $\| f(\mathbf{p}_{i+1}) - \mathbf{b} \|^2$. It is computed by approximating f to the first order:

$$\begin{aligned}
\mathbf{\Delta}_i &= \arg\min_{\mathbf{\Delta}} \left\| f(\mathbf{p}_i + \mathbf{\Delta}) - \mathbf{b} \right\|^2 \\
&= \arg\min_{\mathbf{\Delta}} \left\| f(\mathbf{p}_i) + \mathbf{J\Delta} - \mathbf{b} \right\|^2 && \text{First order approximation: } f(\mathbf{p}_i + \mathbf{\Delta}) \approx f(\mathbf{p}_i) + \mathbf{J\Delta} \\
&= \arg\min_{\mathbf{\Delta}} \left\| \varepsilon_i + \mathbf{J\Delta} \right\|^2 && \varepsilon_i = f(\mathbf{p}_i) - \mathbf{b} \text{ denotes the residual at iteration } i
\end{aligned}$$

$\mathbf{\Delta}_i$ is the solution of the system $\mathbf{J\Delta} = -\varepsilon_i$ in the least-squares sense:

$\mathbf{\Delta}_i = -\mathbf{J}^+ \varepsilon_i$ where $\mathbf{J}^+$ is the pseudo-inverse of $\mathbf{J}$

Non-Linear Least-Squares: The Levenberg-Marquardt Alg.

In the Gauss-Newton algorithm:

$$\mathbf{\Delta}_i = -\left(\mathbf{J}^{\mathrm{T}}\mathbf{J}\right)^{-1}\mathbf{J}^{\mathrm{T}}\varepsilon_i$$

In the Levenberg-Marquardt algorithm:

$$\mathbf{\Delta}_i = -\left(\mathbf{J}^{\mathrm{T}}\mathbf{J} + \lambda\mathbf{I}\right)^{-1}\mathbf{J}^{\mathrm{T}}\varepsilon_i$$

Levenberg-Marquardt Algorithm:

0. Initialize λ with a small value: λ = 0.001

1. Compute $\mathbf{\Delta}_i$ and $E(\mathbf{p}_i + \mathbf{\Delta}_i)$

2. If $E(\mathbf{p}_i + \mathbf{\Delta}_i) > E(\mathbf{p}_i)$: λ ← 10 λ and go back to 1 [*happens when the linear approximation of f is too rough*]

3. If $E(\mathbf{p}_i + \mathbf{\Delta}_i) < E(\mathbf{p}_i)$: λ ← λ / 10, $\mathbf{p}_{i+1} \leftarrow \mathbf{p}_i + \mathbf{\Delta}_i$ and go back to 1.

Non-Linear Least-Squares: The Levenberg-Marquardt Alg.

$$\boldsymbol{\Delta}_i = -\left(\mathbf{J}^{\mathrm{T}}\mathbf{J} + \lambda\mathbf{I}\right)^{-1}\mathbf{J}^{\mathrm{T}}\varepsilon_i$$

- When λ is small, LM behaves similarly to the Gauss-Newton algorithm.
- When λ becomes large, LM behaves similarly to a steepest descent to guarantee convergence.

Possible Parameterizations of the Rotation Matrix

Rotation in 3D space has only 3 degrees of freedom.

It would be awkward to use the nine elements as its parameters.

Possible parameterizations:

- Euler Angles;
- Quaternions;
- Exponential Map.

All have singularities, can be avoided by locally reparameterizing the rotation.

Exponential map has the best properties.

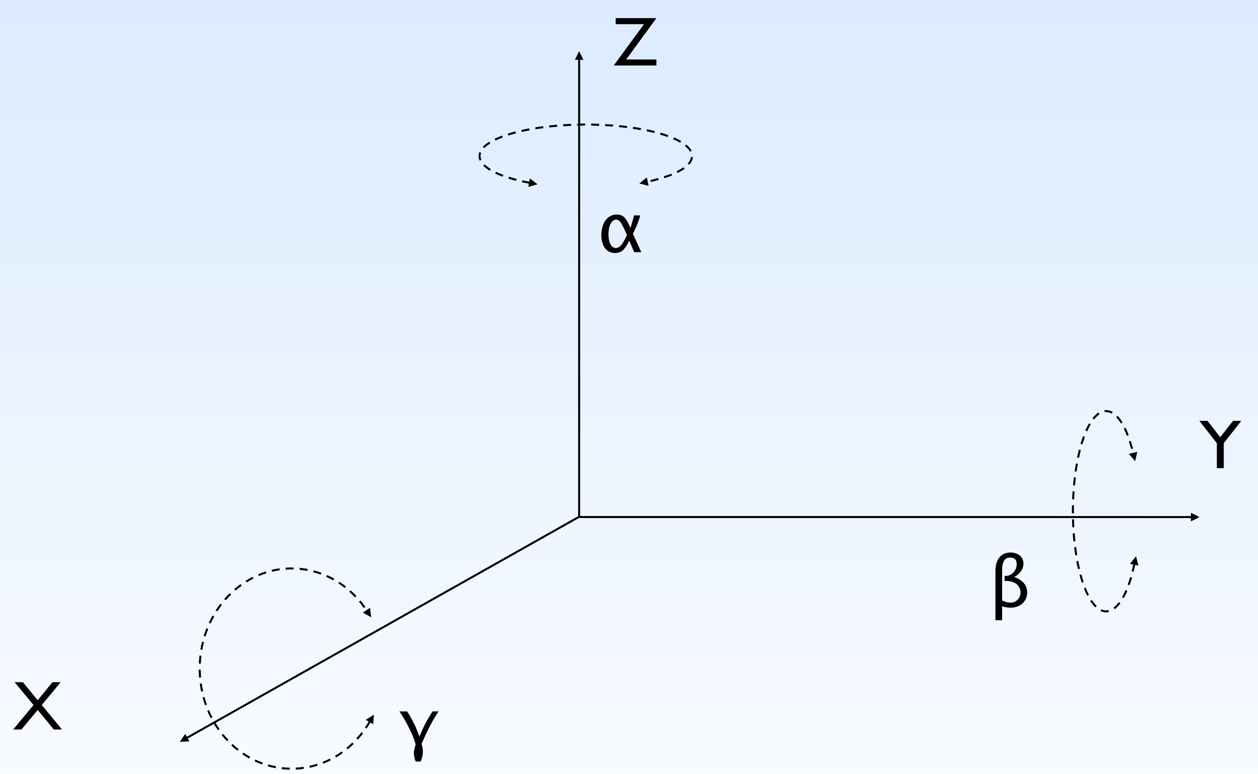
[From Grassia JGT98]

Euler Angles

Rotation defined by angles of rotation around the X-, Y-, and Z- axes.

Different conventions. For example:

$$\mathbf{R} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}$$



Gimbal Lock

When β = π/2,

$$
\mathbf{R} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}
$$

$$
= \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}
$$

$$
= \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & \sin\gamma & \cos\gamma \\ 0 & \cos\gamma & -\sin\gamma \\ 1 & 0 & 0 \end{bmatrix}
$$

$$
= \begin{bmatrix} 0 & \cos\alpha\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\cos\gamma + \sin\alpha\sin\gamma \\ 0 & \sin\alpha\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\cos\gamma - \cos\alpha\sin\gamma \\ -1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & \sin(\gamma-\alpha) & \cos(\gamma-\alpha) \\ 0 & \cos(\gamma-\alpha) & -\sin(\gamma-\alpha) \\ -1 & 0 & 0 \end{bmatrix}
$$

Gimbal Lock and Optimization

When $\beta = \pi/2$,

$$\mathbf{R} = \begin{bmatrix} 0 & \sin(\gamma - \alpha) & \cos(\gamma - \alpha) \\ 0 & \cos(\gamma - \alpha) & -\sin(\gamma - \alpha) \\ -1 & 0 & 0 \end{bmatrix}$$

the rotation by γ can be cancelled by taking $\alpha=\gamma$.

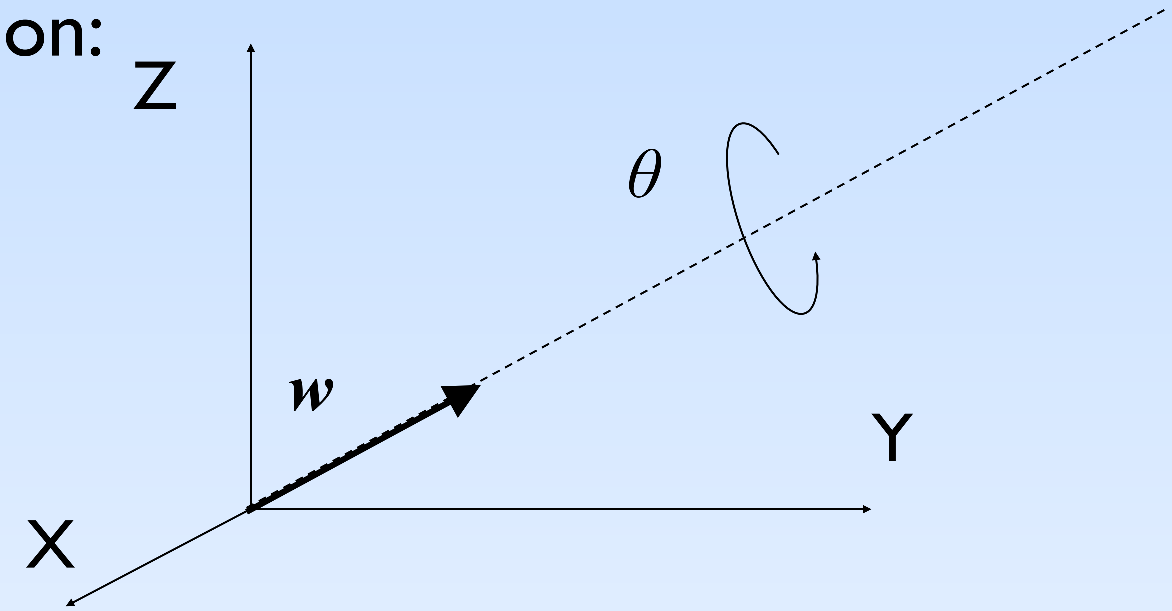
That means that

- for each possible angle θ, all (α, $\beta = \pi/2$, $\gamma = \alpha + \theta$) correspond to the same rotation matrix

=> for each possible angle θ, there is a flat valley of axis (α, $\beta = \pi/2$, $\gamma = \alpha + \theta$) in the energy to be minimized.

Axis angle representation and quaternions

A rotation about the unit vector $\boldsymbol{w}$ by an angle θ can be represented by the unit quaternion:



Quaternions are hyper-complex numbers that can be written as the linear combination $a+bi+cj+dk$, with $i^2 = j^2 = k^2 = ijk = -1$.

Can also be interpreted as a scalar plus a 3- vector: $(a, \boldsymbol{v})$.

$$q = \left(\cos\frac{\theta}{2}, \boldsymbol{w}\sin\frac{\theta}{2}\right)$$

A Unit Quaternion

To rotate a 3D point $\mathbf{M}$: write it as a quaternion p = (0, $\mathbf{M}$), and take the rotated point p' to be

$$q = \left(\cos\frac{\theta}{2}, \boldsymbol{w}\sin\frac{\theta}{2}\right)$$

No gimbal lock.

$$p' = q.p\bar{q} \quad \text{with} \quad \bar{q} = \left(\cos\frac{\theta}{2}, -\boldsymbol{w}\sin\frac{\theta}{2}\right)$$

The norm of q must be equal to 1. $||\mathbf{q}|| = 1$

In order to enforce this during optimization we have to add regularization term: $k(1 - ||\mathbf{q}||^2)$

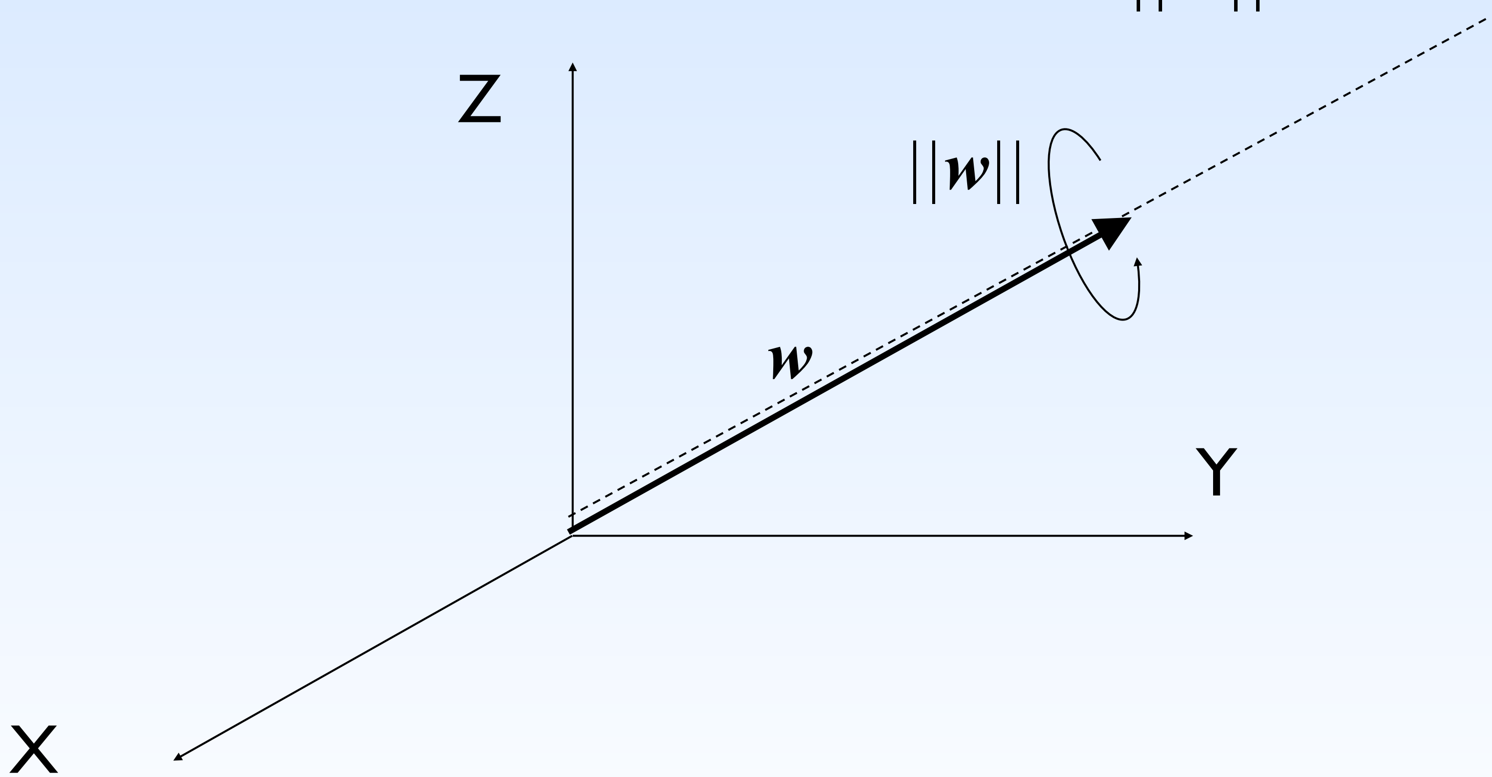
Exponential Maps

No gimbal lock;

No additional constraints; ☺

Singularities occur in a region that can easily be avoided.

Parameterization by a 3D vector $\boldsymbol{w}=[w_1,w_2,w_3]^T$: Rotation around the axis of direction $\boldsymbol{w}$ of an amount of $||\boldsymbol{w}||$



Rodrigues' Formula

skew symmetric matrix

w normalised $\frac{\omega}{||\omega||}$

$$\mathbf{w} \times \mathbf{c} = \mathbf{\Omega c} \quad \hat{\Omega} = \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{bmatrix}, \|\mathbf{w}\|_2 = 1$$

The rotation matrix can be defined as an exponential map $exp : so(3) \rightarrow SO(3)$ given by:

$$R = exp(\theta\hat{\Omega}) = \sum_{k=0}^{\infty} \frac{(\theta\hat{\Omega})^k}{k!} = I + \theta\hat{\Omega} + \frac{1}{2!}(\theta\hat{\Omega})^2 + \frac{1}{3!}(\theta\hat{\Omega})^3 + \dots$$

knowing $\hat{\Omega}^3 = -\hat{\Omega}, \hat{\Omega}^4 = -\hat{\Omega}^2, \hat{\Omega}^5 = \hat{\Omega}, \hat{\Omega}^6 = \hat{\Omega}^2, \hat{\Omega}^7 = -\hat{\Omega}$

$$exp(\theta\hat{\Omega}) = I + \left(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \dots\right)\hat{\Omega} + \left(\frac{\theta^2}{2!} - \frac{\theta^4}{4!} + \frac{\theta^6}{6!} - \dots\right)\hat{\Omega}^2$$

we obtain Rodrigues formula:

$$\mathbf{R}(\mathbf{\Omega}) = \mathbf{I} + sin\theta\hat{\mathbf{\Omega}} + (1 - cos\theta)\hat{\mathbf{\Omega}}^2$$

Rodrigues' Formula

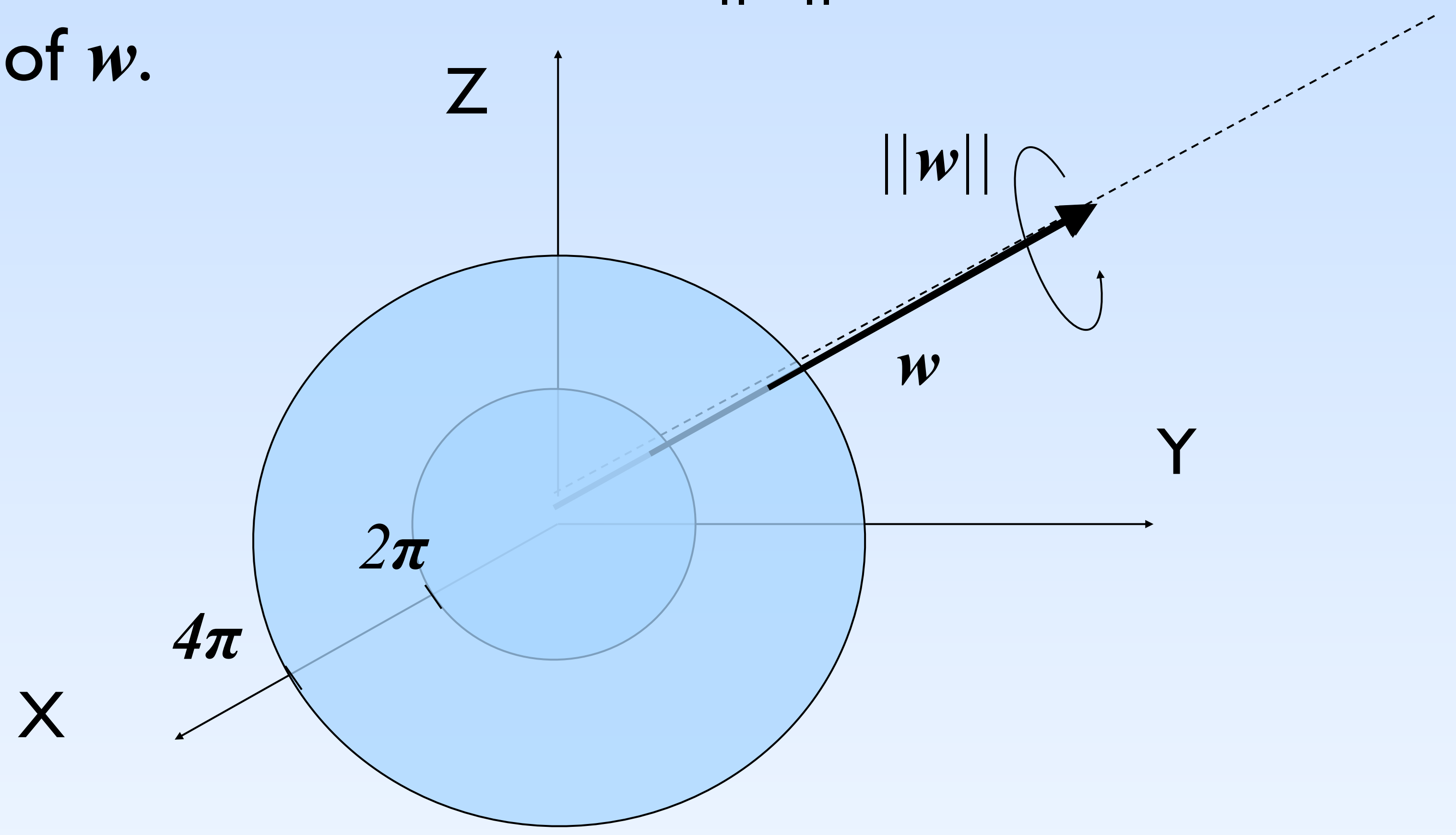
Given that $\hat{\Omega} = \frac{\Omega}{\theta}$ we can rewrite Rodrigues formula as, where Ω is screw symmetric matrix (not normalized) and $\theta = ||\omega||$:

$$R(\theta\Omega) = \exp(\theta\Omega) = I + \frac{\sin(\theta)}{\theta}\Omega + \frac{(1 - cos(\theta))}{\theta^2}\Omega^2$$

The Singularities of Exponential Maps

Rotation around the axis of direction $\boldsymbol{w}$ of an amount of $||\boldsymbol{w}||$

→ Singularities for $\boldsymbol{w}$ such that $||\boldsymbol{w}|| = 2n\pi$: No rotation, whatever the direction of $\boldsymbol{w}$.



Avoided during optimization as follows: when $||\boldsymbol{w}||$ becomes close to $2n\pi$, say higher than π, $\boldsymbol{w}$ can be replaced by $\left(1 - \frac{2\pi}{\|\boldsymbol{w}\|}\right)\boldsymbol{w}$

[From Grassia JGT98]

Linearization of small rotations

In 3D tracking:

- the camera motion between consecutive frames can often be assumed to remain small along with the corresponding rotation angles

- use a first order approximation of the rotation

$$\begin{aligned} \mathbf{M}' &= \mathbf{R}\mathbf{M} \\ &\approx (\mathbf{I} + \boldsymbol{\Omega})\mathbf{M} \\ &= \mathbf{M} + \boldsymbol{\Omega}\mathbf{M} \end{aligned}$$

$\boldsymbol{\Omega}$ is skew symmetric matrix

Quaternions with axis-angle parameterisation

To achieve the minimum number of DOF (i.e., 3), it is necessary to revert back to the axis-angle representation, this time using the following quaternion parameterization:

$$q = \Big(\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\big(\frac{w_1}{\theta}, \frac{w_2}{\theta}, \frac{w_3}{\theta}\big)\Big)$$

where $w = [w_1\ w_2\ w_3]^T$ is the axis of rotation and $\theta = \sqrt{(w_1^2 + w_2^2 + w_3^2)}$ is norm of the axis of rotation equivalent to the angle of rotation around the axis.

This allows parameterisation of quaternions with 3 DOF instead of 4, so to avoid necessary constraint on the unit norm of the quaternion.

This also facilitate computation of the derivatives using chai rules:

$$\frac{\partial R(q(w))}{\partial w_i} = \sum_{j=0}^{3} \frac{\partial R(q)}{\partial q_j} \frac{\partial q_j(w)}{\partial w_i}$$

Parameterization of the Rotation Matrix

Conclusion: Use exponential maps.

More details:

[Grassia JGT98] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.132.20&rep=rep1&type=pdf

George Terzakis , Phil Culverhouse , Guido Bugmann , Sanjay Sharma , and Robert Sutton
A Recipe on the Parameterization of Rotation Matrices for Non-Linear Optimization using Quaternions

Computing $\mathbf{J}$

We need to compute $\mathbf{J}$, the Jacobian of f:

$$f(\mathbf{p}) = \begin{bmatrix} u\left(\mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_1)\right) \\ v\left(\mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_1)\right) \\ \vdots \end{bmatrix}$$

Solution 1: Use Maple or Matlab to produce the analytical form, AND the code.

Solution 2

$$f(\mathbf{p}) = \begin{bmatrix} u\left(\mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_1)\right) \\ v\left(\mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_1)\right) \\ \vdots \end{bmatrix} = \begin{bmatrix} f_{\mathbf{M}_1}(\mathbf{p}) \\ \vdots \end{bmatrix}$$

First, decompose f:

$$f_{\mathbf{M}_1}(\mathbf{p}) = \mathbf{m}\left(\tilde{\mathbf{m}}\left(\mathbf{M}_{\mathbf{M}_1}^{cam}(\mathbf{p})\right)\right)$$

where

- $\mathbf{M}_{\mathbf{M}_1}^{cam}(\mathbf{p})$ returns $\mathbf{M}_1$ in the camera coordinates system defined by $\mathbf{p}$;
- $\tilde{\mathbf{m}}\left(\mathbf{M}_{\mathbf{M}_1}^{cam}\right)$ returns the projection of $\mathbf{M}_{\mathbf{M}_1}^{cam}$ in homogeneous coordinates;
- $\mathbf{m}(\tilde{\mathbf{m}})$ returns the 2D vector corresponding to $\tilde{\mathbf{m}}$.

Chain Rule

$$f(\mathbf{p}) = \begin{bmatrix} u\left(\mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_1)\right) \\ v\left(\mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_1)\right) \\ \vdots \end{bmatrix} = \begin{bmatrix} f_{\mathbf{M}_1}(\mathbf{p}) \\ \vdots \end{bmatrix} \text{ with } f_{\mathbf{M}_1}(\mathbf{p}) = \mathbf{m}\left(\tilde{\mathbf{m}}\left(\mathbf{M}_{\mathbf{M}_1}^{cam}(\mathbf{p})\right)\right)$$

$$\mathbf{J} = \left[\frac{\partial f}{\partial \mathbf{p}}\right] = \begin{bmatrix} \dfrac{\partial f_{\mathbf{M}_1}}{\partial \mathbf{p}} \\ \vdots \end{bmatrix} \text{ with } \left[\frac{\partial f_{\mathbf{M}_1}}{\partial \mathbf{p}}\right]_{2\times 6} = \left[\frac{\partial \mathbf{m}}{\partial \tilde{\mathbf{m}}}\right]_{2\times 3} \left[\frac{\partial \tilde{\mathbf{m}}}{\partial \mathbf{M}_{\mathbf{M}_1}^{cam}}\right]_{3\times 3} \left[\frac{\partial \mathbf{M}_{\mathbf{M}_1}^{cam}}{\partial \mathbf{p}}\right]_{3\times 6}$$

Chain Rule

$$\mathbf{m}(\tilde{\mathbf{m}}) = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \dfrac{U}{W} \\ \dfrac{V}{W} \end{bmatrix} \text{ with } \tilde{\mathbf{m}} = \begin{bmatrix} U \\ V \\ W \end{bmatrix}$$

$$\frac{\partial \mathbf{m}}{\partial \tilde{\mathbf{m}}} = \begin{bmatrix} \dfrac{\partial u}{\partial U} & \dfrac{\partial u}{\partial V} & \dfrac{\partial u}{\partial W} \\ \dfrac{\partial v}{\partial U} & \dfrac{\partial v}{\partial V} & \dfrac{\partial v}{\partial W} \end{bmatrix} = \begin{bmatrix} 1/W & 0 & -\dfrac{U}{W^2} \\ 0 & 1/W & -\dfrac{V}{W^2} \end{bmatrix}$$

Chain Rule

$$\tilde{\mathbf{m}}\left(\mathbf{M}_{\mathbf{M}_1}^{cam}\right) = \mathbf{A}\mathbf{M}_{\mathbf{M}_1}^{cam}$$

$$\frac{\partial \tilde{\mathbf{m}}}{\partial \mathbf{M}_{\mathbf{M}_1}^{cam}} = \mathbf{A}$$

Chain Rule

$$\mathbf{M}_{\mathbf{M}_1}^{cam}(\mathbf{p}) = \mathbf{R}(\mathbf{p})\mathbf{M}_1 + \mathbf{T}$$

With $\mathbf{p} = [r_1, r_2, r_3, t_1, t_2, t_3]^\mathsf{T}$ $\left(\mathbf{T} = [t_1, t_2, t_3]^\mathsf{T}\right)$:

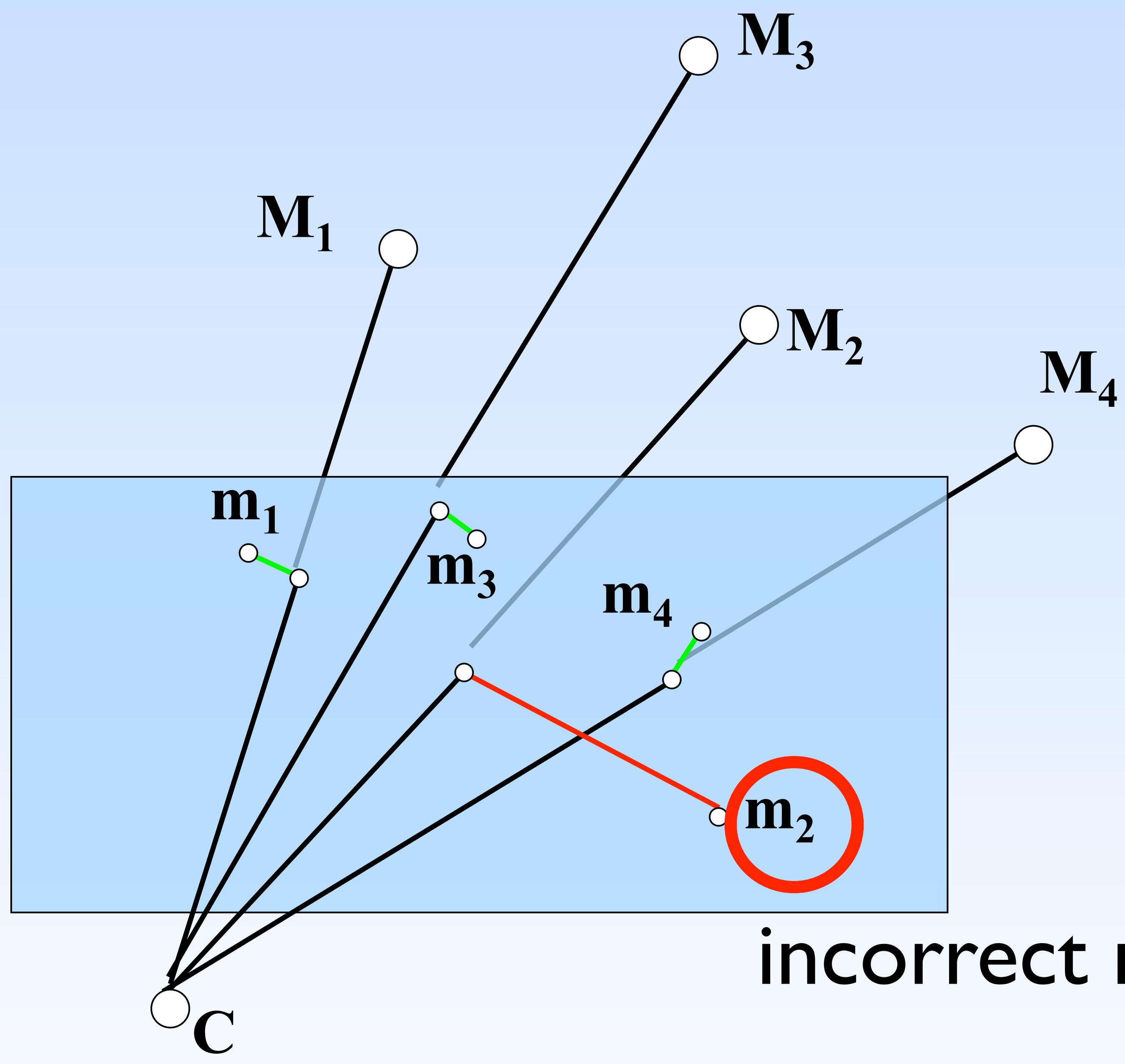
$$\frac{\partial \mathbf{M}_{\mathbf{M}_1}^{cam}}{\partial \mathbf{p}} = \left[\frac{\partial \mathbf{M}_{\mathbf{M}_1}^{cam}}{\partial r_1} \quad \frac{\partial \mathbf{M}_{\mathbf{M}_1}^{cam}}{\partial r_2} \quad \frac{\partial \mathbf{M}_{\mathbf{M}_1}^{cam}}{\partial r_3} \quad \frac{\partial \mathbf{M}_{\mathbf{M}_1}^{cam}}{\partial t_1} \quad \frac{\partial \mathbf{M}_{\mathbf{M}_1}^{cam}}{\partial t_2} \quad \frac{\partial \mathbf{M}_{\mathbf{M}_1}^{cam}}{\partial t_3} \right]$$

$$= \left[\left[\frac{\partial \mathbf{R}}{\partial r_1}\right]\mathbf{M}_1 \quad \left[\frac{\partial \mathbf{R}}{\partial r_2}\right]\mathbf{M}_1 \quad \left[\frac{\partial \mathbf{R}}{\partial r_3}\right]\mathbf{M}_1 \quad \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix} \right]$$

The matrices $\left[\frac{\partial \mathbf{R}}{\partial r_i}\right]_{3\times 3}$ can be computed from the expansion of $\mathbf{R}$.

In C, one can use the `cvRodrigues` function from the OpenCV library.

What if there are Outliers ?

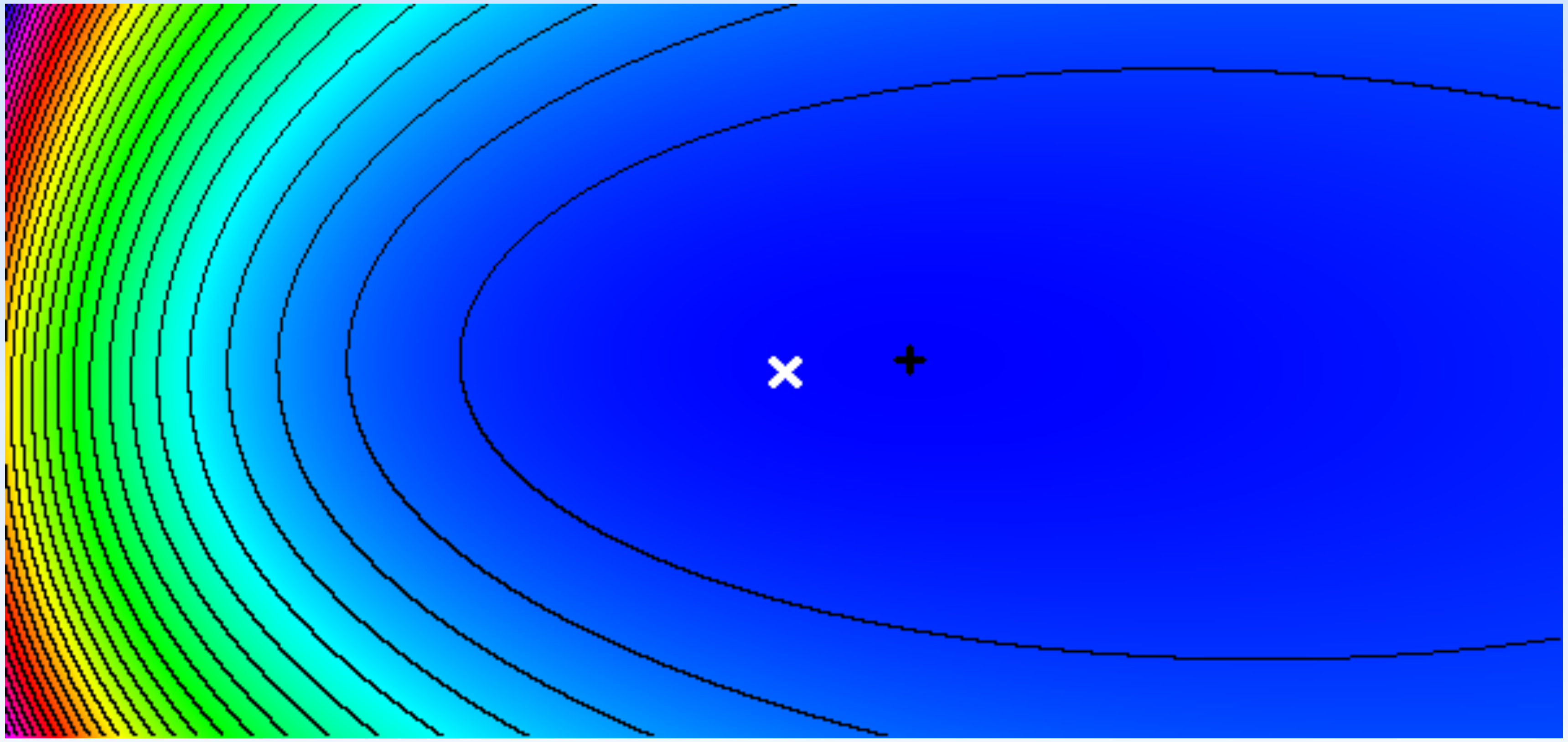


incorrect measure

Gaussian Noise on the Projections + 20% outliers

White cross: true camera position;

Black cross: global minimum of the objective function.



What Happened ?

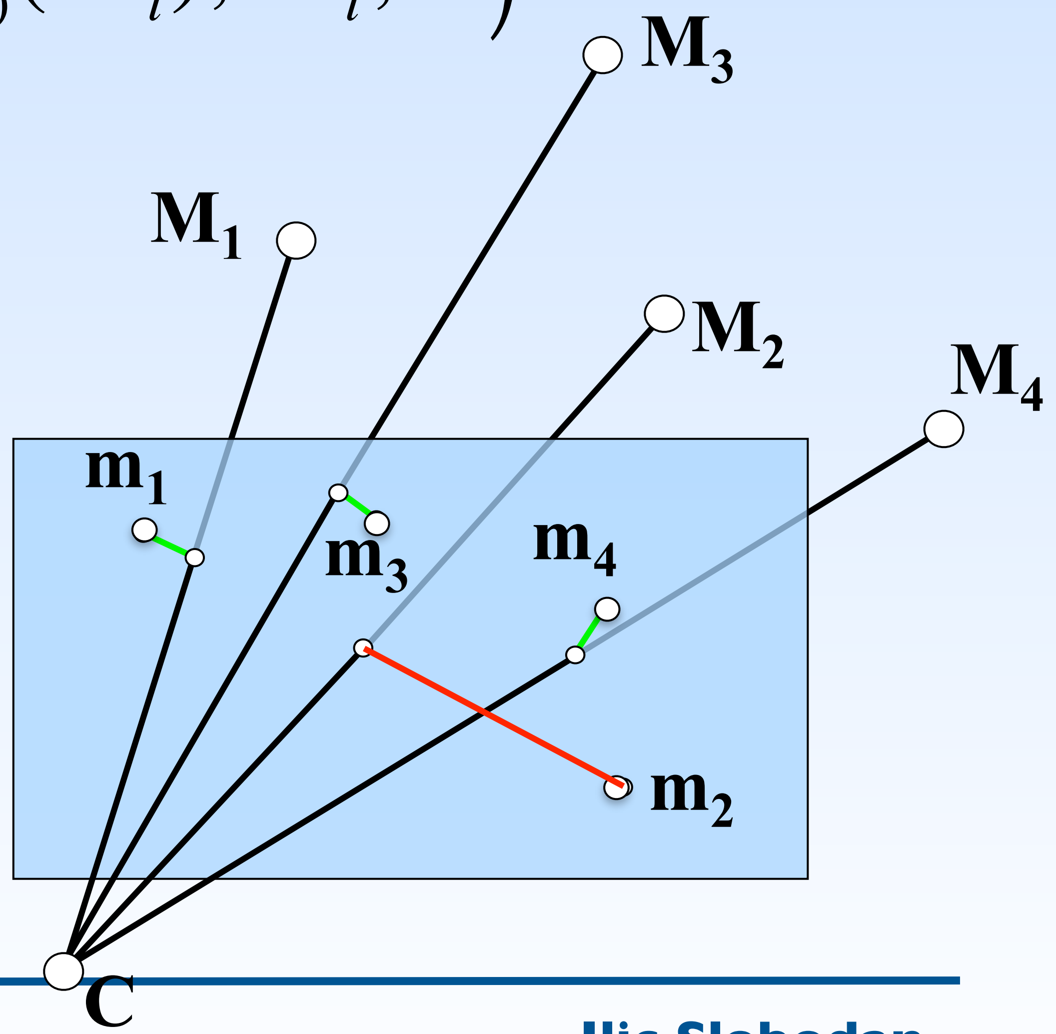
Bayesian interpretation:

$$\arg\min_{\mathbf{p}} \sum_i \left\| \mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_i) - \mathbf{m}_i \right\|^2$$

$$= \quad \arg\max_{\mathbf{p}} \prod_i N\left(\mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_i);\ \mathbf{m}_i, \sigma\mathbf{I}\right)$$

The error on the 2D point locations $\mathbf{m}_i$ is assumed to have a Gaussian (Normal) distribution with identical covariance matrices $\sigma\mathbf{I}$, and independent;

This assumption is violated when $\mathbf{m}_i$ is an outlier.



(the 2 equivalent formulations)

$$\min_{\mathbf{p}} \sum_i \left\| \mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_i) - \mathbf{m}_i \right\|^2$$

$$= \min_{\mathbf{p}} \sum_i \left(\mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_i) - \mathbf{m}_i \right)^{\mathrm{T}} \left(\mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_i) - \mathbf{m}_i \right)$$

$$= \min_{\mathbf{p}} \sum_i \left(\mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_i) - \mathbf{m}_i \right)^{\mathrm{T}} \begin{pmatrix} 1/\sigma & 0 \\ 0 & 1/\sigma \end{pmatrix} \left(\mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_i) - \mathbf{m}_i \right)$$

$-\log$

$$= \max_{\mathbf{p}} \exp \sum_i -\left(\mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_i) - \mathbf{m}_i \right)^{\mathrm{T}} \Sigma_{\mathbf{m}} \left(\mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_i) - \mathbf{m}_i \right)$$

$$\text{with } \Sigma_{\mathbf{m}} = \begin{pmatrix} 1/\sigma & 0 \\ 0 & 1/\sigma \end{pmatrix}$$

$$= \max_{\mathbf{p}} \prod_i \frac{1}{\sqrt{(2\pi)^2 |\Sigma_{\mathbf{m}}|}} \exp\left(-\left(\mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_i) - \mathbf{m}_i \right)^{\mathrm{T}} \Sigma_{\mathbf{m}} \left(\mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_i) - \mathbf{m}_i \right) \right)$$

$$= \max_{\mathbf{p}} \prod_i N\left(\mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_i); \mathbf{m}_i, \Sigma_{\mathbf{m}} \right)$$

Robust estimation

Idea:

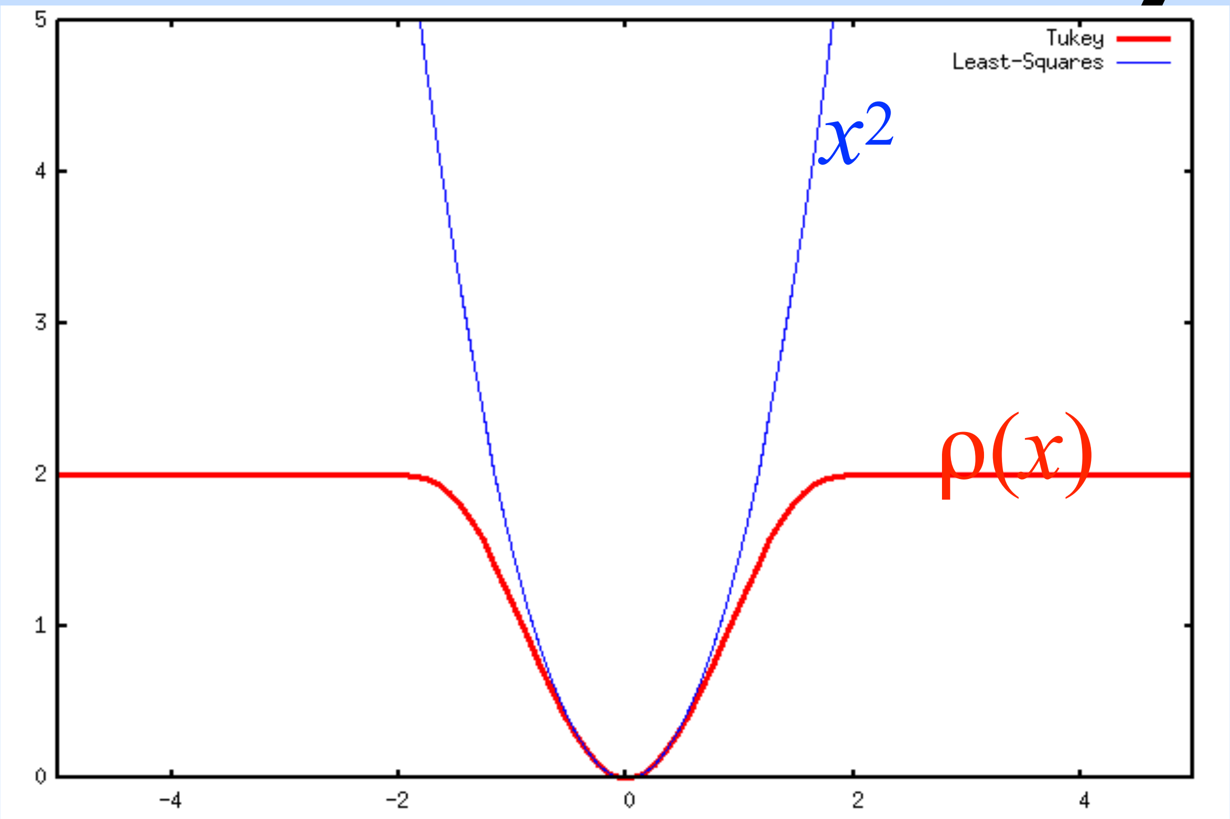
Replace the Normal distribution by a more suitable distribution,

or equivalently replace the least-squares estimator by a "robust estimator":

$$\min_{\mathbf{p}} \sum_i \left\| \mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_i) - \mathbf{m}_i \right\|^2 = \sum_i r_i^2$$

$$\rightarrow \quad \min_{\mathbf{p}} \sum_i \rho\left(\mathrm{Proj}_{\mathbf{A},\mathbf{R}(\mathbf{p}),\mathbf{T}(\mathbf{p})}(\mathbf{M}_i) - \mathbf{m}_i \right) = \sum_i \rho(r_i)$$

Example of an M-estimator: The Tukey Estimator



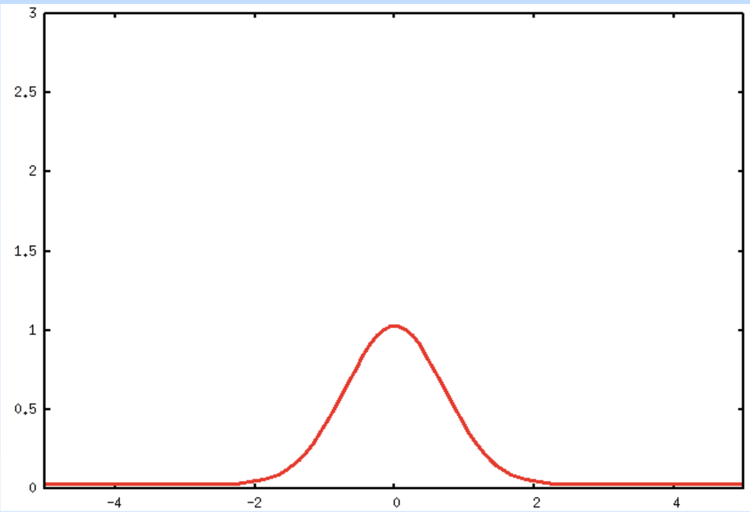
$$\begin{cases} \text{if } |x| \le c & \rho(x) = \dfrac{c^2}{6}\left(1 - \left[1 - \left(\dfrac{x}{c}\right)^2\right]^3\right) \\ \text{if } |x| > c & \rho(x) = \dfrac{c^2}{6} \end{cases}$$

The Tukey estimator assumes the measures follow a distribution that is a mixture of:

- a Normal distribution, for the inliers,
- a uniform distribution, for the outliers.

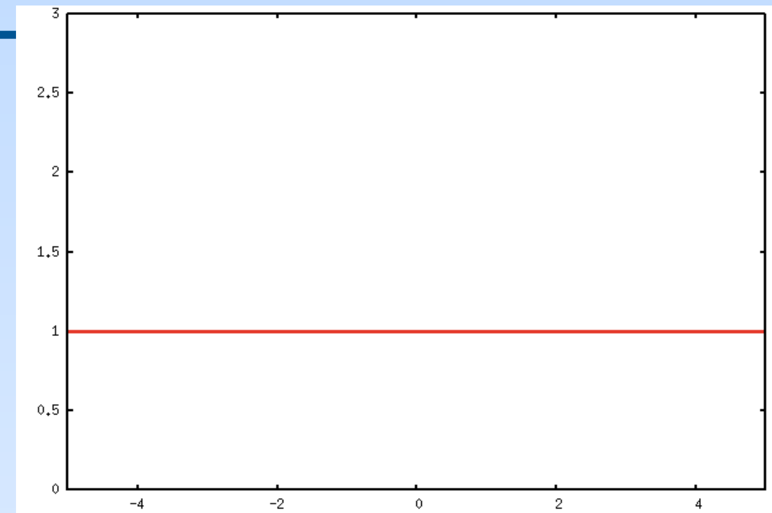
The threshold c is usually taken to be proportional to the measured standard deviation of the residual errors for inlier data.

Normal distribution (inliers)



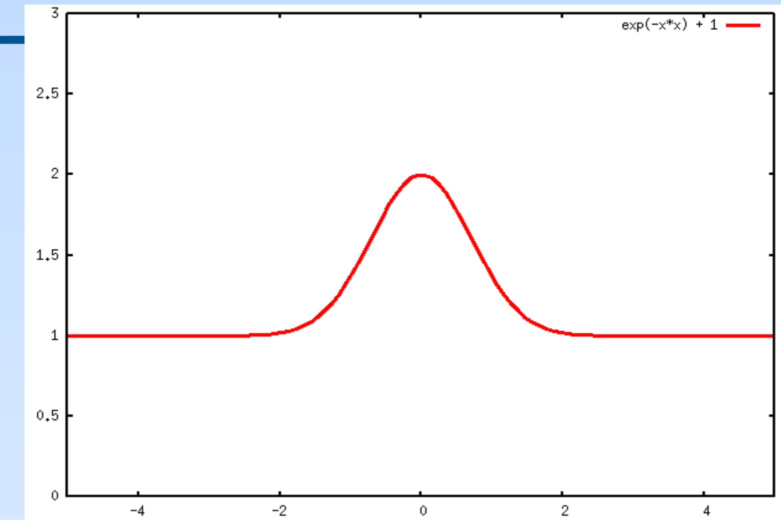
+

Uniform distribution (outliers)

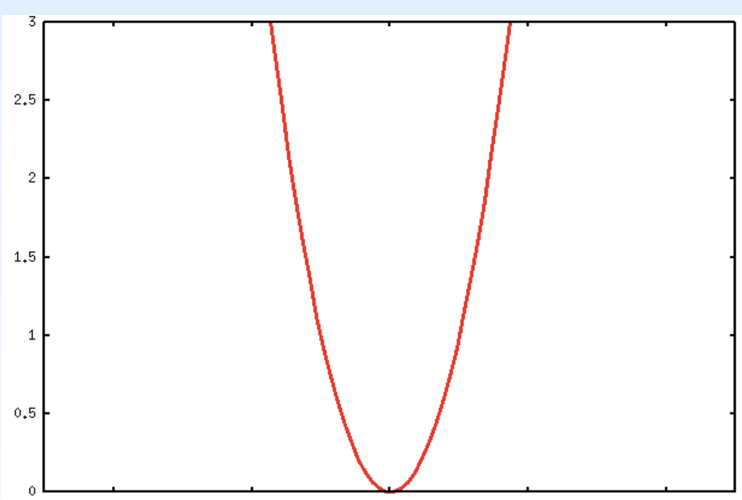


=

Mixture

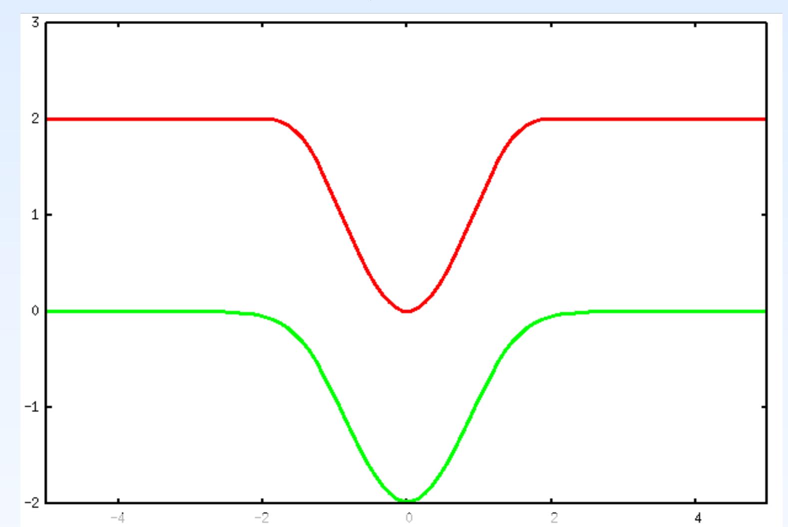


-log(.)



Least-squares

-log(.)

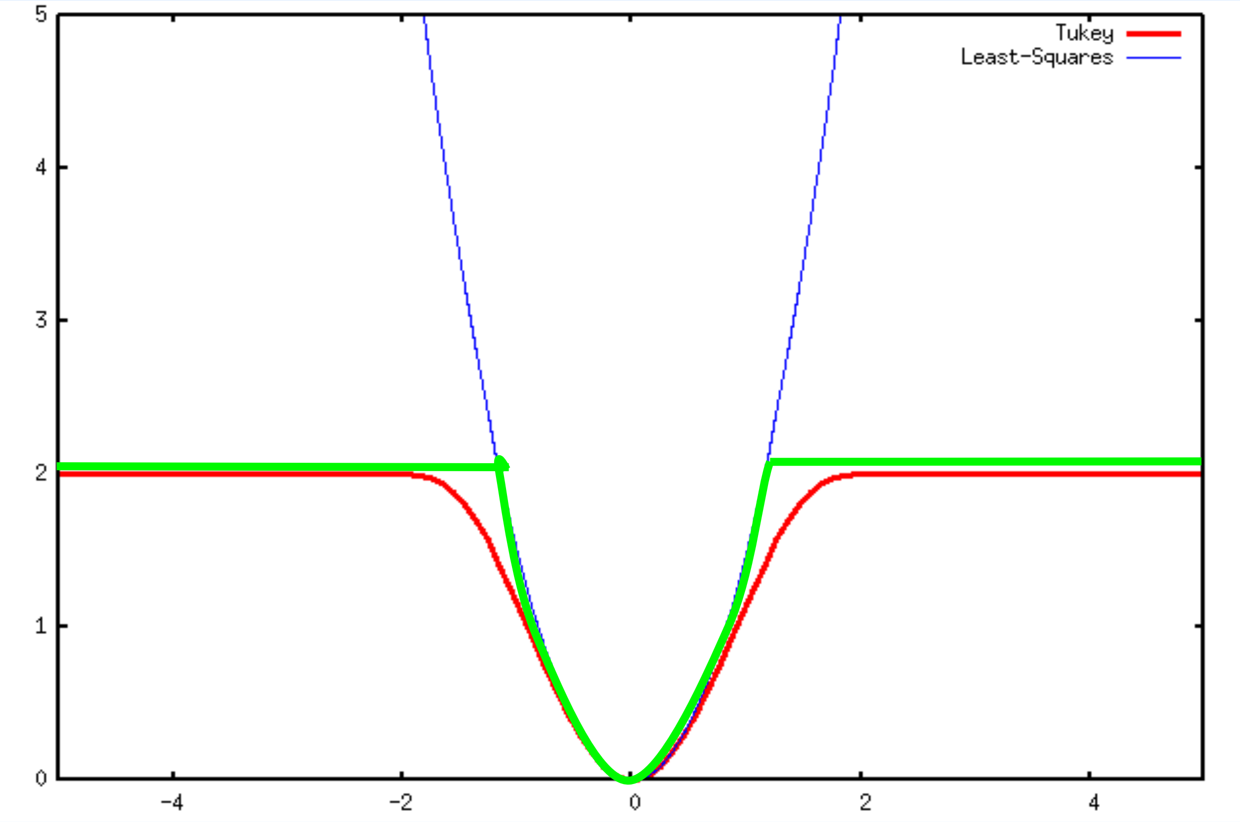


Tukey estimator

The Tukey Estimator in Levenberg-Marquardt Optimization

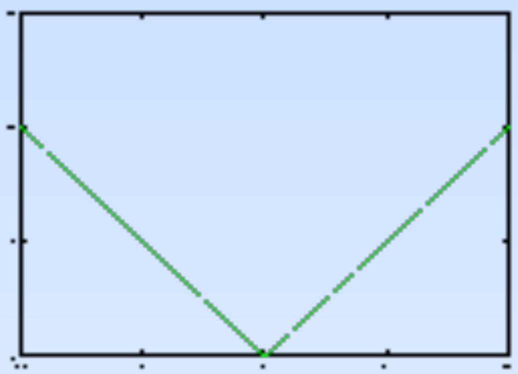
Use the following approximation:

$$\begin{cases} \text{if } |x| \le \tilde{c} \quad \rho(x) = \dfrac{1}{2}x^2 \quad (\text{least} - \text{squares}) \\ \text{if } |x| > \tilde{c} \quad \rho(x) = \dfrac{1}{2}\tilde{c}^2 \quad (\text{constante}) \end{cases}$$

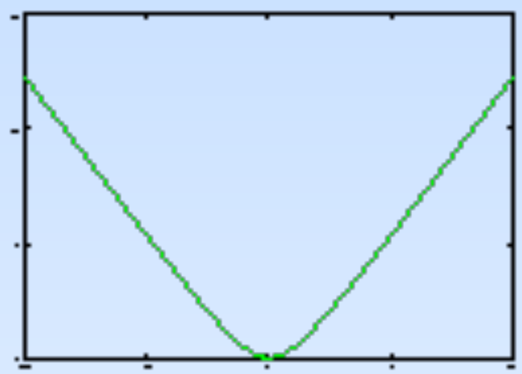


Other M-Estimators

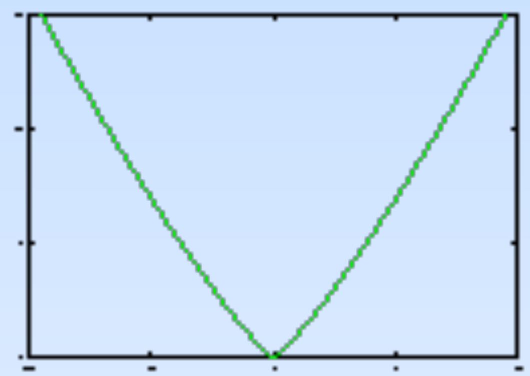
Least-absolute



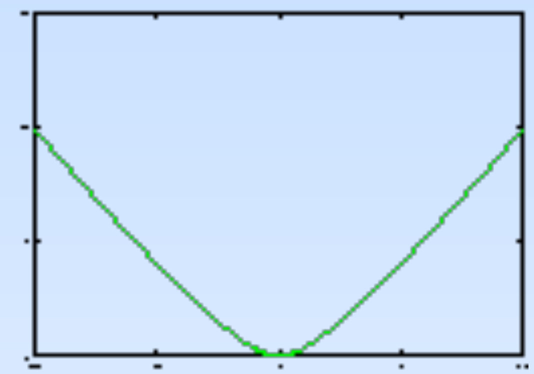
$L_1 - L_2$



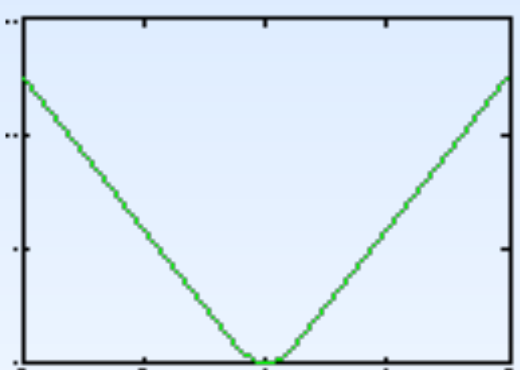
Least-power



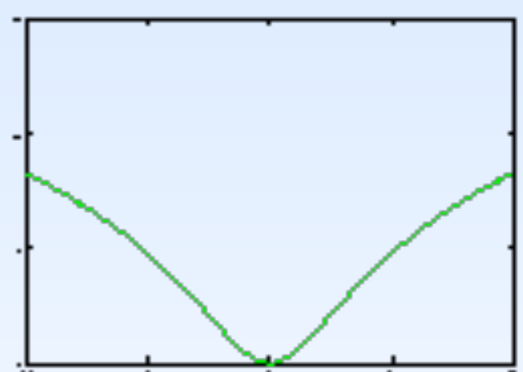
Fair



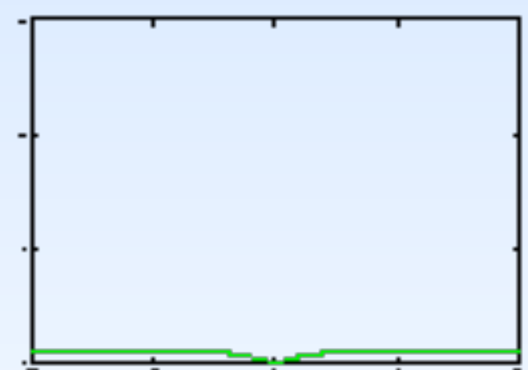
Huber



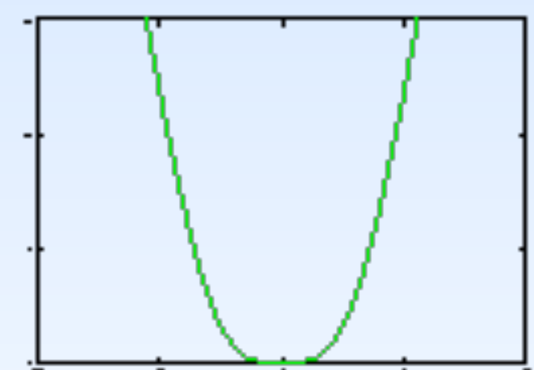
Cauchy



Geman-McClure



Welsch



Use of robust estimator with GN or LM minimisation

The Gauss-Newton and Levenberg-Marquardt algorithms can still be applied to minimize the sum of residual errors $E(\theta) = \sum_i r_i^2$ after the introduction of M-estimators $E(\theta) = \sum_i \rho(r_i)$, even if the M-estimators can be complex functions. We solve this by finding derivative of the objective function in aspect to parameters:

$$\frac{\partial E}{\partial \theta} = \sum_i \rho^{'}(r_i) r_i \theta_i$$

This is simply done by weighting the residuals ri at each iteration step: Each ri is replaced by therefore the weight should be chosen as:

$$w_i = \frac{\rho(r_i)}{ri}$$

In the case of the LevenbergMarquardt algorithm, Δi can be computed as be changed as:

$$\mathbf{\Delta_i} = -(\mathbf{J}^T \mathbf{W} \mathbf{J} + \lambda \mathbf{I})^{-1} \mathbf{J}^T \mathbf{W} \epsilon_i$$

where weight matrix is $\mathbf{W} = diag(\ldots w_i \ldots)$

Scale of the residuals

- M-estimator (Tukey and Huber) constant **c** has been chosen assuming that measurements have normal distribution (standard deviation of 1 and zero mean) and therefore they provide asymptotic efficiency of 95% of linear regression. **c=4.685** Tukey and **c=1.345** (Huber)

- However, measurements with outliers are not normally(Gaussian) distributed, so the residuals must be scaled, i.e. every $\rho(r_i)$ should be replaced with $\rho(r_i/s)$ where **s** is estimated scale parameter.

- The simplest estimation of **s** is done using median absolut deviation of the residuals:

$$MAD = median\{|r_i|\}$$

where $\hat{s} = MAD/0.6745$, which is based on the idea expectation of MAD being $E(MAD) = 0.6745$ for normal distribution.

Drawbacks of the Tukey Estimator

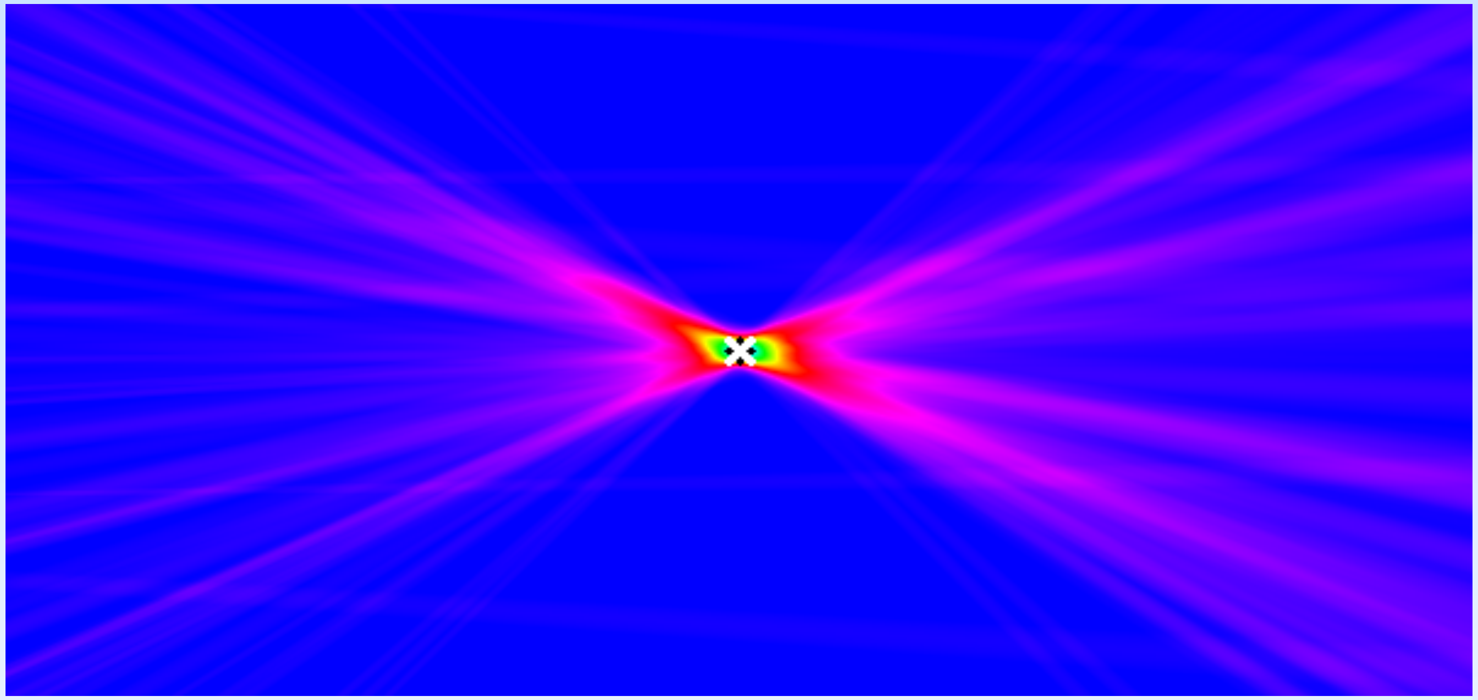
- Non-convex → creates local minimas;

- Function becomes flat when too far from the global minimum.

Gaussian Noise on the Projections + 20% outliers + Tukey estimator

White cross: true camera position;

Black cross: global minimum of the object function.



The global minimum is very close to the true camera pose.

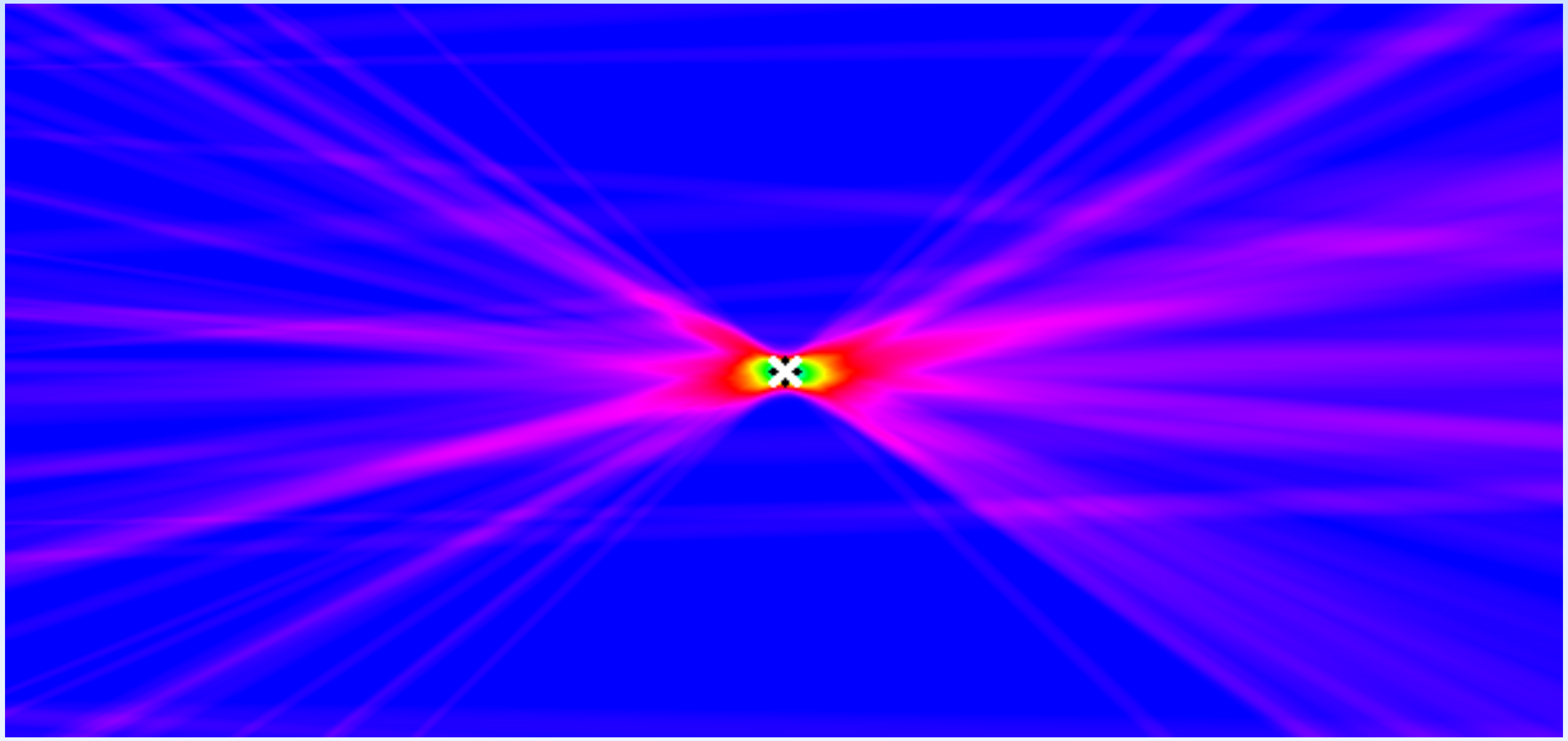
BUT:

- local minima;
- the objective function is flat where all the correspondences are considered outliers.

Gaussian Noise on the Projections + 50% outliers + Tukey estimator

White cross: true camera position;

Black cross: global minimum of the object function.

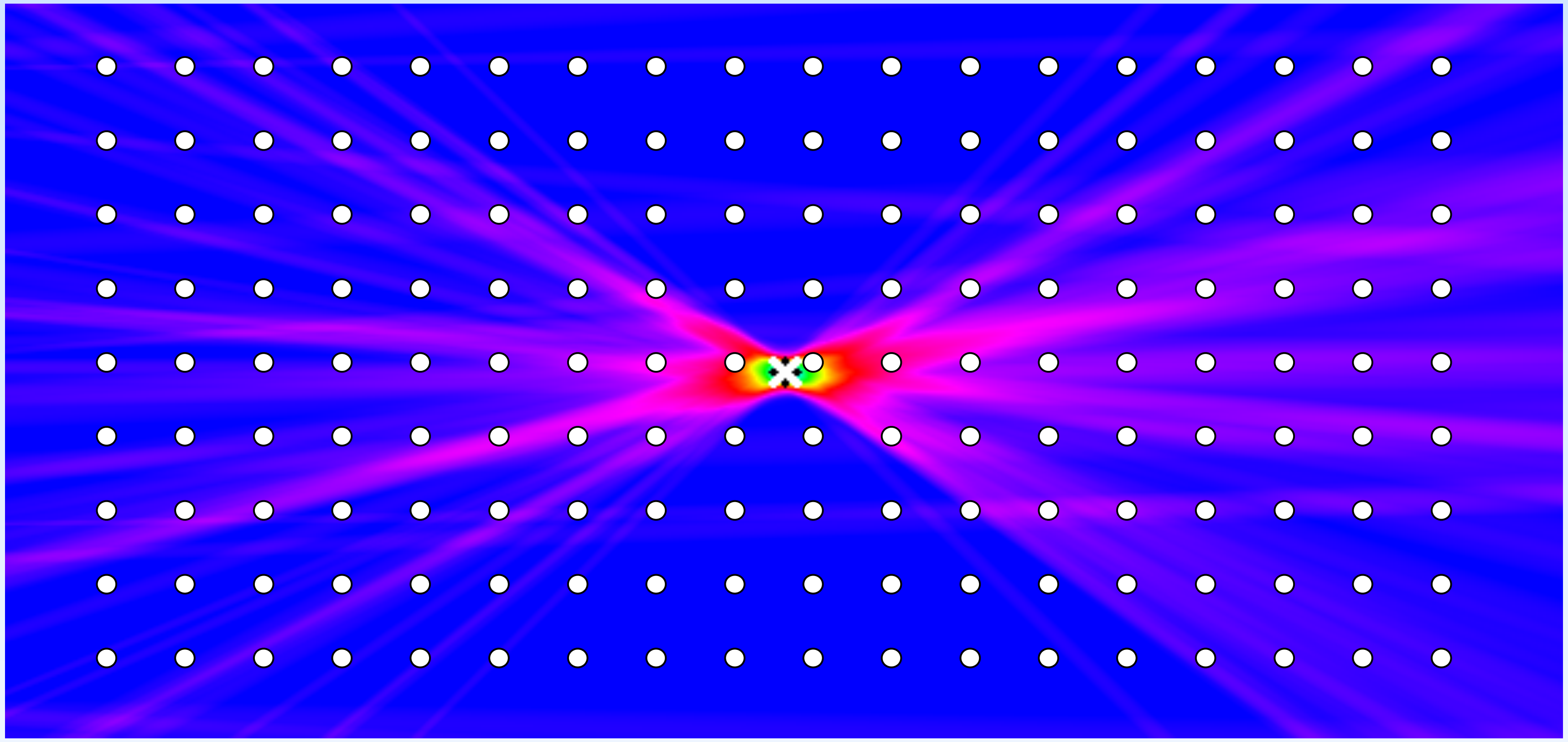


Even more local minimums.

Numerical optimization can get trapped into a local minimum.

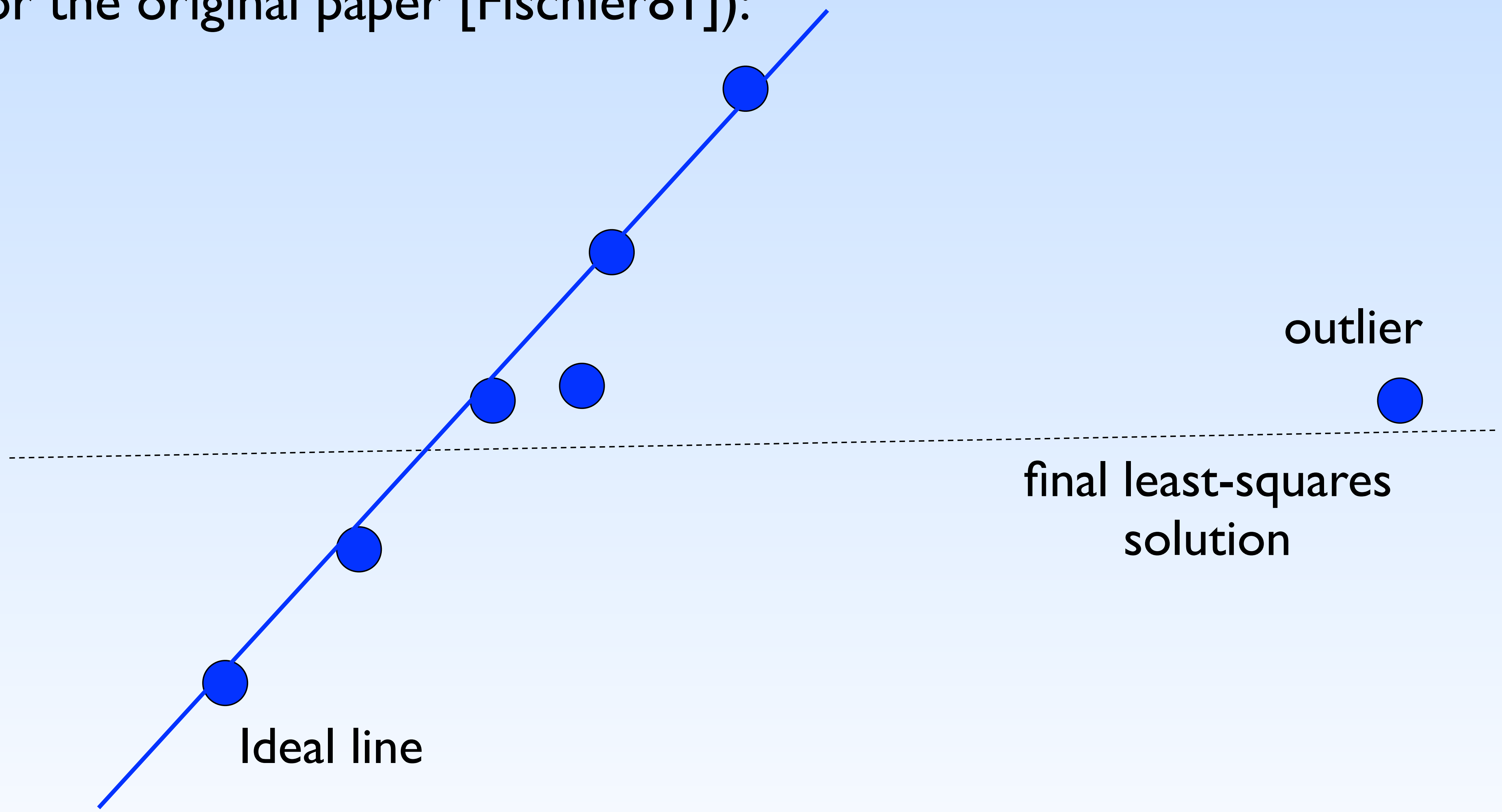
RANSAC

Idea: sampling the space of solutions (the camera pose space here):



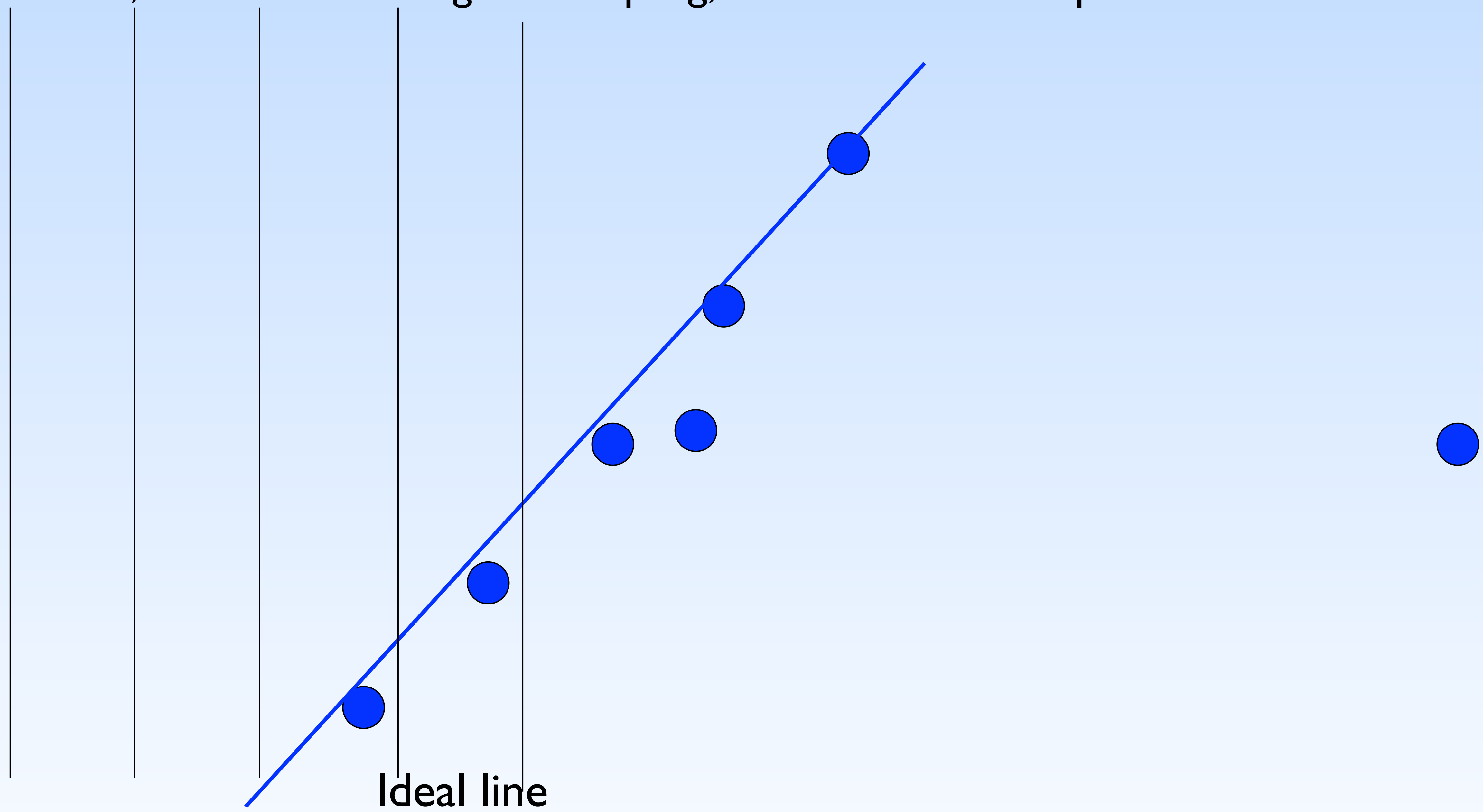
RANSAC
RANdom SAmple Consensus

Line fitting: the "Throwing Out the worst residual" heuristics can fail (Example for the original paper [Fischler81]):



RANSAC

As before, we could do a regular sampling, but would not be optimal:



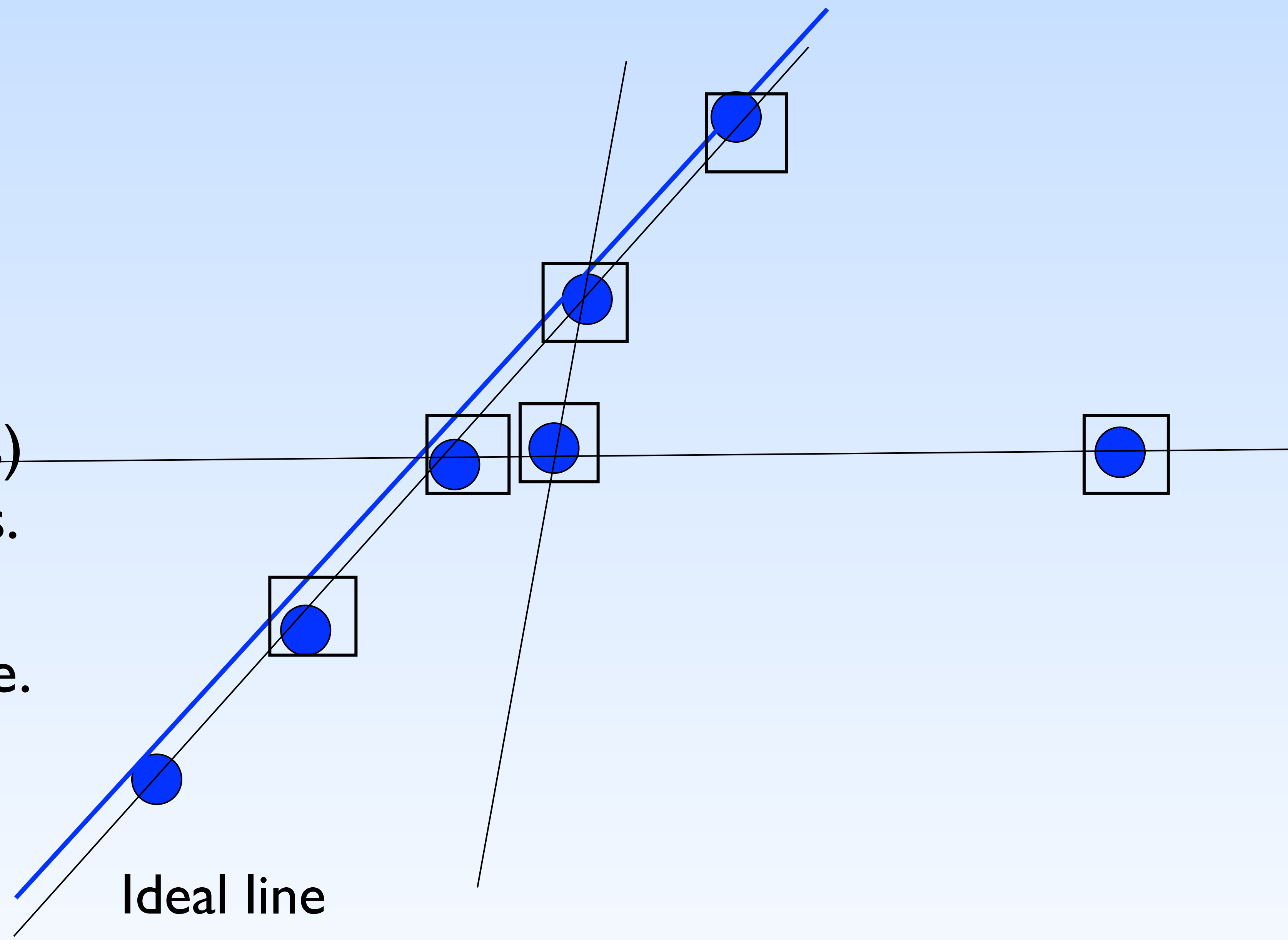
RANSAC

Idea:

•Generate hypotheses from subsets of the measurements.

• If a subset contains no gross errors, the estimated parameters (the hypothesis) are closed to the true ones.

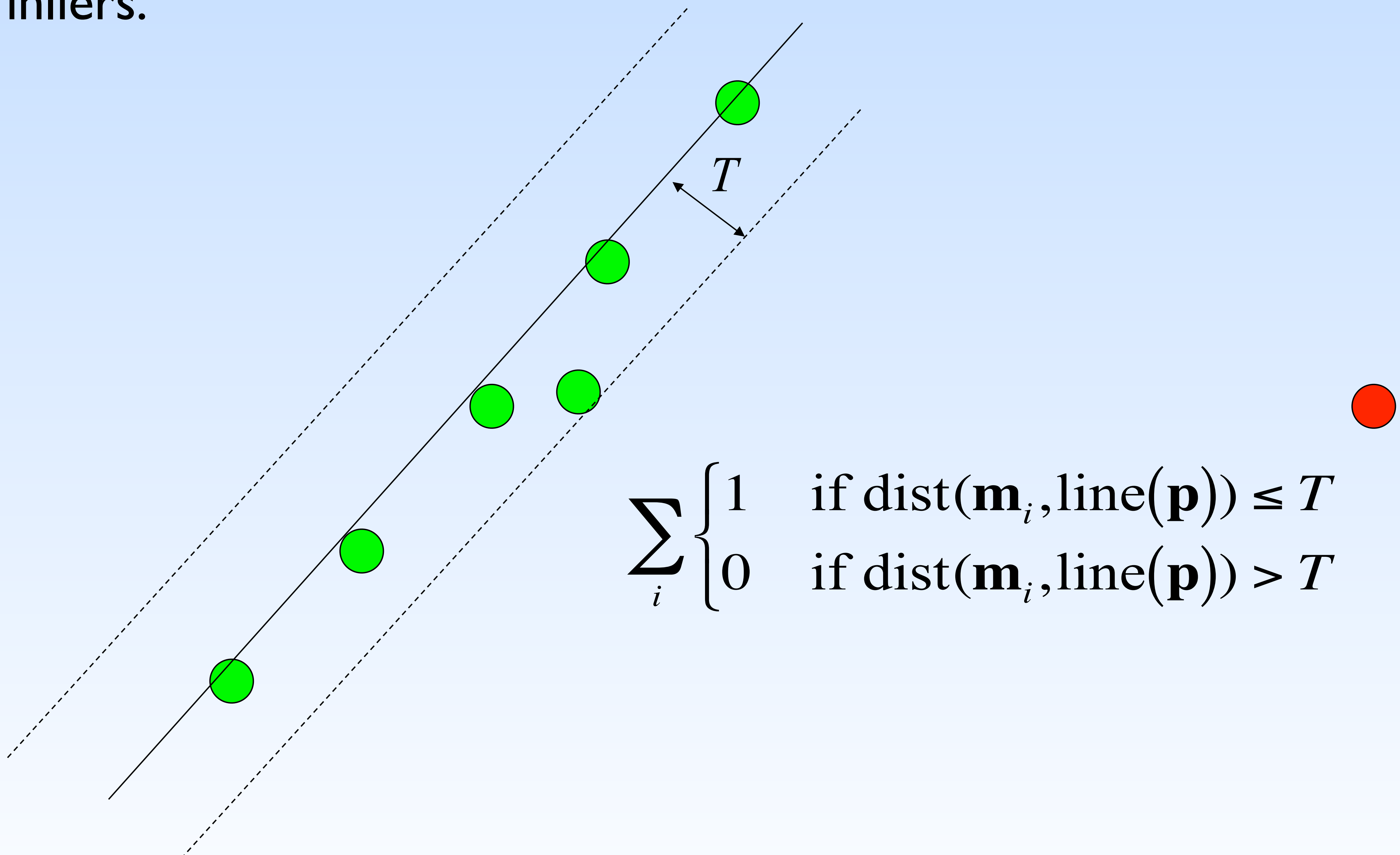
•Take several subsets at random, retain the best one.



The quality of a hypothesis is evaluated by the number of measures that lie "close enough" to the predicted line.

We need to choose a threshold (T) to decide if the measure is "close enough".

RANSAC returns the best hypothesis, i.e the hypothesis with the largest number of inliers.



$$\sum_i \begin{cases} 1 & \text{if } \mathrm{dist}(\mathbf{m}_i, \mathrm{line}(\mathbf{p})) \leq T \\ 0 & \text{if } \mathrm{dist}(\mathbf{m}_i, \mathrm{line}(\mathbf{p})) > T \end{cases}$$

How many samples to choose?

e = probability that a point is an outlier
s = number of points in a sample
N = number of samples (we want to compute this)
p = desired probability that we get a good sample

Solve the following for N:

$$1 - (1 - (1 - e)^s)^N = p$$

Where in the world did that come from? ….

From Robert Colins, Penn State University

How many samples to choose?

e = probability that a point is an outlier
s = number of points in a sample
N = number of samples (we want to compute this)
p = desired probability that we get a good sample

$$1 - (1 - (\underbrace{1 - e}_{\text{Probability that choosing one point yields an inlier}})^s)^N = p$$

How many samples to choose?

e = probability that a point is an outlier
s = number of points in a sample
N = number of samples (we want to compute this)
p = desired probability that we get a good sample

$$1 - (1 - \underbrace{(1 - e)^s}_{\text{Probability of choosing s inliers in a row (sample only contains inliers)}})^N = p$$

How many samples to choose?

e = probability that a point is an outlier
s = number of points in a sample
N = number of samples (we want to compute this)
p = desired probability that we get a good sample

$$1 - (\underbrace{1 - (1 - e)^s}_{\text{Probability that one or more points in the sample were outliers (sample is contaminated).}})^N = p$$

How many samples to choose?

e = probability that a point is an outlier
s = number of points in a sample
N = number of samples (we want to compute this)
p = desired probability that we get a good sample

$$1 - \underbrace{\left(1 - (1 - e)^{s}\right)^{N}}_{\text{Probability that N samples were contaminated.}} = p$$

How many samples to choose?

e = probability that a point is an outlier
s = number of points in a sample
N = number of samples (we want to compute this)
p = desired probability that we get a good sample

$$\underbrace{1 - (1 - (1 - e)^{s})^{N}}_{} = p$$

Probability that at least one sample was not contaminated (at least one sample of s points is composed of only inliers).

How many samples?

Choose N so that, with probability p, at least one random sample is free from outliers. e.g. p=0.99

$$(1-(1-e)^s)^N = 1-p$$

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

	proportion 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

Pose Estimation

To apply RANSAC to pose estimation, we need a way to compute a camera pose from a subset of measurements, for example a P3P algorithm.

Since RANSAC only provides a solution estimated with a limited number of data, it must be followed by a robust minimization to refine the solution.