



Practical Course: Vision-based Navigation SS 2018

Lecture 3. State Estimation

Dr. Jörg Stückler, Dr. Xiang Gao
Vladyslav Usenko, Prof. Dr. Daniel Cremers



Contents

- From state estimation to least square
- Batch least square
- Application: estimate camera pose by iterative method

Contents

- From state estimation to least square
- Batch least square
- Application: estimate camera pose by iterative method

1. From state estimation to least square

- Recall the motion model and observation model

$$\begin{cases} \mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k, \mathbf{w}_k) \\ \mathbf{z}_{k,j} = h(\mathbf{y}_j, \mathbf{x}_k, \mathbf{v}_{k,j}) \end{cases} .$$

- How to estimate the unknown variables given the observation data?

1. Batch state estimation

- Batch approach
 - Give all the measurements
 - To estimate all the state variables
- State variables:

$$\mathbf{x} = \{\mathbf{x}_1, \dots, \mathbf{x}_N, \mathbf{y}_1, \dots, \mathbf{y}_M\}.$$

Observation and input:

$$\mathbf{u} = \{u_1, u_2, \dots\}, \mathbf{z} = \{z_{k,j}\}$$

- Our purpose:

$$P(\mathbf{x}|\mathbf{z}, \mathbf{u}).$$

- Bayes' Rule:

$$p(\mathbf{x}|\mathbf{u}, \mathbf{z}) = \frac{P(\mathbf{z}|\mathbf{x}, \mathbf{u})p(\mathbf{x}|\mathbf{u})}{P(\mathbf{z}|\mathbf{u})}$$

Likelihood

Priori

Posteriori

1. From state estimation to least square

- It is usually hard to write out the full distribution of Bayes' formula, but we can:
- MAP: Maximum A Posteriori

$$\begin{aligned}x_{MAP} &= \operatorname{argmax}_x P(x|u, z) = \operatorname{argmax}_x \frac{P(z|x, u)P(x|u)}{P(z|u)} \\ &= \operatorname{argmax}_x P(z|x)P(x|u)\end{aligned}$$

Drop u because z is not relevant with u

Drop denominator because it is not relevant with x

- “In which state it is most likely to produce such measurements”

1. From state estimation to least square

- From MAP to batch least square
- We assume the noise variables are independent, so that the joint pdf can be factorized:

$$P(z|x) = \prod_{k=0}^K P(z_k|x_k)$$

- Let's consider a single observation:
 - Affected by white Gaussian noise:

$$z_{k,j} = h(\mathbf{y}_j, \mathbf{x}_k) + \mathbf{v}_{k,j},$$

$$\mathbf{v}_{k,j} \sim N(0, \mathbf{Q}_{k,j})$$

- The observation model gives us a conditional pdf:

$$P(z_{j,k} | \mathbf{x}_k, \mathbf{y}_j) = N(h(\mathbf{y}_j, \mathbf{x}_k), \mathbf{Q}_{k,j}).$$

- Then how to compute the MAP of \mathbf{x}, \mathbf{y} given \mathbf{z} ?

1. From state estimation to least square

- Gaussian distribution (matrix form)

$$P(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^N \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right).$$

- Take minus logarithm at both sides:

$$-\ln(P(\mathbf{x})) = \frac{1}{2} \ln\left((2\pi)^N \det(\boldsymbol{\Sigma})\right) + \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}).$$

Constant w.r.t \mathbf{x}

Mahalanobis distance (sigma-norm)

- Maximum of $P(\mathbf{x})$ is equivalent to minimum of $-\ln(P(\mathbf{x}))$

1. From state estimation to least square

- Take this into the MAP:

$$\text{Max: } P(\mathbf{z}_{j,k} | \mathbf{x}_k, \mathbf{y}_j) = N(h(\mathbf{y}_j, \mathbf{x}_k), \mathbf{Q}_{k,j}).$$

Information matrix

$$\rightarrow \mathbf{x}_k, \mathbf{y}_j = \operatorname{argmin} \left(\left(\mathbf{z}_{k,j} - h(\mathbf{y}_j, \mathbf{x}_k) \right)^T \mathbf{Q}_{j,k}^{-1} \left(\mathbf{z}_{k,j} - h(\mathbf{y}_j, \mathbf{x}_k) \right) \right)$$

Error or residual of single observation

- We turn a MAP problem into a least square problem

1. From state estimation to least square

- Batch least square
- Original problem

$$\begin{cases} \mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k, \mathbf{w}_k) \\ \mathbf{z}_{k,j} = h(\mathbf{y}_j, \mathbf{x}_k, \mathbf{v}_{k,j}) \end{cases}.$$

$$\mathbf{x}_{MAP} = \operatorname{argmax} P(\mathbf{z}|\mathbf{x})P(\mathbf{x}|\mathbf{u})$$

Least square
Define the errors(residuals)

$$\begin{aligned} \mathbf{e}_{v,k} &= \mathbf{x}_k - f(\mathbf{x}_{k-1}, \mathbf{u}_k) \\ \mathbf{e}_{y,j,k} &= \mathbf{z}_{k,j} - h(\mathbf{x}_k, \mathbf{y}_j), \end{aligned}$$

- Sum of the squared residuals:

$$\min J(\mathbf{x}) = \sum_k \mathbf{e}_{v,k}^T \mathbf{R}_k^{-1} \mathbf{e}_{v,k} + \sum_k \sum_j \mathbf{e}_{y,k,j}^T \mathbf{Q}_{k,j}^{-1} \mathbf{e}_{y,k,j}.$$

1. From state estimation to least square

$$J(\mathbf{x}) = \sum_k e_{v,k}^T \mathbf{R}_k^{-1} e_{v,k} + \sum_k \sum_j e_{y,k,j}^T \mathbf{Q}_{k,j}^{-1} e_{y,k,j}.$$

- Some notes:
 - Because of noise, when we take the estimated trajectory and map into the models, they won't fit perfectly
 - Then we adjust our estimation to get a better estimation (minimize the error)
 - The error distribution is affected by noise distribution (information matrix)
- Structure of the least square problem
 - Sum of many squared errors
 - The dimension of total state variable maybe high
 - But single error item is easy (only related to two states in our case)
 - If we use Lie group and Lie algebra, then it's a non-constrained least square

Contents

- From state estimation to least square
- Batch least square
- Application: estimate camera pose by iterative method

2. Batch least square

- How to solve a least square problem?
 - Non-linear, discrete time, non-constrained
- Let's start from a simple example

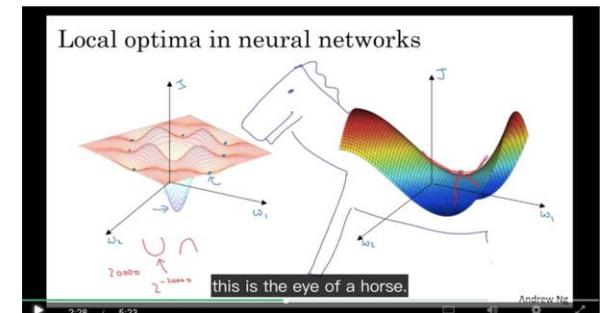
- Consider minimizing a squared error:
- When J is simple, just solve:

$$\frac{dJ}{dx} = 0$$

$$\min J(x) = \min \frac{1}{2} \|f(x)\|_2^2$$

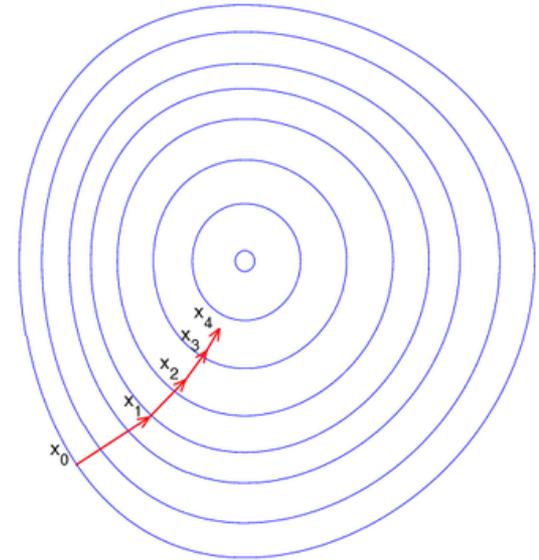
$$x \in \mathbb{R}^n$$

- And we will find the maxima/minima/saddle points



2. Batch least square

- When J is a complicated function:
 - $dJ/dx=0$ is hard to solve
 - We use **iterative methods**
- Iterative methods
 1. Start from a initial estimation x_0
 2. At iteration k , we find a incremental Δx_k to make $\|f(x_k + \Delta x_k)\|_2^2$ become smaller
 3. If Δx_k is small enough, stop (converged)
 4. If not, set $x_{k+1} = x_k + \Delta x_k$ and return to step 2.



2. Batch least square

- How to find the incremental part?
- By the gradient
- Taylor expansion of the object function:

$$\|f(\mathbf{x} + \Delta\mathbf{x})\|_2^2 \approx \|f(\mathbf{x})\|_2^2 + \underbrace{\mathbf{J}(\mathbf{x}) \Delta\mathbf{x}}_{\text{Jacobian}} + \frac{1}{2} \underbrace{\Delta\mathbf{x}^T \mathbf{H} \Delta\mathbf{x}}_{\text{Hessian}}.$$

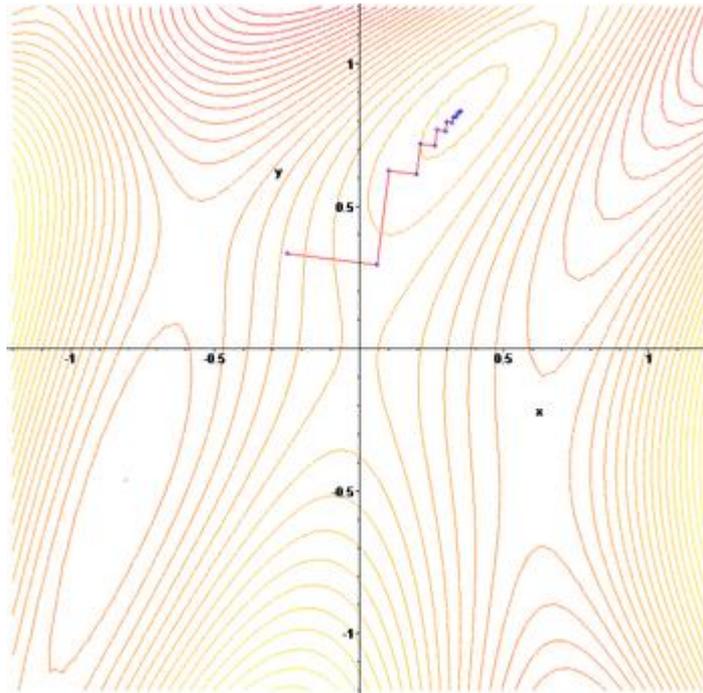
- First order methods and second order methods
- First order: (Steepest descent)

$$\min_{\Delta\mathbf{x}} \|f(\mathbf{x})\|_2^2 + J\Delta\mathbf{x} \quad \text{Incremental will be:} \quad \Delta\mathbf{x}^* = -\mathbf{J}^T(\mathbf{x}).$$

Usually we need a step size

2. Batch least square

- Zig-zag in steepest descent



Other shortcomings

- Slow convergence speed
- Slow when close to the minimum

2. Batch least square

- Second order methods

$$\|f(\mathbf{x} + \Delta\mathbf{x})\|_2^2 \approx \|f(\mathbf{x})\|_2^2 + \mathbf{J}(\mathbf{x}) \Delta\mathbf{x} + \frac{1}{2} \Delta\mathbf{x}^T \mathbf{H} \Delta\mathbf{x}.$$

- Solve an increment to minimize it:

$$\Delta\mathbf{x}^* = \arg \min \|f(\mathbf{x})\|_2^2 + \mathbf{J}(\mathbf{x}) \Delta\mathbf{x} + \frac{1}{2} \Delta\mathbf{x}^T \mathbf{H} \Delta\mathbf{x}.$$

- Let the derivative to $\Delta\mathbf{x}$ be zero, then we get: $\mathbf{H} \Delta\mathbf{x} = -\mathbf{J}^T.$
- This is called Newton's method

2. Batch least square

- Second order method converges more quickly than first order methods
- But the Hessian matrix maybe hard to compute: $H\Delta x = -J^T$.
- Can we avoid the Hessian matrix and also keeps second order's convergence speed?
 - Gauss-Newton
 - Levenberg-Marquardt

2. Batch least square

- Gauss-Newton

- Taylor expansion of $f(x)$: $f(x + \Delta x) \approx f(x) + J(x) \Delta x$.

- Then the squared error becomes:

$$\begin{aligned} \frac{1}{2} \|f(x) + J(x) \Delta x\|^2 &= \frac{1}{2} (f(x) + J(x) \Delta x)^T (f(x) + J(x) \Delta x) \\ &= \frac{1}{2} \left(\|f(x)\|_2^2 + 2f(x)^T J(x) \Delta x + \Delta x^T J(x)^T J(x) \Delta x \right). \end{aligned}$$

- Also let its derivative with Δx be zero:

$$2J(x)^T f(x) + 2J(x)^T J(x) \Delta x = 0.$$

$$J(x)^T J(x) \Delta x = -J(x)^T f(x).$$

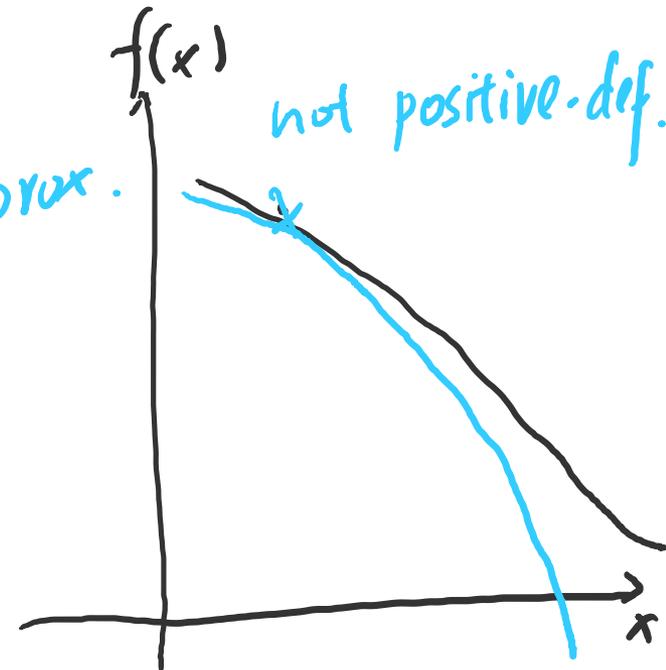
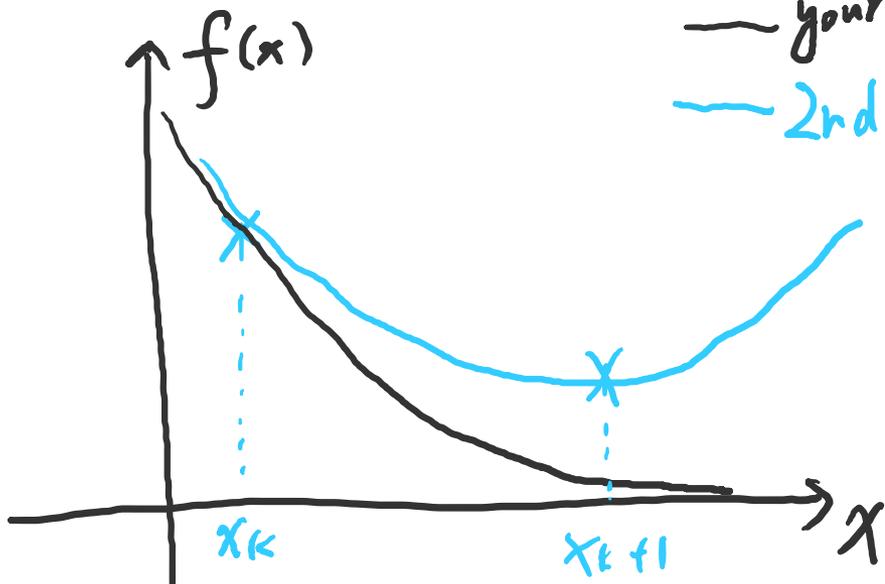
 H  g

$$H \Delta x = g.$$

2. Batch least square

$$J(x)^T J(x) \Delta x = -J(x)^T f(x).$$

- Gauss-Newton use $J(x)^T J(x)$ as an approximation of the Hessian
 - Therefore avoiding the computation of H in the Newton's method
- But $J(x)^T J(x)$ is only semi-positive definite
 - H maybe singular when $J^T J$ has null space



2. Batch least square

- Levenberg-Marquardt method
 - Trust region approach: approximation is only valid in a region
 - Evaluate if the approximation is good:

$$\rho = \frac{f(x + \Delta x) - f(x)}{J(x) \Delta x}.$$

Real descent/approx. descent

- If rho is large, increase the region
 - If rho is small, decrease the region
- LM optimization: $\min_{\Delta x_k} \frac{1}{2} \|f(x_k) + J(x_k)\Delta x_k\|^2, s. t. \|\Delta x_k\|^2 \leq \mu$
 - Assume the approximation is only good within a ball

2. Batch least square

- Trust region problem:

$$\min_{\Delta x_k} \frac{1}{2} \|f(x_k) + J(x_k)\Delta x_k\|^2, \text{ s. t. } \|\Delta x_k\|^2 \leq \mu$$

- Expand it just like in G-N's case, the incremental will be:

$$(J(x_k)^T J(x_k) + \lambda I)\Delta x_k = g \quad \lambda(\|\Delta x_k\|^2 - \mu) = 0$$

- This λI increase the semi-positive definite property of the Hessian
 - Also balancing the first-order and second-order items

2. Batch least square

- Other methods
 - Dog-leg method
 - Conjugate gradient method
 - Quasi-Newton's method
 - Pseudo-Newton's method
 - ...
- You can find more in optimization books if you are interested
- In SLAM, we use G-N or L-M to solve camera's motion, pixel's movement, optical-flow, etc.

2. Batch least square

- Problem in the Practical Assignment
- Curve fitting: find best parameters a, b, c from the observation data:

Curve function: $y = \exp(ax^2 + bx + c) + w$,

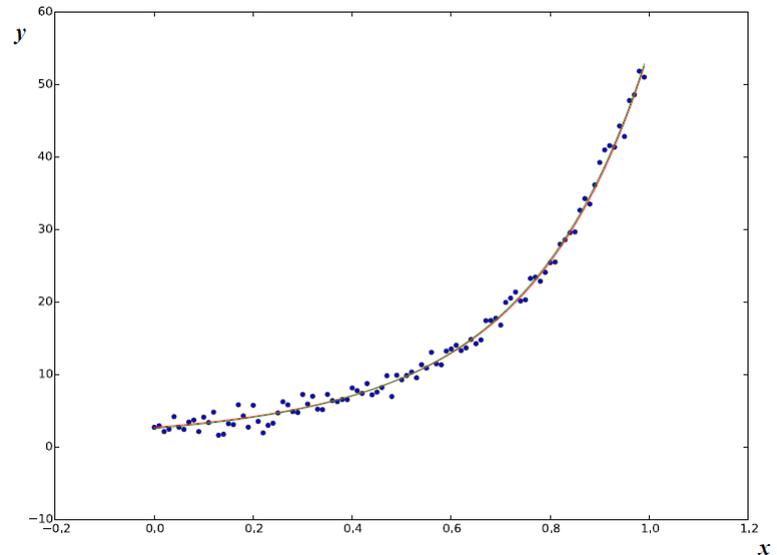
- Error:

$$e_i = y_i - \exp(ax_i^2 + bx_i + c)$$

- Least square problem:

a, b, c

$$= \operatorname{argmin} \sum_{i=1}^N \|y_i - \exp(ax_i^2 + bx_i + c)\|^2$$



2. Batch least square

- You are asked to solve this problem with a hand-written Gauss-Newton's method and use optimization libraries.
- Libraries:
 - Google Ceres Solver <http://ceres-solver.org/>
 - G2O: <https://github.com/RainerKuemmerle/g2o>
- You can choose one of them to implement your estimation

2. Batch least square

- Google Ceres
 - An optimization library for solving least square problems
 - Tutorial: <http://ceres-solver.org/tutorial.html>
 - Define your residual class as a functor (overload the () operator)

```
struct ExponentialResidual {
    ExponentialResidual(double x, double y)
        : x_(x), y_(y) {}

    template <typename T>
    bool operator()(const T* const m, const T* const c, T* residual) const {
        residual[0] = T(y_) - exp(m[0] * T(x_) + c[0]);
        return true;
    }

private:
    // Observations for a sample.
    const double x_;
    const double y_;
};
```

2. Batch least square

- Build the optimization problem:

```
double m = 0.0;
double c = 0.0;

Problem problem;
for (int i = 0; i < kNumObservations; ++i) {
    CostFunction* cost_function =
        new AutoDiffCostFunction<ExponentialResidual, 1, 1, 1>(
            new ExponentialResidual(data[2 * i], data[2 * i + 1]));
    problem.AddResidualBlock(cost_function, NULL, &m, &c);
}
```

- With auto-diff, Ceres will compute the Jacobians for you

2. Batch least square

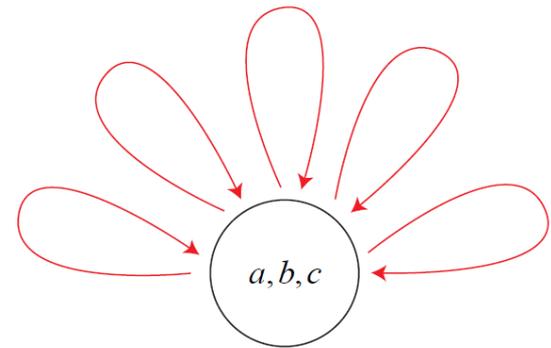
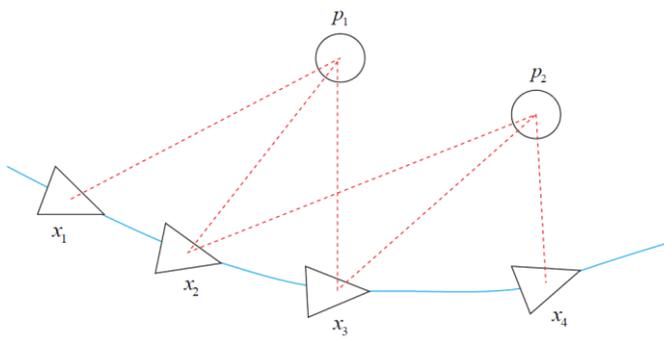
- Finally solve it by calling the `Solve()` function and get the result summary
- You can set some parameters like number of iterations, stop conditions or the linear solver type.

```
Solver::Options options;  
options.max_num_iterations = 25;  
options.linear_solver_type = ceres::DENSE_QR;  
options.minimizer_progress_to_stdout = true;
```

```
Solver::Summary summary;  
Solve(options, &problem, &summary);
```

2. Batch least square

- G2O
 - General Graph Optimization
 - Need to convert the least square problem into a graph
- Graph Optimization
 - State variables are vertices
 - Residuals/Errors are edges connecting those vertices
 - Edges can be unary/binary/multiple



2. Batch least square

- Use g2o to solve your least square problem
 - Define your vertices and edges (or use the built-in vertices and edges in g2o)
 - Build the problem by adding vertices and edges into it
 - Set the optimization parameters (linear solver type, iterations, etc.)
 - Call solve function
 - Fetch the results from the graph

2. Batch least square

- Tutorial of g2o
 - <http://ais.informatik.uni-freiburg.de/publications/papers/kuemmerle11icra.pdf>
 - Doc/ in the github repo: <https://github.com/RainerKuemmerle/g2o>
 - Examples

2. Batch least square

- Summary
 - In the batch estimation, we estimate all the status variable given all the measurements and input
 - The batch estimation problem can be formulated into a least square problem, after solving it we get a MAP estimation
 - The least square problem can be solved by iterative methods like gradient descent, Newton's method, Gauss-Newton or Levenberg-Marquardt method
 - The least square problem can also be represented by a graph and forms a (factor) graph optimization problem

Contents

- From state estimation to least square
- Batch least square
- Application: estimate camera pose by iterative method

3. Application: estimate camera pose

- Suppose we want to estimate the camera pose
- We have several observations from the projection function

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_i = K(RP_i + t) = KTP_i$$

- Minimizing the reprojection error:

$$(R, t)^* = T^* = \operatorname{argmin} \frac{1}{2} \sum_{i=1}^N \|u_i - \pi(RP_i + t)\|_2^2$$

- Where $\pi(\cdot)$ is the projection equation (observation model)

3. Application: estimate camera pose

- Linearize the error: $e_i(x \oplus \Delta x) \approx e_i(x) + J(x)\Delta x$
- Derivative is defined by SE(3) disturb model:

$$\begin{aligned} \frac{\partial e}{\partial T} &= \lim_{\delta\xi \rightarrow 0} \frac{e(\delta\xi \oplus T) - e(T)}{\delta\xi} \\ &= \lim_{\delta\xi \rightarrow 0} \frac{\frac{1}{Z}K(\delta\xi \oplus T)P - \frac{1}{Z}KTP}{\delta\xi} \end{aligned}$$

- Let $P' = TP$ then use chain rule: $\frac{\partial e}{\partial T} = \frac{\partial e}{\partial P'} \frac{\partial P'}{\partial T}$
- For P' we have:

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix}.$$



$$u = f_x \frac{X'}{Z'} + c_x, \quad v = f_y \frac{Y'}{Z'} + c_y.$$

$$\frac{\partial e}{\partial P'} = - \begin{bmatrix} \frac{\partial u}{\partial X'} & \frac{\partial u}{\partial Y'} & \frac{\partial u}{\partial Z'} \\ \frac{\partial v}{\partial X'} & \frac{\partial v}{\partial Y'} & \frac{\partial v}{\partial Z'} \end{bmatrix} = - \begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} \end{bmatrix}.$$

3. Application: estimate camera pose

- The second item: $\frac{\partial(TP')}{\partial T} = \begin{bmatrix} I & -P'^{\wedge} \\ 0^T & 0^T \end{bmatrix}$ See Lecture 2.
- Remove the homogeneous part:

$$\frac{\partial(TP')}{\partial T} = [I \quad -P'^{\wedge}]$$

- Put them together:

$$\frac{\partial e}{\partial T} = - \begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} & -\frac{f_x X' Y'}{Z'^2} & f_x + \frac{f_x X^2}{Z'^2} & -\frac{f_x Y'}{Z'} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} & -f_y - \frac{f_y Y'^2}{Z'^2} & \frac{f_y X' Y'}{Z'^2} & \frac{f_y X'}{Z'} \end{bmatrix}.$$

3. Application: estimate camera pose

- If we want to take the derivative of Point P

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_i = K(RP_i + t) = KTP_i$$

$$\frac{\partial e}{\partial P} = \frac{\partial e}{\partial P'} \frac{\partial P'}{\partial P} = - \begin{bmatrix} f_x/Z' & 0 & -f_x X'/Z'^2 \\ 0 & f_y/Z' & -f_y Y'/Z'^2 \end{bmatrix}^R$$

- P is not relevant to translation t

3. Application: estimate camera pose

- We can also compute these Jacobians in $SO(3)$
- With Jacobian in manifold it will be easy to perform Gauss-Newton iterations to solve the camera's motion iteratively

Any Questions?