CSE276C - Factor Graph for Mapping and Localization

Henrik I. Christensen

Computer Science and Engineering
University of California, San Diego

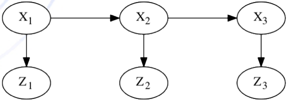
November 2024

Background Material

- F. Dellaert, Factor Graphs and GTSAM: A Hands-on Introduction, GT Tech Report, 2012
- M. Kaess & F. Dellaert, Factor graphs for robot perception, Foundations and Trends in Robotics, 2017.
- Optional – C. Stachniss, Graph SLAM in 90 minutes, University of Bonn. 2015.
- `https://github.com/SLAM-Handbook-contributors/slam-handbook-public-release/blob/main/main.pdf` SLAM HandBook (2024)

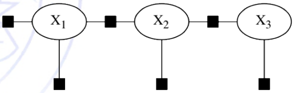
Outline

Factor Graphs



- Simple HMM model for interaction
- Here a 1-order chain

Factor Graphs



- Conversion to a factor graph
- Nodes are functions and arcs causal links with factors that express conditional probabilities

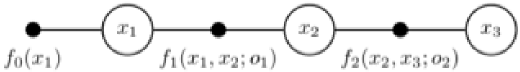
$$P(X1, X2, X3|Z1, Z2, Z3) \propto P(X1)P(X2|X1)P(X3|X2)L(X1; z1)L(X2; z2)L(X3; z3)$$

where $L(X_t; z) \propto P(Z_t = z|Xt)$

Our objective is maximize $f(X1, X2, X3) = \prod_i f_i(X_i)$

Outline

Simple motion



- Consider a simple example of a robot moving
- The uniary factor $f_0(x_1)$ is our prior knowledge about initial position
- The binary factors $f_1(x_1, x_2; o_1)$ and $f_2(x_2, x_3, o_2)$ connect the graph where o_i represents odometric measurements

C++ implementation - Graph Initialization

```cpp
// Create an empty nonlinear factor graph
NonlinearFactorGraph graph;

// Add a Gaussian prior on pose x_1
Pose2 priorMean(0.0, 0.0, 0.0);
noiseModel::Diagonal::shared_ptr priorNoise =
  noiseModel::Diagonal::Sigmas(Vector3(0.3, 0.3, 0.1));
graph.add(PriorFactor<Pose2>(1, priorMean, priorNoise));

// Add two odometry factors
Pose2 odometry(2.0, 0.0, 0.0);
noiseModel::Diagonal::shared_ptr odometryNoise =
  noiseModel::Diagonal::Sigmas(Vector3(0.2, 0.2, 0.1));
graph.add(BetweenFactor<Pose2>(1, 2, odometry, odometryNoise));
graph.add(BetweenFactor<Pose2>(2, 3, odometry, odometryNoise));
```

C++ implementation - Estimating a solution

```
// create (deliberately inaccurate) initial estimate
Values initial;
initial.insert(1, Pose2(0.5, 0.0, 0.2));
initial.insert(2, Pose2(2.3, 0.1, -0.2));
initial.insert(3, Pose2(4.1, 0.1, 0.1));

// optimize using Levenberg-Marquardt optimization
Values result = LevenbergMarquardtOptimizer(graph, initial).optimize();
```

C++ implementation - Results

```
Initial Estimate:
Values with 3 values:
Value 1: (0.5, 0, 0.2)
Value 2: (2.3, 0.1, -0.2)
Value 3: (4.1, 0.1, 0.1)

Final Result:
Values with 3 values:
Value 1: (-1.8e-16, 8.7e-18, -9.1e-19)
Value 2: (2, 7.4e-18, -2.5e-18)
Value 3: (4, -1.8e-18, -3.1e-18)
```

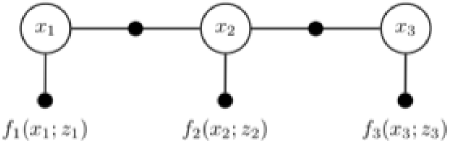
- The correct pose(s) are very well recovered

Outline

Localization

- Odometry alone is not that interesting
- What if there are measurements of landmarks?
- Integration of measurements and world maps into the estimation process
- Assume we get a set of feature measurements - z_i
- We can model the sensors $(f_1(x_1; z_1),\ f_2(x_2; z_2)$, and $f_3(x_3; z_3))$

Setting up the network



- We have to create customized factors for the landmark sensing
- We need to generate the Gaussian likelihood

$$L(q; m) = \exp\left\{-\frac{1}{2}||h(q) - m||^2_\Sigma\right\} = f(q)$$

where m is the measurement and h(q) is the estimate of the feature (q) in the map

Computing h(q)

- The term $h(q)$ is a projection of the feature (q) into robot coordinates.
- Assume $q = (q_x, q_y, q_\theta)^T$
- We can do a "basic calculation"

$$h(q) = \left[\begin{array}{c} q_{xr} \\ q_{yr} \end{array} \right] = Hq = \left[\begin{array}{ccc} \cos(q_\theta) & -\sin(q_\theta) & 0 \\ \sin(q_\theta) & \cos(q_\theta) & 0 \end{array} \right] q$$

C++ Implementation of Custom Factors

```
class UnaryFactor: public NoiseModelFactor1<Pose2> {
  double mx_, my_; ///< X and Y measurements

public:
  UnaryFactor(Key j, double x, double y, const SharedNoiseModel& model):
    NoiseModelFactor1<Pose2>(model, j), mx_(x), my_(y) {}

  Vector evaluateError(const Pose2& q,
                       boost::optional<Matrix&> H = boost::none) const
  {
    const Rot2& R = q.rotation();
    if (H) (*H) = (gtsam::Matrix(2, 3) <<
            R.c(), -R.s(), 0.0,
            R.s(), R.c(), 0.0).finished();
    return (Vector(2) << q.x() - mx_, q.y() - my_).finished();
  }
};
```

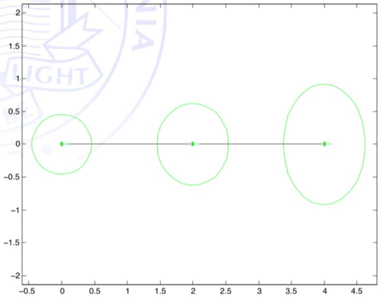
C++ Implementation - use of custom factors

```
// add unary measurement factors, like GPS, on all three poses
noiseModel::Diagonal::shared_ptr unaryNoise =
 noiseModel::Diagonal::Sigmas(Vector2(0.1, 0.1)); // 10cm std on x,y
graph.add(boost::make_shared<UnaryFactor>(1, 0.0, 0.0, unaryNoise));
graph.add(boost::make_shared<UnaryFactor>(2, 2.0, 0.0, unaryNoise));
graph.add(boost::make_shared<UnaryFactor>(3, 4.0, 0.0, unaryNoise));
```

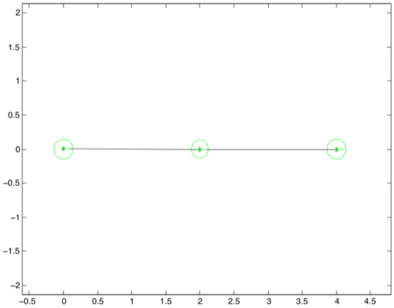
Localization Results

```
Final Result:
Values with 3 values:
Value 1: (-1.5e-14, 1.3e-15, -1.4e-16)
Value 2: (2, 3.1e-16, -8.5e-17)
Value 3: (4, -6e-16, -8.2e-17)

x1 covariance:
      0.0083      4.3e-19     -1.1e-18
     4.3e-19       0.0094      -0.0031
    -1.1e-18      -0.0031       0.0082
x2 covariance:
      0.0071      2.5e-19     -3.4e-19
     2.5e-19       0.0078      -0.0011
    -3.4e-19      -0.0011       0.0082
x3 covariance:
     0.0083     4.4e-19     1.2e-18
    4.4e-19      0.0094      0.0031
    1.2e-18      0.0031       0.018
```

Visualization of localization impact



Odometry results

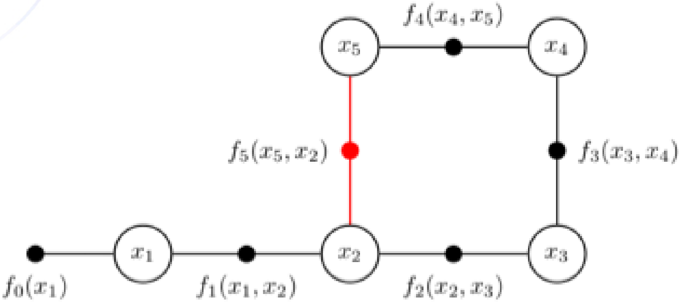


Localization results

Outline

Pose SLAM

- A simple way to combine multiple movements and measurements
- Well described in the literature (Durrant-Whyte et al. 1996)
- A key feature is loop closing. How to update when you return to a former location?

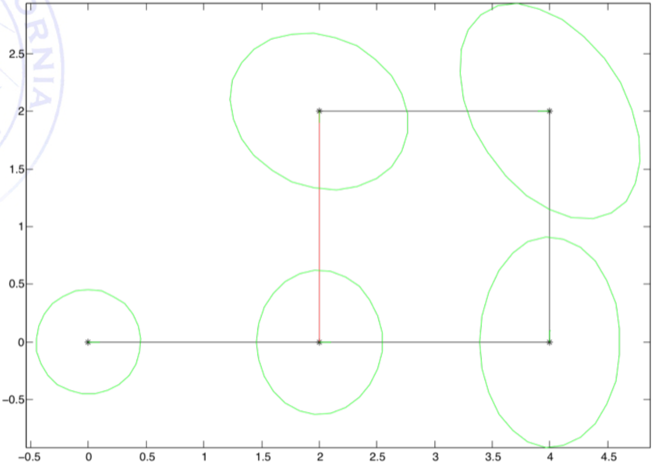


Loop closing

- The estimation process is automatic, once you add the additional link

```
NonlinearFactorGraph graph;
noiseModel::Diagonal::shared_ptr priorNoise =
  noiseModel::Diagonal::Sigmas(Vector3(0.3, 0.3, 0.1));
graph.add(PriorFactor<Pose2>(1, Pose2(0, 0, 0), priorNoise));

// Add odometry factors
noiseModel::Diagonal::shared_ptr model =
  noiseModel::Diagonal::Sigmas(Vector3(0.2, 0.2, 0.1));
graph.add(BetweenFactor<Pose2>(1, 2, Pose2(2, 0, 0     ), model));
graph.add(BetweenFactor<Pose2>(2, 3, Pose2(2, 0, M_PI_2), model));
graph.add(BetweenFactor<Pose2>(3, 4, Pose2(2, 0, M_PI_2), model));
graph.add(BetweenFactor<Pose2>(4, 5, Pose2(2, 0, M_PI_2), model));

// Add the loop closure constraint
graph.add(BetweenFactor<Pose2>(5, 2, Pose2(2, 0, M_PI_2), model));
```

Estimation result with loop closing

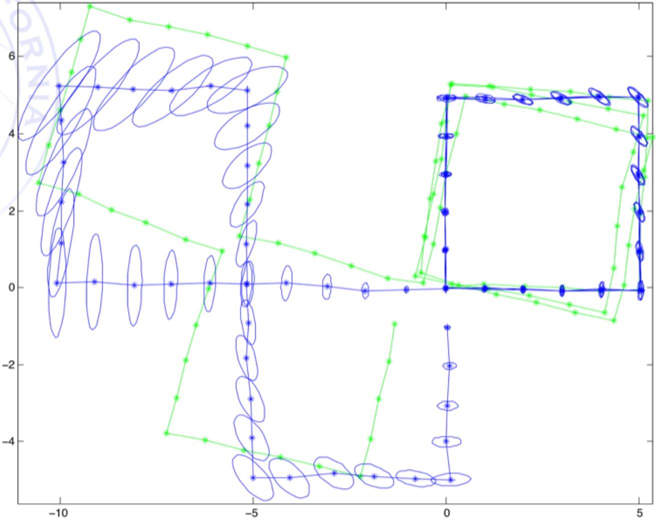


Multiple languages wrapped for GTSAM

- GTSAM has multiple language bindings such as Python, Matlab, ...
- Small MATLAB example below

```
graph = NonlinearFactorGraph;
priorNoise = noiseModel.Diagonal.Sigmas([0.3; 0.3; 0.1]);
graph.add(PriorFactorPose2(1, Pose2(0, 0, 0), priorNoise));

%% Add odometry factors
model = noiseModel.Diagonal.Sigmas([0.2; 0.2; 0.1]);
graph.add(BetweenFactorPose2(1, 2, Pose2(2, 0, 0   ), model));
graph.add(BetweenFactorPose2(2, 3, Pose2(2, 0, pi/2), model));
graph.add(BetweenFactorPose2(3, 4, Pose2(2, 0, pi/2), model));
graph.add(BetweenFactorPose2(4, 5, Pose2(2, 0, pi/2), model));

%% Add pose constraint
graph.add(BetweenFactorPose2(5, 2, Pose2(2, 0, pi/2), model));
```

Small example



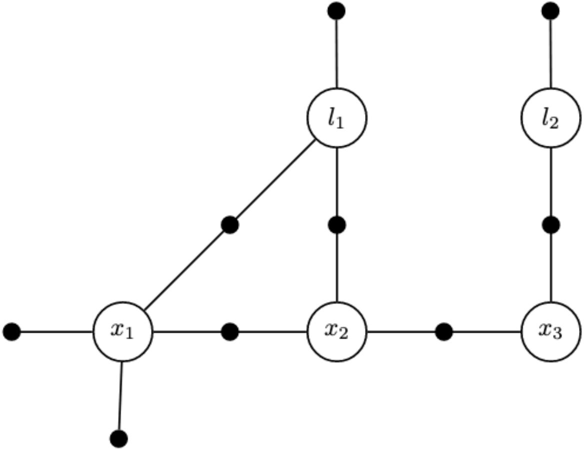
Matlab code for the simple examaple

```
%% Initialize graph, initial estimate, and odometry noise
datafile = findExampleDataFile('w100.graph');
model = noiseModel.Diagonal.Sigmas([0.05; 0.05; 5*pi/180]);
[graph,initial] = load2D(datafile, model);

%% Add a Gaussian prior on pose x_0
priorMean = Pose2(0, 0, 0);
priorNoise = noiseModel.Diagonal.Sigmas([0.01; 0.01; 0.01]);
graph.add(PriorFactorPose2(0, priorMean, priorNoise));

%% Optimize using Levenberg-Marquardt optimization and get marginals
optimizer = LevenbergMarquardtOptimizer(graph, initial);
result = optimizer.optimizeSafely;
marginals = Marginals(graph, result);
```

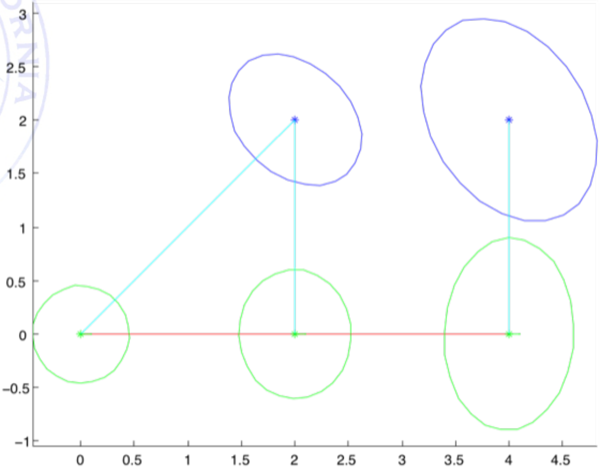
Outline

Landmark-based SLAM

- In many applications we will have a well defined set of landmarks such as doors, windows, traffic signs, ...
- We can use these landmarks as measurements to estimate the robot pose
- The same landmark may be seen multiple times or just a single time.



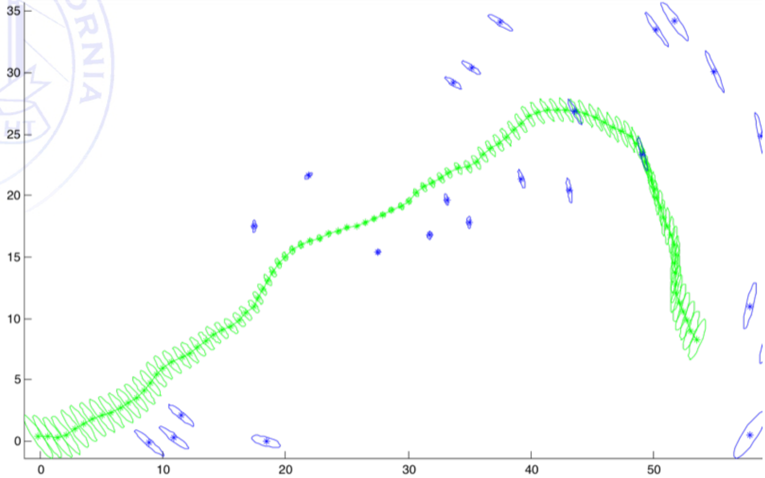
Estimation of the resulting graph



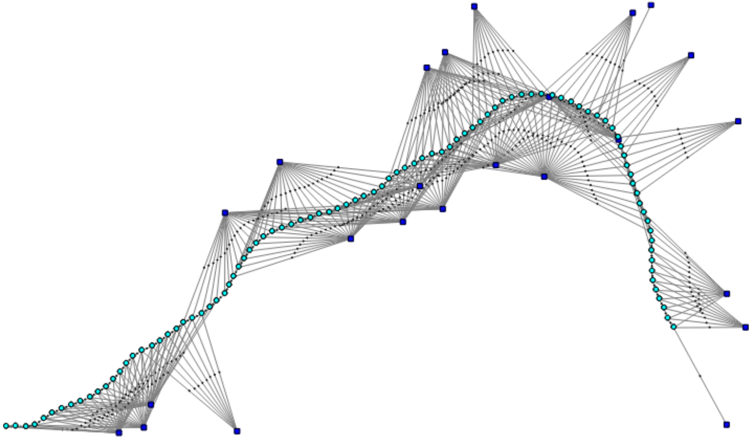
Corresponding Matlab code

```
% Create graph container and add factors to it
graph = NonlinearFactorGraph;

% Create keys for variables
i1 = symbol('x',1); i2 = symbol('x',2); i3 = symbol('x',3);
j1 = symbol('l',1); j2 = symbol('l',2);

% Add prior
priorMean = Pose2(0.0, 0.0, 0.0); % prior at origin
priorNoise = noiseModel.Diagonal.Sigmas([0.3; 0.3; 0.1]);
% add directly to graph
graph.add(PriorFactorPose2(i1, priorMean, priorNoise));

% Add odometry
odometry = Pose2(2.0, 0.0, 0.0);
odometryNoise = noiseModel.Diagonal.Sigmas([0.2; 0.2; 0.1]);
graph.add(BetweenFactorPose2(i1, i2, odometry, odometryNoise));
graph.add(BetweenFactorPose2(i2, i3, odometry, odometryNoise));

% Add bearing/range measurement factors
degrees = pi/180;
brNoise = noiseModel.Diagonal.Sigmas([0.1; 0.2]);
graph.add(BearingRangeFactor2D(i1, j1, Rot2(45*degrees), sqrt(8), brNoise));
graph.add(BearingRangeFactor2D(i2, j1, Rot2(90*degrees), 2, brNoise));
graph.add(BearingRangeFactor2D(i3, j2, Rot2(90*degrees), 2, brNoise));
```

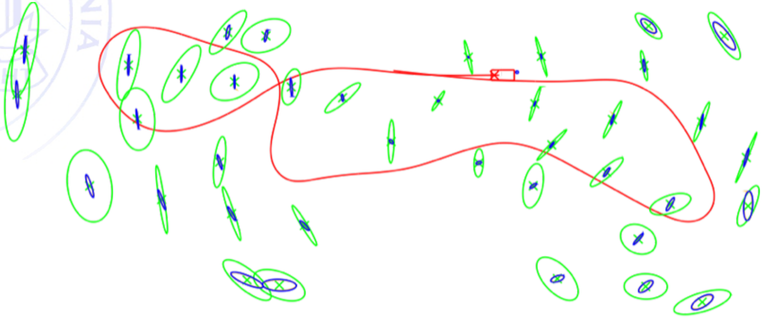
Bigger example 1



Bigger example 1 - Factor Graph



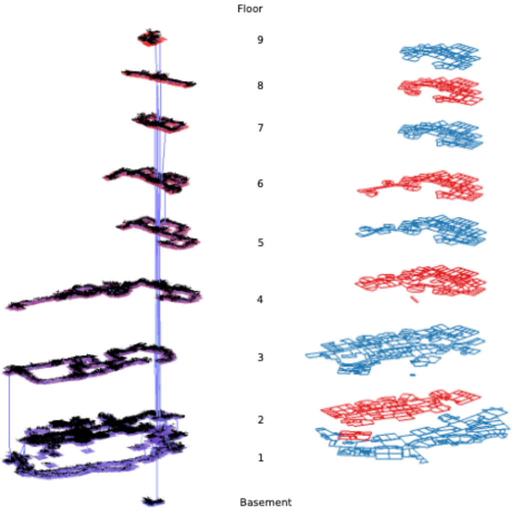
Bigger example 2 - Victoria Park



Computing modes

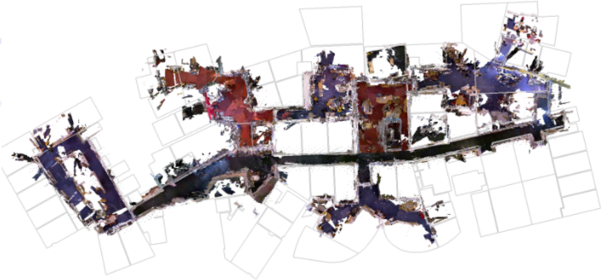
- SAM has multiple computing modes
- Full scale graph optimization
- iSAM which is incremental computing in real-time
- Tactonic SAM which is optimized for use of sub-maps

Hierarchical Maps of MIT Strata Center



The 10 floors of the MIT Strata Center

Sample Map of MIT Strata Center



The 2nd floor of the MIT Strata Center

Outline

Summary

- Factor graphs are a powerful tool for representing and solving estimation problems
- We exemplified its use with a set of examples for basic motion to integration of landmarks
- Today a very widely used tool for many problems in robotics
- Use by most major companies that do mapping and estimation
- Also widely used in computer vision for structure from motion challenge
- FNT paper gives a detailed view of the use-cases and underlying math