EECE 5550
Mobile Robotics

Lecture 11:
Simultaneous Localization and Mapping

Derya Aksaray

Assistant Professor

Department of Electrical and Computer Engineering

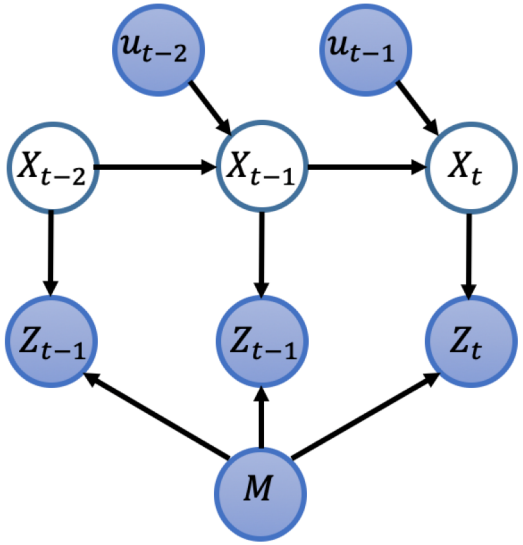
Northeastern University

Recap: Two fundamental problems in robot perception

Localization: Where am I?

Given: Prior $p(x_0)$, map m, controls $u_{0:t-1}$, measurements $z_{1:t}$

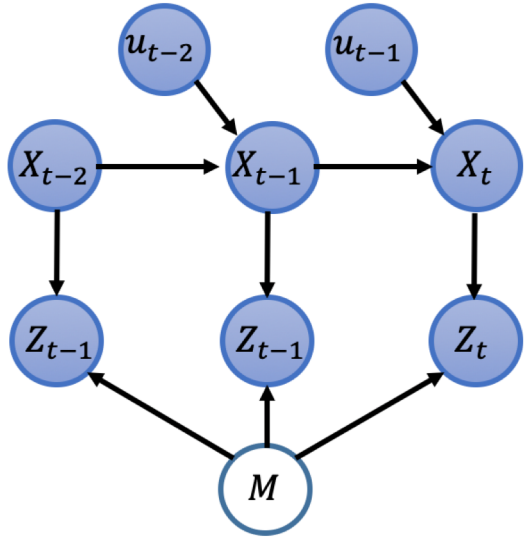
Estimate: Belief $p(x_t | m, u_{0:t-1}, z_{1:t})$ over robot pose



Mapping: What's around me?

Given: Robot poses $x_{0:t}$, measurements $z_{1:t}$

Estimate: Belief $p(m|x_{0:t}, z_{1:t})$ over the map M

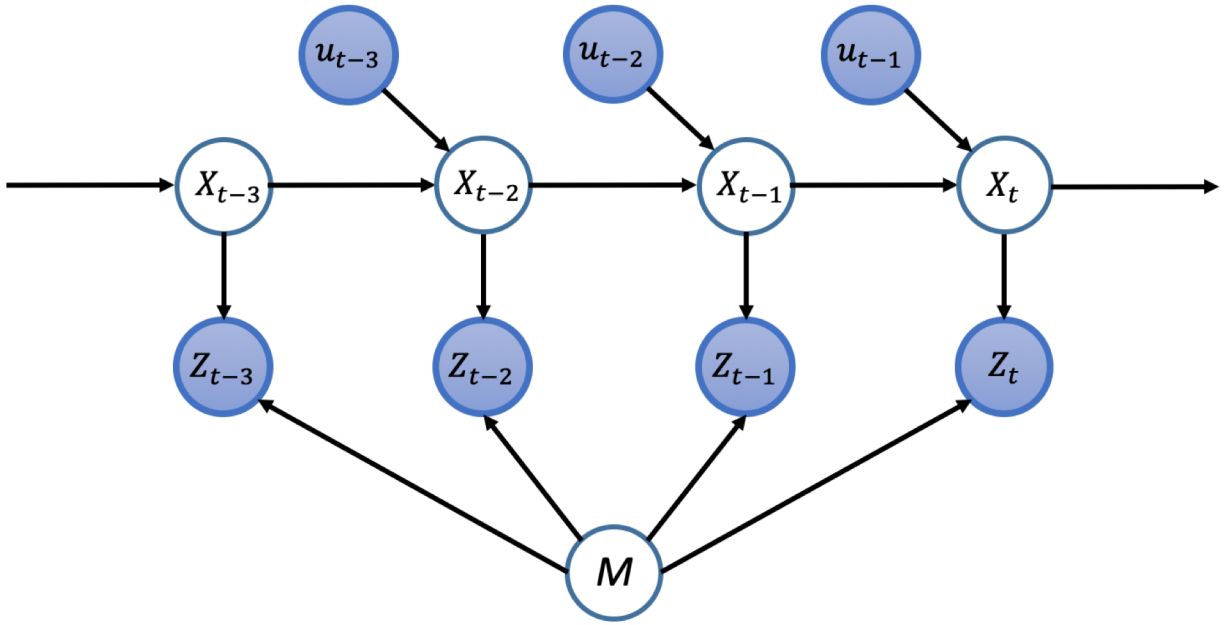


Simultaneous Localization and Mapping (SLAM)

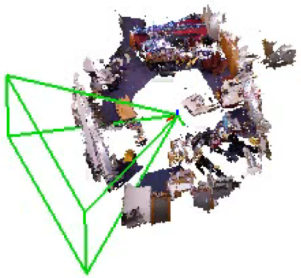
Computing the robot's poses and the map of the environment at the same time

Given: Prior $p(x_0)$, controls $u_{0:t-1}$, measurements $z_{1:t}$

Estimate: Joint belief $p(x_{0:t}, m|u_{0:t-1}, z_{1:t})$ over sequence of robot poses $x_{0:t}$ and map

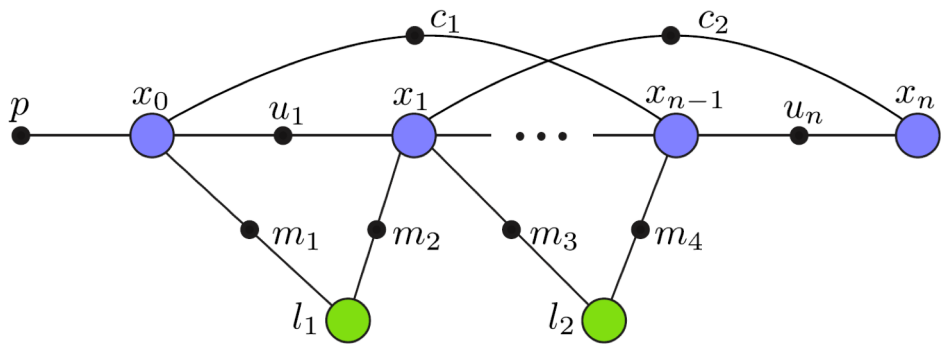


Simultaneous Localization and Mapping (SLAM)



Plan of the day

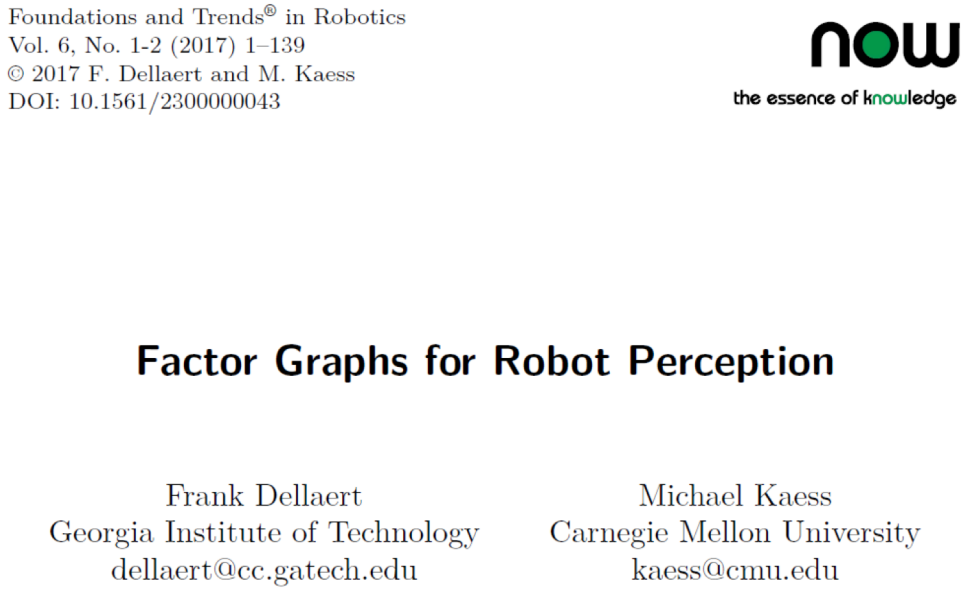
- Motivating example
- SLAM problem formulation
- Factor graphs
- Solving the SLAM problem via maximum-likelihood estimation
- Anatomy of a modern SLAM system
- Practicalities



$$p(Z|\Theta) = \prod_i p_i(Z_i|\,\Theta_i)$$

$$\Theta_i = \{\theta_j \in \Theta \mid (p_i, \theta_j) \in E\}$$

References



https://www.cs.columbia.edu/~allen/F19/NOTES/slam_paper.pdf

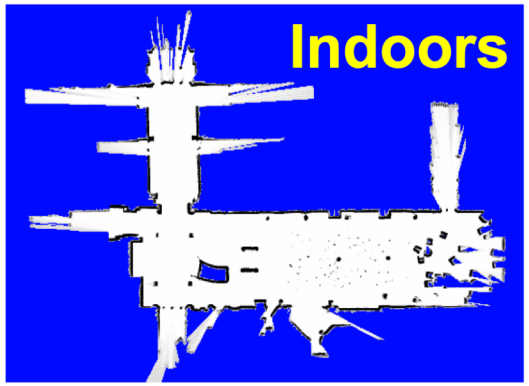
https://theairlab.org/tartanslamseries/

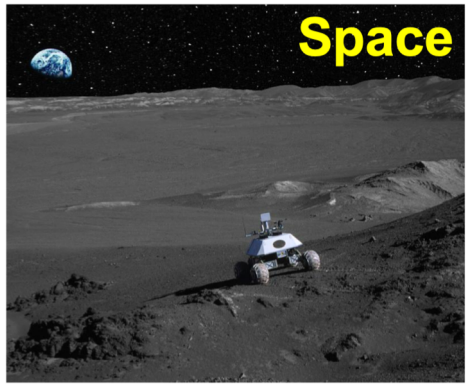
Papers

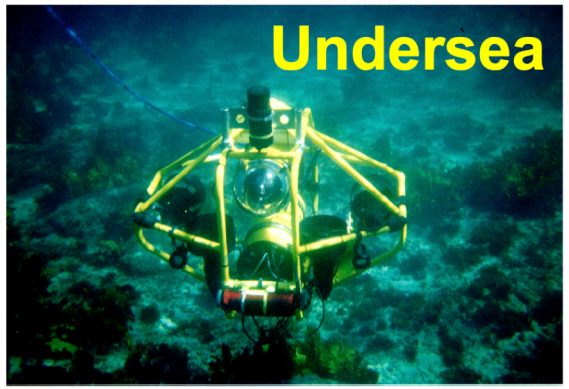
- “Factor Graphs for Robot Perception”
- “Factor Graphs and GTSAM: A Hands-On Introduction”
- “ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras”
- “Bags of Binary Words for Fast Place Recognition in Image Sequences”
- ”Past, Present, and Future of SLAM: Towards the Robust-Perception Age”

Simultaneous Localization and Mapping (SLAM)

- Enables operation in *unknown environments* (exploration)
- An *essential enabling technology* for mobile robots

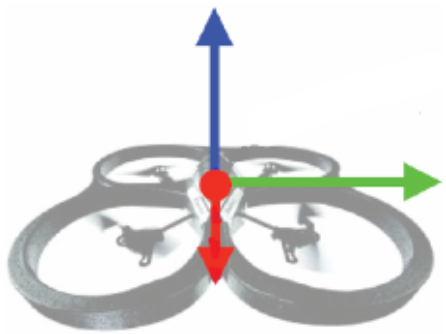




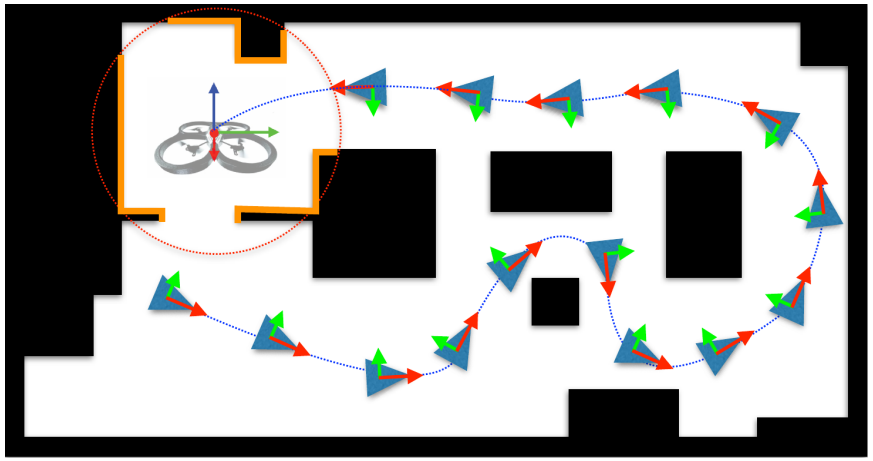


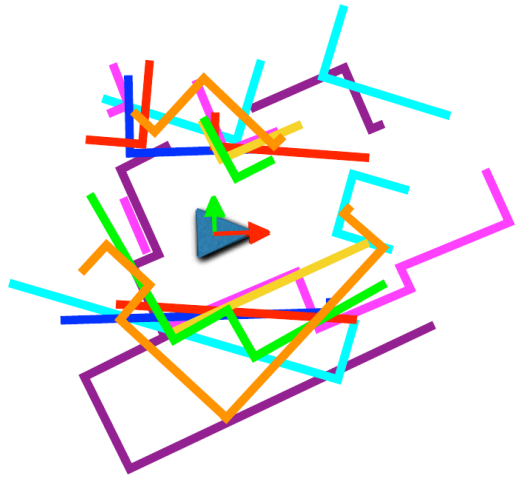
A concrete example

Consider a robot exploring some initially unknown environment …

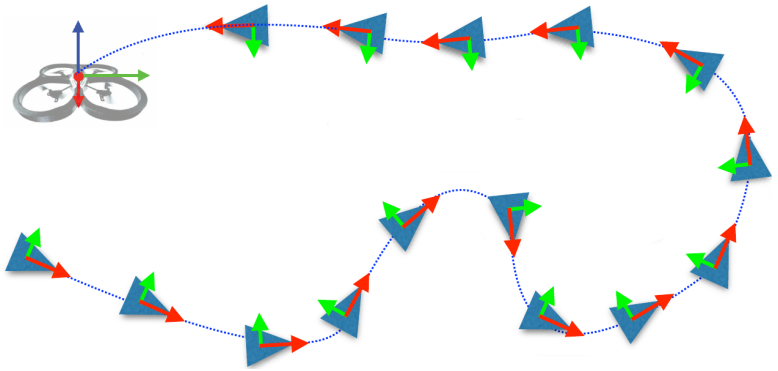


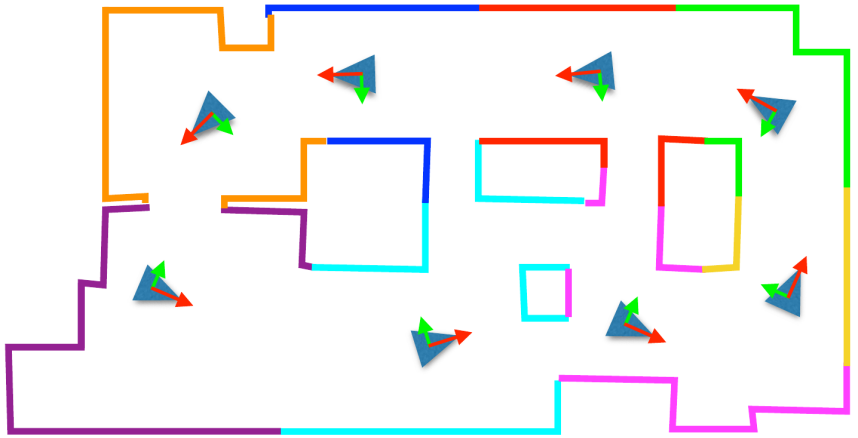
A concrete example





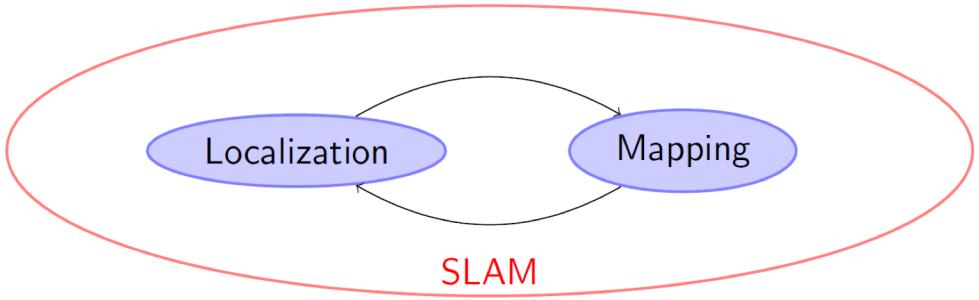
If we knew the positions,





Simultaneous Localization and Mapping (SLAM)

Much harder than localization or mapping alone:



1. Robot path and map are both unknown.
    - Increases the dimension of the problem

Consider a robot moving on a 2D space:

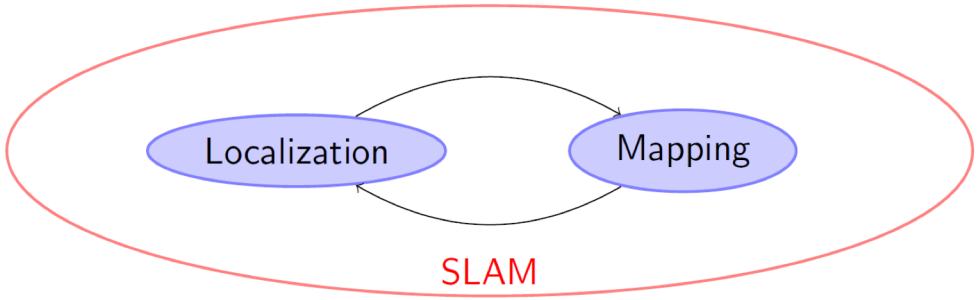
$$x_{Localization} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} \qquad x_{SLAM} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \\ m_1 \\ \vdots \\ m_n \end{bmatrix}$$

where m_i is the coordinate of landmark i

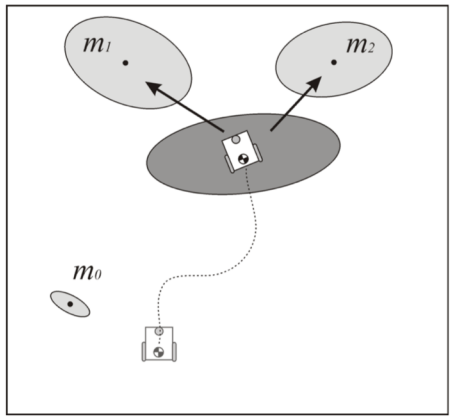
Unknowns at each time t:	3	$3 + 2n$

Simultaneous Localization and Mapping (SLAM)

Much harder than localization or mapping alone:

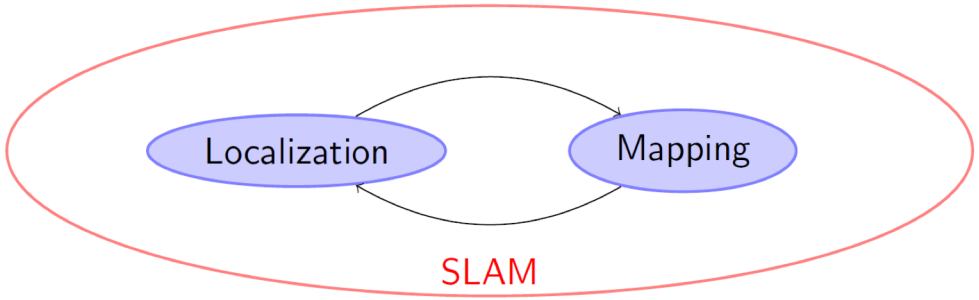


1. Robot path and map are both unknown.
   - Increases the dimension of the problem
2. Map and pose estimates are correlated.

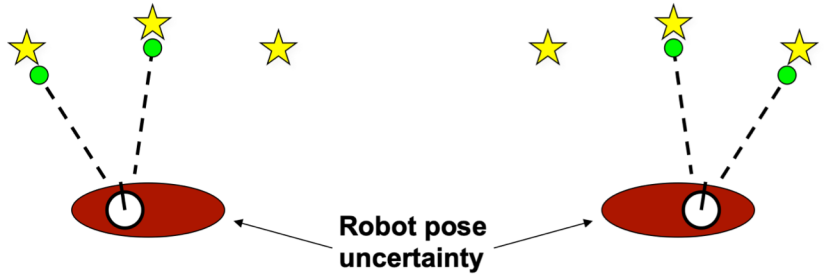


Simultaneous Localization and Mapping (SLAM)

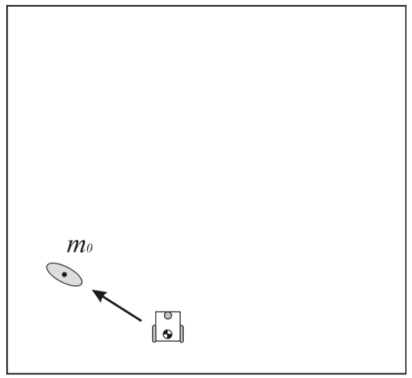
Much harder than localization or mapping alone:



1. Robot path and map are both unknown.
   - Increases the dimension of the problem
2. Map and pose estimates are correlated.
3. Mapping between observations and landmarks is unknown.
   - Wrong data association
   - More uncertainty -> more ambiguity in data assoc.



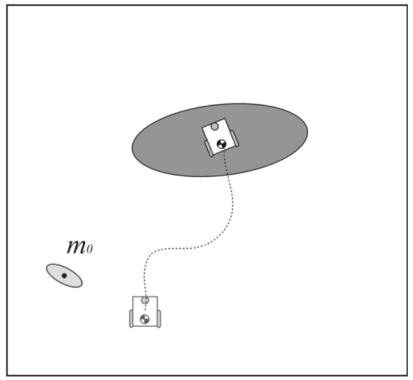
SLAM overview



Initially no uncertainty

New feature

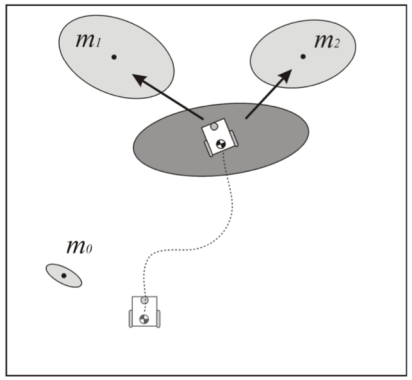
Sensor error model



Robot moves

Pose uncertainty grows

Stochastic dyn. model

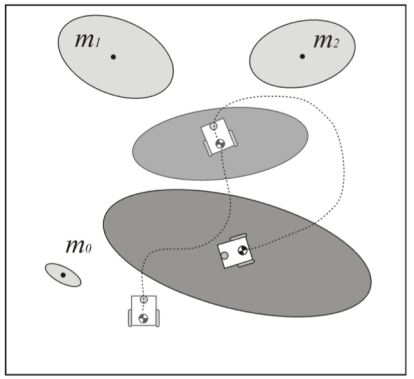


Two new features

Position estimate

Feature location estimate

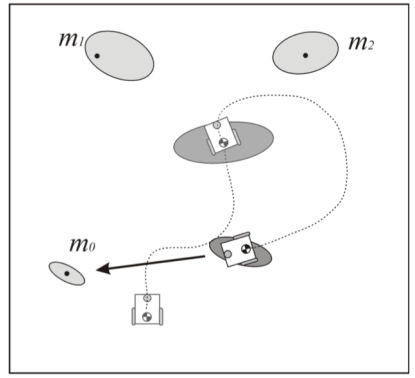
Correlated



Robot moves

Pose uncertainty grows

Stochastic dyn. model



Loop closure

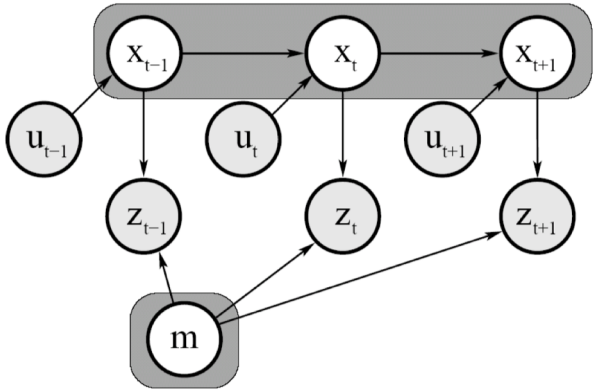
Already observed features

Uncertainty shrinks

SLAM

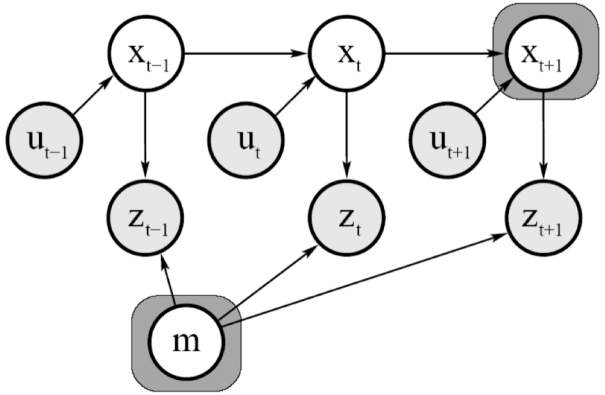
Offline (Full) SLAM: sensor data is collected, then estimate entire path&map

$$p(x_{0:T}, m | u_{0:T-1}, z_{1:T})$$



Online SLAM: the map and the latest pose are estimated during the operation of the robot

$$p(x_t, m | u_{0:t-1}, z_{1:t}) = \int \int \dots \int p(x_{1:t}, m | u_{0:t-1}, z_{1:t}) dx_1 dx_2 dx_{t-1}$$ *



Ignoring the past poses -> marginalizing out the previous poses

*Integrations are done one at a time.

Three main SLAM paradigms

Extended Kalman filter SLAM

- The earliest method
- Computationally demanding

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} \quad \Sigma_k = \begin{bmatrix} \sigma_x^2 & \sigma_{xy}^2 & \sigma_{x\theta}^2 \\ \sigma_{yx}^2 & \sigma_y^2 & \sigma_{y\theta}^2 \\ \sigma_{\theta x}^2 & \sigma_{\theta y}^2 & \sigma_\theta^2 \end{bmatrix}$$

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_R \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \vdots \\ \mathbf{m}_n \end{bmatrix} \quad \Sigma_k = \begin{bmatrix} \Sigma_R & \Sigma_{RM_1} & \Sigma_{RM_2} & \cdots & \Sigma_{RM_n} \\ \Sigma_{M_1R} & \Sigma_{M_1} & \Sigma_{M_1M_2} & \cdots & \Sigma_{M_1M_n} \\ \Sigma_{M_2R} & \Sigma_{M_2M_1} & \Sigma_{M_2} & \cdots & \Sigma_{M_2M_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \Sigma_{M_nR} & \Sigma_{M_nM_1} & \Sigma_{M_nM_2} & \cdots & \Sigma_{M_n} \end{bmatrix}$$

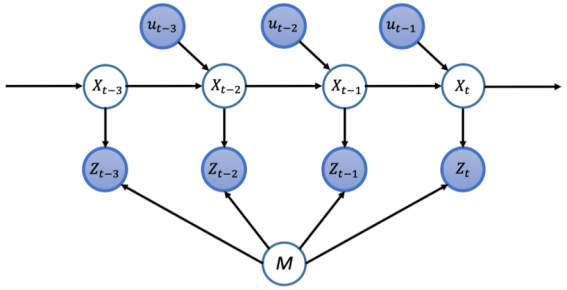
Three main SLAM paradigms

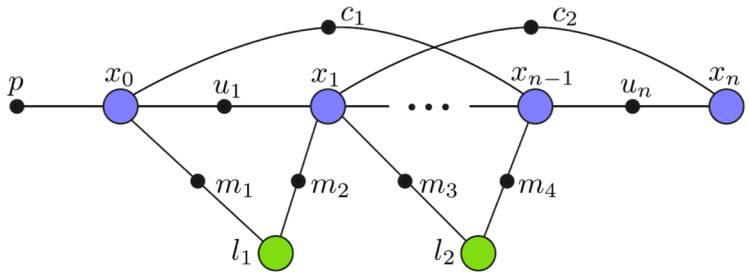
Extended Kalman filter SLAM

- The earliest method
- Computationally demanding

Graph-based SLAM

- Optimization over sparse graphs
- More scalable but mostly offline SLAM
- Nodes: Poses, landmarks; Edges: Spatial constraints
- Build the graph and find a node configuration minimizing error due to the constraints.





Three main SLAM paradigms

Extended Kalman filter SLAM

- The earliest method
- Computationally demanding

Graph-based SLAM

- Optimization over sparse graphs
- More scalable but mostly offline SLAM
- Nodes: Poses, landmarks; Edges: Spatial constraints
- Build the graph and find a node configuration minimizing error due to the constraints.

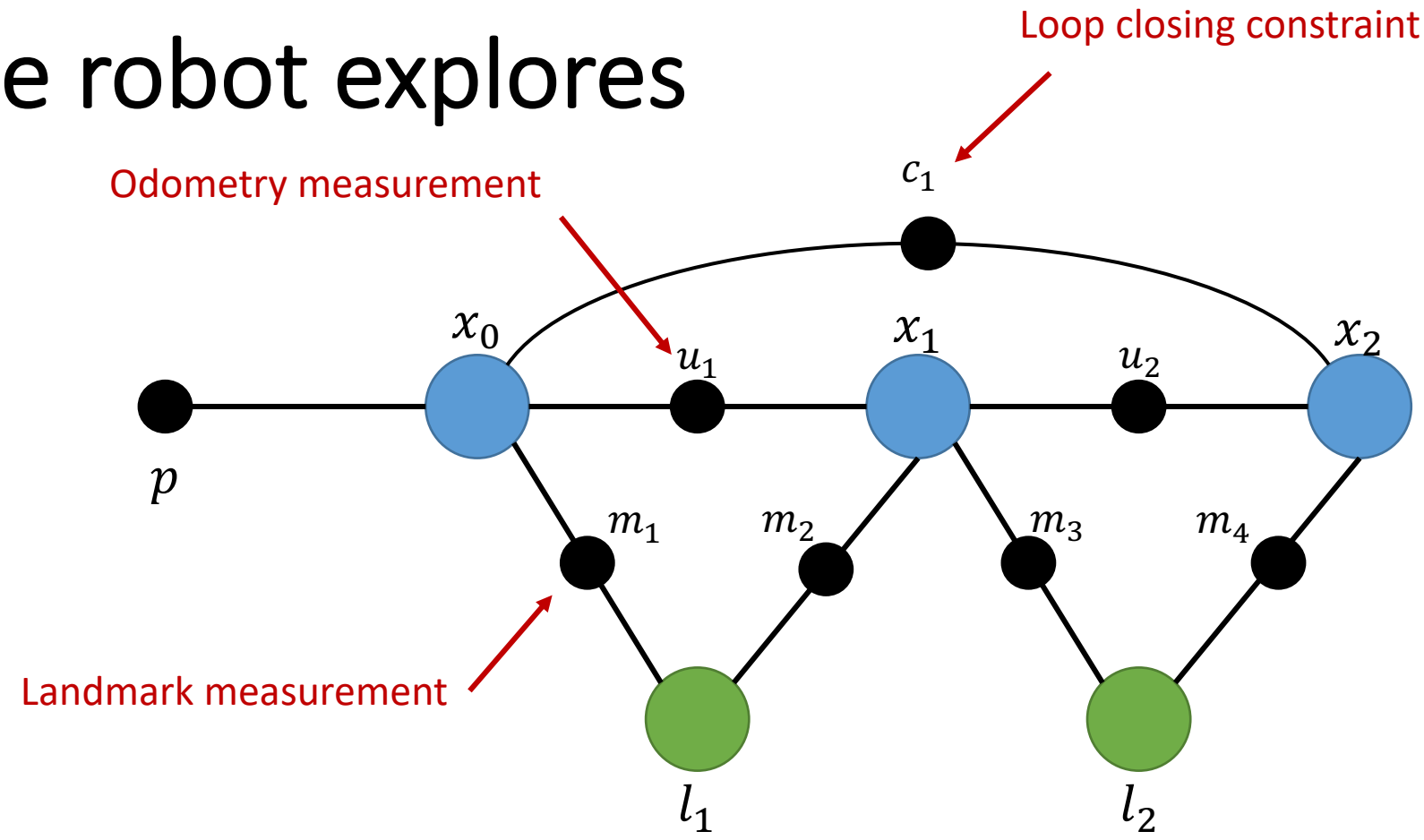
A modern approach

SLAM over factor graphs

Particle filter SLAM

- Online SLAM
- Less restrictive assumptions
- Number of particles – large problems

As the robot explores



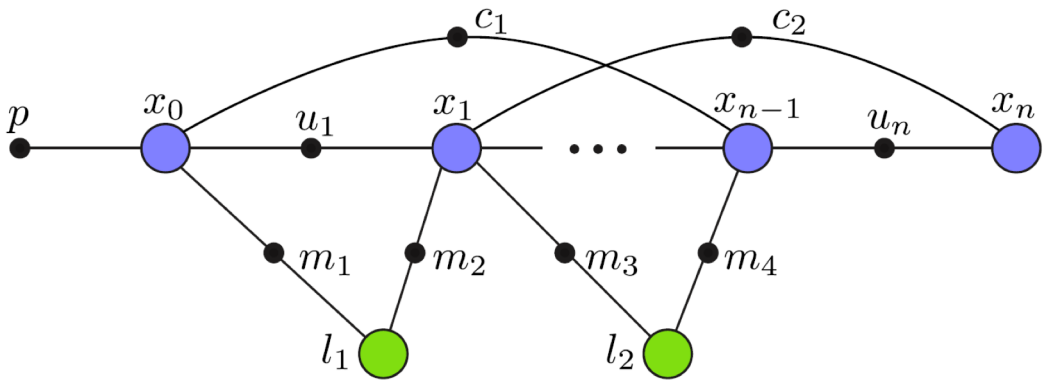
We build up a *graph* of *noisy spatial relations* …

Factor graphs

A *factor graph* $G = (\Theta, F, E)$ is a bipartite graph that models the factorization of a function $f: \Omega \to \mathbb{R}$.

$$f(\Theta) = \prod_i f_i(\Theta_i)$$

$$\Theta_i = \{\theta_j \in \Theta \mid (f_i, \theta_j) \in E\}$$



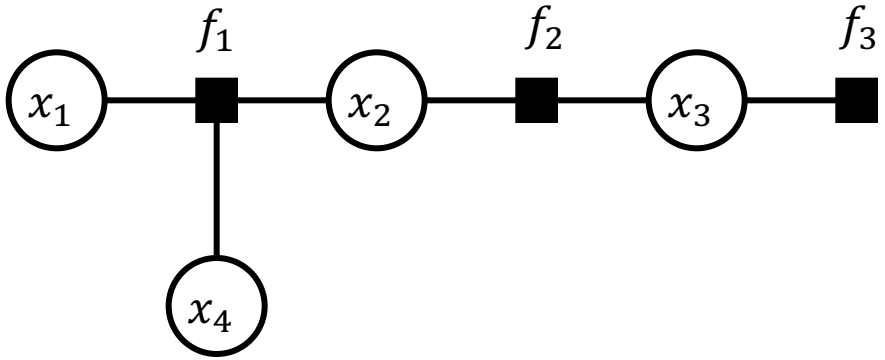
Here:

- Θ is the set of *variable nodes*
- F is the set of *factor nodes*
- E is the edge set. G has an edge $e_{ij} = (f_i, \theta_j)$ if and only if variable θ_j is an argument of factor f_i.

Factor graph - example

$$f(\Theta) = \prod_i f_i(\Theta_i)$$

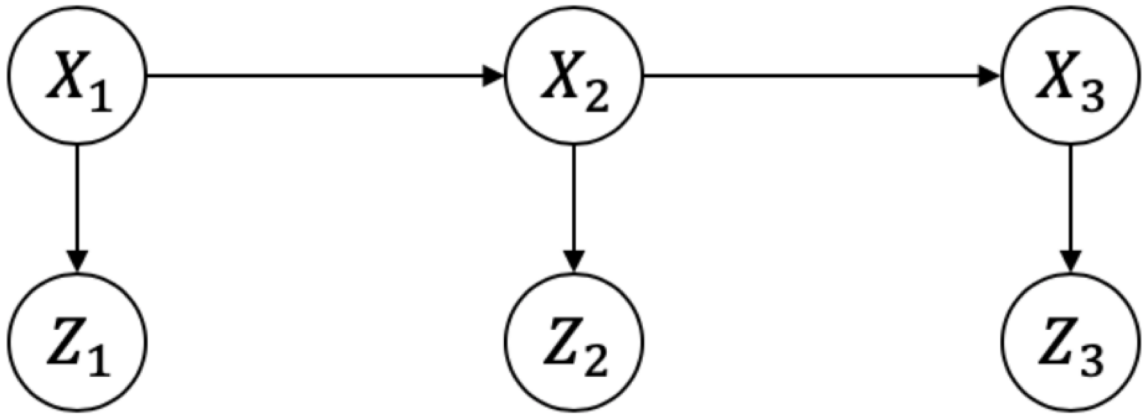
$$f(x_1, x_2, x_3, x_4) = f_1(x_1, x_2, x_3) f_2(x_2, x_3) f_3(x_3)$$



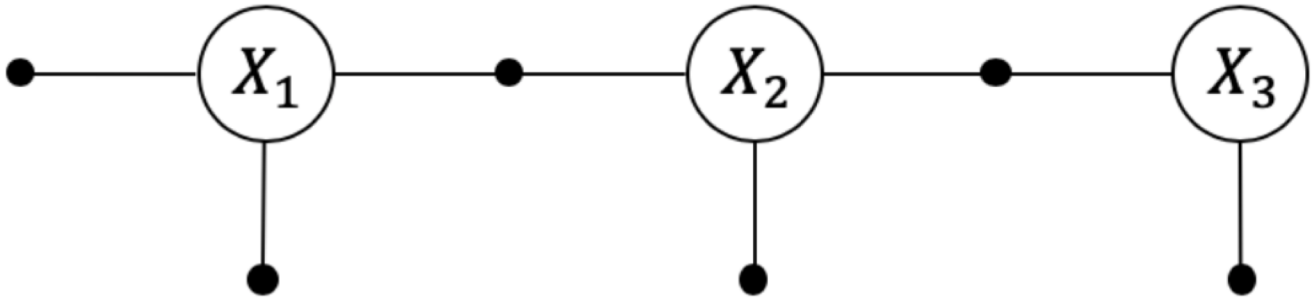
Probability distribution can be simplified!

Describe and visualize independence

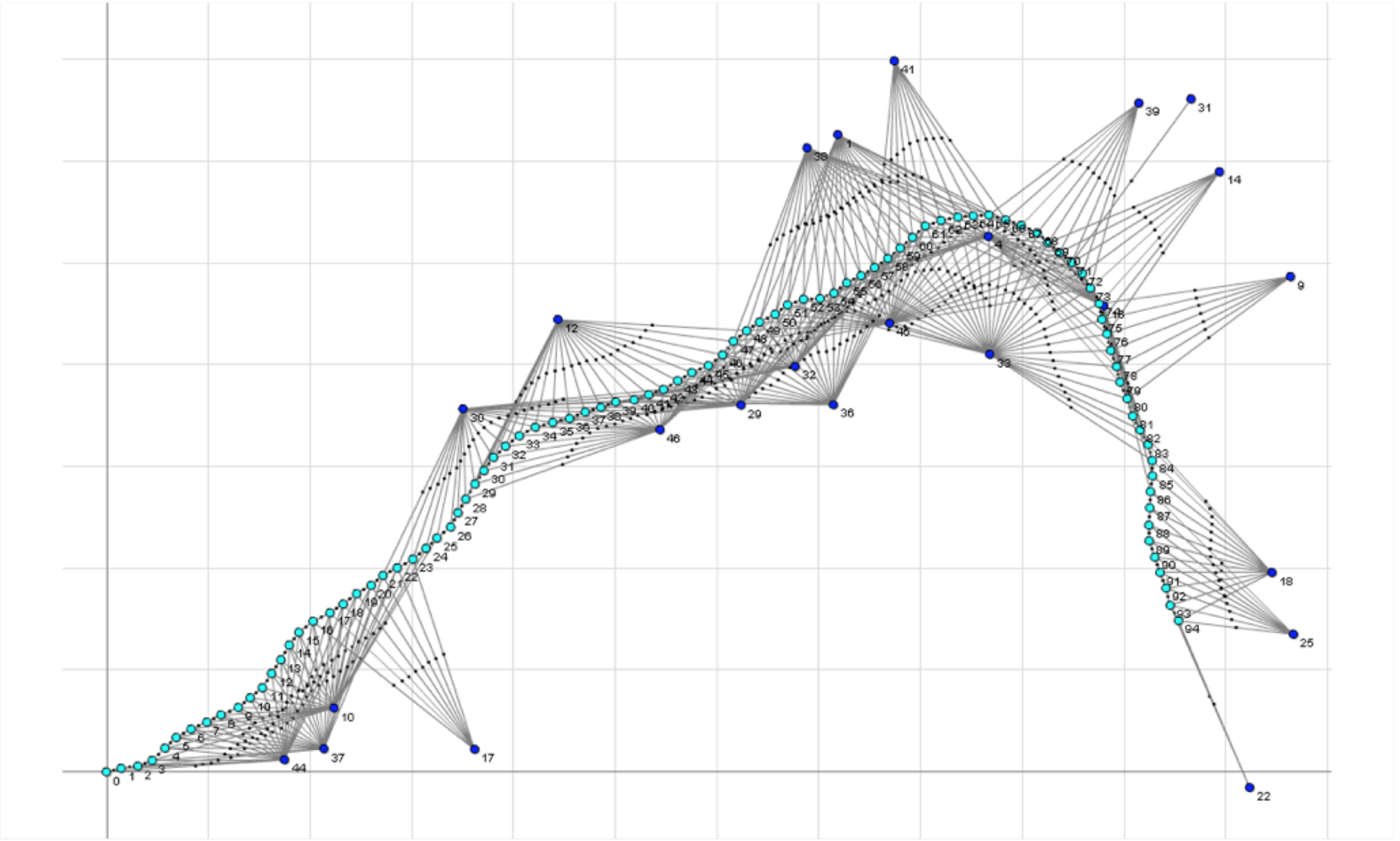
Converting Bayes Nets into Factor Graphs



Measurements are known, they become a fixed parameter in the corresponding factor.



SLAM and factor graphs



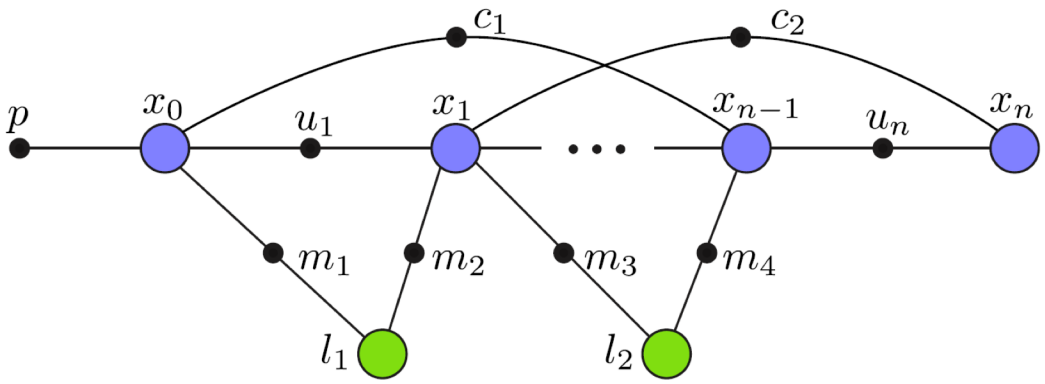
The SLAM estimation problem

Given: A factor graph representation $G = (\Theta, F, E)$ of the joint distribution for the network of noisy spatial relations:

$$\Theta_i = \{\theta_j \in \Theta \mid (p_i, \theta_j) \in E\}$$

$$p(Z|\Theta) = \prod_i p_i(Z_i|\,\Theta_i)$$

$$\propto \sum_i \log p_i(Z_i|\,\Theta_i)$$



Find: The *maximum likelihood estimate* $\widehat{\Theta}_{MLE}$ for the variables Θ:

$$\widehat{\Theta}_{MLE} = argmin_{\Theta} \sum -\log p_i(Z_i|\Theta_i)$$

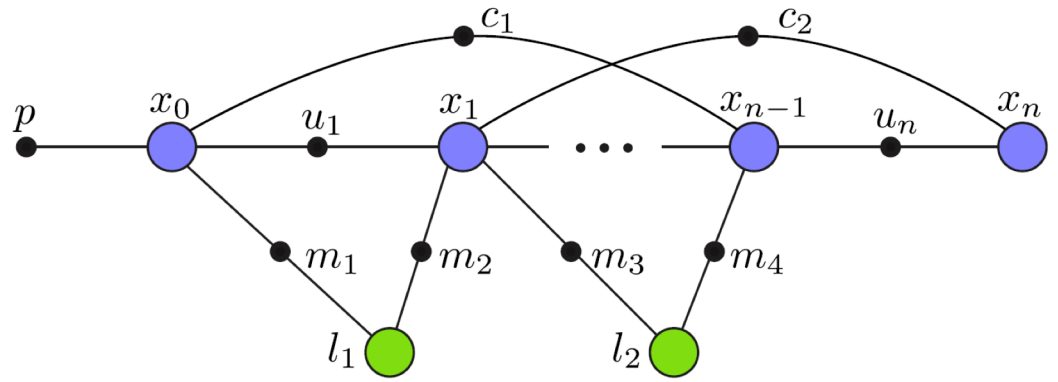
⇒This is the *point estimate* that *best explains* the available data.

Key features of the SLAM inference problem

The SLAM problem is:

- Nonlinear
- High-dimensional
- Nonconvex

⇒This is a challenging optimization problem

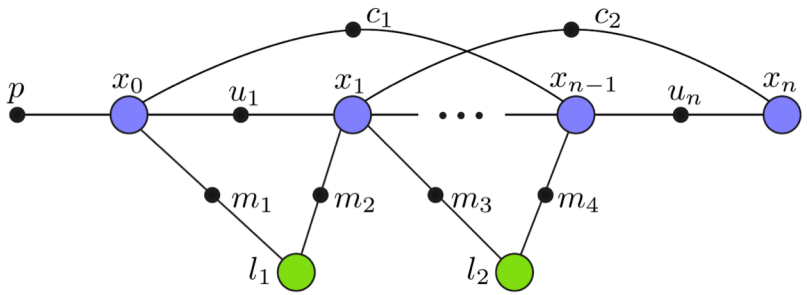


$$\widehat{\Theta}_{MLE} = argmin_{\Theta} \sum -\log p_i(Z_i|\Theta_i)$$

However: It is also *sparse*.
⇒This enables *efficient maximum likelihood estimation*

The "standard model" of SLAM

$$\widehat{\Theta}_{MLE} = argmin_{\Theta} \sum -\log p_i(Z_i|\Theta_i)$$



Let's **assume** that each factor p_i models a nonlinear measurement $z_i = h(\Theta_i) + \epsilon_i$, where $\epsilon_i \sim N(0, \Sigma_i)$ is additive Gaussian noise. Then:

$$p_i(Z_i|\Theta_i) \propto \exp\left(-\frac{1}{2}\|z_i - h_i(\Theta_i)\|^2_{\Sigma_i}\right)$$

The error between what we expect to measure and what we measure

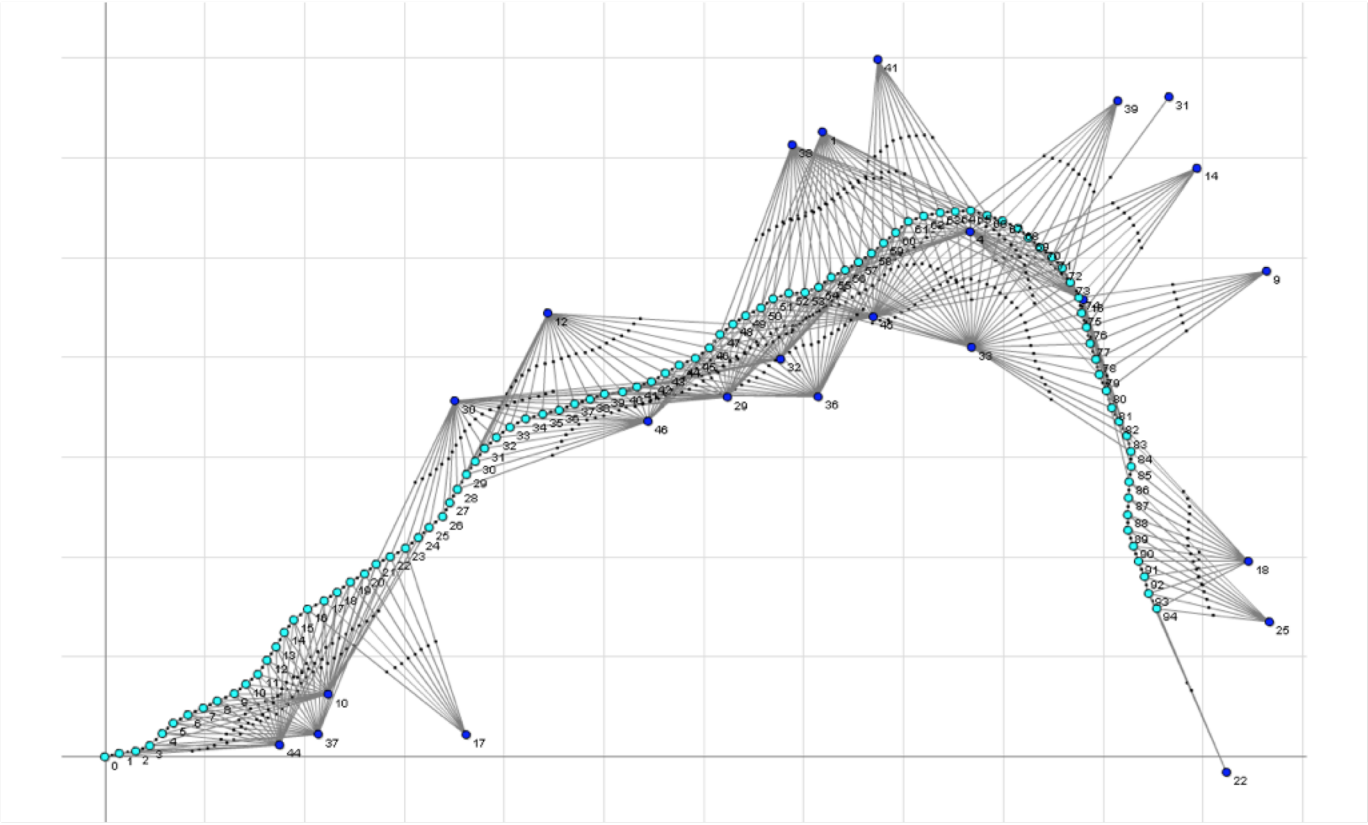
⇒Maximum likelihood inference is equivalent to a *sparse nonlinear least-squares* (NLS) problem:

$$\widehat{\Theta}_{MLE} = argmin_{\Theta} \sum_i \|z_i - h_i(\Theta_i)\|^2_{\Sigma_i}$$

Payoff: Sparse NLS problems can be processed *very* efficiently.

$$\|e\|^2_{\Sigma} \triangleq e^T \Sigma^{-1} e$$

SLAM and factor graphs



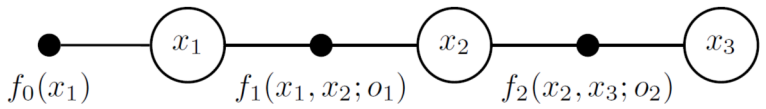
Current state-of-the-art SLAM approaches are all based upon sparse nonlinear least-squares estimation over factor graphs.

Software libraries:

- Ceres
- iSAM / GTSAM
- g2o

Example: GTSAM

A simple (toy) factor graph



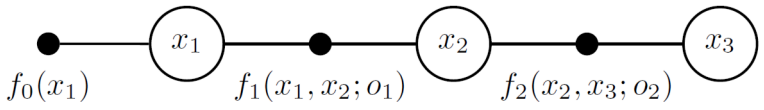
Constructing the factor graph in GTSAM

```
1  // Create an empty nonlinear factor graph
2  NonlinearFactorGraph graph;
3
4  // Add a Gaussian prior on pose x_1
5  Pose2 priorMean(0.0, 0.0, 0.0);
6  noiseModel::Diagonal::shared_ptr priorNoise =
7    noiseModel::Diagonal::Sigmas(Vector_(3, 0.3, 0.3, 0.1));
8  graph.add(PriorFactor<Pose2>(1, priorMean, priorNoise));
9
10 // Add two odometry factors
11 Pose2 odometry(2.0, 0.0, 0.0);
12 noiseModel::Diagonal::shared_ptr odometryNoise =
13   noiseModel::Diagonal::Sigmas(Vector_(3, 0.2, 0.2, 0.1));
14 graph.add(BetweenFactor<Pose2>(1, 2, odometry, odometryNoise));
15 graph.add(BetweenFactor<Pose2>(2, 3, odometry, odometryNoise));
```

Listing 1: Excerpt from examples/OdometryExample.cpp

https://gtsam.org/get_started/

Example: GTSAM



A simple (toy) factor graph

Optimizing the factor graph in GTSAM

```
1  // create (deliberatly inaccurate) initial estimate
2  Values initial;
3  initial.insert(1, Pose2(0.5, 0.0, 0.2));
4  initial.insert(2, Pose2(2.3, 0.1, -0.2));
5  initial.insert(3, Pose2(4.1, 0.1, 0.1));
6
7  // optimize using Levenberg-Marquardt optimization
8  Values result = LevenbergMarquardtOptimizer(graph, initial).optimize();
```

Listing 2: Excerpt from examples/OdometryExample.cpp

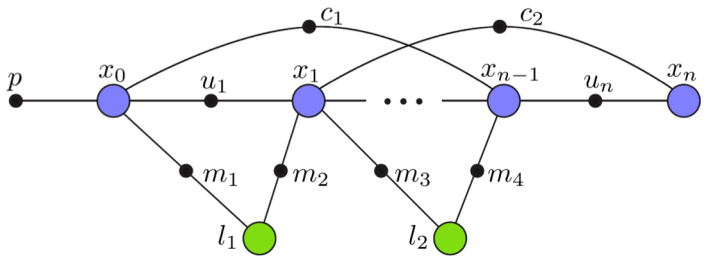
```
Initial Estimate:
Values with 3 values:
Value 1: (0.5, 0, 0.2)
Value 2: (2.3, 0.1, -0.2)
Value 3: (4.1, 0.1, 0.1)
```

```
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 SLAM estimation problem

Main takeaways:

- Factor graphs provide a *simple*, *flexible*, and *elegant* language for modeling machine perception problems



- Under the assumption of additive Gaussian noise:

$$z_i = h(\Theta_i) + \epsilon_i, \text{ where } \epsilon_i \sim N(0, \Sigma_i)$$

maximum likelihood estimation reduces to nonlinear least-squares:

$$\widehat{\Theta}_{MLE} = argmin_{\Theta} \sum_i \|z_i - h_i(\Theta_i)\|^2_{\Sigma_i}$$

$$p(Z|\Theta) = \prod_i p_i(Z_i|\,\Theta_i)$$

$$\Theta_i = \{\theta_j \in \Theta \mid (p_i, \theta_j) \in E\}$$

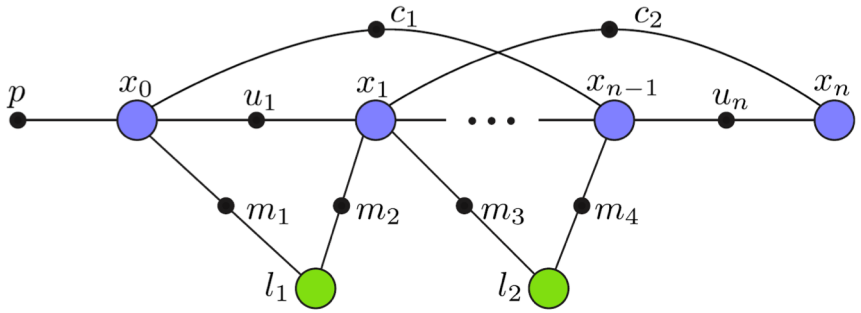
- We can process large-scale but *sparse* NLS problems *very* efficiently

SLAM practicalities

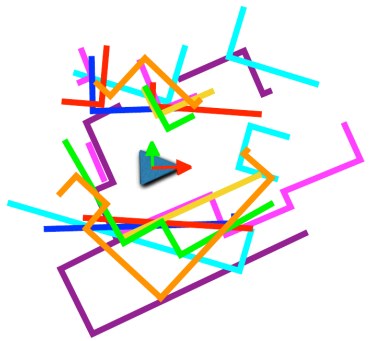
Recap: Factor graph models + nonlinear least-squares estimation provide a general and effective means of solving large-scale geometric estimation problems.

$$p(Z|\Theta) = \prod_i p_i(Z_i|\,\Theta_i)$$

$$\Theta_i = \{\theta_j \in \Theta \mid (p_i, \theta_j) \in E\}$$

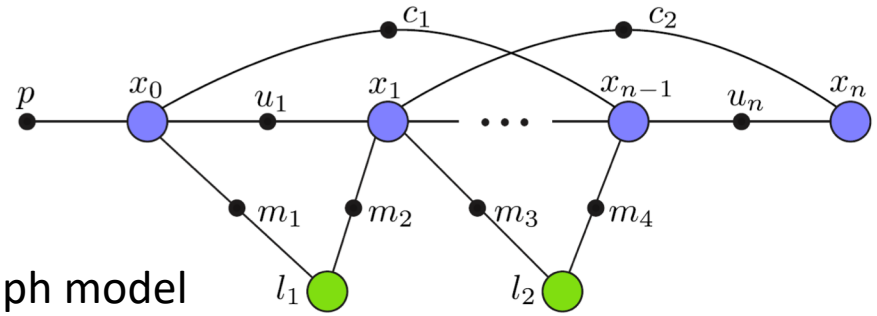


BUT: How are we supposed to obtain these factor graph models??



Raw sensor data

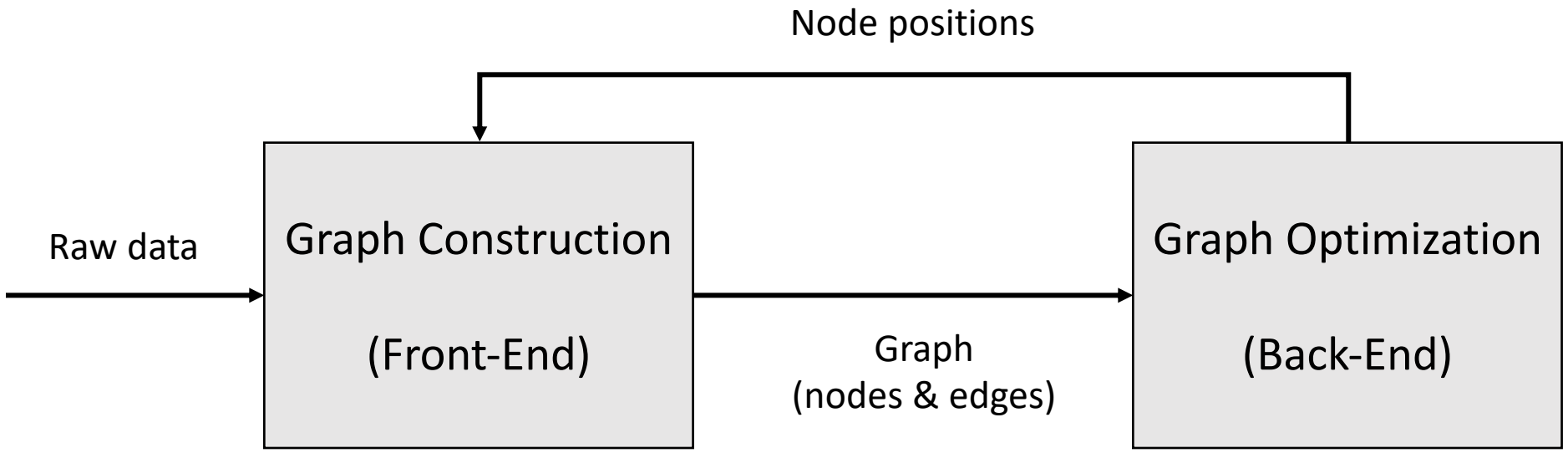
???



Factor graph model

Anatomy of a modern SLAM system

Interplay of front-end and back-end



Anatomy of a modern SLAM system

Front end: Build factor graph model of the SLAM estimation problem from sensor data

- Feature extraction
  - Process the raw sensor data to extract specific features / measurements / entities that will appear in the back-end factor graph model
- Data association
  - Feature extraction can identify *some* object of interest in raw sensor data. However, in order to construct a factor graph, we must also know *which one* it is.
  - the problem of associating *observations* (in our sensor data) with the *entities* (in the world) that produced them.

Back end: Perform optimization over the factor graph to recover maximum likelihood estimate $\widehat{\Theta}_{MLE}$ for SLAM solution

- Loop closure
  - Recognizing an already observed feature after a long exploration path
  - Uncertainties in robot and landmark estimates can be reduced

The majority of a SLAM system's complexity is devoted to constructing the factor graph model!

Feature extraction

Main idea: Process the raw sensor data to extract specific features / measurements / entities that will appear in the back-end factor graph model

Examples:

- **Feature points** and their descriptors
- **Keyframes** (camera poses + image data)
- **Objects**

ORB feature points

NB: ORB-SLAM2 uses projective (camera) observations of 3D points, extracted from images using the ORB feature detector

Each detected image feature $z_i \in \mathbb{R}^2$ gives rise to a *factor* of the form $p(z_i|t_i, x_i)$, where:

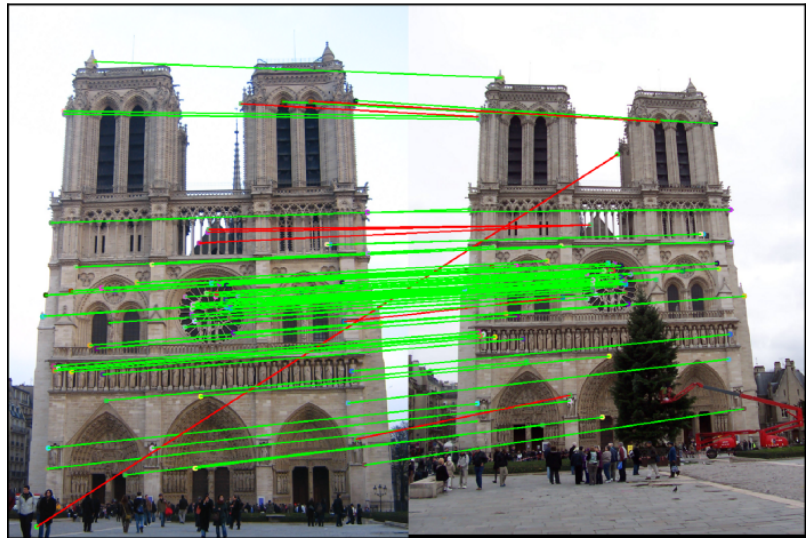
- $t_i \in \mathbb{R}^3$ is the position of the 3D point that produced the feature z_i
- $x_i \in SE(3)$ is the pose of the camera from which the image was taken

Data association

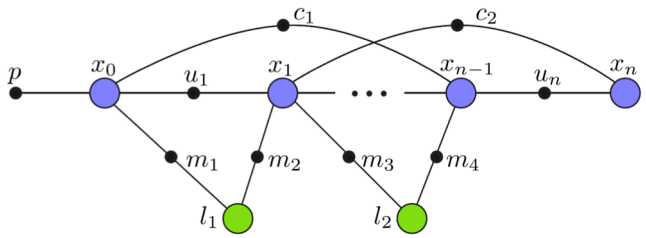
Main idea: Feature extraction can identify *some* object of interest in raw sensor data. However, in order to construct a factor graph, we must also know *which one* it is.

Data association is the problem of associating *observations* (in our sensor data) with the *entities* (in the world) that produced them.

Ex: In imagery, data association amounts to deciding which 2D features (in the image) correspond to the same 3D point (in the world)



NB: These (estimated!) associations determine the *edge set* in our factor graph



Data association: Easy and Hard Cases

Easy case: ***Feature tracking*** It's (relatively) easy to track features over a *short sequence* of measurements. Since the sensor is not moving far, we have a good idea of *where to look*

Data association: Easy and Hard Cases

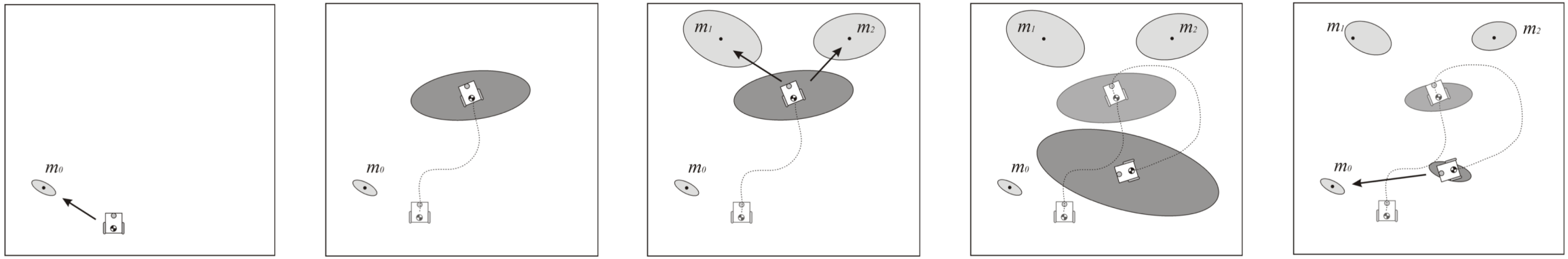
Hard case: ***Loop closures*** Conversely, if we revisit a location after traveling a long distance (i.e. if we *close a loop*), we may have high uncertainty in our position (due to *drift*)

- High pose uncertainty ⇒ no strong constraint on *which features to consider*
- Large changes in view can make identifying correspondences much harder

BUT: Loop closures are *essential for correcting drift!*

Loop closures

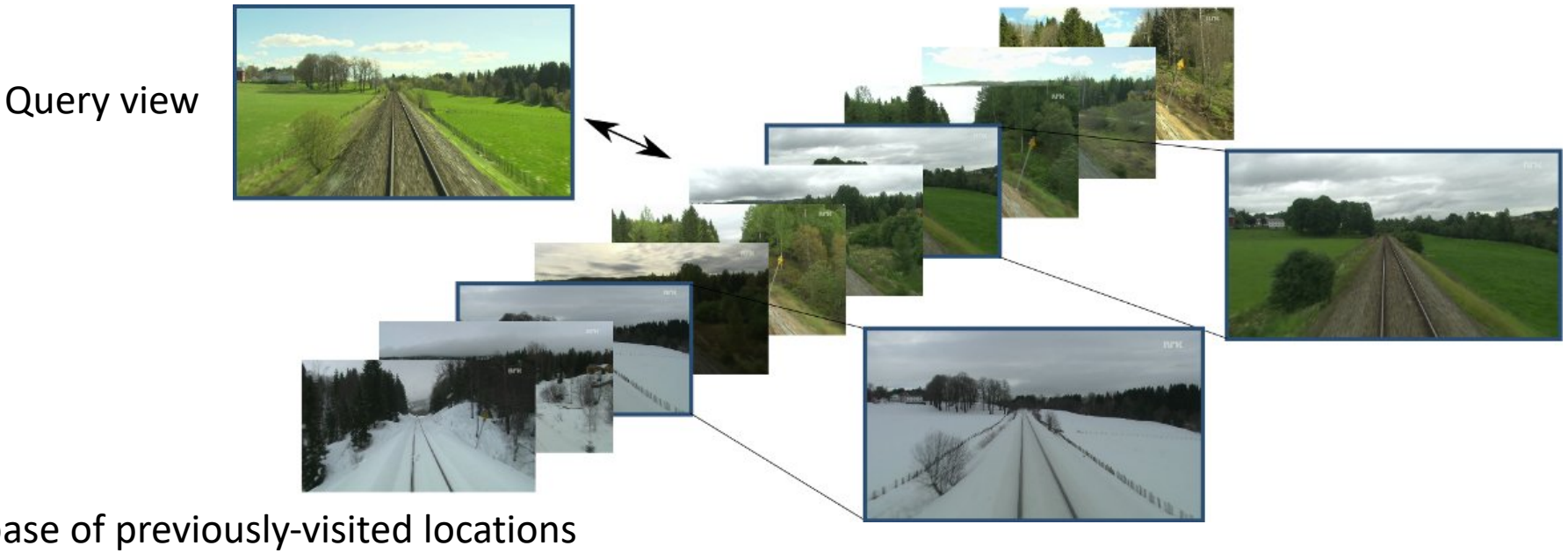
- Reducing uncertainty once recognizing an already observed feature



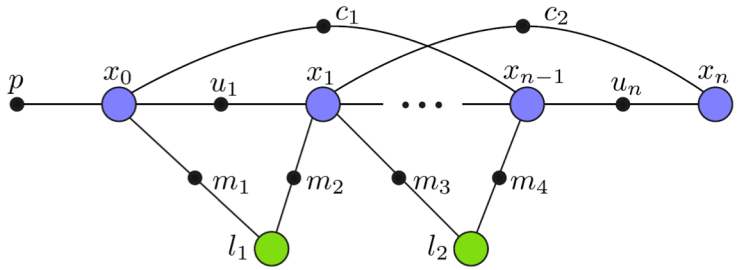
Finding loop closures: Place recognition

Problem: How can we recognize if we have visited a place before?

Main idea: Think of this as a search problem: try to find a *previous* view that matches the *current* view.



Summary



- Factor graphs provide a *simple*, *flexible*, and *elegant* language for modeling machine perception problems
- Under the assumption of additive Gaussian noise:

$$z_i = h(\Theta_i) + \epsilon_i, \text{ where } \epsilon_i \sim N(0, \Sigma_i)$$

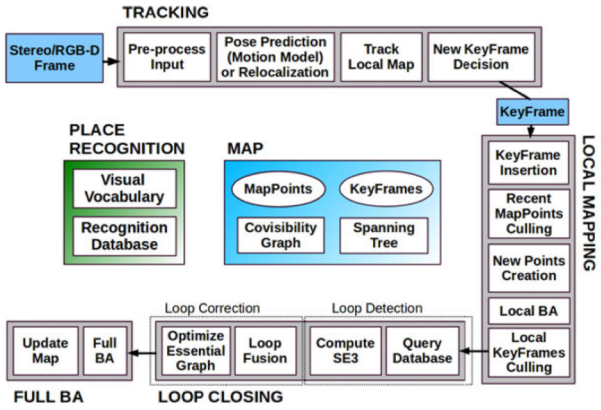
$$p(Z|\Theta) = \prod_i p_i(Z_i|\,\Theta_i)$$

Maximum likelihood estimation reduces to NLS:

$$\Theta_i = \{\theta_j \in \Theta \mid (p_i, \theta_j) \in E\}$$

$$\widehat{\Theta}_{MLE} = argmin_{\Theta} \sum_i \|z_i - h_i(\Theta_i)\|^2_{\Sigma_i}$$

- We can process *sparse* NLS problems *very* efficiently



BUT: A SLAM *system also* needs to address the *front-end*

- Feature extraction + Data association

⇒These account for most of the (practical) complexity!