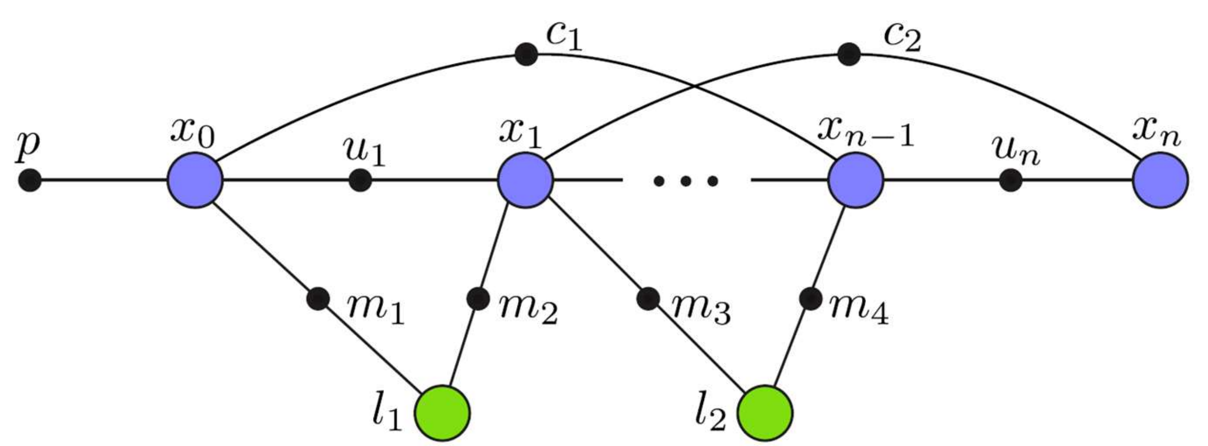
EECE 5550: Mobile Robotics



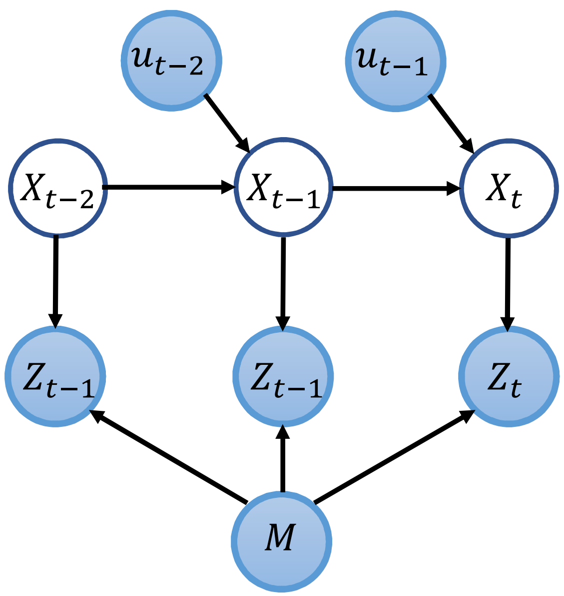
Lecture 13: Simultaneous Localization and Mapping

Recap: Two fundamental problems in robot perception

Localization: Where am I?

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

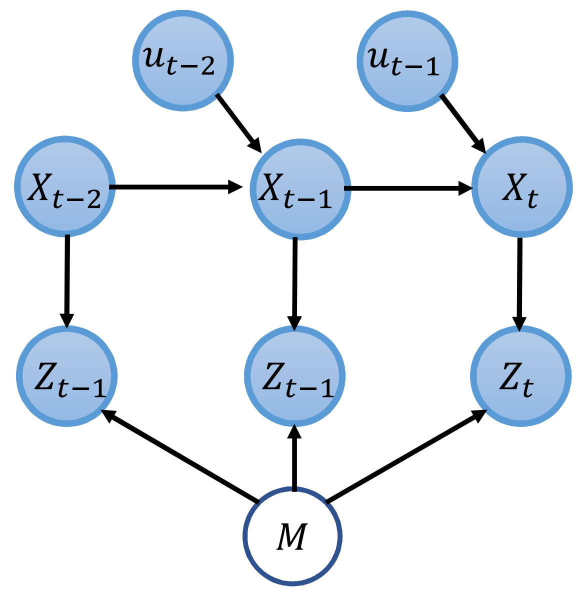
Estimate: Belief $p(x_t|\ m, u_{0:t-1}, z_{1:t})$ over robot pose



Mapping: What's around me?

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

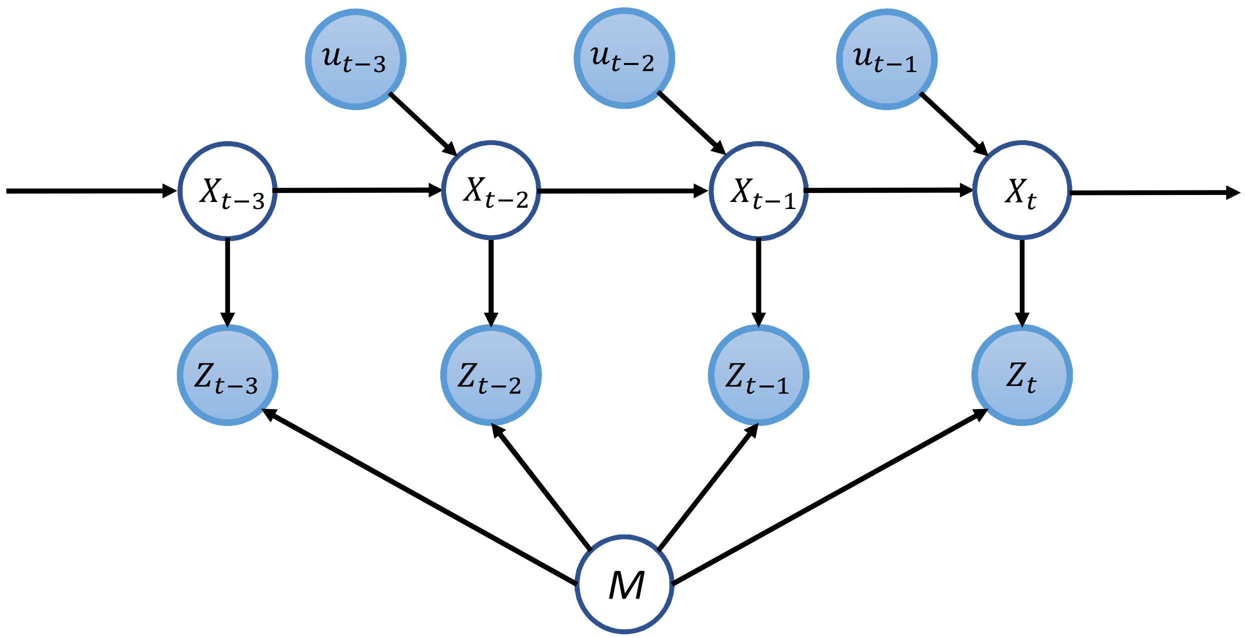
Estimate: Belief $p(m|x_{0:t}, z_{1:t})$ over the map *M*



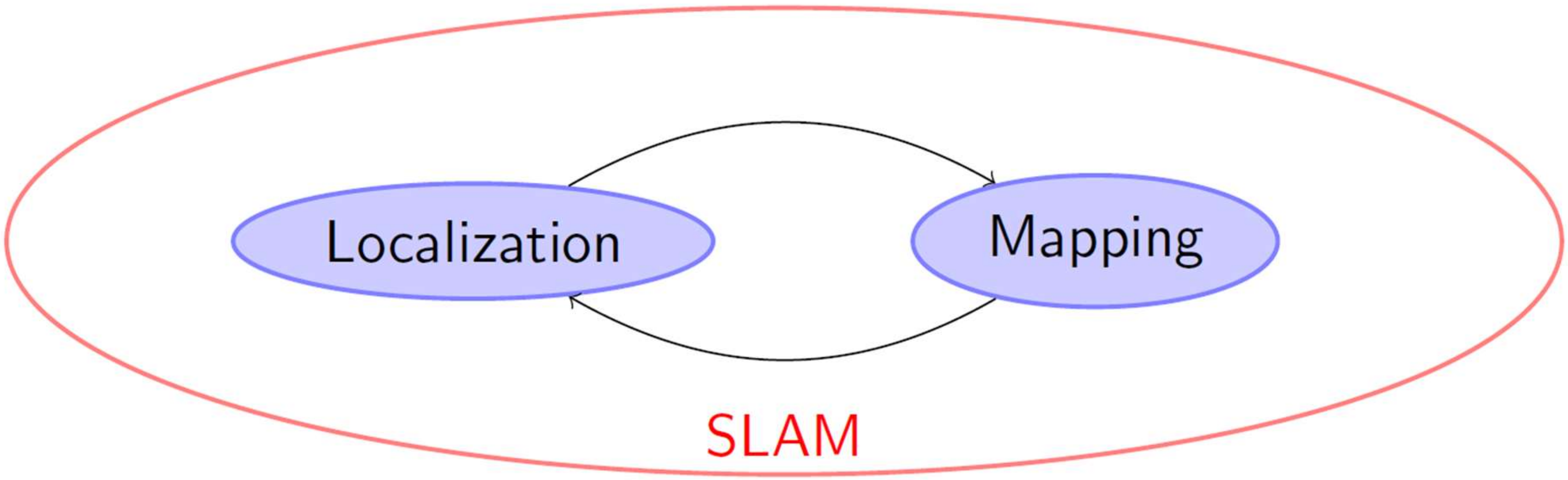
Simultaneous Localization and Mapping (SLAM): The Big One

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

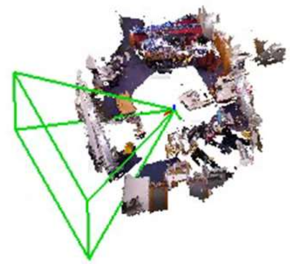


Simultaneous Localization and Mapping (SLAM)



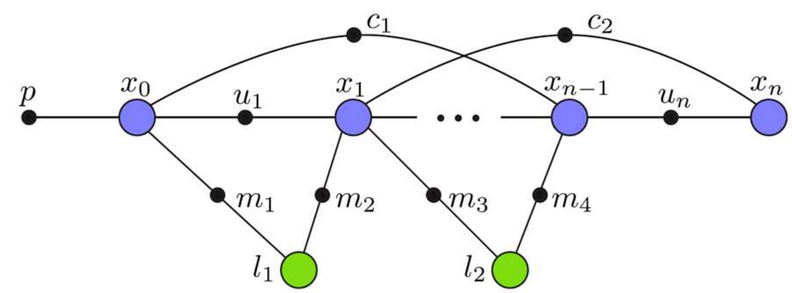
- **Much** harder than localization or mapping alone
- Enables operation in *unknown environments* (exploration)
- An *essential enabling technology* for mobile robots

Simultaneous Localization and Mapping (SLAM): The Big One



Plan of the day

- Motivating example
- Factor graphs
- SLAM problem formulation
- 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

Foundations and Trends® in Robotics
Vol. 6, No. 1-2 (2017) 1–139

DOI: 10.1561/2300000043

now
the essence of knowledge

Factor Graphs for Robot Perception

Frank Dellaert
Georgia Institute of Technology
dellaert@cc.gatech.edu

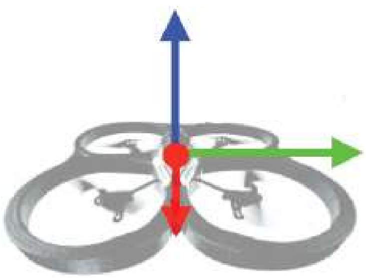
Michael Kaess
Carnegie Mellon University
kaess@cmu.edu

Papers

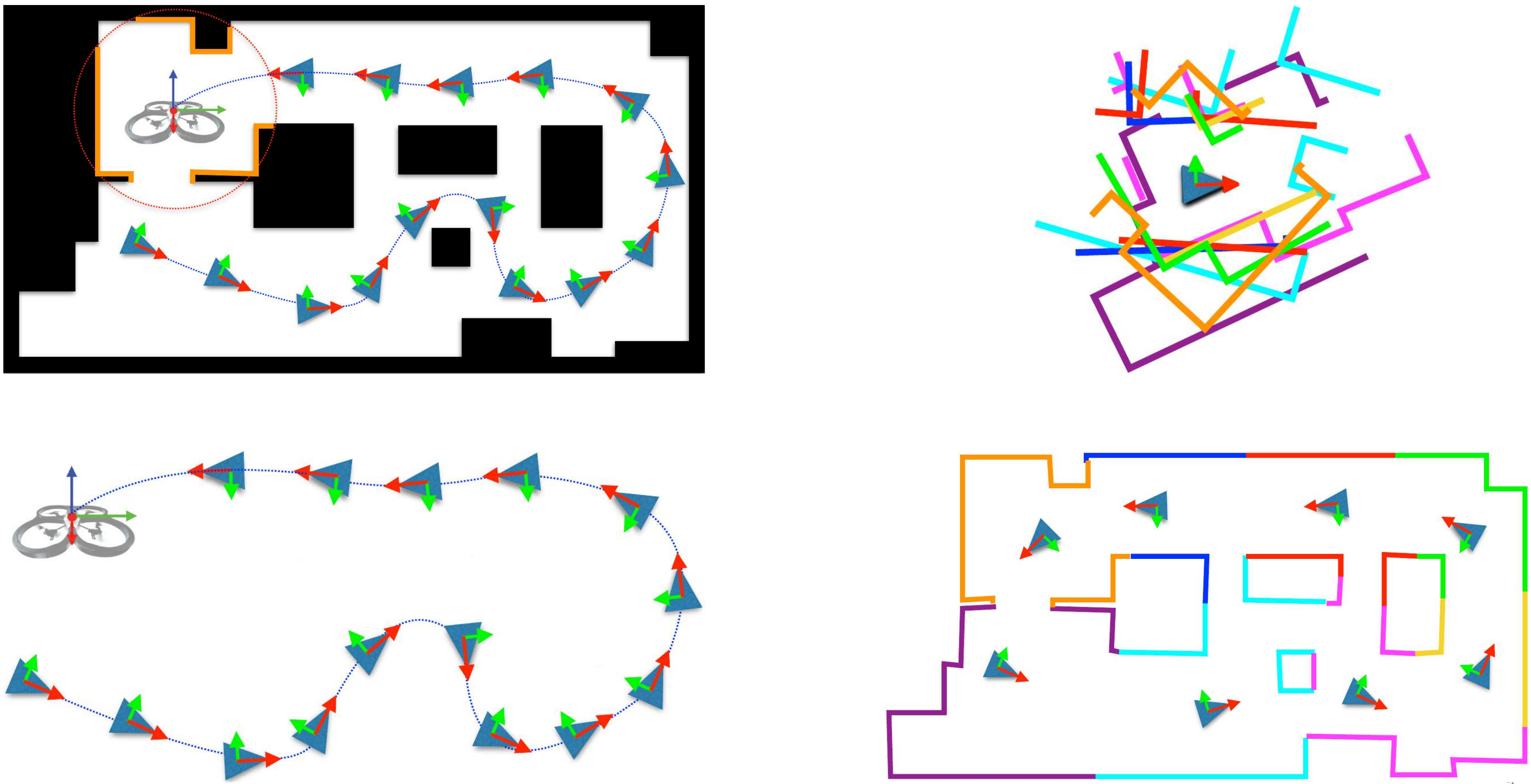
- "Factor Graphs for Robot Perception"
- "Factor Graphs and GTSAM: A Hands-On Introduction"
- "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras"
- "Bags of Binary Words for Fast Place Recognition in Image Sequences"

A concrete example

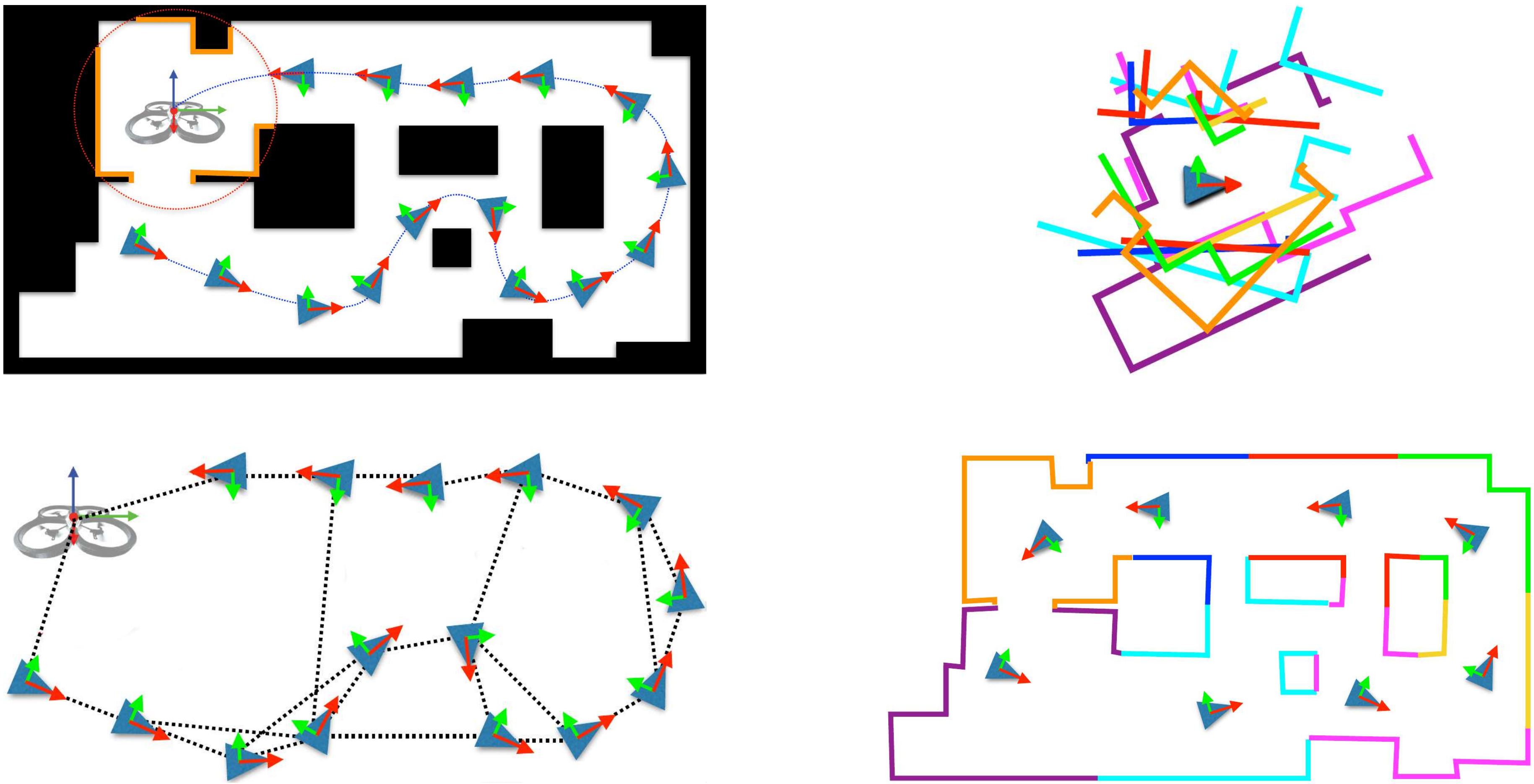
Consider a robot exploring some initially unknown environment …



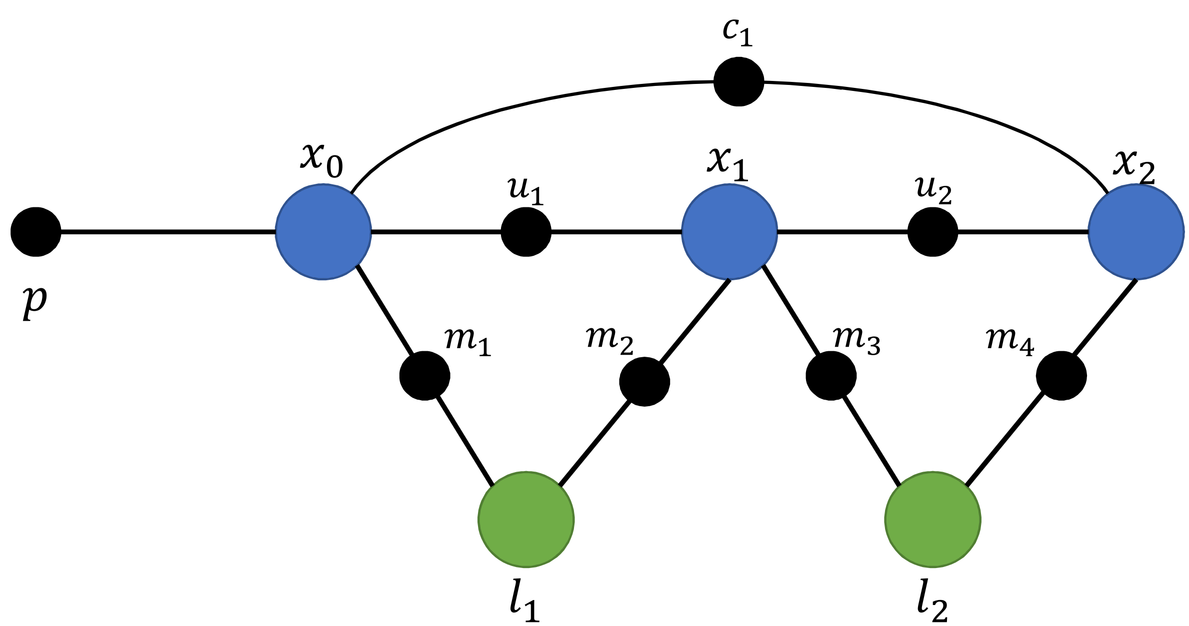
A concrete example



A concrete example



As the robot explores



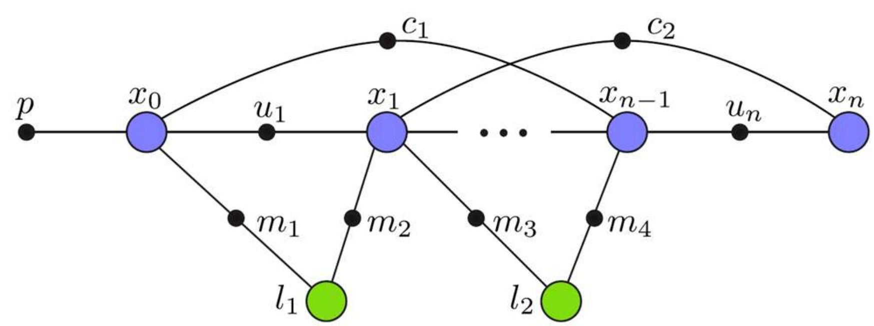
We build up a *graph* of *noisy spatial relations* …

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:

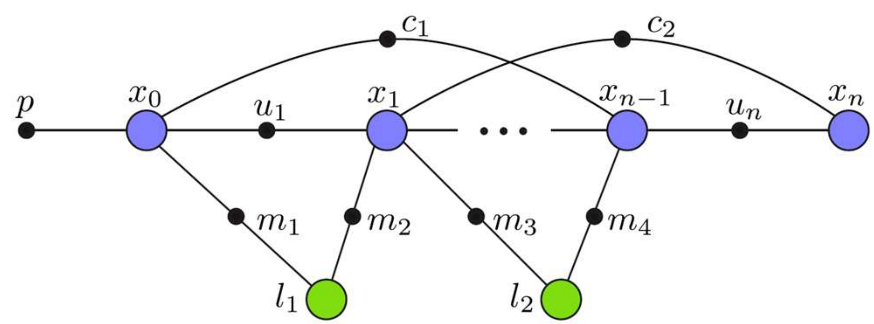
- Θ is the set of *variable nodes*
- *F* is the set of *factor nodes*
- *E* is the edge set. *G* has an edge $e_{ij} = (f_i, \theta_j)$ if and only if variable θ_j is an argument of factor f_i.

The SLAM estimation problem

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

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

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



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

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

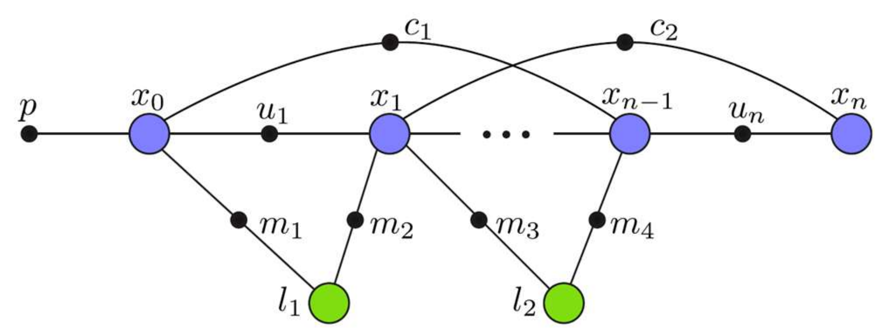
⇒This is the *point estimate* that *best explains* the available data.

Key features of the SLAM inference problem

The SLAM problem is:

- Nonlinear
- High-dimensional
- Nonconvex

⇒This is a challenging optimization problem



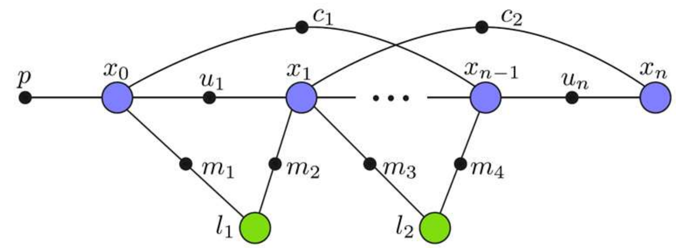
$$\widehat{\Theta}_{ML} = argmin_{\Theta} \sum -\log p_i(Z_i|\Theta_i)$$

However: It is also *sparse*.

⇒This enables *efficient maximum likelihood estimation*

The "standard model" of SLAM

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



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:

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

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

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

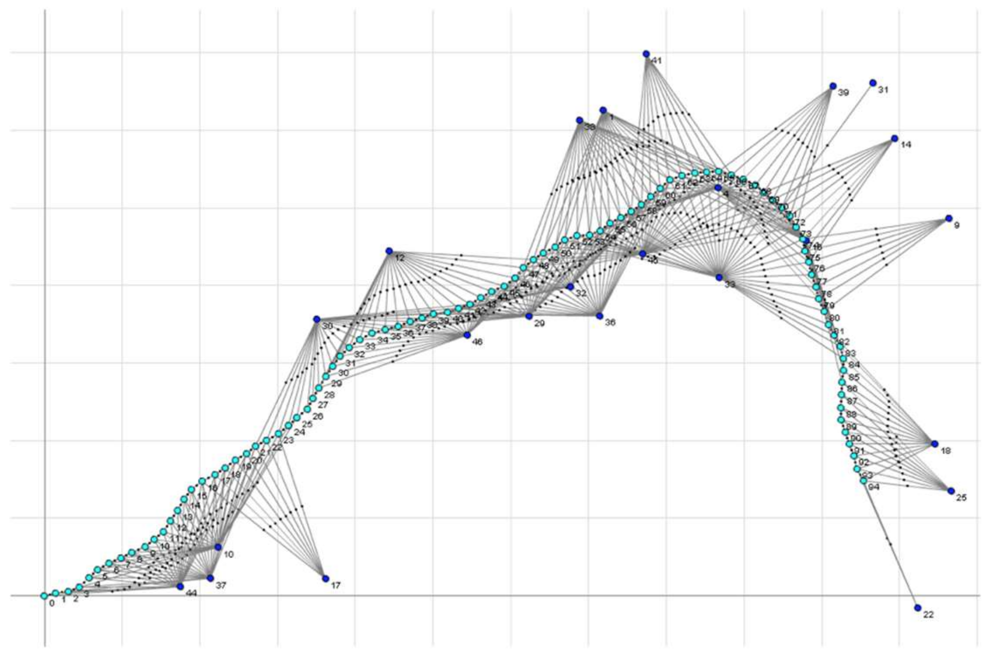
Payoff: Sparse NLS problems can be processed *very* efficiently. (More on this next time …)

Software for solving sparse NLS problems

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

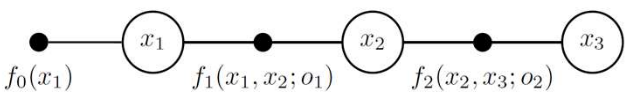
Software libraries:

- Ceres
- iSAM / GTSAM
- g2o



Example: GTSAM

A simple (toy) factor graph



Constructing the factor graph in GTSAM

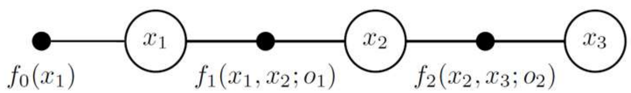
```
// Create an empty nonlinear factor graph
NonlinearFactorGraph graph;

// Add a Gaussian prior on pose x_1
Pose2 priorMean(0.0, 0.0, 0.0);
noiseModel::Diagonal::shared_ptr priorNoise =
  noiseModel::Diagonal::Sigmas(Vector_(3, 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(Vector_(3, 0.2, 0.2, 0.1));
graph.add(BetweenFactor<Pose2>(1, 2, odometry, odometryNoise));
graph.add(BetweenFactor<Pose2>(2, 3, odometry, odometryNoise));
```

Listing 1: Excerpt from examples/OdometryExample.cpp

Example: GTSAM



A simple (toy) factor graph

Optimizing the factor graph in GTSAM

```
// create (deliberatly 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();
```

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

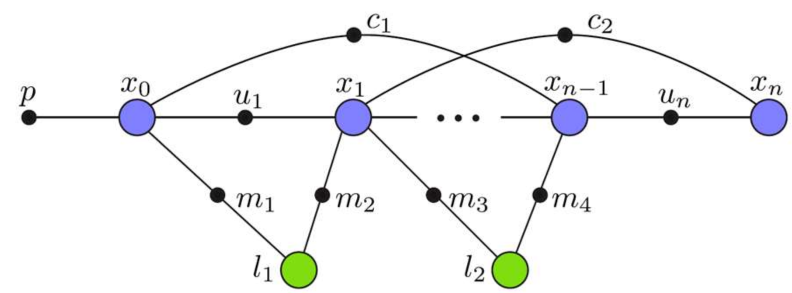
- Under the assumption of additive Gaussian noise:

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

maximum likelihood estimation reduces to nonlinear least-squares:

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

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



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

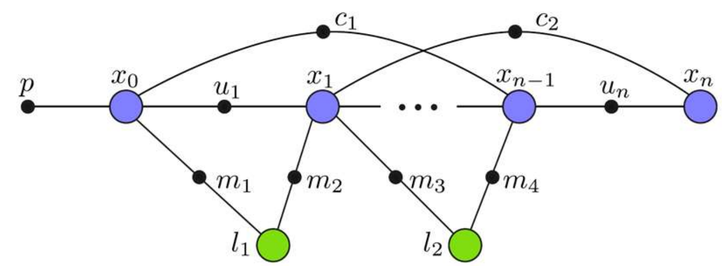
$$\Theta_i = \{\theta_j \in \Theta \mid (p_i, \theta_j) \in E\}$$

SLAM practicalities

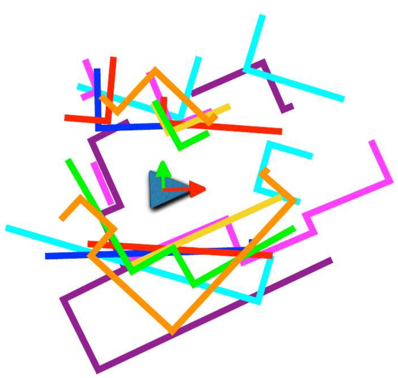
Recap: Factor graph models + nonlinear least-squares estimation provide a general and effective means of solving large-scale geometric estimation problems.

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

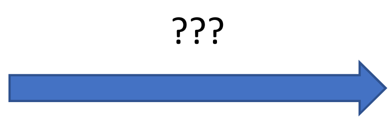
$$\Theta_i = \{\theta_j \in \Theta \mid (p_i, \theta_j) \in E\}$$

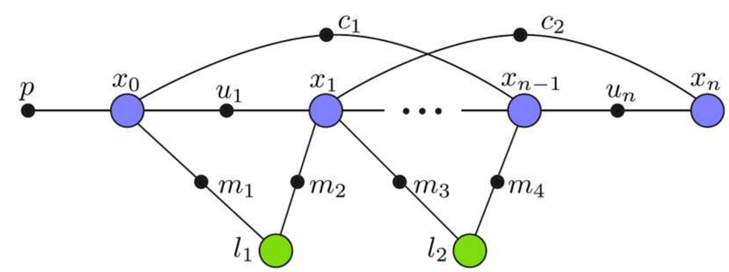


BUT: How are we supposed to obtain these factor graph models??



Raw sensor data





Factor graph model

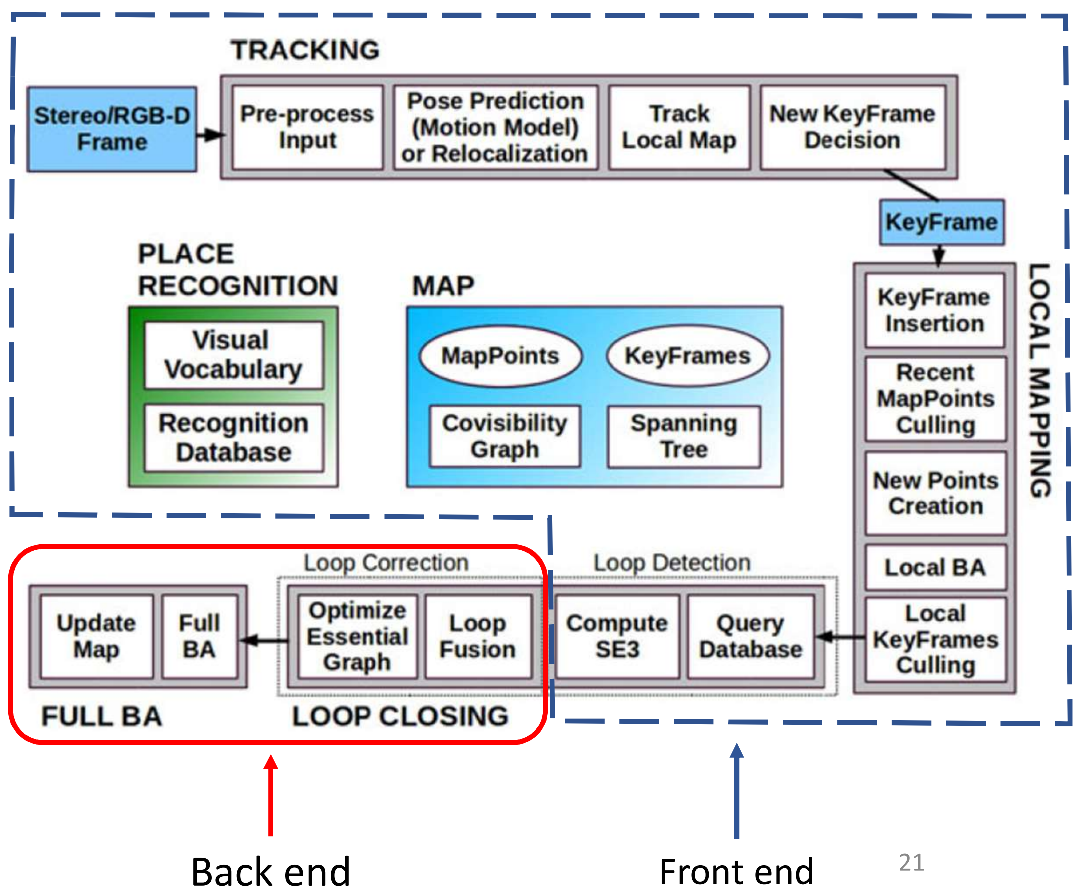
Anatomy of a modern SLAM system

Modern SLAM systems are typically composed of two components:

- **Front end:** Build factor graph model of the SLAM estimation problem from sensor data
  - Feature extraction
  - Data association
- **Back end:** Perform optimization over the factor graph to recover maximum likelihood estimate $\widehat{\Theta}_{MLE}$ for SLAM solution

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

ORB-SLAM2 system architecture



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

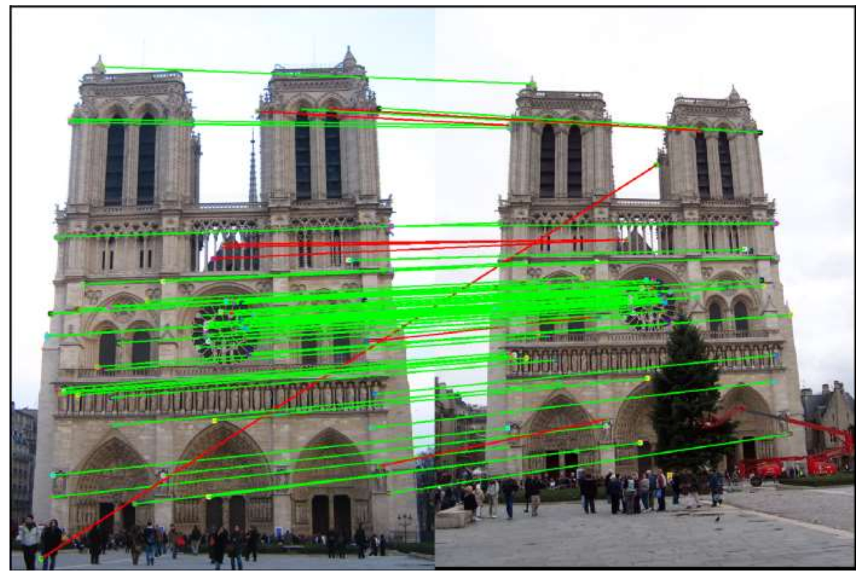
Each detected image feature $z_i \in \mathbb{R}^2$ gives rise to a *factor* of the form $p(z_i | t_i, x_i)$, where:

- $t_i \in \mathbb{R}^3$ 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

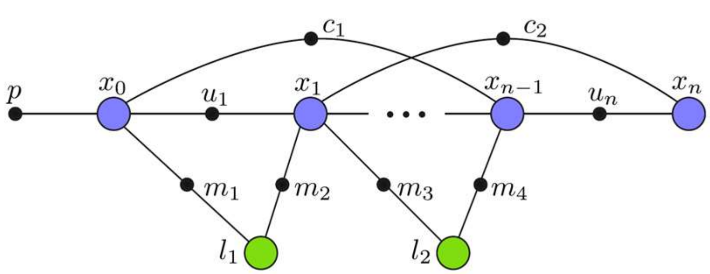
Main idea: Feature extraction can identify *some* object of interest in raw sensor data. However, in order to construct a factor graph, we must also know *which one* it is.

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!*

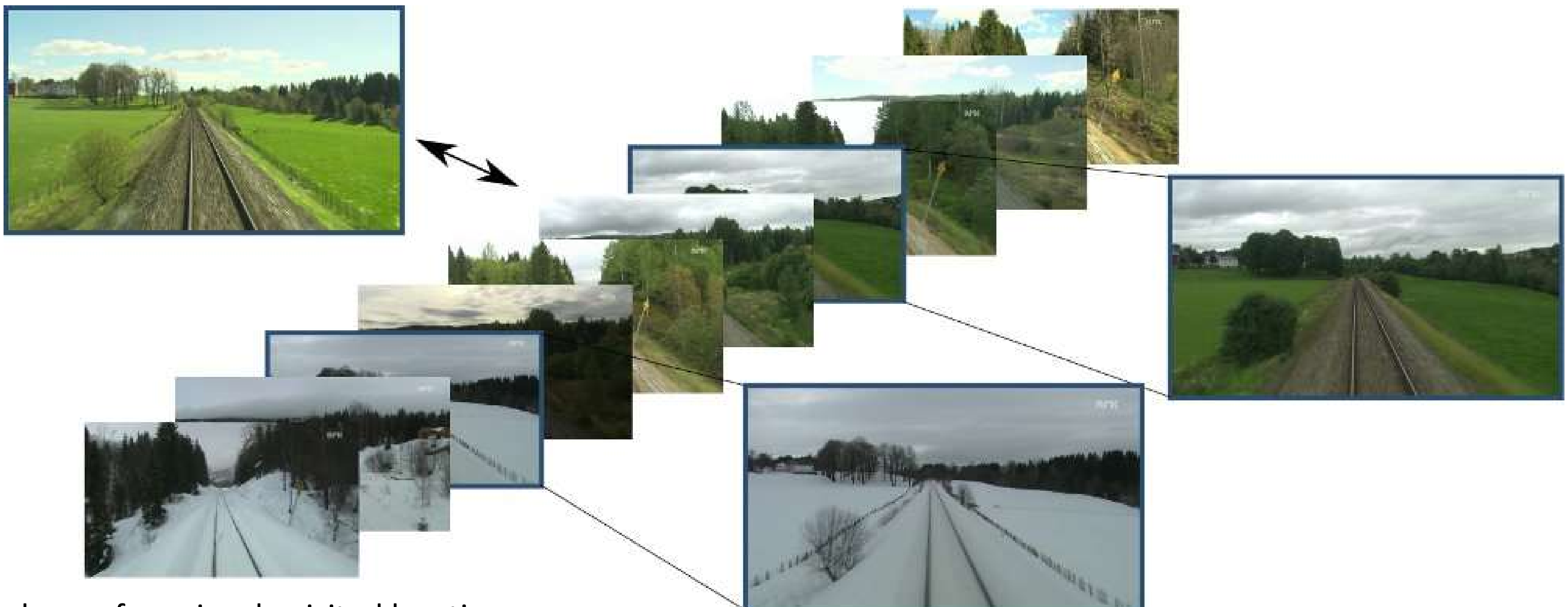
Loop closures correct drift

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.

Query view



Database of previously-visited locations

Bag-of-Words Place Recognition

Bag-of-words place recognition is one of the most common approaches to place recognition.

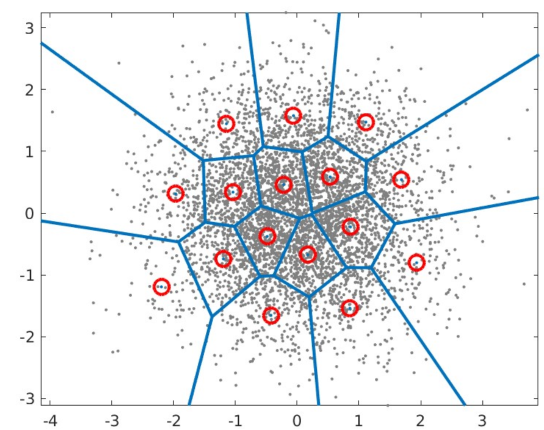
Main idea: Treat images as "bags" of "visual words" (set of [quantized] descriptors extracted from feature points in the image).

Then: we can treat *place recognition* as *text retrieval*!

- Image feature ⇔ "word" in a vocabulary
- Image (list of visual words) ⇔ Document
- Image similarity search ⇔ Similar document retrieval

Operationally, given a query image, we:

- **Extract visual features**, and map to quantized vocabulary
  ⇒This gives a representation of our image as a sparse feature vector x, in which x_k is the *count* of the kth word in the image
- **Retrieve similar images** by finding similar feature vectors in a database

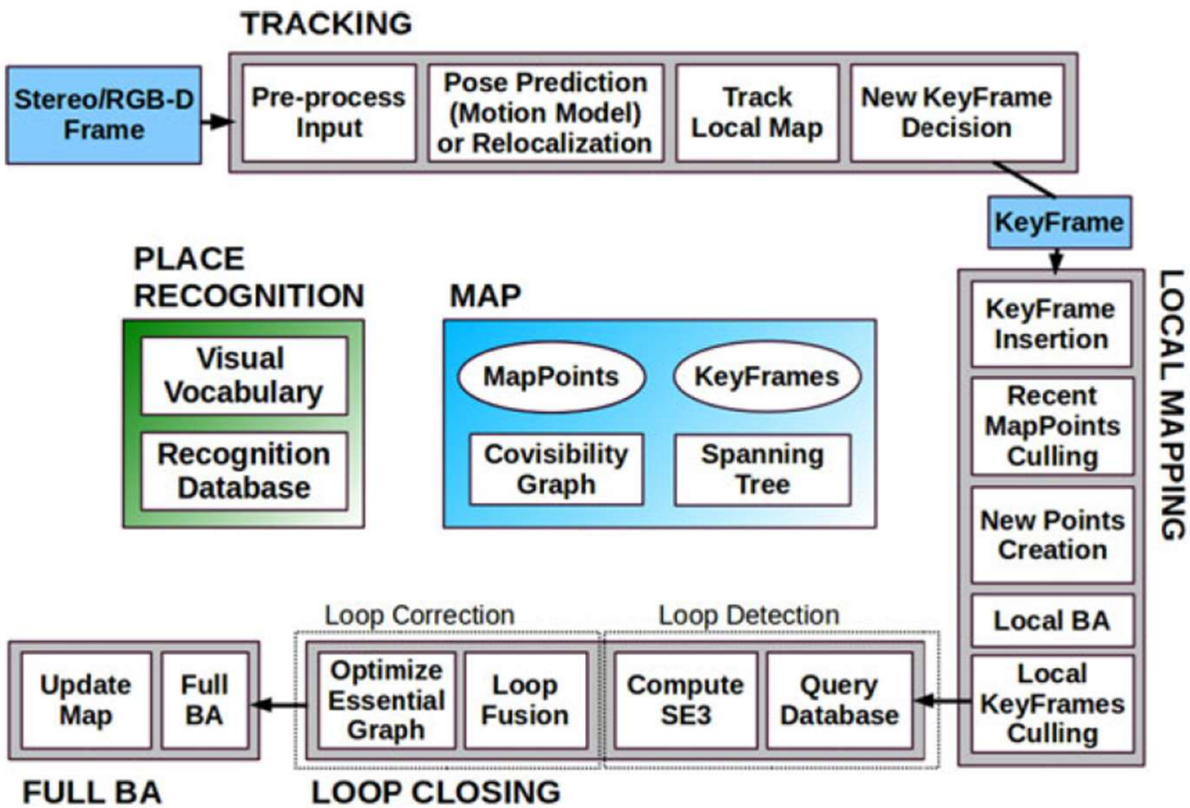


Vocabulary: quantization of visual descriptors

Bag of words place recognition

Putting it all together

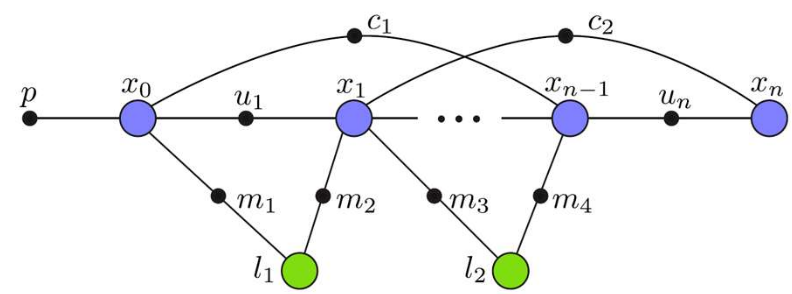
ORB-SLAM2 system architecture



Putting it all together

Summary

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



- Under the assumption of additive Gaussian noise:

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

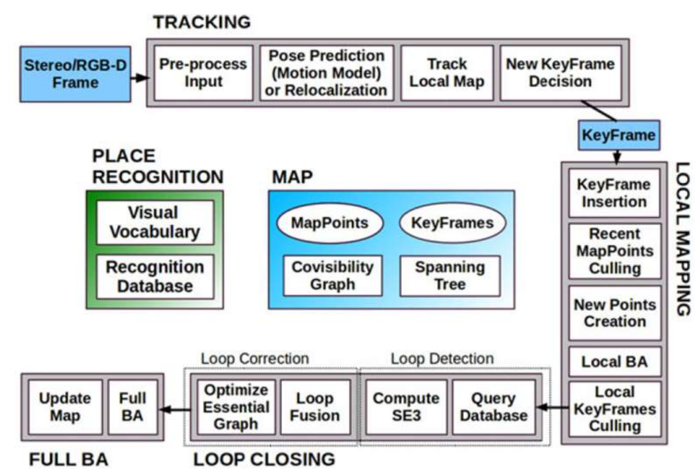
$$p(Z|\Theta) = \prod_i p_i(Z_i|\,\Theta_i)$$

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

Maximum likelihood estimation reduces to NLS:

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

- 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!