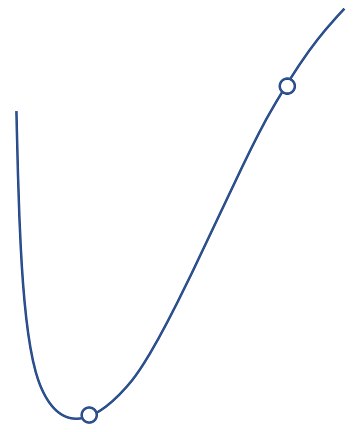
Bundle Adjustment

David Arnon

Bundle Adjustment
Levenberg – Marquardt Algorithm

- The LMA interpolates between the Gauss–Newton algorithm and steepest descent.

- When far from the optimum it acts as a steepest descent and when near the optimum performs Gauss-Newton iterations.



Bundle Adjustment
Levenberg – Marquardt Algorithm

- Quadratic: $\boxed{H\Delta x = -g}$
- Linear: $\Delta x = -\frac{1}{\lambda} g \quad \Rightarrow \quad \boxed{\lambda I \Delta x = -g}$

- $(H + \lambda I)\Delta x = -g$

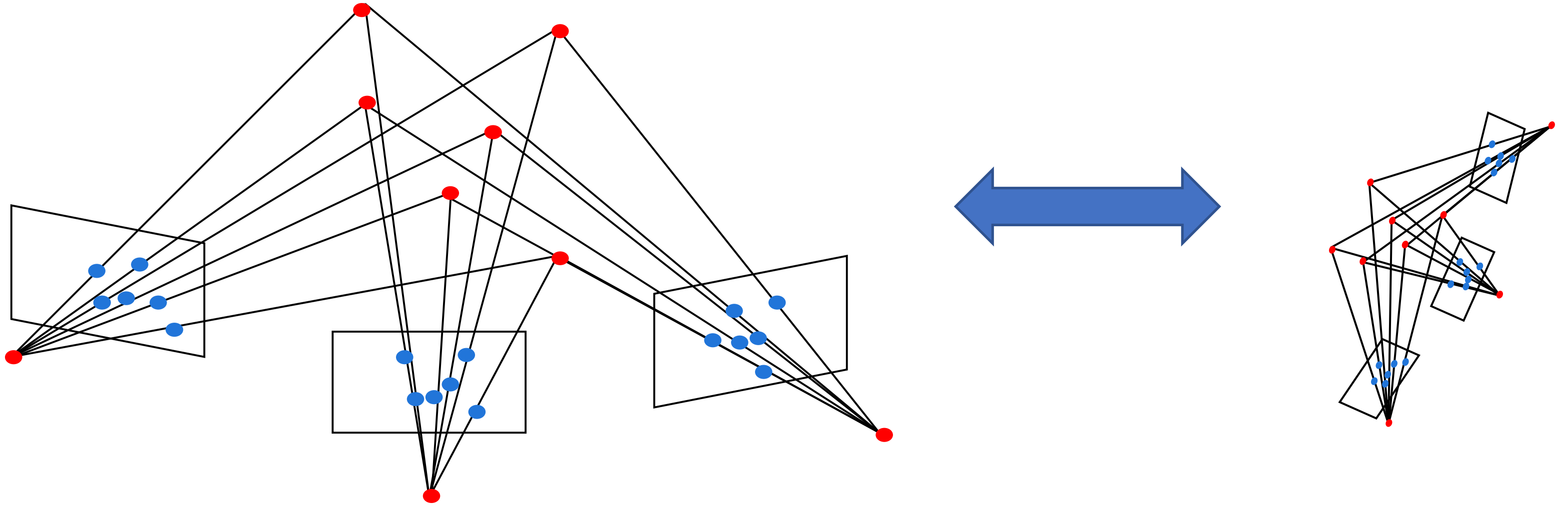
- Small λ $\Rightarrow$ Gauss-Newton step
- Large λ $\Rightarrow$ Gradient decent (and small step)

- Good iteration: decrease λ
  - Larger step, maybe near optimum
- Bad iteration: increase λ
  - Smaller step, overshot optimum

Bundle Adjustment
Degrees of freedom

- Only relative constraints
- 7 degrees of freedom:
    - 3 translation, 3 rotation, 1 scale

Image by Cabe Atwell



Bundle Adjustment
Degrees of freedom

$$\boxed{\lambda p = \mathrm{K}[R|t]\begin{pmatrix} X \\ 1 \end{pmatrix} = \mathrm{K}(RX + t)}$$

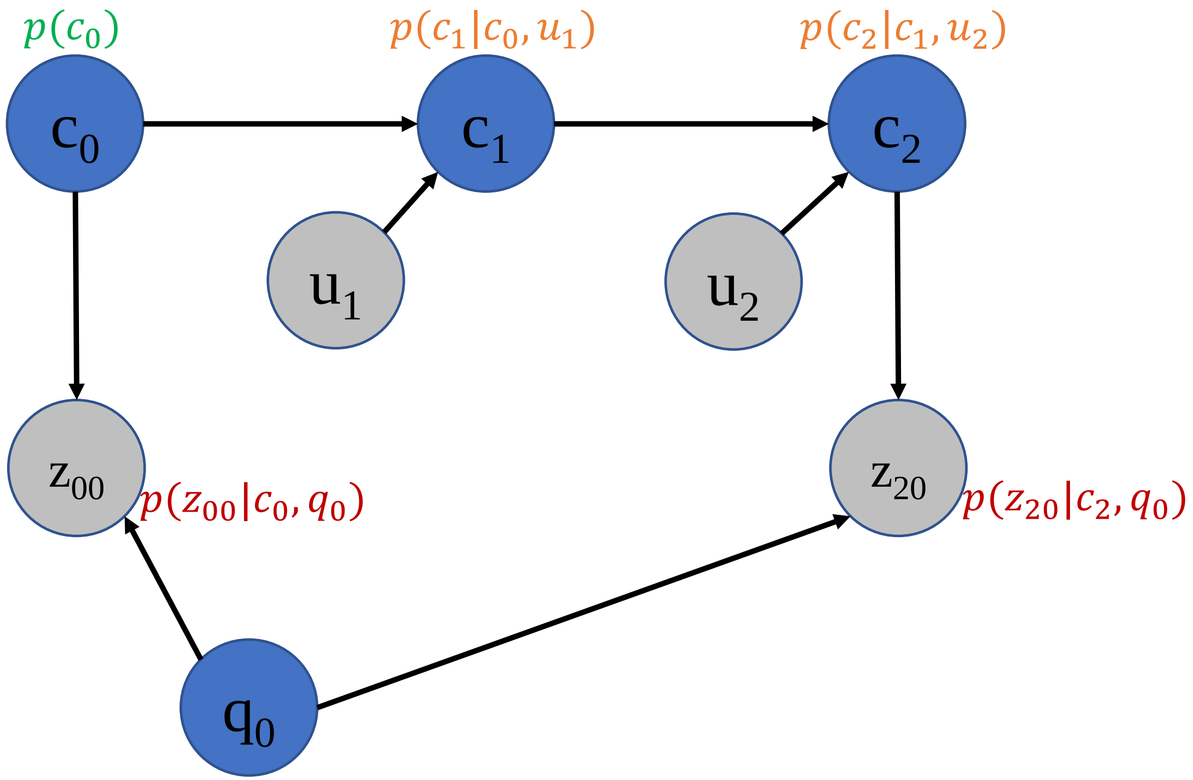
- $\mathrm{K}[R|st]\begin{pmatrix} sX \\ 1 \end{pmatrix}$
  - $\mathrm{K}(sRX + st) = s\mathrm{K}(RX + t) = s\lambda p \propto \lambda p$
- $\mathrm{K}[R|t - Ru]\begin{pmatrix} X + u \\ 1 \end{pmatrix}$
  - $\mathrm{K}(R(X + u) + t - Ru) = \mathrm{K}(RX + Ru + t - Ru) = \mathrm{K}(RX + t) = \lambda p$
- $\mathrm{K}[RQ^T|t]\begin{pmatrix} QX \\ 1 \end{pmatrix}$
  - $\mathrm{K}(RQ^T QX + t) = \mathrm{K}(RX + t) = \lambda p$

Ghostbusters 1984

Bundle Adjustment

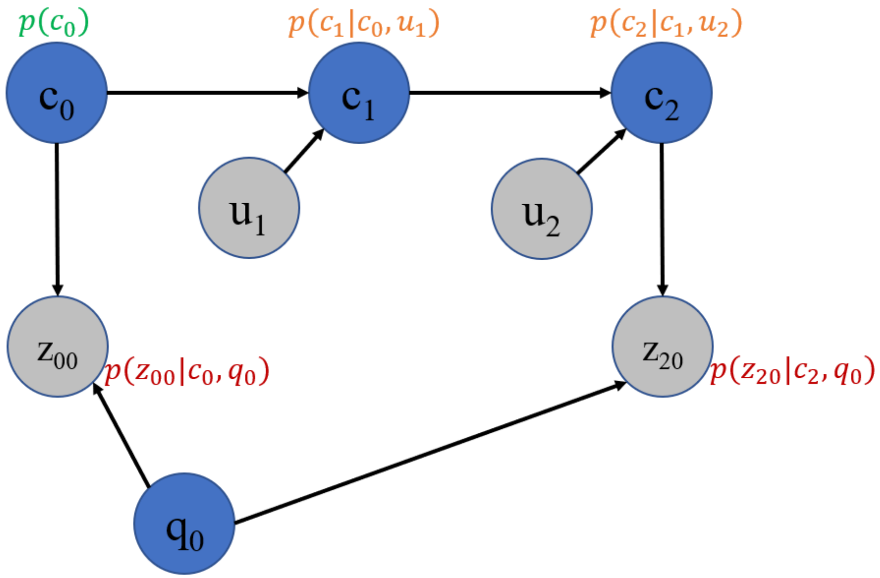
Factor Graph

Graphical Model
Factorization



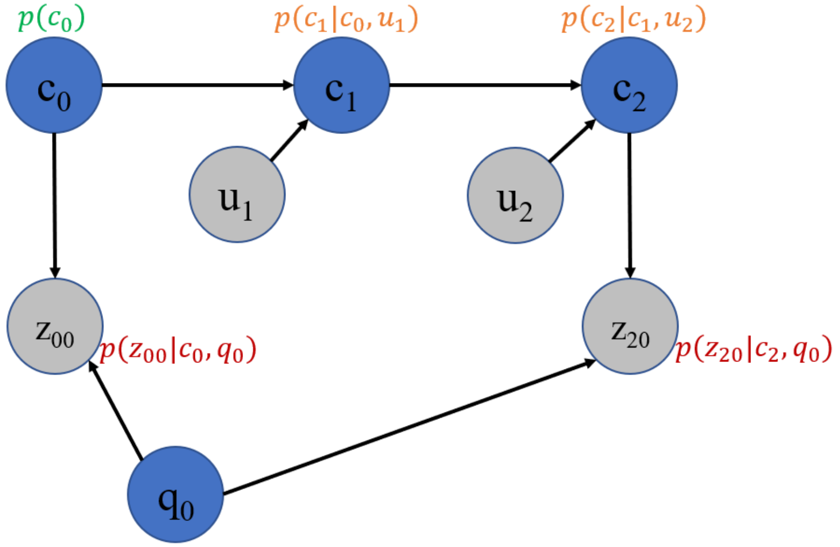
$$p(c_{0:2}, q_0 | u_{1:2}, z_{00}, z_{20}) \propto p(c_0)p(z_{00}|c_0,q_0)p(z_{20}|c_2,q_0)p(c_1|c_0,u_1)p(c_2|c_1,u_2)$$

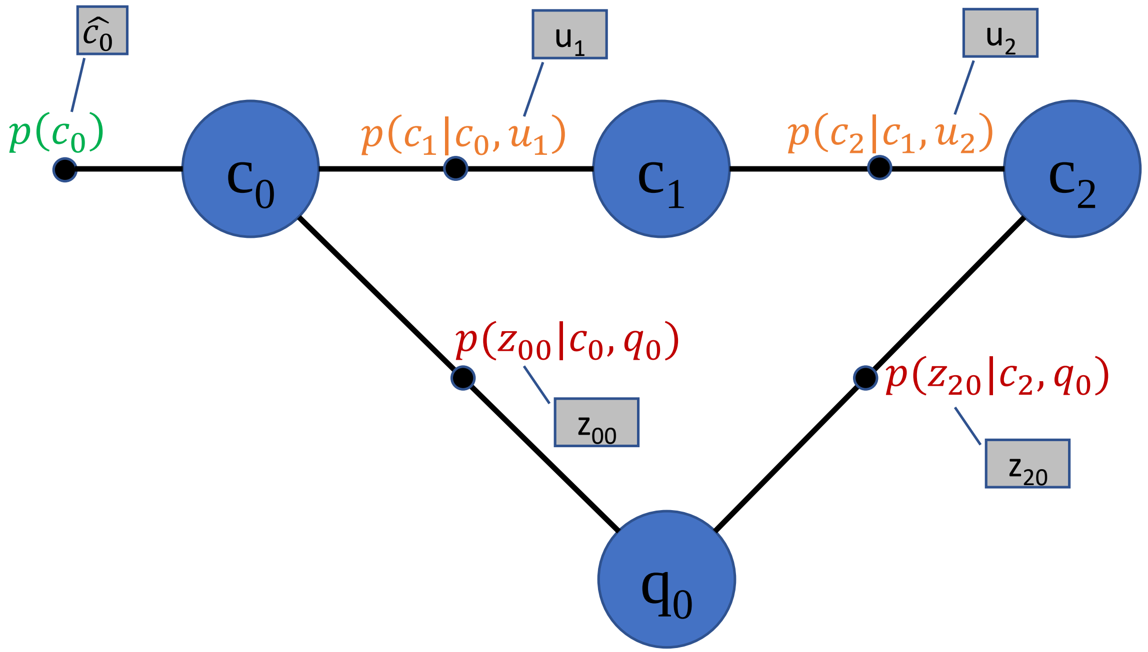
Factorization



- $p(c_{0:2}, q_0|z_{00}, z_{20}, u_{1:2})$
- $= \frac{1}{p(z_{00}, z_{20}|u_{1:2})} p(z_{00}, z_{20}|c_{0:2}, q_0, u_{1:2}) p(c_{0:2}, q_0|u_{1:2})$
- $\propto p(z_{00}, z_{20}|c_{0:2}, q_0, u_{1:2}) p(c_{0:2}, q_0|u_{1:2})$
- $= p(z_{00}|c_0, q_0) p(z_{20}|c_2, q_0) p(c_{0:2}, q_0|u_{1:2})$
- $= p(z_{00}|c_0, q_0) p(z_{20}|c_2, q_0) p(q_0|c_{0:2}, u_{1:2}) p(c_2|c_{0:1}, u_{1:2}) p(c_1|c_0, u_{1:2}) p(c_0|u_{1:2})$
- $\propto p(z_{00}|c_0, q_0) p(z_{20}|c_2, q_0) p(c_2|c_1, u_2) p(c_1|c_0, u_1) p(c_0)$

Factor Graph

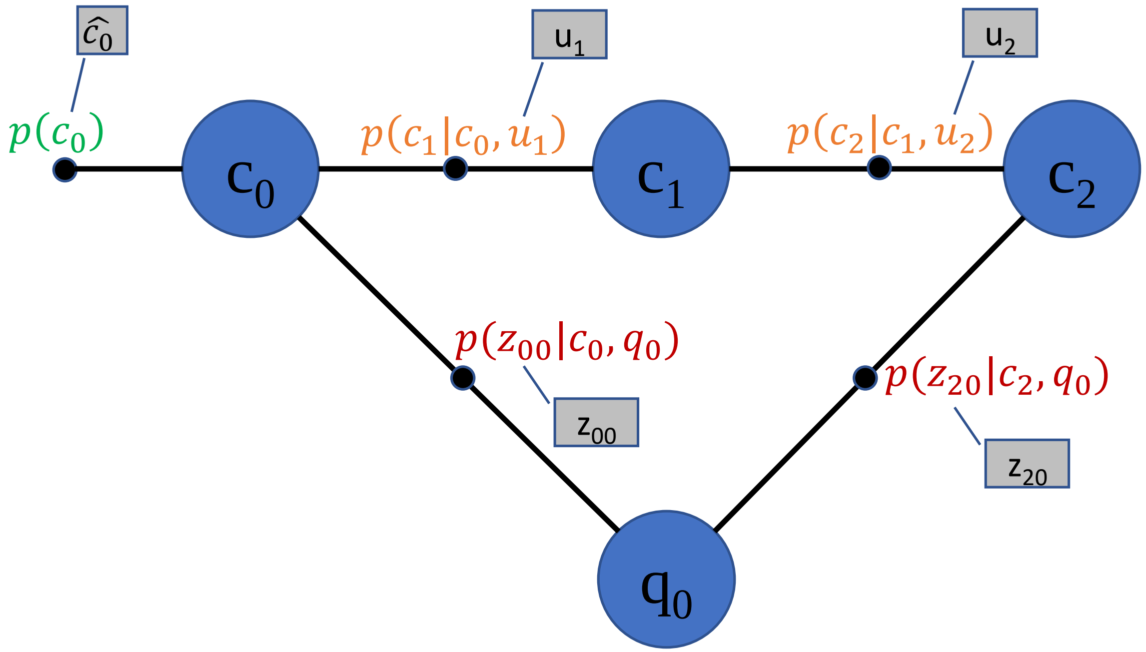


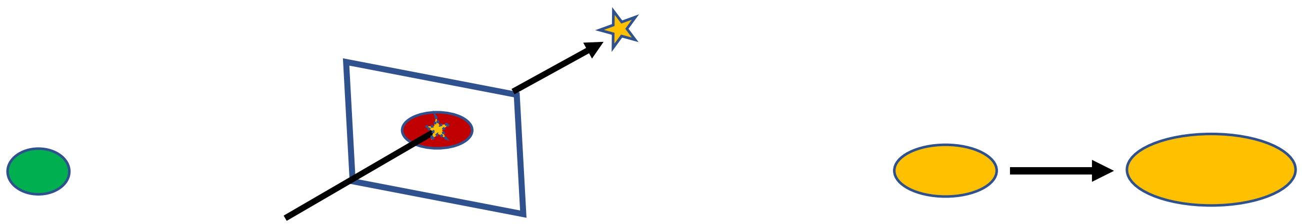


$$p(c_{0:2}, q_0|\, u_{1:2}, z_{00}, z_{20}) \propto p(c_0)p(z_{00}|c_0, q_0)p(z_{20}|c_2, q_0)p(c_1|c_0, u_1)p(c_2|c_1, u_2)$$

Factor Graph

$$f\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ q_0 \end{pmatrix} \doteq \begin{bmatrix} c_0 \\ \pi(c_0, q_0) \\ \pi(c_2, q_0) \\ c_1 - c_0 \\ c_2 - c_1 \end{bmatrix} \stackrel{!}{=} \begin{array}{cc} \begin{bmatrix} \hat{c}_0 \\ z_{00} \\ z_{20} \\ u_1 \\ u_2 \end{bmatrix} & \begin{array}{c} \hat{c}_0 \\ z_{00} \\ z_{20} \\ u_1 \\ u_2 \end{array} \end{array}$$





$$p(c_{0:2}, q_0 | u_{1:2}, z_{00}, z_{20}) \propto p(c_0) p(z_{00}|c_0, q_0) p(z_{20}|c_2, q_0) p(c_1|c_0, u_1) p(c_2|c_1, u_2)$$

Factor Graph

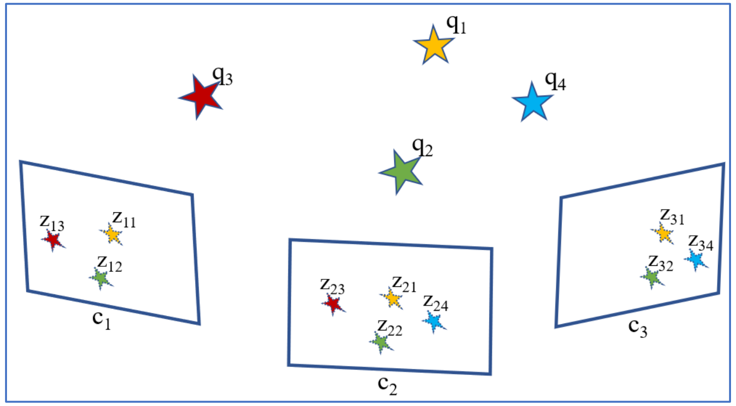
- Assuming Gaussian error, a least squares problem can be constructed given:
  - Measurement z with covariance Σ
  - Measurement function f
  - Jacobian J

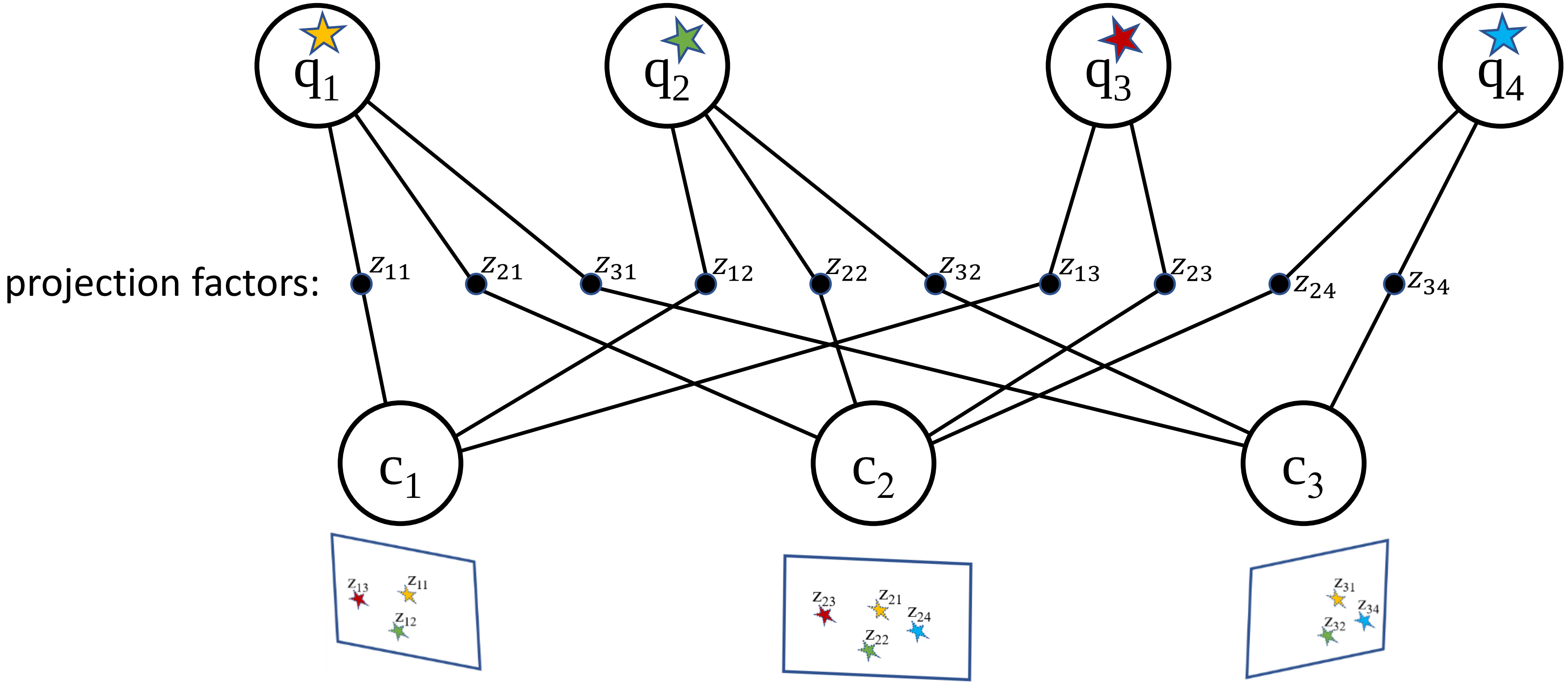
$$\boxed{H\Delta x = -g}$$

$$\boxed{\begin{aligned} H &\doteq J(x_i)^T \Sigma^{-1} J(x_i) \\ g &\doteq J(x_i)^T \Sigma^{-1} \Delta z_i \end{aligned}}$$

$$f\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ q_0 \end{pmatrix} \doteq \begin{bmatrix} c_0 \\ \pi(c_0, q_0) \\ \pi(c_2, q_0) \\ c_1 - c_0 \\ c_2 - c_1 \end{bmatrix} \stackrel{!}{=} \begin{bmatrix} \widehat{c_0} \\ z_{00} \\ z_{20} \\ u_1 \\ u_2 \end{bmatrix}$$

Bundle Adjustment
Factor Graph

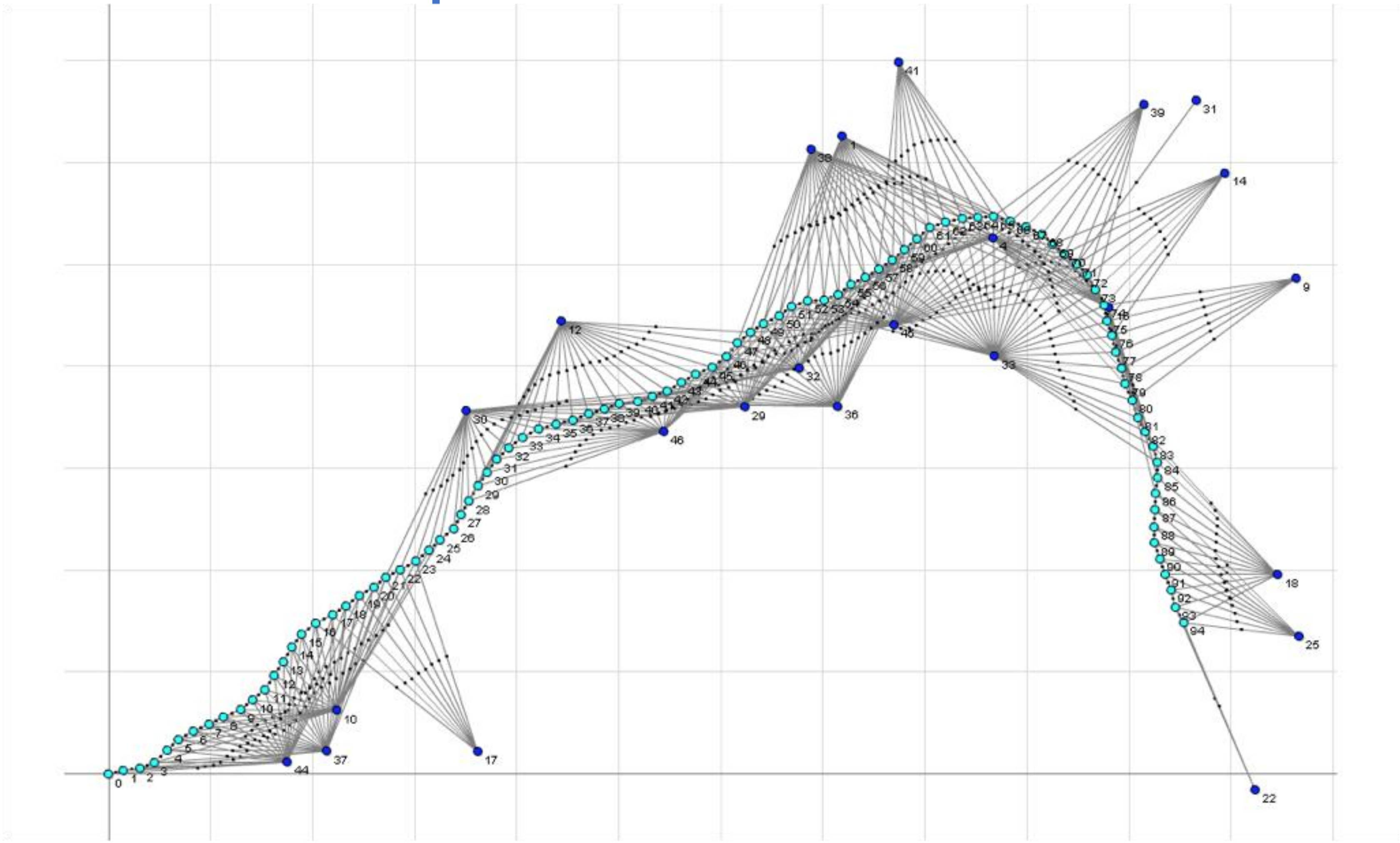




$$f(x) \doteq \begin{bmatrix} \pi(c_1, q_1) \\ \pi(c_1, q_2) \\ \pi(c_1, q_3) \\ \pi(c_2, q_1) \\ \pi(c_2, q_2) \\ \pi(c_2, q_3) \\ \pi(c_2, q_4) \\ \pi(c_3, q_1) \\ \pi(c_3, q_2) \\ \pi(c_3, q_4) \end{bmatrix} \overset{!}{\doteq} \begin{bmatrix} z_{11} \\ z_{12} \\ z_{13} \\ z_{21} \\ z_{22} \\ z_{23} \\ z_{24} \\ z_{31} \\ z_{32} \\ z_{34} \end{bmatrix}$$

Bundle Adjustment

Factor Graph



Courtesy of Dellaert06ijrr: “Square Root SAM”

GTSAM

- https://github.com/borglab/gtsam

What is GTSAM?

GTSAM is a C++ library that implements smoothing and mapping (SAM) in robotics and vision, using Factor Graphs and Bayes Networks as the underlying computing paradigm rather than sparse matrices.

- *g2o*
- *ceres-solver*

GTSAM

https://github.com/borglab/gtsam/blob/develop/python/gtsam/tests/test_StereoVOExample.py

```python
import numpy as np
import gtsam
from gtsam import symbol

## Assumptions
#   - For simplicity this example is in the camera's coordinate frame
#   - X: right, Y: down, Z: forward
#   - Pose x1 is at the origin, Pose 2 is 1 meter forward (along Z-axis)
#   - x1 is fixed with a constraint, x2 is initialized with noisy values
#   - No noise on measurements

## Create keys for variables
x1 = symbol('x',1)
x2 = symbol('x',2)
l1 = symbol('l',1)
l2 = symbol('l',2)
l3 = symbol('l',3)

## Create graph container and add factors to it
graph = gtsam.NonlinearFactorGraph()

## add a constraint on the starting pose
first_pose = gtsam.Pose3()
graph.add(gtsam.NonlinearEqualityPose3(x1, first_pose))

## Create realistic calibration and measurement noise model
# format: fx fy skew cx cy baseline
K = gtsam.Cal3_S2Stereo(1000, 1000, 0, 320, 240, 0.2)
stereo_model = gtsam.noiseModel.Diagonal.Sigmas(np.array([1.0, 1.0, 1.0]))

```

GTSAM

https://github.com/borglab/gtsam/blob/develop/python/gtsam/tests/test_StereoVOExample.py

```python
## Add measurements
# pose 1
graph.add(gtsam.GenericStereoFactor3D(gtsam.StereoPoint2(520, 480, 440), stereo_model, x1, l1, K))
graph.add(gtsam.GenericStereoFactor3D(gtsam.StereoPoint2(120,  80, 440), stereo_model, x1, l2, K))
graph.add(gtsam.GenericStereoFactor3D(gtsam.StereoPoint2(320, 280, 140), stereo_model, x1, l3, K))

#pose 2
graph.add(gtsam.GenericStereoFactor3D(gtsam.StereoPoint2(570, 520, 490), stereo_model, x2, l1, K))
graph.add(gtsam.GenericStereoFactor3D(gtsam.StereoPoint2( 70,  20, 490), stereo_model, x2, l2, K))
graph.add(gtsam.GenericStereoFactor3D(gtsam.StereoPoint2(320, 270, 115), stereo_model, x2, l3, K))

## Create initial estimate for camera poses and landmarks
initialEstimate = gtsam.Values()
initialEstimate.insert(x1, first_pose)
# noisy estimate for pose 2
initialEstimate.insert(x2, gtsam.Pose3(gtsam.Rot3(), gtsam.Point3(0.1,-.1,1.1)))
expected_l1 = gtsam.Point3( 1,  1, 5)
initialEstimate.insert(l1, expected_l1)
initialEstimate.insert(l2, gtsam.Point3(-1,  1, 5))
initialEstimate.insert(l3, gtsam.Point3( 0,-.5, 5))

## optimize
optimizer = gtsam.LevenbergMarquardtOptimizer(graph, initialEstimate)
result = optimizer.optimize()

## check equality for the first pose and point
pose_x1 = result.atPose3(x1)

point_l1 = result.atPoint3(l1)
```

GTSAM

- https://github.com/borglab/gtsam/tree/develop/gtsam/geometry
- *NonlinearFactorGraph*
- *PriorFactorPose3*
- *GenericStereoFactor3D*
- *Values*
- *Pose3*
- *Cal3_S2Stereo*
- *StereoPoint2(uL, uR, v)*
- *noiseModel.Isotropic.Sigma / noiseModel.Diagonal.Sigmas*
- *LevenbergMarquardtOptimizer.optimize*
- *gtsam.utils.plot_3d_points / gtsam.utils.plot_trajectory*

GTSAM

- When the system is undetermined:

```
97      result = optimizer.optimize()

Exception has occurred: RuntimeError ×

Indeterminant linear system detected while working near variable
8646911284551352327 (Symbol: x7).

Thrown when a linear system is ill-posed.  The most common cause for this
error is having underconstrained variables.  Mathematically, the system is
underdetermined.  See the GTSAM Doxygen documentation at
http://borg.cc.gatech.edu/ on gtsam::IndeterminantLinearSystemException for
more information.
  File "/mnt/d/PycharmProjects/gtsam/gtsam_sfm_python.py", line 97, in main
    result = optimizer.optimize()
  File "/mnt/d/PycharmProjects/gtsam/gtsam_sfm_python.py", line 111, in <module>
    main()
```

Factor Graph
Implementing Factors

```
// Holds a polygon 2d point, minimal dimension 2, represented by Vector2d
class PolygonVertex : public g2o::BaseVertex<2,Eigen::Vector2d>
{
public:
    void oplusImpl(const double* u) {
        Eigen::Vector2d::ConstMapType update(u);
        _estimate += update;
    }
};

// Edge for a single point measurement
// Measurement dimension 2, represented by Vector2d, vertex type - PolygonVertex
class PolygonVertexDataEdge : public g2o::BaseUnaryEdge<2, Eigen::Vector2d, PolygonVertex>
{
public:
    void computeError() {
        const PolygonVertex* v = static_cast<const PolygonVertex*>(_vertices[0]);
        _error = v->estimate() - _measurement;
    }

    void linearizeOplus() override {
        // err = v->estimate() - _measurement;
        // d_err_d_p:
        _jacobianOplusXi.setIdentity();
    }
};
```