Practical Course: Vision-based Navigation
SS 2018

Lecture 5. Backend

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

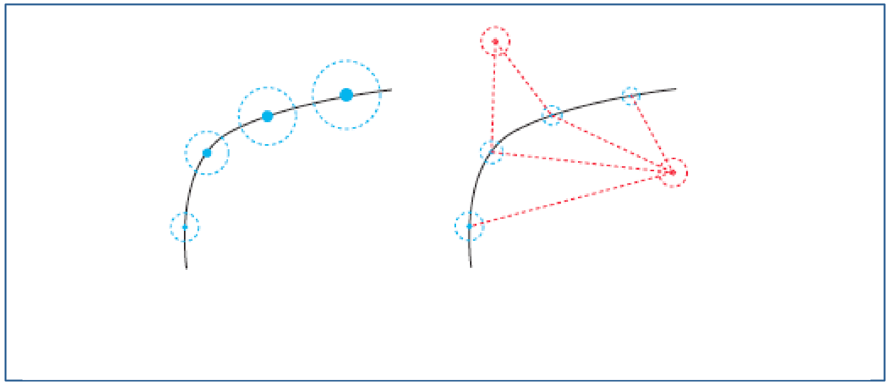
Contents

Contents

1. Recursive Optimization

- Backend
  - Estimate the state variables from the noisy data
- Batch way
  - Estimate the best state given all the data
  - Bundle Adjustment in visual SLAM system
- Incremental way
  - Keep the current (best) estimation, update it when new data is arrived
  - Also throw away the past information
  - Kalman Filter: Linear system + Gaussian noise
  - Extended Kalman Filter: Nonlinear system + Gaussian noise
  - Sliding window filter & multiple state constraint Kalman Filter

1. Recursive Optimization



- A simple example
- When we walk blindfolded:
  - At the beginning we know where we are
  - Roughly estimate the distance of each step
  - Uncertainty accumulates over time
- When you open your eyes at some time:
  - Can observe the soundings
  - Uncertainty in each step is still the same
  - But can be corrected by observation
  - Overall uncertainty can be kept within a certain range

1. Recursive Optimization

- Recall the motion and observation model:

$$\begin{cases} x_k = f\left(x_{k-1}, u_k\right) + w_k \\ z_{k,j} = h\left(y_j, x_k\right) + v_{k,j} \end{cases} \qquad k = 1, \ldots, N, \; j = 1, \ldots, M.$$

- Let's start from Bayes filter
- Use x_k to denote the unknown variables in time k:

$$x_k \triangleq \{x_k, y_1, \ldots, y_m\}.$$

- Then the model can be simplified as:

$$\begin{cases} x_k = f\left(x_{k-1}, u_k\right) + w_k \\ z_k = h\left(x_k\right) + v_k \end{cases} \qquad k = 1, \ldots, N.$$

1. Recursive Optimization

- We show how to derive the recursive approach from batch approach
- Estimate the current state given data from 0 to k:

$$P(x_k | x_0, u_{1:k}, z_{1:k}).$$

- Bayes' rule (switch z_k):

$$P\left(x_k | x_0, u_{1:k}, z_{1:k}\right) \propto \underbrace{P\left(z_k | x_k\right)}_{\boxed{\text{Likehood}}} \underbrace{P\left(x_k | x_0, u_{1:k}, z_{1:k-1}\right)}_{\boxed{\text{Prior}}}.$$

- Expand the prior:

$$P\left(x_k | x_0, u_{1:k}, z_{1:k-1}\right) = \int \underbrace{P\left(x_k | x_{k-1}, x_0, u_{1:k}, z_{1:k-1}\right)}_{\boxed{\text{Motion model prediction}}} \underbrace{P\left(x_{k-1} | x_0, u_{1:k}, z_{1:k-1}\right)}_{\boxed{\text{Estimation in k-1}}} \mathrm{d}x_{k-1}. \tag{10.6}$$

1. Recursive Optimization

$$P(x_k|x_0, u_{1:k}, z_{1:k-1}) = \int P(x_k|x_{k-1}, x_0, u_{1:k}, z_{1:k-1})\, P(x_{k-1}|x_0, u_{1:k}, z_{1:k-1})\, \mathrm{d}x_{k-1}. \tag{10.6}$$

- Different ways to treat this equation:
  - Assume the Markov property: we assume x_k is only relevant to x_k-1
  - Don’t assume Markov property: x_k is relevant to all previous state

1. Recursive Optimization

- By assuming the Markov's property:

$$P\left(x_k | x_{k-1}, x_0, u_{1:k}, z_{1:k-1}\right) = P\left(x_k | x_{k-1}, u_k\right).$$

- The second item becomes:

$$P\left(x_{k-1} | x_0, u_{1:k}, z_{1:k-1}\right) = P\left(x_{k-1} | x_0, u_{1:k-1}, z_{1:k-1}\right).$$

- This equation (Bayes' rule) shows how to recursively estimate the status
  - But we haven't set the specific form of motion and obs model
- In Linear-Gaussian (LG) system, the recursive approach will lead to Kalman Filter (KF)

1. Recursive Optimization

- Derivation of KF in LG system

$$\begin{cases} x_k = A_k x_{k-1} + u_k + w_k \\ z_k = C_k x_k + v_k \end{cases} \qquad k = 1, \ldots, N.$$

$w_k \sim N(0, R). \quad v_k \sim N(0, Q).$ are noises

- Assume the state variables are Gaussian

$$P(x_{k-1}) = N\left(\hat{x}_{k-1}, \hat{P}_{k-1}\right) \qquad \bar{x}_k, \bar{P}_k$$

Posterior Prior

- Use different notations since we need Bayes' rule

1. Recursive Optimization

- Some conclusions to start with:
- Linear transform of Gaussian distribution:
  - Assume $x \sim N(\mu, \Sigma), y = Ax + b$, then y is also Gaussian and satisfies:

$$E[y] = E[Ax + b] = AE[x] + b = A\mu + b$$

$$\begin{aligned} Cov[y] &= E[(y - E[y])(y - E[y])^T] \\ &= E[A(x - \mu)(x - \mu)^T A^T] = A\Sigma A^T \end{aligned}$$

1. Recursive Optimization

- With this we can derive the prior at time k using motion model:

$$\begin{cases} \boldsymbol{x}_k = \boldsymbol{A}_k \boldsymbol{x}_{k-1} + \boldsymbol{u}_k + \boldsymbol{w}_k \\ \boldsymbol{z}_k = \boldsymbol{C}_k \boldsymbol{x}_k + \boldsymbol{v}_k \end{cases} \qquad k = 1, \ldots, N.$$

- With motion model:

$$P(\boldsymbol{x}_k | \boldsymbol{x}_0, \boldsymbol{u}_{1:k}, \boldsymbol{z}_{1:k-1}) = N\left(\boldsymbol{A}_k \hat{\boldsymbol{x}}_{k-1} + \boldsymbol{u}_k, \boldsymbol{A}_k \hat{\boldsymbol{P}}_k \boldsymbol{A}_k^{\mathrm{T}} + \boldsymbol{R}\right).$$

- This equation gives the prior distribution, denoted it as:

$$\bar{\boldsymbol{x}}_k = \boldsymbol{A}_k \hat{\boldsymbol{x}}_{k-1} + \boldsymbol{u}_k, \quad \bar{\boldsymbol{P}}_k = \boldsymbol{A}_k \hat{\boldsymbol{P}}_{k-1} \boldsymbol{A}_k^{\mathrm{T}} + \boldsymbol{R}.$$

- From observation model we know:

$$P(z_k | x_k) = N(C_k x_k, Q).$$

- Compute the posterior model:

$$P(\boldsymbol{x}_k | \boldsymbol{x}_0, \boldsymbol{u}_{1:k}, \boldsymbol{z}_{1:k}) \propto P(\boldsymbol{z}_k | \boldsymbol{x}_k) P(\boldsymbol{x}_k | \boldsymbol{x}_0, \boldsymbol{u}_{1:k}, \boldsymbol{z}_{1:k-1}).$$

1. Recursive Optimization

- A small trick: we assume the posterior is also Gaussian, so:

$$N(\hat{x}_k, \hat{P}_k) = \eta N(C_k x_k, Q_k) \cdot N(\bar{x}_k, \bar{P}_k)$$

- Since they are all Gaussian, so we just expand it and compare the linear and quadratic coefficients
- The exponential part is:

$$\left(x_k - \hat{x}_k\right)^{\mathrm{T}} \hat{P}_k^{-1} \left(x_k - \hat{x}_k\right) = \left(z_k - C_k x_k\right)^{\mathrm{T}} Q^{-1} \left(z_k - C_k x_k\right) + \left(x_k - \bar{x}_k\right)^{\mathrm{T}} \bar{P}_k^{-1} \left(x_k - \bar{x}_k\right).$$

- Compare the coefficients of x_k, for the quadratic part we have:

$$\boxed{\hat{P}_k^{-1} = C_k^{\mathrm{T}} Q^{-1} C_k + \bar{P}_k^{-1}.}$$

1. Recursive Optimization

- For the linear part we have:

$$\left(x_k - \hat{x}_k\right)^{\mathrm{T}} \hat{P}_k^{-1} \left(x_k - \hat{x}_k\right) = \left(z_k - C_k x_k\right)^{\mathrm{T}} Q^{-1} \left(z_k - C_k x_k\right) + \left(x_k - \bar{x}_k\right)^{\mathrm{T}} \bar{P}_k^{-1} \left(x_k - \bar{x}_k\right).$$

$$-2\hat{x}_k^{\mathrm{T}} \hat{P}_k^{-1} x_k = -2 z_k^{\mathrm{T}} Q^{-1} C_k x_k - 2\bar{x}_k^{\mathrm{T}} \bar{P}_k^{-1} x_k$$

- Rearrange it:

$$\boxed{\hat{P}_k^{-1} \hat{x}_k = C_k^{\mathrm{T}} Q^{-1} z_k + \bar{P}_k^{-1} \bar{x}_k}$$

- Left multiply $\hat{P}_k$ and define: $K = \hat{P}_k C_k^{\mathrm{T}} Q^{-1}$, then we have:

$$\begin{aligned} \hat{x}_k &= \hat{P}_k C_k^{\mathrm{T}} Q^{-1} z_k + \hat{P}_k \bar{P}_k^{-1} \bar{x}_k \\ &= K z_k + \left(I - K C_k\right) \bar{x}_k = \bar{x}_k + K\left(z_k - C_k \bar{x}_k\right). \end{aligned}$$

Innovation part

K: Kalman gain

$$\boxed{\hat{P}_k^{-1} = C_k^{\mathrm{T}} Q^{-1} C_k + \bar{P}_k^{-1}.}$$

1. Recursive Optimization

- Kalman gain: $K = \hat{P}_k C_k^{\mathrm{T}} Q^{-1}$ requires $\hat{P}_k$
- Another form:

$$\begin{aligned} K &= \left(C_k^T Q^{-1} C_k + \bar{P}_k^{-1}\right)^{-1} C_k^T Q^{-1} \\ &= \bar{P}_k C_k^T \left(Q + C_k \bar{P}_k C^T\right)^{-1} \end{aligned}$$

- This requires the Sherman-Morrison-Woodbury identities:

$$\left(\boldsymbol{A}^{-1} + \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C}\right)^{-1} \equiv \boldsymbol{A} - \boldsymbol{A}\boldsymbol{B}\left(\boldsymbol{D} + \boldsymbol{C}\boldsymbol{A}\boldsymbol{B}\right)^{-1}\boldsymbol{C}\boldsymbol{A} \tag{2.75a}$$

$$\left(\boldsymbol{D} + \boldsymbol{C}\boldsymbol{A}\boldsymbol{B}\right)^{-1} \equiv \boldsymbol{D}^{-1} - \boldsymbol{D}^{-1}\boldsymbol{C}\left(\boldsymbol{A}^{-1} + \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C}\right)^{-1}\boldsymbol{B}\boldsymbol{D}^{-1} \tag{2.75b}$$

$$\boldsymbol{A}\boldsymbol{B}\left(\boldsymbol{D} + \boldsymbol{C}\boldsymbol{A}\boldsymbol{B}\right)^{-1} \equiv \left(\boldsymbol{A}^{-1} + \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C}\right)^{-1}\boldsymbol{B}\boldsymbol{D}^{-1} \tag{2.75c}$$

$$\left(\boldsymbol{D} + \boldsymbol{C}\boldsymbol{A}\boldsymbol{B}\right)^{-1}\boldsymbol{C}\boldsymbol{A} \equiv \boldsymbol{D}^{-1}\boldsymbol{C}\left(\boldsymbol{A}^{-1} + \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C}\right)^{-1} \tag{2.75d}$$

1. Recursive Optimization

- Two steps in Kalman filter

1. Prediction

$$\bar{x}_k = A_k \hat{x}_{k-1} + u_k, \qquad \bar{P}_k = A_k \hat{P}_{k-1} A_k^T + R$$

2. Correction
    - Compute Kalman gain:

$$K = \bar{P}_k C_k^T \left(Q + C_k \bar{P}_k C^T\right)^{-1}$$

    - Update the estimation:

$$\hat{x}_k = \bar{x}_k + K(z_k - C_k \bar{x}_k)$$
$$\hat{P}_k = (I - K C_k)\bar{P}_k$$

1. Recursive Optimization

- Some notes on Kalman filter
  - Kalman filter is the BLUE (best linear unbiased estimate) estimation in LG system
  - Kalman filter gives the same result as MAP in LG system
    - This is because the mode and mean are same in Gauss distribution
    - We can also derive KF through optimization way
    - Or by choose a best Kalman gain to get the best estimation

1. Recursive Optimization

- Extended KF in NL systems:

$$\begin{cases} x_k = f\left(x_{k-1}, u_k\right) + w_k \\ z_k = h\left(x_k\right) + v_k \end{cases} \qquad k = 1, \ldots, N.$$

- We take the Taylor expansion in current estimate:

$$x_k \approx f\left(\hat{x}_{k-1}, u_k\right) + \left.\frac{\partial f}{\partial x_{k-1}}\right|_{\hat{x}_{k-1}} \left(x_{k-1} - \hat{x}_{k-1}\right) + w_k.$$

Denoted as F

$$z_k \approx h\left(\bar{x}_k\right) + \left.\frac{\partial h}{\partial x_k}\right|_{\bar{x}_k} \left(x_k - \hat{x}_k\right) + n_k.$$

Denoted as H

1. Recursive Optimization

$$\begin{cases} x_k = f(x_{k-1}, u_k) + w_k \\ z_k = h(x_k) + v_k \end{cases} \qquad k = 1, \ldots, N.$$

- Then employ the conclusions in KF:
- Prediction:

$$\bar{x}_k = f(\hat{x}_{k-1}, u_k), \quad \bar{P}_k = F \hat{P}_k F^{\mathrm{T}} + R_k.$$

- Correction:
  - Kalman gain: $K_k = \bar{P}_k H^{\mathrm{T}} (H \bar{P}_k H^{\mathrm{T}} + Q_k)^{-1}.$
  - Update: $\hat{x}_k = \bar{x}_k + K_k (z_k - h(\bar{x}_k)), \hat{P}_k = (I - K_k H) \bar{P}_k.$

1. Recursive Optimization

- Discussion of KF and EKF
- Advantages
  - Clean and simple
  - Do not require any property of motion and observation model
  - Can be used for multiple sensor fusion
- Disadvantages
  - Need to assume Markov property (which is not satisfied when we have loop closure)
  - May diverge if the observations have outliers
  - Linearization may have error if the model has strong nonlinearity
  - Gaussian approximation may not be accurate for some variables
  - Need to store the mean and covariance matrix for all status

Contents

2. Batch Optimization

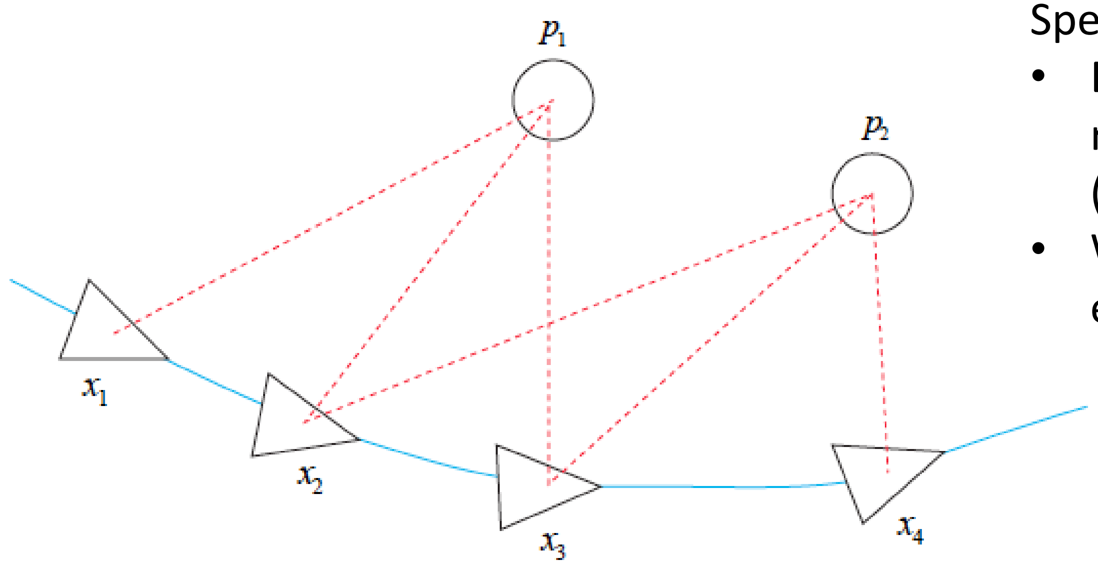
- Batch optimization
- We've shown some conclusions in Lecture 3
- MAP estimation is equivalent to least square solution

$$\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}\|e_{ij}\|^2 = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}\|z_{ij} - h(\xi_i, p_j)\|^2.$$

- It is called Bundle Adjustment when used in visual SLAM systems
  - We have a bundle of lights and adjust the cameras to fit the observation model

2. Batch Optimization

- BA and graph optimization
  - Least square in BA can be represented as a graph G={V,E}
  - Where V is the node set containing the optimization variables
  - And E is the edge set containing the observation errors



Special pattern in BA:

- Each observation is only related to two variables (nodes)
- We don’t have point-point edges (structure prior)

2. Batch Optimization

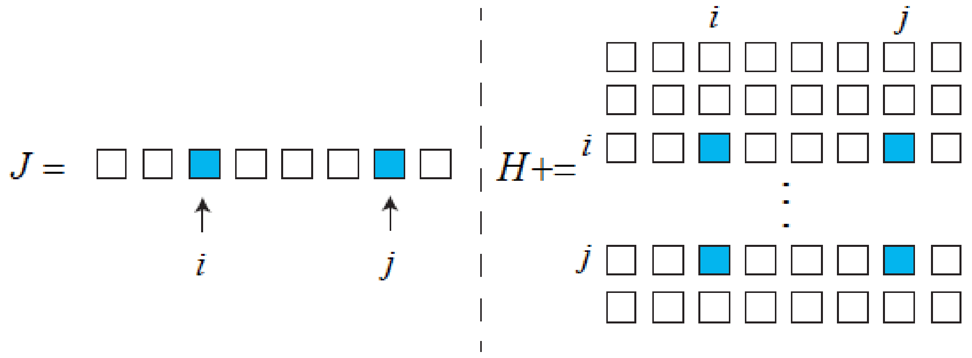
- According to optimization theory we will finally need to solve the normal equation:

$$H \Delta x = -b$$

- Each edge contributes to this H by: $H = \sum_{i,j} J_{ij}^T J_{ij}$
- Consider an observation regarding to i-th camera and j-th point:

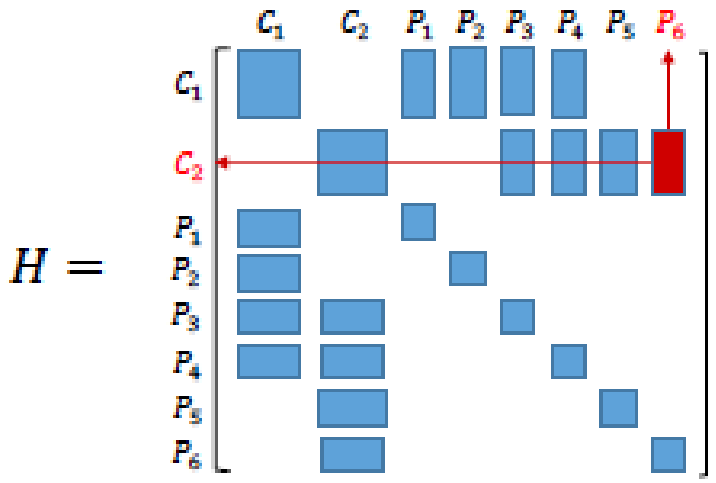
$$\boldsymbol{J}_{ij}(\boldsymbol{x}) = \left(\mathbf{0}_{2\times6}, \dots \mathbf{0}_{2\times6}, \frac{\partial \boldsymbol{e}_{ij}}{\partial \boldsymbol{\xi}_i}, \mathbf{0}_{2\times6}, \dots \mathbf{0}_{2\times3}, \dots \mathbf{0}_{2\times3}, \frac{\partial \boldsymbol{e}_{ij}}{\partial \boldsymbol{p}_j}, \mathbf{0}_{2\times3}, \dots \mathbf{0}_{2\times3} \right).$$

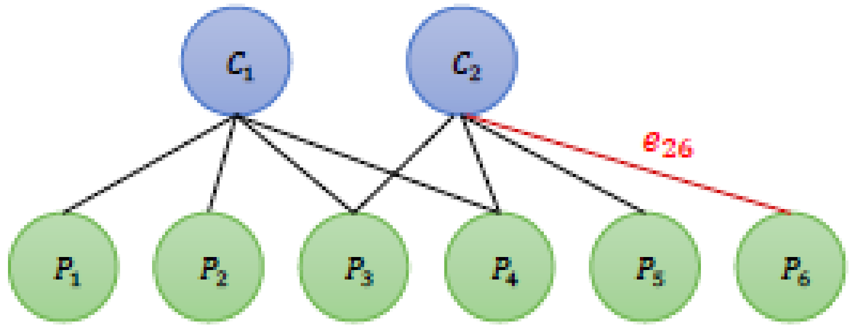
- This is a sparse matrix that only has two non-zero entries:



2. Batch Optimization

- If we set the order of the overall status by keeping the cameras at first and points at last, then the H matrix has the special form:

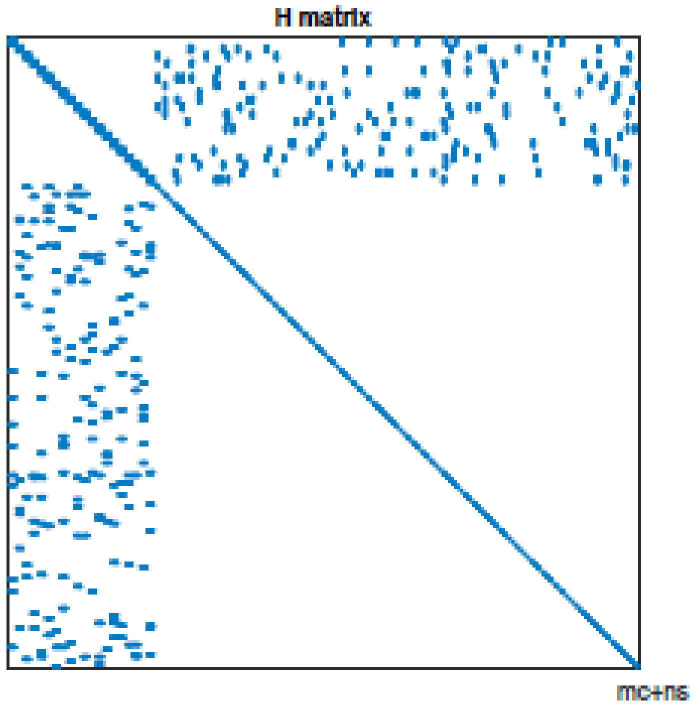

$H =$
C_1 C_2 P_1 P_2 P_3 P_4 P_5 P_6
C_1 C_2 P_1 P_2 P_3 P_4 P_5 P_6



- The relationship of the graph and H matrix:
  - Each edge in the graph is corresponding to a non-zero block in H

2. Batch Optimization

- In real-world BA the number of points is far more than cameras, so the H will be:



The Arrow-like H matrix

2. Batch Optimization

- For a dense H matrix we need to inverse it to solve the normal equation, which has O(n^3) complexity
- But in BA this can be accelerated by employing the special structure of H
- Split the blocks in H:

$$\begin{bmatrix} \mathbf{B} & \mathbf{E} \\ \mathbf{E}^{\mathrm{T}} & \mathbf{C} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x}_c \\ \Delta \boldsymbol{x}_p \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \end{bmatrix}.$$

B and C are diagonal block matrices
E and E^T is dense and the non-zero blocks are corresponding to real observations

- Idea:
  - Since C is block diagonal, we use Gaussian elimination to eliminate the E and E^T

$$\begin{bmatrix} \mathbf{I} & -\mathbf{E}\mathbf{C}^{-1} \\ 0 & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{B} & \mathbf{E} \\ \mathbf{E}^{\mathrm{T}} & \mathbf{C} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x}_c \\ \Delta \boldsymbol{x}_p \end{bmatrix} = \begin{bmatrix} \mathbf{I} & -\mathbf{E}\mathbf{C}^{-1} \\ 0 & \mathbf{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \end{bmatrix}. \Rightarrow \begin{bmatrix} \mathbf{B} - \mathbf{E}\mathbf{C}^{-1}\mathbf{E}^{\mathrm{T}} & 0 \\ \mathbf{E}^{\mathrm{T}} & \mathbf{C} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x}_c \\ \Delta \boldsymbol{x}_p \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} - \mathbf{E}\mathbf{C}^{-1}\boldsymbol{w} \\ \boldsymbol{w} \end{bmatrix}.$$

2. Batch Optimization

- So the normal equation becomes:

$$\begin{bmatrix} B - EC^{-1}E^{\mathrm{T}} & 0 \\ E^{\mathrm{T}} & C \end{bmatrix} \begin{bmatrix} \Delta x_c \\ \Delta x_p \end{bmatrix} = \begin{bmatrix} v - EC^{-1}w \\ w \end{bmatrix}.$$

- Solve it in two steps:
  1. Solve the upper part to get Δx_c
  2. Take it into the lower part and get Δx_p
- This is called Marginalization or Schur complement
  - We can also use other approaches like Cholesky decomposition to solve this sparse linear problem

2. Batch Optimization

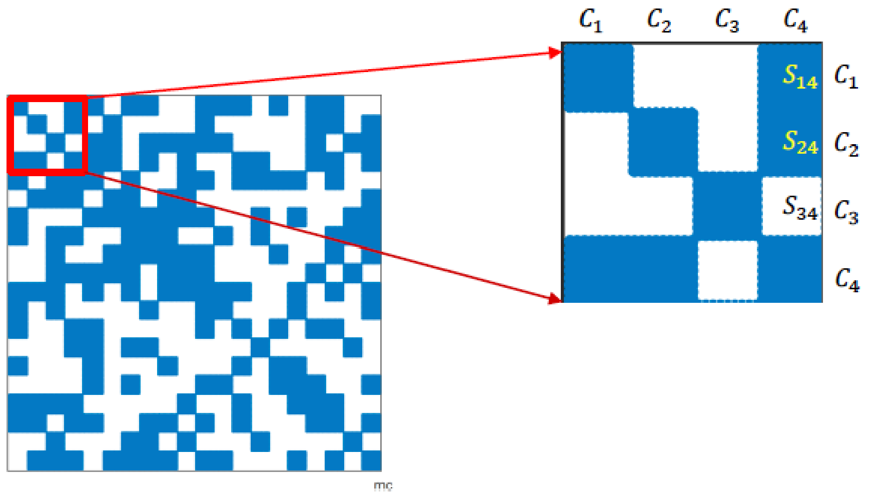
- Marginalization
  - From the probabilistic theory, it means:
  - $P(x_c, x_p) = P(x_c) \cdot P(x_p | x_c).$ Joint = Marginal * Conditional

  - In BA, we marginalize all the points into the cameras to make the acceleration
  - And in KF & EKF, we actually marginalize all the past state into the current state

  - We can also choose to marginalize part of the points or part of the cameras

2. Batch Optimization

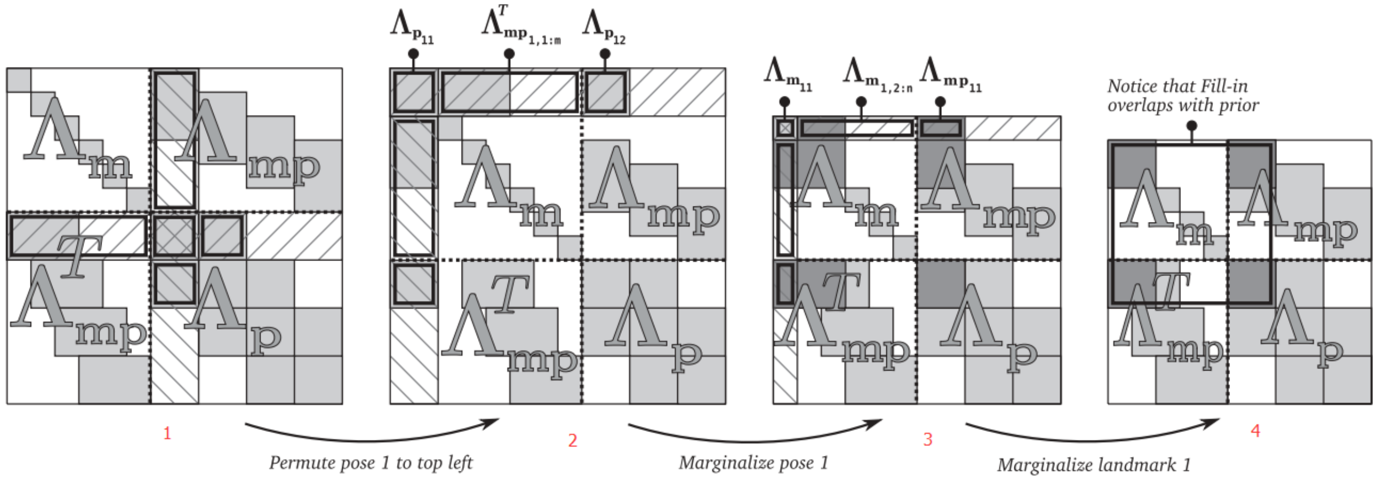
- After marginalization, the top-left corner of H won't have the sparse structure again:



- But it shows the co-visibility relationship of the cameras
  - The non-zero block in i,j means camera i and camera j have observed at least one same point

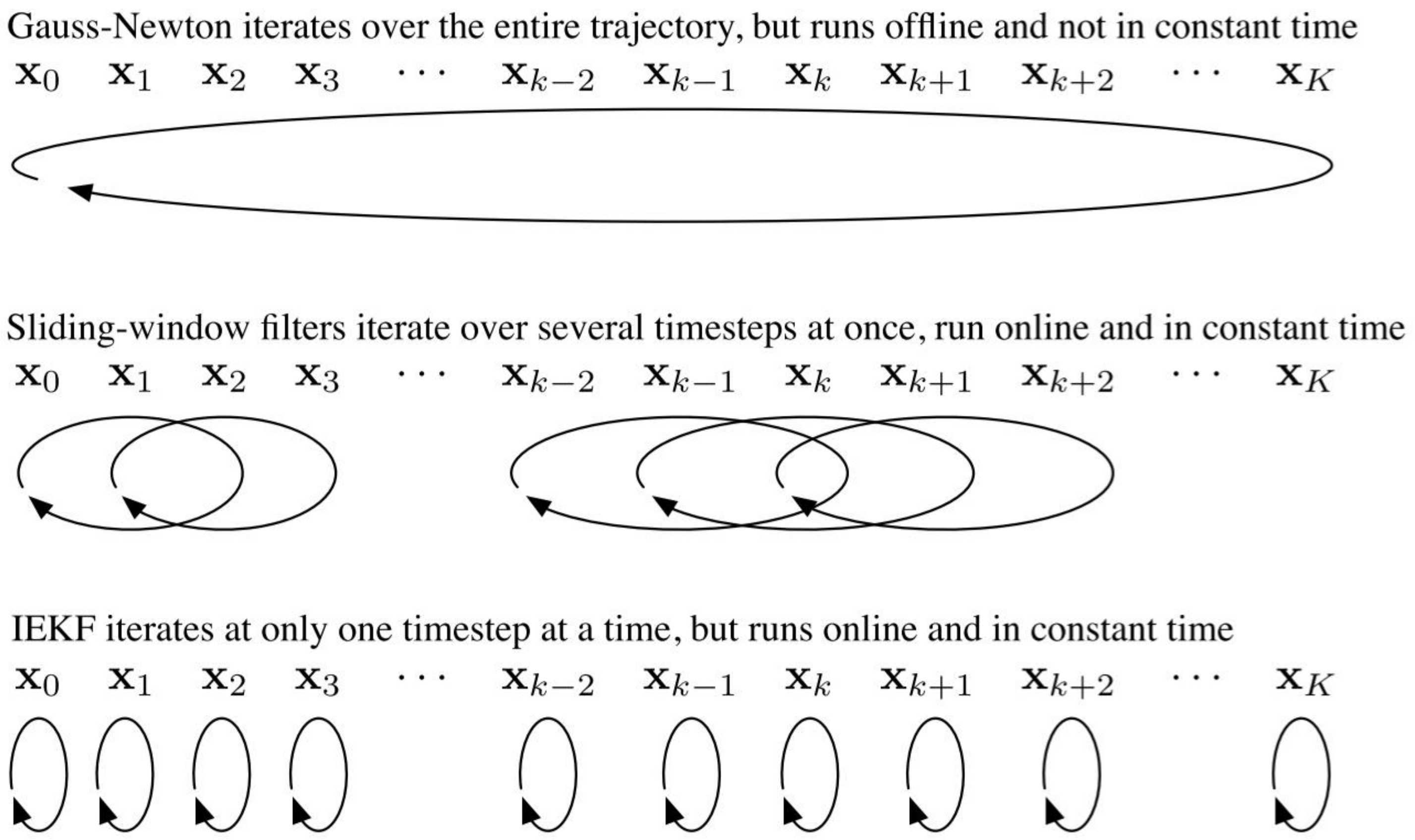
2. Batch Optimization

- Marginalization will fill the original matrix and make it no longer sparse
- So in KF & EKF, the covariance matrix is not sparse
- And in recursive problems, we can
  - Just use a dense matrix but keep it small (like EKF, only keeps the current camera estimation)
  - Or use a special marginalization strategy to keep it sparse



2. Batch Optimization

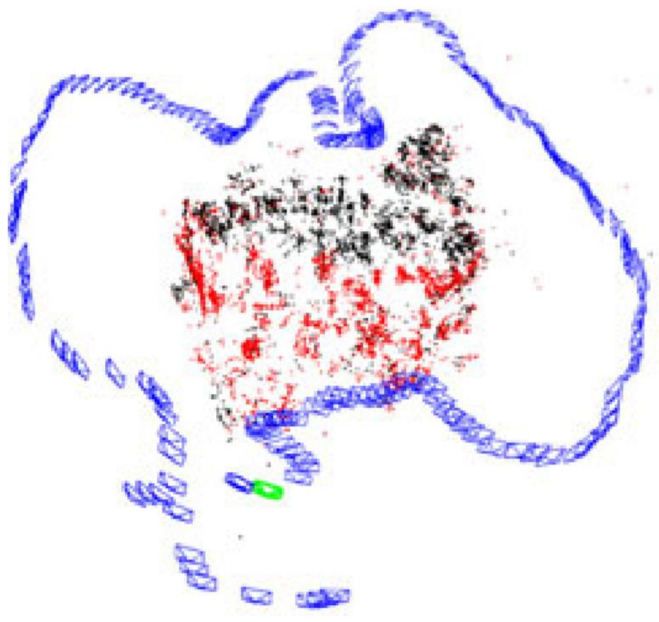
- Comparison of recursive and batch approaches:


Gauss-Newton iterates over the entire trajectory, but runs offline and not in constant time
$\mathbf{x}_0$ $\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\cdots$ $\mathbf{x}_{k-2}$ $\mathbf{x}_{k-1}$ $\mathbf{x}_k$ $\mathbf{x}_{k+1}$ $\mathbf{x}_{k+2}$ $\cdots$ $\mathbf{x}_K$
Sliding-window filters iterate over several timesteps at once, run online and in constant time
$\mathbf{x}_0$ $\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\cdots$ $\mathbf{x}_{k-2}$ $\mathbf{x}_{k-1}$ $\mathbf{x}_k$ $\mathbf{x}_{k+1}$ $\mathbf{x}_{k+2}$ $\cdots$ $\mathbf{x}_K$
IEKF iterates at only one timestep at a time, but runs online and in constant time
$\mathbf{x}_0$ $\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\cdots$ $\mathbf{x}_{k-2}$ $\mathbf{x}_{k-1}$ $\mathbf{x}_k$ $\mathbf{x}_{k+1}$ $\mathbf{x}_{k+2}$ $\cdots$ $\mathbf{x}_K$

2. Batch Optimization

- Apply BA in SLAM
  - Manage a keyframe set and map point set

- Batch approach
  - Use BA to optimize part of the graph
  - Keep others fixed
- Recursive approach (sliding window)
  - Keep a constant number of keyframes
  - Use BA to optimize the keyframe and points inside the window
  - Marginalize old keyframe and points when new data arrived

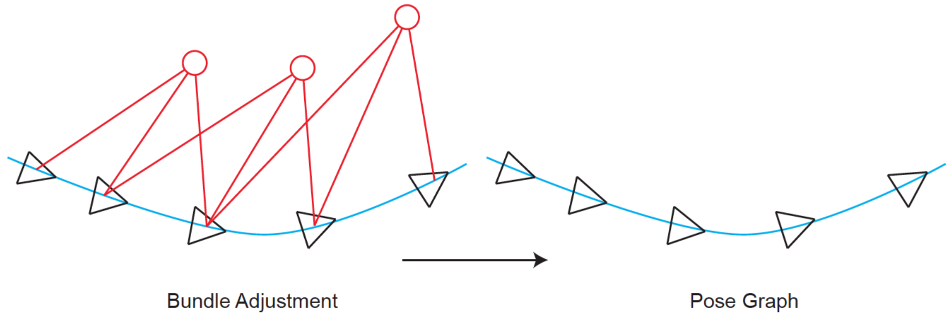


Contents

3. Pose graph

- BA usually needs much computation resource
  - So we put it in a single backed thread
  - Modern CPU need several seconds to solve a problem with 100 cameras and 100,000 points

- If we build a problem that only has cameras and no points, then the computation can be greatly reduced



3. Pose graph

- Pose graph
  - Vertex: cameras only
  - Edge: camera transform as observation

$$\boldsymbol{\Delta T}_{ij} = \boldsymbol{T}_i^{-1}\boldsymbol{T}_j.$$

  - Error:

$$\begin{aligned}\boldsymbol{e}_{ij} &= \ln\left(\Delta \boldsymbol{T}_{ij}^{-1}\boldsymbol{T}_i^{-1}\boldsymbol{T}_j\right)^{\vee} \\ &= \ln\left(\exp((-\boldsymbol{\xi}_{ij})^{\wedge})\exp((-\boldsymbol{\xi}_i)^{\wedge})\exp(\boldsymbol{\xi}_j^{\wedge})\right)^{\vee}.\end{aligned}$$

3. Pose graph

- Jacobians

$$
\begin{aligned}
\hat{e}_{ij} &= \ln\left(\boldsymbol{T}_{ij}^{-1}\boldsymbol{T}_i^{-1}\exp((-\boldsymbol{\delta\xi}_i)^\wedge)\exp(\delta\boldsymbol{\xi}_j^\wedge)\boldsymbol{T}_j\right)^\vee \\
&= \ln\left(\boldsymbol{T}_{ij}^{-1}\boldsymbol{T}_i^{-1}\boldsymbol{T}_j\exp\left(\left(-\mathrm{Ad}(\boldsymbol{T}_j^{-1})\boldsymbol{\delta\xi}_i\right)^\wedge\right)\exp((\mathrm{Ad}(\boldsymbol{T}_j^{-1})\boldsymbol{\delta\xi}_j)^\wedge)\right)^\vee \\
&\approx \ln\left(\boldsymbol{T}_{ij}^{-1}\boldsymbol{T}_i^{-1}\boldsymbol{T}_j\left[\boldsymbol{I}-(\mathrm{Ad}(\boldsymbol{T}_j^{-1})\boldsymbol{\delta\xi}_i)^\wedge+(\mathrm{Ad}(\boldsymbol{T}_j^{-1})\boldsymbol{\delta\xi}_j)^\wedge\right]\right)^\vee \\
&\approx \boldsymbol{e}_{ij}+\frac{\partial \boldsymbol{e}_{ij}}{\partial\boldsymbol{\delta\xi}_i}\boldsymbol{\delta\xi}_i+\frac{\partial \boldsymbol{e}_{ij}}{\partial\boldsymbol{\delta\xi}_j}\boldsymbol{\delta\xi}_j
\end{aligned}
.
$$

$$
\frac{\partial \boldsymbol{e}_{ij}}{\partial\boldsymbol{\delta\xi}_i} = -\boldsymbol{\mathcal{J}}_r^{-1}(\boldsymbol{e}_{ij})\mathrm{Ad}(\boldsymbol{T}_j^{-1}).
$$

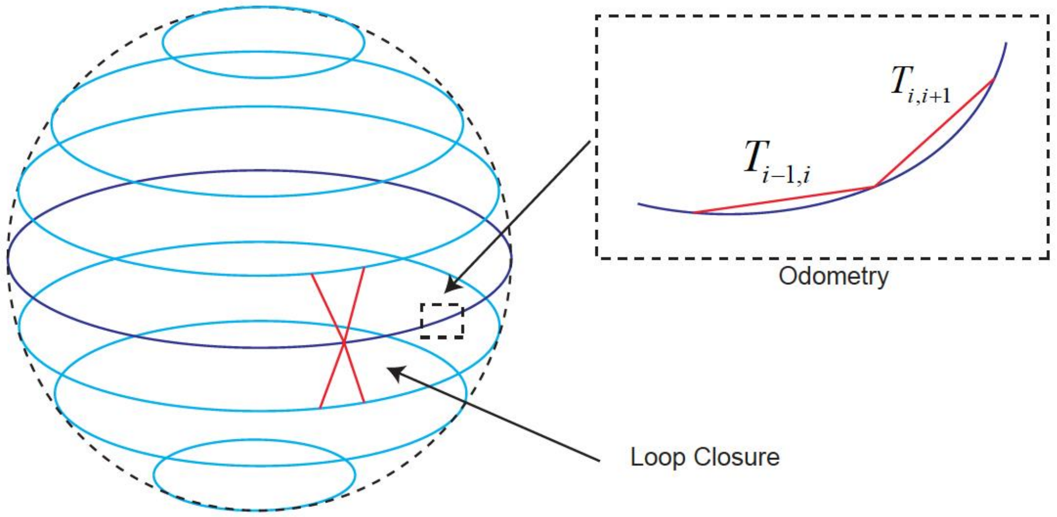
$$
\frac{\partial \boldsymbol{e}_{ij}}{\partial\boldsymbol{\delta\xi}_j} = \boldsymbol{\mathcal{J}}_r^{-1}(\boldsymbol{e}_{ij})\mathrm{Ad}(\boldsymbol{T}_j^{-1}).
$$

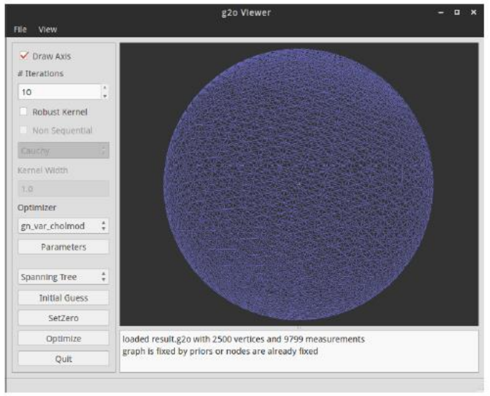
$$
\boldsymbol{\mathcal{J}}_r^{-1}(\boldsymbol{e}_{ij}) \approx \boldsymbol{I}+\frac{1}{2}\begin{bmatrix}\boldsymbol{\phi}_e^\wedge & \boldsymbol{\rho}_e^\wedge \\ \boldsymbol{0} & \boldsymbol{\phi}_e^\wedge\end{bmatrix}.
$$

Adjoint: $\boxed{\exp\left((\mathrm{Ad}(\boldsymbol{T})\boldsymbol{\xi})^\wedge\right) = \boldsymbol{T}\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{T}^{-1}.}$

3. Pose graph

- Assignment in pose graph
- Pose ball

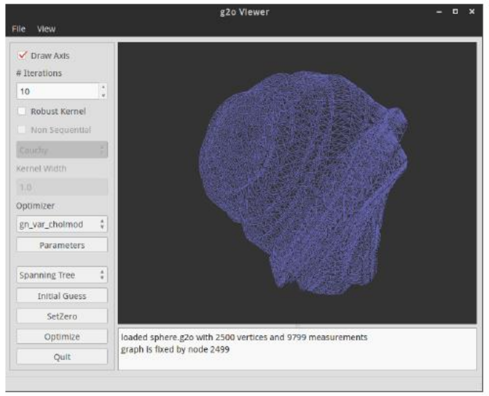




[✓ Draw Axis]
Iterations: 10
Optimizer: gn_var_cholmod

loaded result.g2o with 2500 vertices and 9799 measurements
graph is fixed by priors or nodes are already fixed

Add Noise



[✓ Draw Axis]
Iterations: 10
Optimizer: gn_var_cholmod

loaded sphere.g2o with 2500 vertices and 9799 measurements
graph is fixed by node 2499

Any questions?