Georgia Tech Institute for Robotics and Intelligent Machines

GTSAM *4.0* Tutorial
Theory, Programming, and Applications

GTSAM: https://bitbucket.org/gtborg/gtsam
Examples: https://github.com/dongjing3309/gtsam-examples

Jing Dong
2016-11-19

License CC BY-NC-SA 3.0

Outline

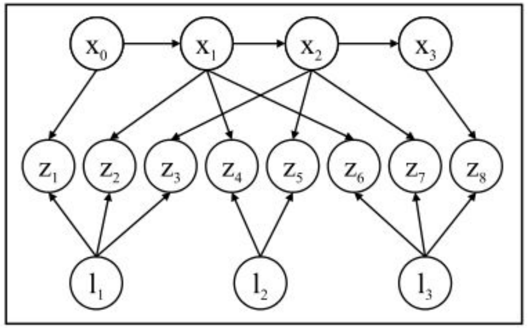
Georgia Tech Institute for Robotics and Intelligent Machines

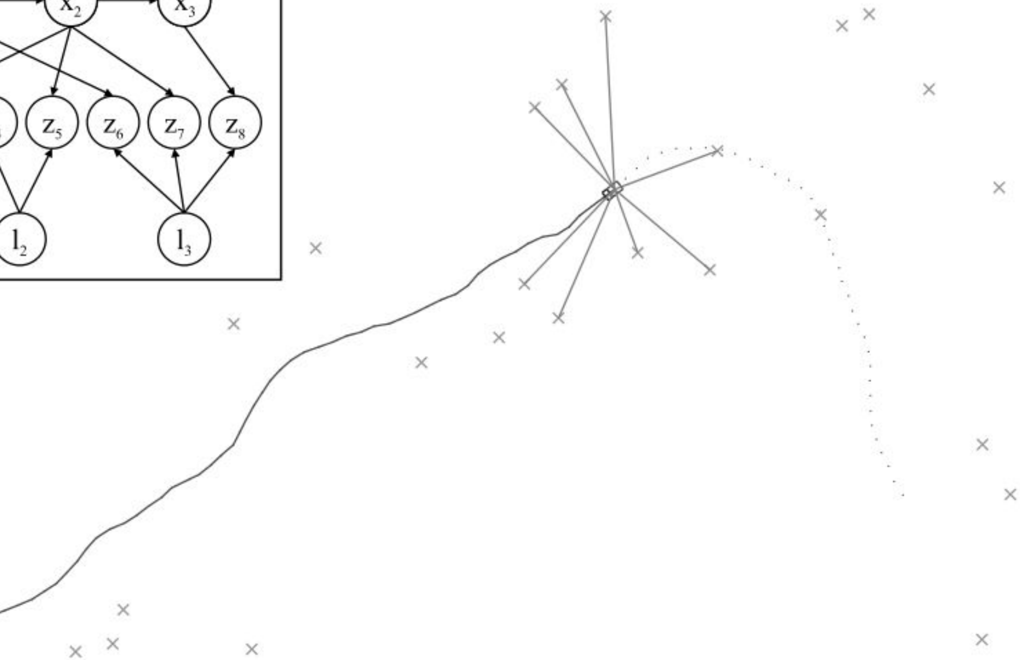
- Theory
  - SLAM as a Factor Graph
  - SLAM as a Non-linear Least Squares
  - Optimization on Manifold/Lie Groups
  - iSAM2 and Bayes Tree
- Programming
  - First C++ example
  - Use GTSAM in Matlab
  - Write your own factor
  - Expression: Automatic Differentiation (AD) (New in 4.0!)
  - Traits: Optimize any type in GTSAM (New in 4.0!)
  - Use GTSAM in Python (New in 4.0!)
- Applications
  - Visual-Inertial Odometry
  - Structure from Motion (SfM)
  - Multi-Robot SLAM: Coordinate Frame and Distrubuted Optimization
  - Multi-View Stereo and Optical Flow
  - Motion Planning

Outline

- Theory
  - SLAM as a Factor Graph
  - SLAM as a Non-linear Least Squares
  - Optimization on Manifold/Lie Groups
  - iSAM2 and Bayes Tree
- Programming
  - First C++ example
  - Use GTSAM in Matlab
  - Write your own factor
  - Expression: Automatic Differentiation (AD) (New in 4.0!)
  - Traits: Optimize any type in GTSAM (New in 4.0!)
  - Use GTSAM in Python (New in 4.0!)
- Applications
  - Visual-Inertial Odometry
  - Structure from Motion (SfM)
  - Multi-Robot SLAM: Coordinate Frame and Distrubuted Optimization
  - Multi-View Stereo and Optical Flow
  - Motion Planning

SLAM as a Bayes Net





$$P(X, L, Z) = P(x_0) \prod_{i=1}^{M} P(x_i | x_{i-1}, u_i) \prod_{k=1}^{K} P(z_k | x_{i_k}, l_{j_k})$$

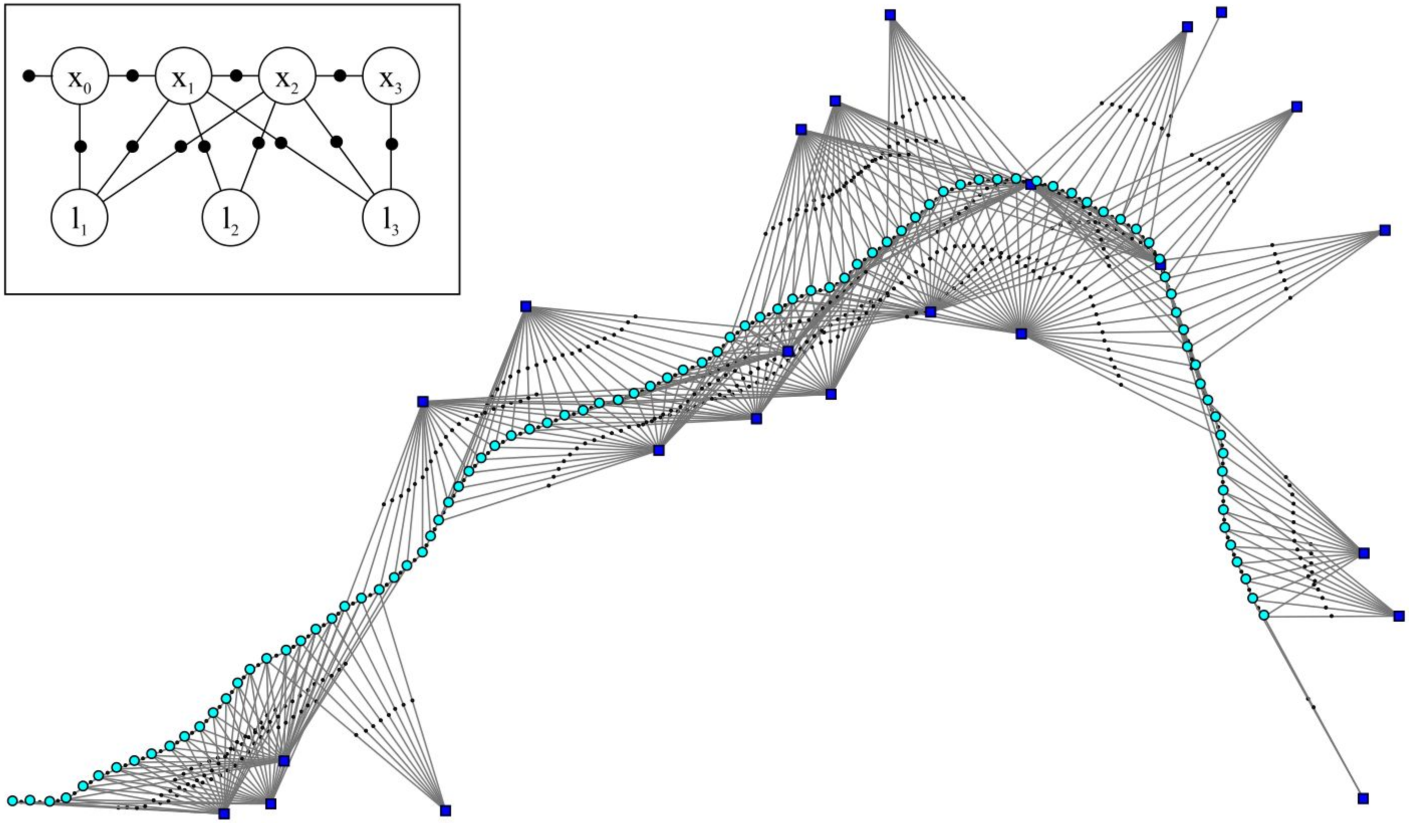
$$x_i = f_i(x_{i-1}, u_i) + w_i \quad \Leftrightarrow$$

$$P(x_i | x_{i-1}, u_i) \propto \exp -\frac{1}{2} \| f_i(x_{i-1}, u_i) - x_i \|^2_{\Lambda_i}$$

$$z_k = h_k(x_{i_k}, l_{j_k}) + v_k \quad \Leftrightarrow$$

$$P(z_k | x_{i_k}, l_{j_k}) \propto \exp -\frac{1}{2} \| h_k(x_{i_k}, l_{j_k}) - z_k \|^2_{\Sigma_k}$$

SLAM as a Factor Graph



$$\phi_0(x_0) \propto P(x_0)$$

$$P(\Theta) \propto \prod_i \phi_i(\theta_i) \prod_{\{i,j\}, i<j} \psi_{ij}(\theta_i, \theta_j)$$

$$\psi_{(i-1)i}(x_{i-1}, x_i) \propto P(x_i | x_{i-1}, u_i)$$

$$\Theta \triangleq (X, L)$$

$$\psi_{i_k j_k}(x_{i_k}, l_{j_k}) \propto P(z_k | x_{i_k}, l_{j_k})$$

SLAM as a Non-linear Least Squares

- Maximum a posteriori (MAP) estimation

$$f(\Theta) = \prod_i f_i(\Theta_i) \qquad \Theta \triangleq (X, L) \quad \text{for each} \quad f_i(\Theta_i) \propto \exp\left(-\frac{1}{2}\|h_i(\Theta_i) - z_i\|^2_{\Sigma_i}\right)$$

$$\Theta^* = \arg\max_{\Theta} f(\Theta)$$

- Log likelihood

$$\arg\min_{\Theta}(-\log f(\Theta)) = \arg\min_{\Theta} \frac{1}{2}\sum_i \|h_i(\Theta_i) - z_i\|^2_{\Sigma_i}$$

Non-linear Least Squares

- Gauss-Newton method:

$$\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}}\{F(\mathbf{x})\} \ ,$$

where

$$F(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{m}(f_i(\mathbf{x}))^2 = \frac{1}{2}\|\mathbf{f}(\mathbf{x})\|^2 = \frac{1}{2}\mathbf{f}(\mathbf{x})^\top\mathbf{f}(\mathbf{x})$$

- Linear approximation of the vector function (get Jacobians)

$$\begin{aligned}\mathbf{f}(\mathbf{x}+\mathbf{h}) \ &= \ \mathbf{f}(\mathbf{x}) + \mathbf{J}(\mathbf{x})\mathbf{h} + O(\|\mathbf{h}\|^2) \\ \mathbf{f}(\mathbf{x}+\mathbf{h}) \ &\simeq \ \boldsymbol{\ell}(\mathbf{h}) \ \equiv \ \mathbf{f}(\mathbf{x}) + \mathbf{J}(\mathbf{x})\mathbf{h}\end{aligned}$$

with $(\mathbf{J}(\mathbf{x}))_{ij} = \dfrac{\partial f_i}{\partial x_j}(\mathbf{x})$

- Quadratic approximation of the cost error function (get Hessian)

$$\begin{aligned}F(\mathbf{x}+\mathbf{h}) \simeq L(\mathbf{h}) &\equiv \tfrac{1}{2}\boldsymbol{\ell}(\mathbf{h})^\top\boldsymbol{\ell}(\mathbf{h}) \\ &= \tfrac{1}{2}\mathbf{f}^\top\mathbf{f} + \mathbf{h}^\top\mathbf{J}^\top\mathbf{f} + \tfrac{1}{2}\mathbf{h}^\top\mathbf{J}^\top\mathbf{J}\mathbf{h} \\ &= F(\mathbf{x}) + \mathbf{h}^\top\mathbf{J}^\top\mathbf{f} + \tfrac{1}{2}\mathbf{h}^\top\mathbf{J}^\top\mathbf{J}\mathbf{h}\end{aligned}$$

$$(\mathbf{J}^\top\mathbf{J})\mathbf{h}_{\mathrm{gn}} \ = \ -\mathbf{J}^\top\mathbf{f} \ .$$

Linear Least Squares

- Gauss-Newton method: Given a set of initial values, linearize the non-linear problem **around current values**, and solve linear least square problems iteratively.

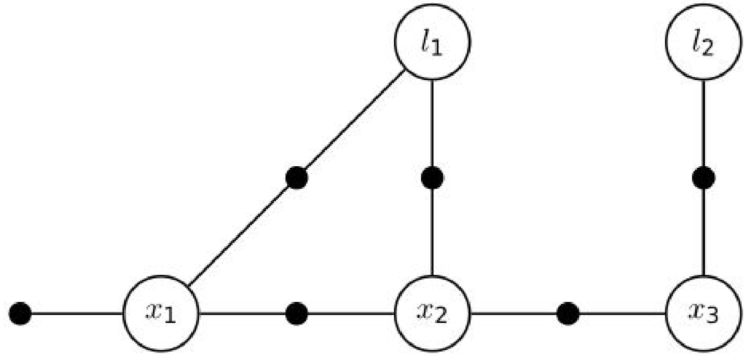
$$\arg\min_{\Theta}\left(-\log f(\Theta)\right) = \arg\min_{\Theta} \frac{1}{2}\sum_i \|h_i(\Theta_i) - z_i\|^2_{\Sigma_i}$$

$$\arg\min_{\Delta}\left(-\log f(\Delta)\right) = \arg\min_{\Delta} \|A\Delta - \mathbf{b}\|^2$$

Given $\|\Delta\|^2_{\Sigma} = \Delta^T\Sigma^{-1}\Delta = \Delta^T\Sigma^{-\frac{T}{2}}\Sigma^{-\frac{1}{2}}\Delta = \left\|\Sigma^{-\frac{1}{2}}\Delta\right\|^2$

- Other method like Levenberg–Marquardt or Trust Region methods are also fine, since they are just using different updating strategy.

Example



$$A = \begin{array}{c} \begin{array}{ccccc} l_1 & l_2 & x_1 & x_2 & x_3 \end{array} \\ \begin{bmatrix} X & & X & & \\ X & & & X & \\ & X & & & X \\ & & X & & \\ & & X & X & \\ & & & X & X \end{bmatrix} \end{array}$$

Linear Least Squares

$$\delta^* = \underset{\delta}{\operatorname{argmin}} \ \|A\delta - b\|_2^2$$

- QR decomposition

$$Q^T A = \begin{bmatrix} R \\ 0 \end{bmatrix} \quad Q^T b = \begin{bmatrix} d \\ e \end{bmatrix}$$

$$R\delta = d$$

- Cholesky decomposition

$$A^T A \delta^* = A^T b$$

$$\mathcal{I} \triangleq A^T A = R^T R$$

$$\text{first } R^T y = A^T b \text{ and then } R\delta^* = y$$

Full SAM approach

Alg. 1 General structure of the smoothing solution to SLAM with a direct equation solver (Cholesky, QR). Steps 3-6 can optionally be iterated and/or modified to implement the Levenberg-Marquardt algorithm.

Repeat for new measurements in each step:

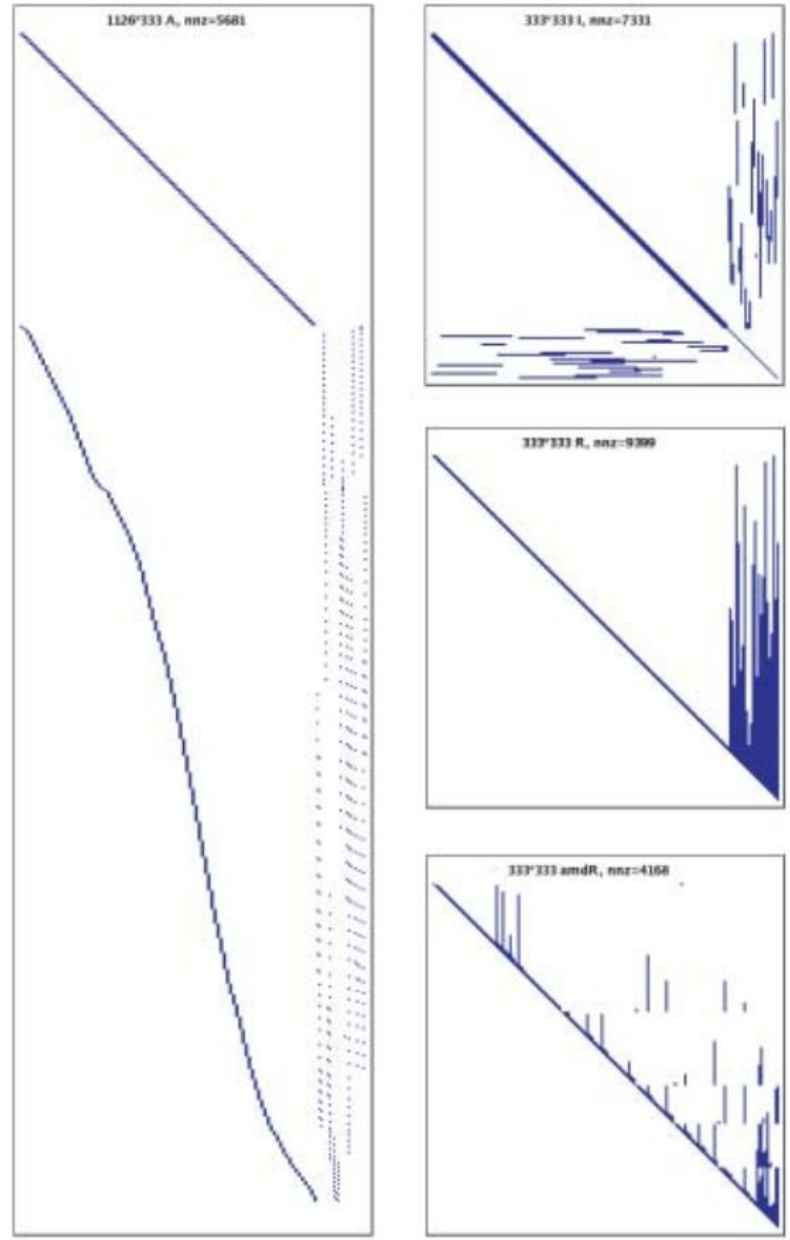
1. Add new measurements.
2. Add and initialize any new variables.
3. Linearize at current estimate Θ.
4. Factorize with QR or Cholesky.
5. Solve by backsubstitution to obtain Δ.
6. Obtain new estimate $\Theta' = \Theta \bigoplus \Delta$.

Dellaert, Frank, and Michael Kaess. "Square Root SAM: Simultaneous localization and mapping via square root information smoothing." The International Journal of Robotics Research 25.12 (2006): 1181-1203.

Ordering

- Select the correct column ordering does matter since it decide the sparsity of information matrix

- Use COLAMD to find the best ordering just based on information matrix



Optimization on Manifold/Lie Groups

- Lie group:

Lie groups are not as easy to treat as the vector space $\mathbb{R}^n$ but nevertheless have a lot of structure. To generalize the concept of the total derivative above we just need to replace $a \oplus \xi$ in (1.3) with a suitable operation in the Lie group G. In particular, the notion of an exponential map allows us to define a mapping from **local coordinates** ξ back to a neighborhood in G around a,

$$a \oplus \xi \stackrel{\Delta}{=} a \exp\left(\hat{\xi}\right) \tag{3.1}$$

with $\xi \in \mathbb{R}^n$ for an n-dimensional Lie group. Above, $\hat{\xi} \in \mathfrak{g}$ is the Lie algebra element corresponding to the vector ξ, and $\exp \hat{\xi}$ the exponential map. Note that if G is equal to $\mathbb{R}^n$ then composing with the exponential map $ae^{\hat{\xi}}$ is just vector addition $a + \xi$.

Dellaert, Frank. "Derivatives and Differentials" in GTSAM repository /doc/math.pdf

Optimization on Manifold/Lie Groups

- General manifold (if not Lie group):

General manifolds that are not Lie groups do not have an exponential map, but can still be handled by defining a **retraction** $\mathscr{R} : \mathscr{M} \times \mathbb{R}^n \to \mathscr{M}$, such that

$$a \oplus \xi \stackrel{\Delta}{=} \mathscr{R}_a(\xi)$$

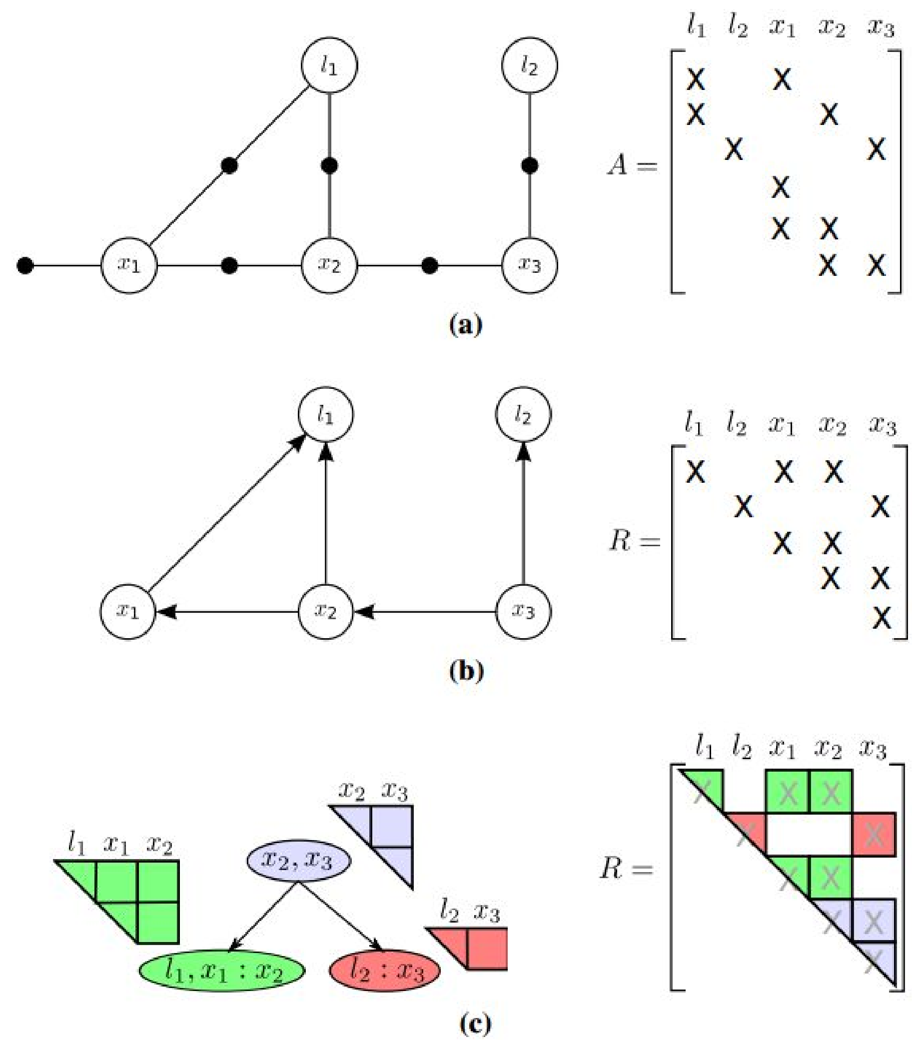
A retraction [?] is required to be tangent to geodesics on the manifold $\mathscr{M}$ at a. We can define many retractions for a manifold $\mathscr{M}$, even for those with more structure. For the vector space $\mathbb{R}^n$ the retraction is just vector addition, and for Lie groups the obvious retraction is simply the exponential map, i.e., $\mathscr{R}_a(\xi) = a \cdot \exp \hat{\xi}$. However, one can choose other, possibly computationally attractive retractions, as long as around a they agree with the geodesic induced by the exponential map, i.e.,

$$\lim_{\xi \to 0} \frac{\left| a \cdot \exp \hat{\xi} - \mathscr{R}_a(\xi) \right|}{|\xi|} = 0$$

Dellaert, Frank. "Derivatives and Differentials" in GTSAM repository /doc/math.pdf

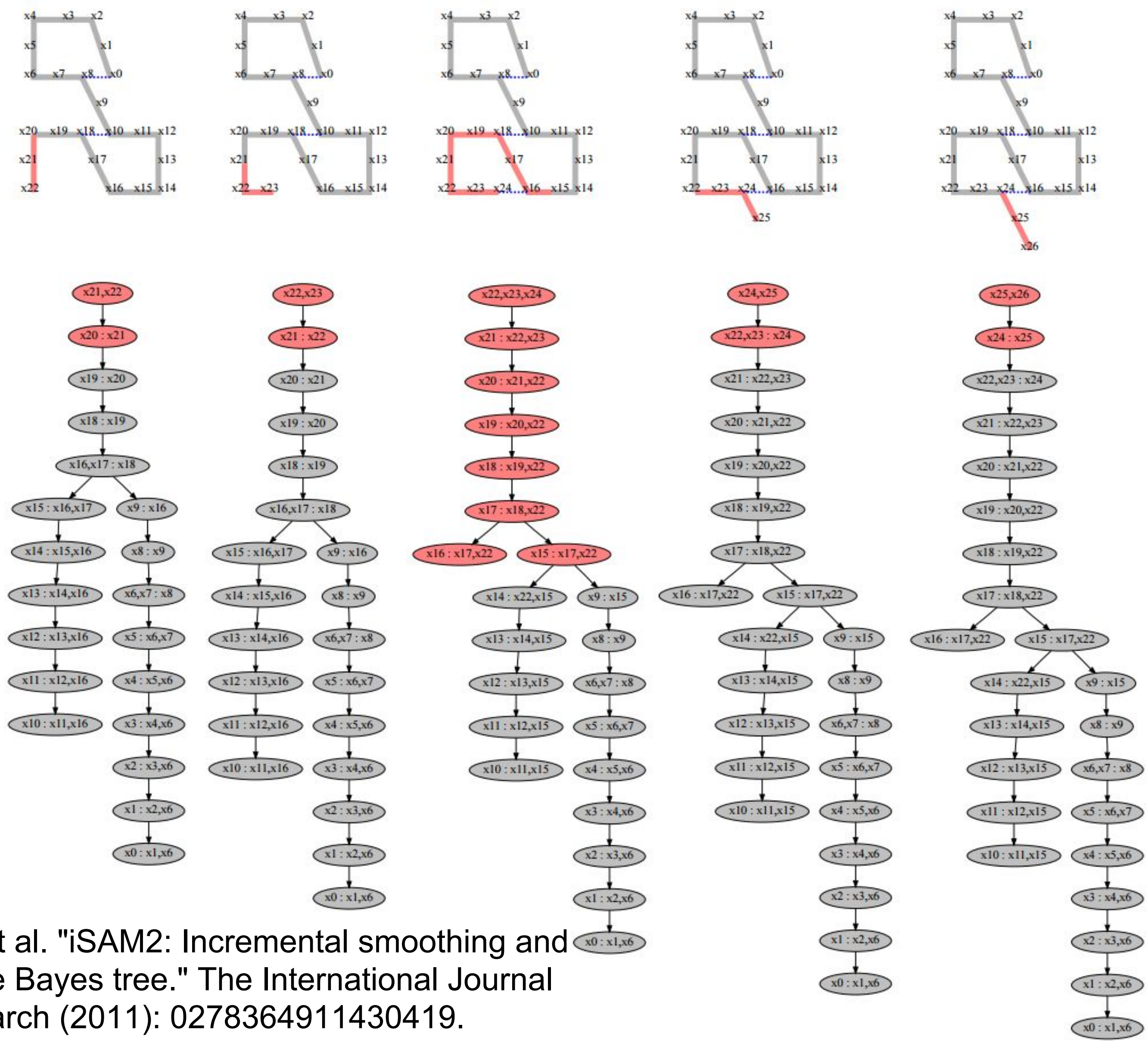
iSAM2 and Bayes tree

- iSAM2 is used to perform incremental inference (optimization) problems: when small part of the problem is changed and major part remain unchanged.
- Use Bayes tree as back-end data strcuture



Kaess, Michael, et al. "iSAM2: Incremental smoothing and mapping using the Bayes tree." The International Journal of Robotics Research (2011): 0278364911430419.

iSAM2 and Bayes tree



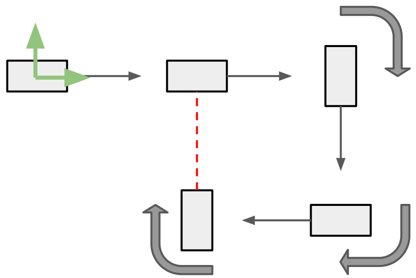
Kaess, Michael, et al. "iSAM2: Incremental smoothing and mapping using the Bayes tree." The International Journal of Robotics Research (2011): 0278364911430419.

Outline

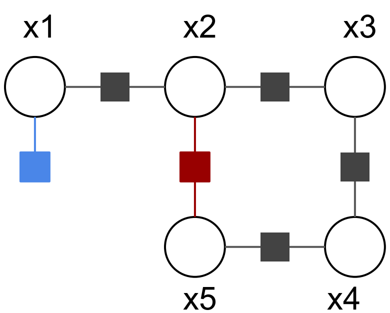
- Theory
  - SLAM as a Factor Graph
  - SLAM as a Non-linear Least Squares
  - Optimization on Manifold/Lie Groups
  - iSAM2 and Bayes Tree
- Programming
  - First C++ example
  - Use GTSAM in Matlab
  - Write your own factor
  - Expression: Automatic Differentiation (AD) (New in 4.0!)
  - Traits: Optimize any type in GTSAM (New in 4.0!)
  - Use GTSAM in Python (New in 4.0!)
- Applications
  - Visual-Inertial Odometry
  - Structure from Motion (SfM)
  - Multi-Robot SLAM: Coordinate Frame and Distrubuted Optimization
  - Multi-View Stereo and Optical Flow
  - Motion Planning

First C++ Example

Georgia Tech Institute for Robotics and Intelligent Machines



1. Build factor graph
2. Give initial values (this is a little bit tricky and highly application-related, design your strategy based on your application!)
3. Optimize!
4. (Optional) Post process, like calculate marginal distributions

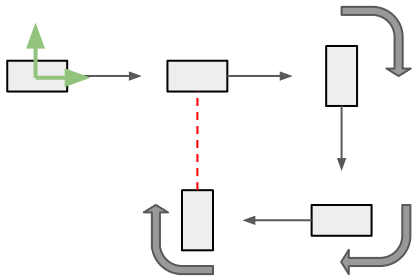


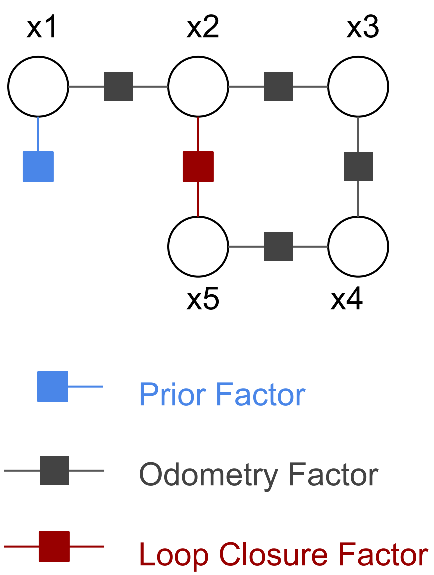
Prior Factor

Odometry Factor

Loop Closure Factor

First C++ Example





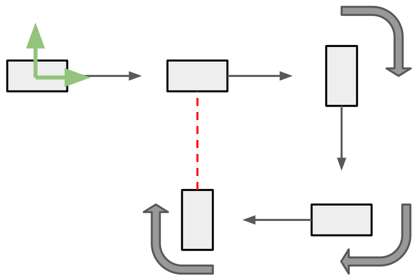
1. Build Factor Graph

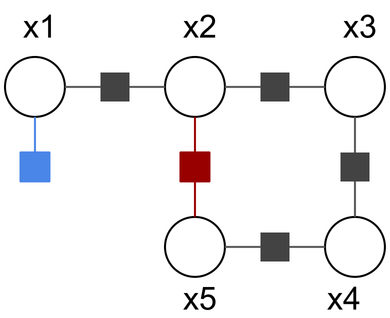
```cpp
// Create a factor graph container
NonlinearFactorGraph graph;

// Add a prior on the first pose, setting it to the origin
// The prior is needed to fix/align the whole trajectory at world frame
// A prior factor consists of a mean value and a noise model (covariance matrix)
noiseModel::Diagonal::shared_ptr priorModel = noiseModel::Diagonal::Sigmas(Vector3(1.0, 1.0, 0.1));
graph.add(PriorFactor<Pose2>(Symbol('x', 1), Pose2(0, 0, 0), priorModel));

// odometry measurement noise model (covariance matrix)
noiseModel::Diagonal::shared_ptr odomModel = noiseModel::Diagonal::Sigmas(Vector3(0.5, 0.5, 0.1));

// Add odometry factors
// Create odometry (Between) factors between consecutive poses
// robot makes 90 deg right turns at x3 - x5
graph.add(BetweenFactor<Pose2>(Symbol('x', 1), Symbol('x', 2), Pose2(5, 0, 0), odomModel));
graph.add(BetweenFactor<Pose2>(Symbol('x', 2), Symbol('x', 3), Pose2(5, 0, -M_PI_2), odomModel));
graph.add(BetweenFactor<Pose2>(Symbol('x', 3), Symbol('x', 4), Pose2(5, 0, -M_PI_2), odomModel));
graph.add(BetweenFactor<Pose2>(Symbol('x', 4), Symbol('x', 5), Pose2(5, 0, -M_PI_2), odomModel));

// loop closure measurement noise model
noiseModel::Diagonal::shared_ptr loopModel = noiseModel::Diagonal::Sigmas(Vector3(0.5, 0.5, 0.1));

// Add the loop closure constraint
graph.add(BetweenFactor<Pose2>(Symbol('x', 5), Symbol('x', 2), Pose2(5, 0, -M_PI_2), loopModel));

// print factor graph
graph.print("\nFactor Graph:\n");
```

https://github.com/dongjing3309/gtsam-examples/blob/master/cpp/examples/Pose2SLAMExample.cpp

First C++ Example

Georgia Tech Institute for Robotics and Intelligent Machines





Prior Factor

Odometry Factor

Loop Closure Factor

2. Noisy Initial Values

```
// initial varible values for the optimization
// add random noise from ground truth values
Values initials;
initials.insert(Symbol('x', 1), Pose2(0.2, -0.3, 0.2));
initials.insert(Symbol('x', 2), Pose2(5.1, 0.3, -0.1));
initials.insert(Symbol('x', 3), Pose2(9.9, -0.1, -M_PI_2 - 0.2));
initials.insert(Symbol('x', 4), Pose2(10.2, -5.0, -M_PI + 0.1));
initials.insert(Symbol('x', 5), Pose2(5.1, -5.1, M_PI_2 - 0.1));

// print initial values
initials.print("\nInitial Values:\n");
```

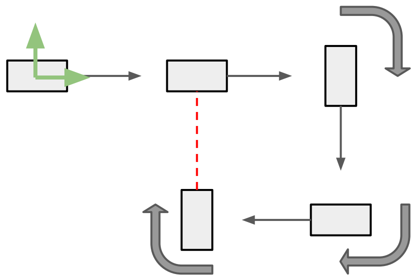
3. Optimize!

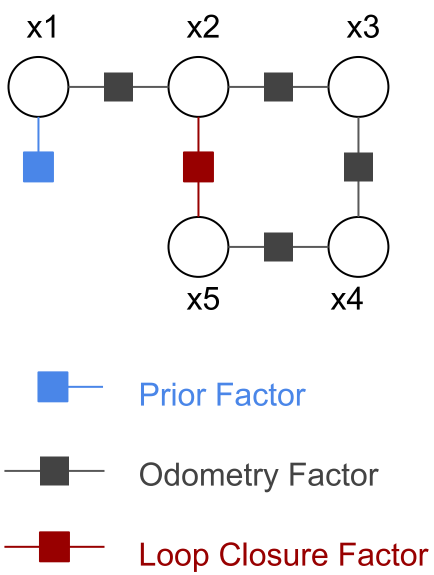
```
// Use Gauss-Newton method optimizes the initial values
GaussNewtonParams parameters;

// print per iteration
parameters.setVerbosity("ERROR");

// optimize!
GaussNewtonOptimizer optimizer(graph, initials, parameters);
Values results = optimizer.optimize();

// print final values
results.print("Final Result:\n");
```

4. (Optinal) Post Process like Marginals

```
// Calculate marginal covariances for all poses
Marginals marginals(graph, results);

// print marginal covariances
cout << "x1 covariance:\n" << marginals.marginalCovariance(Symbol('x', 1)) << endl;
cout << "x2 covariance:\n" << marginals.marginalCovariance(Symbol('x', 2)) << endl;
cout << "x3 covariance:\n" << marginals.marginalCovariance(Symbol('x', 3)) << endl;
cout << "x4 covariance:\n" << marginals.marginalCovariance(Symbol('x', 4)) << endl;
cout << "x5 covariance:\n" << marginals.marginalCovariance(Symbol('x', 5)) << endl;
```

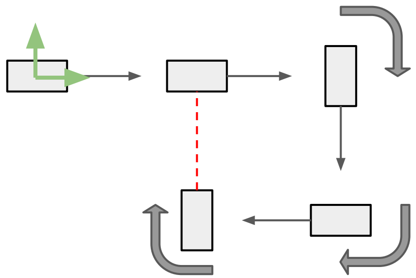
First C++ Example

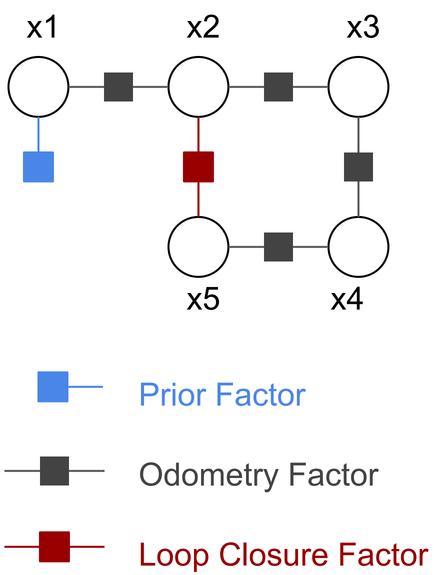





```
Initial error: 18.510326
newError: 0.122934358
errorThreshold: 0.122934358 > 0
absoluteDecrease: 18.3873916591 >= 1e-05
relativeDecrease: 0.993358606565 >= 1e-05
newError: 8.85829965247e-06
errorThreshold: 8.85829965247e-06 > 0
absoluteDecrease: 0.12292549938 >= 1e-05
relativeDecrease: 0.999927942848 >= 1e-05
newError: 3.68234845905e-15
errorThreshold: 3.68234845905e-15 > 0
absoluteDecrease: 8.85829964879e-06 < 1e-05
relativeDecrease: 0.999999999584 >= 1e-05
converged
errorThreshold: 3.68234845905e-15 <? 0
absoluteDecrease: 8.85829964879e-06 <? 1e-05
relativeDecrease: 0.999999999584 <? 1e-05
iterations: 3 >? 100
Final Result:
Values with 5 values:
Value x1: (N5gtsam5Pose2E) (-3.17592454561e-18, 5.21439530413e-19, 2.17083859205e-20)

Value x2: (N5gtsam5Pose2E) (5, 7.60341342619e-19, 1.73447953203e-20)

Value x3: (N5gtsam5Pose2E) (10.0000000015, -4.40576430129e-09, -1.5707963267)

Value x4: (N5gtsam5Pose2E) (10.0000000114, -5.00000003139, 3.14159265352)

Value x5: (N5gtsam5Pose2E) (4.99999999784, -5.00000000264, 1.57079632663)

x1 covariance:
                  1  1.09613818193e-18 -3.52006030097e-17
 1.09613818193e-18                  1  1.42108547152e-16
-3.52006030097e-17  1.42108547152e-16               0.01
x2 covariance:
              1.25 -2.18298661793e-16  -8.8071537939e-17
-2.18298661793e-16                1.5               0.05
 -8.8071537939e-17               0.05               0.02
x3 covariance:
     2.70000000047 -8.21534004474e-10    -0.155000000029
-8.21533972918e-10      1.45000000006  -0.00499999990562
   -0.155000000029  -0.00499999990562    0.0264999999985
x4 covariance:
    2.1125000074  0.800000006448 -0.120000000784
  0.800000006448   2.80000000387 -0.170000000296
 -0.120000000784 -0.170000000296 0.0279999999952
x5 covariance:
   1.69999999991 -0.224999999954 0.0449999999659
 -0.224999999954   2.06250000049 -0.127500000037
 0.0449999999659 -0.127500000037 0.0264999999968
```

Use GTSAM in Matlab



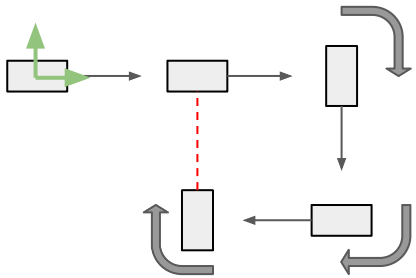


1. Build Factor Graph

```
% Create a factor graph container
graph = NonlinearFactorGraph;

% Add a prior on the first pose, setting it to the origin
% The prior is needed to fix/align the whole trajectory at world frame
% A prior factor consists of a mean value and a noise model (covariance matrix)
priorModel = noiseModel.Diagonal.Sigmas([1.0, 1.0, 0.1]');
graph.add(PriorFactorPose2(symbol('x', 1), Pose2(0, 0, 0), priorModel));

% odometry measurement noise model (covariance matrix)
odomModel = noiseModel.Diagonal.Sigmas([0.5, 0.5, 0.1]');

% Add odometry factors
% Create odometry (Between) factors between consecutive poses
% robot makes 90 deg right turns at x3 - x5
graph.add(BetweenFactorPose2(symbol('x', 1), symbol('x', 2), Pose2(5, 0, 0), odomModel));
graph.add(BetweenFactorPose2(symbol('x', 2), symbol('x', 3), Pose2(5, 0, -pi/2), odomModel));
graph.add(BetweenFactorPose2(symbol('x', 3), symbol('x', 4), Pose2(5, 0, -pi/2), odomModel));
graph.add(BetweenFactorPose2(symbol('x', 4), symbol('x', 5), Pose2(5, 0, -pi/2), odomModel));

% loop closure measurement noise model
loopModel = noiseModel.Diagonal.Sigmas([0.5, 0.5, 0.1]');

% Add the loop closure constraint
graph.add(BetweenFactorPose2(symbol('x', 5), symbol('x', 2), Pose2(5, 0, -pi/2), loopModel));

% print factor graph
graph.print('\nFactor Graph:\n');
```

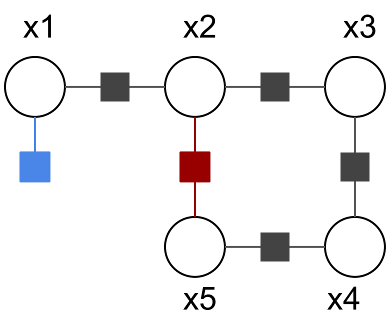
https://github.com/dongjing3309/gtsam-examples/blob/master/matlab/Pose2SLAMExample.m

Use GTSAM in Matlab



2. Noisy Initial Values

```
% initial varible values for the optimization
% add random noise from ground truth values
initials = Values;
initials.insert(symbol('x', 1), Pose2(0.2, -0.3, 0.2));
initials.insert(symbol('x', 2), Pose2(5.1, 0.3, -0.1));
initials.insert(symbol('x', 3), Pose2(9.9, -0.1, -pi/2 - 0.2));
initials.insert(symbol('x', 4), Pose2(10.2, -5.0, -pi + 0.1));
initials.insert(symbol('x', 5), Pose2(5.1, -5.1, pi/2 - 0.1));

% print initial values
initials.print('\nInitial Values:\n');
```



3. Optimize!

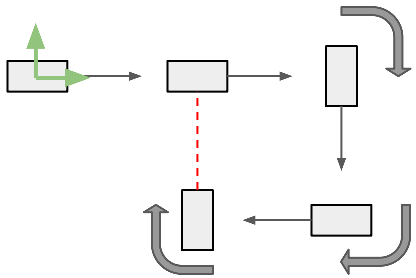
```
% Use Gauss-Newton method optimizes the initial values
parameters = GaussNewtonParams;

% print per iteration
parameters.setVerbosity('ERROR');

% optimize!
optimizer = GaussNewtonOptimizer(graph, initials, parameters);
results = optimizer.optimizeSafely();

% print final values
results.print('Final Result:\n');
```

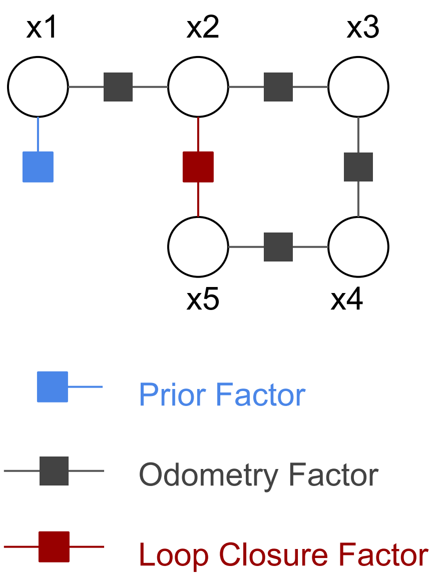
Prior Factor

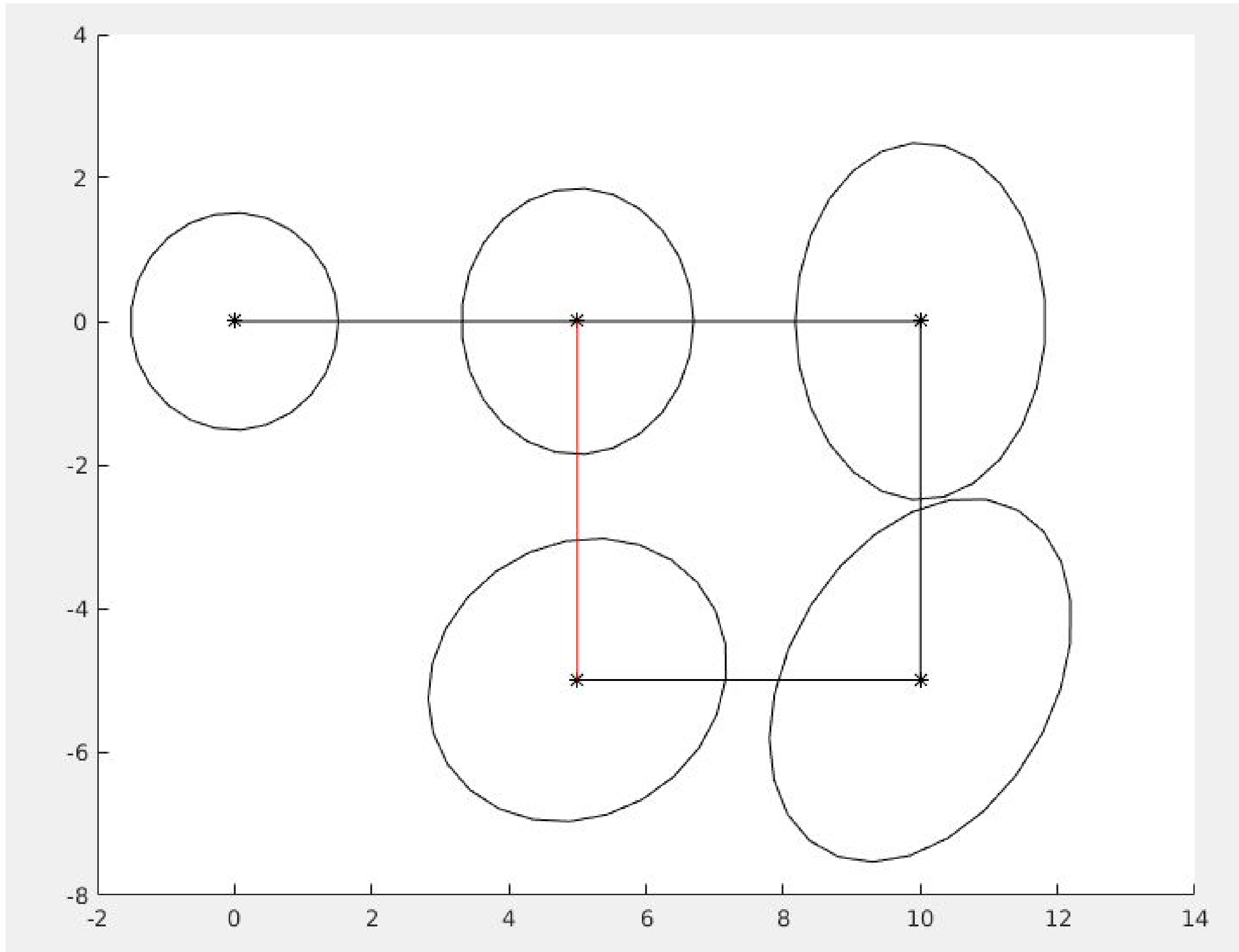
Odometry Factor

Loop Closure Factor

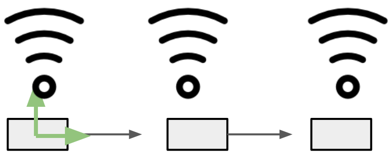
Use GTSAM in Matlab

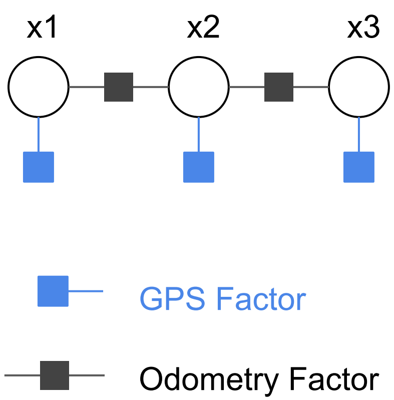






Write your own factor





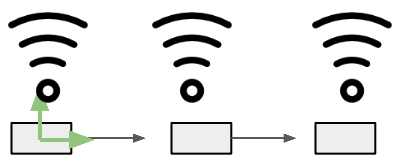
- GTSAM doesn’t have factors for all sensor...
- Customize your factor based on your sensors
- Design a cost function to minimize

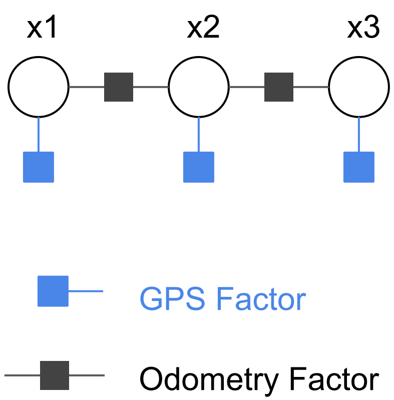
- Here we consider a position-only measurement (like GPS), the error is difference of estimated position and measured position.

$$e = [x - m_x, y - m_y]^T$$

Georgia Tech | Institute for Robotics and Intelligent Machines

Write your own factor





Derived from a GTSAM NoiseModelFactor unary factor class

```
class GPSPose2Factor: public gtsam::NoiseModelFactor1<gtsam::Pose2> {

private:
  // measurement information
  double mx_, my_;

public:

  /**
   * Constructor
   * @param poseKey    associated pose varible key
   * @param model      noise model for GPS snesor, in X-Y
   * @param m          Point2 measurement
   */
  GPSPose2Factor(gtsam::Key poseKey, const gtsam::Point2 m, gtsam::SharedNoiseModel model) :
      gtsam::NoiseModelFactor1<gtsam::Pose2>(model, poseKey), mx_(m.x()), my_(m.y()) {}

  // error function
  // @param p    the pose in Pose2
  // @param H    the optional Jacobian matrix, which use boost optional and has default null pointer
  gtsam::Vector evaluateError(const gtsam::Pose2& p, boost::optional<gtsam::Matrix&> H = boost::none) const {

    // note that use boost optional like a pointer
    // only calculate jacobian matrix when non-null pointer exists
    if (H) *H = (gtsam::Matrix23() << 1.0, 0.0, 0.0,
                                      0.0, 1.0, 0.0).finished();

    // return error vector
    return (gtsam::Vector2() << p.x() - mx_, p.y() - my_).finished();
  }

};
```

Contains measurement

Initial Base class by variable key and noise model

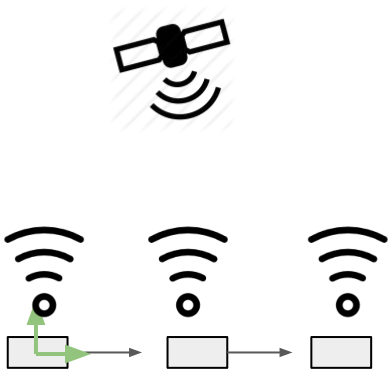
Implement evaluateError function for cost

Optional Jacobians are needed (generally the hardest part!!!)

Return cost vector

https://github.com/dongjing3309/gtsam-examples/blob/master/cpp/GPSPose2Factor.h

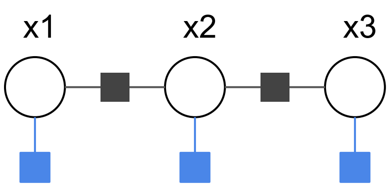
Write your own factor



Insert in Factor Graph

Noise model dimension should match error vector dimension

```
// 2D 'GPS' measurement noise model, 2-dim
noiseModel::Diagonal::shared_ptr gpsModel = noiseModel::Diagonal::Sigmas(Vector2(1.0, 1.0));

// Add the GPS factors
// note that there is NO prior factor needed at first pose, since GPS provides
// the global positions (and rotations given more than 1 GPS measurements)
graph.add(GPSPose2Factor(Symbol('x', 1), Point2(0, 0), gpsModel));
graph.add(GPSPose2Factor(Symbol('x', 2), Point2(5, 0), gpsModel));
graph.add(GPSPose2Factor(Symbol('x', 3), Point2(10, 0), gpsModel));
```



GPS Factor

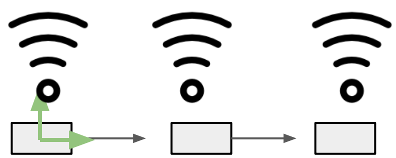
Odometry Factor

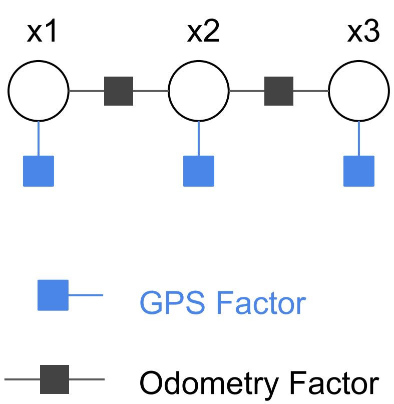
Results

```
Final Result:
Values with 3 values:
Value x1: (N5gtsam5Pose2E) (-4.48009679975e-09, -1.16228632318e-09, 9.13457071163e-12)

Value x2: (N5gtsam5Pose2E) (5.00000000041, -2.83556543846e-10, 8.48152159162e-12)

Value x3: (N5gtsam5Pose2E) (10.0000000008, -9.88525542862e-10, 8.48152159416e-12)

x1 covariance:
   0.446153846154 -3.0054380073e-11 8.47086723929e-12
-3.0054380073e-11    0.851851851992    -0.103703703667
8.47086723929e-12   -0.103703703667    0.0274074073781
x2 covariance:
    0.384615384615 -2.02724947854e-13 -1.21164126794e-11
-2.02724947854e-13      0.40740740743  -0.00740740738071
-1.21164126794e-11  -0.00740740738071     0.0274074073842
x3 covariance:
   0.446153846154 2.31023131834e-11 6.69379396621e-12
2.31023131834e-11    0.851851851723     0.10370370364
6.69379396621e-12     0.10370370364    0.0374074073842
```

https://github.com/dongjing3309/gtsam-examples/blob/master/cpp/examples/Pose2GPSExample.cpp

Use your own factor in Matlab





- Factors are defined in C++, how to use in Matlab?
- Technique: GTSAM can generate .mex file and .m file for given C++ code (classes and functions)
- Usage: declear classes/functions needed in Matlab in a `{project_name}.h` file, and call `wrap_and_install_library` in CMake

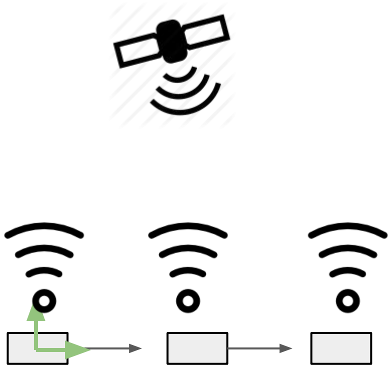
gtsamexamples.h

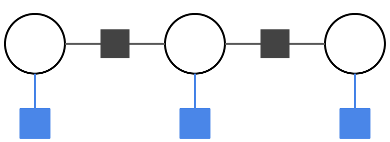
```
namespace gtsamexamples {

// GPS factor for Pose2
#include <cpp/GPSPose2Factor.h>

virtual class GPSPose2Factor : gtsam::NoiseModelFactor {
  GPSPose2Factor(size_t poseKey, const gtsam::Point2& m, gtsam::noiseModel::Base* model);
};

} // namespace gtsamexamples
```

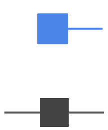
CMakeLists.txt

```
# Wrapping to MATLAB
if(EXAMPLES_BUILD_MATLAB_TOOLBOX)
  # wrap
  include(GtsamMatlabWrap)
  wrap_and_install_library(gtsamexamples.h ${PROJECT_NAME} "${CMAKE_CURRENT_SOURCE_DIR}" "")
endif()
```

Use your own factor in Matlab

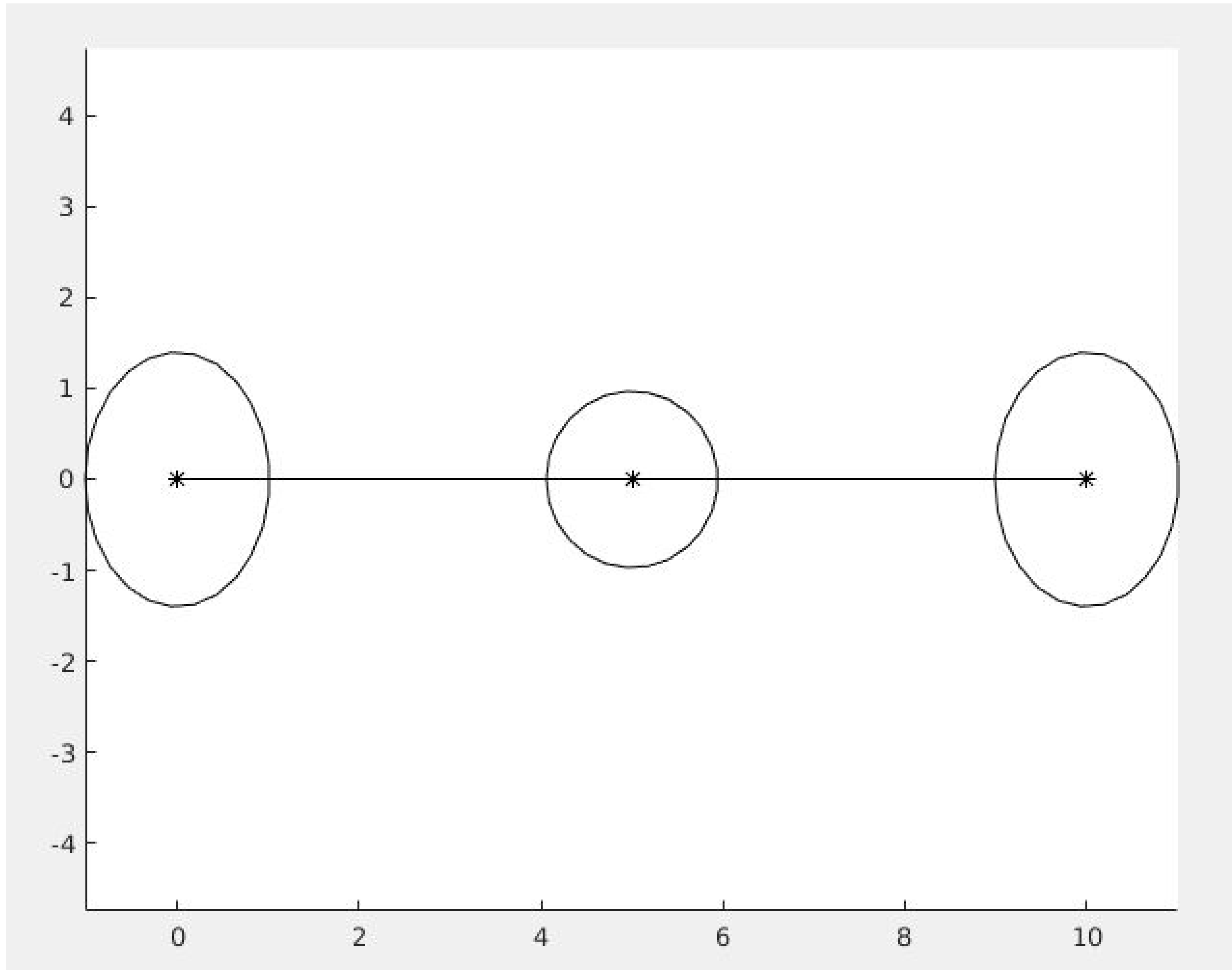






GPS Factor

Odometry Factor



Expression: Automatic Differentiation (AD)

- Recall that the hardset part to write your own factor is the Jacobians!
- If the cost function can be decomposed to several functions which have Jacobians easier to calculate, we can apply chain rule:

$$e = f(g(h(x)))$$

$$\frac{\partial e}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial h}\frac{\partial h}{\partial x}$$

- Automatic Differentiation (AD) can do this for you, by just providing each function plus jacobians!

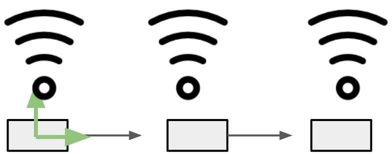
Expression: Automatic Differentiation (AD)

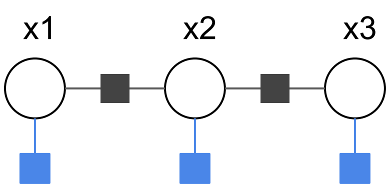
- GTSAM implements AD by `Expression`
- An `Expression` can be a variable, a function, or a constant
- `Expression` can take Expressions as input to apply chain rule

- Example: compute `func_a` of x1 and x2, then calculate the `func_b` of `func_a` result and a constant c1

```
// Expression type for Point3
typedef Expression<Point3> Point3_

// Expressions for variables
Point3_ x1('x'1), x2('x',2);
// Expressions for const
Point3_ c1(Point3(1., 2., 3.));

// Expressions for function func_b(func_a(x1, x2), c1)
Point3_ g(&func_a, x1, x2);
Point3_ f(&func_b, g, c1);

// OR calculate the Expression g at once
Point3_ f(&func_b, Point3_(&func_a, x1, x2), c1);
```

Expression example: GPS expression





GPS Factor

Odometry Factor

functions.h

```
// function project Pose2 to Point2
gtsam::Point2 projectPose2(const gtsam::Pose2& pose,
    gtsam::OptionalJacobian<2,3> H = boost::none);
```

functions.cpp

```
/* ************************************************************************* */
gtsam::Point2 projectPose2(const gtsam::Pose2& pose, gtsam::OptionalJacobian<2,3> H) {

  // jacobian: left 2x2 identity + right 2x1 zero
  if (H) *H = (gtsam::Matrix23() << 1.0, 0.0, 0.0,
                                    0.0, 1.0, 0.0).finished();

  // return translation
  return gtsam::Point2(pose.x(), pose.y());
}
```

Design your cost function as usual

expressions.h

```
// expression project Pose2 to Point2
inline gtsam::Point2_ projectPose2_(const gtsam::Pose2_& pose) {
  return gtsam::Point2_(&projectPose2, pose);
}
```

Convert your cost function as expression

Pose2GPSExpressionExample.cpp

```
  // Add the GPS factors, by composing expressions
  // note that there is NO prior factor needed at first pose, since GPS provides
  // the global positions (and rotations given more than 1 GPS measurements)
  graph.addExpressionFactor(projectPose2_(x1_), Point2(0, 0), gpsModel);
  graph.addExpressionFactor(projectPose2_(x2_), Point2(5, 0), gpsModel);
  graph.addExpressionFactor(projectPose2_(x3_), Point2(10, 0), gpsModel);
```

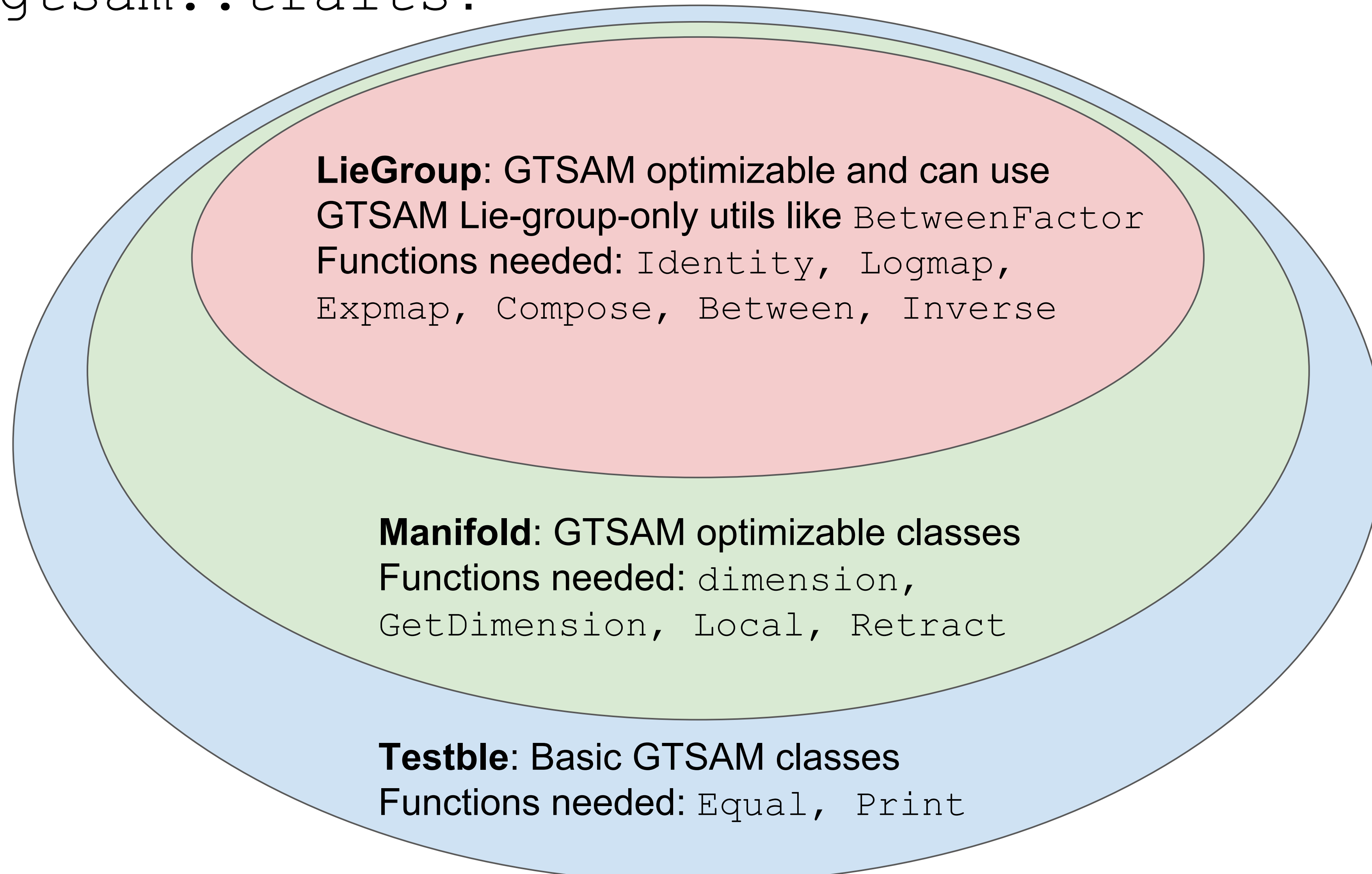
Expression factor has error |f(x) - z|^2

Traits: Optimize any type in GTSAM

- You may want to optimize variable types other than GTSAM provided Vector, SE(2), SO(3), SE(3), etc… (although GTSAM provides a lot!)
  - e.g. State space of a mobile manipulator (mobile base + a 7 DOF arm) is SE(2) x R(7).
- You may not have access to change the types
  - e.g. You are using some classes by other libs like g2o, ceres, etc.)

- `gtsam::traits` are a step towards making GTSAM more modern and more efficient, by defining type properties such as dimensionality, group-ness, etc with `boost::traits` style meta-functions.
- Data structure `gtsam::Values` can now take any type, provided the necessary `gtsam::traits` are defined.

How GTSAM understand objects by `gtsam::traits`?



`gtsam::traits` example

- A minimal custom 2D point R(2) class
- Can be treated as a Lie group (a vector space is a naive Lie group)
- But nothing about Lie group property inside class

```
namespace gtsamexamples {

// A minimal 2D point class, 'c' meas custom
struct Point2c {
  double x;
  double y;

  // convenience constructor
  Point2c(double xi, double yi) : x(xi), y(yi) {}
};

} // namespace gtsamexamples
```

- Traits must be in namespace `gtsam`
- `gtsam::traits` is a *template specialization* for type `Point2c`
- Fill in the functions needed in `gtsam::traits`, depends on the type you want to define for `Point2c` (Testable / Manifold / LieGroup)

```
// traits must in namespace gtsam
namespace gtsam {

template<>
struct traits<gtsamexamples::Point2c> {
```

Georgia Tech Institute for Robotics and Intelligent Machines

`gtsam::traits` example

```cpp
namespace gtsamexamples {

// A minimal 2D point class, 'c' meas custom
struct Point2c {
  double x;
  double y;

  // convenience constructor
  Point2c(double xi, double yi) : x(xi), y(yi) {}
};

} // namespace gtsamexamples
```

Functions as Testble

```cpp
/**
 * Basic (Testable)
 */

// print
static void Print(const gtsamexamples::Point2c& m, const std::string& str = "") {
  std::cout << str << "(" << m.x << ", " << m.y << ")" << std::endl;
}

// equality with optional tol
static bool Equals(const gtsamexamples::Point2c& m1, const gtsamexamples::Point2c& m2,
    double tol = 1e-8) {
  if (fabs(m1.x - m2.x) < tol && fabs(m1.y - m2.y) < tol)
    return true;
  else
    return false;
}
```

Functions as Manifold

```cpp
/**
 * Manifold
 */

// use enum dimension
enum { dimension = 2 };
static int GetDimension(const gtsamexamples::Point2c&) { return dimension; }

// Typedefs needed
typedef gtsamexamples::Point2c ManifoldType;
typedef Eigen::Matrix<double, dimension, 1> TangentVector;

// Local coordinate of Point2c is naive (since vectorspace)
static TangentVector Local(const gtsamexamples::Point2c& origin,
    const gtsamexamples::Point2c& other) {
  return Vector2(other.x - origin.x, other.y - origin.y);
}

// Retraction back to manifold of Point2c is naive (since vectorspace)
static gtsamexamples::Point2c Retract(const gtsamexamples::Point2c& origin,
    const TangentVector& v) {
  return gtsamexamples::Point2c(origin.x + v(0), origin.y + v(1));
}
```

`gtsam::traits` example

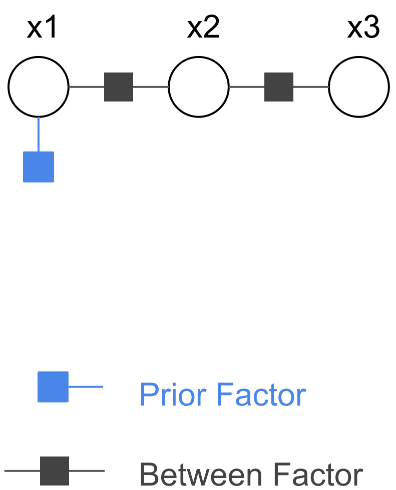
Georgia Tech Institute for Robotics and Intelligent Machines

```
namespace gtsamexamples {

// A minimal 2D point class, 'c' meas custom
struct Point2c {
  double x;
  double y;

  // convenience constructor
  Point2c(double xi, double yi) : x(xi), y(yi) {}
};

} // namespace gtsamexamples
```

Functions as Lie group

```
/**
 * Lie group
 */

// indicate this group using operator *,
// if uses +/- then use option additive_group_tag
typedef multiplicative_group_tag group_flavor;

// typedefs
typedef OptionalJacobian<dimension, dimension> ChartJacobian;

static gtsamexamples::Point2c Identity() {
  return gtsamexamples::Point2c(0, 0);
}

static TangentVector Logmap(const gtsamexamples::Point2c& m,
    ChartJacobian Hm = boost::none) {
  if (Hm) *Hm = Matrix2::Identity();
  return Vector2(m.x, m.y);
}

static gtsamexamples::Point2c Expmap(const TangentVector& v,
    ChartJacobian Hv = boost::none) {
  if (Hv) *Hv = Matrix2::Identity();
  return gtsamexamples::Point2c(v(0), v(1));
}

static gtsamexamples::Point2c Compose(const gtsamexamples::Point2c& m1,
    const gtsamexamples::Point2c& m2,
    ChartJacobian H1 = boost::none, ChartJacobian H2 = boost::none) {
  if (H1) *H1 = Matrix2::Identity();
  if (H2) *H2 = Matrix2::Identity();
  return gtsamexamples::Point2c(m1.x + m2.x, m1.y + m2.y);
}

static gtsamexamples::Point2c Between(const gtsamexamples::Point2c& m1,
    const gtsamexamples::Point2c& m2, //
    ChartJacobian H1 = boost::none, ChartJacobian H2 = boost::none) {
  if (H1) *H1 = -Matrix2::Identity();
  if (H2) *H2 = Matrix2::Identity();
  return gtsamexamples::Point2c(m2.x - m1.x, m2.y - m1.y);
}

static gtsamexamples::Point2c Inverse(const gtsamexamples::Point2c& m, //
    ChartJacobian H = boost::none) {
  if (H) *H = -Matrix2::Identity();
  return gtsamexamples::Point2c(-m.x, -m.y);
}
};
```

gtsam::traits example



CustomPoint2Example.cpp

```cpp
// first state prior noise model (covariance matrix)
noiseModel::Diagonal::shared_ptr priorModel = noiseModel::Diagonal::Sigmas(Vector2(0.2, 0.2));

// add prior factor on first state (at origin)
graph.add(PriorFactor<Point2c>(Symbol('x', 1), Point2c(0, 0), priorModel));

// odometry measurement noise model (covariance matrix)
noiseModel::Diagonal::shared_ptr odomModel = noiseModel::Diagonal::Sigmas(Vector2(0.5, 0.5));

// Add odometry factors
// Create odometry (Between) factors between consecutive point2c
graph.add(BetweenFactor<Point2c>(Symbol('x', 1), Symbol('x', 2), Point2c(2, 0), odomModel));
graph.add(BetweenFactor<Point2c>(Symbol('x', 2), Symbol('x', 3), Point2c(2, 0), odomModel));
graph.add(BetweenFactor<Point2c>(Symbol('x', 3), Symbol('x', 4), Point2c(2, 0), odomModel));
graph.add(BetweenFactor<Point2c>(Symbol('x', 4), Symbol('x', 5), Point2c(2, 0), odomModel));

// print factor graph
graph.print("\nFactor Graph:\n");


// initial varible values for the optimization
// add random noise from ground truth values
Values initials;
initials.insert(Symbol('x', 1), Point2c(0.2, -0.3));
initials.insert(Symbol('x', 2), Point2c(2.1, 0.3));
initials.insert(Symbol('x', 3), Point2c(3.9, -0.1));
initials.insert(Symbol('x', 4), Point2c(5.9, -0.3));
initials.insert(Symbol('x', 5), Point2c(8.2, 0.1));

// print initial values
initials.print("\nInitial Values:\n");
```

```
Final Result:
Values with 5 values:
Value x1: (N13gtsamexamples7Point2cE) (4.5777435178e-33, 9.1554870356e-33)

Value x2: (N13gtsamexamples7Point2cE) (2, 7.39557098645e-32)

Value x3: (N13gtsamexamples7Point2cE) (4, 4.93038065763e-32)

Value x4: (N13gtsamexamples7Point2cE) (6, 4.93038065763e-32)

Value x5: (N13gtsamexamples7Point2cE) (8, 4.93038065763e-32)
```

All code shown in this section can be found in:
https://github.com/dongjing3309/gtsam-examples

Outline

- Theory
  - SLAM as a Factor Graph
  - SLAM as a Non-linear Least Squares
  - Optimization on Manifold/Lie Groups
  - iSAM2 and Bayes Tree
- Programming
  - First C++ example
  - Use GTSAM in Matlab
  - Write your own factor
  - Expression: Automatic Differentiation (AD) (New in 4.0!)
  - Traits: Optimize any type in GTSAM (New in 4.0!)
  - Use GTSAM in Python (New in 4.0!)
- Applications
  - Visual-Inertial Odometry
  - Structure from Motion (SfM)
  - Multi-Robot SLAM: Coordinate Frame and Distrubuted Optimization
  - Multi-View Stereo and Optical Flow
  - Motion Planning

Visual-Inertial Odometry

- IMU: Pre-integrated measurements between key-frames
- Visual landmarks: Structure-less factor by Schur complement

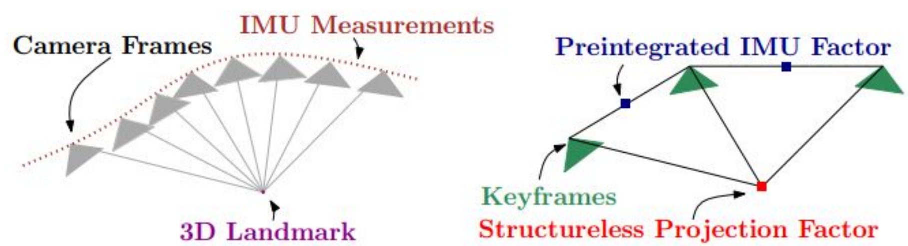


Fig. 3: Left: visual and inertial measurements in VIO. Right: factor graph in which several IMU measurements are summarized in a single preintegrated IMU factor and a structureless vision factor constraints keyframes observing the same landmark.

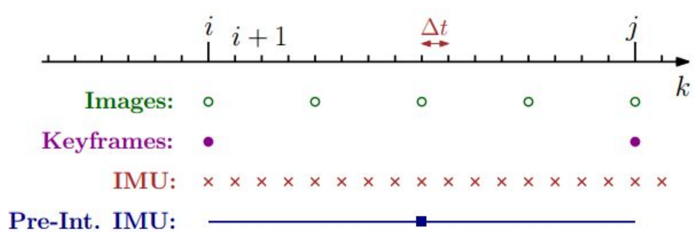
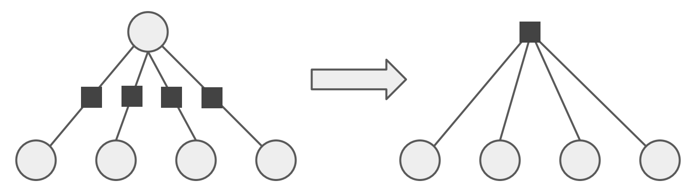


Fig. 4: Different rates for IMU and camera.



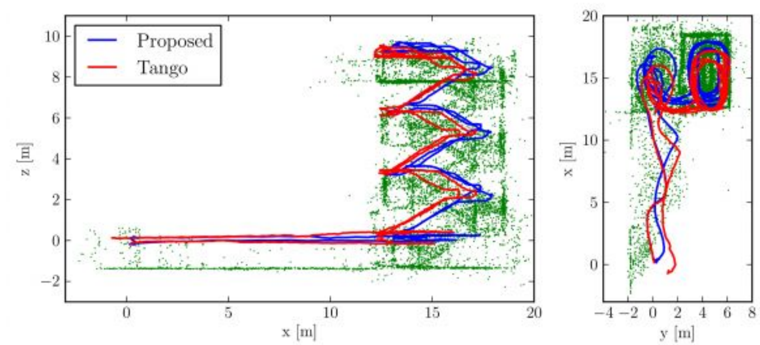
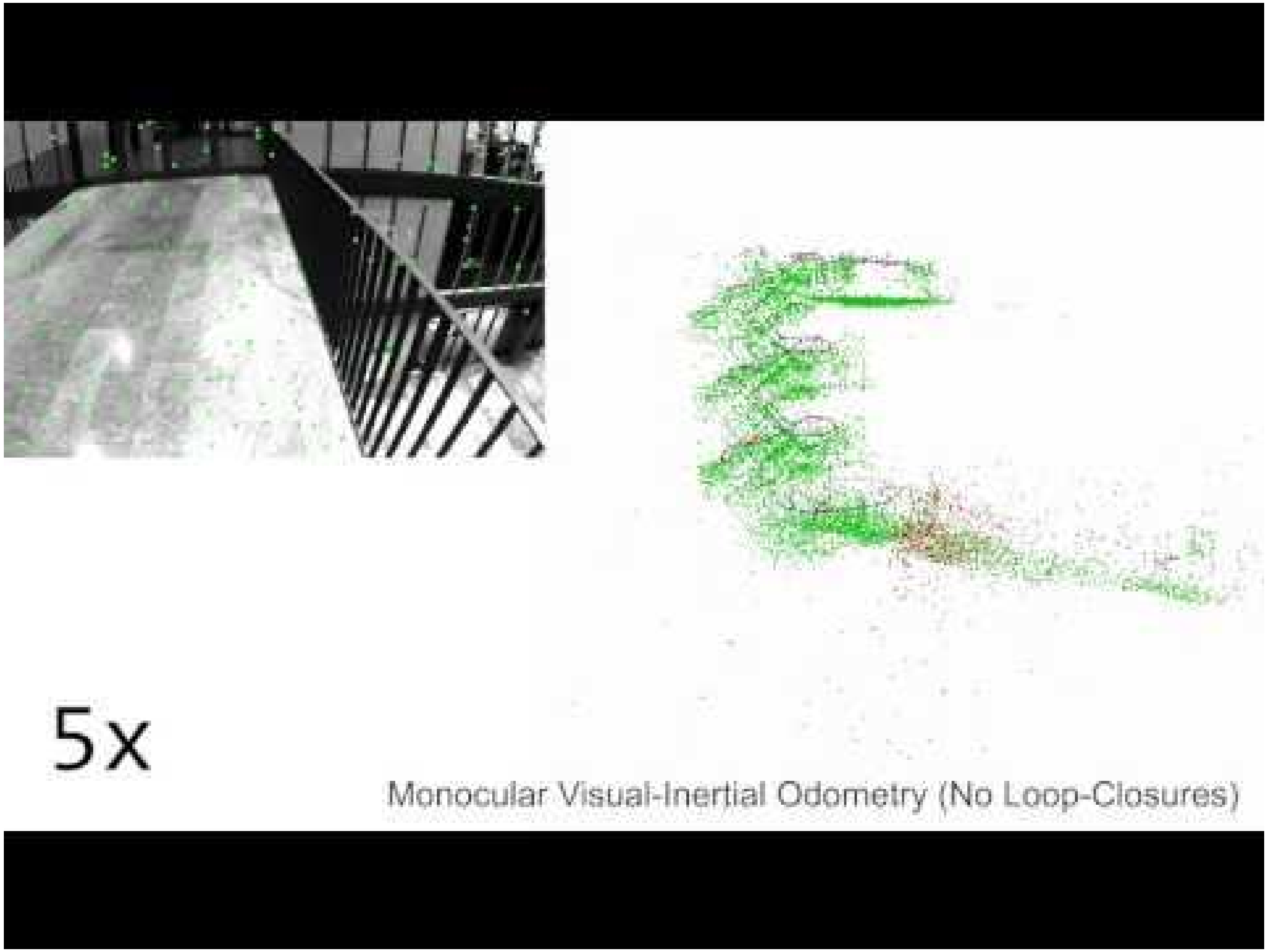


Fig. 19: Real test comparing the proposed VIO approach against Google Tango. The 160m-long trajectory starts at $(0, 0, 0)$ (ground floor), goes up till the 3rd floor of a building, and returns to the initial point. The figure shows a side view (left) and a top view (right) of the trajectory estimates for our approach (blue) and Tango (red). Google Tango accumulates 1.4m error, while the proposed approach only has 0.5m drift. 3D points triangulated from our trajectory estimate are shown in green for visualization purposes.

Forster, Christian, et al. "On-Manifold Preintegration for Real-Time Visual-Inertial Odometry." arXiv preprint arXiv:1512.02363 (2015).
Carlone, Luca, et al. "Eliminating conditionally independent sets in factor graphs: A unifying perspective based on smart factors." 2014 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2014.

Visual-Inertial Odometry



https://youtu.be/CsJkci5lfco

Structure from Motion (SfM)

- Large-scale spatio-temporal (4D) reconstruction for agriculture (offline)
- Multi sensor: camera, GPS, IMU

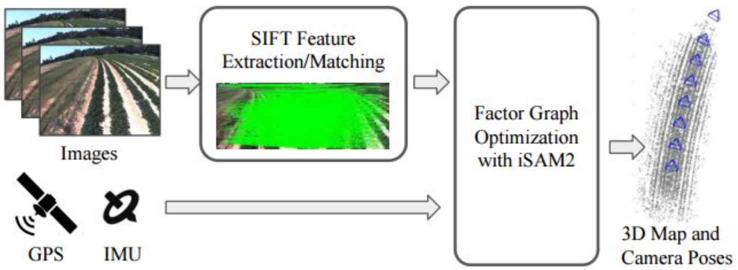


Fig. 3: Overview of multi-sensor SLAM system.

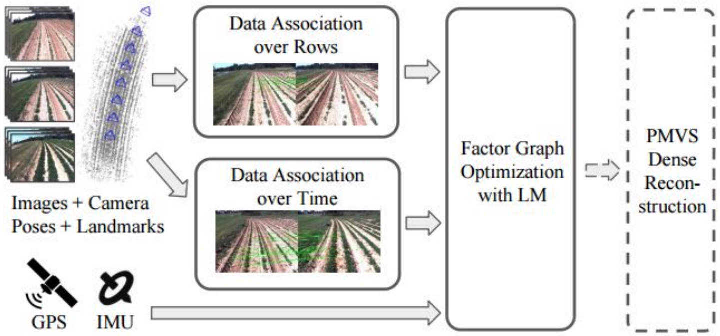


Fig. 9: Overview of 4D reconstruction pipeline. Dash box of PMVS dense reconstruction step means it is optional.

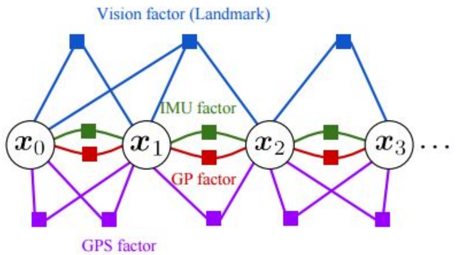
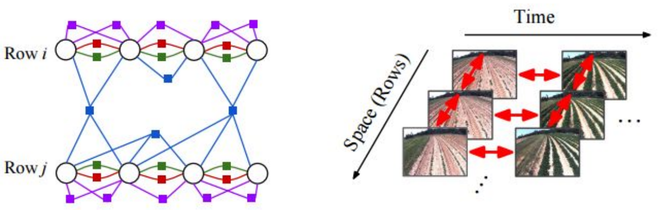


Fig. 4: Factor graph of multi-sensor SLAM.



(a) 4D factor graph (b) 4D data association pattern

Fig. 10: (a) Factor graph of two rows with data association, connected vision factors are shared (matched) landmarks in two rows. (b) Data association pattern of 4D reconstruction.

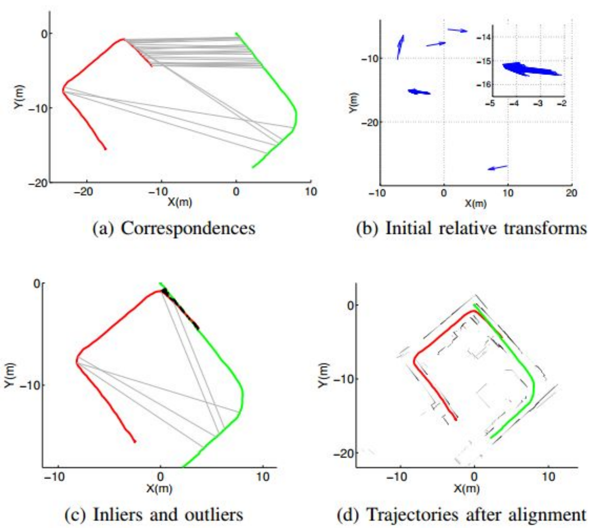
Dong, Jing, et al. "4D Crop Monitoring: Spatio-Temporal Reconstruction for Agriculture." arXiv preprint arXiv:1610.02482 (2016).

Structure from Motion (SfM)

https://youtu.be/BgLILIsKWzI

Multi-Robot SLAM

- Solve initial relative transformation -> a common reference frame
- Distributed optimization



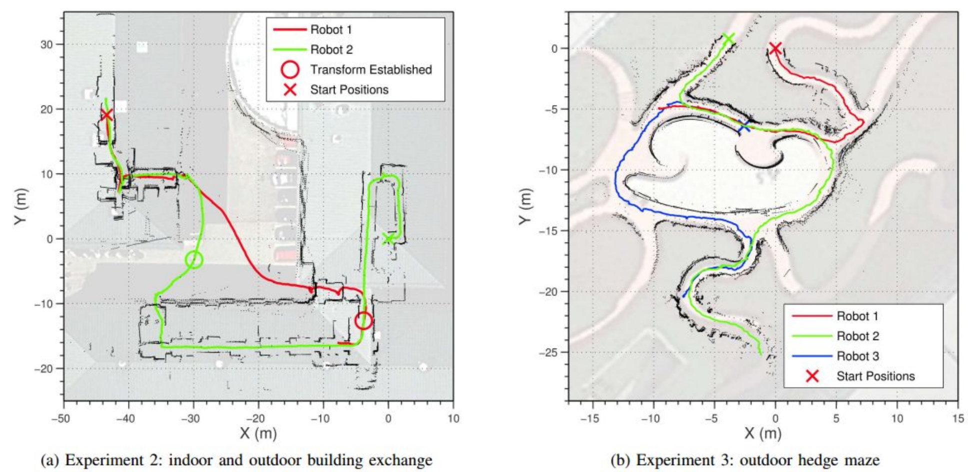
(a) Correspondences

(b) Initial relative transforms

(c) Inliers and outliers

(d) Trajectories after alignment

Fig. 2: The process of computing an initial relative transform. (a) shows correspondences between two trajectories. (b) depicts the transforms calculated through correspondences in (a) with an inlier cluster highlighted. (c) shows aligned trajectories with inliers and outliers in black and gray, respectively. (d) shows the resulting trajectories with scans.



(a) Experiment 2: indoor and outdoor building exchange

(b) Experiment 3: outdoor hedge maze

Fig. 5: Aligned trajectories resulting from our approach for two outdoor experiments on top of satellite imagery. The points at which the two robots in experiment 2 established a common reference frame are marked with circles.

Dong, Jing, et al. "Distributed real-time cooperative localization and mapping using an uncertainty-aware expectation maximization approach." 2015 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2015.

Multi-Robot SLAM

https://youtu.be/m_bLSdsT2kg

Dense Multi-View Stereo and Optical Flow

- Simiar to MRF, but use factor graph and least square optimization

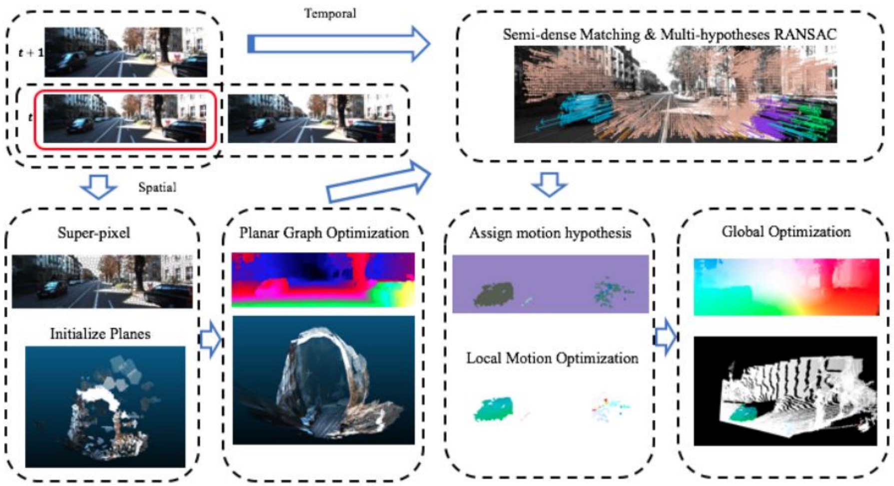


Fig. 1. An overview of our system: we estimate the 3D scene flow w.r.t. the reference image (the red bounding box), a stereo image pair and a temporal image pair as input. Image annotations show the results at each step. We assign a motion hypothesis to each superpixel as an initialization and optimize the factor graph for more accurate 3D motion. Finally, after global optimization, we show a projected 2D flow map in the reference frame and its 3D scene motion (static background are plotted in white).

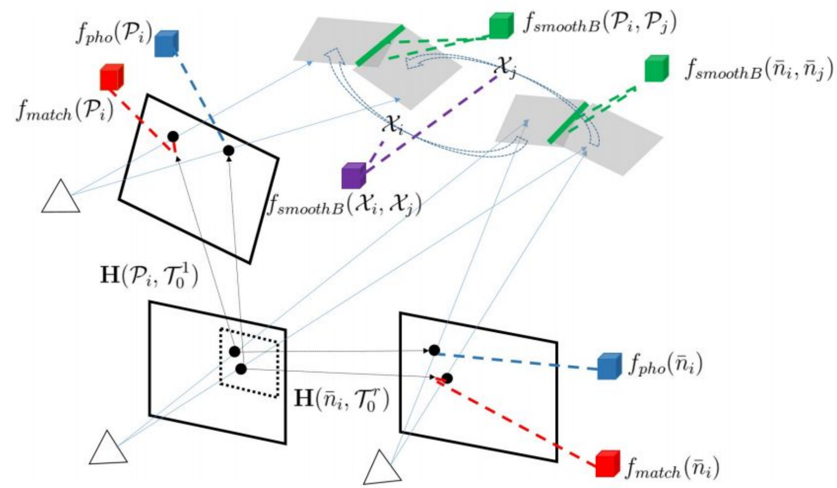


Fig. 2. The proposed factor graph for this scene flow problem. The unary factors are set up based on the homography transform relating two pixels, given $\mathcal{P}$. Binary factors are set up based on locally smooth and rigid assumptions. In this graph, a three-view geometry is used to explain factors for simplicity. Any other views can be constrained by incorporating the same temporal factors in this graph.

Lv, Zhaoyang, et al. "A Continuous Optimization Approach for Efficient and Accurate Scene Flow." European Conference on Computer Vision. Springer International Publishing, 2016.

Dense Multi-View Stereo and Optical Flow

https://youtu.be/2A7IOipPNBA

Georgia Tech | Institute for Robotics and Intelligent Machines

Motion Planning

- Solve trajectory optimization problems
- Minimize smooth cost + collision cost

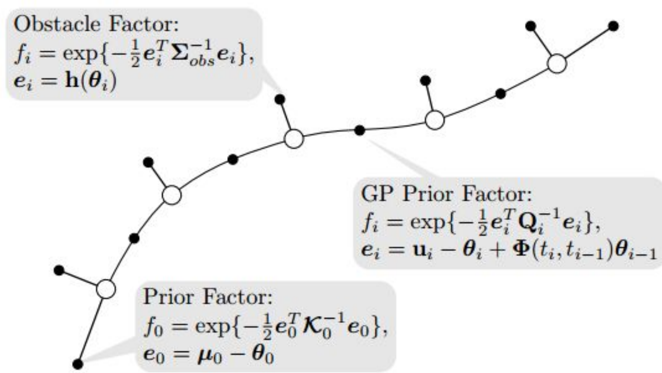


Fig. 1: A factor graph of an example trajectory optimization problem showing optimized states (white circles) and three kinds of factors (black dots), namely prior factors on start and goal states, obstacle cost factors on each state, and GP prior factors that connect consecutive states.

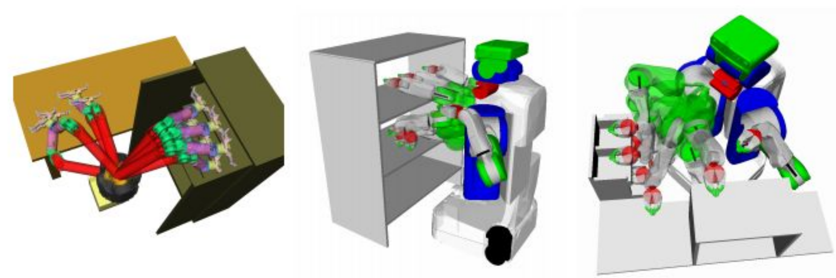
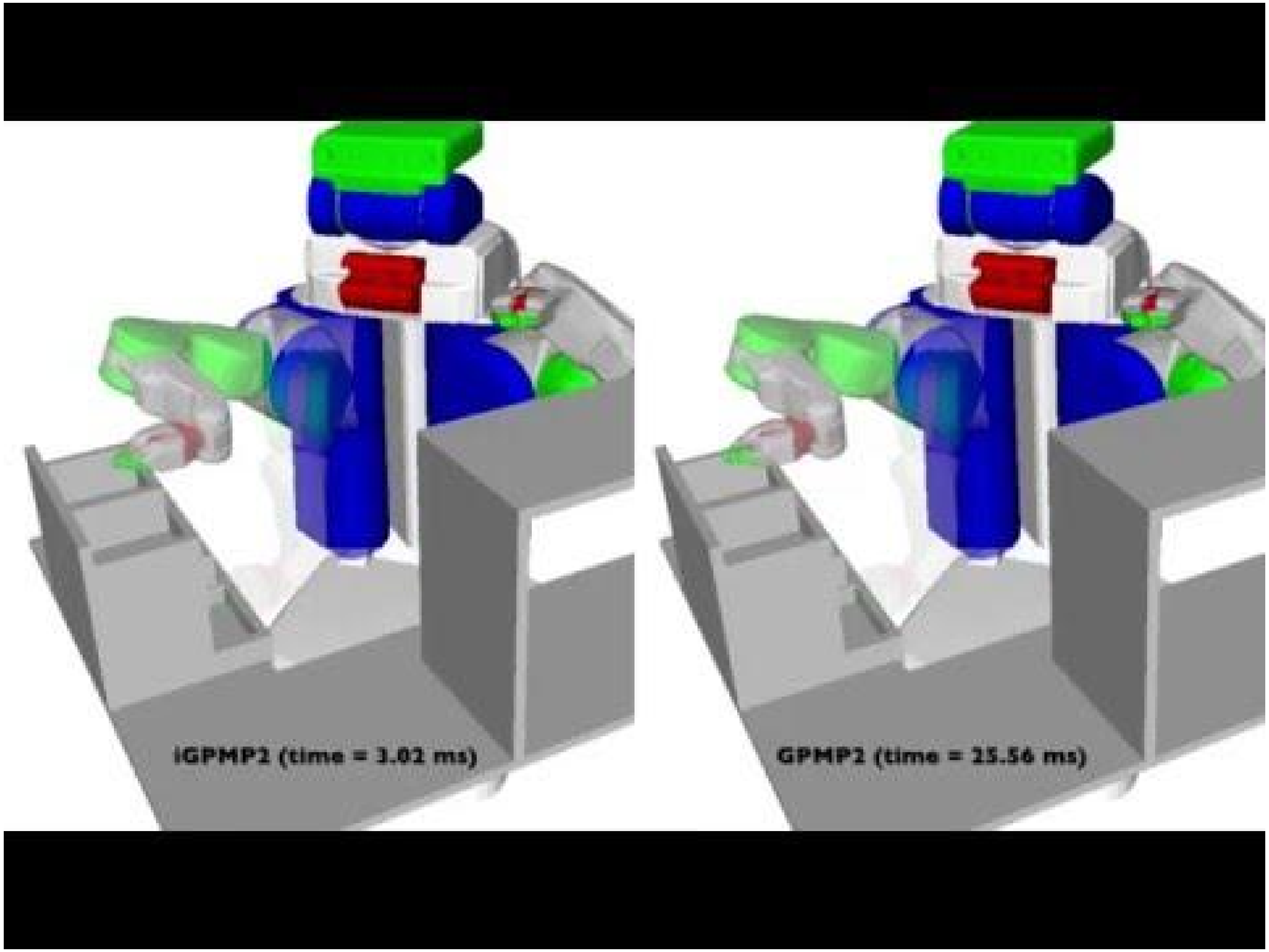


Fig. 5: Environments used for evaluation with robot start and goal configurations showing the WAM dataset (left), and PR2 dataset in *bookshelves* (center) and *industrial* scenes (right).

Dong, Jing, et al. "Motion Planning as Probabilistic Inference using Gaussian Processes and Factor Graphs." Robotics: Science and Systems (RSS), 2016

Motion Planning



https://youtu.be/mVA8qhGf7So

Acknowledgement

Georgia Tech Institute for Robotics and Intelligent Machines

Many thanks to my advisors!

Prof. Frank Dellaert
Prof. Byron Boots

and many thanks to collaborators and labmates!

Prof. Nathan Michael
Prof. Glen C. Rains
Luca Carlone
Vadim Indelman
Erik Nelson
Mustafa Mukadam
Zhaoyang Lv
Duy-Nguyen Ta
Yong-Dian Jian
and many...