컴퓨터비전특론 Bonus assignment

전자전기공학부 석사과정 윤시원 (학번 : 2023000853)

Bonus Assignment of ELEC812 Advanced Computer Vision

Prof. Kyuman Lee (Kyungpook National University)

Spring 2023, Due: June 6

Literature review on given *Survey Paper*

동시 위치추정 및 지도작성(SLAM)은 환경의 모델(지도)과 그 안에서 움직이는 로봇의 상태를 동시에 추정하는 것을 의미한다. 해당 survey paper에서는 2016년 당시 SLAM 연구 현황과 향후 발전 방향을 고려한다. 우선, SLAM에 대한 방법론을 제시한다. 그런 다음, 장기간 매핑에서의 견고성과 확장성, 매핑을 위한 메트릭 및 의미론적 표현, 이론적인 성능 보장, 동적 SLAM과 탐사, 그리고 기타 새로운 분야를 포함한 관련 연구를 검토한다. 관련 키워드로는 팩터 그래프, 위치추정, 지도작성, 최대 사후 확률 추정, 지각, 로봇, 센싱, 동시 위치추정 및 지도작성 (SLAM) 이 있다.

SLAM (Simultaneous Localization and Mapping)은 로봇에 장착된 센서를 이용하여 환경을 인지하고, 동시에 로봇의 상태를 추정한다. 간단한 경우 로봇의 상태는 자세(position, orientation)로 설명 가능하다. 하지만 로봇의 속도, 센서의 편향, 보정 매개변수 등과 같은 다른 요소들도 상태에 포함될 수 있다.

SLAM의 목적은 로봇의 위치와 자세를 추정하는 동시에, 로봇이 인식한 환경의 모델을 구축하는 것이다. 로봇이 이동하면서 주변 환경 정보를 수집하고, 그 정보를 기반으로 로봇의 상태를 업데이트하고 환경의 모델을 갱신하는 과정 등이 해당한다. 이를 통해 로봇은 자신의 위치를 추적하면서 동시에 주변 환경을 인식하고, 이후에는 이 정보를 활용하여 자율적으로 탐사하거나 목표를 달성하는 등의 작업을 수행할 수 있게 된다. mapping 의 경우 이러한 일련의 과정을 거쳐 로봇이 작동하는 환경을 나타내는 지표들(예: 랜드마크의 위치, 장애물)을 표현한 것이다.

해당 논문에서 중요하게 다루는 주제 중 관심이 있었던 것은 ‘Loop Closure’ 였다. odometry 추정을 통한 mapping 으로 주변 환경에 대해 지도 생성을 하는 경우 시간에 따라 누적 오차가 발생한다. 이 경우 먼저 지도를 그리고, 가본 곳(알려진 지역)을 재방문할 때 loop closing 을 해서 오차를 보정하고 지도를 재설정할 수 있다. 사전에 지도가 제공되지 않는 상황에서 지도를 구축해야만 하는 시나리오에서 해당 방식은 매우 유용할 것이다.

이 서베이 논문에서는 loop closing 을 하지 않을 경우 단점에 대해서도 서술하고 있다. 오도메트리를 수행하면서 loop closure 를 무시하는 로봇은 세계를 "무한한 복도"로 해석하게 된다. 이 경우 실제로는 가까운 두 지점이 odometry map 상으로는 먼 거리로 표현될 수 있는 것이다. 로봇은 계속해서 새로운 영역을 탐색하는 것처럼 보이는데, loop closure event 는 로봇에게 이 "복도"가 자기 자신이 이전에 가본 장소와 교점을 가진다는 사실을 알려 준다.

즉, loop closure 를 함으로써 로봇의 실제 환경에 대한 이해도가 매우 높아지게 되며, 환경에 대해 올바르게 이해하면 특정 지점간 최단 거리를 찾는 등의 작업이 가능해진다. loop closure 를 하지 않게 되면, 환경에 대해 올바르게 인식하지 못할 가능성이 높아지며, 실제로는 가까운 두 지점을 매우 먼 거리로 잘못 해석할 수 있다.

CV method/algorithm

해당 survey paper 에서 다룬 자료 중 이번 assignment 에서 다룰 자료는 다음과 같다.

[150] Y. Latif, C. Cadena, and J. Neira, “Robust loop closing over time for pose graph slam,” Int. J. Robot. Res., vol. 32, no. 14, pp. 1611–1626, 2013.

코드는 다음 인터넷 사이트에 공개되어 있다.

https://github.com/ylatif/rrr

위 논문은 시간에 따라 누적되는 data에 대한 강력한 loop closing에 대해 다룬다. 이 논문은 loop closing의 정확성과 신뢰성을 향상시키기 위한 방법을 제안한다. 시간에 따라 loop closing을 점진적으로 강화하는 방식으로 구축하고, 이를 통해 정확한 맵 및 경로 추정을 달성하는 것이다.

논문 abstract 에 따르면 해당 연구의 독창성은 RRR 알고리즘을 사용하여 시각 또는 레이저와 같은 다양한 odometry system 과 다양한 frontend loop-closing 기술에 대해 실험하는 것에 있다. 실험에는 효율적인 그래프 최적화 프레임워크인 g2o를 백엔드로 사용한다. (깃허브 코드에서도 g2o 를 먼저 install 해야 한다고 안내하고 있다.) 이를 통해 알고리즘은 최적화 기술을 활용하여 환경 모델의 정확성을 향상시킨다.

RRR 알고리즘에 대해 간략히 설명하면 다음과 같다.

1. 합의 기반 장소 인식: RRR 알고리즘은 여러 센서 및 frontend loop closing 기술에서 얻은 정보를 모두 고려한다. 이를 통해 다양한 인식 결과를 종합적으로 분석하고 올바른 결정을 내릴 수 있다.
2. 점진적 운영 확장: RRR 알고리즘은 점진적으로 환경을 구축하는 SLAM 시스템에서도 사용할 수 있다. 이는 지속적으로 새로운 데이터를 누적 통합하고, 과거의 잘못된 loop closure를 수정하고, 환경 모델을 개선하는 데 도움을 준다.

3. 다중 세션 및 관련 시나리오 처리: RRR 알고리즘은 여러 세션에서 수집된 데이터나 공간적 관련에 구애받지 않는 시나리오에서도 공통적으로 처리할 수 있다. 이는 다양한 환경에서 일관된 결과를 얻을 수 있도록 하는 robust 한 성질을 가짐을 의미한다.

RRR 알고리즘은 다양한 SLAM 시나리오에서의 장소 인식에 대한 robustness를 제공하며, 여러 실험과 비교를 통해 그 효과를 입증한다.

Code Review at the *function level*

다음 코드는 RRR 을 적용해 어떻게 loop closing 최적화를 하는지에 대한 코드이다.

```
#include "include/RRR.hpp"
#include "g2o/types/slam2d/edge_se2.h"
#include "g2o/types/slam2d/vertex_se2.h"
#include "g2o/types/slam3d/edge_se3.h"
#include "g2o/types/slam3d/vertex_se3.h"
// RRR과 관련된 typedef를 정의
typedef RRR<G2O_Interface<g2o::VertexSE2, g2o::EdgeSE2>> RRR_2D_G2O;
typedef RRR<G2O_Interface<g2o::VertexSE3, g2o::EdgeSE3>> RRR_3D_G2O;
// main : 입력 파일 읽고 g2o optimizer 초기화
// argv : graph file
int main(int argc, char **argv)
{
  // 생략 : 입력 파일 읽는 부분
  ...
  // g2o optimizer 초기화
  g2o::SparseOptimizer optimizer;
  auto linearSolver = g2o::make_unique<LinearSolverEigen<BlockSolverX::PoseMatrixType>
>();
  linearSolver->setBlockOrdering(false);
  auto blockSolver = g2o::make_unique<BlockSolverX>(std::move(linearSolver));
  g2o::OptimizationAlgorithmGaussNewton *solverGauss = new g2o::OptimizationAlgorithmG
aussNewton(std::move(blockSolver));
  optimizer.setAlgorithm(solverGauss);
  //optimizer 에 graph file load
  optimizer.load(argv[1]);
  // RRR 객체 생성, optimizer 에 대한 pointer 전달
  RRR_2D_G2O rrr(clusteringThreshold, nIter);
  rrr.setOptimizer(&optimizer);
  rrr.robustify();
  // loop closure 를 이전에 제거하지 않았다면, 지금 제거.
  rrr.removeIncorrectLoops();
  /* 위 알고리즘을 통해 correct loop closure 에 대해서만 graph 작성, incorrect 일 경우
  위 방법 중 하나를 통해 제거한다.
  */
  std::cout << std::endl;
  std::cout << "Output written to rrr-solved.g2o" << std::endl;
  // correct loop closure 만 포함된 그래프를 출력.
  rrr.write("rrr-solved.g2o");
  //optimizer.save("g2o-saved.g2o");
  return 0;
}
```