

Practical introduction to **Factor Graphs** with **GTSAM**

José Luis Blanco Claraco
Engineering Department
University of Almería

May, 10th 2022

Our research group at UAL

Automatic Control, Robotics and Mechatronics TEP-197

<http://arm.ual.es>

<http://www.ciesol.es/en/content/modeling-and-automatic-control>

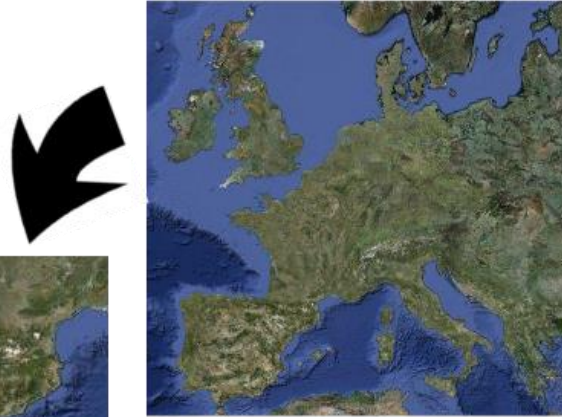


Escuela Superior de Ingeniería - Universidad de Almería
Modeling and Automatic Control Unit at CIESOL R&D Center



UNIVERSITY OF ALMERÍA

Almería: around
250.000 people



New University (1995←Granada)
Centralized campus



Students: >12.000

Teaching staff: 1.000

Administration staff: 550

Agreements UNIBS:

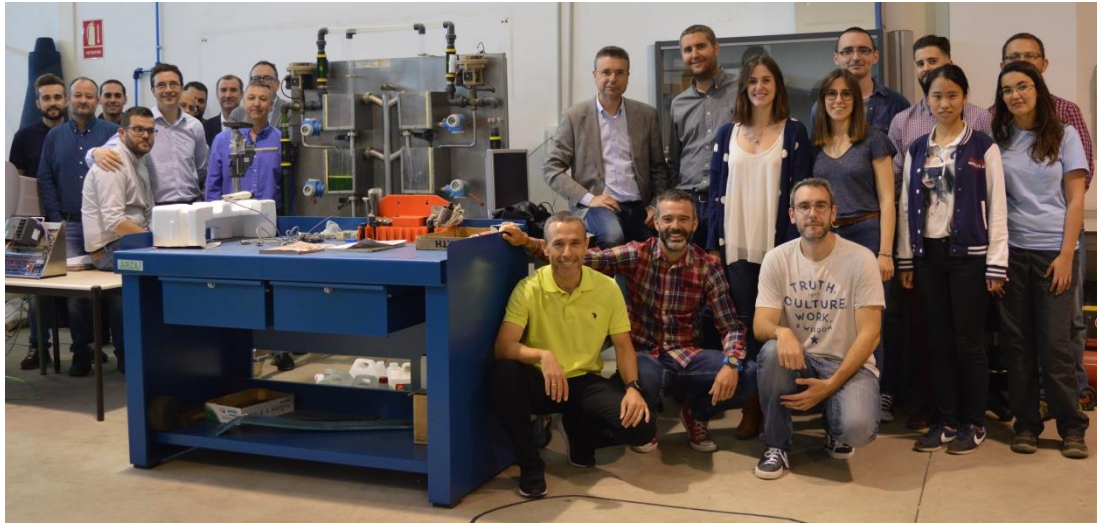
- Double degree
- ERASMUS
- Master Thesis

(Prof. Antonio Visioli)

Average monthly cost of
living in Almería: < 500 €

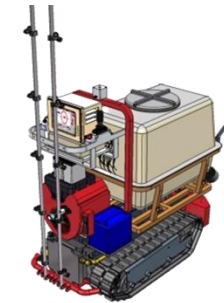
Grade on **Industrial
Electronics**

Automatic control, Robotics and Mechatronics



Automatic control, Robotics and Mechatronics

- **Modeling and control of solar plants**
- **Modeling, control and robotics in agriculture**
- **Energy efficiency and comfort control in buildings**
- **Modeling and control of photobioreactors**
- **Big-Data e IoT, open data and data sharing and management applied to agrifood and energy sectors**
- **Education in Engineering**
- **Autonomous vehicles and robots**
- **Design of robots**
- **Vehicle dynamics**
- **Vibration analysis**



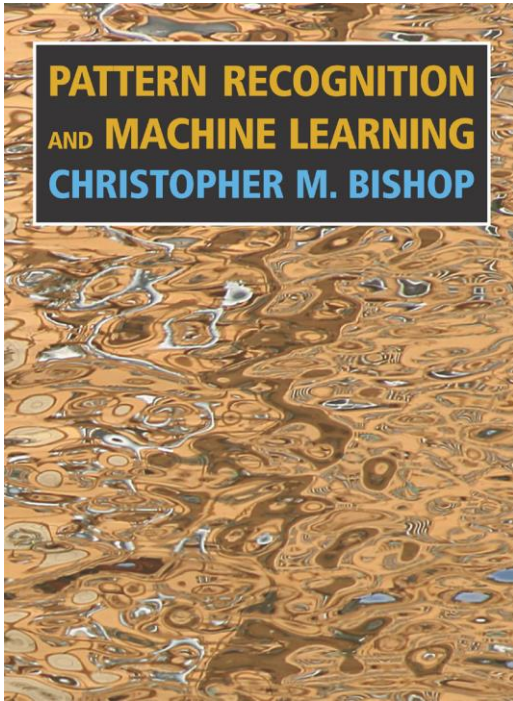
Automatic control, Robotics and Mechatronics

	UAL	PSA	Shared	OVERALL
Ph. D.	14	3		17
Predocctoral students	10		2	12
External members	2 (1 ES, 1 PT)	1 (AUS)		3
Overall				32

External collaborators	15 (ES), 6 (INT)
------------------------	------------------

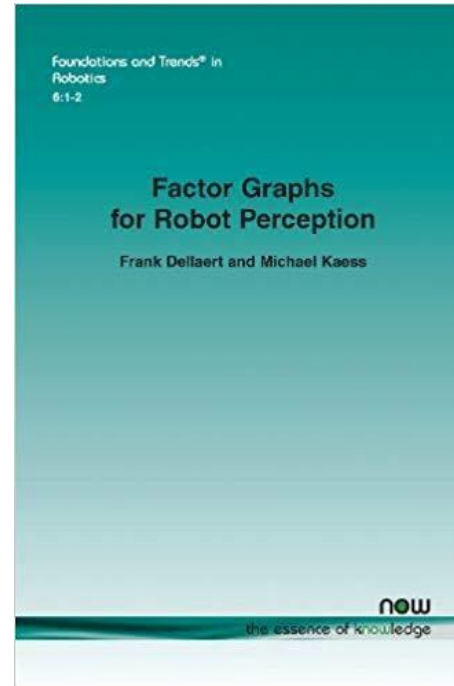
GTSAM & Factor graphs references

References



C.M. Bishop
(Microsoft)

“Pattern Recognition and Machine Learning”
Christopher M. Bishop, **2006**



Frank Dellaert
(Georgia Tech)



Michael Kaess
(Carnegie Mellon University)

“Factor Graphs for Robot Perception”
Frank Dellaert, Michael Kaess, **2017**

Probability and estimation bases

Probability Density Function (PDF) examples

Model of a PT-100 temperature probe

y : output [V]

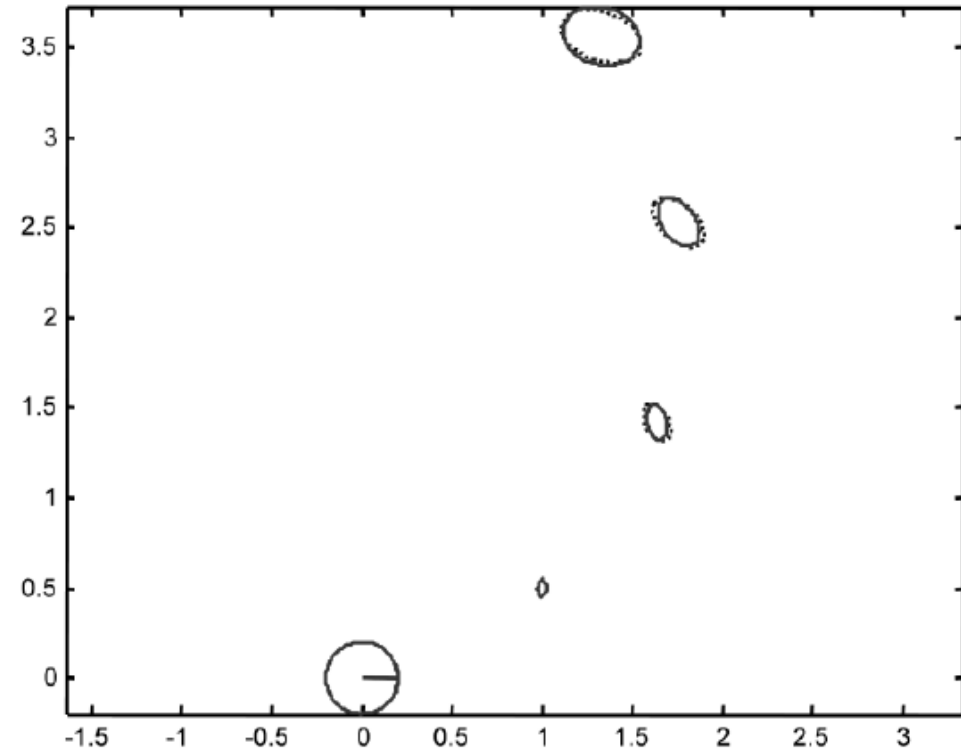
x : actual temperature [K]

n : random additive noise

$$y = f(x) + n$$

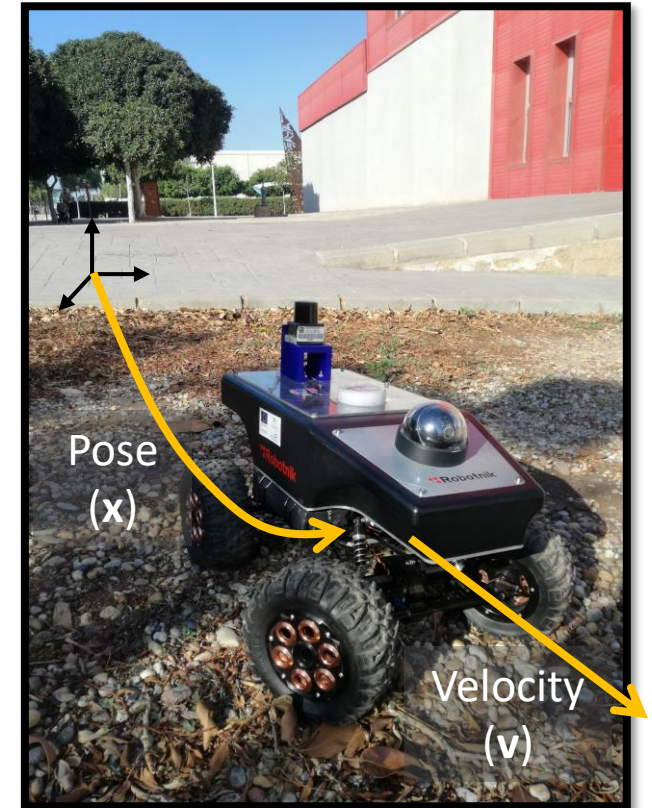
$$n \sim N(0, \sigma^2)$$

Wheeled robot motion model

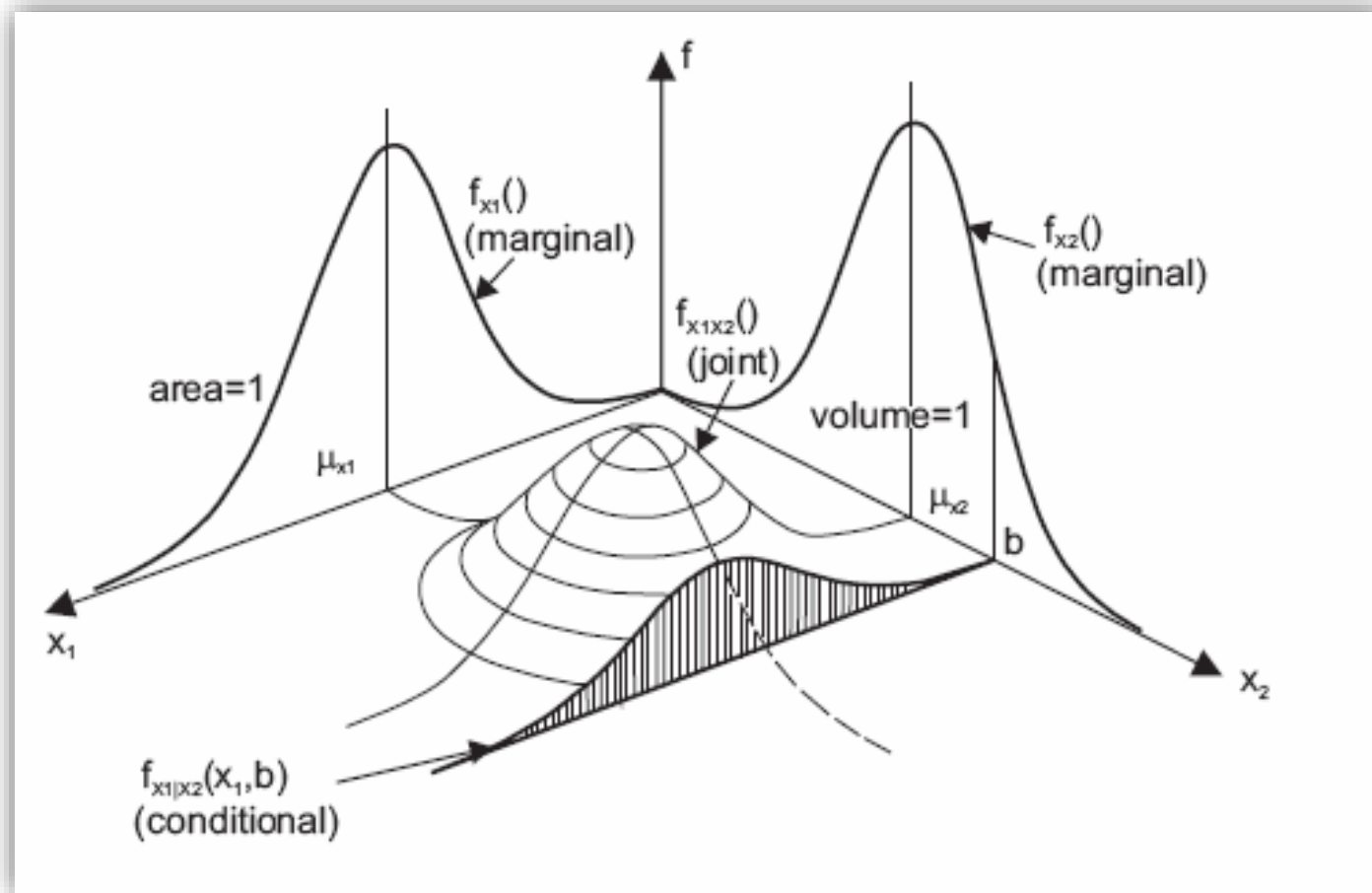


PDFs of several variables: concepts

- Marginal pdfs: $p(\mathbf{x})$
 $p(\mathbf{v})$
- Joint pdf: $p(\mathbf{x}, \mathbf{v})$
- Conditional pdf: $p(\mathbf{x} | \mathbf{v} = \mathbf{v}_0)$



PDFs of several variables: concepts



Matos, Jose. (2008). UNCERTAINTY TREATMENT IN CIVIL ENGINEERING NUMERICAL MODELS.

Moment form:

$$\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

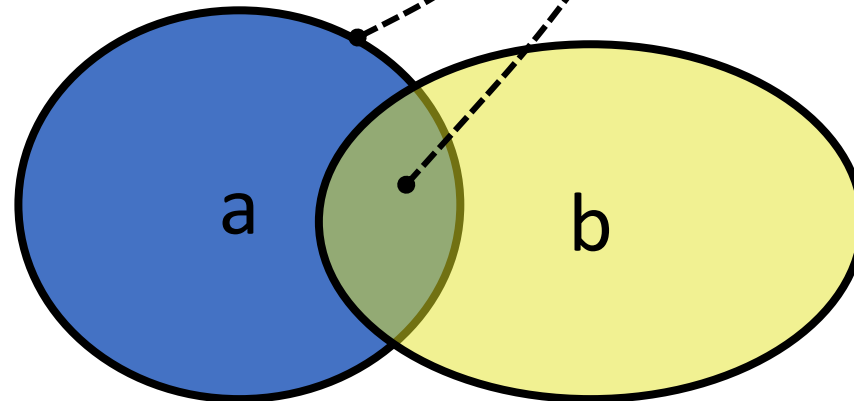
$$p(\mathbf{x}) = \text{constant} \cdot \exp\left(-\frac{1}{2}\|\mathbf{x} - \boldsymbol{\mu}\|_{\boldsymbol{\Sigma}}\right)$$

Relationship between joint and conditional

It can be verified that:

$$p(b|a) = \frac{p(a,b)}{p(a)}$$

$$\Leftrightarrow \overset{\text{Joint}}{p(a,b)} = \underset{\text{Conditional}}{p(b|a)} \overset{\text{Marginal}}{p(a)}$$



PDF factorization

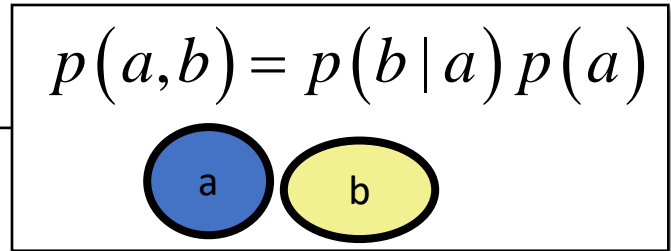
Given a joint pdf: $p(a, b, c, d, e)$

To factor it: splitting it into a **product** of **smaller** pdfs, e.g:

$$p(a, b, c, d, e) = p(a) p(b|c) p(d|e)$$

In large problems, we easily end up having 100s or 1000s of variables, so the dimensionality of the functions matter.

PDF factorization

$$p(a, b) = p(b | a) p(a)$$


Tools and rules to factorize:

- Independence: $p(a, b) = p(a) p(b) \quad a \perp b$
- Conditional independence: $p(a, b | c) = p(a | c) p(b | c) \quad a \perp b | c$

- Bayes' rule:

$$p(a | b) = p(a) \frac{p(b | a)}{p(b)}$$

- Law of total probability: $p(a) = \sum_{\forall b} p(a | b) p(b)$
("Marginalize b out")

Maximum a posterior (MAP) estimation

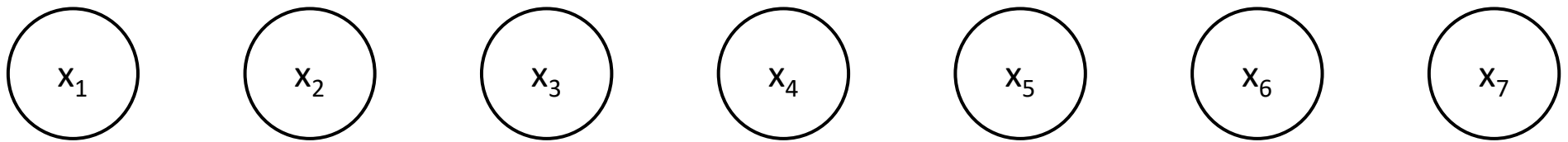
Find the set of parameters \mathbf{X} that maximize the likelihood of a set of observations \mathbf{Z} .

$$\begin{aligned}\mathbf{x}^* &= \arg \max_{\mathbf{x}} p(\mathbf{x} | \mathbf{z}) \\ &= \arg \max_{\mathbf{x}} l(\mathbf{z}; \mathbf{x}) \\ &= \arg \min_{\mathbf{x}} \underbrace{-\log(l(\mathbf{z}; \mathbf{x}))}_{\text{Log-likelihood}}\end{aligned}$$

→ Estimation as a **numerical optimization** problem! $\mathbf{x}^* = \arg \min_{\mathbf{x}} f(\mathbf{x})$

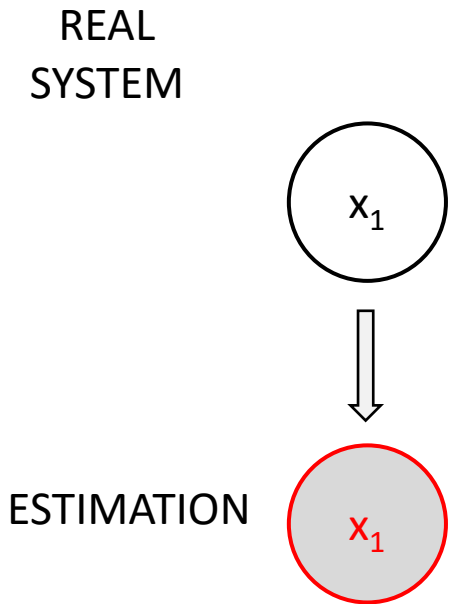
Types of estimators for dynamic systems

- A dynamic system has a state vector \mathbf{x} that evolves over time:



Types of estimators for dynamic systems

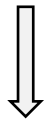
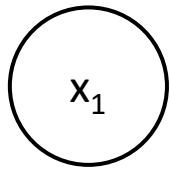
- **Filters**: They just estimate the last (“current”) system state.



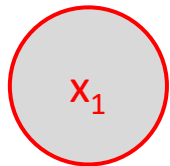
Types of estimators for dynamic systems

- **Fixed-lag smoother**: They estimate the last “n” system states.

REAL
SYSTEM



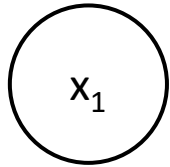
ESTIMATION



Types of estimators for dynamic systems

- **Batch estimator**: Obtain results after processing the entire dataset.

REAL
SYSTEM



ESTIMATION

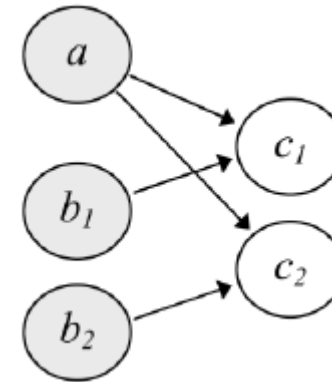
Graphical models. Bayes net

Why using graphs?

Algebraic manipulation

$$\begin{aligned}
 & \underbrace{p(x_t, m | z^t, u^t)}_{\text{Posterior for } t} && \text{Bayes rule on } z_t \\
 & && \propto \\
 & p(z_t | x_t, m, z^{t-1}, u^t) p(x_t, m | z^{t-1}, u^t) && z_t \perp\!\!\!\perp z^{t-1}, u^t \mid x_t, m \\
 & \underbrace{p(z_t | x_t, m)}_{\text{Observation model}} p(x_t, m | z^{t-1}, u^t) && \text{Law of total probability on } x_{t-1} \text{ to obtain recursivity} \\
 & p(z_t | x_t, m) \int_{-\infty}^{\infty} p(x_t, m | z^{t-1}, u^t, x_{t-1}) p(x_{t-1} | z^{t-1}, u^t) dx_{t-1} && x_t \perp\!\!\!\perp m \mid x_{t-1}, u_t \\
 & z_t | x_t, m \int_{-\infty}^{\infty} \underbrace{p(x_t | z^{t-1}, u^t, x_{t-1}) p(m | z^{t-1}, u^t, x_{t-1})}_{\text{Factored due to conditional independence}} p(x_{t-1} | z^{t-1}, u^t) dx_{t-1} && = \\
 & z_t | x_t, m \int_{-\infty}^{\infty} p(x_t | z^{t-1}, u^t, x_{t-1}) \underbrace{p(m | z^{t-1}, u^t, x_{t-1}) p(x_{t-1} | z^{t-1}, u^t)}_{\text{These terms can be joined}} dx_{t-1} && p(a|b)p(b) = p(a, b) \\
 & p(z_t | x_t, m) \int_{-\infty}^{\infty} p(x_t | z^{t-1}, u^t, x_{t-1}) p(x_{t-1}, m | z^{t-1}, u^t) dx_{t-1} && u_t \perp\!\!\!\perp x_{t-1}, m \mid \emptyset \\
 & p(z_t | x_t, m) \int_{-\infty}^{\infty} p(x_t | z^{t-1}, u^t, x_{t-1}) \underbrace{p(x_{t-1}, m | z^{t-1}, u^{t-1})}_{\text{Posterior for } t-1} dx_{t-1} && x_t \perp\!\!\!\perp z^{t-1}, u^{t-1} \mid u_t, z \\
 & p(z_t | x_t, m) \int_{-\infty}^{\infty} \underbrace{p(x_t | u_t, x_{t-1})}_{\text{Transition model}} p(x_{t-1}, m | z^{t-1}, u^{t-1}) dx_{t-1} &&
 \end{aligned}$$

Graphical models



Definition

The formalism of **graphical models** allows us to represent:

- a set of r.v.s and
- their conditional independence assumptions.

It was born as a fusion of **probability theory** and **graph theory**, and plays a central role in many machine learning techniques (Bishop, 2006).

Taxonomy

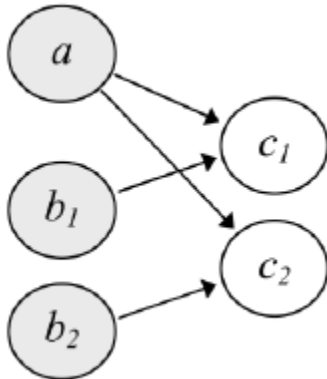
(Non-exhaustive) taxonomy of graphical models:

- Bayesian Network (BN)
- Dynamic BN (BN)
- Markov Random Field (MRF)
- Factor graphs

Purpose

Probabilistic graphical models provide a mechanism to compactly describe complex probability densities by exploiting the **structure** in them.

→ Efficient factorizations of PDFs



Edges in a graph are important, but the **lack of them** is what is even more relevant.

Main application in our scope: **inference**.

Bayesian Network (BN)

- A **directed acyclic** graphical model.
- **Nodes** represent variables: both, knowns and unknowns.
- **Directed edges** carry a semantic meaning of **causality**.

Allow **encoding a human expert's knowledge** as a highly-efficient, sparse probabilistic model



Bayesian Network (BN)

Mathematically, a BN encodes a factorization:

$$p(\mathbf{X}) = \prod_i p(x_i | \mathbf{p}_i)$$

$\mathbf{X} = \{x_1, \dots, x_n\}$: all variables

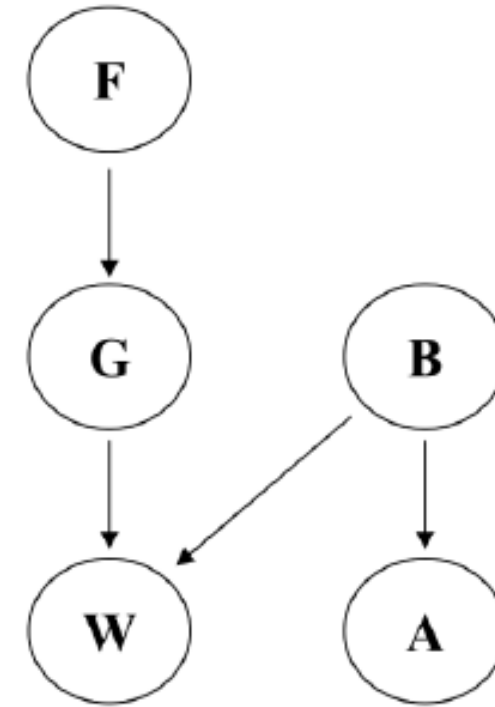
x_i : the i 'th variable

\mathbf{p}_i : parents of the i 'th variable

Bayesian Network (BN)

$$p(\mathbf{X}) = \prod_i p(x_i | \mathbf{p}_i)$$

$$p(A, B, G, W, F) = ?$$



F: Car was refueled

G: Car has gas

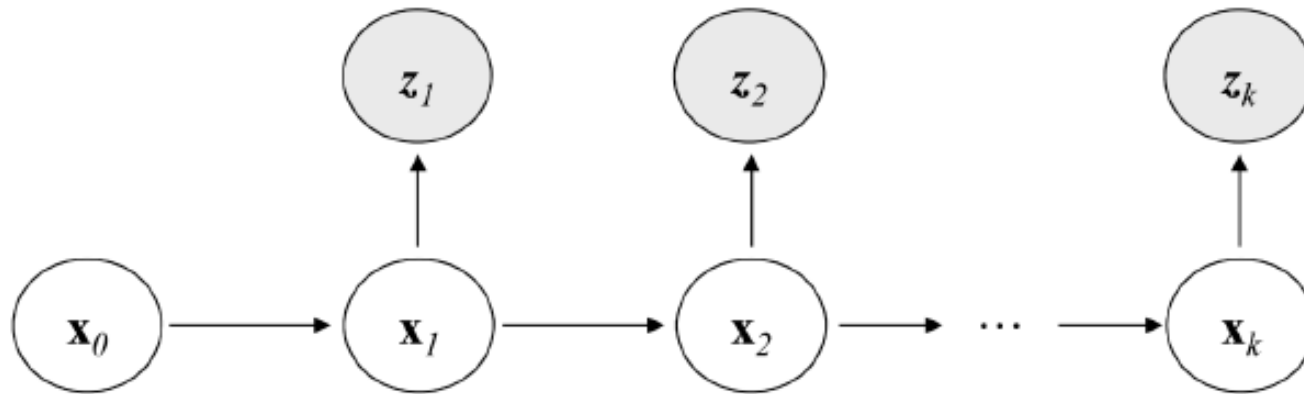
B: Car batteries are OK

W: Car starts

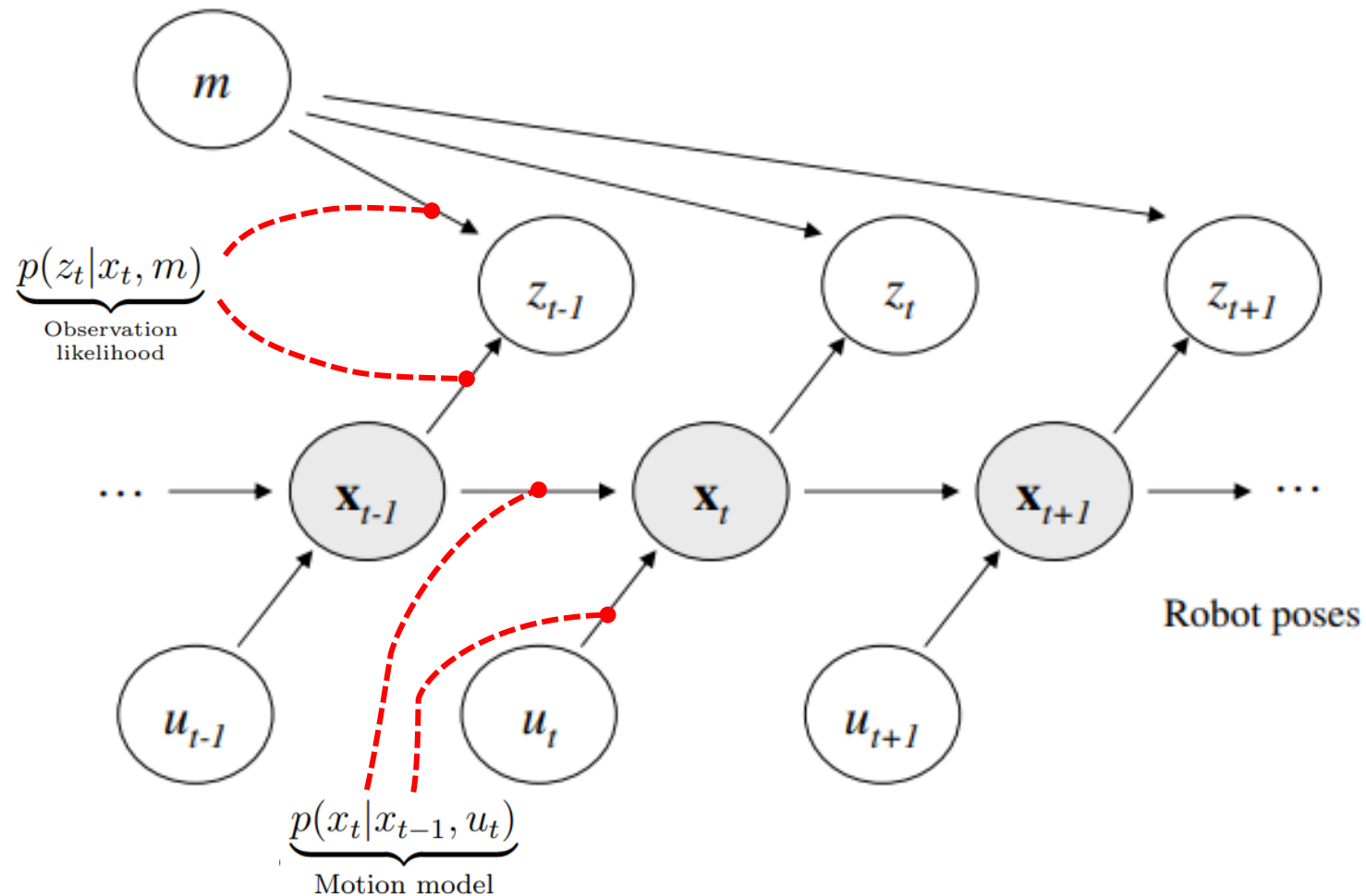
A: Car audio works

Dynamic Bayesian Network (DBN)

A BN over variables that have a dynamic state over time. Implicit Markov property.



DBN example: vehicle localization



Graphical models. Markov random field

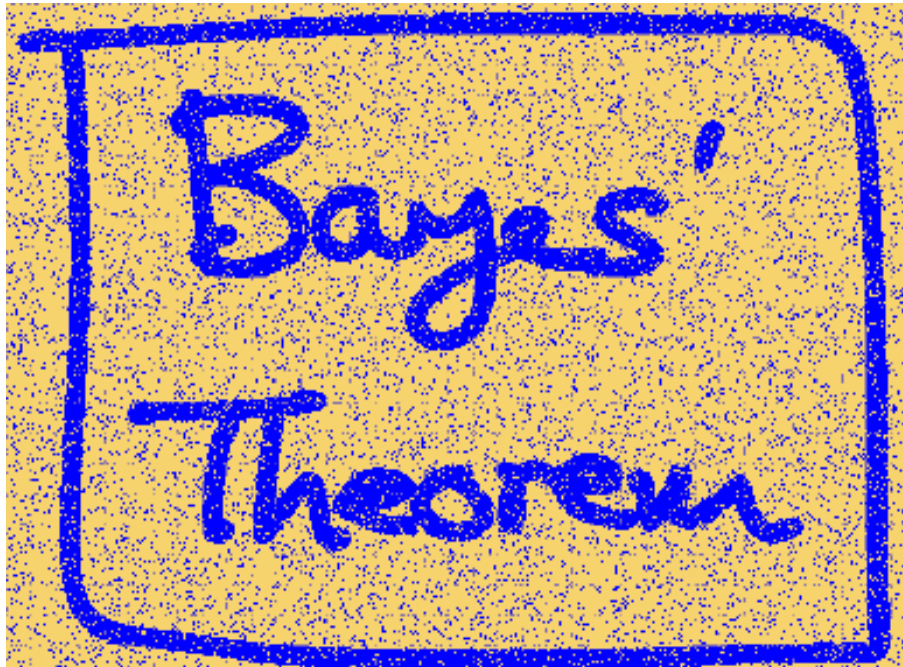
Markov Random Field (MRF)

- An **undirected** graphical model. May (and often will) contain loops.
- **Nodes** represent variables: both, knowns and unknowns.
- **Edges** does **not** carry information about causality, only about some “relationship”.

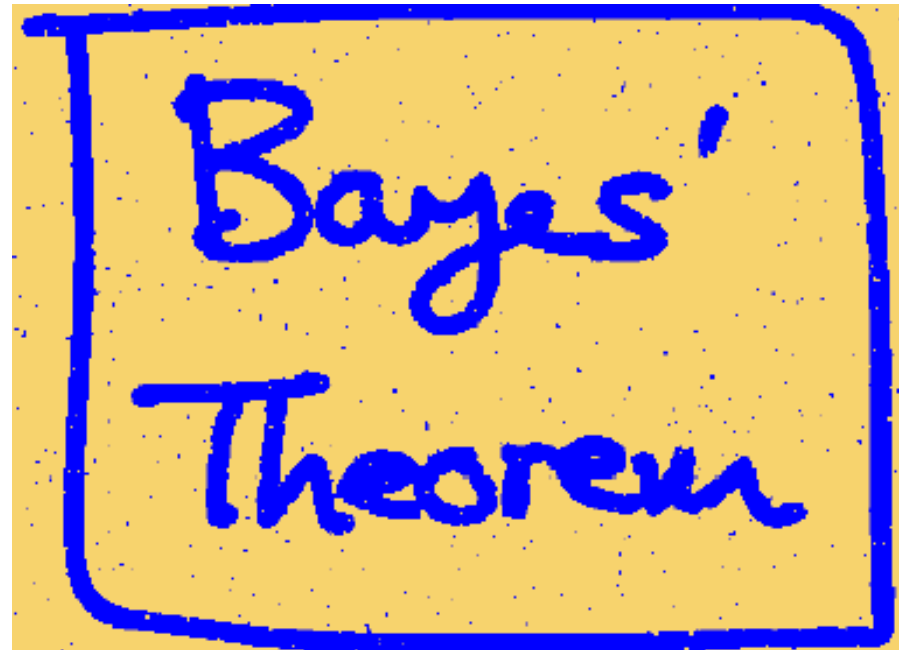
A DBN can be converted into a MRF (not covered here).

MRF applications: image de-noising

Noisy input



Restored

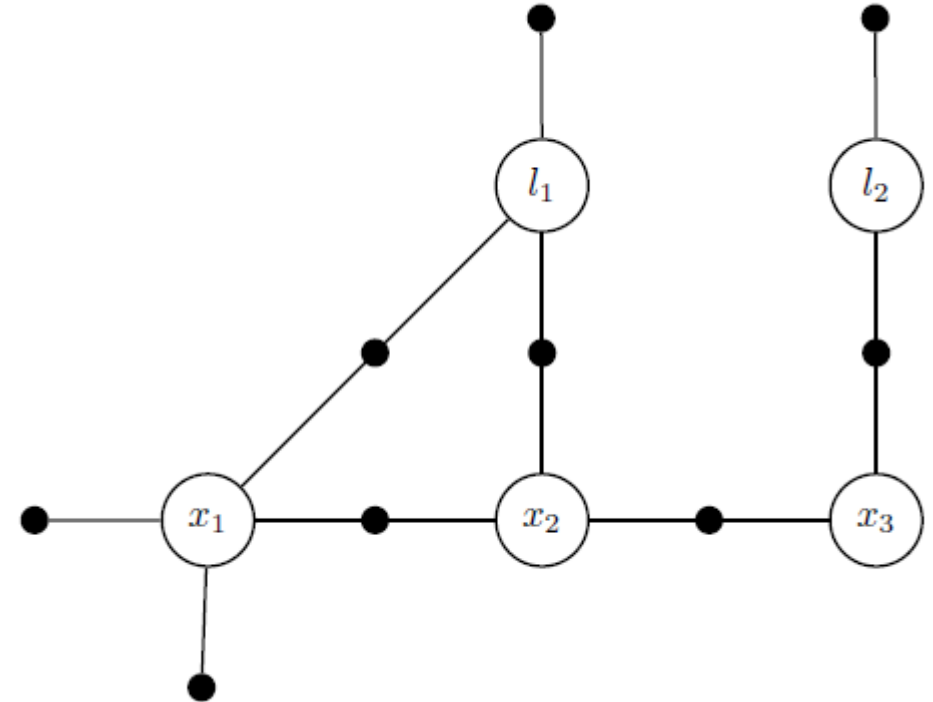


Factor graphs

Factor graph (FG)

- A **bi-partite** graphical model.
Can contain loops.
- **Two types of nodes:** variables and factors.
- **Undirected edges:** only between variables and factors.

Unary, binary,... n-ary factors.



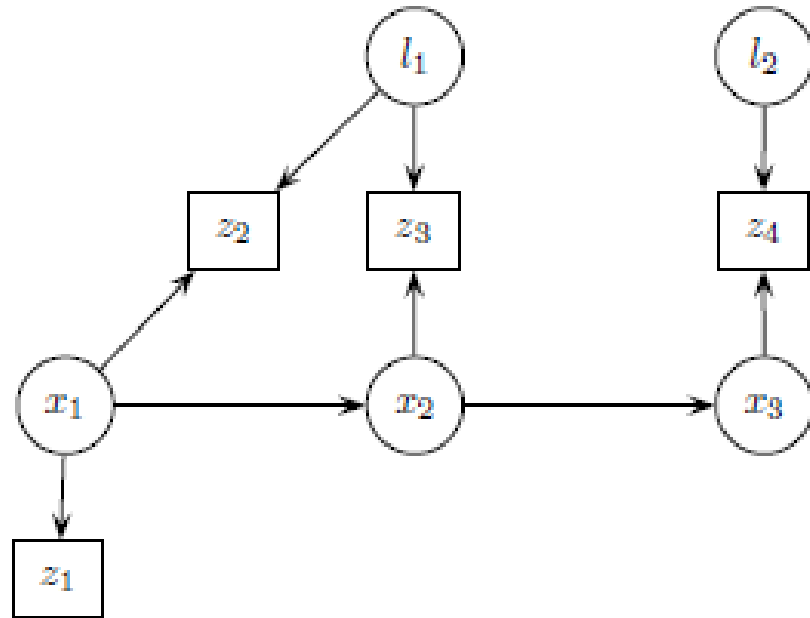
"Factor Graphs for Robot Perception"
Frank Dellaert, Michael Kaess, 2017

Bayes Network → Factor graph

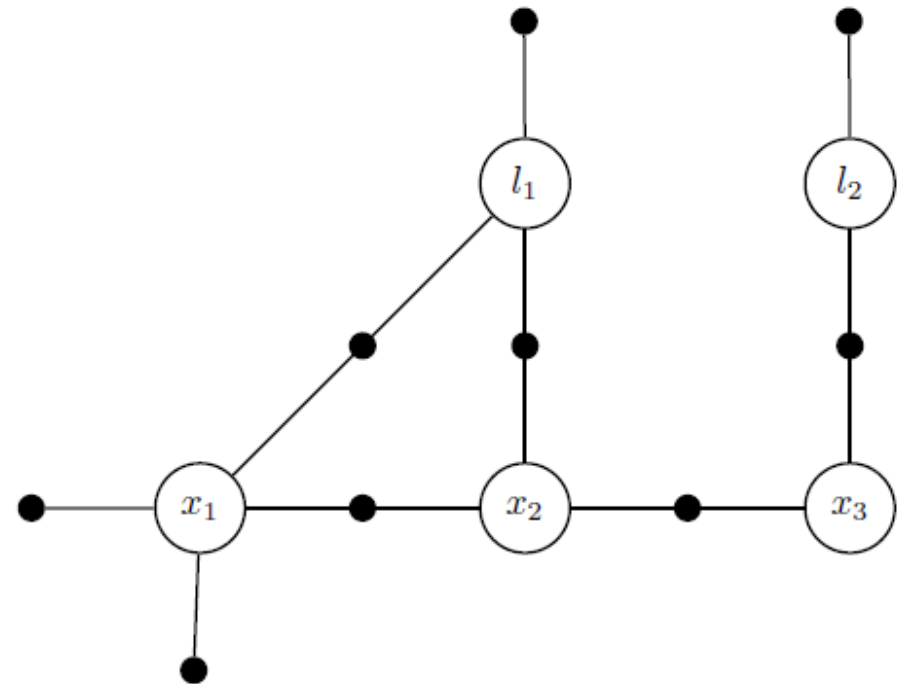
- “every node in a Bayes net denotes a conditional density on the corresponding variable and its parent nodes. Hence, the conversion is quite simple: every Bayes net node splits in both a variable node and a factor node in the corresponding factor graph. **The factor is connected to the variable node**, as well as the variable nodes corresponding to the **parent nodes** in the Bayes net. If some nodes in the Bayes net are evidence nodes, i.e., they are given as known variables, we omit the corresponding variable nodes: the known variable simply becomes a fixed parameter in the corresponding factor.” [Dellaert&Kaess, 2017]

Bayes Network \rightarrow Factor graph

Bayes Network



Factor graph



Factor graph (FG) factorization

Mathematically, a FG encodes a factorization:

$$p(\mathbf{X}) = \prod_i f_i(\mathbf{x}_i)$$

$\mathbf{X} = \{x_1, \dots, x_n\}$: all variables

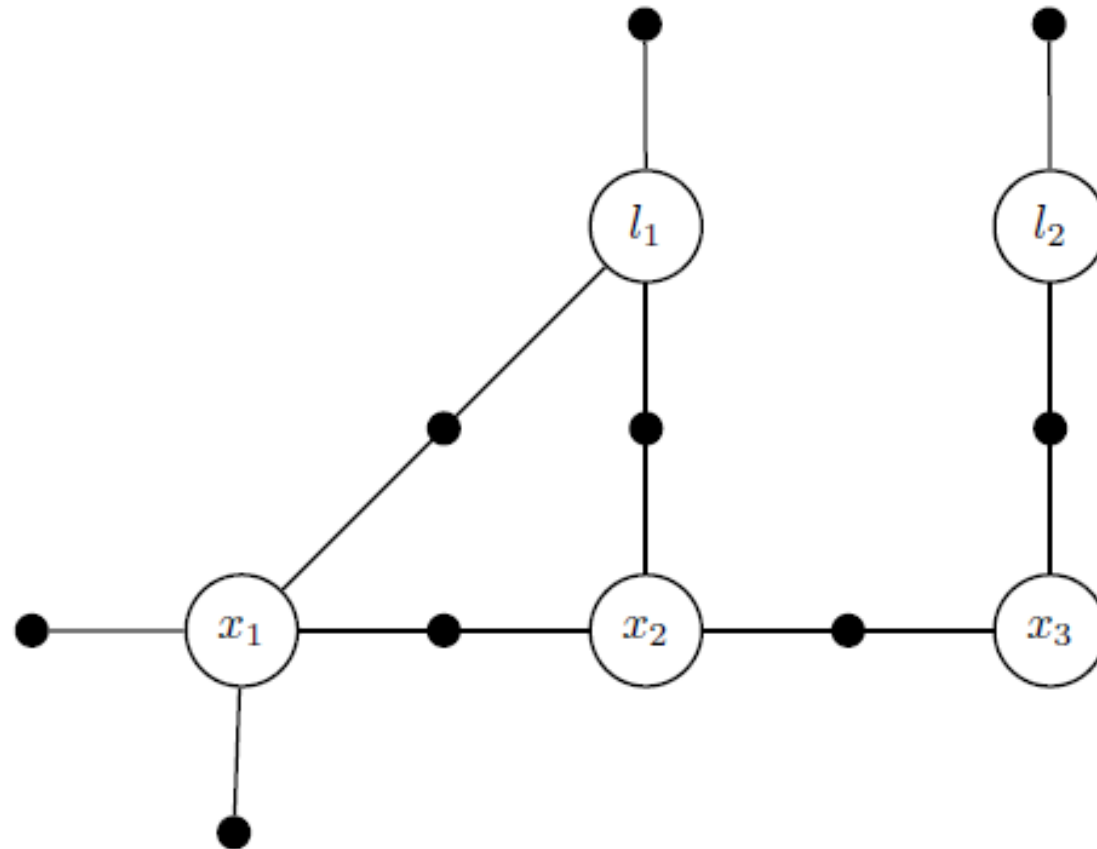
\mathbf{x}_i : all variables touching the i 'th factor

f_i : the i 'th factor

Factor graph (FG) factorization

$$p(\mathbf{X}) = \prod_i f_i(\mathbf{x}_i)$$

$$p(\mathbf{X}) = \dots$$



Uses of FGs

- Sampling.
- Evaluation.
- Optimization:
 - **MAP estimator.** Gradient-based (Gauss-Newton).
 - How to do efficiently? (→ Literature)
 - Message-based optimization (not discussed here)

Error models

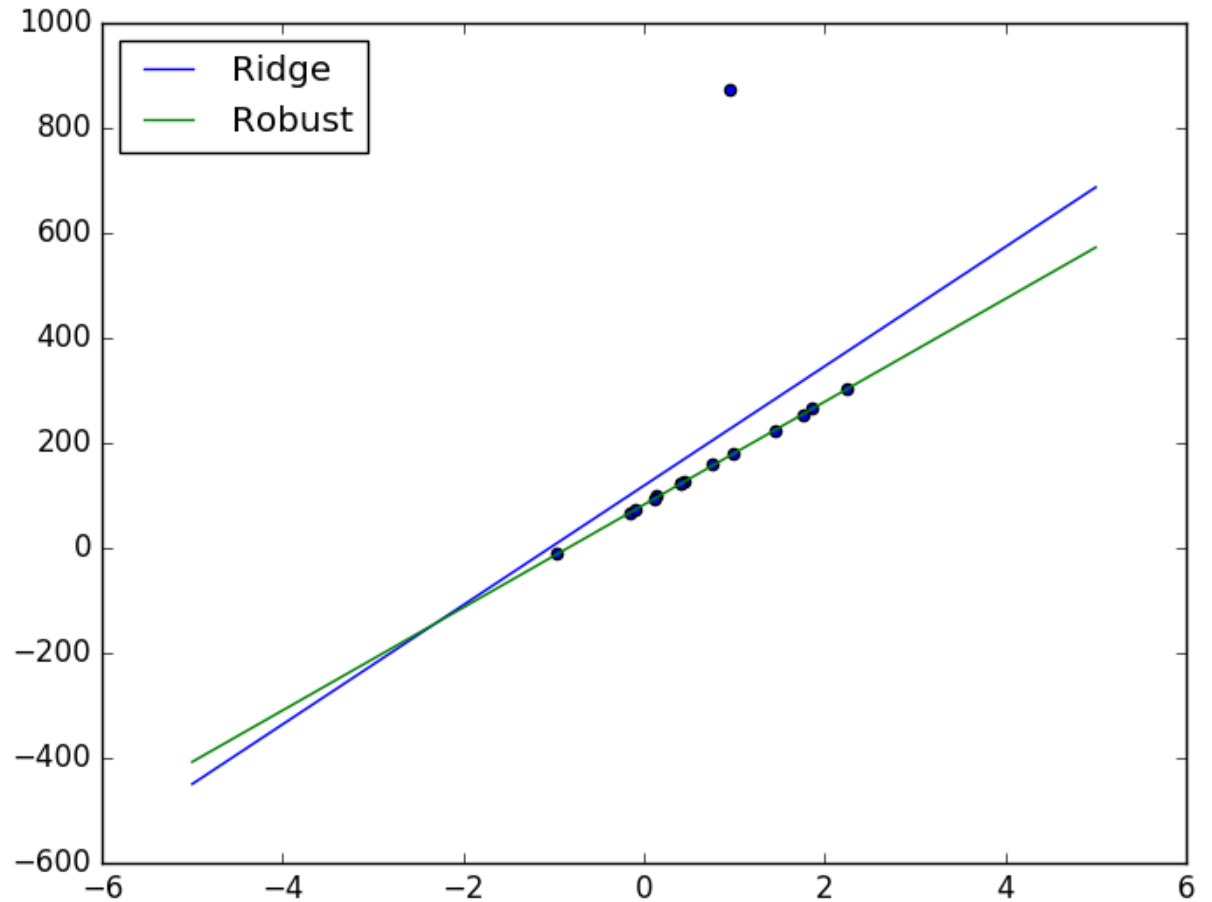
- Each factor model a constraint, and must include a measure of uncertainty, confidence. The “strength” in the mass-spring model.
- Most common model: Gaussian.

$$y = f(x) + n \quad \rightarrow \quad e(\mathbf{x}) = \|f(\mathbf{x}) - \mathbf{z}\|_{\Sigma} = (f(\mathbf{x}) - \mathbf{z})^T \Sigma^{-1} (f(\mathbf{x}) - \mathbf{z})$$

Problem \rightarrow Too strict with outliers!

Error models: robust kernels

- Just **one outlier** is enough to ruin a least-squares estimator.

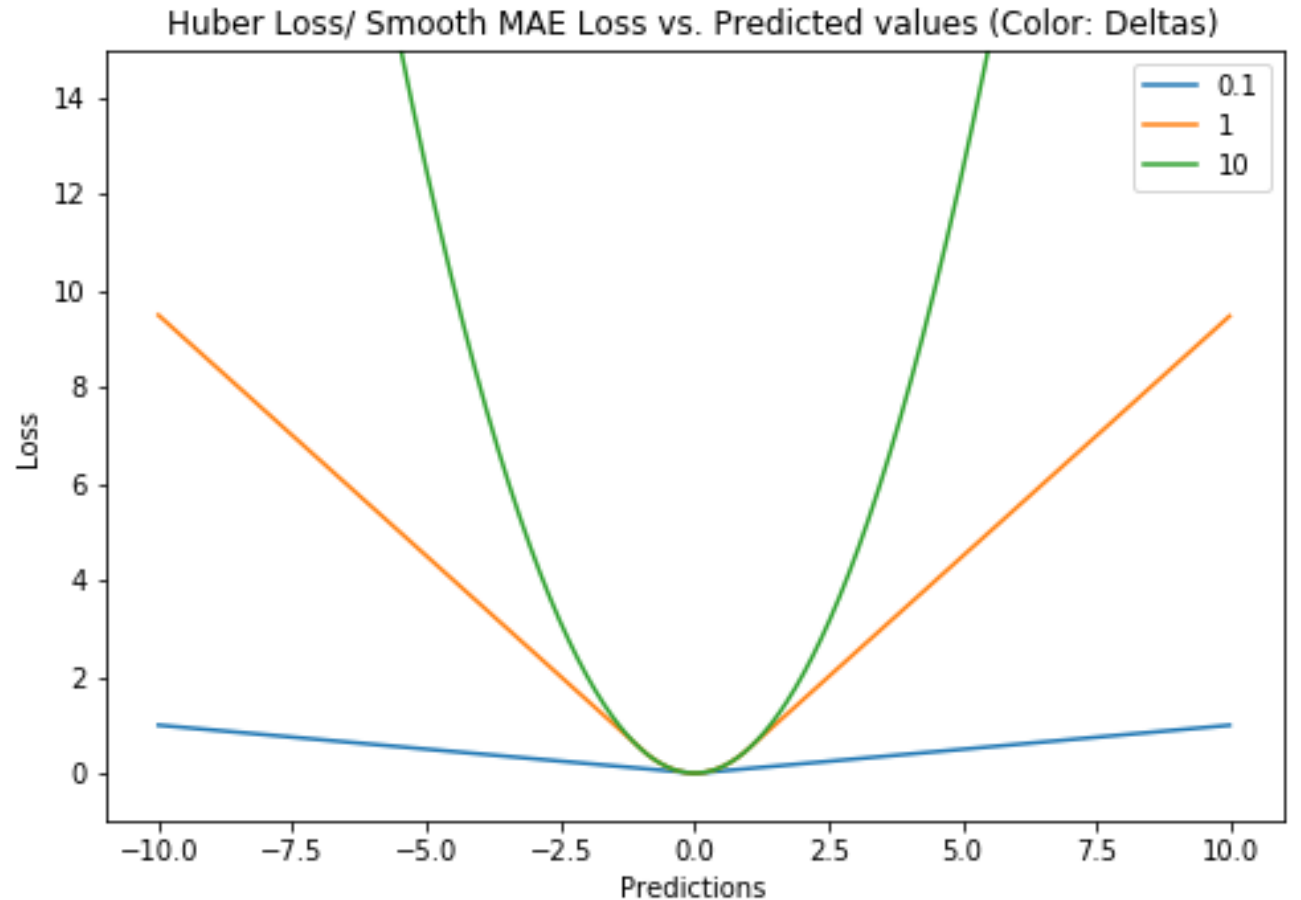


Error models: robust kernels

Robust **M-estimators**: based on modified cost functions.

Well-known kernels:

- Pseudo-Huber,
- Cauchy,
- Geman-McClure,
- Tukey.



Relationship between graphs and sparse matrix Algebra

Motivation

- Linearization \rightarrow Jacobian of factors.
- Variable ordering \rightarrow Important.
- Learn about the most common nonlinear algorithms: GN, LM, DL, etc.

Nonlinear factor graphs

MAP of nonlinear factor graphs becomes iteratively solving **Ax=b**

$$\begin{aligned} X^{MAP} &= \operatorname{argmax}_X \phi(X) & \phi_i(X_i) &\propto \exp \left\{ -\frac{1}{2} \|h_i(X_i) - z_i\|_{\Sigma_i}^2 \right\} \\ &= \operatorname{argmax}_X \prod_i \phi_i(X_i). \end{aligned}$$

$$X^{MAP} = \operatorname{argmin}_X \sum_i \|h_i(X_i) - z_i\|_{\Sigma}^2$$

"Factor Graphs for Robot Perception" Frank Dellaert, Michael Kaess, **2017**

Nonlinear factor graphs

Linearization:

$$h_i(X_i) = h_i(X_i^0 + \Delta_i) \approx h_i(X_i^0) + H_i \Delta_i,$$

$$H_i \triangleq \left. \frac{\partial h_i(X_i)}{\partial X_i} \right|_{X_i^0}$$

$$\begin{aligned} \Delta^* &= \operatorname{argmin}_{\Delta} \sum_i \left\| h_i(X_i^0) + H_i \Delta_i - z_i \right\|_{\Sigma_i}^2 \\ &= \operatorname{argmin}_{\Delta} \sum_i \left\| H_i \Delta_i - \{z_i - h_i(X_i^0)\} \right\|_{\Sigma_i}^2 \end{aligned}$$

“Factor Graphs for Robot Perception” Frank Dellaert, Michael Kaess, **2017**

Nonlinear factor graphs

Whitening:

$$\|e\|_{\Sigma}^2 \stackrel{\Delta}{=} e^{\top} \Sigma^{-1} e = \left(\Sigma^{-1/2} e \right)^{\top} \left(\Sigma^{-1/2} e \right) = \left\| \Sigma^{-1/2} e \right\|_2^2$$

$$\begin{aligned} A_i &= \Sigma_i^{-1/2} H_i \\ b_i &= \Sigma_i^{-1/2} \left(z_i - h_i(X_i^0) \right) \end{aligned}$$

$$\begin{aligned} \Delta^* &= \operatorname{argmin}_{\Delta} \sum_i \|A_i \Delta_i - b_i\|_2^2 \\ &= \operatorname{argmin}_{\Delta} \|A \Delta - b\|_2^2, \end{aligned}$$

Direct methods for LS:

- Cholesky (+ forward and back-substitution)
- QR

Iteration: Gauss-Newton, DogLeg, Lev-Marq.

"Factor Graphs for Robot Perception" Frank Dellaert, Michael Kaess, 2017

Sparsity

- Sparsity of Jacobians (and hence, Hessians $H=J'J$) are key for efficiency.

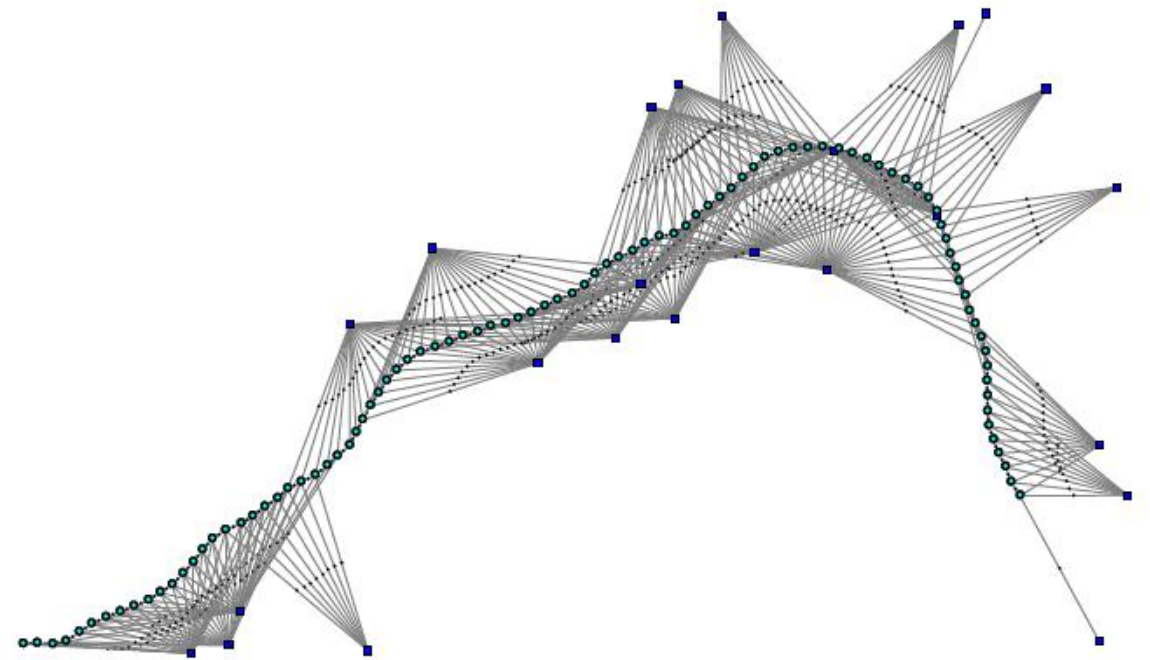
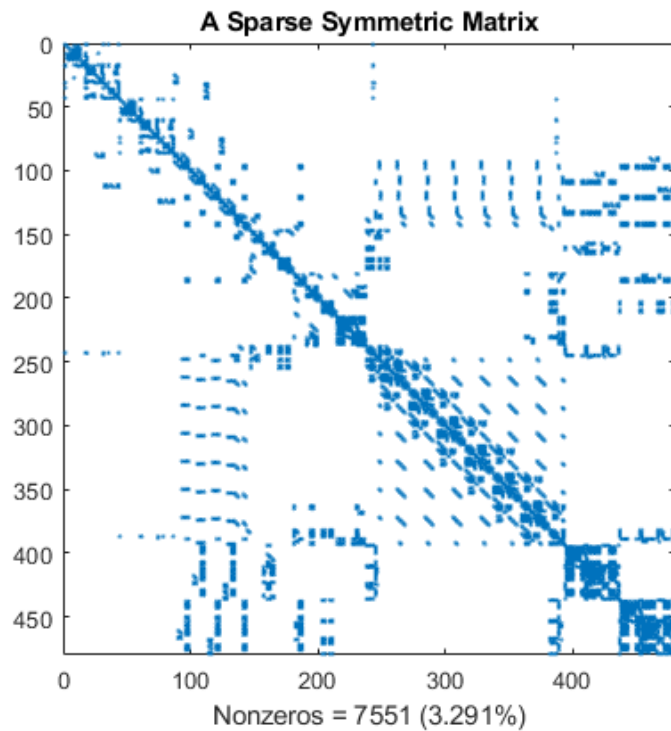
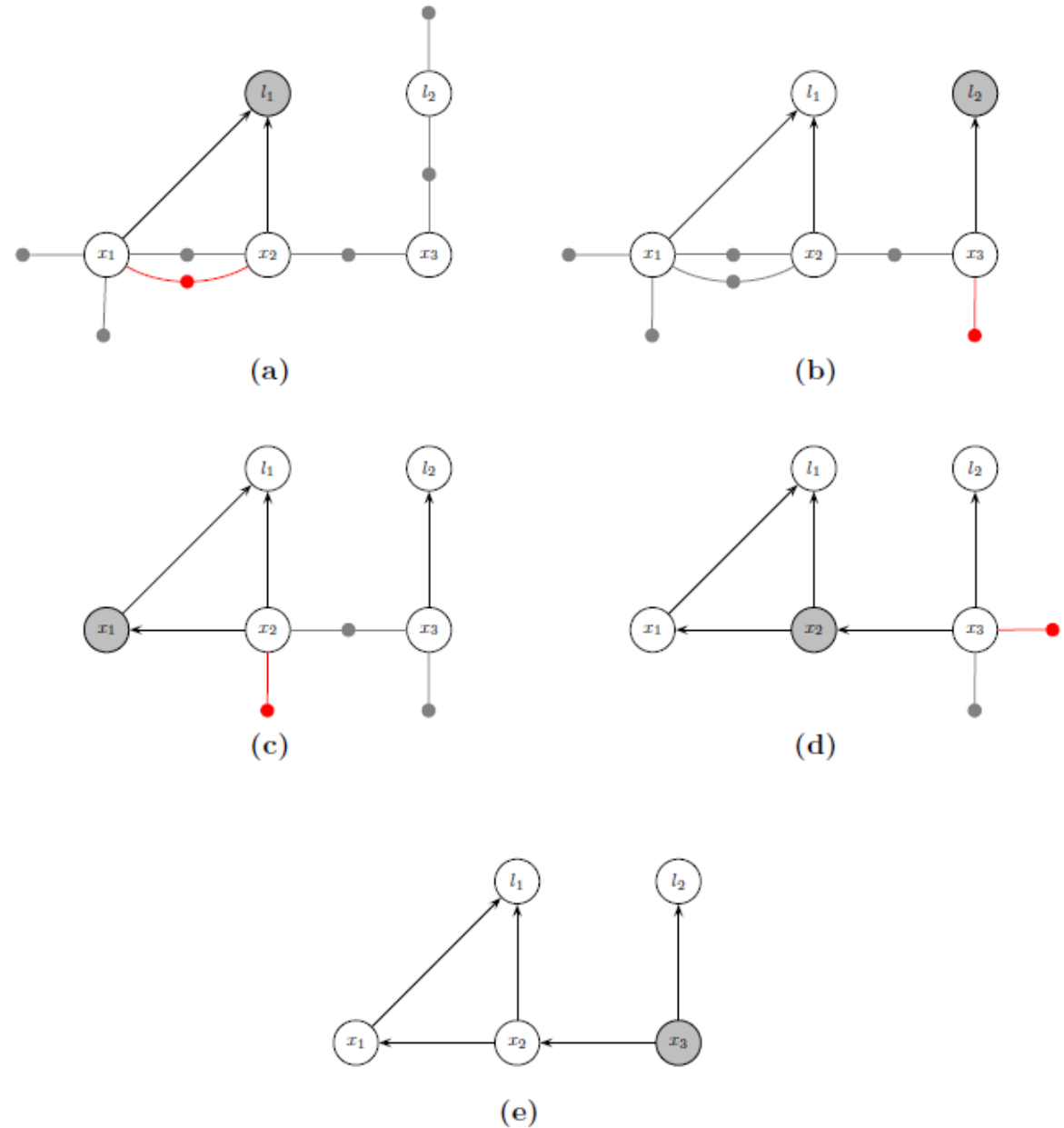
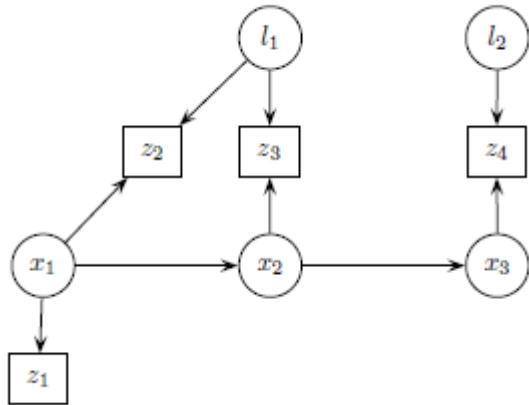


Figure 2.1: Factor graph for a larger, simulated SLAM example.

The elimination algorithm

- Example. Ordering: L1,L2,X1,X2,X3.

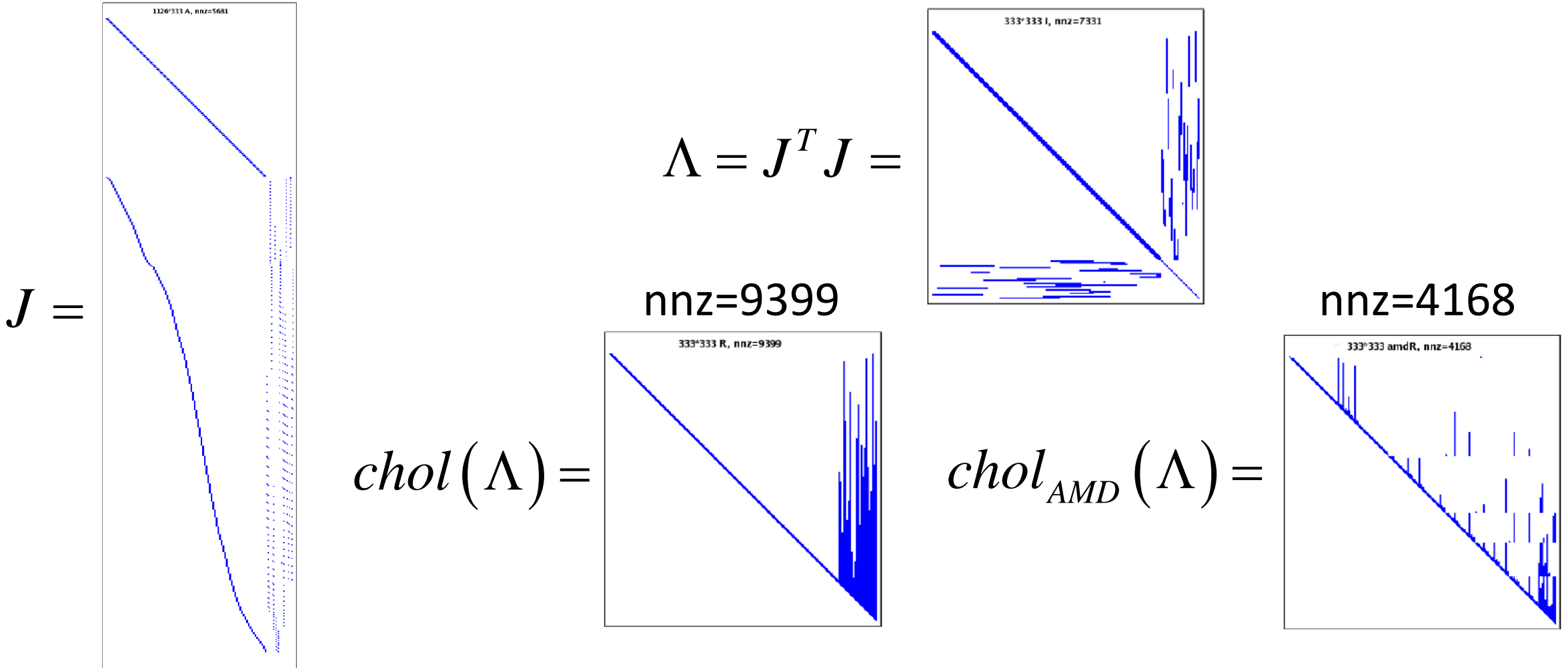
$$p(l_1, l_2, x_1, x_2, x_3) = p(l_1|x_1, x_2)p(l_2|x_3) \\ p(x_1|x_2)p(x_2|x_3)p(x_3)$$



The importance of ordering

- Dominating cost: factorization of sparse matrices for local factors while eliminating.

The importance of ordering



Example applications of FGs

Linear–quadratic regulator (LQR)

$$x_{k+1} = Ax_k + Bu_k$$

$$L_x(x_k) = x_k^T Q x_k$$

$$L_u(u_k) = u_k^T R u_k$$

$$\operatorname{argmin}_{u_{1 \sim k}} \sum_{i=1}^n x_i^T Q x_i + u_i^T R u_i$$

$$\text{s.t. } x_{t+1} = Ax_t + Bu_t \text{ for } t = 1 \text{ to } T - 1$$

(Based on GTSAM blog post by Gerry Chen and Yetong Zhang)

Linear–quadratic regulator (LQR)

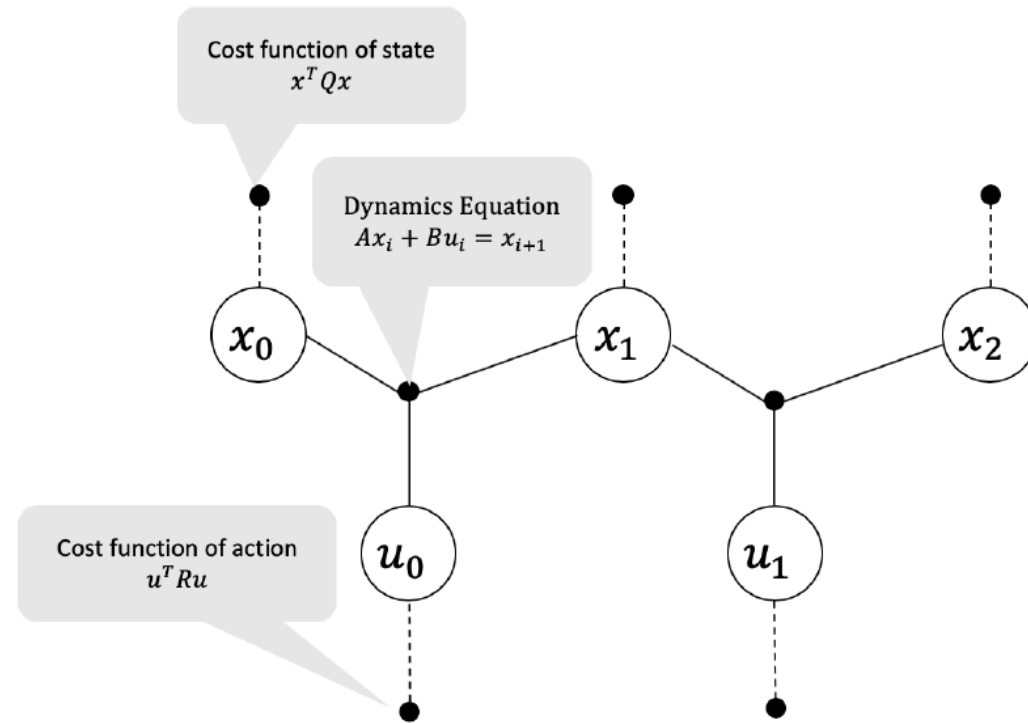
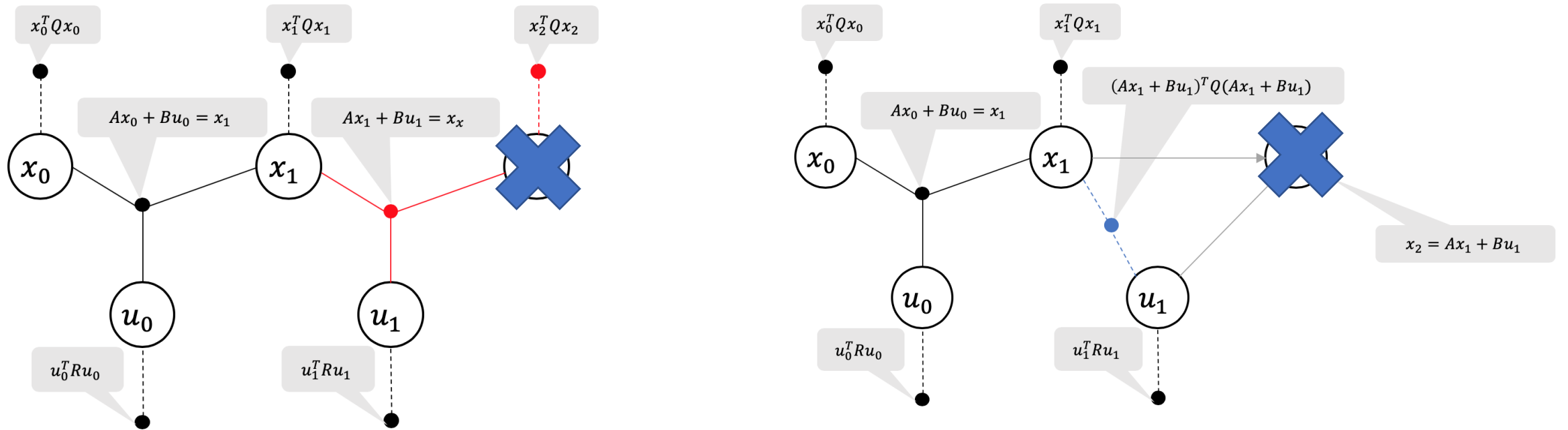


Figure 2 Factor graph structure for an LQR problem with 3 time steps. The cost factors are marked with dashed lines and the dynamics constraint factors are marked with solid lines.

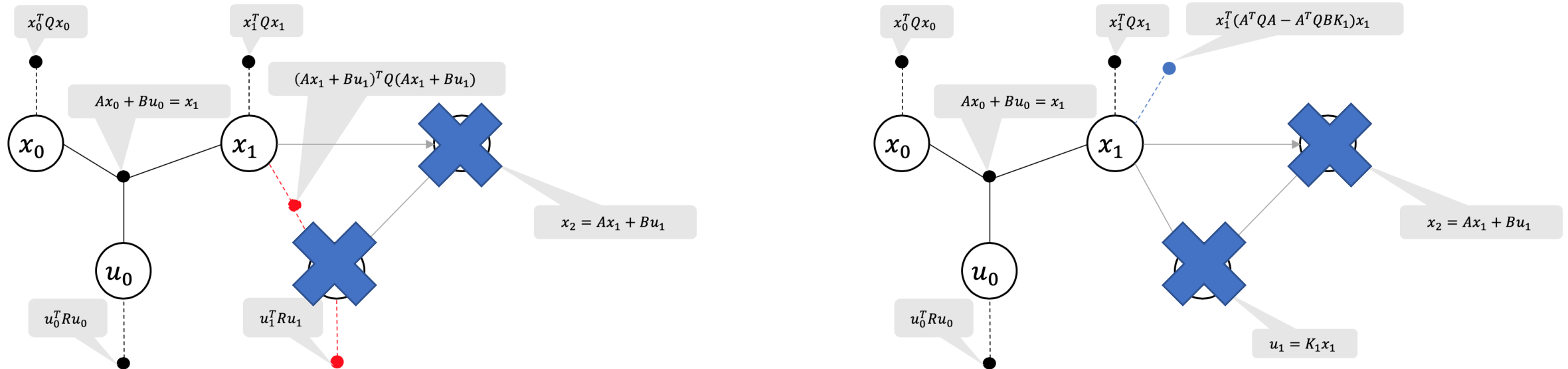
(Based on GTSAM blog post by Gerry Chen and Yetong Zhang)

Linear-quadratic regulator (LQR)



(Based on GTSAM blog post by Gerry Chen and Yetong Zhang)

Linear-quadratic regulator (LQR)



(Based on GTSAM blog post by Gerry Chen and Yetong Zhang)

Linear–quadratic regulator (LQR)

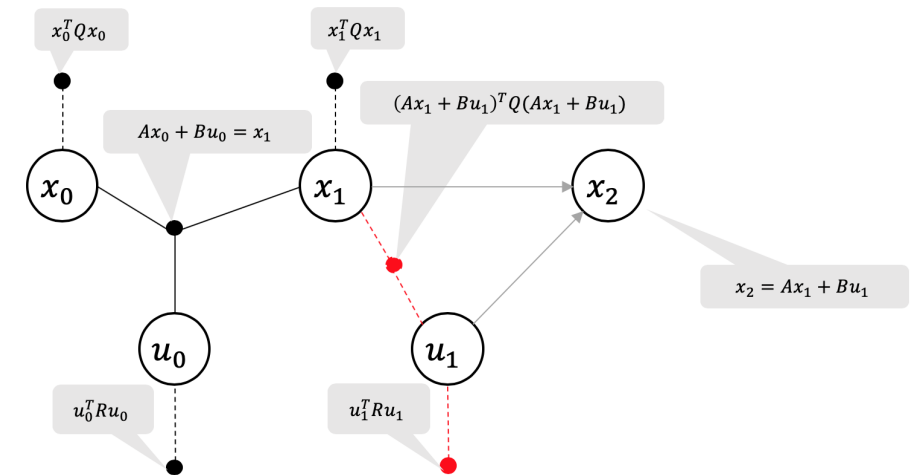
Adding the control cost (3) to (6), the combined cost of the two red factors in Figure 4a is given by:

$$\phi_3(x_1, u_1) = u_1^T R u_1 + (Ax_1 + Bu_1)^T Q (Ax_1 + Bu_1) \quad (7)$$

We minimize ϕ_3 by setting the derivative of (7) wrt u_1 to zero yielding the expression for the optimal control input u_1^* as

$$\begin{aligned} u_1^*(x_1) &= -(R + B^T Q B)^{-1} B^T Q A x_1 \\ &= K_1 x_1 \end{aligned} \quad (8)$$

where $K_1 := -(R + B^T Q B)^{-1} B^T Q A$.



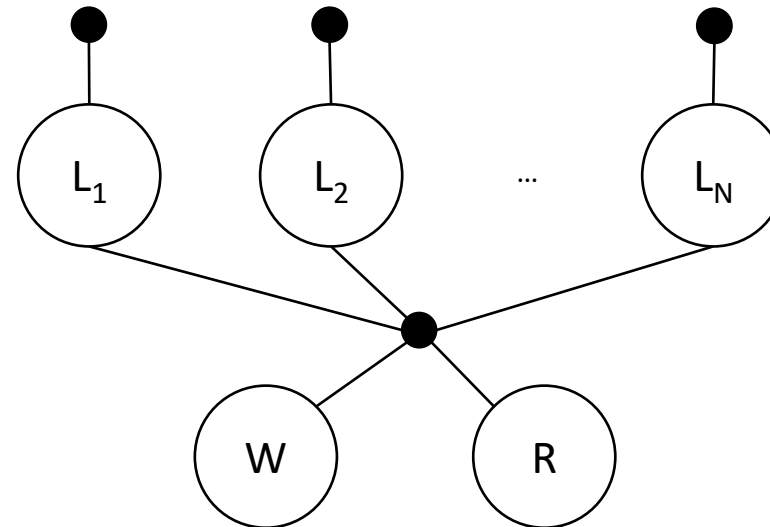
Equivalent to the well-known Ricatti equation solution.

(Based on GTSAM blog post by Gerry Chen and Yetong Zhang)

Nonintrusive load monitoring (NILM)

- Identifying which appliances are ON/OFF from electric signals, e.g. “load disaggregation”. W : real power, R : reactive power. $L_i = \text{ON/OFF}$.

$$p(\mathbf{L}, W, R) = p(W, R | \mathbf{L}) \{p(L_1) \cdot p(L_2) \cdots p(L_N)\}$$



Goutam, Y. G., Chandra, M. G., Srinivasarengan, K., & Kadhe, S. (2013, August). On electrical load disaggregation using factor graphs. In *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)* (pp. 1759-1764). IEEE.

Applications to robotics & vision

- Large systems of keyframes and observations.
- SLAM vs localization.
- SE(3) vs Euclidean

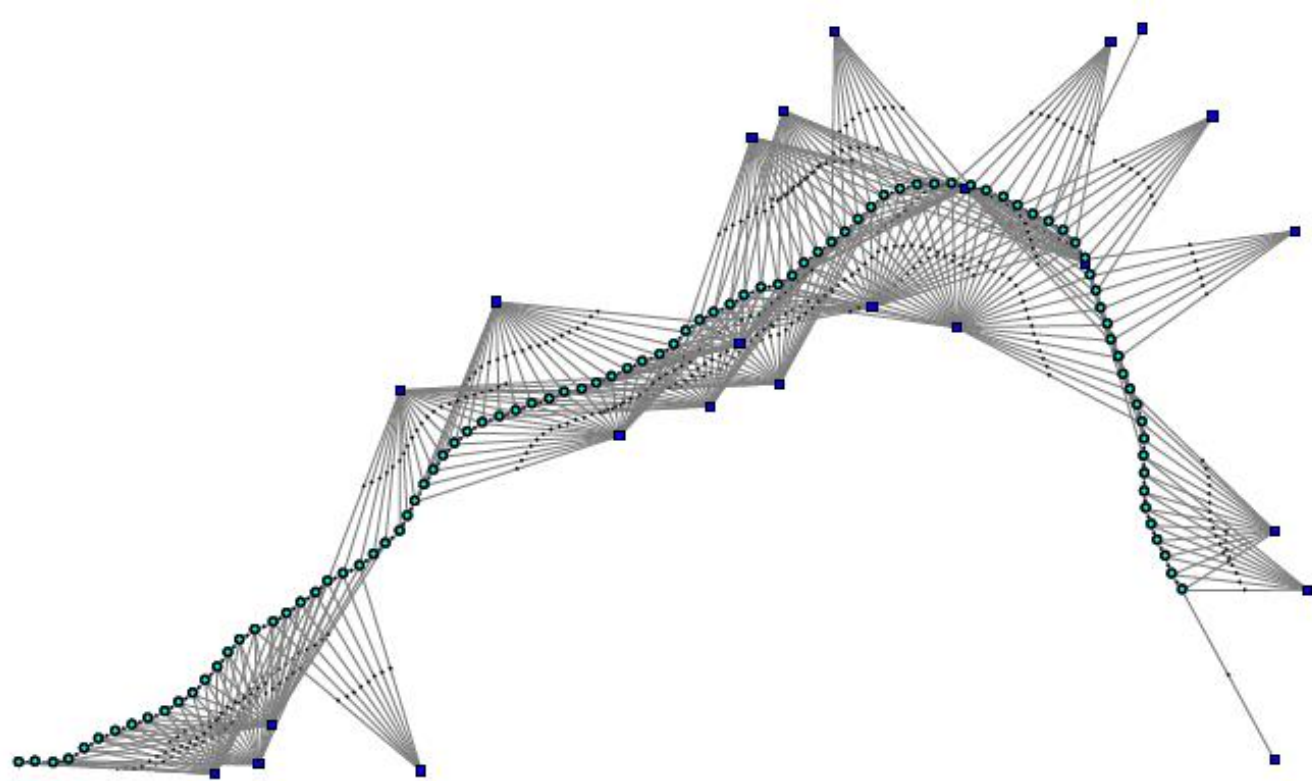
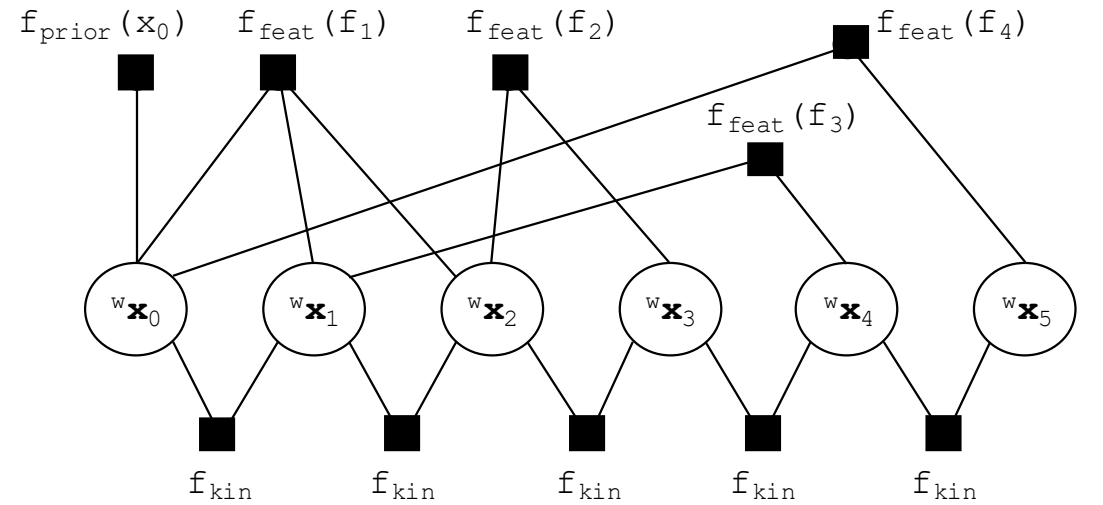
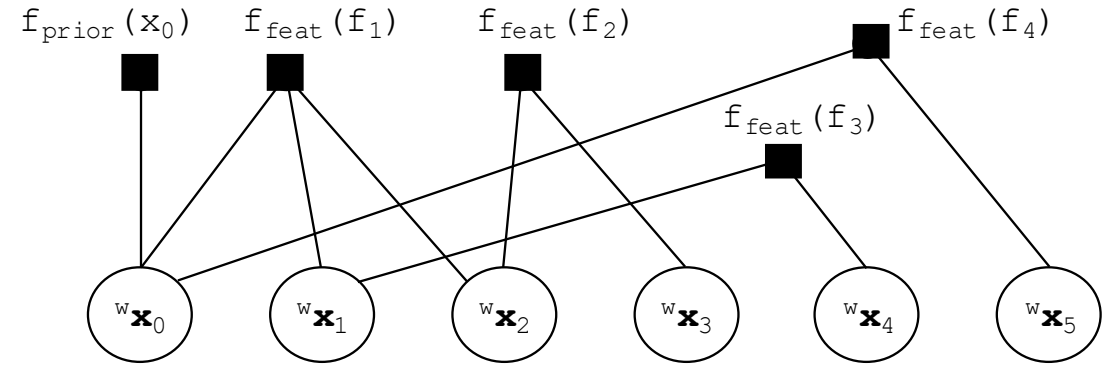
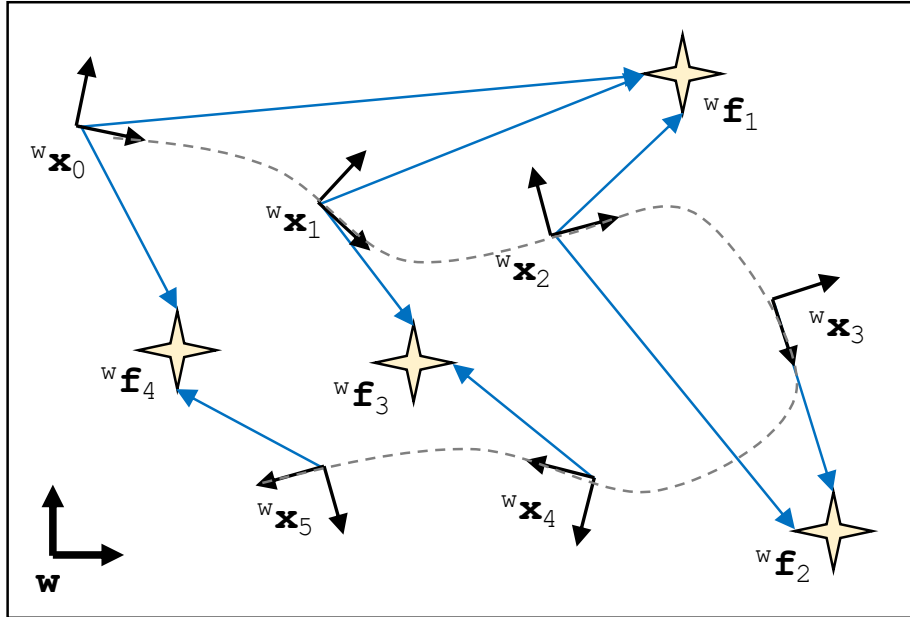
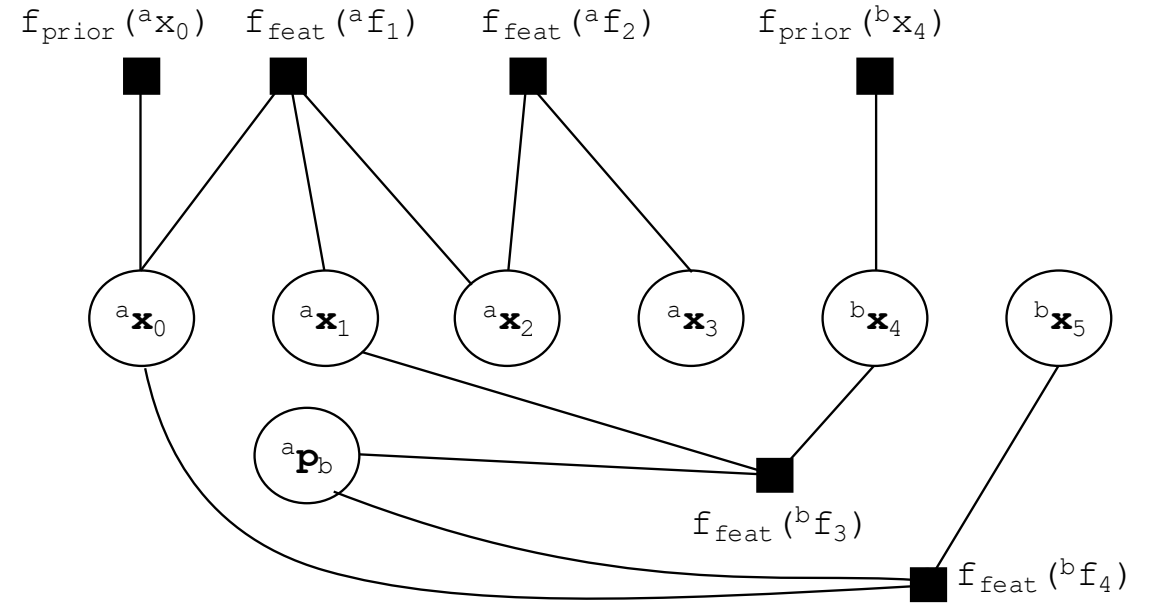
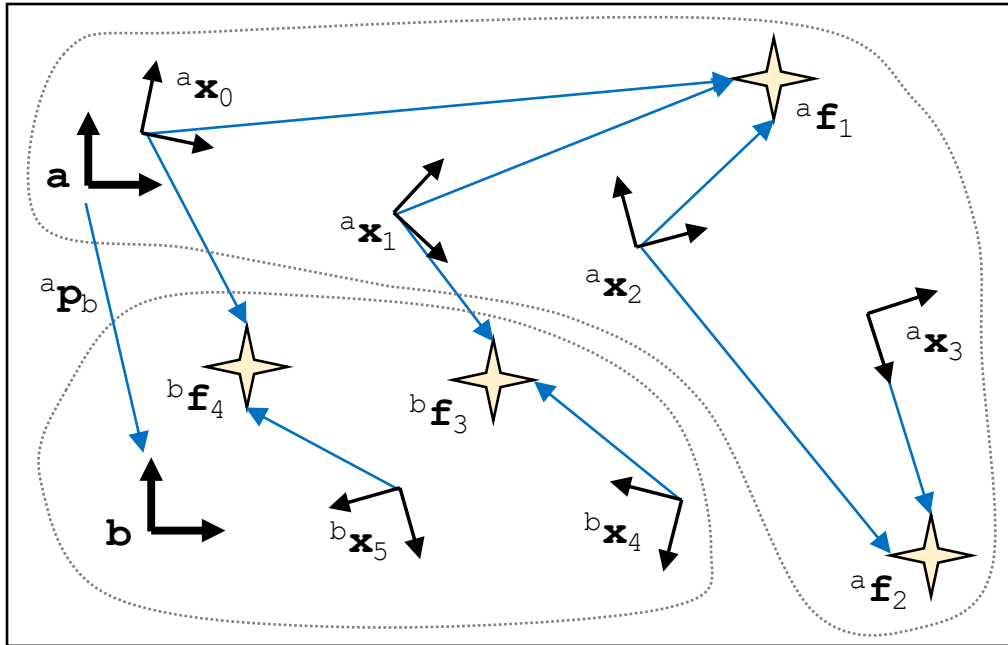


Figure 2.1: Factor graph for a larger, simulated SLAM example.

Applications to robotics & vision



Applications to robotics & vision

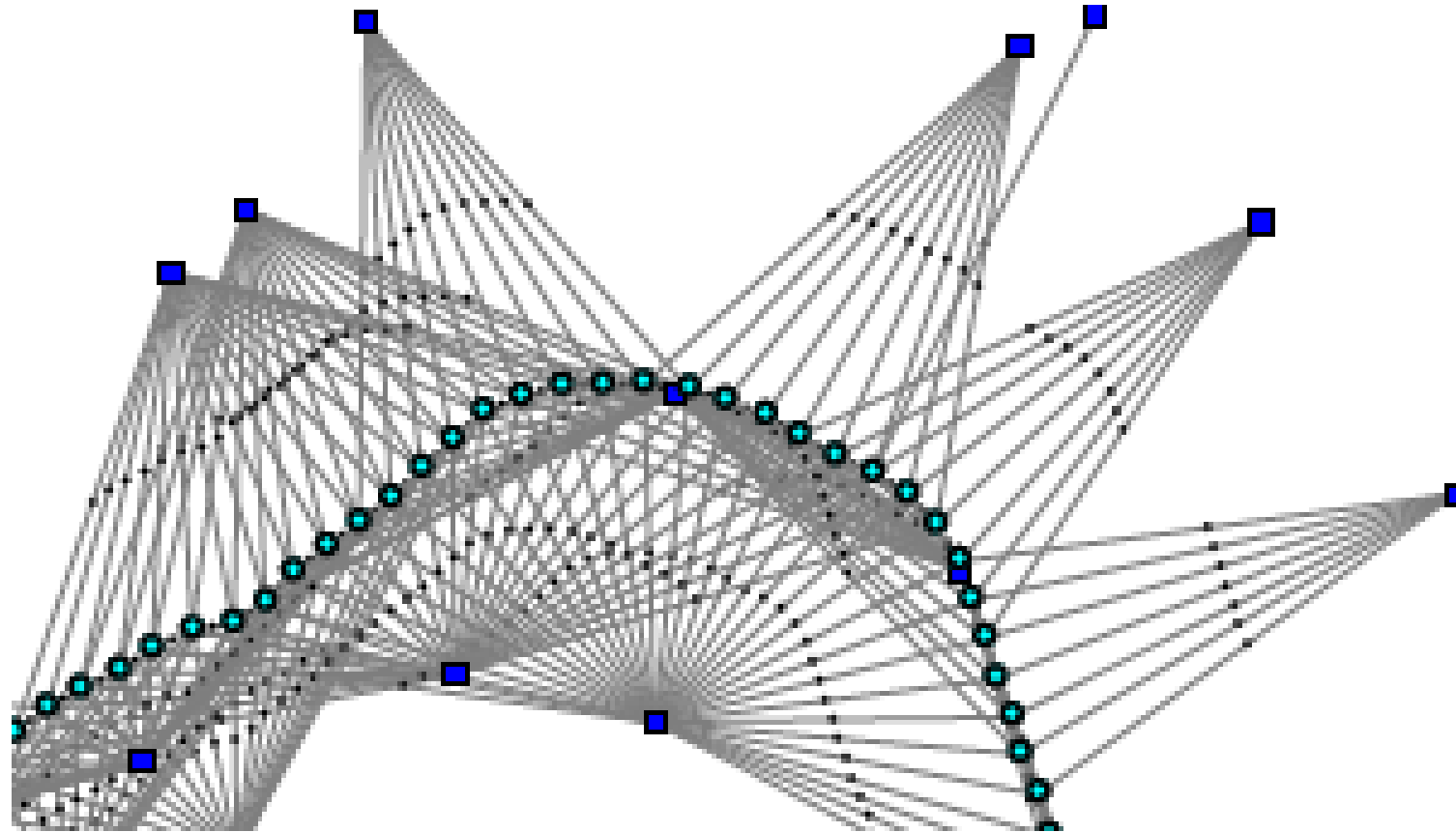


Applications to robotics & vision

C++ libraries for factor graphs (tailored to SLAM):

- g2o: A General Framework for Graph Optimization [Grisetti et al.]
<https://github.com/RainerKuemmerle/g2o>
- GTSAM: GeorgiaTech Smoothing and Mapping [Dellaert et al.]
<https://github.com/borglab/gtsam>

Bundle adjustment, poses-only, structure-only optimization





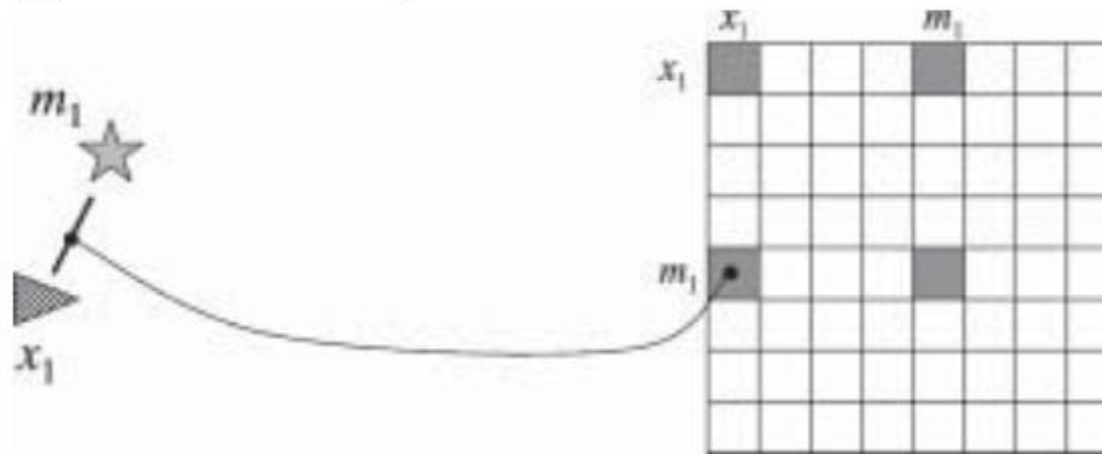
Applications to robotics & vision

- Poses-only: “factorizing out” some variables comes with the cost of a denser information matrix (in principle).
- The “mass-spring” model: “Removing a mass and all springs attached to it is equivalent to adding new springs between all affected masses”.

(Next slides from: Thrun, S., & Montemerlo, M. (2006). The graph SLAM algorithm with applications to large-scale mapping of urban structures. The International Journal of Robotics Research, 25(5-6), 403-429.)

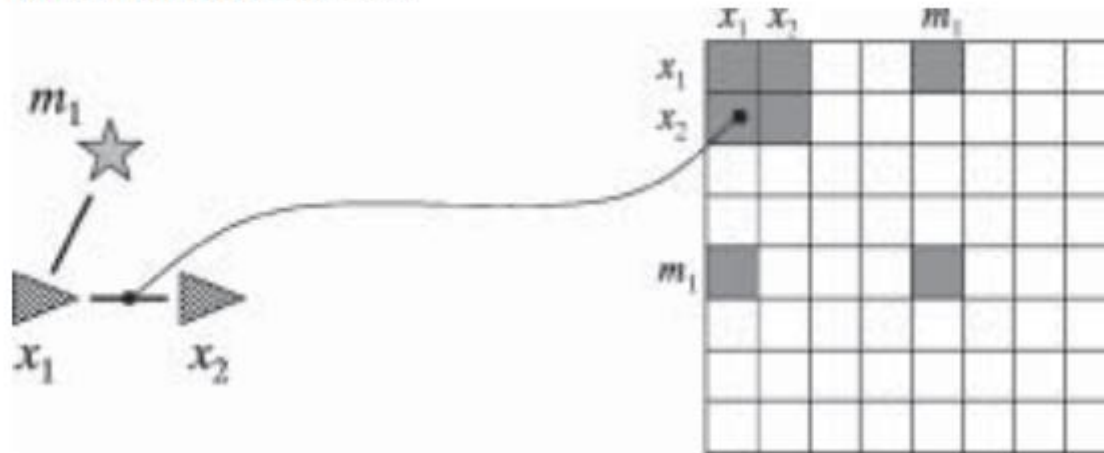
Applications to robotics & vision

(a) Observation is landmark m_1



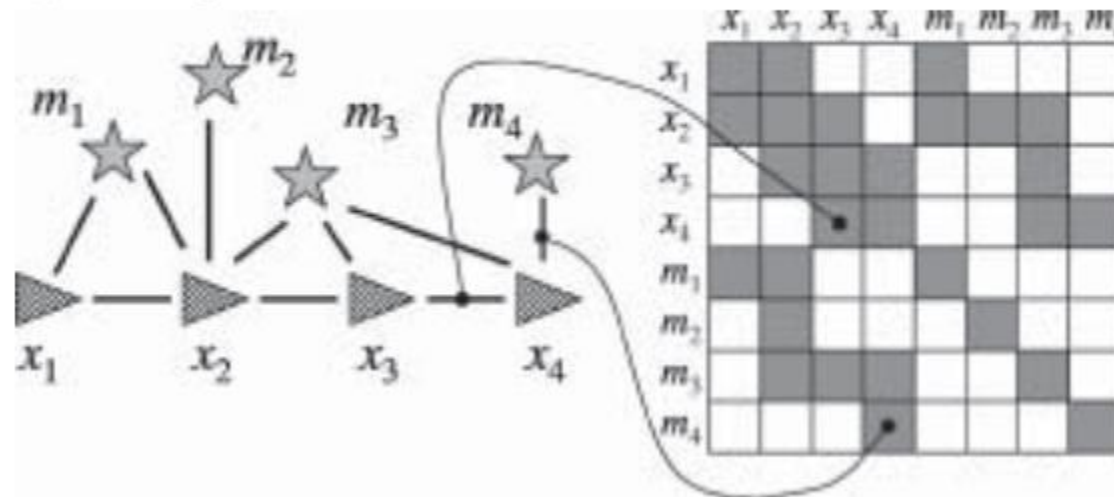
Applications to robotics & vision

(b) Robot motion from x_1 to x_2



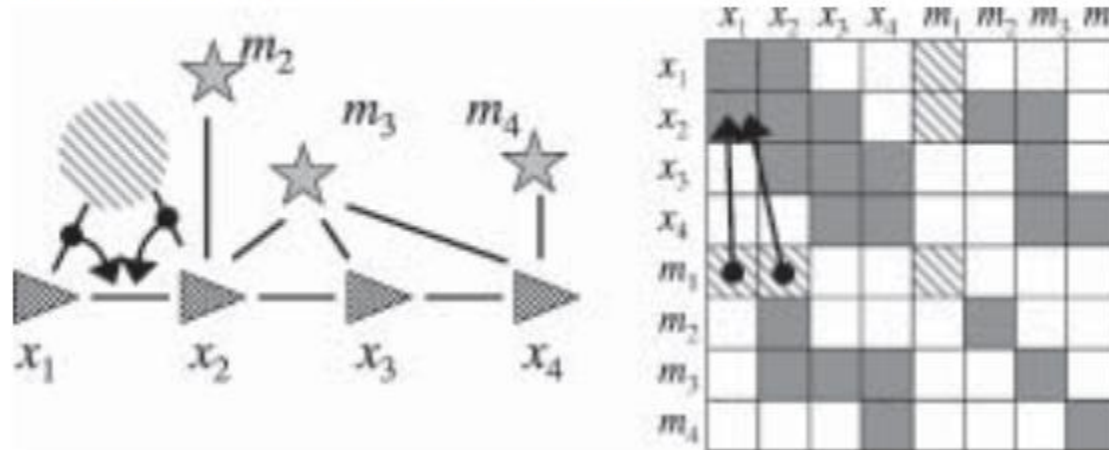
Applications to robotics & vision

(c) Several steps later



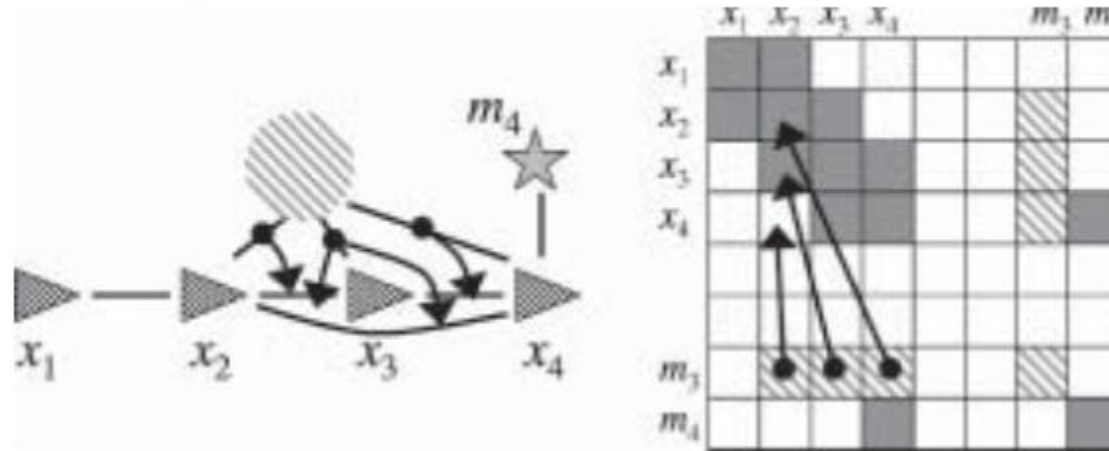
Applications to robotics & vision

(a) The removal of m_1 changes the link between x_1 and x_2

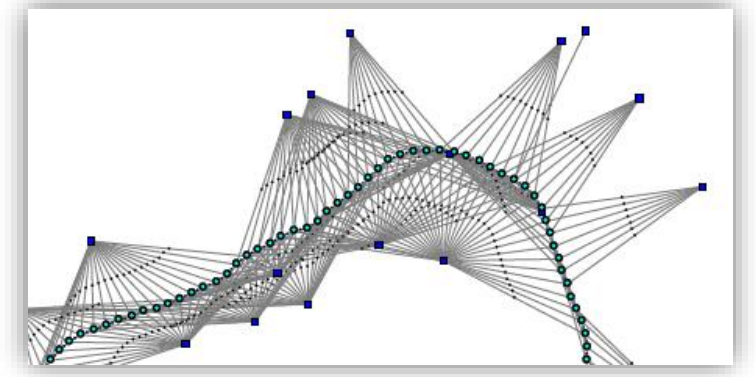


Applications to robotics & vision

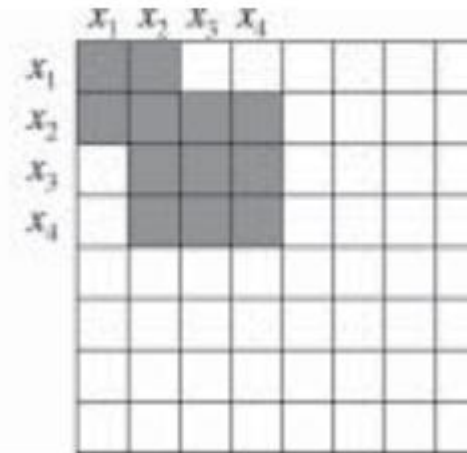
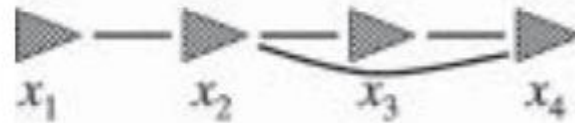
(b) The removal of m_3 introduces a new link between x_2 and x_4



Applications to robotics & vision

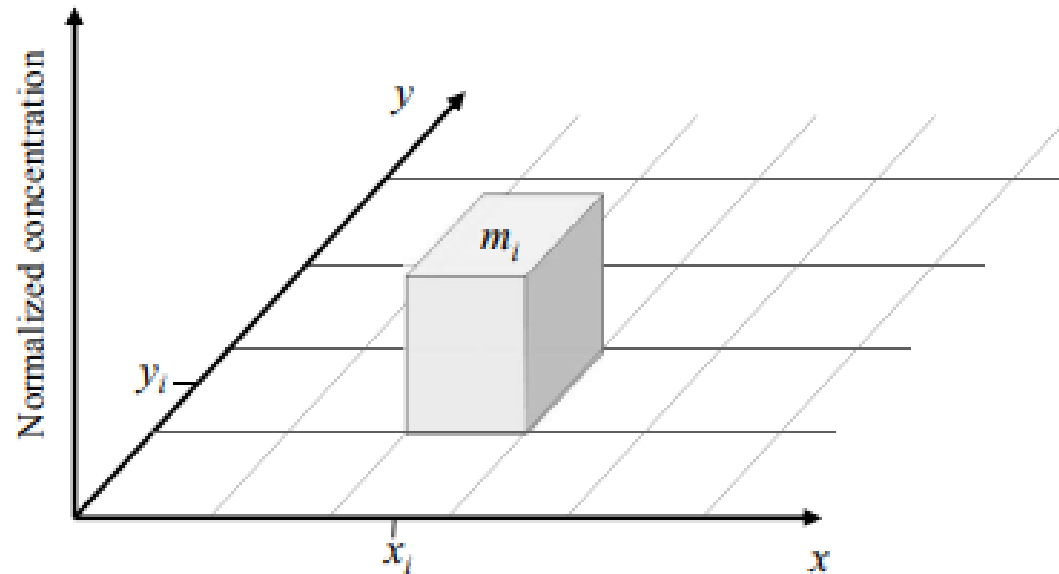


(c) Final result after removing all map features



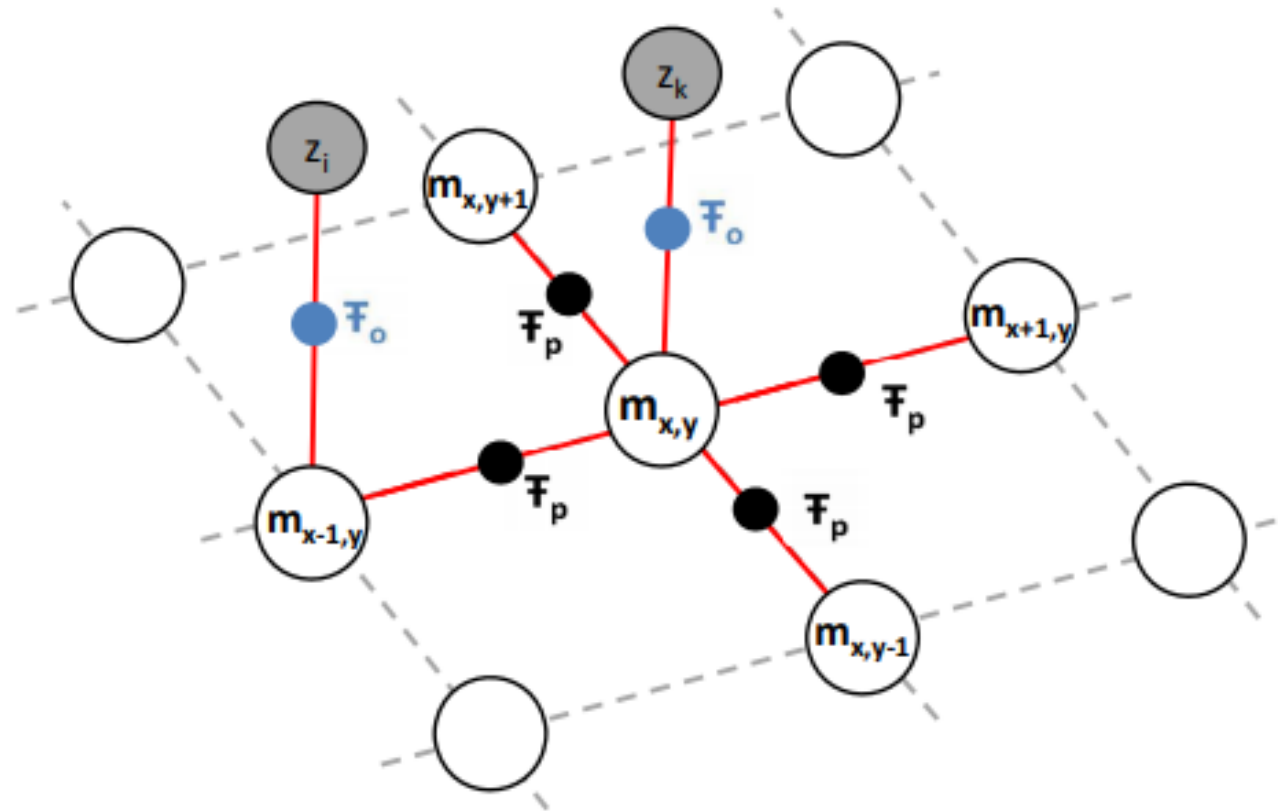
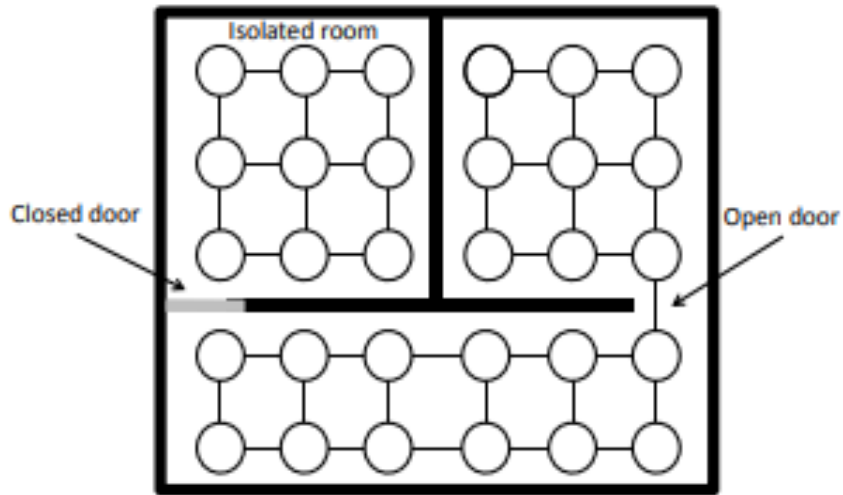
Time-varying gas mapping with a mobile robot

- Model the environment as a discrete set of cells with gas concentrations as the “unknown”. Include obstacles information.

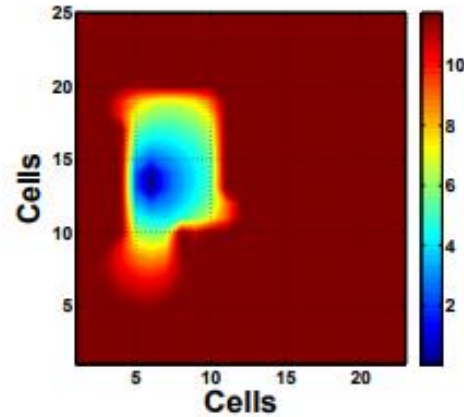


Monroy, J. G., Blanco, J. L., & Gonzalez-Jimenez, J. (2016). Time-variant gas distribution mapping with obstacle information. *Autonomous Robots*, 40(1), 1-16.

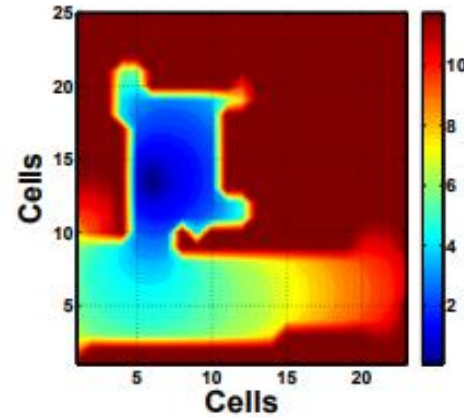
Time-varying gas mapping with a mobile robot



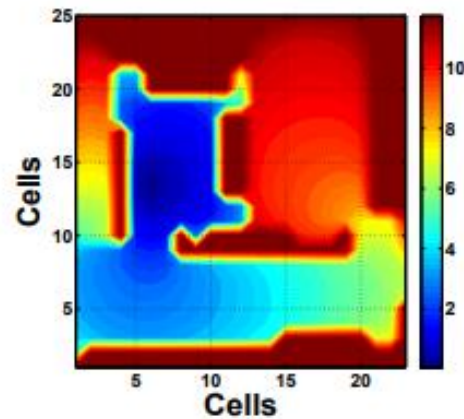
Time-varying gas mapping with a mobile robot



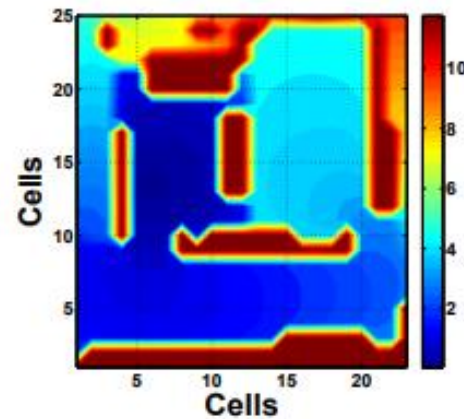
(a) $\sigma_p^2 = 5$



(b) $\sigma_p^2 = 2$



(c) $\sigma_p^2 = 1.25$

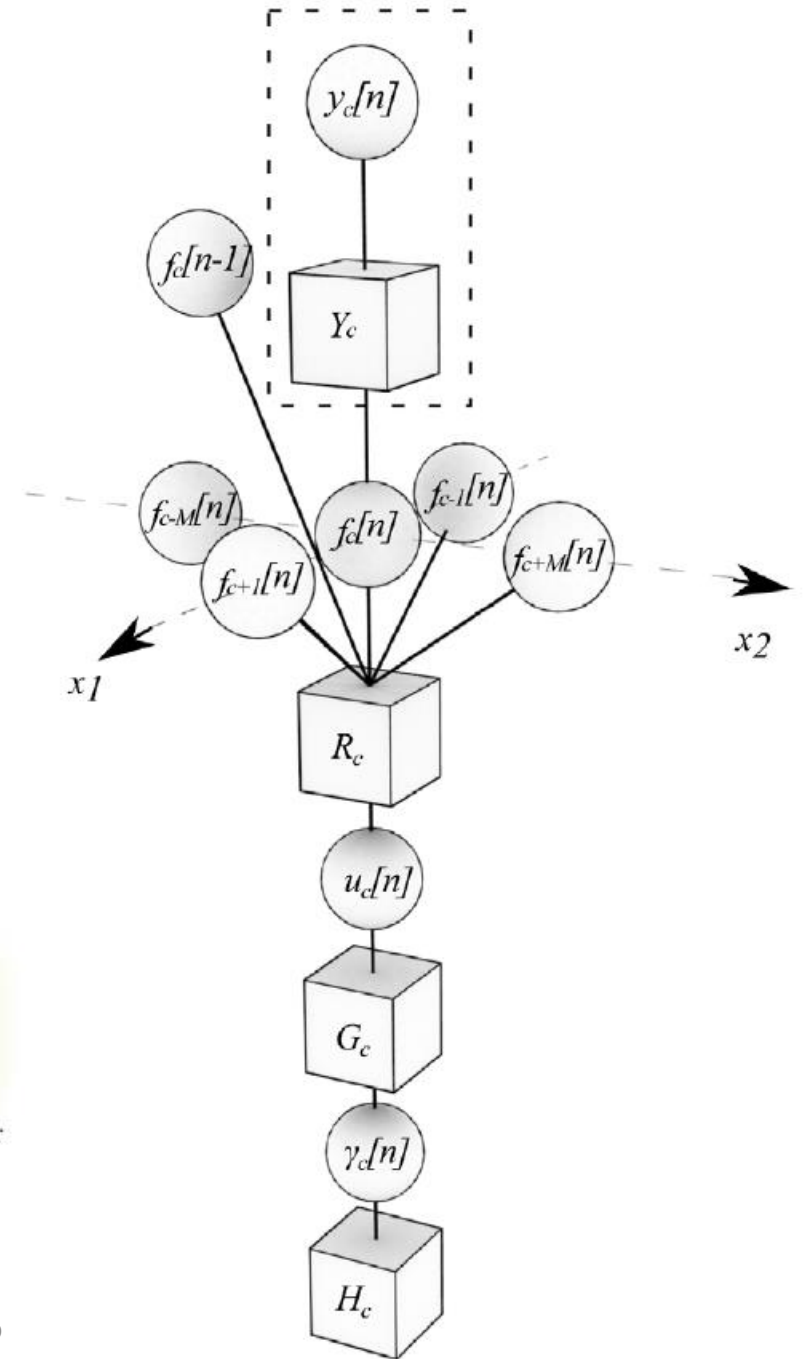
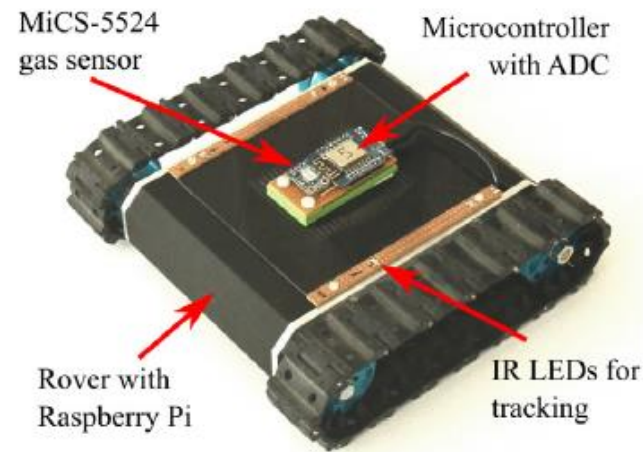


(d) $\sigma_p^2 = 0.5$

Gas mapping in dynamic environments

- Exploring a gas diffusion process using a multi-robot system. The physical behavior of the diffusion process is modeled using a Partial Differential Equation (PDE).

Wiedemann, T., Shutin, D., & Lilienthal, A. J. (2019). Model-based gas source localization strategy for a cooperative multi-robot system—A probabilistic approach and experimental validation incorporating physical knowledge and model uncertainties. *Robotics and Autonomous Systems*.



Manipulator dynamics with factor graphs

- Mechanical systems: kinematics can be also formulated as FGs.
- “This paper describes a **unified method solving for inverse, forward, and hybrid dynamics problems** for robotic manipulators with either open kinematic chains or closed kinematic loops based on factor graphs”.

Xie, M., & Dellaert, F. (2019). A Unified Method for Solving Inverse, Forward, and Hybrid Manipulator Dynamics using Factor Graphs. *arXiv preprint arXiv:1911.10065*.

A general framework for modeling and dynamic simulation of multibody systems using factor graphs

Nonlinear Dynamics - Springer Nature (2021)

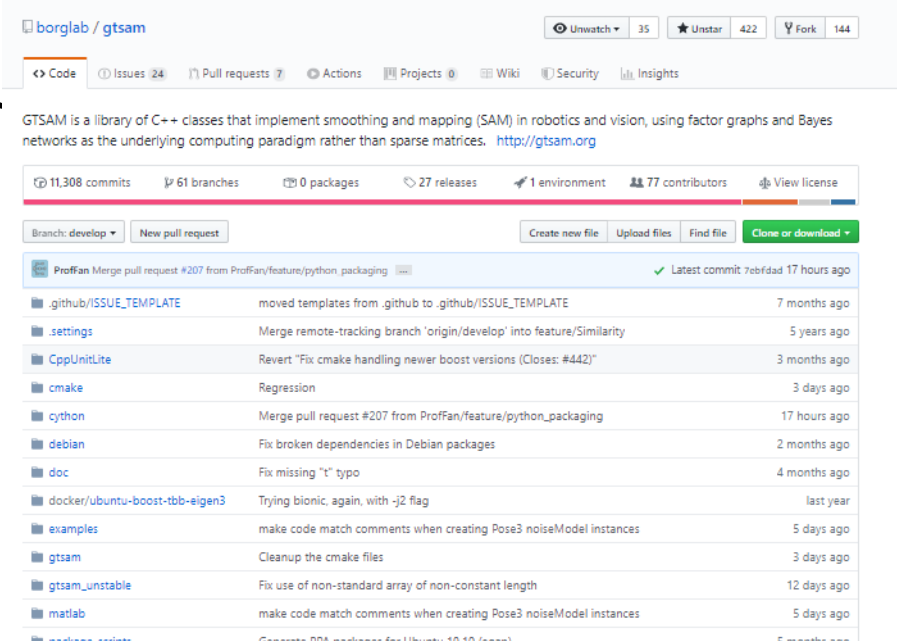
Jose-Luis Blanco-Claraco·Antonio Lanza·Giulio Reina

Introduction to GTSAM

GTSAM

- C++ library.
- Frank Dellaert, Georgia Tech “Institute for Robotics and Intelligent Machines”.
- Tailored to SLAM but coded as a general purpose library.
- Matlab and Python bindings.

<https://github.com/borglab/gtsam/>



borglab / gtsam

Unwatch 35 Unstar 422 Fork 144

Code Issues (24) Pull requests (7) Actions Projects (0) Wiki Security Insights

GTSAM is a library of C++ classes that implement smoothing and mapping (SAM) in robotics and vision, using factor graphs and Bayes networks as the underlying computing paradigm rather than sparse matrices. <http://gtsam.org>

11,308 commits 61 branches 0 packages 27 releases 1 environment 77 contributors View license

Branch: develop New pull request Create new file Upload files Find file Clone or download

ProfFan Merge pull request #207 from ProfFan/feature/python_packaging Latest commit 7ebfdad 17 hours ago

.github/ISSUE_TEMPLATE	moved templates from .github to .github/ISSUE_TEMPLATE	7 months ago
.settings	Merge remote-tracking branch 'origin/develop' into feature/Similarity	5 years ago
CppUnitLite	Revert "Fix cmake handling newer boost versions (Closes: #442)"	3 months ago
cmake	Regression	3 days ago
cython	Merge pull request #207 from ProfFan/feature/python_packaging	17 hours ago
debian	Fix broken dependencies in Debian packages	2 months ago
doc	Fix missing "t" typo	4 months ago
docker/ubuntu-boost-tbb-eigen3	Trying bionic, again, with -j2 flag	last year
examples	make code match comments when creating Pose3 noiseModel instances	5 days ago
gtsam	Cleanup the cmake files	3 days ago
gtsam_unstable	Fix use of non-standard array of non-constant length	12 days ago
matlab	make code match comments when creating Pose3 noiseModel instances	5 days ago
package_scripts	Generate PPA packages for Ubuntu 19.10 (eoan)	5 months ago

GTSAM

- Main C++ classes:
 - **NonlinearFactorGraph** : The Factor graph.
 - **FactorXXX**: Factors.
Arguments:
 - N Keys. Can be built with “Symbol” (e.g. “X1”, “V2”).
 - Observed value.
 - Noise model, e.g. “noiseModel::Diagonal::Sigmas()”
 - **Values** : “Variant” container for initial and final values of all keys.
Associative container: Key → Value.
 - **noiseModel**: Noise models, Gaussians, isotropic, M-estimation kernels,...

GTSAM: PlanarSLAMExample.cpp

```
// As this is a planar SLAM example, we will use Pose2 variables (x, y, theta) to represent
// the robot positions and Point2 variables (x, y) to represent the landmark coordinates.
#include <gtsam/geometry/Pose2.h>
#include <gtsam/geometry/Point2.h>

// Each variable in the system (poses and landmarks) must be identified with a unique key.
// We can either use simple integer keys (1, 2, 3, ...) or symbols (X1, X2, L1).
// Here we will use Symbols
#include <gtsam/inference/Symbol.h>

// In GTSAM, measurement functions are represented as 'factors'. Several common factors
// have been provided with the library for solving robotics/SLAM/Bundle Adjustment problems.
// Here we will use a RangeBearing factor for the range-bearing measurements to identified
// landmarks, and Between factors for the relative motion described by odometry measurements.
// Also, we will initialize the robot at the origin using a Prior factor.
#include <gtsam/slam/PriorFactor.h>
#include <gtsam/slam/BetweenFactor.h>
#include <gtsam/sam/BearingRangeFactor.h>

// When the factors are created, we will add them to a Factor Graph. As the factors we are using
// are nonlinear factors, we will need a Nonlinear Factor Graph.
#include <gtsam/nonlinear/NonlinearFactorGraph.h>
```

GTSAM: PlanarSLAMExample.cpp

```
// Finally, once all of the factors have been added to our factor graph, we will want to
// solve/optimize to graph to find the best (Maximum A Posteriori) set of variable values.
// GTSAM includes several nonlinear optimizers to perform this step. Here we will use the
// common Levenberg-Marquardt solver
#include <gtsam/nonlinear/LevenbergMarquardtOptimizer.h>

// Once the optimized values have been calculated, we can also calculate the marginal covariance
// of desired variables
#include <gtsam/nonlinear/Marginals.h>

// The nonlinear solvers within GTSAM are iterative solvers, meaning they linearize the
// nonlinear functions around an initial linearization point, then solve the linear system
// to update the linearization point. This happens repeatedly until the solver converges
// to a consistent set of variable values. This requires us to specify an initial guess
// for each variable, held in a Values container.
#include <gtsam/nonlinear/Values.h>

using namespace std;
using namespace gtsam;

int main(int argc, char** argv) {

    // Create a factor graph
    NonlinearFactorGraph graph;

    // Create the keys we need for this simple example
    static Symbol x1('x',1), x2('x',2), x3('x',3);
    static Symbol l1('l',1), l2('l',2);
```

GTSAM: PlanarSLAMExample.cpp

```
// Add a prior on pose x1 at the origin. A prior factor consists of a mean and a noise model (covariance)
Pose2 prior(0.0, 0.0, 0.0); // prior mean is at origin
noiseModel::Diagonal::shared_ptr priorNoise = noiseModel::Diagonal::Sigmas(Vector3(0.3, 0.3, 0.1)); // 3
graph.emplace_shared<PriorFactor<Pose2> >(x1, prior, priorNoise); // add directly to graph

// Add two odometry factors
Pose2 odometry(2.0, 0.0, 0.0); // create a measurement for both factors (the same in this case)
noiseModel::Diagonal::shared_ptr odometryNoise = noiseModel::Diagonal::Sigmas(Vector3(0.2, 0.2, 0.1)); /
graph.emplace_shared<BetweenFactor<Pose2> >(x1, x2, odometry, odometryNoise);
graph.emplace_shared<BetweenFactor<Pose2> >(x2, x3, odometry, odometryNoise);

// Add Range-Bearing measurements to two different landmarks
// create a noise model for the landmark measurements
noiseModel::Diagonal::shared_ptr measurementNoise = noiseModel::Diagonal::Sigmas(Vector2(0.1, 0.2)); //
// create the measurement values - indices are (pose id, landmark id)
Rot2 bearing11 = Rot2::fromDegrees(45),
    bearing21 = Rot2::fromDegrees(90),
    bearing32 = Rot2::fromDegrees(90);
double range11 = std::sqrt(4.0+4.0),
    range21 = 2.0,
    range32 = 2.0;

// Add Bearing-Range factors
graph.emplace_shared<BearingRangeFactor<Pose2, Point2> >(x1, 11, bearing11, range11, measurementNoise);
graph.emplace_shared<BearingRangeFactor<Pose2, Point2> >(x2, 11, bearing21, range21, measurementNoise);
graph.emplace_shared<BearingRangeFactor<Pose2, Point2> >(x3, 12, bearing32, range32, measurementNoise);

// Print
graph.print("Factor Graph:\n");
```

GTSAM: PlanarSLAMExample.cpp

```
// Create (deliberately inaccurate) initial estimate
Values initialEstimate;
initialEstimate.insert(x1, Pose2(0.5, 0.0, 0.2));
initialEstimate.insert(x2, Pose2(2.3, 0.1, -0.2));
initialEstimate.insert(x3, Pose2(4.1, 0.1, 0.1));
initialEstimate.insert(l1, Point2(1.8, 2.1));
initialEstimate.insert(l2, Point2(4.1, 1.8));

// Print
initialEstimate.print("Initial Estimate:\n");

// Optimize using Levenberg-Marquardt optimization. The optimizer
// accepts an optional set of configuration parameters, controlling
// things like convergence criteria, the type of linear system solver
// to use, and the amount of information displayed during optimization.
// Here we will use the default set of parameters. See the
// documentation for the full set of parameters.
LevenbergMarquardtOptimizer optimizer(graph, initialEstimate);
Values result = optimizer.optimize();
result.print("Final Result:\n");

// Calculate and print marginal covariances for all variables
Marginals marginals(graph, result);
print(marginals.marginalCovariance(x1), "x1 covariance");
print(marginals.marginalCovariance(x2), "x2 covariance");
print(marginals.marginalCovariance(x3), "x3 covariance");
print(marginals.marginalCovariance(l1), "l1 covariance");
print(marginals.marginalCovariance(l2), "l2 covariance");

return 0;
}
```

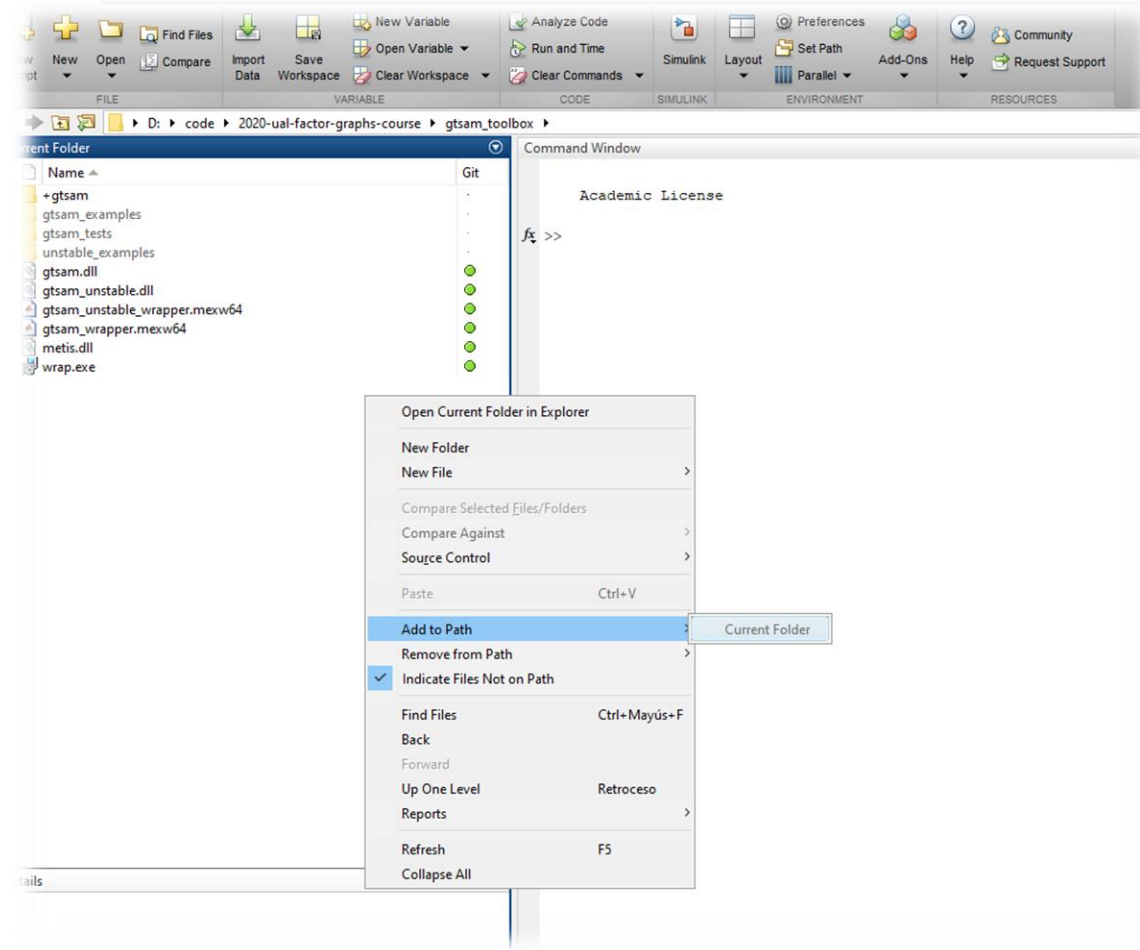
GTSAM: Matlab wrapper

GTSAM MATLAB: Installation

- From sources:
 - Enable building the MATLAB wrapper in CMake.
 - Set an installation directory.
 - Build the INSTALL target.
- Precompiled version for Windows and MATLAB 64bits:
 - <https://github.com/jlblancoc/factor-graphs-course>
 - Clone or download as ZIP and uncompress.

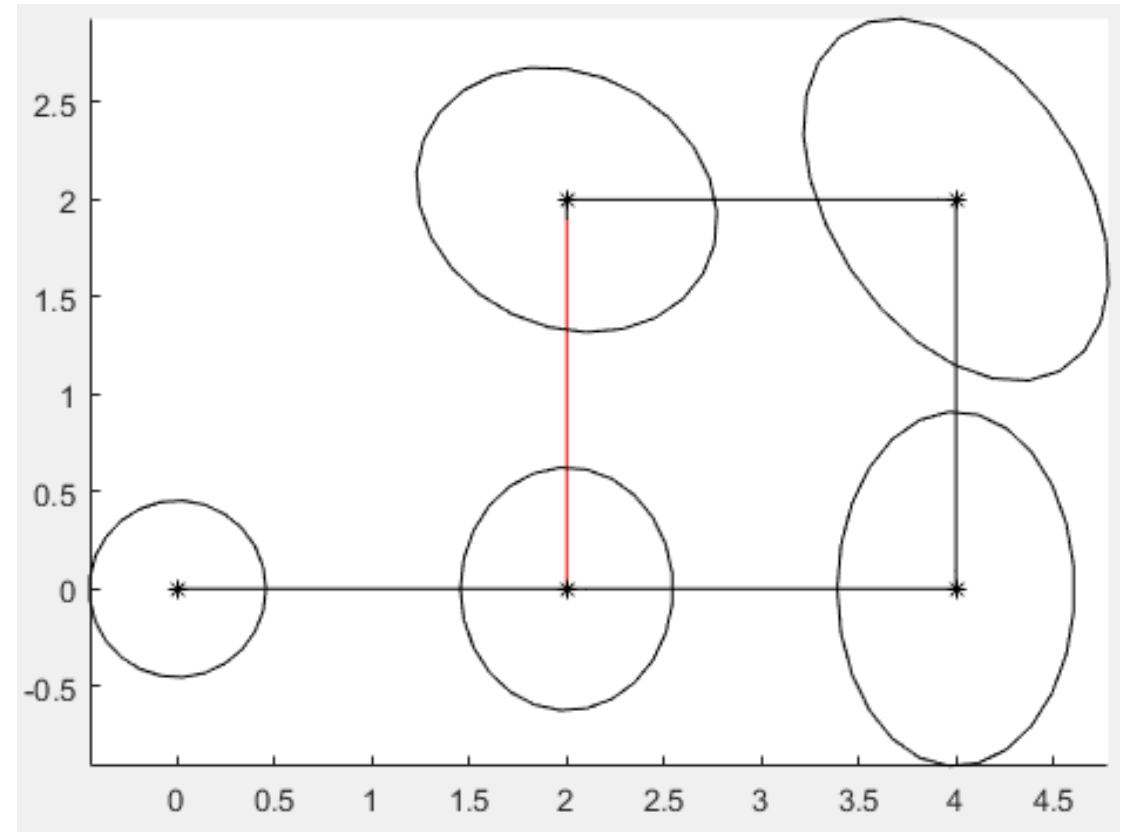
GTSAM MATLAB: Installation

- From MATLAB:
 - “Add to path” → gtsam_toolbox
- (If built from sources manually, also add the “bin” directory to the system PATH).



GTSAM MATLAB: Exercise 1

- Run Pose2SLAMExample.m
- Analyze the code and the results.



GTSAM MATLAB: Exercise 2

- Modify the file `Pose2SLAMExample.m` to:
 - Remove the observation between Keyframe 5 & 2. Observe the final covariances.
 - Restore the removed edge, and introduce small errors in the relative poses.
 - After that, define a different covariance and create a new edge between 1 & 5. Experiment with different information matrices and edge observed values.