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

● 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 Distributed 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 Distributed Optimization
  - Multi-View Stereo and Optical Flow
  - Motion Planning
SLAM as a Bayes Net

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

\[ x_i = f_i(x_{i-1}, u_i) + w_i \quad \Leftrightarrow \quad z_k = h_k(x_{i_k}, l_{jk}) + v_k \]

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

\[ P(z_k|x_{i_k}, l_{jk}) \propto \exp \left( -\frac{1}{2} \| h_k(x_{i_k}, l_{jk}) - z_k \|^2_{\Sigma_k} \right) \]
SLAM as a Factor Graph

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

\[ \Theta \overset{\Delta}{=} (X, L) \]

\[ \phi_0(x_0) \propto P(x_0) \]

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

\[ \psi_{ik,jk}(x_{ik}, l_{jk}) \propto P(z_k|x_{ik}, l_{jk}) \]
SLAM as a Non-linear Least Squares

- Maximum a posteriori (MAP) estimation

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

- Log likelihood

\[ \arg \min_{\Theta} (-\log f(\Theta)) = \arg \min_{\Theta} \frac{1}{2} \sum_i \| h_i(\Theta_i) - z_i \| \Sigma_i \]
Non-linear Least Squares

- Gauss-Newton method:
  \[ x^* = \arg\min_x \{ F(x) \}, \]
  where
  \[ F(x) = \frac{1}{2} \sum_{i=1}^{m} (f_i(x))^2 = \frac{1}{2} \|f(x)\|^2 = \frac{1}{2} f(x)^T f(x) \]

- Linear approximation of the vector function (get Jacobians)
  \[ f(x+h) = f(x) + J(x)h + O(\|h\|^2) \]
  \[ f(x+h) \approx \ell(h) \equiv f(x) + J(x)h \]

- Quadratic approximation of the cost error function (get Hessian)
  \[ F(x+h) \approx L(h) \equiv \frac{1}{2} \ell(h)^T \ell(h) \]
  \[ = \frac{1}{2} f^T f + h^T J^T f + \frac{1}{2} h^T J^T Jh \]
  \[ = F(x) + h^T J^T f + \frac{1}{2} h^T J^T Jh \]
  \[ (J^T J)h_{gn} = -J^T f. \]
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.

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

\[
\arg\min_{\Delta} (-\log f(\Delta)) = \arg\min_{\Delta} \| A\Delta - b \|^2
\]

Given \( \| \Delta \|_\Sigma^2 = \Delta^T \Sigma^{-1} \Delta = \Delta^T \Sigma^{-\frac{1}{2}} \Sigma^{-\frac{1}{2}} \Delta = \| \Sigma^{-\frac{1}{2}} \Delta \|^2 \)

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

\[
\begin{bmatrix}
  l_1 & l_2 & x_1 & x_2 & x_3 \\
  X & X & X & X & X \\
  X & X & X & X & X \\
  X & X & X & X & X \\
\end{bmatrix}
\]
Linear Least Squares

$$\delta^* = \arg\min_{\delta} \| A\delta - b \|_2^2$$

- 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

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

$$\mathcal{L} \overset{\Delta}{=} A^T A = R^T R$$

first $R^T y = A^T b$ 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 Θ' = Θ ⊕ Δ.

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.
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 $\xi$ back to a neighborhood in $G$ around $a$,

$$a \oplus \xi \triangleq a \exp \left( \xi \right)$$  \hspace{1cm} (3.1)

with $\xi \in \mathbb{R}^n$ for an $n$-dimensional Lie group. Above, $\xi \in g$ is the Lie algebra element corresponding to the vector $\xi$, and $\exp \xi$ the exponential map. Note that if $G$ is equal to $\mathbb{R}^n$ then composing with the exponential map $ae^\xi$ 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** $R : M \times \mathbb{R}^n \rightarrow M$, such that

$$a \oplus \xi \overset{A}{=} R_a(\xi)$$

A retraction [?] is required to be tangent to geodesics on the manifold $M$ at $a$. We can define many retractions for a manifold $M$, 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., $R_a(\xi) = a \cdot \exp \hat{\xi}$. 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.,

$$\lim_{\xi \rightarrow 0} \frac{|a \cdot \exp \hat{\xi} - R_a(\xi)|}{|\xi|} = 0$$

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 structure

iSAM2 and Bayes tree

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 Distributed Optimization
  ○ Multi-View Stereo and Optical Flow
  ○ Motion Planning
First C++ Example

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
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

2. Noisy Initial Values

```cpp
// initial variable 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, -5.6, -M_PI + 0.1));
 initials.insert(Symbol('x', 5), Pose2(5.1, -5.1, M_PI/2 - 0.1));

// print initial values
 initials.print("Initial Values:
");
```

3. Optimize!

```cpp
// 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:
");
```

4. (Optional) Post Process like Marginals

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

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

```
Initial error: 18.510326
newError: 0.122934358
diffError: 0.122934358 > 0
absouluteDecrease: 18.3873916591 >= 1e-05
relativeDecrease: 0.9933586666656 >= 1e-05
newError: 8.858299652476-06
diffError: 8.858299652476-06 > 0
absouluteDecrease: 0.122934358 >= 1e-05
relativeDecrease: 0.99992549938 >= 1e-05
newError: 3.58234845905e-15
diffError: 3.58234845905e-15 > 0
absouluteDecrease: 0.0035829964879e-06 < 1e-05
relativeDecrease: 0.999999999584 >= 1e-05
converged: errorThreshold: 3.58234845905e-15 < 7 0
absouluteDecrease: 8.85829964879e-06 < 1e-05
relativeDecrease: 0.999999999584 < 1e-05
iterations: 3 > 7 100
Final Result:
Values with 5 values:
Value x1: (N5gtsamPose2E) (-3.17392454561e-18, 5.21439530413e-19, 2.17083852905e-20)
Value x2: (N5gtsamPose2E) (5, 7.60341342619e-19, 1.73447953293e-20)
Value x3: (N5gtsamPose2E) (10.0000000015, -4.40576430129e-09, -1.5707963267)
Value x4: (N5gtsamPose2E) (10.0000000014, -5.000000003139, 3.14159265352)
Value x5: (N5gtsamPose2E) (4.99999999784, -5.00000000264, 1.57079632663)
```

**Prior Factor**

**Odometry Factor**

**Loop Closure Factor**
Use GTSAM in Matlab

1. Build Factor Graph

```matlab
% 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

2. Noisy Initial Values

```matlab
% initial variable values for the optimization
initials = Values;
initials.insert(symbol('x', 1), Pose2(8.2, -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, -5.8, -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!

3. Optimize!

```matlab
% Use Gauss-Newton method optimizes the initial values
parameters = GaussNewtonParams;
parameters.setVerbosity('ERROR');

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

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

- Prior Factor
- Odometry Factor
- Loop Closure Factor
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.

\[ e = [x - m_x, y - m_y]^T \]
Write your own factor

Derived from a GTSAM NoiseModelFactor unary factor class

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

```
50    // 2D 'GPS' measurement noise model, 2-dim
51    noiseModel::Diagonal::shared_ptr gpsModel = noiseModel::Diagonal::Sigmas(Vector2(1.8, 1.8));
52
53    // Add the GPS factors
54    // note that there is NO prior factor needed at first pose, since GPS provides
55    // the global positions (and rotations given more than 1 GPS measurements)
56    graph.add(GPSPose2Factor(Symbol('x'), 1, Point2(0, 0), gpsModel));
57    graph.add(GPSPose2Factor(Symbol('x'), 2, Point2(5, 0), gpsModel));
58    graph.add(GPSPose2Factor(Symbol('x'), 3, Point2(10, 0), gpsModel));
```

Results

https://github.com/dongjing3309/gtsam-examples/blob/master/cpp/examples/Pose2GPSEExample.cpp

Noise model dimension should match error vector dimension
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: declare 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
Expression: Automatic Differentiation (AD)

- Recall that the hardest 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 \( \text{func}_a \) of \( x_1 \) and \( x_2 \), then calculate the \( \text{func}_b \) of \( \text{func}_a \) result and a constant \( c_1 \)

```cpp
// 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

functions.h

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

functions.cpp

```cpp
18 // **********************************************/
19 gtsam::Point2 projectPose2(const gtsam::Pose2& pose, gtsam::OptionalJacobian<2,3> H) {
20 // jacobian: left 2x2 identity + right 2x1 zero
21 if (H) H = (gtsam::Matrix23()) << 1.0, 0.0, 0.8, 0.0, 1.0, 0.8).finished();
22 // return translation
23 return gtsam::Point2(pose.x(), pose.y());
24 }
```

expressions.h

```cpp
20 // expression project Pose2 to Point2
21 inline gtsam::Point2_projectPose2(const gtsam::Pose2& pose) {
22 return gtsam::Point2(&projectPose2, pose);
23 }
```

Pose2GPSExpressionExample.cpp

```cpp
59 // Add the GPS factors, by composing expressions
60 // note that there is no prior factor needed at first pose, since GPS provides
61 // the global positions (and rotations given more than 1 GPS measurements)
62 graph.addExpressionFactor(projectPose2_x1_, Point2(0, 0), gpsModel);
63 graph.addExpressionFactor(projectPose2_x2_, Point2(5, 0), gpsModel);
64 graph.addExpressionFactor(projectPose2_x3_, Point2(15, 0), gpsModel);
```

Design your cost function as usual

Convert your cost function as expression

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?

**LieGroup**: GTSAM optimizable and can use GTSAM Lie-group-only utils like `BetweenFactor`
Functions needed: `Identity`, `Logmap`, `Expmap`, `Compose`, `Between`, `Inverse`

**Manifold**: GTSAM optimizable classes
Functions needed: `dimension`, `GetDimension`, `Local`, `Retract`

**Testble**: Basic GTSAM classes
Functions needed: `Equal`, `Print`
gtsam::traits example

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

```cpp
namespace gtsamexamples {

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

  // convenience constructor
  Point2c(double x1, double y1) : x(x1), y(y1) {}
};
}
```

- 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)

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

template<>
struct traits<gtsamexamples::Point2c> {
```
# Functions as Testable

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

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

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

/**
 * Manifold
 */

// use enum dimension
enum { dimension = 2 }; //_TypeDefs needed

#define gtsam::Point2c ManifoldType;
#endif

typedef Eigen::Matrix<double, dimension, 1> TangentVector;

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

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

# Functions as Manifold

---

**Georgia Institute for Robotics and Intelligent Machines**
gtsam::traits example

namespace gtsamexamples {

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

    // convenience constructor
    Point2c(double x1, double y1) : x(x1), y(y1) {}

    // 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

```}

```c++
40  // first state prior noise model (covariance matrix)
41  noiseModel::Diagonal::shared_ptr priorModel = noiseModel::Diagonal::Sigmas(Vector2(0.2, 0.2));
42
43  // add prior factor on first state (at origin)
44  graph.add(PriorFactor<Point2>(Symbol('x'), 1, Point2(0, 0), priorModel));
45
46  // odometry measurement noise model (covariance matrix)
47  noiseModel::Diagonal::shared_ptr odomModel = noiseModel::Diagonal::Sigmas(Vector2(0.5, 0.5));
48
49  // Add odometry factors
50  // Create odometry (Between) factors between consecutive point2c
51  graph.add(BetweenFactor<Point2>(Symbol('x'), 1, Symbol('x'), 2, Point2c(2, 2), odomModel));
52  graph.add(BetweenFactor<Point2>(Symbol('x'), 2, Symbol('x'), 3, Point2c(2, 2), odomModel));
53  graph.add(BetweenFactor<Point2>(Symbol('x'), 3, Symbol('x'), 4, Point2c(2, 2), odomModel));
54  graph.add(BetweenFactor<Point2>(Symbol('x'), 4, Symbol('x'), 5, Point2c(2, 2), odomModel));
55
56  // print factor graph
57  graph.print("\nFactor Graph:\n");
58
59
60  // initial variable values for the optimization
61  // add random noise from ground truth values
62  Values Initials;
63  initials.insert(Symbol('x'), 1, Point2c(0.2, -0.3));
64  initials.insert(Symbol('x'), 2, Point2c(2.1, 0.9));
65  initials.insert(Symbol('x'), 3, Point2c(3.9, -0.1));
66  initials.insert(Symbol('x'), 4, Point2c(5.9, -0.3));
67  initials.insert(Symbol('x'), 5, Point2c(8.2, 0.1));
68
69  // print initial values
70  initials.print("\nInitial Values:\n");
```

```
Final Result:
Values with 5 values:
Value x1: (N13gtsamexamples7Point2cE) (4.5777435178e-33, 9.1554876356e-33)
Value x2: (N13gtsamexamples7Point2cE) (2, 7.3955709645e-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 Distributed 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


Visual-Inertial Odometry

https://youtu.be/CsJkci5lfco

Monocular Visual-Inertial Odometry (No Loop-Closures)

https://youtu.be/CsJkci5lfco
Structure from Motion (SfM)

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

Structure from Motion (SfM)

https://youtu.be/BgLlLlsKWzI
Multi-Robot SLAM

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

Multi-Robot SLAM

https://youtu.be/m_bLSdsT2kg
Dense Multi-View Stereo and Optical Flow

- Similar 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 $P$. 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.
Dense Multi-View Stereo and Optical Flow

https://youtu.be/2A7IOipPNBA
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).

Motion Planning

https://youtu.be/mVA8qhGf7So
Acknowledgement

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...