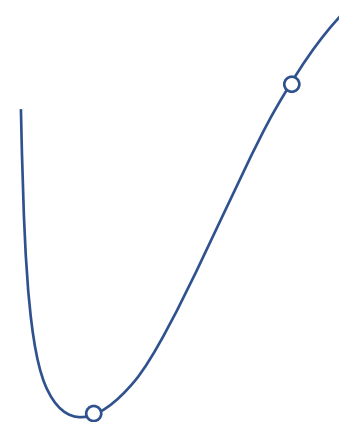# Bundle Adjustment

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

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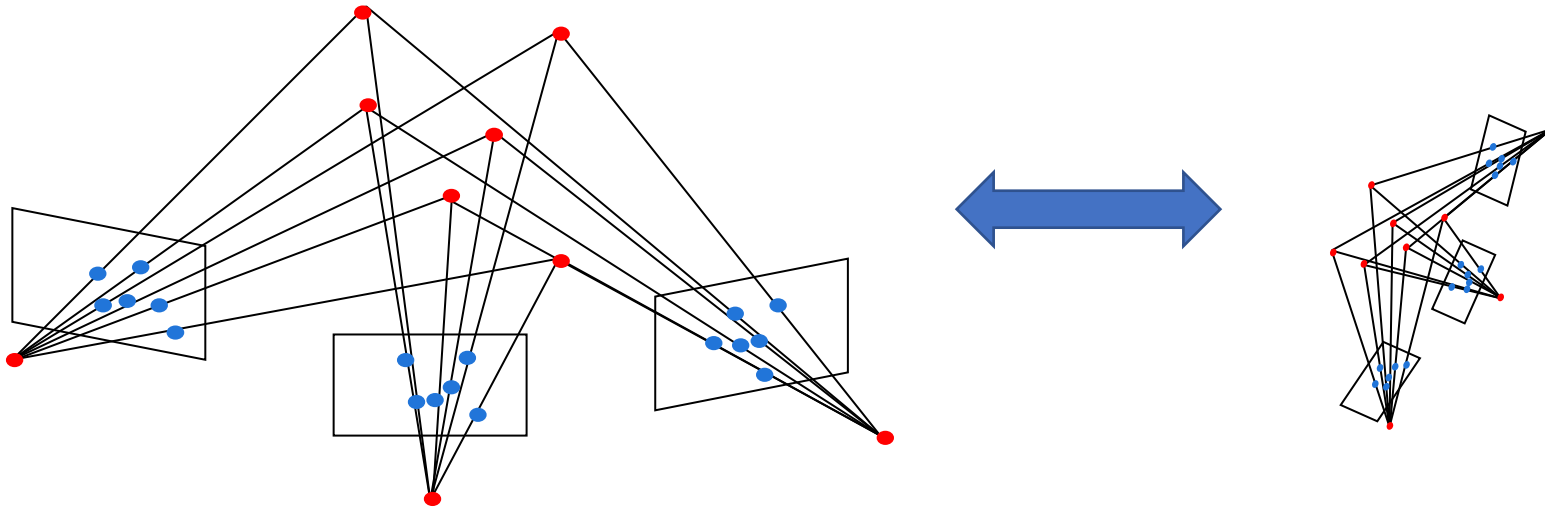- Bad iteration: increase λ
  - Smaller step, overshot optimum

Bundle Adjustment
Degrees of freedom

- Only relative constraints

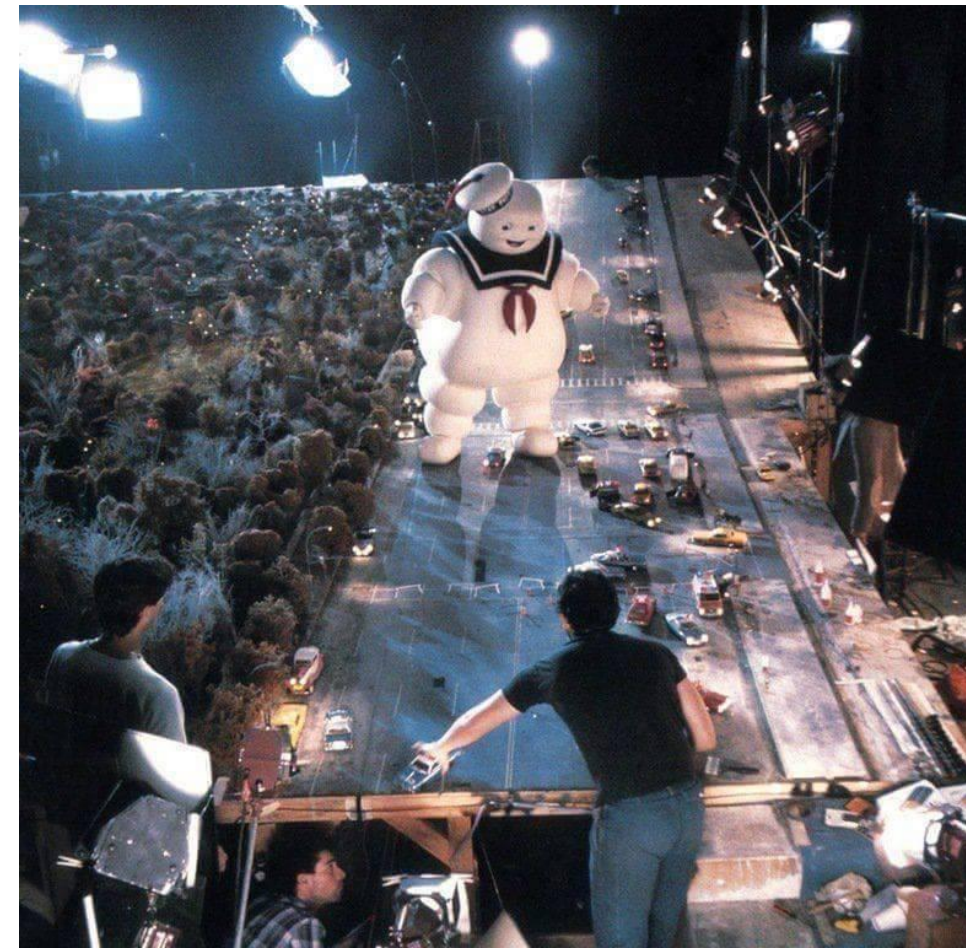- 7 degrees of freedom:
  - 3 translation, 3 rotation, 1 scale

Bundle Adjustment
Degrees of freedom

Ghostbuster 1984

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

- $\mathrm{K}[R|st]\begin{pmatrix} sX \\ 1 \end{pmatrix}$
  - $\mathrm{K}(sRX + st) = s\mathrm{K}(RX + t) = s\lambda p = \lambda p$

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

- $\mathrm{K}[RQ^T|t]\begin{pmatrix} QX \\ 1 \end{pmatrix}$
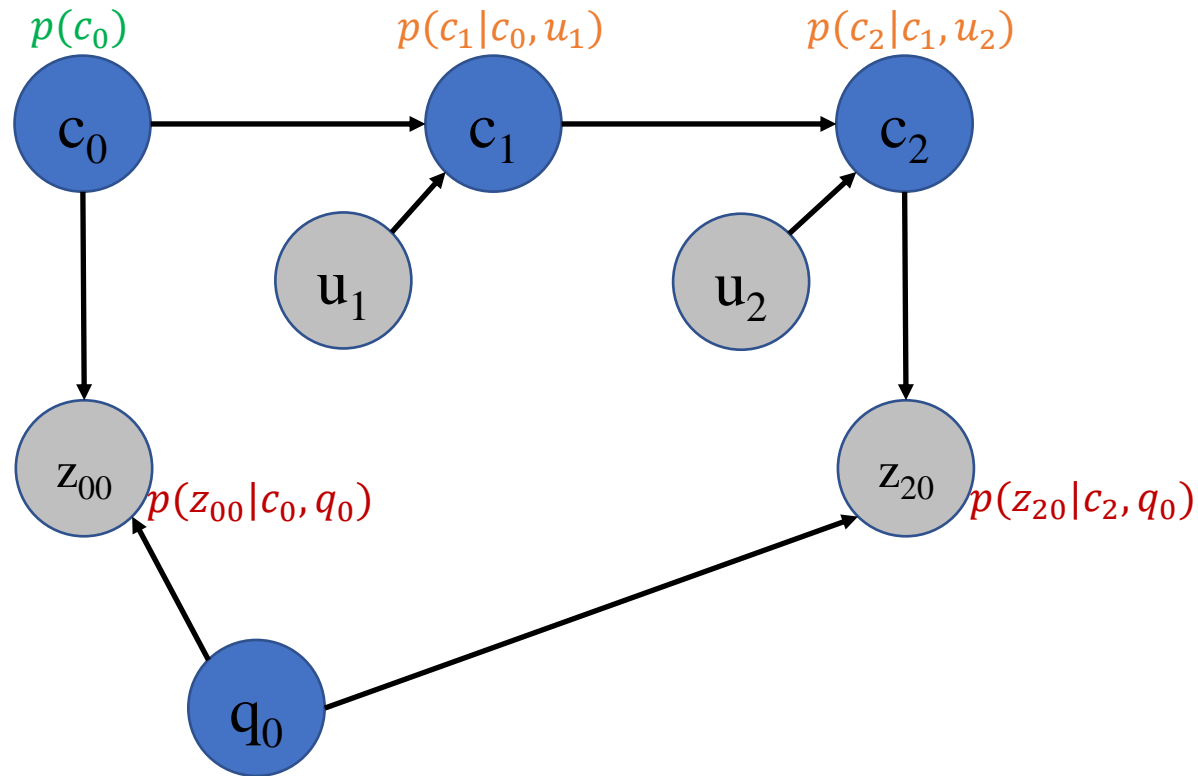 - $\mathrm{K}(RQ^T QX + t) = \mathrm{K}(RX + t) = \lambda p$

Bundle Adjustment
Factor Graph
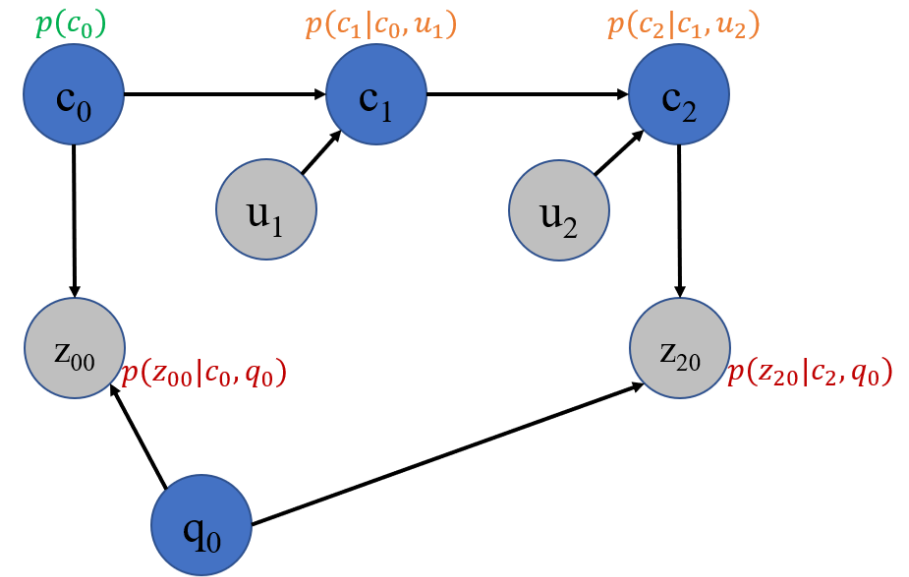
Graphical Model
Factorization

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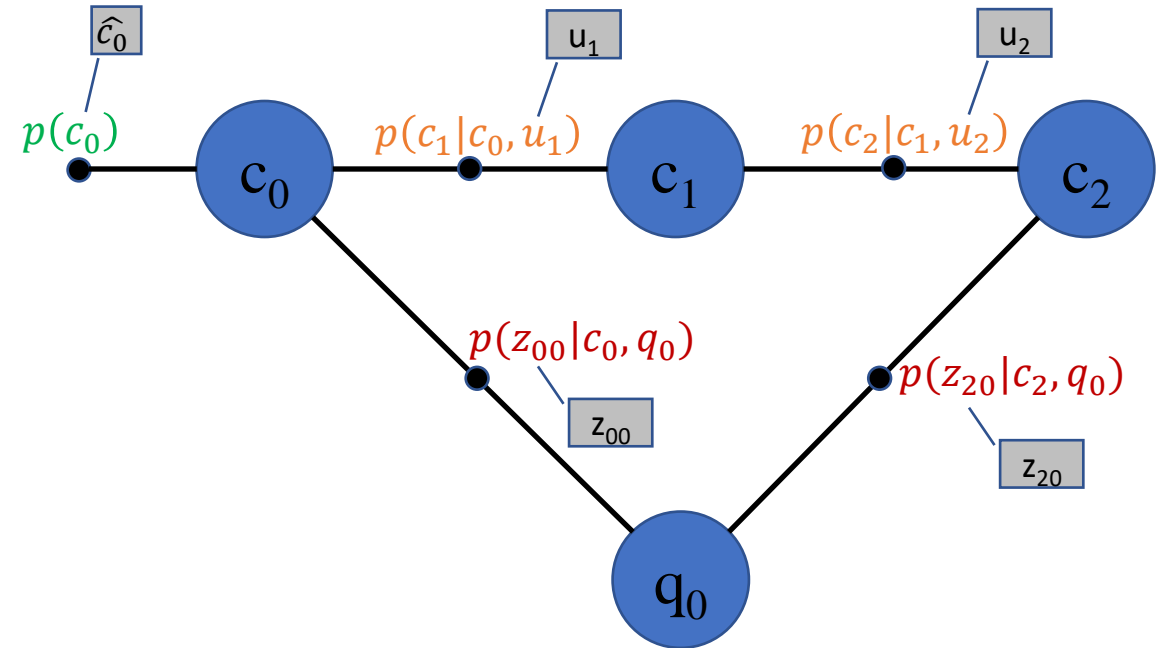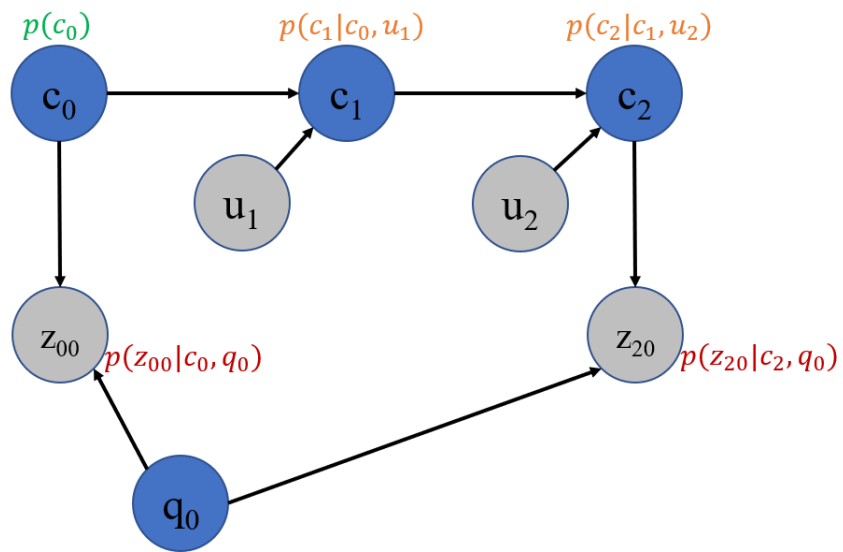
Factorization

- $p(c_{0:2}, q_0 | z_{00}, z_{20}, u_{1:2})$

- $= \frac{1}{p(z_{00}, z_{20}|u_{1:2})} p(z_{00}, z_{20}|c_{0:2}, q_0, u_{1:2}) p(c_{0:2}, q_0|u_{1:2})$

- $\propto p(z_{00}, z_{20}|c_{0:2}, q_0, u_{1:2}) p(c_{0:2}, q_0|u_{1:2})$

- $= p(z_{00}|c_0, q_0) p(z_{20}|c_2, q_0) p(c_{0:2}, q_0|u_{1:2})$

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

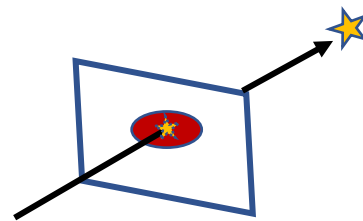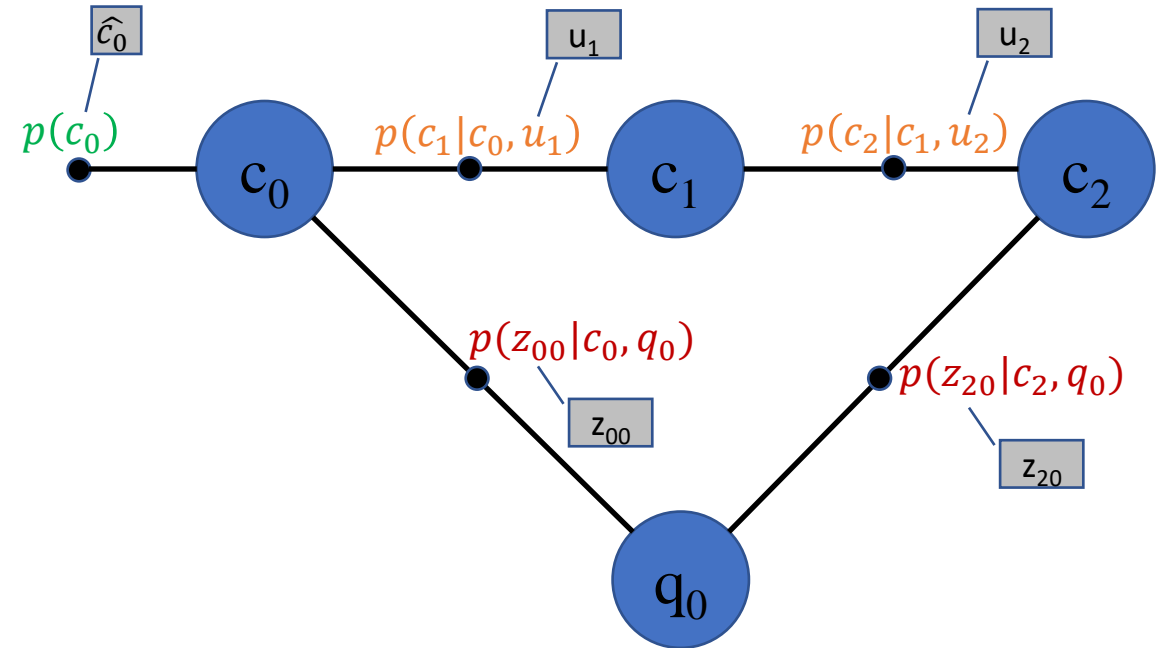- $\propto p(z_{00}|c_0, q_0) p(z_{20}|c_2, q_0) p(c_2|c_1, u_2) p(c_1|c_0, u_1) p(c_0)$

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

$$f\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ q_0 \end{pmatrix} \doteq \begin{bmatrix} c_0 \\ \pi(c_0, q_0) \\ \pi(c_2, q_0) \\ c_1 - c_0 \\ c_2 - c_1 \end{bmatrix} \overset{!}{=} \begin{bmatrix} \widehat{c}_0 \\ z_{00} \\ z_{20} \\ u_1 \\ u_2 \end{bmatrix} \begin{matrix} \widehat{c}_0 \\ z_{00} \\ z_{20} \\ u_1 \\ u_2 \end{matrix}$$

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

Factor Graph

- Assuming Gaussian error,
  a least squares problem can
  be constructed given:

  - Measurement z with covariance Σ
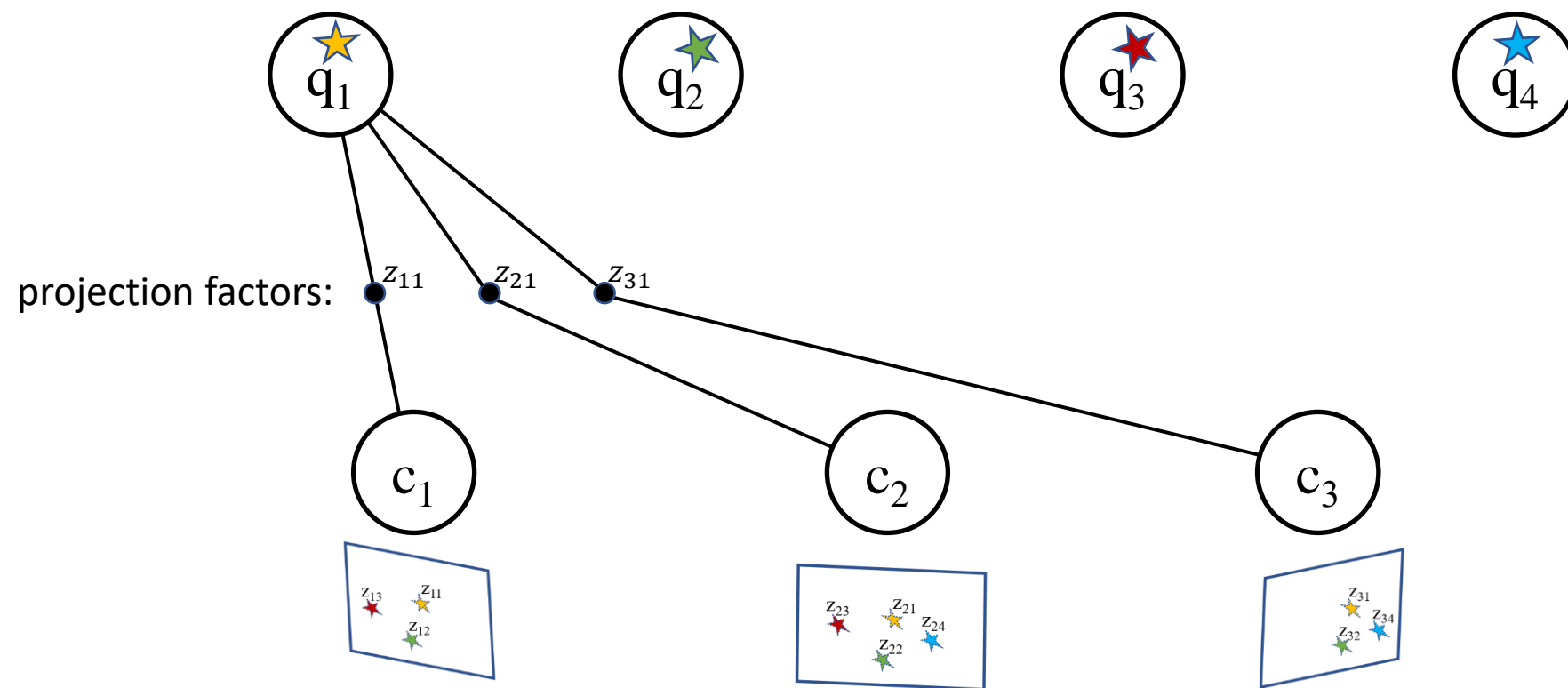
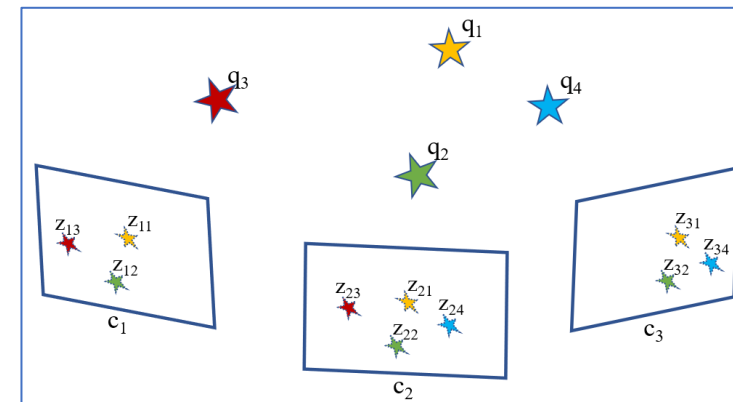 - Measurement function f

  - Jacobian J

$$H \Delta x = -g$$

$$H \doteq J(x_i)^T \Sigma^{-1} J(x_i)$$
$$g \doteq J(x_i)^T \Sigma^{-1} \Delta z_i$$

$$f\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ q_0 \end{pmatrix} \doteq \begin{bmatrix} c_0 \\ \pi(c_0, q_0) \\ \pi(c_2, q_0) \\ c_1 - c_0 \\ c_2 - c_1 \end{bmatrix} \overset{!}{=} \begin{bmatrix} \hat{c}_0 \\ z_{00} \\ z_{20} \\ u_1 \\ u_2 \end{bmatrix}$$
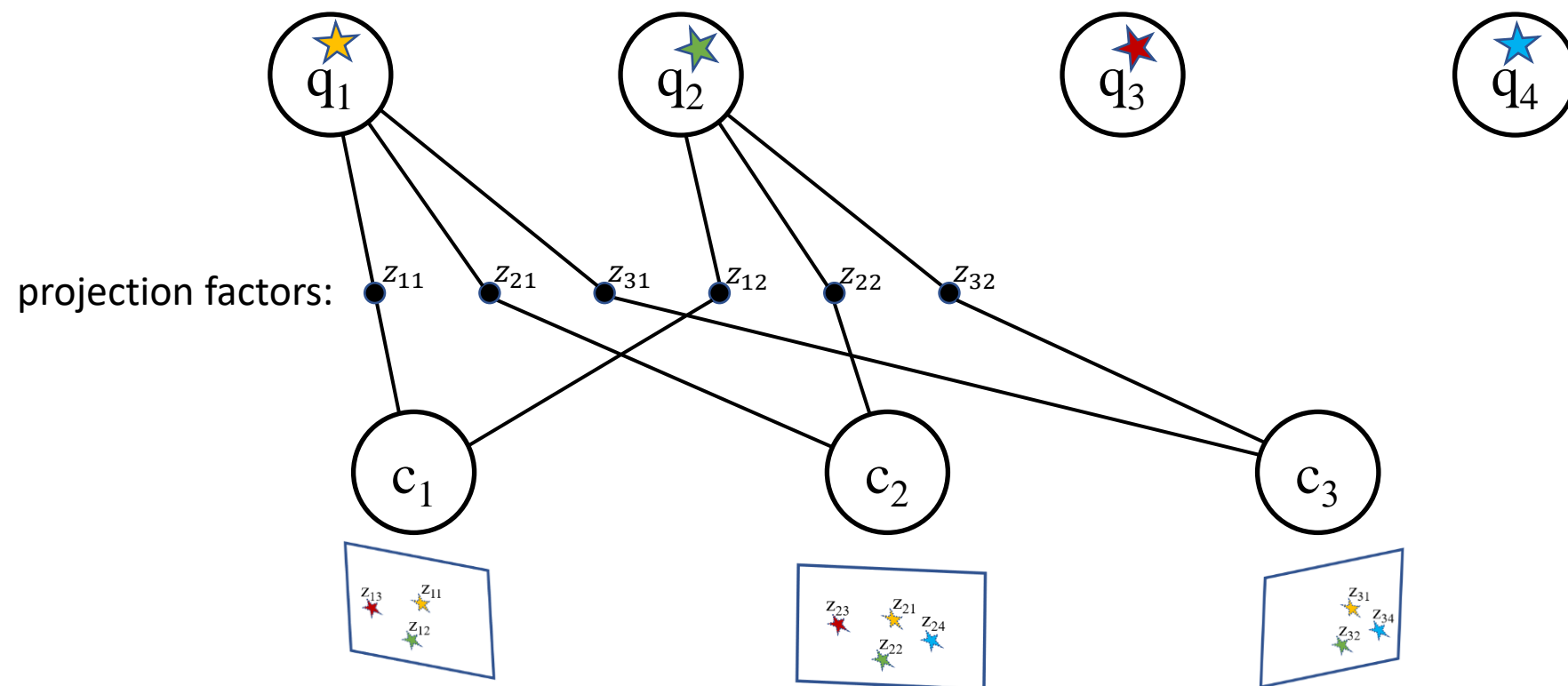
Bundle Adjustment
Factor Graph

projection factors:

$$f(x) \doteq \begin{bmatrix} \pi(c_1, q_1) \\ \\ \pi(c_2, q_1) \\ \\ \pi(c_3, q_1) \end{bmatrix} \overset{!}{=} \begin{bmatrix} z_{11} \\ \\ z_{21} \\ \\ z_{31} \end{bmatrix}$$

Bundle Adjustment
Factor Graph

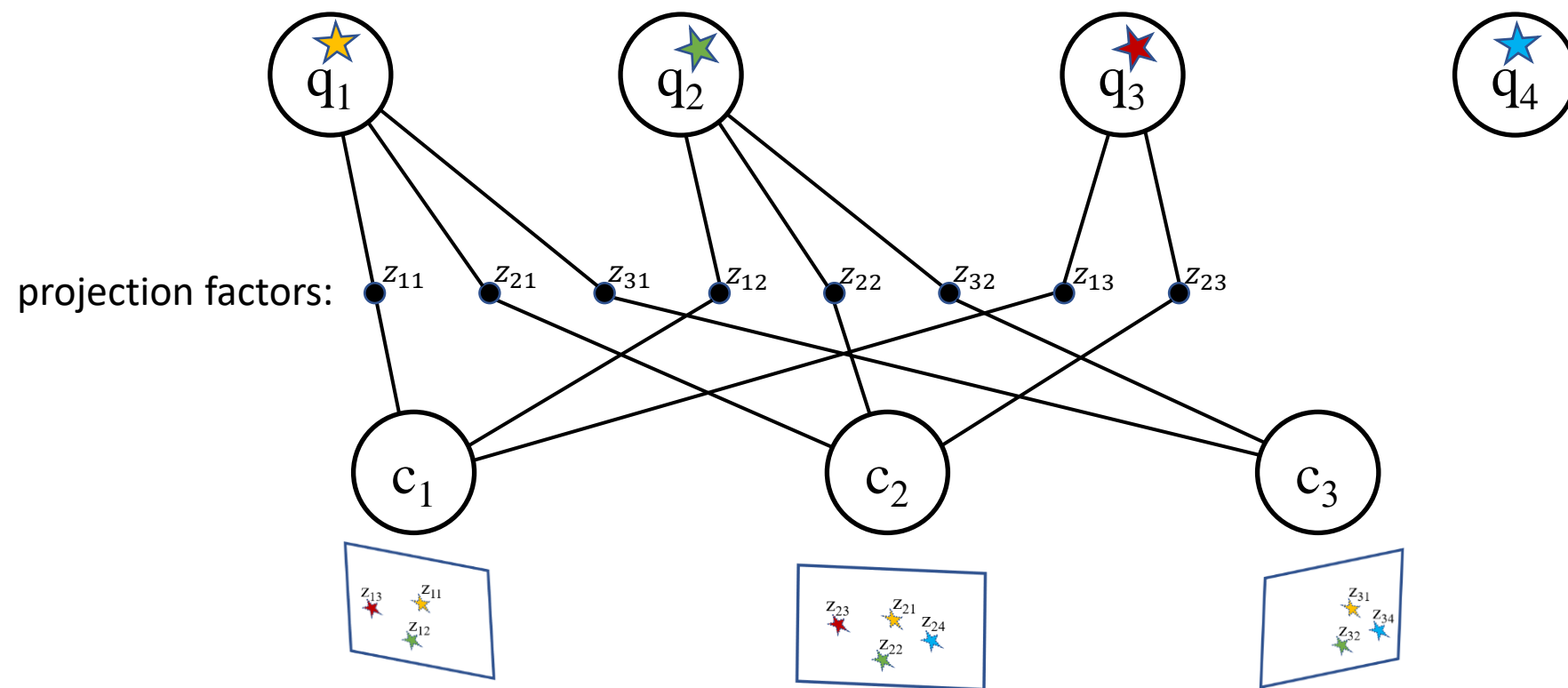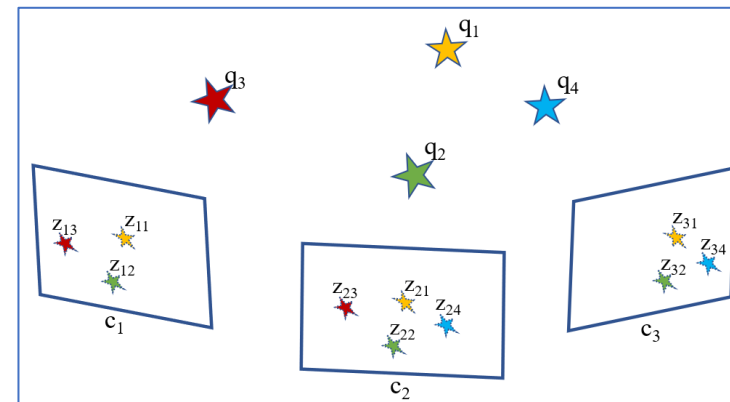projection factors:
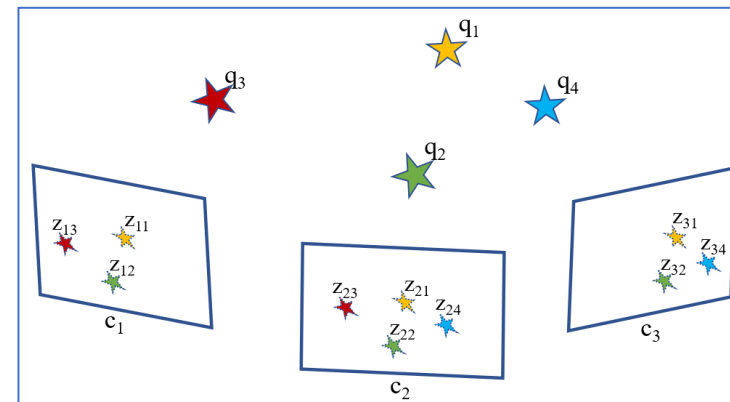
$$f(x) \doteq \begin{bmatrix} \pi(c_1, q_1) \\ \pi(c_1, q_2) \\ \\ \pi(c_2, q_1) \\ \pi(c_2, q_2) \\ \\ \pi(c_3, q_1) \\ \pi(c_3, q_2) \end{bmatrix} \stackrel{!}{=} \begin{bmatrix} z_{11} \\ z_{12} \\ \\ z_{21} \\ z_{22} \\ \\ z_{31} \\ z_{32} \end{bmatrix}$$

Bundle Adjustment
Factor Graph

q_1

q_3

q_4

q_2

z_{13} z_{11}

z_{12}

z_{23} z_{21} z_{24}

z_{22}

z_{31} z_{34}

z_{32}

c_1

c_2

c_3

q_1 q_2 q_3 q_4

projection factors:

z_{11} z_{21} z_{31} z_{12} z_{22} z_{32} z_{13} z_{23}

c_1 c_2 c_3

z_{13} z_{11}

z_{12}

z_{23} z_{21} z_{24}

z_{22}

z_{31} z_{34}

z_{32}

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

Bundle Adjustment
Factor Graph

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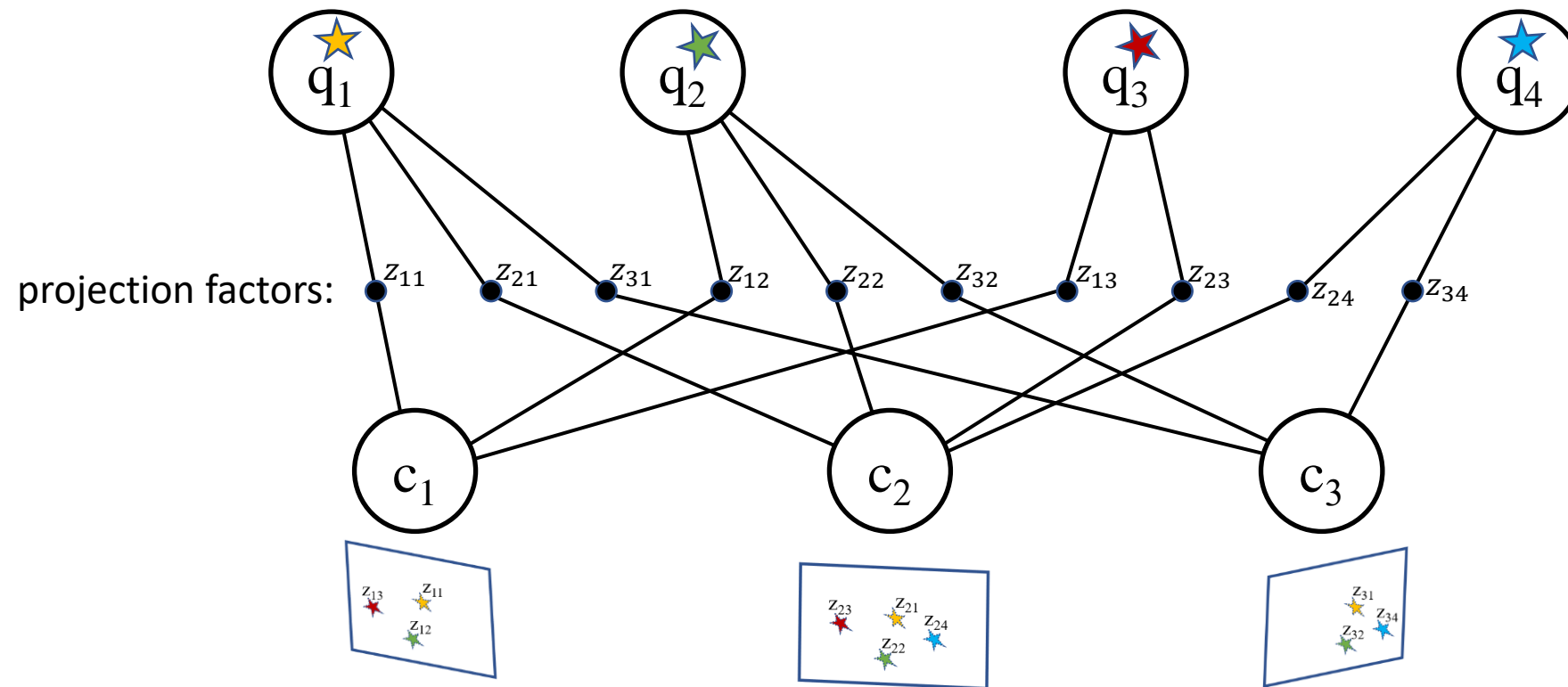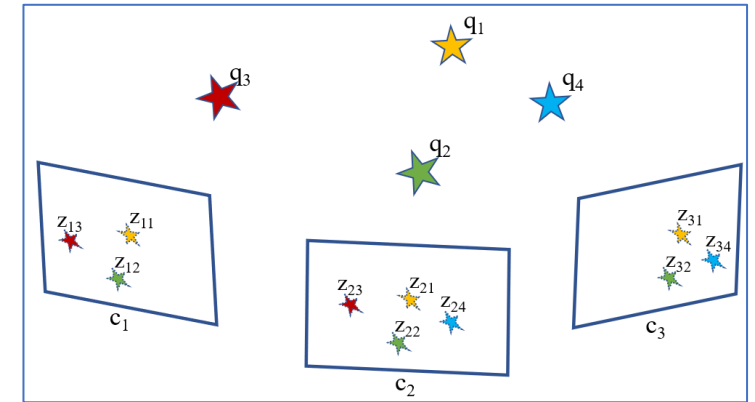
$$f(x) \doteq \begin{bmatrix} \pi(c_1, q_1) \\ \pi(c_1, q_2) \\ \pi(c_1, q_3) \\ \pi(c_2, q_1) \\ \pi(c_2, q_2) \\ \pi(c_2, q_3) \\ \pi(c_2, q_4) \\ \pi(c_3, q_1) \\ \pi(c_3, q_2) \\ \pi(c_3, q_4) \end{bmatrix} \overset{!}{=} \begin{bmatrix} z_{11} \\ z_{12} \\ z_{13} \\ z_{21} \\ z_{22} \\ z_{23} \\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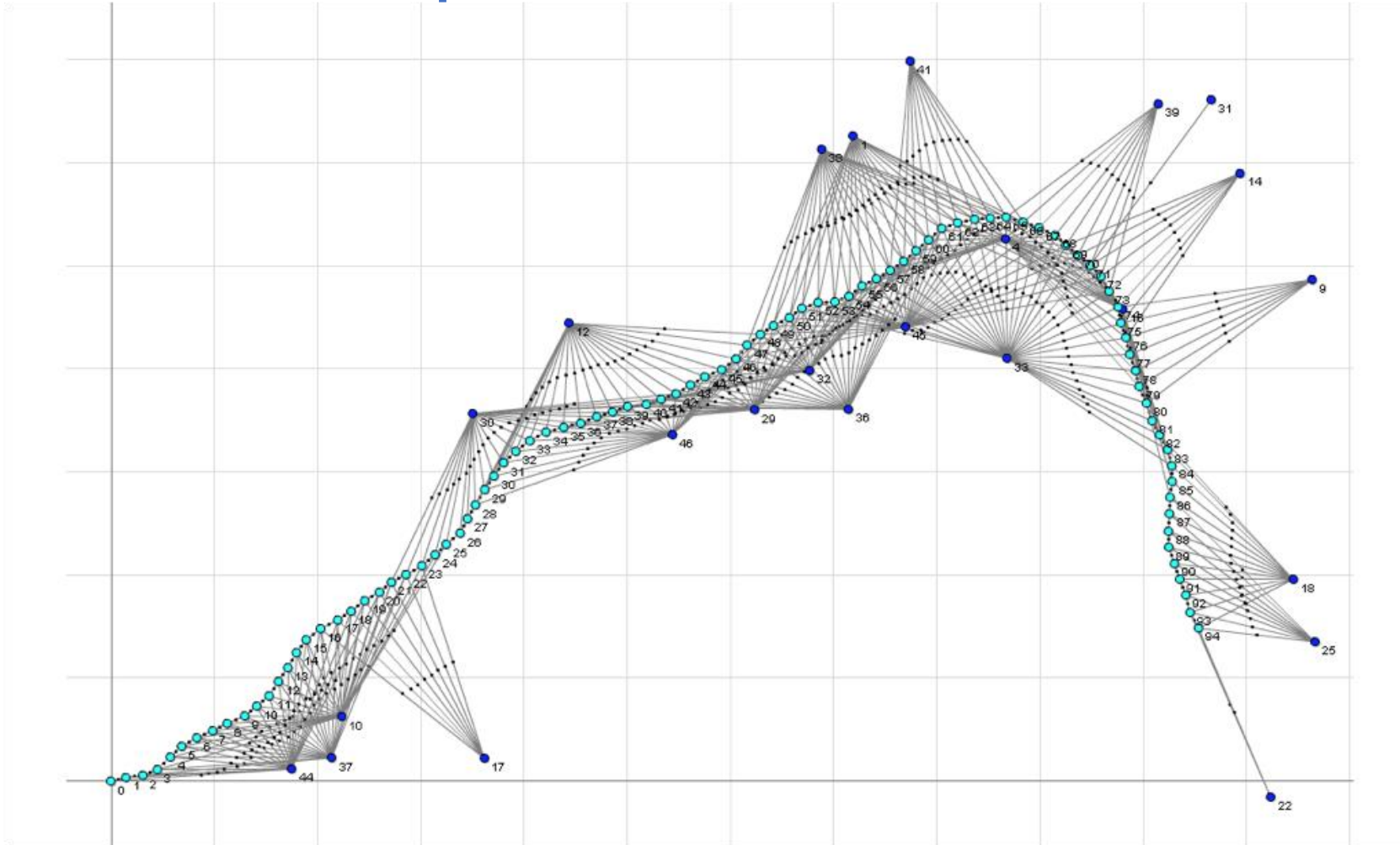
Bundle Adjustment
Factor Graph

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
1   import numpy as np
2   import gtsam
3   from gtsam import symbol
4
5   ## Assumptions
6   #  - For simplicity this example is in the camera's coordinate frame
7   #  - X: right, Y: down, Z: forward
8   #  - Pose x1 is at the origin, Pose 2 is 1 meter forward (along Z-axis)
9   #  - x1 is fixed with a constraint, x2 is initialized with noisy values
10  #  - No noise on measurements
11
12  ## Create keys for variables
13  x1 = symbol('x',1)
14  x2 = symbol('x',2)
15  l1 = symbol('l',1)
16  l2 = symbol('l',2)
17  l3 = symbol('l',3)
18
19  ## Create graph container and add factors to it
20  graph = gtsam.NonlinearFactorGraph()
21
22  ## add a constraint on the starting pose
23  first_pose = gtsam.Pose3()
24  graph.add(gtsam.NonlinearEqualityPose3(x1, first_pose))
25
26  ## Create realistic calibration and measurement noise model
27  # format: fx fy skew cx cy baseline
28  K = gtsam.Cal3_S2Stereo(1000, 1000, 0, 320, 240, 0.2)
29  stereo_model = gtsam.noiseModel.Diagonal.Sigmas(np.array([1.0, 1.0, 1.0]))
30
```

GTSAM

```python
31    ## Add measurements
32    # pose 1
33    graph.add(gtsam.GenericStereoFactor3D(gtsam.StereoPoint2(520, 480, 440), stereo_model, x1, l1, K))
34    graph.add(gtsam.GenericStereoFactor3D(gtsam.StereoPoint2(120,  80, 440), stereo_model, x1, l2, K))
35    graph.add(gtsam.GenericStereoFactor3D(gtsam.StereoPoint2(320, 280, 140), stereo_model, x1, l3, K))
36
37    #pose 2
38    graph.add(gtsam.GenericStereoFactor3D(gtsam.StereoPoint2(570, 520, 490), stereo_model, x2, l1, K))
39    graph.add(gtsam.GenericStereoFactor3D(gtsam.StereoPoint2( 70,  20, 490), stereo_model, x2, l2, K))
40    graph.add(gtsam.GenericStereoFactor3D(gtsam.StereoPoint2(320, 270, 115), stereo_model, x2, l3, K))
41
42    ## Create initial estimate for camera poses and landmarks
43    initialEstimate = gtsam.Values()
44    initialEstimate.insert(x1, first_pose)
45    # noisy estimate for pose 2
46    initialEstimate.insert(x2, gtsam.Pose3(gtsam.Rot3(), gtsam.Point3(0.1,-.1,1.1)))
47    expected_l1 = gtsam.Point3( 1,  1, 5)
48    initialEstimate.insert(l1, expected_l1)
49    initialEstimate.insert(l2, gtsam.Point3(-1,  1, 5))
50    initialEstimate.insert(l3, gtsam.Point3( 0,-.5, 5))
51
52    ## optimize
53    optimizer = gtsam.LevenbergMarquardtOptimizer(graph, initialEstimate)
54    result = optimizer.optimize()
55
56    ## check equality for the first pose and point
57    pose_x1 = result.atPose3(x1)
58
59    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:

Factor Graph
Implementing Factors

```cpp
/// 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);
};


/// 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();
    void linearizeOplus() override;
};
```

Factor Graph
Implementing Factors

```cpp
void g2oPolygon::PolygonVertex::oplusImpl(const double* u)
{
    Eigen::Vector2d::ConstMapType update(u);
    _estimate += update;
}

//===================================================

void g2oPolygon::PolygonVertexDataEdge::computeError()
{
    const PolygonVertex* v = static_cast<const PolygonVertex*>(_vertices[0]);
    _error = v->estimate() - _measurement;
}

void g2oPolygon::PolygonVertexDataEdge::linearizeOplus()
{
    // err = v->estimate() - _measurement;
    // d_err_d_p:
    _jacobianOplusXi.setIdentity();
}
```