Bundle Adjustment: A Tutorial

Siddharth Choudhary
What is Bundle Adjustment?

- Refines a visual reconstruction to produce jointly optimal 3D structure and viewing parameters.

- ‘bundle’ refers to the bundle of light rays leaving each 3D feature and converging on each camera center.
Re Projection Error

Reprojection error: \[ \| \mathbf{q}_{ij} - P(C_i, X_j) \| \]

Objective function:
\[
g(C, X) = \sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij} \| \mathbf{q}_{ij} - P(C_i, X_j) \|^2
\]

Indicator variable:
1 if point \( j \) is visible in camera \( i \)
0 otherwise
Some Notations

- Structure and Cameras being parameterized by a single large vector ‘x’
- Small displacement in x represented by $\partial x$
- Observations denoted by ‘z’
- Predicted values at parameter value x, denoted by $z = z(x)$
- Residual prediction error, $\Delta z(x) = z - z(x)$
- Cost Function = $f(x) = f\left(\text{predz}(x)\right)$
Objective Function

- Minimization of weighted sum of squared error (SSE) cost function:

\[
f(x) \equiv \frac{1}{2} \sum_i \Delta z_i(x)^T W_i \Delta z_i(x), \quad \Delta z_i(x) \equiv z_i - z_i(x)
\]
Some Facts about Non linear least squares

- Least-squares fitting is a maximum likelihood estimation of the fitted parameters if the measurement errors are independent and normally distributed with constant standard deviation.

- The probability distribution of the sum of a very large number of very small random deviations almost always converges to a normal distribution.
Disadvantage of Non Linear Least Squares

- It is highly sensitive to outliers, because the Gaussian has extremely small tails compared to most real measurement error distribution.

( It is the reason of using Hierarchical SFM )

Gaussian Tail problem and its effects is addressed in the paper ‘Pushing the envelope of modern bundle adjustment techniques, CVPR 2010’
Optimization Techniques

- Gradient Descent Method
- Newton-Rhapson Method
- Gauss – Newton Method
- Levenberg – Marquardt Method
Gradient Descent Method

- A first-order optimization algorithm.
- To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point.

While $k < k_{\text{max}}$

$$x_k = x_{k-1} - \lambda \nabla f(x_{k-1})$$
It is robust when x is far from optimum but has poor final convergence.

( this fact is used in designing the LM iteration )
Newton - Rhapson Method

- It is a second order optimization method
- Newton's method can often converge remarkably quickly, especially if the initial guess is close to the desired root.

\[
\begin{align*}
\frac{df}{dx}(x + \delta x) & \approx H \delta x + g \\
\delta x & = -H^{-1}g
\end{align*}
\]
Newton – Rhapson Method

- For quadratic function it converges in one iteration
- For other general function, its asymptotic convergence is quadratic
- The disadvantage of this method is the high computation complexity of $H^{-1}$
The Gauss–Newton algorithm is a method used to solve non-linear least squares problems.

\[ f(x) \equiv \frac{1}{2} \Delta z(x)^T W \Delta z(x) \]

\[ g \equiv \frac{df}{dx} = \Delta z^T W J \]

\[ H \equiv \frac{d^2f}{dx^2} = J^T W J + \sum_i (\Delta z^T W)_i \frac{d^2z_i}{dx^2} \]

\[ H \approx J^T W J. \quad \frac{d^2z_i}{dx^2} \approx 0 \]

\[ (J^T W J) \delta x = -J^T W \Delta z \]
For well-parametrized bundle problems under an outlier-free least squares cost model evaluated near the cost minimum, the Gauss-Newton approximation is usually very accurate.
Levenberg – Marquardt Algorithm

- The LMA interpolates between the Gauss–Newton algorithm (GNA) and the method of gradient descent.

- When far from the minimum it acts as a steepest descent and it performs Gauss Newton iteration when
  \[(H + \lambda W)\delta x = -g\]
Levenberg – Marquardt Algorithm

- It takes in to account the best of both gradient descent and gauss newton method

\[
\lambda \gg 1 \Rightarrow \text{Gradient Descent Method} \\
\lambda < 1 \Rightarrow \text{Gauss – Newton Method}
\]
General Facts about optimization methods

- Second order optimization methods like Gauss – Newton and LM requires a few but heavy iterations.

- First order optimization methods like Gradient descent requires a lot of light iterations.
General Implementation

Issues

- Exploit the problem structure
- Use factorization effectively
- Use stable local parametrizations
- Scaling and preconditioning
Computational Bottleneck in LM Iteration

\[ \delta x = -(H + \lambda W)^{-1} g \]

\[ H^{-1} \approx (J^T W J)^{-1} \]
computation is the main bottleneck
Fig. 3. The network graph, parameter connection graph, Jacobian structure and Hessian structure for a toy bundle problem with five 3D features A–E, four images 1–4 and two camera calibrations $K_1$ (shared by images 1,2) and $K_2$ (shared by images 3,4). Feature A is seen in images 1,2; B in 1,2,4; C in 1,3; D in 2–4; and E in 3,4.
$H^{-1}$ Calculation Strategies

- The Schur Complement and the reduced camera system
- Cholesky Decomposition
- Sparse Factorization
  - Variable Ordering
    - Top down ordering
    - Bottom up ordering
  - Preconditioning
  - Conjugate Gradient method
  - Multigrid Methods
Schur Complement

\[ H \delta x = \begin{pmatrix} U \\ W^T \end{pmatrix} \begin{pmatrix} W \\ V \end{pmatrix} \begin{pmatrix} \delta_a \\ \delta_b \end{pmatrix} = \begin{pmatrix} \varepsilon_a \\ \varepsilon_b \end{pmatrix} \]

Left Multiply

\[
\begin{pmatrix}
I & -WV^{*-1} \\
0 & I
\end{pmatrix}
\]

to both sides

\[
\begin{pmatrix}
U - WV^{-1}W^T & 0 \\
W^T & V
\end{pmatrix}
\begin{pmatrix}
\delta_a \\
\delta_b
\end{pmatrix} = \begin{pmatrix}
\varepsilon_a - WV^{-1}\varepsilon_b \\
\varepsilon_b
\end{pmatrix}
\]

Reduced Camera System
Cholesky Decomposition

Decompose the matrix $A$ into $A = LL^T$, where $L$ is a lower triangular matrix

\[
A = \begin{pmatrix} a_{11} & * \\ a_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} \lambda_{11} & 0 \\ l_{21} & L_{22} \end{pmatrix} \begin{pmatrix} \lambda_{11} & l_{21}^T \\ 0 & L_{22}^T \end{pmatrix} = \begin{pmatrix} \lambda_{11}^2 & * \\ \lambda_{11} l_{21} & l_{21} l_{21}^T + L_{22} L_{22}^T \end{pmatrix}
\]

1. Partition $A = \begin{pmatrix} \alpha_{11} & * \\ a_{21} & A_{22} \end{pmatrix}$

2. $\alpha_{11} \leftarrow \lambda_{11} = \sqrt{\alpha_{11}}$

3. $a_{21} \leftarrow l_{21} = a_{21} / \lambda_{11}$

4. $A_{22} \leftarrow A_{22} - l_{21} l_{21}^T$

5. continue recursively with $A_{22}$
Sparse Factorization methods

- Since both the Hessian and the reduced camera system is sparse for large scale systems, sparse factorization methods are preferred.
  - Variable Ordering
  - Preconditioning
  - Conjugate Gradient Method
  - Parallel Multigrid Methods
There is a phenomenon of *fill-in.*

After each step, we have more number of non-zeros which lead to more number of floating point operations.
Basic Cholesky Factorization on Sparse Matrices

- The effect of cholesky factorization after variables are reordered creates the least fill-in.

- The task of variable ordering is to reorder the matrix to create the least fill in.
Matrix Re-ordering

Finding the ordering which results in the least fill-in is a NP-complete problem

Some of the heuristics used are:

- Minimum Degree Reordering (Bottom – up approach)
- Nested Dissection (Top – Down approach)

- These methods gives an idea of sparsity and structure of matrices.
**Elimination Graph**

- Graph $G(A)$ of symmetric $n \times n$ matrix $A$ is an undirected graph having $n$ vertices with edges between vertices $i$ and $j$ if $a_{ij} \neq 0$.

- At each step of Cholesky factorization algorithm, corresponding vertex is eliminated from the graph.
Elimination Graph

- Neighbors of eliminated vertex in previous graph become clique (fully connected subgraph) in modified graph.

- Entries of $A$ that were initially zero, may become non zero entries, called fill.
Elimination Graph

\[
\begin{bmatrix}
\times & \times & \times & \times \\
\times & \times & \times & \times \\
\times & \times & \times & \times \\
\times & \times & \times & \times \\
\end{bmatrix}
\rightarrow
\begin{bmatrix}
\times & \times & ++ & ++ \\
\times & \times & ++ & ++ \\
\times & \times & ++ & ++ \\
\times & \times & ++ & ++ \\
\end{bmatrix}
\]
Minimum Degree Reordering

- Since finding the order of vertices with minimum fill in is a NP-Complete problem

- This is a greedy algorithm such that after each iteration we select a vertex with minimum degree.

- This is a bottom up method trying to minimize fill-in locally and greedily at each step, at the risk of global short sightedness
Nested Dissection

- Form the Elimination Graph.

- Recursively partition the graph into subgraphs using separators, small subsets of vertices the removal of which allows the graph to be partitioned into subgraphs with at most a constant fraction of the number of vertices.

- Perform Cholesky decomposition (a variant of Gaussian elimination for symmetric matrices), ordering the elimination of the variables by the recursive structure of the partition: each of the two subgraphs formed by removing the separator is eliminated first, and then the separator vertices are eliminated.
Preconditioning

- A Preconditioner $P$ of a matrix $A$ is a matrix such that $P^{-1}A$ has a smaller condition number than $A$

- $\kappa(A) = \|A\| \|A^{-1}\|$

- If $P = A$, it gives a single iteration convergence, and finding the pre conditioner is as difficult as solving the linear system
Condition Number

- Defines the ill-conditioning or well-conditioning of a matrix

\[ \kappa(A) = \|A\| \|A^{-1}\| \]

- We cannot trust the solution if the system is ill-conditioned

\[ \kappa(H) = \kappa(J^TJ) = \kappa^2(J) \]
so Hessian has a very large condition number, it requires a good preconditioning for conjugate gradient method

- Rate of convergence increases as the condition number of the matrix decreases
Conjugate Gradient Method

- It is an iterative method to solve a sparse system large enough to be handled by Cholesky decomposition.

- Converges in at most $n$ steps where $n$ is the size of the matrix.
Conjugate Gradient Method

\[
\begin{align*}
  r_0 &:= b - Ax_0 \\
p_0 &:= r_0 \\
k &:= 0 \\
\text{repeat} \\
  \alpha_k &:= \frac{r_k^T r_k}{p_k^T Ap_k} \\
x_{k+1} &:= x_k + \alpha_k p_k \\
r_{k+1} &:= r_k - \alpha_k Ap_k \\
\text{if } r_{k+1} \text{ is sufficiently small then exit loop end if} \\
  \beta_k &:= \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \\
p_{k+1} &:= r_{k+1} + \beta_k p_k \\
k &:= k + 1 \\
\text{end repeat} \\
\text{The result is } x_{k+1}
\end{align*}
\]
Thank You