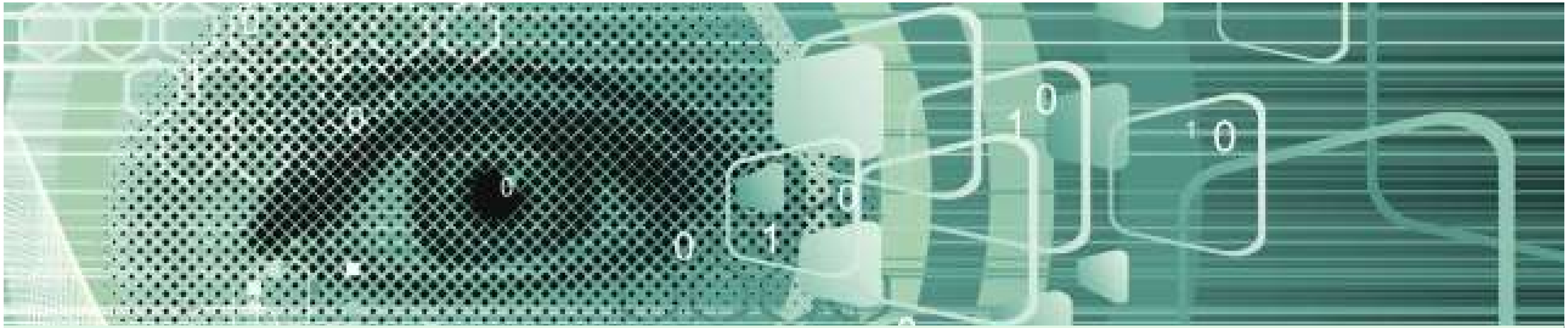


Lecture 08- Nonlinear least square & RANSAC

EE382-Visual localization & Perception

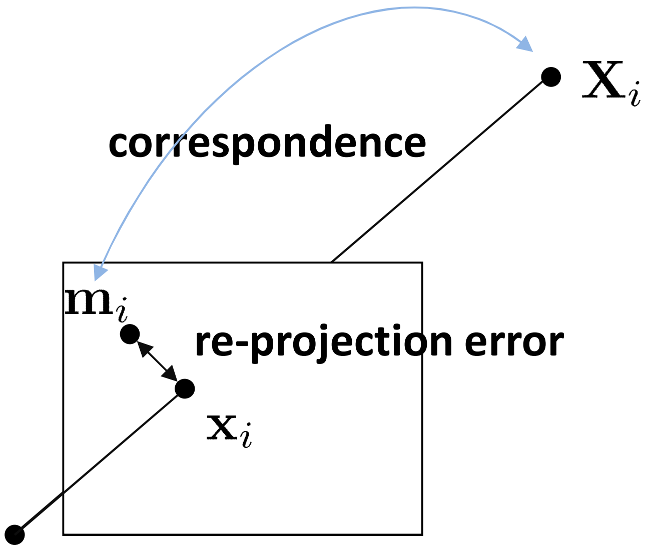
Danping Zou @Shanghai Jiao Tong University

Nonlinear optimization problem

- **Optimization approach**
  - Nonlinear least squared problem

$$\min_{R,\mathbf{t}} \sum_{i=1}^{N} r_i(R,\mathbf{t})^2 \quad (N \geq 3)$$

  - Where $r_i(\cdot)$ is the re-projection error of the corresponding points $\mathbf{X}_i \leftrightarrow \mathbf{m}_i$.



Nonlinear least square problem

- Find a minimizer $\mathbf{x}^*$ of the nonlinear objective function $f(\mathbf{x})$:

$$f(\mathbf{x}) = \frac{1}{2}\|\mathbf{r}(\mathbf{x})\|^2 = \frac{1}{2}\sum_{i=1}^{m} r_i(\mathbf{x})^2$$

- where $\mathbf{x} \in \mathbb{R}^n$,

$$\mathbf{r}(\mathbf{x}) = \begin{pmatrix} r_1(\mathbf{x}) \\ \vdots \\ r_m(\mathbf{x}) \end{pmatrix} \in \mathbb{R}^m$$

$$r_i(\mathbf{x}) = y_i - h_i(\mathbf{x}), \quad i = 1, ..., m$$

- Here y_i are the measured data. The nonlinearity arises ***only*** from the ***observation function*** $h_i(\mathbf{x})$.

Derivative & Jacobian matrix

- **Vector derivative** - Let $\mathbf{x} = (x_1 \cdots x_n)^T \in \mathbb{R}^n$, The *1×n vector derivative* operator is denoted by :

$$\frac{\partial}{\partial \mathbf{x}} = (\frac{\partial}{\partial x_1} \cdots \frac{\partial}{\partial x_n})$$

- For a general m-dimensional nonlinear function $\mathbf{h}(\mathbf{x}) \in \mathbb{R}^m$, we define :

$$\frac{\partial}{\partial \mathbf{x}}\mathbf{h}(x) \triangleq \begin{pmatrix} \frac{\partial}{\partial \mathbf{x}} h_1(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial \mathbf{x}} h_m(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} \frac{\partial}{\partial x_1} h_1(\mathbf{x}) & \cdots & \frac{\partial}{\partial x_n} h_1(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_1} h_m(\mathbf{x}) & \cdots & \frac{\partial}{\partial x_n} h_m(\mathbf{x}) \end{pmatrix} \in \mathbb{R}^{m \times n}$$

This is known as the ***Jacobian matrix***.

Computation of Jacobian matrix

- The following results are very useful :

$$\frac{\partial}{\partial \mathbf{x}} \mathbf{a}^T \mathbf{x} = \frac{\partial}{\partial \mathbf{x}} \mathbf{x}^T \mathbf{a} = \mathbf{a}^T$$

(Inner product)

$$\frac{\partial}{\partial \mathbf{x}} \mathbf{A} \mathbf{x} = \mathbf{A}$$

(Matrix-vector product)

$$\frac{\partial}{\partial \mathbf{x}} \mathbf{x}^T \Omega \mathbf{x} = 2\mathbf{x}^T \Omega = 2\Omega \mathbf{x}^T \quad (\Omega = \Omega^T)$$

(Weighted 2-norm squared)

$$\frac{\partial}{\partial \mathbf{x}} \mathbf{h}(\mathbf{g}(\mathbf{x})) = \frac{\partial \mathbf{h}}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial \mathbf{x}}$$

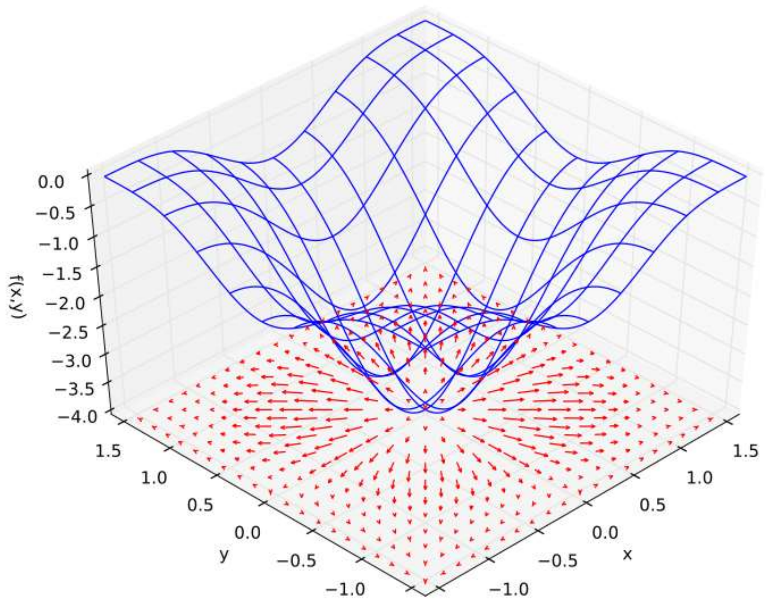
(Chain rule)

Gradient

- The **gradient** of a scalar-valued function $f(x)$ is defined by

$$\nabla_{\mathbf{x}} f(\mathbf{x}) \triangleq \left(\frac{\partial}{\partial \mathbf{x}} f(\mathbf{x})\right)^T$$

- The **gradient** gives that local ***direction of steepest ascent*** (increase in value) in the domain of definition.



The gradients of the function

$$f(x,y) = -(\cos(2x) + \cos(2y))^2$$

are indicated by the vector fields on the bottom plane [from Wikipedia]

Taylor Series Expansion

- The Taylor series expansion of a vector-valued function $\mathbf{h}(\mathbf{x})$ about a point $\mathbf{x}_0$ as (First-order)

$$\mathbf{h}(\mathbf{x}) = \mathbf{h}(\mathbf{x}_0 + \Delta\mathbf{x}) \approx \mathbf{h}(\mathbf{x}_0) + \frac{\partial \mathbf{h}(\mathbf{x}_0)}{\partial \mathbf{x}}\Delta x$$

- Let $\Delta\mathbf{y} = \mathbf{h}(\mathbf{x}) - \mathbf{h}(\mathbf{x}_0)$, the Jacobian matrix $H(\mathbf{x}) \triangleq \frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}}$ provides the *linearization* of $\mathbf{h}(\mathbf{x})$,

$$\Delta\mathbf{y} \approx H(\mathbf{x}_0)\Delta\mathbf{x}$$

$$\mathbf{dy} = H(\mathbf{x}_0)\mathbf{dx}$$

Taylor Series Expansion

- The **Taylor series expansion** of a scalar-valued function $f(\mathbf{x})$ is written as (second order)

$$f(\mathbf{x}_0 + \Delta\mathbf{x}) \approx f(\mathbf{x}_0) + \frac{\partial f(\mathbf{x}_0)}{\partial \mathbf{x}}\Delta x + \frac{1}{2}\Delta\mathbf{x}^T \mathcal{H}(\mathbf{x}_0)\Delta\mathbf{x}$$

- $\mathcal{H}(\mathbf{x}_0)$ denotes the ***Hessian matrix*** of second-order partial derivatives,

$$\mathcal{H}(\mathbf{x}_0) \triangleq \nabla^2 f(\mathbf{x}) = \frac{\partial^2 f(\mathbf{x})}{\partial x^2} = [\frac{\partial f(\mathbf{x})}{\partial x_i \partial x_j}]$$

Nonlinear least square problems

- The objective function of the nonlinear least square problem can be written as :

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{r}(\mathbf{x})^T\mathbf{r}(\mathbf{x})$$

- Its Jacobian matrix is computed as

$$\frac{\partial f}{\partial x_j} = \sum_{k=1}^{m} r_k(\mathbf{x})\frac{\partial r_k(\mathbf{x})}{\partial x_j}$$

- And it follows that the gradient is the vector

$$\boxed{\nabla f(\mathbf{x}) = J(\mathbf{x})^T\mathbf{r}(\mathbf{x})}$$

$$J(\mathbf{x}) = \frac{\partial \mathbf{r}(\mathbf{x})}{\partial \mathbf{x}}$$

Nonlinear least square problems

- The **Hessian matrix** of the objective function is computed as

$$\nabla^2 f(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}} \nabla f(\mathbf{x})$$

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} = \sum_{k=1}^{m} \frac{\partial r_k(\mathbf{x})}{\partial x_i} \frac{\partial r_k(\mathbf{x})}{\partial x_j} + \sum_{k=1}^{m} \frac{\partial^2 r_k(\mathbf{x})}{\partial x_i \partial x_j}$$

$$\boxed{\nabla^2 f(\mathbf{x}) = J(\mathbf{x})^T J(\mathbf{x}) + \sum_{k=1}^{m} r_k(\mathbf{x}) \nabla^2 r_k(\mathbf{x})}$$

When $\mathbf{r}(\mathbf{x}) \to \mathbf{0}$, $\mathcal{H}(\mathbf{x}) \approx J(\mathbf{x})^T J(\mathbf{x})$

Newton's method

- Iteratively update the variable, $\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}$, until the objective function $f(\mathbf{x} + \Delta\mathbf{x})$ does not decrease.
- At each step, it tries to find a step $\Delta\mathbf{x}$ that leads to a local minimum value :

$$\frac{\partial f(\mathbf{x}+\Delta\mathbf{x})}{\partial \Delta\mathbf{x}} = 0$$

- From second-order Taylor expansion, we have

$$\frac{\partial}{\partial \Delta\mathbf{x}}[f(\mathbf{x}) + \nabla f(\mathbf{x})^T \Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T \mathcal{H}(\mathbf{x})\Delta\mathbf{x}]$$

$$\boxed{\nabla f(\mathbf{x}) + \mathcal{H}(\mathbf{x})\Delta\mathbf{x} = 0}$$

Newton's method

- Since we have $\nabla f(\mathbf{x}) = J(\mathbf{x})^T \mathbf{r}(\mathbf{x})$ and

$$\mathcal{H}(\mathbf{x}) = \nabla^2 f(\mathbf{x}) = J(\mathbf{x})^T J(\mathbf{x}) + \boxed{\sum_{k=1}^{m} r_k(\mathbf{x}) \nabla^2 r_k(\mathbf{x})}$$
$$= J(\mathbf{x})^T J(\mathbf{x}) + S(\mathbf{x})$$

- The incremental update is computed as

$$\Delta \mathbf{x} = -(J(\mathbf{x})^T J(\mathbf{x}) + S(\mathbf{x}))^{-1} J^T(\mathbf{x}) \mathbf{r}(\mathbf{x})$$

$$\boxed{= (H(\mathbf{x})^T H(\mathbf{x}) + S(\mathbf{x}))^{-1} H^T(\mathbf{x}) \mathbf{r}(\mathbf{x})}$$

$$\boxed{J(\mathbf{x}) = \frac{\partial \mathbf{r}(\mathbf{x})}{\partial \mathbf{x}} = -\frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}} = -H(\mathbf{x})}$$

Gauss-Newton method

- Use the first-order Taylor expansion to approximate the vector-valued function $\mathbf{h}(\mathbf{x})$ instead of the objective function $f(\mathbf{x})$,

$$\mathbf{h}(\mathbf{x}_0 + \Delta\mathbf{x}) \approx \mathbf{h}(\mathbf{x}_0) + H(\mathbf{x}_0)\Delta\mathbf{x}$$

- So the objective function can be approximated by

$$\begin{aligned} f(\mathbf{x} + \Delta\mathbf{x}) &= \tfrac{1}{2}(\mathbf{y} - \mathbf{h}(\mathbf{x} + \Delta\mathbf{x}))^T(\mathbf{y} - \mathbf{h}(\mathbf{x} + \Delta\mathbf{x})) \\ &\approx \tfrac{1}{2}(\mathbf{r}(\mathbf{x}) - H(\mathbf{x})\Delta\mathbf{x})^T(\mathbf{r}(\mathbf{x}) - H(\mathbf{x})\Delta\mathbf{x}) \end{aligned}$$

$$\frac{\partial}{\partial \Delta\mathbf{x}} f(\mathbf{x} + \Delta\mathbf{x}) = -H^T\mathbf{r} + H^T H \Delta\mathbf{x} = 0$$

$$\boxed{\Delta\mathbf{x} = (H^T H)^{-1} H^T \mathbf{r}(\mathbf{x})}$$

Gauss-Newton method

- Step 1 : Start from an initial point $\mathbf{x}_0$
- Step 2 : solve an incremental step $\Delta\mathbf{x}$,

$$\boxed{\Delta\mathbf{x} = (H^T H)^{-1} H^T \mathbf{r}}$$

- Step 3 : Update the solution $\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}$
- Repeat Step 2~ Step 3 until convergence.

- Practical issues:
  - The initial guess $\mathbf{x}_0$ should be close to the real solution.
  - $(H^T H)$ may be ill-conditioned or not positive-definite.

Levenberg-Marquardt method

- Add a damping term to limit the length of step at each iteration.

$$\Delta\mathbf{x}^{LM} \leftarrow \underbrace{\min f(\mathbf{x}+\Delta x) + \lambda \frac{1}{2}\|\Delta x\|^2}_{f'(\mathbf{x}+\Delta\mathbf{x})}$$

$$\frac{\partial}{\Delta\mathbf{x}} f'(\mathbf{x}+\Delta\mathbf{x}) = 0$$

$$\boxed{\Delta\mathbf{x}^{LM} = (H^T H + \lambda I)^{-1} H^T \mathbf{r}}$$

About the damping parameter

- The damping parameter λ has several effects:

1. it ensures the positive definite of the coefficient matrix $A = (H^T H + \lambda I)$ and avoids bad condition.

2. For large values of λ we get

$$\Delta \mathbf{x}^{LM} \approx H^T \mathbf{r}/\lambda = -\nabla f/\lambda$$

It is a short step in the steepest descent direction.

3. If λ is small, the step is close to the one obtained Gauss-Newton method, which is good in the final stages of iteration, which $\mathbf{x} \rightarrow \mathbf{x}^*$

Choose the damping parameter

- The incremental of the objective function predicted by the linear model is given by

$$l(\Delta\mathbf{x}) = (\mathbf{r} - H\Delta\mathbf{x})^T(\mathbf{r} - H\Delta\mathbf{x}) - \mathbf{r}^T\mathbf{r}$$

$$= \Delta\mathbf{x}^T H^T H \Delta\mathbf{x} - 2\Delta\mathbf{x}^T H^T \mathbf{r}$$

- The incremental predicted by the LM step is computed as

$$l(\Delta\mathbf{x}^{LM}) = (\Delta\mathbf{x}^{LM})^T (H^T H \Delta\mathbf{x}^{LM} - 2H^T\mathbf{r})$$

$$\boxed{\Delta\mathbf{x}^{LM} = (H^T H + \lambda I)^{-1} H^T \mathbf{r}}$$

$$= (\Delta\mathbf{x}^{LM})^T (\lambda\Delta\mathbf{x}^{LM} + H^T\mathbf{r})$$

Choose the damping parameter

- The updating is controlled by the *gain ratio*

$$\rho = \frac{f(\mathbf{x}+\Delta\mathbf{x}^{LM})-f(\mathbf{x})}{l(\Delta\mathbf{x}^{LM})}$$

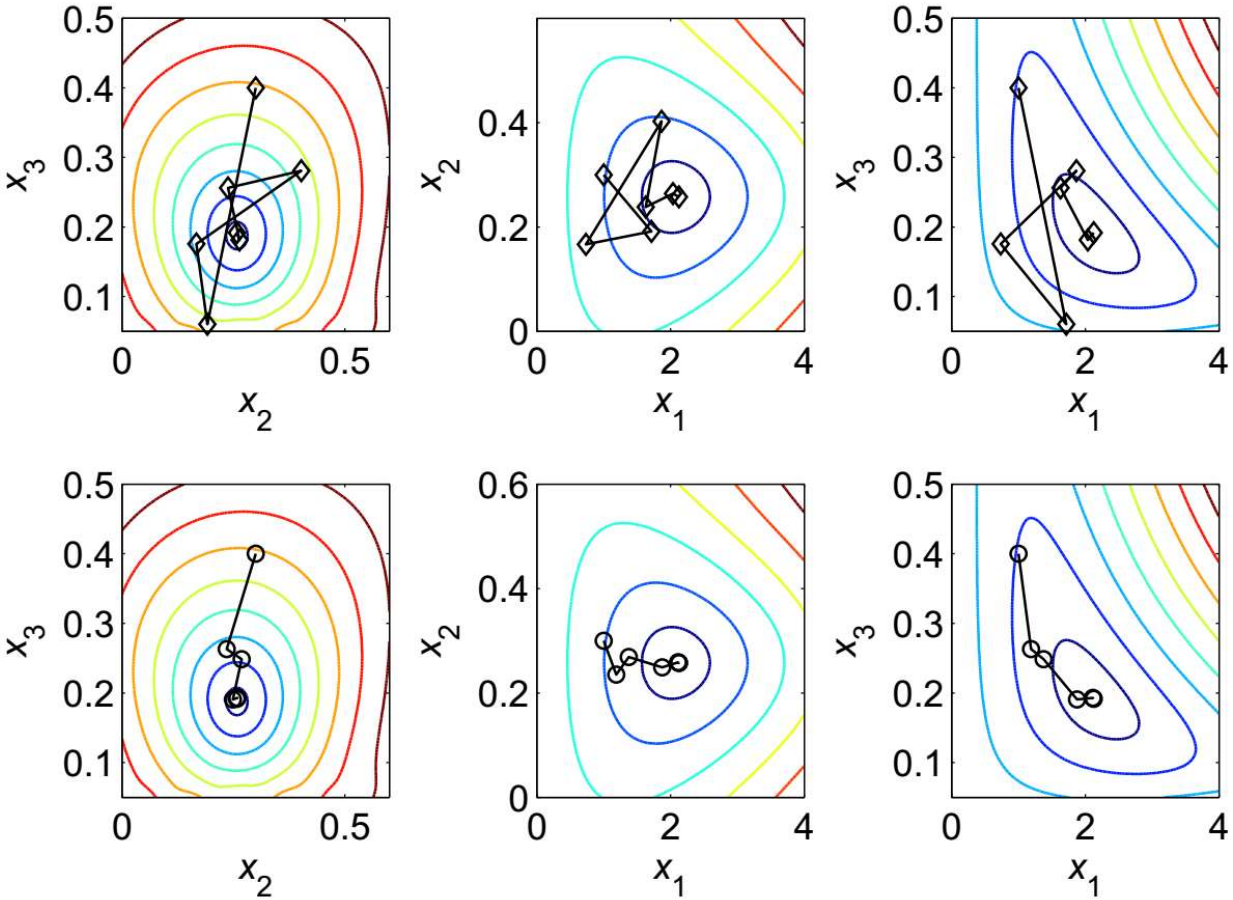
- A large value of gain ratio indicates the current linear model is a good approximation to the objective function. We can decrease λ at the next LM step.
- A small value indicates a poor approximation and we should increase λ to reduce the step length and get closer to the steepest descent direction.

Levenberg-Marquardt algorithm

- Initialization: $A = H^T H$, $\lambda = \max\{a_{ii}\}$
- Repeat until the step length vanishes, $\|\Delta\mathbf{x}^{LM}\| \to 0$, or the gradient of $f(\mathbf{x})$ vanishes, $\nabla f = -H^T\mathbf{r} \to 0$:
  a) Solve $(A + \lambda I)\Delta\mathbf{x} = H^T\mathbf{r}$ to get $\Delta\mathbf{x}^{LM}$
  b) $\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}^{LM}$
  c) Adjust the damping parameter by checking the *gain ratio*
     1. $\rho > 0$: Good approximation, **decrease** the damping parameter
     2. $\rho \leq 0$: Bad approximation, **increase** the damping parameter

Gauss-Newton v.s. Levenberge-Marquardt

- Top row (Gauss-Newton) and Bottom (Levenberg-Marquardt)



Summary

- Nonlinear least square problem :

$$f(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{m} r_i(\mathbf{x})^2 \quad (r_i(\mathbf{x}) = y_i - h_i(\mathbf{x}))$$

- Jacobian matrix : $H(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}}\mathbf{h}(x)$
- Gradient : $\nabla f(\mathbf{x}) = (\frac{\partial}{\partial \mathbf{x}} f(\mathbf{x}))^T$
- Taylor series expansion
- Hessian matrix
- Different solutions:

Newton's method	$\Delta\mathbf{x} = (H^T H + S)^{-1} H^T \mathbf{r}$
Gauss-Newton method	$\Delta\mathbf{x} = (H^T H)^{-1} H^T \mathbf{r}$
Levenberg-Marquardt method	$\Delta\mathbf{x} = (H^T H + \lambda I)^{-1} H^T \mathbf{r}$

Parameter perturbations

- Some times the parameters are not vectors, such as rotations. In such case, the parameters cannot be updated by a simple vector addition :

$$\mathbf{x} \not\leftarrow \mathbf{x} + \Delta\mathbf{x}$$

- We define the operator $\boxplus$ to represent "**adding**" a perturbation to our parameters.
- Consider a parameter perturbation vector $\Delta\mathbf{x} \in \mathbb{R}^n$

$$\boxplus : \mathcal{X} \times \mathbb{R}^n \rightarrow \mathcal{X}, \ (\mathcal{X} - \text{parameter space})$$

Parameter perturbations

- If the parameter space is a vector space, $\mathcal{X} = \mathbf{R}^n$,

$$\mathbf{x} \boxplus \Delta\mathbf{x} = \mathbf{x} + \Delta\mathbf{x}$$

- If the parameter is a Lie group (like rotation).

$$\mathbf{x} \boxplus \Delta\mathbf{x} = \mathbf{x} \otimes \exp(\Delta\mathbf{x}^\wedge) \quad \text{(Right multiplication)}$$

$$= \exp(\Delta\mathbf{x}^\wedge) \otimes \mathbf{x} \quad \text{(Left multiplication)}$$

- $\wedge : \mathbb{R}^n \rightarrow so(n)$ is an one-to-one mapping from the perturbation vector space to *lie algebra*.

A few words on group/algebra

- Both group and algebra are concepts from abstract algebra. They are both algebraic structures.
- A **algebraic structure** is a set with one or more finitary **operations** defined on it that satisfies a list of axioms.
- Examples:
  - Matrices : matrix group
  - 3D Rotation matrices : SO(3) Lie group

$$R^T R = I, \det(R) = 1$$

  - 3x3 Skew-symmetric matrices : so(3) Lie algebra

$$A = -A^T \in \mathbb{R}^{3\times 3} \qquad A = \begin{bmatrix} 0 & a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_2 & 0 \end{bmatrix} = [\mathbf{a}]_\times$$

Rotation perturbations

- When the parameter is represented by a rotation matrix :

$$\mathbf{x} \boxplus \Delta\mathbf{x} \leftrightarrow R \cdot \delta R$$

- Matrix exponential : $A \in \mathbb{R}^{n \times n}$

$$\exp(A) = I + A + \frac{1}{2!}A^2 + \frac{1}{3!}A^3 + \dots$$

- The incremental rotation is computed from the perturbation vector by :

$$\begin{aligned} \delta R &= \exp([\Delta\mathbf{x}]_\times) = I + [\Delta\mathbf{x}]_\times + [\Delta\mathbf{x}]_\times [\Delta\mathbf{x}]_\times + \dots \\ &\approx I + [\Delta\mathbf{x}]_\times \end{aligned}$$

$$\boxed{\mathbf{x} \boxplus \Delta\mathbf{x} \approx R(I + [\Delta\mathbf{x}]_\times)}$$

Rotation perturbations

- When the parameter is represented by a unit quaternion:

$$\mathbf{x} \boxplus \Delta\mathbf{x} \leftrightarrow \mathbf{q} \otimes \delta\mathbf{q}$$

- Definition of a unit quaternion:

$$\mathbf{q} = \begin{bmatrix} \cos(\theta/2) \\ \mathbf{u}\sin(\theta/2) \end{bmatrix} \xrightarrow{\theta \to 0} \mathbf{q} \approx \begin{bmatrix} 1 \\ \mathbf{u}\theta/2 \end{bmatrix}$$

- Let the perturbation vector $\Delta\mathbf{x} \triangleq \mathbf{u}\theta$, we have

$$\mathbf{x} \boxplus \Delta\mathbf{x} \approx \mathbf{q}\{\mathbf{x}\} \otimes \begin{bmatrix} 1 \\ \Delta\mathbf{x}/2 \end{bmatrix}$$

Jacobian matrix with respect to non-vector parameters

- How do we compute the Jacobian matrix if the parameter is not a vector?
- We need to compute $\frac{\partial}{\partial \Delta \mathbf{x}} \mathbf{h}(x)$ instead of $\frac{\partial}{\partial \mathbf{x}} \mathbf{h}(x)$ because we want to establish the relationship between the perturbation and the changed value.

$$H(\mathbf{x}) = \frac{\partial}{\partial \Delta \mathbf{x}} \mathbf{h}(x)$$

- The change of the function value after a small perturbation is described by

$$\frac{\partial}{\partial \Delta \mathbf{x}} \mathbf{h} = \frac{\mathbf{h}(\mathbf{x} \boxplus \Delta \mathbf{x}) - \mathbf{h}(\mathbf{x})}{\Delta \mathbf{x}} \Big|_{\Delta \mathbf{x} \to 0}$$

Numerical differentiation

- The Jacobian matrix can be evaluated by numerical differentiation if the analytic approach is too complicated.
- At each time, the perturbation is only enabled in a single dimension.

$$\Delta \mathbf{x}^{(j)} = \left[0 \cdots \delta \cdots 0\right]^T \in \mathbb{R}^n$$

$$\delta \text{ is a small value: } \max(|10^{-4} x_i|, 10^{-6})$$

$$\boxed{H(:,j) \approx \frac{\mathbf{h}(\mathbf{x} \boxplus \Delta \mathbf{x}^{(j)}) - \mathbf{h}(\mathbf{x})}{\delta}}$$

Summary

- Non-vector parameters
- Perturbation operator $\boxplus : \mathcal{X} \times \mathbb{R}^n \to \mathcal{X},$
- Rotation perturbation :

$$\mathbf{x} \boxplus \Delta\mathbf{x} \leftrightarrow R \cdot \delta R \qquad \mathbf{x} \boxplus \Delta\mathbf{x} \approx R(I + [\Delta\mathbf{x}]_\times)$$

$$\mathbf{x} \boxplus \Delta\mathbf{x} \leftrightarrow \mathbf{q} \otimes \delta\mathbf{q} \qquad \mathbf{x} \boxplus \Delta\mathbf{x} \approx \mathbf{q}\{\mathbf{x}\} \otimes \begin{bmatrix} 1 \\ \Delta\mathbf{x}/2 \end{bmatrix}$$

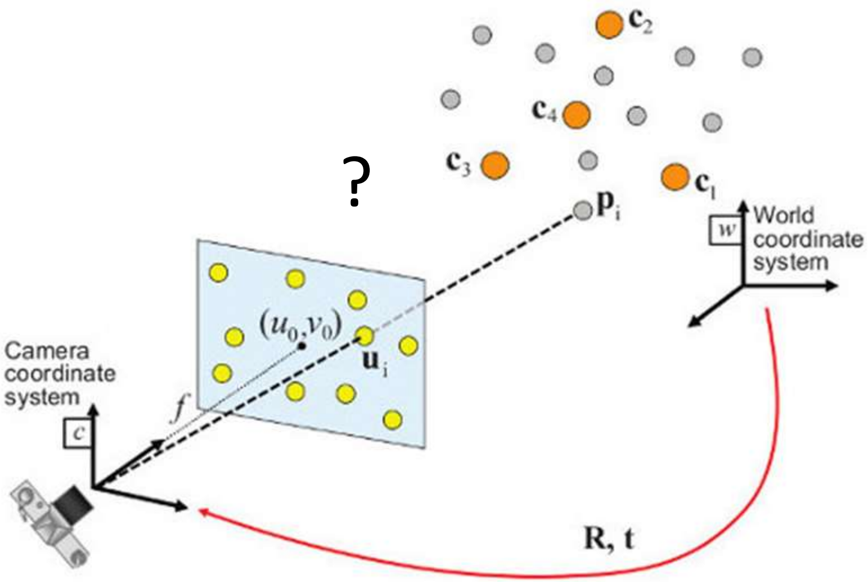
- Jacobian matrix with respect to non-vector parameters:

$$\frac{\partial}{\partial \Delta\mathbf{x}}\mathbf{h} = \frac{\mathbf{h}(\mathbf{x} \boxplus \Delta\mathbf{x}) - \mathbf{h}(\mathbf{x})}{\Delta\mathbf{x}} \Big|_{\Delta\mathbf{x} \to 0}$$

- Numerical differentiation

RANdom SAmple Consensus(RANSAC)

- What if there exists outliers in the data ?
  - Feature matching could not be always correct.



RANSAC = **Hypothesis** and **Verification**

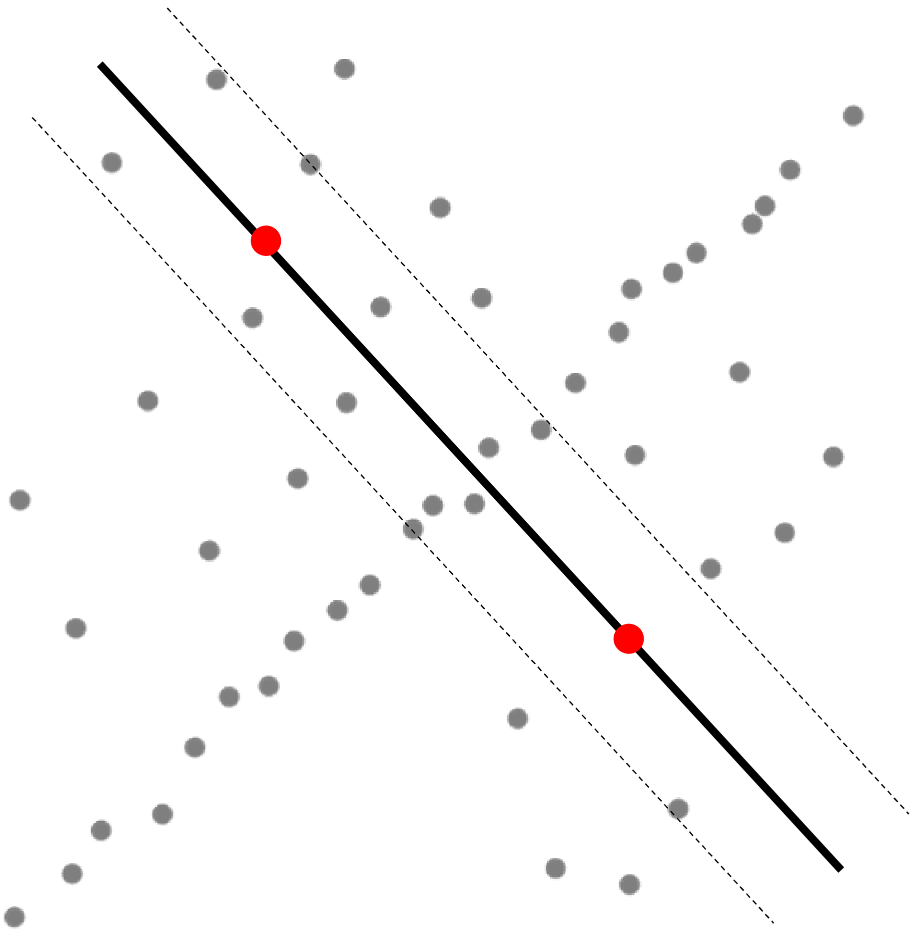
RANdom **SA**mple **C**onsensus(RANSAC)

- Repeat the following steps for N times
  - **Hypothesis**
    - Randomly select the minimum number of data required to determine the model parameter
    - Solve the model parameters
  - **Verification**
    - Determine the inliers (consensus set) that fit with the solved model.
    - The consensus set with the largest number of elements is recorded.

- The model is finally refined using the elements within the consensus set.

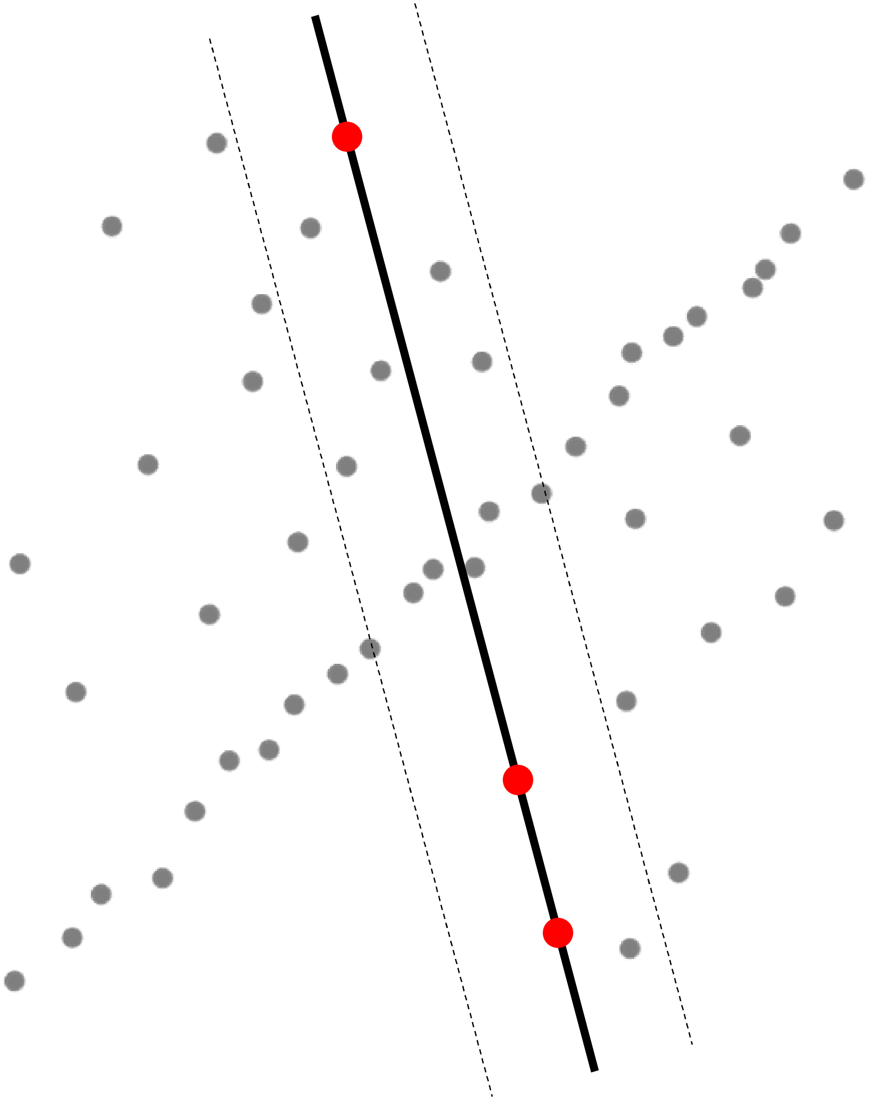
RANSAC

- Select m samples randomly
- Estimate the model from the sampled points
- Find the consensus set (inliers)



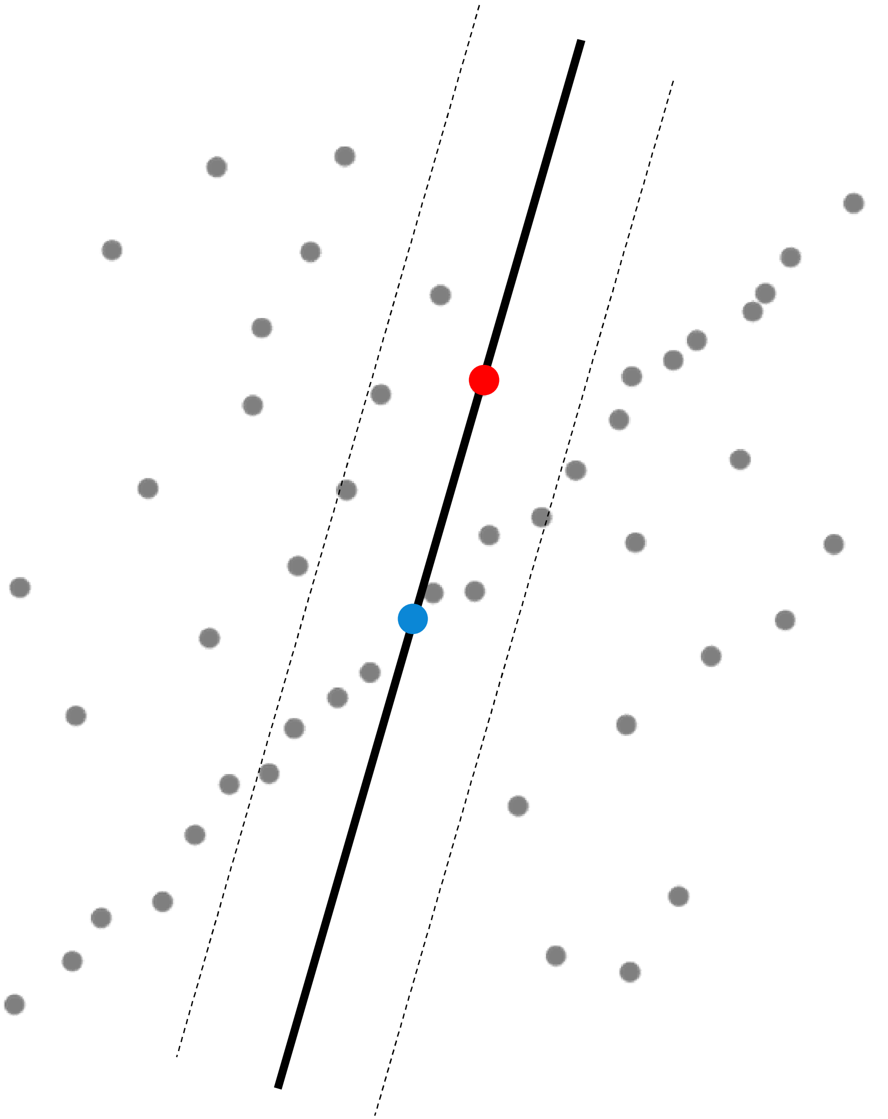
RANSAC

- Select m samples randomly
- Estimate the model from the sampled points
- Find the consensus set (inliers)
- **Repeat sampling**



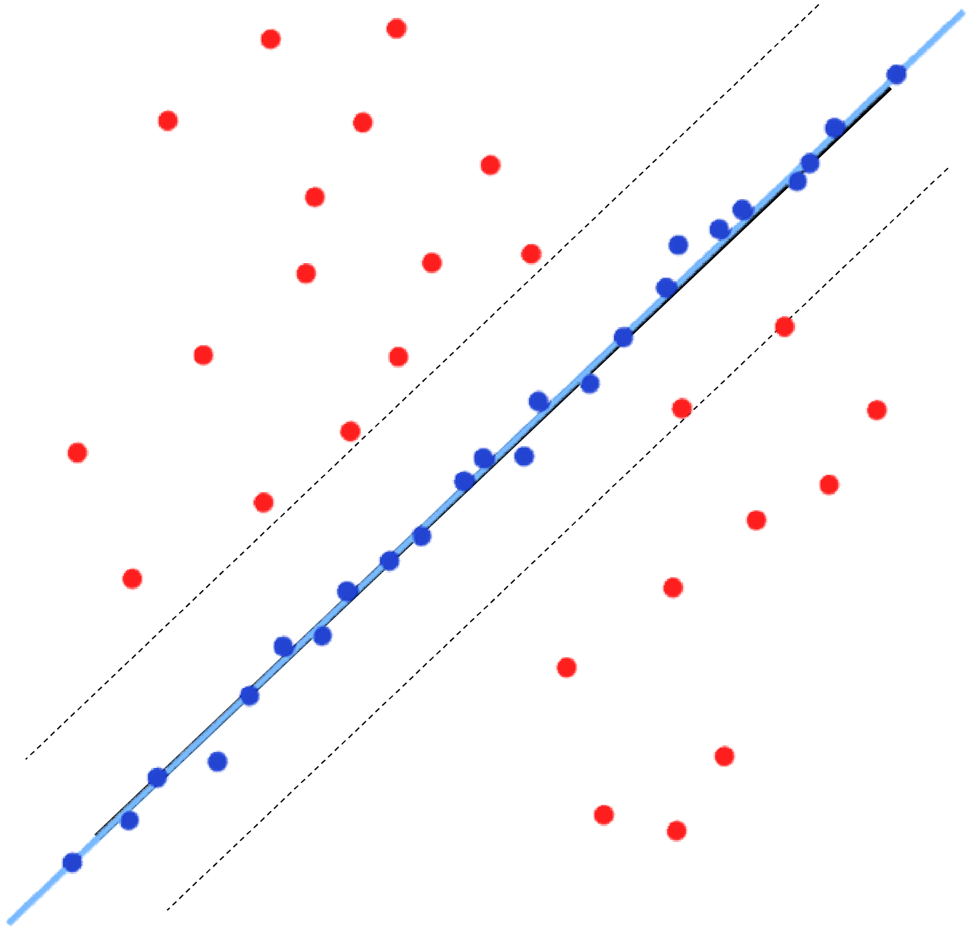
RANSAC

- Select m samples randomly
- Estimate the model from the sampled points
- Find the consensus set (inliers)
- **Repeat sampling**



RANSAC

- Select m samples randomly
- Estimate the model from the sampled points
- Find the consensus set (inliers)
- Repeat sampling
- **A best model is the one with maximum number inliers**



RANSAC

- How many times do we need to do sampling to get a noisy-free subset ?

RANSAC

- Let p be the probability of getting a noisy-free subset. What we want maybe

$$p > 99\%$$

- Let the inlier ratio of the data be :

$$u = \frac{\#\text{noisy-free data points}}{\#\text{total points}}$$

RANSAC

- Repeat sampling until noisy-free model has been sampled…
  - #1 => contain noisy points $(1-u^M)$
  - #2 => contain noisy points $(1-u^M)$
  - … … ……
  - #N => contain noisy points $(1-u^M)$
  - #N+1=> only noisy free points
- The probability of getting N times noisy model is

$$(1-u^M)^N$$

- The probability of getting a noisy-free model after N times sampling is :

$$p = 1 - (1-u^M)^N$$

RANSAC

- Therefore, the number of sampling required is

$$N = \frac{\log(1-p)}{\log(1-u^M)}$$

- Example:
  - $p = 0.99, m = 3, u = 0.5 \rightarrow N \approx 35$
  - $p = 0.95, m = 3, u = 0.5 \rightarrow N \approx 22$
  - $p = 0.99, m = 3, u = 0.8 \rightarrow N \approx 6$
  - $p = 0.99, m = 5, u = 0.8 \rightarrow N \approx 12$

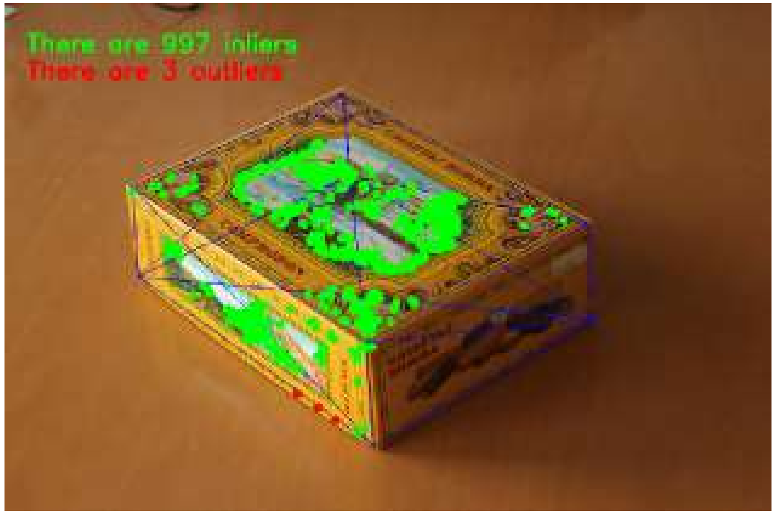
Software for pose estimation

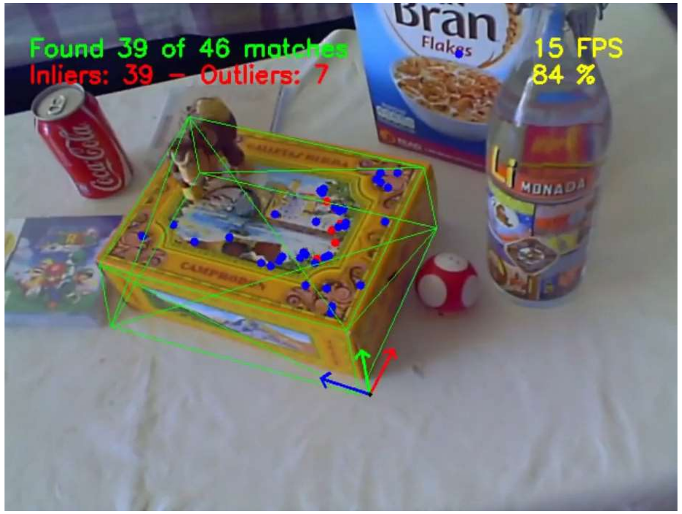
- OpenCV
  - Calib3d. Camera Calibration and 3D Reconstruction

```
void solvePnPRansac(InputArray objectPoints,     // 3D points (Coordinates are known)
                    InputArray imagePoints,      // 2D image points
                    InputArray cameraMatrix,     //K – camera intrinsic matrix
                    InputArray distCoeffs,       //distortion coefficients
                    OutputArray rvec,            //rotation vector (can be converted into
                                                 // rotation matrix)
                    OutputArray tvec,            //translation vector
                    bool useExtrinsicGuess=false,
                    int iterationsCount=100,
                    float reprojectionError=8.0, //for RANSAC
                    int minInliersCount=100,     //for inliers
                    OutputArray inliers=noArray(),
                    int flags=ITERATIVE)
```

A tutorial about pose estimation

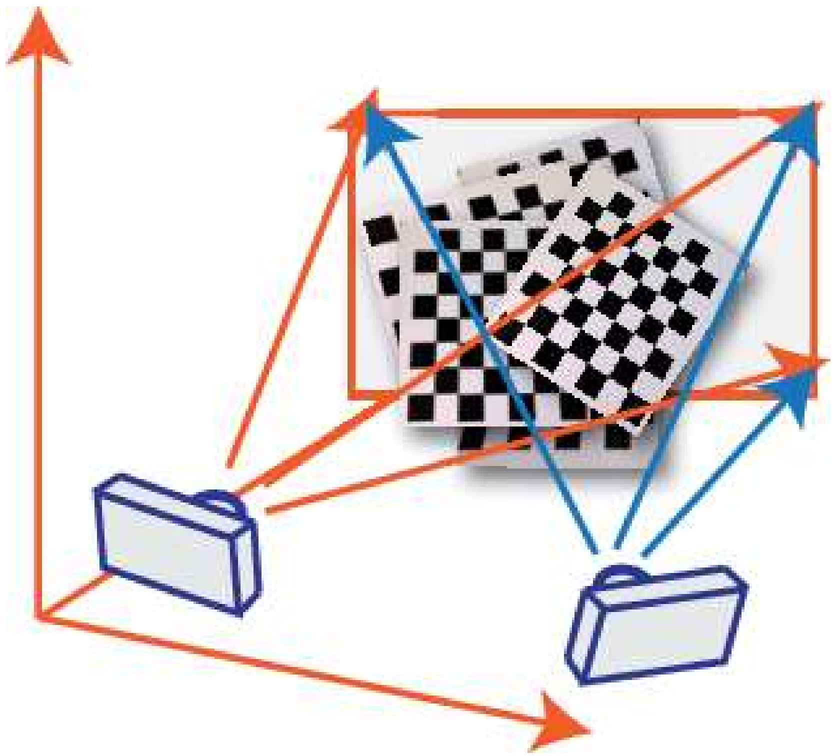
- https://docs.opencv.org/master/dc/d2c/tutorial_real_time_pose.html



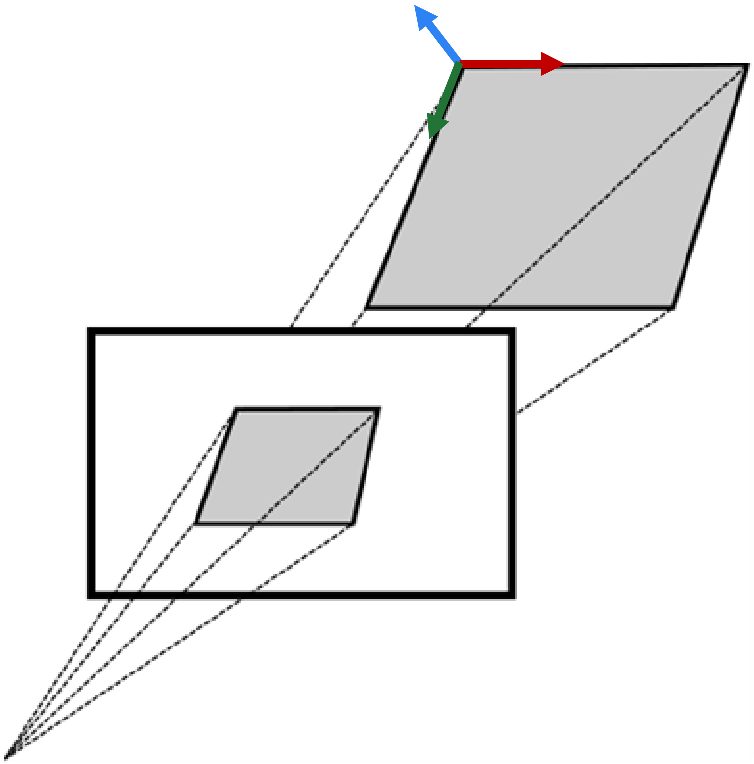


Pose estimation of planar objects

- What about if the object is a planar object ? Like the checker board pattern for camera calibration ?



Pose estimation of a planar object



$$\mathbf{x} \sim \mathbf{K}[\mathbf{R}\ \mathbf{t}]\mathbf{X} \qquad \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix}$$

$$= \mathbf{K}[\mathbf{r}_1\ \mathbf{r}_2\ \mathbf{r}_3\ \mathbf{t}]$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \mathbf{K}[\mathbf{r}_1\ \mathbf{r}_2\ \mathbf{t}] \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

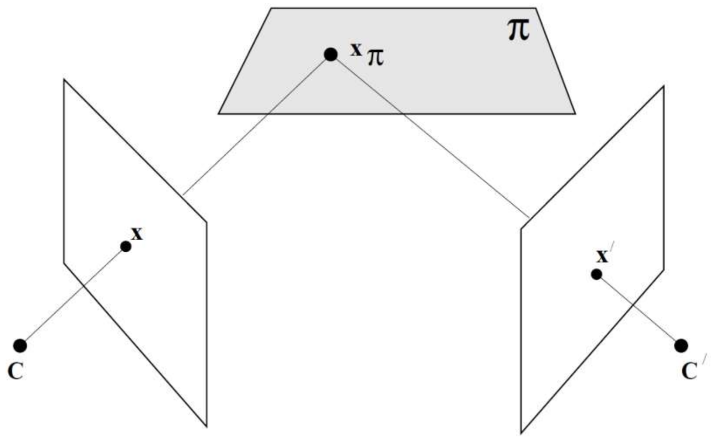
$$\boxed{\mathbf{x} \sim \mathbf{H}\tilde{\mathbf{X}}}$$

Homogenous transform

- Homogenous transform (Homography) is a 3x3 non singular matrix

$$H = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}$$

- Homography relates two images of a planar object in 3D space together.



$$\left\{\begin{aligned} \mathbf{x}_\pi &= (x_1, x_2, x_3, x_4)^T \\ \mathbf{x} &= [I \ \mathbf{0}]\mathbf{x}_\pi \\ \mathbf{x}' &= [R \ t]\mathbf{x}_\pi \\ \pi &= (\mathbf{n}, d) \\ \pi^T \mathbf{x}_\pi &= 0 \end{aligned}\right.$$

$$\mathbf{x}' \overset{?}{\sim} \mathbf{x}$$

Homogenous transform

- $\mathbf{x}' = [R\ t]\mathbf{X}_\pi$

$$\Rightarrow \quad \mathbf{x}' = R\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + x_4 t$$

$$\Rightarrow \quad \mathbf{x}' = R\mathbf{x} + x_4 t \qquad (\mathbf{x} = [I\ \mathbf{0}]\mathbf{X}_\pi)$$

$$\Rightarrow \quad \mathbf{x}' = R\mathbf{x} - \frac{\mathbf{n}^T\mathbf{x}}{d}t \qquad (\pi^T\mathbf{x}_\pi = 0 \text{ and } \pi \text{ is not a infinite plane})$$

$$\Rightarrow \quad \mathbf{x}' = R\mathbf{x} - \frac{\mathbf{n}^T t}{d}\mathbf{x}$$

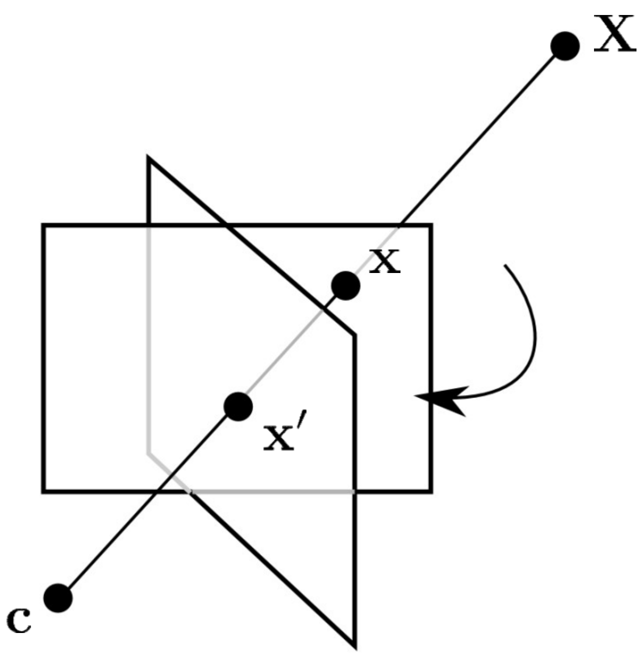
$$\Rightarrow \quad \mathbf{x}' = \boxed{(R - \frac{\mathbf{n}^T t}{d})}\mathbf{x}$$

$$\boxed{\mathbf{x}' = H\mathbf{x}}$$

We can also extract R, t from homography if the plane equation is known.

Homogenous transform

- If the camera purely rotates, for any points(not necessary in the same plane), all their images between two frames are related by a Homography.



$$\mathbf{x} = [I \ \mathbf{0}]\mathbf{X}$$

$$\mathbf{x}' = [R \ \mathbf{0}]\mathbf{X}$$

$$\mathbf{x}' = R\mathbf{x}$$

Compute homogenous transform

- Homography matrix has eight degree of freedom.
- At least four points are required to compute the Homography.

Let $\mathbf{H} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & 1 \end{bmatrix}$, for each pair of

correspondence : $(x_i, y_i, 1) \leftrightarrow (x'_i, y'_i, 1)$,we have

$$x'_i = \frac{h_1 x_i + h_2 y_i + h_3}{h_7 x_i + h_8 y_i + 1} \qquad y'_i = \frac{h_4 x_i + h_5 y_i + h_6}{h_7 x_i + h_8 y_i + 1}$$

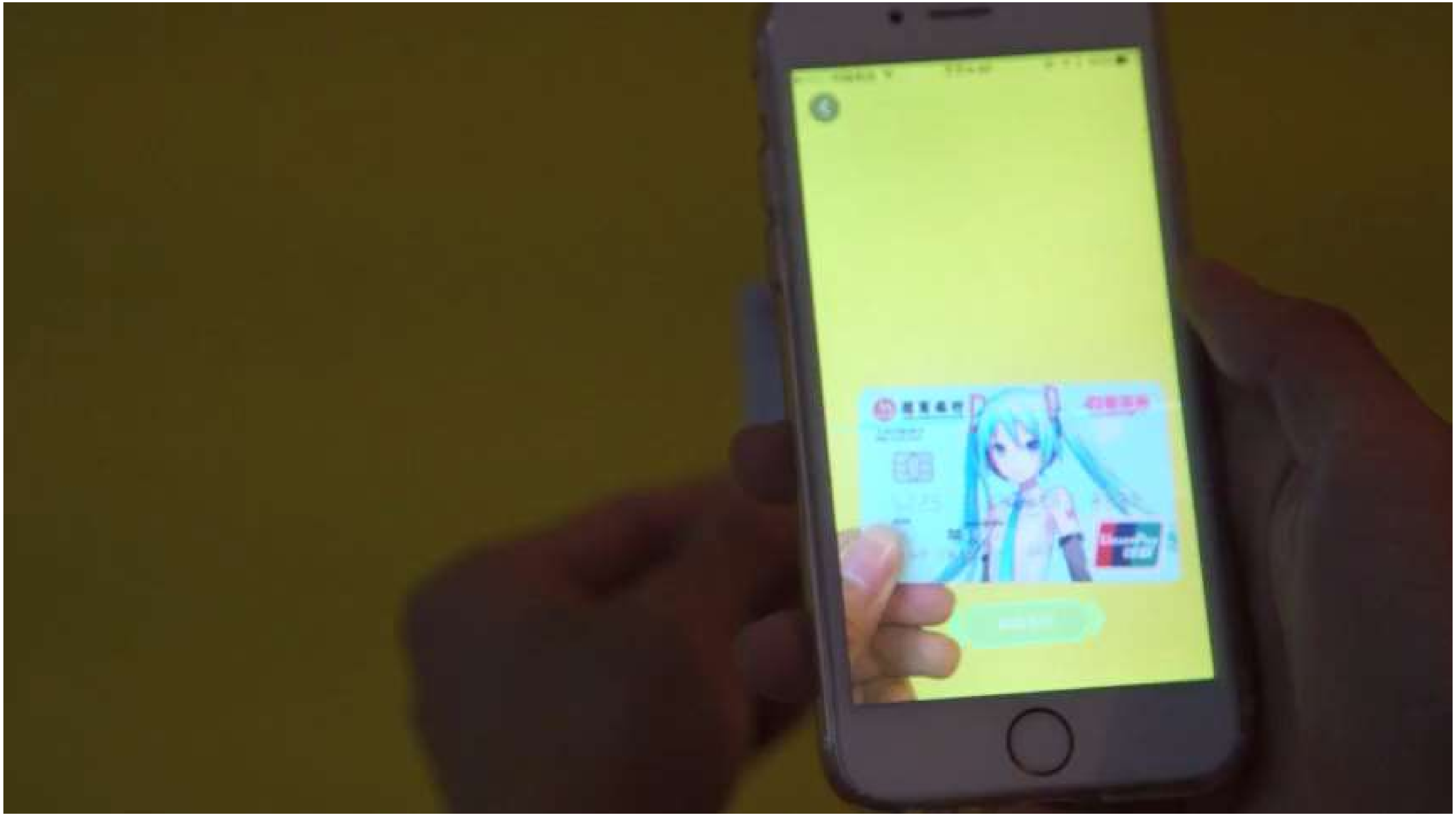
$$\begin{array}{c} \cdots\cdots \\ \begin{pmatrix} x_i & y_i & 1 & 0 & 0 & 0 & x'_i x_i & x'_i y_i \\ 0 & 0 & 0 & x_i & y_i & 1 & y'_i x_i & y'_i y_i \end{pmatrix} \\ \cdots\cdots \end{array} \mathbf{h} = \begin{array}{c} \cdots \\ \begin{pmatrix} -x'_i \\ -y'_i \end{pmatrix} \\ \cdots \end{array}$$

$$\mathbf{h} = (h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8)^T$$

Applications

- Augmented Reality

Applications



Applications

Applications