06.29 study

Date Created	@June 28, 2025 10:26 PM

Pose Graph Optimization By GTSAM

예제 코드

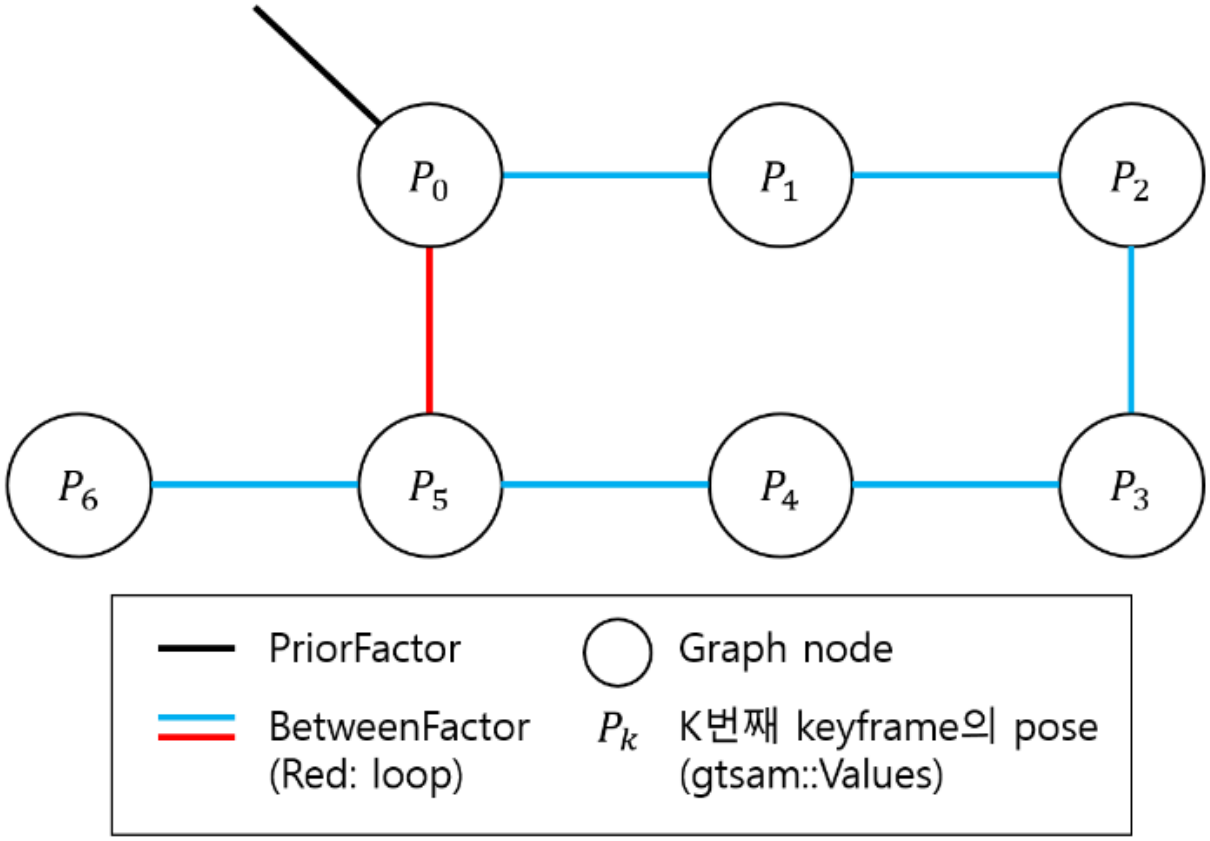
```
////// GTSAM headers
#include <gtsam/geometry/Rot3.h>
#include <gtsam/geometry/Point3.h>
#include <gtsam/geometry/Pose3.h>
#include <gtsam/slam/PriorFactor.h>
#include <gtsam/slam/BetweenFactor.h>
#include <gtsam/nonlinear/NonlinearFactorGraph.h>
#include <gtsam/nonlinear/LevenbergMarquardtOptimizer.h>
#include <gtsam/nonlinear/Values.h>
#include <gtsam/nonlinear/ISAM2.h>

using namespace std;

////// GTSAM variables
shared_ptr<gtsam::ISAM2> m_isam_handler = nullptr;
gtsam::NonlinearFactorGraph m_gtsam_graph;
gtsam::Values m_init_esti; // initial estimation (초기 위치 추정치)
gtsam::Values m_corrected_esti; //Graph optimized된 보정된 위치 추정치
int m_keyframe_index = 0;

////// GTSAM init
gtsam::ISAM2Params isam_params_;
isam_params_.relinearizeThreshold = 0.01;
isam_params_.relinearizeSkip = 1;
m_isam_handler = std::make_shared<gtsam::ISAM2>(isam_params_);
```

```
////// Odometry callback function
void odometry_callback_function(current_odometry) //실제 함수 아님, pseudo cod
{
 if (!initialized) // 최초 1회만 odometry를 Priorfactor로 추가
 {
  gtsam::noiseModel::Diagonal::shared_ptr prior_noise = gtsam::noiseModel::Dia
  // for the first odometry, priorfactor
  m_gtsam_graph.add(gtsam::PriorFactor<gtsam::Pose3>(0, odometry_to_gtsam
  m_init_esti.insert(m_current_keyframe_idx, odometry_to_gtsam_pose(current_o
  m_keyframe_index++;
  initialized = true;
 }
 else //그 이후 odometry callback 마다
 {
  if (if_keyframe_or_not(current_odometry)) //keyframe인지 검사하고 keyframe이
  {
   ///// 1. keyframe사이의 pose 변화를 graph에 추가
   gtsam::noiseModel::Diagonal::shared_ptr odom_noise = gtsam::noiseModel::D
   gtsam::Pose3 pose_from = odometry_to_gtsam_pose(last_odometry);
   gtsam::Pose3 pose_to = odometry_to_gtsam_pose(current_odometry);
   // 직전, 현재 keyframe odometry 사이의 odometry 변화값을 BetweenFactor로 그
   m_gtsam_graph.add(gtsam::BetweenFactor<gtsam::Pose3>(m_keyframe_ind
   m_init_esti.insert(m_keyframe_index, pose_to);
   m_keyframe_index++;
   last_odometry = current_odometry; //다음 iteration을 위해 직전 odometry 저장

   ///// 2. loop closing factor
   bool if_loop_closed = false;
   // 과거의 keyframe들과 현재 keyframe을 비교해서, loop closing이
   // 일어날 수 있을 가능성이 있는지 파악 (예: 일정 거리 이내에 있으나 시간이 일정 시간 이
   if (if_loop_candidate_or_not(current_odometry))
   {
    //가장 loop 가능성이 높은 keyframe 반환
    the_most_loop_likely_keyframe = get_the_most_loop_likely_keyframe(curren
    //현재 keyframe, loop 가능성 높은 keyframe 사이를 매칭해서 pose 변환 반환 (e.g
```

```
    loop_match_result = loop_matching(current_odometry, the_most_loop_likely

    // 실제로 loop-closed 되었으면
    if (loop_match_result.loop_closed)
    {
      noise = loop_match_result.noise;
      loop_pose_tf = loop_match_result.pose_transformation;
      past_pose = the_most_loop_likely_keyframe.pose;
      past_pose_index = the_most_loop_likely_keyframe.pose.index;
      current_odometry_index = m_keyframe_index-1; // becaus of ++ from the a

      // 현재 keyframe과 loop-closed 된 keyframe간의
      // pose 변화만큼을 graph에 BetweenFactor로 추가
      gtsam::noiseModel::Diagonal::shared_ptr loop_noise = gtsam::noiseModel:
      gtsam::Pose3 pose_from = odometry_to_gtsam_pose(loop_pose_tf * currer
      gtsam::Pose3 pose_to = odometry_to_gtsam_pose(past_pose);
      m_gtsam_graph.add(gtsam::BetweenFactor<gtsam::Pose3>(current_odom
      if_loop_closed = true;
    }
  }

  ///// 3. Optimize
  //m_corrected_esti = gtsam::LevenbergMarquardtOptimizer(m_gtsam_graph,
  m_isam_handler->update(m_gtsam_graph, m_init_esti);
  m_isam_handler->update();
  if (if_loop_closed)
  {
    m_isam_handler->update();
    m_isam_handler->update();
    m_isam_handler->update();
  }
  m_gtsam_graph.resize(0);
  m_init_esti.clear();
  // 보정된 위치 추정치
  m_corrected_esti = m_isam_handler->calculateEstimate();
}
```

```
  }
}
```

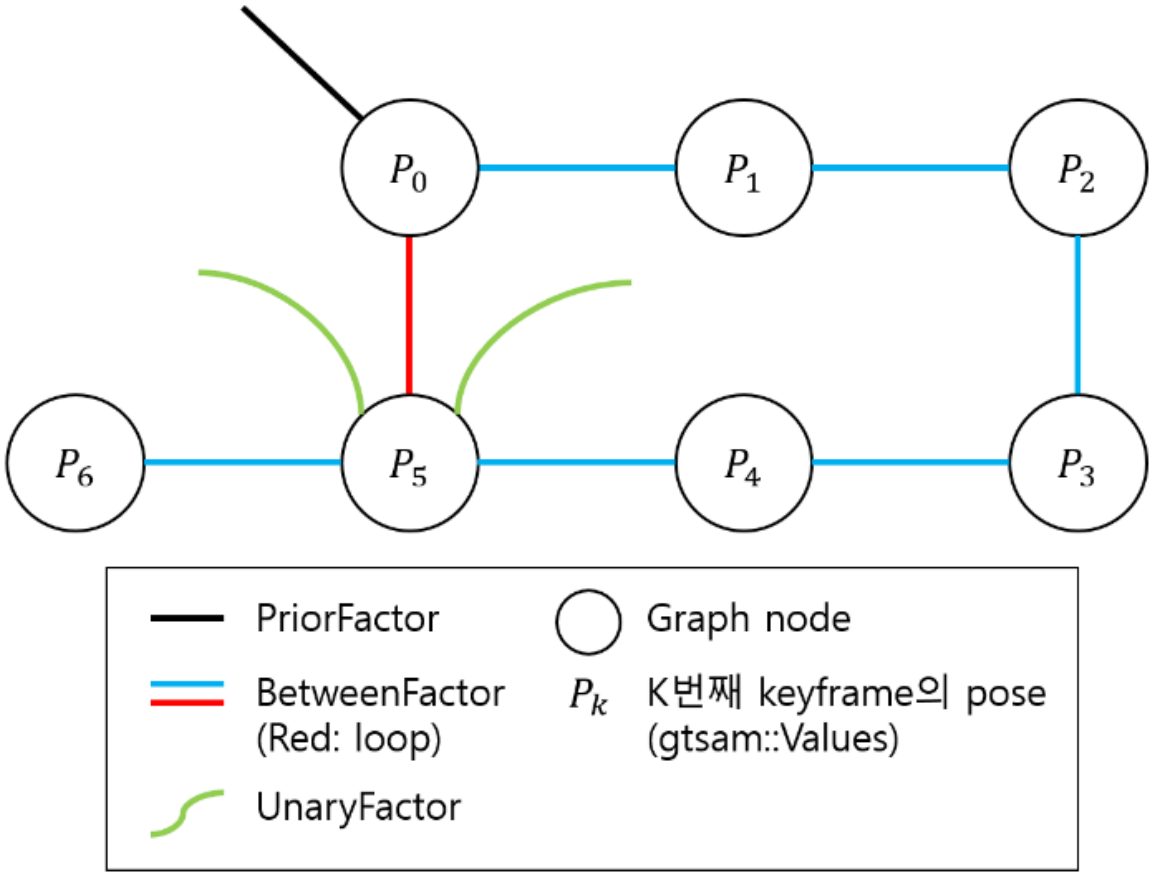
1. 최초 odometry는 PriorFactor로 graph에 추가
2. keyframe 사이의 pose 변화량을 BetweenFactor로 graph에 추가한다.
    - keyframe 계산 없이 모든 odometry를 graph에 추가하면 연산량도 매우 크고 오히려 redundancy가 accuracy를 해친다 → 불필요한 중복이 오히려 정확도에 악영향을 끼침
    - 예를 들어, 로봇이 거의 움직이지 않고 가만히 서 있을 때 odometry 센서는 미세한 노이즈만 계속 출력.
    - 이 쓸모없는 정보를 계속 그래프에 추가하면 “움직이지 않았따”는 중요한 정보 대신 “노이즈 만큼 미세하게 움직였따”는 잘못된 정보가 수없이 쌓이게 되어 오히려 전체 지도의 정확도를 떨어뜨리는 요인이 됨
3. 현재 keyframe과 과거 keyframes 사이에 loop-closing을 검사 및 계산해서 BetweenFactor로 graph에 추가한다.
4. Graph를 optimize 한다



Factor vs Values in GTSAM

- 코드를 보면, graph에 priorfactor, betweenfactor로 odometry의 변화량, loop-closure constraint등을 더해줬는데, **gtsam::Values(** `m_init_esti` **)**에 한번 더 insert 해준다.
- 게다가 loop-closing 계산 뒤 BetweenFactor를 add해주는 곳에서는 Values에 아무런 값도 insert하지 않는다.
- **graph (Factor)는 연결**만 나타내고 있다고 생각하면 편하고, **Values는 변하는 값**이라고 생각하면 된다.
    - Factor는 node 사이의 joint probabilty, 즉, **확률 값**을 나타내고, Values는 말 그대로 **위치 추정치** 값이다.

- 그러니까, graph (Factor)만 가지고 있어도 연결도 나타내고 안에 값도 들어있으니 optimize 할 때마다 값도 알아서 변하고 우리는 변한 값만 출력해서 받아 쓰면 되지 않는가?
- 다음 그림을 보면, 조금 더 이해가 쉬움



  - 만약 내가 헷갈렸던 것처럼 Factor만 가지고 있다고 생각해보자.
    원래 1절에서 등장한 그림만 봤을때는 아무런 문제가 없을 것이다. Odometry로만 graph가 구성되어있으니 loop closing이 일어나든 말든 node 자체의 값이 변해버리면 되지 않나? 생각했던 것인데,
  - 아래 그림처럼 UnaryFactor가 추가되었다고 생각해보자 (예: GPS 센서 값 등).
    Loop closing이 일어나서 P5 노드의 값 자체가 바뀌어버리면 UnaryFactor가 의미하는 값이 뭔가 이상해진다.
  - 분명 loop closing이 일어나기 전의 P5 노드의 값에 대한 관계를 그래프에 추가한 것인데, P5노드의 값이 바뀌었으니 UnaryFactor 값도 그에 맞게 바뀌어야 하는 건가?

근데 GPS 센서 측정 값인데 그게 바뀌어도 되나?

- 따라서 loop closing이 일어나든 말든, UnaryFactor가 추가되든 말든 graph는 graph대로 연결 관계에 대한 값을 고정되게 가지고 있고, 이와 별개로 위치 추정치는 따로 변수로 (Values) 가지고 있으면 이런 문제가 없다.

gtsam::ISAM2.update vs gtsam::LevenbergMarquardtOptimizer

- 코드에서 optimize하는 부분을 보면 주석된 부분이 있다.
- 다시 잘 살펴 보면, 한 줄로 해결할 수 있을 것 같은데 굳이 여러 줄로 나누어서 optimize하고 graph랑 Values 초기화하고, 보정된 값을 획득한다.
- 실제로 한 줄짜리 LM Optimizer로 optimize해도 동일한 결과를 획득한다. => **하지만 연산 시간이 엄청 길어진다. Graph가 점점 증가하면 할수록.**
- **GTSAM은 Georgia Tech SAM인데, 논문에서 정식 명칙은 Incremental SAM이다.**
- 즉, 한 줄 짜리 LM Optimizer는 그래프 전체를 받아와서 전체에 대해서 nonlinear optimization을 수행한다. 연산량이 많고, 점점 많아질 수밖에 없다.
- ISAM update는 incremental하게 graph의 변화된 부분만 파악해서 update 및 optimization을 수행한다.

```
//// 이 한 줄이랑
{
 m_corrected_esti = gtsam::LevenbergMarquardtOptimizer(m_gtsam_graph,
}
//// 이 여러 줄이랑 같은 역할임
{
 m_isam_handler->update(m_gtsam_graph, m_init_esti);
 m_gtsam_graph.resize(0);
 m_init_esti.clear();
 m_corrected_esti = m_isam_handler->calculateEstimate();
}
```

4. `gtsam::ISAM2.update(graph, initial_estimation)` vs `gtsam::ISAM2.update()`

- 위 3절과 마찬가지로 코드에 이상한 점이 있다. gtsam::ISAM2.update(graph, esti)로 한번 update 했는데 (update에서 optimize도 함께 이루어짐), 여러 번 반복해서, 심지어 함수 인자도 없이 update한다.

```
m_isam_handler->update(m_gtsam_graph, m_init_esti); //한 번 했는데
m_isam_handler->update(); //여러 번 인자도 없이 반복한다.
if (if_loop_closed) //심지어 loop-closing 되면
{
  m_isam_handler->update(); //인자도 없이 또 반복한다.
  m_isam_handler->update(); //인자도 없이 또또 반복한다.
  m_isam_handler->update(); //인자도 없이 또또또 반복한다.
}
```

- 즉, 그냥 여러 번 호출하면 여러 번 iteration으로 optimize하는 것과 같으므로 원하는 만큼 해라! 이거임.
- 실제로 ISAM2 헤더파일의 update 부분을 보면, 아무 인자 없이 update 함수를 호출해도 다 default 인자로 넘겨주게 되어있고, ISAM2 소스파일의 update 부분을 보면, 인자로 전달 받은 graph의 변화된 부분만 추적해서 알아서 optimize한다.
- 그러면 굳이 코드에서 m_gtsam_graph.resize(0) 할 필요 없지 않나? => 없어도 되지만 update 함수 내에서 “graph의 변화된 부분을 추적”하는 연산을 최소화하기 위해 필요하다.
- 그러면 update를 매우 많이 호출하면 완전 빨리 수렴하고 값도 정확해 지는거 아님? => 이것은 경험에 의한 것... 실제로 LIO-SAM 코드를 보면 5회만 실행하는데, 저자에 따르면 그게 가장 좋았다고 한다..
- 너무 많이 호출하면 시간도 아깝고 뭔가 발산하는 양상을 띈다.

시행착오

- mt_lam_sw
  - 기준맵이 없는 구간에서는 기준맵 기반의 localization을 진행하다가 기준맵이 없는 구간에 진입해서 mms map 생성 후 다시 기준맵 있는 구간으로 돌아왔을 때 기준맵과

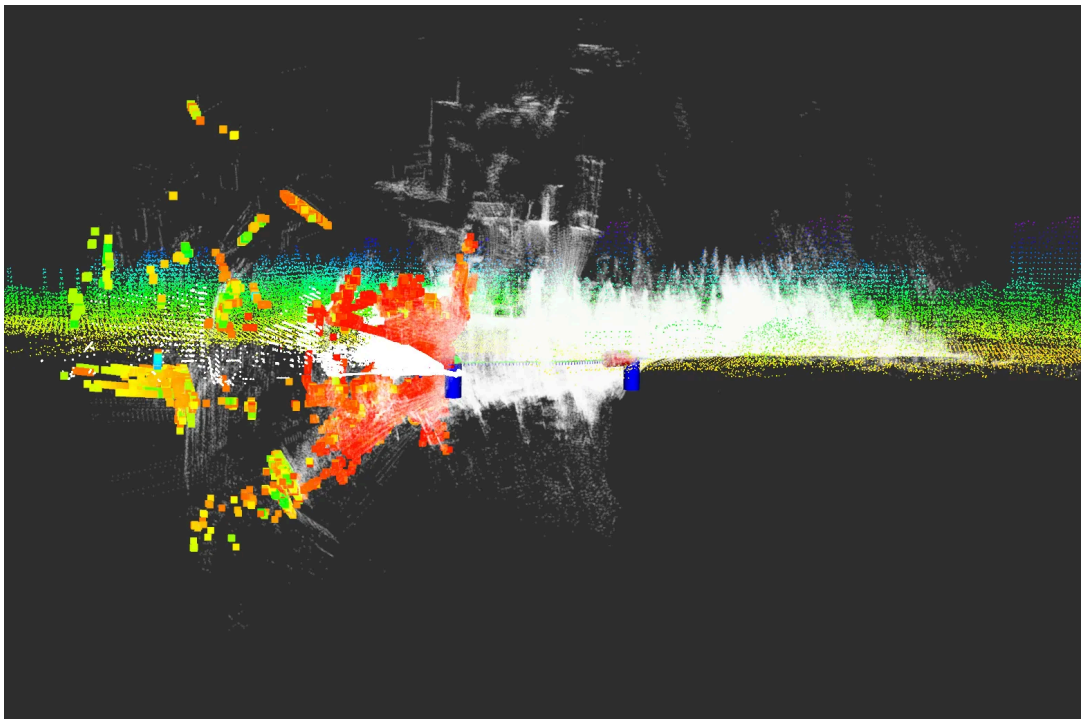
align이 잘 되어야 함

- 하지만 실제로는 z error가 생겨서 GTSAM을 이용하여 Pose Graph Optimization을 시도
- GTSAM Factor를 추가할 때 noise를 설정하게 되어있음

```
void initNoises( void )
{
    gtsam::Vector priorNoiseVector6(6);
    priorNoiseVector6 << 1e-12, 1e-12, 1e-12, 1e-12, 1e-12, 1e-12;
    priorNoise = noiseModel::Diagonal::Variances(priorNoiseVector6);

    gtsam::Vector odomNoiseVector6(6);
    odomNoiseVector6 << 1e-12, 1e-12, 1e-12, 1e-12, 1e-12, 1e-12;
    odomNoise = noiseModel::Diagonal::Variances(odomNoiseVector6);

    double loopNoiseScore = 1e-12;
    gtsam::Vector robustNoiseVector6(6);
    robustNoiseVector6 << loopNoiseScore, loopNoiseScore, loopNoiseSc
    robustLoopNoise = gtsam::noiseModel::Robust::Create(
        gtsam::noiseModel::mEstimator::Cauchy::Create(1),
        gtsam::noiseModel::Diagonal::Variances(robustNoiseVector6)
    );

    double bigNoiseTolerentToXY = 1.0;
    double gpsAltitudeNoiseScore = 1.0;
    gtsam::Vector robustNoiseVector3(3);
    robustNoiseVector3 << bigNoiseTolerentToXY, bigNoiseTolerentToXY,
    robustGPSNoise = gtsam::noiseModel::Robust::Create(
        gtsam::noiseModel::mEstimator::Cauchy::Create(1.0),
        gtsam::noiseModel::Diagonal::Variances(robustNoiseVector3)
    );
}
```

  - noise를 통해 factor의 신뢰도 설정 가능

- 이 값이 매우 예민하여 조금만 바꿔도 전체적인 최적화가 틀어져 버릴 수 있음



- z error를 해소하기 위해 GPS의 의 z를 GT 삼아서 최적화 하기 위해 GPS factor의 z noise를 매우 낮게 설정하면 나머지(translation xy, rotation)값이 틀어져 버릴 수 있음
- 여러가지 시도 해본 결과 noise는 해당 센서의 실제 오차 값을 default로 설정 후 조금씩 실험적으로 바꾸는걸 추천
- 또한 하나의 factor의 noise를 너무 작게하거나 너무 크게 하면 전체 그래프 최적화가 틀어져 버릴 수 있음
- research 결과 z값만 optimized pose로 사용하는건 의미가 없고 Z축 에러는 다른 축의 에러와 복합적으로 얽혀있기 때문에, Z만 따로 교정할 수는 없습니다. PGO는 6-DoF 전체를 함께 최적화해야 의미가 있습니다.
- 사실 ICP odometry이든, GPS factor든 어느 하나에 무한 신뢰를 주는 것은 잘못된 방법
- GPS quality가 좋은 구간이 있고, ICP odometry가 좋은 구간이 있기 때문에 상황에 맞게 adaptive하게 노이즈를 설정하는 것이 가장 좋은 방법

- LIO-SAM 코드를 보면 gps factor의 noise를 설정할 때 현재 gps topic 의 position_covariance를 noise로 넣어줘서 현재 gps quality에 맞게 noise 를 설정해줌