programmers
자율주행 데브코스 3기

Team 스리슬램

SLAM PoC Project 회고

2022.06.30.목

발표자 한은기(PM)

Contents

Team 스리슬램 SLAM PoC Project 회고

Part 1

Project Goal

- 고객사 최초 제시 목표
- 갑작스러운 요구사항
- 최종 상세 목표

Part 2

Strategy

- 역할 분배 및 일정 계획
- 프로젝트 수행 전략
- 기준 SLAM 선정

Part 3

Process

- 구현 내용 요약
- 구현 내용 상세
- 개발 결과

Part 4

Review

- 팀원 회고 종합
- 공유 자료

Contents

Team 스리슬램 SLAM PoC Project 회고

고객사 최초 제시 목표
: 폭스바겐 社

모듈러 아키텍처를 가진 VSLAM 프레임워크 제작하기

우리는 VSLAM 엔지니어가 없습니다.

앞으로 채용하려고 하는데,
이를 위해서 미래에 사용할
VSLAM 프레임워크가 필요합니다.

PTAM의 코드를 한 번 본 적이 있는데
너무 예전 코드이고
아키텍쳐도 이상하더군요

1. 다양한 알고리즘을
원하는 대로 사용해보고 실험을 해볼 수 있는,
모듈러 아키텍쳐를 가진
VSLAM 프레임워크를 만들어 주세요

2. 이 프레임워크를
다른 유명한 오픈소스 슬램과 비교해서
성능 비교를 해주세요
(성능이 더 좋아야하는 건 아닙니다)

갑작스러운 요구사항 1 · 2
(feat. 고객님 이러시면 아니됩니다)

- ORB-SLAM을 포함하여
  오픈소스 SLAM 3개를 빌드하고
  profiler를 통해 성능을 비교해주세요.
- 단, 이 open-source SLAM들은
  CI 동작이 필요 없습니다.

- 개발한 모듈러 프레임워크와,
  [갑작스러운 요구사항1]에서 빌드한
  SLAM들을 모두 Xycar에서
  CI를 활성화주세요.
- KITTI Dataset을 사용해 위 SLAM들을
  Xycar에서 가동시켜주세요

Detailed Requirements

최종 상세 요구사항 (파트별)

Frontend Requirements

최소 5개의 실험 가능 기능 개발

Backend Requirements

최소 5개의 실험 가능 기능 개발

Loop Closure Requirements

On/Off 기능을 포함하여
최소 5개의 실험 가능 기능 개발

Image Processing Requirements

ORB, AKAZE를 포함해
최소 3개의 실험 가능 기능 개발

CI/CD에서 자동 빌드 지원

Stereo 카메라 지원

GitHub Repository
링크 공유

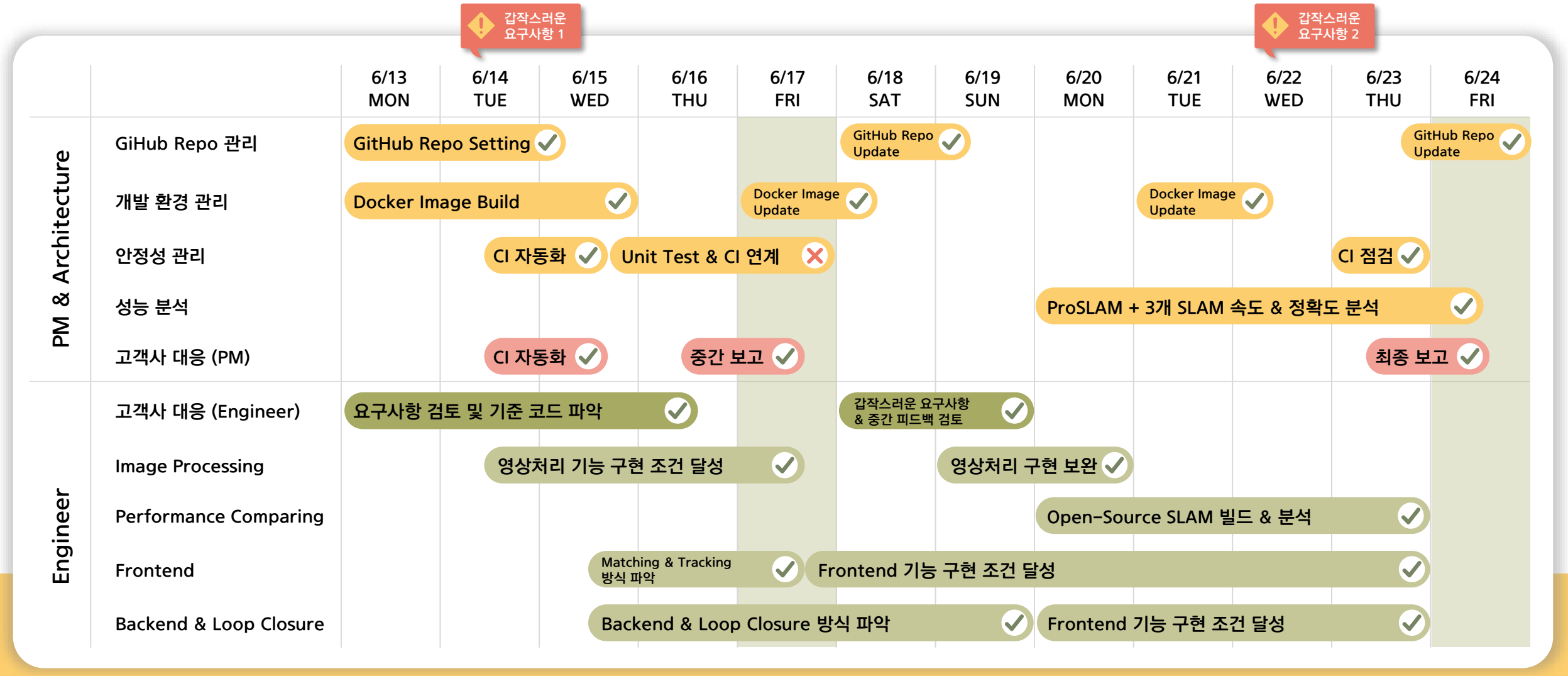
Contents

Team 스리슬램 SLAM PoC Project 회고

역할 배분

한은기	이창준	이현진	유희평
PM & Architecture	Image Processing	Frontend	Backend
■ 프로젝트 일정 및 수행 관리 ■ 고객사 미팅 및 발표 ■ GitHub Repository 관리 ■ 안정성: CI/CD, Unit Test ■ 성능 비교: Profiler 등 ■ Docker Image 제작 / 배포	■ 특징점 검출 및 기술자를 선택 구동하도록 모듈러 기능 / 구조 구현 ■ 다른 SLAM 빌드 및 Docker 이미지 만들기	■ ProSLAM의 Frontend 방식 파악 및 실험 가능 모듈 연구 ■ Matching 관련 기능을 선택 구동하도록 모듈러 기능 / 구조 구현	■ ProSLAM의 Backend 방식 및 Loop Closure 기능 파악 및 실험 가능 모듈 연구 ■ Bundle Adjustment, Aligner, Loop Closure 등 기능을 선택 구동하도록 모듈러 기능 / 구조 구현

일정 계획 with Gantt Chart



모듈러 아키텍처 설계 및 타 SLAM 간의 성능 비교를 위한 프로젝트 수행 전략

모듈러 아티텍처 구현

영상처리는 구현 형태가 비교적 선명

Frontend ~ Loop Closure 각각에서 실험 가능 부분을 찾아내기 위해 선행적으로 **코드와 이론 분석** 수행

모든 실험 가능 (모듈러 기능)은 **configuration 파일에서 수정**하여 구동할 수 있도록 구현할 것

다른 SLAM 빌드 / 비교

KITTI Dataset에서 수행가능한 **다른 SLAM 3개** 선정 (ORB SLAM 必)

Easy Profiler를 이용해 **속도 분석**

Evo를 이용해 Pose **정확도 분석**

개발 환경 및 팀워크 전략

통일된 환경에서 개발 진행 위하여 **Docker Image**를 만들어 배포하고 **지속적으로 업데이트** 수행

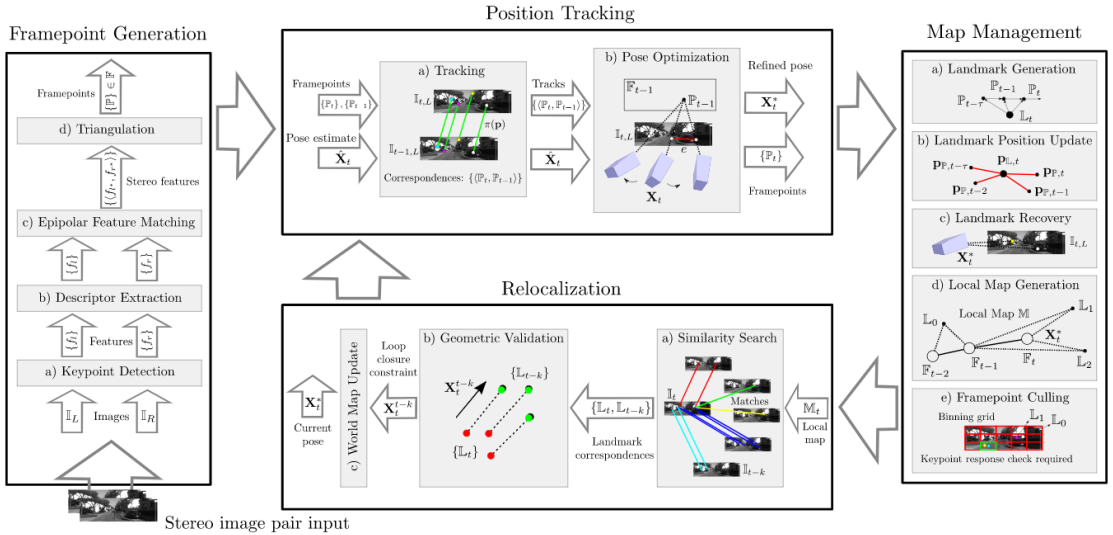
GitHub Actions를 이용하여 **CI 빌드 자동화** 구현

Notion Page를 이용하여 **개발 상황 및 미팅 결과, 각 이슈 공유**

Which model did we use?

기준이 된 SLAM 모델 선정
: ProSLAM

+ 구현과 관한 내용이 논문에 상세하게 기술됨
+ 공개된 소스코드에 자세한 주석이 포함됨
+ 하나의 Thread에서 모든 과정이 진행되어 Delay가 발생할 지라도, 복잡도는 낮음



ProSLAM 공식 GitLab (https://gitlab.com/srrg-software/srrg_proslam)

- Stereo 와 RGB-D Camera만을 지원하여 Monocular Camera 적용은 힘듦
- 모듈화가 잘 되어 있지 않음 ← UnitTest를 시도하다 발견…!

Contents

Team 스리슬램 SLAM PoC Project 회고

Options that we developed

모듈 별 사용 가능한 옵션들 목록 전체보기

Image Processing

- Feature Detector Type
  - ORB
  - AKAZE
  - KAZE
  - FAST
- Descriptor Type
  - ORB
  - AKAZE
  - KAZE
  - BRISK

Frontend

- Matching Type
  - Flan Based
  - Brute-Force
  - Brute-Force L1
  - Brute-Force Hamming
  - Brute-Force Hamming GLUT
  - Brute-Force Hamming SL2
- Homography or Not
  - Homography
  - Lowe's Ratio
- Homography Params
  - RANSAC
  - PROSAC
  - LMEDS

Backend

- Optimization Algorithm
  - Gauss-Newton
  - Levenberg
  - Dogleg
- Linear Solver Type
  - Sparse Cholesky decomposition
  - CSparse
  - Dense Cholesky decomposition

Loop Closure

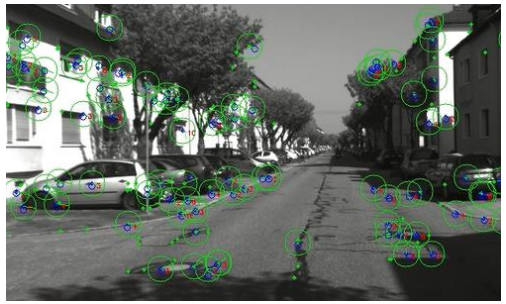
- Local Map Aligner
  - ICP
  - FAST-ICP
- Loop closure Params
  - error_delta_for_convergence
  - maximum_error_kernel
  - maximum_number_of_iterations
  - minimum_number_of_inliers
  - minimum_inlier_ratio
  - anderson_m
- Loop closure
  - On
  - Off

Image Processing 개발 결과
: 원하시는 대로 골라 조합해 쓰세요

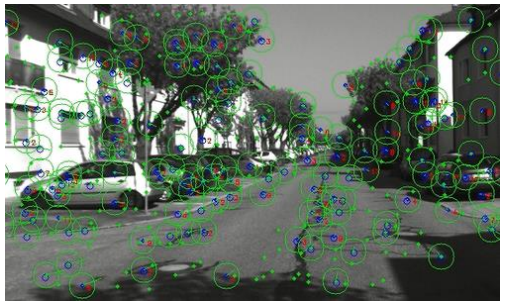
- **Feature Detector Type** 입력 받은 이미지로부터 feature을 추출하는 방식을 선택
  - ORB
  - AKAZE
  - KAZE
  - FAST

(default) ORB

AKAZE



KAZE



FAST

- **Descriptor Type** 추출한 feature을 기반으로 descriptor을 만드는 방식을 선택
  - ORB
  - AKAZE
  - KAZE
  - BRISK

Frontend 개발 결과
: 나의 짝꿍은 어디에

- **Matching Type** — Stereo 카메라를 사용하므로, 좌우 이미지에서 서로 대응하는 점을 찾기
  - Flan Based
  - Brute-Force
  - Brute-Force L1
  - Brute-Force Hamming
  - Brute-Force Hamming GLUT
  - Brute-Force Hamming SL2
- **OpenCV Match or Not** — OpenCV 방식을 쓸지, ProSLAM 방식을 쓸지 결정. True일 시에만 다른 선택이 유효
  - True
  - False
- **Homography or Not** — 양쪽 이미지에서 matching을 찾았다면 각 포인트 쌍에 대해 좋은 matching인지, 즉 잘 matching되었는지를 확인함.
  - Homography → OpenCV의 findHomography() 함수를 이용
  - Low's Ratio
- **Homography Params** — Homography를 사용할 시 선택할 수 있는 파라미터
  - RANSAC
  - PROSAC
  - LMEDS

  상세 설정 가능한 파라미터
  * maximum_ransac_reproject (RANSAC 재투영 허용 개수)
  * maximum_iters (Iteration 횟수)
  * maximum_confidence (정확도)

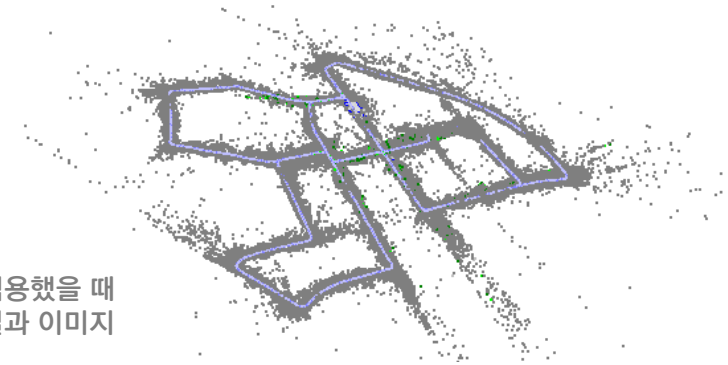
Backend & Loop Closure 개발 결과
: 여기만 10개… 그대는 욕심쟁이…

Optimization Algorithm — Pose Graph Optimization 알고리즘

- Gauss-Newton
- Levenberg
- Dogleg

Linear Solver Type — Solver (decomposition type) 을 선택

- Sparse Cholesky decomposition
- CSparse
- Dense Cholesky decomposition



FAST-ICP를 적용했을 때
align이 잘 된 결과 이미지

Loop closure — Pose Graph를 Optimization하는 알고리즘 선택

- On → ON 설정되어야 pose-graph optimization을 수행 & 아래 내용 선택 가능
- Off

Local Map Aligner — Pose Graph를 Optimization하는 알고리즘 선택

- ICP
- FAST-ICP → 최근 새로 제시된 ICP 방식

Loop closure Params — Pose Graph를 Optimization하는 알고리즘 선택

- error_delta_for_convergence
- maximum_error_kernel
- maximum_number_of_iterations
- minimum_number_of_inliers
- minimum_inlier_ratio
- anderson_m

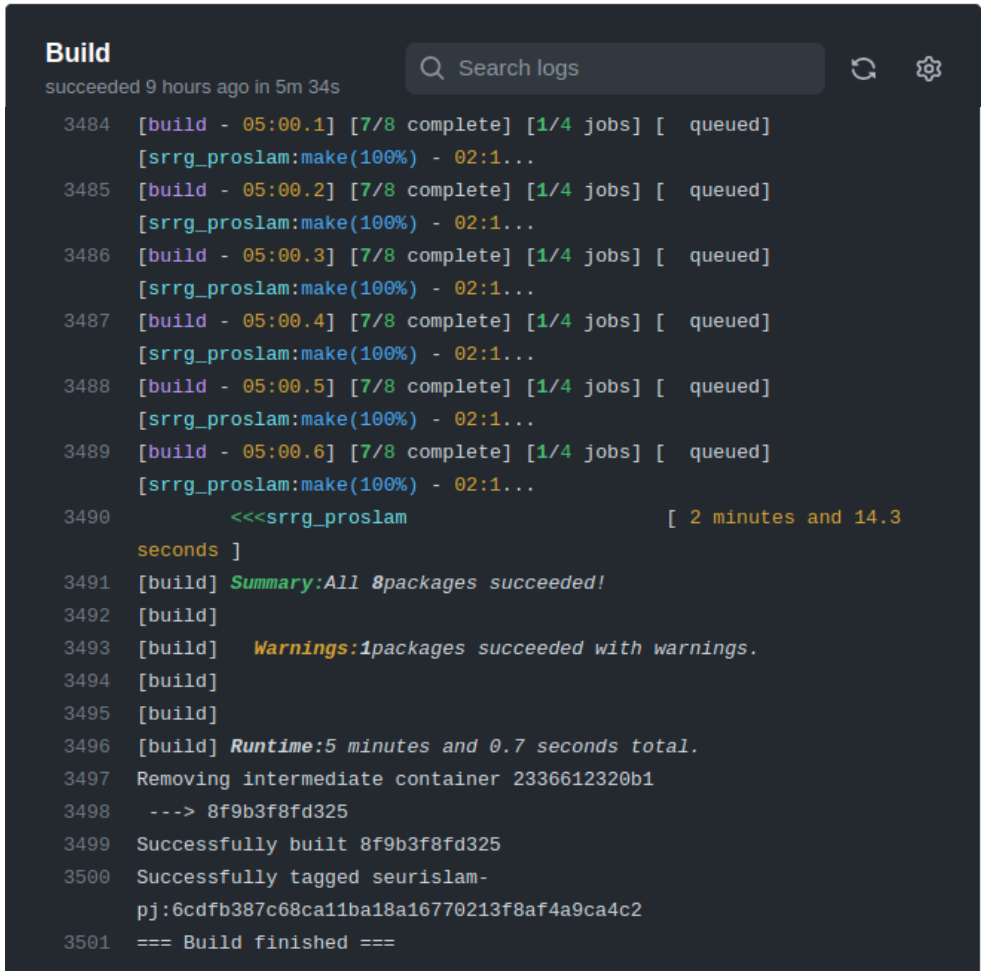
안정성은 CI, Unit Test
성능 평가는 Easy Profiler, Evo (Map)

Docker Container에서 개발

- 동일한 3rd Party Library Version, Directory 구조 공유
- Dataset 및 시스템 설정을 포함
- 자주 사용하는 명령어는 alias를 설정해 편리성 향상

GitHub에서 CI 자동화

- 각자 개발을 진행하므로 충돌 가능성 존재
- GitHub로(remote repo) Push시 자동 빌드 수행
- 충돌 여부 체크 및 코드 공유 기능



안정성은 CI, Unit Test
성능 평가는 Easy Profiler, Evo (Map)

Docker Container에서 개발

- 동일한 3rd Party Library Version, Directory 구조 공유
- Dataset 및 시스템 설정을 포함
- 자주 사용하는 명령어는 alias를 설정해 편리성 향상

GitHub에서 CI 자동화

- 각자 개발을 진행하므로 충돌 가능성 존재
- GitHub로(remote repo) Push시 자동 빌드 수행
- 충돌 여부 체크 및 코드 공유 기능

Fail과 Cancel로 업록진 메일함

안정성은 CI, Unit Test
성능 평가는 Easy Profiler, Evo (Map)

Local에서의 UnitTest

- 빌드를 할 때마다 Unit Test (Google Test) 수행
- 기준이 되는 SLAM의 구조 상 Unit Test를 구현하기에 매우 까다로운 조건이었음
  - 함수 Parameter로 넘겨주기,
    복잡한 상속 관계 …

GitHub에서 UnitTest 자동화

- Actions에서 CI로 빌드 직후 Unit Test를 진행하도록 설정
- 자동화는 성공했으나, Unit Test 자체가 성공적이지 못하여 큰 이득은 없었음

안정성은 CI, Unit Test
성능 평가는 Easy Profiler, Evo (Map)

Local에서의 Uni

- 빌드를 할 때마다 Uni
- 기준이 되는 SLAM의
  매우 까다로운 조건이

GitHub에서 Uni

- Actions에서 CI로 빌
- 자동화는 성공했으나,
  큰 이득은 없었음

```
[==========] Running 3 test
[----------] Global test en
[----------] 3 tests from A
[ RUN      ] AssemblyTest.C
[       OK ] AssemblyTest.C
[ RUN      ] AssemblyTest.P
[       OK ] AssemblyTest.P
[ RUN      ] AssemblyTest.O
16:52:49.293|INFO   |BaseFr
16:52:49.294|INFO   |BaseFr
16:52:49.294|INFO   |BaseFr
16:52:49.294|INFO   |BaseFr
16:52:49.294|INFO   |BaseTr
16:52:49.294|INFO   |BaseTr
16:52:49.296|INFO   |Stereo
16:52:49.296|INFO   |Stereo
[       OK ] AssemblyTest.O
[----------] 3 tests from A

[----------] Global test en
[==========] 3 tests from 1
[  PASSED  ] 3 tests.
```

Test

succeeded now in 2m 54s

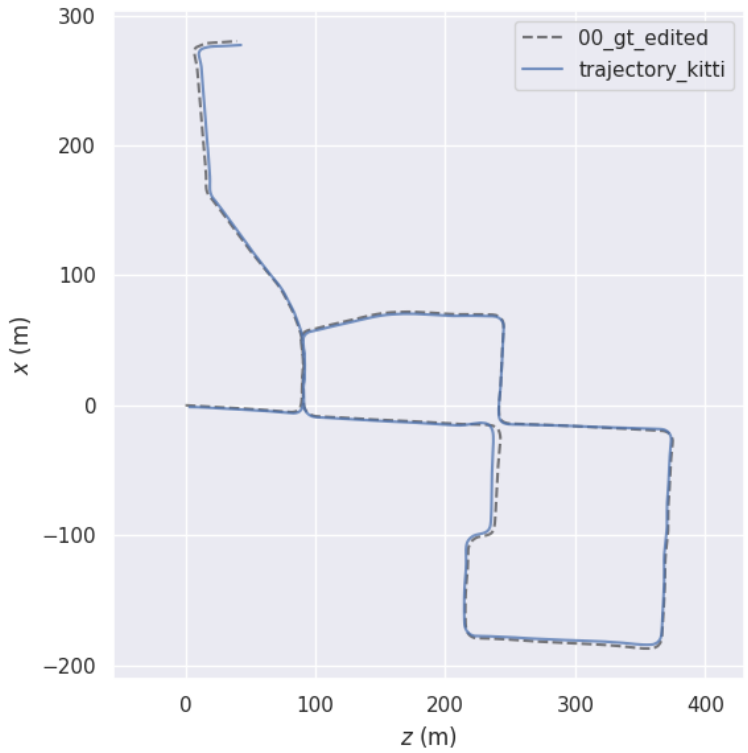
```
2303  Registering message with tag: PINHOLE_IMAGE_MESSAGE
2304  Registered class PinholeImageMessage
2305  Registering message with tag: POSE_MESSAGE
2306  Registered class PoseMessage
2307  Registering message with tag: SPHERICAL_IMAGE_MESSAGE
2308  Registered class SphericalImageMessage
2309  Registering message with tag: STATIC_TRANSFORM
2310  [==========] Running 3 tests from 1 test case.
2311  [----------] Global test environment set-up.
2312  [----------] 3 tests from AssemblyTest
2313  [ RUN      ] AssemblyTest.CameraLoad
2314  [       OK ] AssemblyTest.CameraLoad (198 ms)
2315  [ RUN      ] AssemblyTest.PlaybackTest
2316  [       OK ] AssemblyTest.PlaybackTest (4 ms)
2317  [ RUN      ] AssemblyTest.ORBTest
2318  17:03:31.527|INFO   |BaseFramePointGenerator::configure|detector_type: ORB
2319  17:03:31.527|INFO   |BaseFramePointGenerator::configure|descriptor_type: ORB
2320  17:03:31.527|INFO   |BaseFramePointGenerator::configure|current target number of points: 1197
2321  17:03:31.527|INFO   |BaseFramePointGenerator::configure|current target number of points per image region: 299
2322  17:03:31.527|INFO   |BaseTracker::configure|number of horizontal bins: 63 size: 20
2323  17:03:31.528|INFO   |BaseTracker::configure|number of vertical bins: 19 size: 20
2324  17:03:31.530|INFO   |StereoFramePointGenerator::configure|baseline (m): 0.537166
2325  17:03:31.530|INFO   |StereoFramePointGenerator::configure|number of epipolar lines considered for stereo matching: 1
2326  [       OK ] AssemblyTest.ORBTest (8 ms)
2327  [----------] 3 tests from AssemblyTest (210 ms total)
2328
2329  [----------] Global test environment tear-down
2330  [==========] 3 tests from 1 test case ran. (210 ms total)
2331  [  PASSED  ] 3 tests.
2332  Removing intermediate container 23110e7a68cd
2333   ---> 96cb2c92970a
2334  Successfully built 96cb2c92970a
2335  Successfully tagged seurislam-pj:9a9dd7e06aaab6a73da28568924f42b08b3644c5
2336  === finished ===
```

안정성은 CI, Unit Test
성능 평가는 Easy Profiler, Evo (Map)

기존 ProSLAM Setting

Feature detector ··· ORB

Matcher ····· ProSLAM's



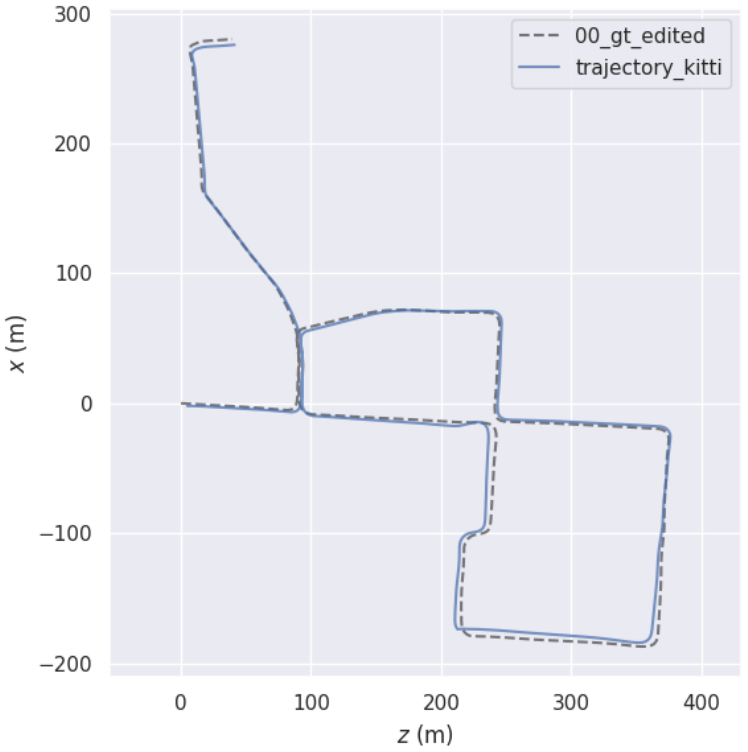
모듈러 아키텍처 이용

Feature detector ··· FAST

Matcher ····· Flann based

Homograpy ····· Use

Homograpy Type ··· LMEDS

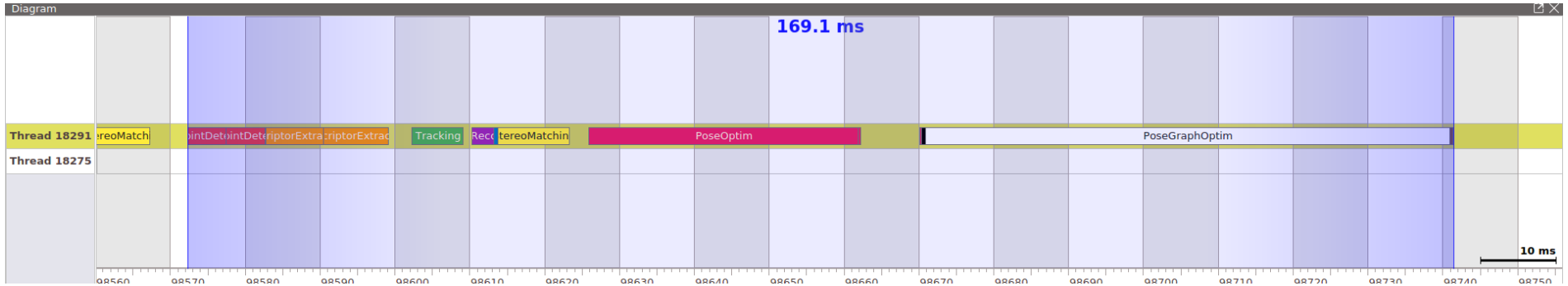


안정성은 CI, Unit Test
성능 평가는 Easy Profiler, Evo (Map)

기존 ProSLAM Setting

Feature detector ··· ORB

Matcher ····· ProSLAM's



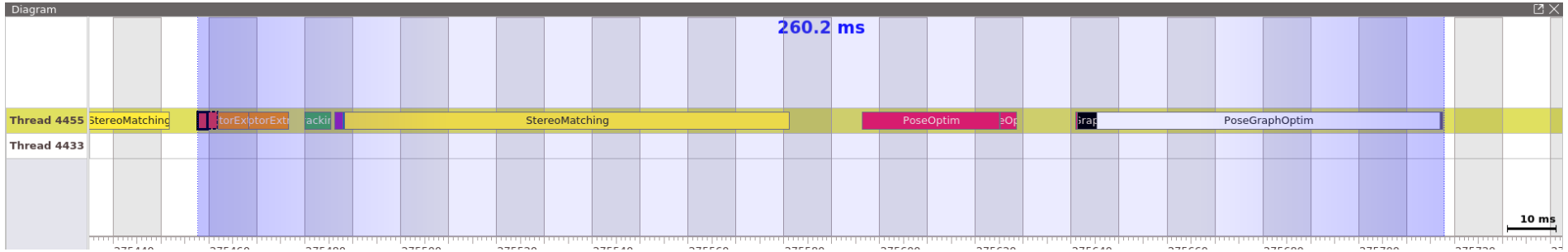
모듈러 아키텍처 이용

Feature detector ··· FAST

Matcher ····· Flann based

Homograpy ····· Use

Homograpy Type ··· LMEDS

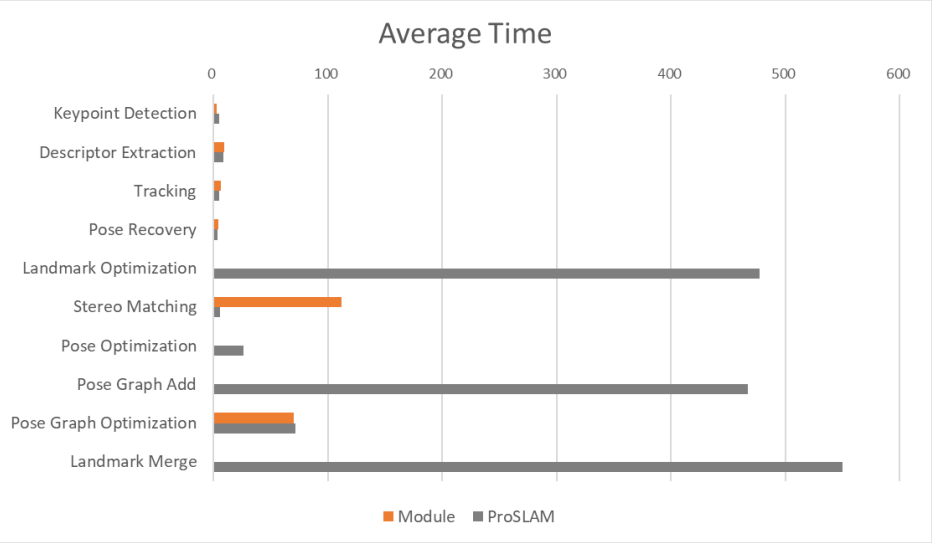


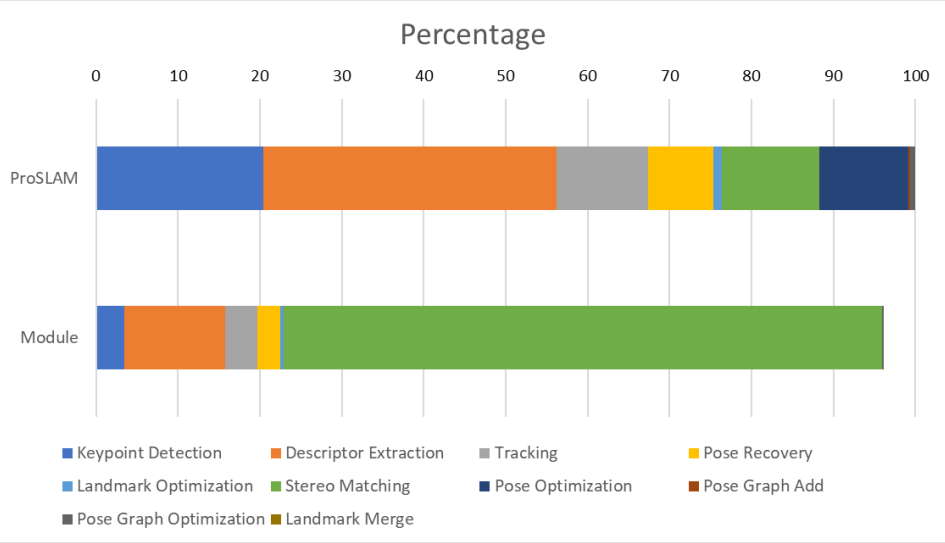
안정성은 CI, Unit Test
성능 평가는 Easy Profiler, Evo (Map)

기존 ProSLAM Setting

Feature detector ··· ORB

Matcher ····· ProSLAM's





모듈러 아키텍처 이용

Feature detector ··· FAST

Matcher ····· Flann based

Homograpy ····· Use

Homograpy Type ··· LMEDS

기존 ProSLAM과 구성이 달라지므로 1:1의 분석은 정확도가 떨어짐

모듈러 아키텍처에서 파라미터를 변경한 버전 간의 비교가 유의미할 것으로 예상

안정성은 CI, Unit Test
성능 평가는 Easy Profiler, Evo (Map)

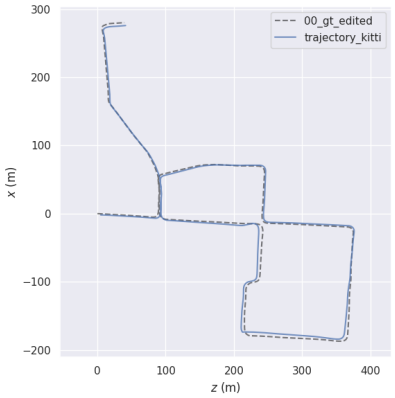
모듈러 아키텍처 (1)

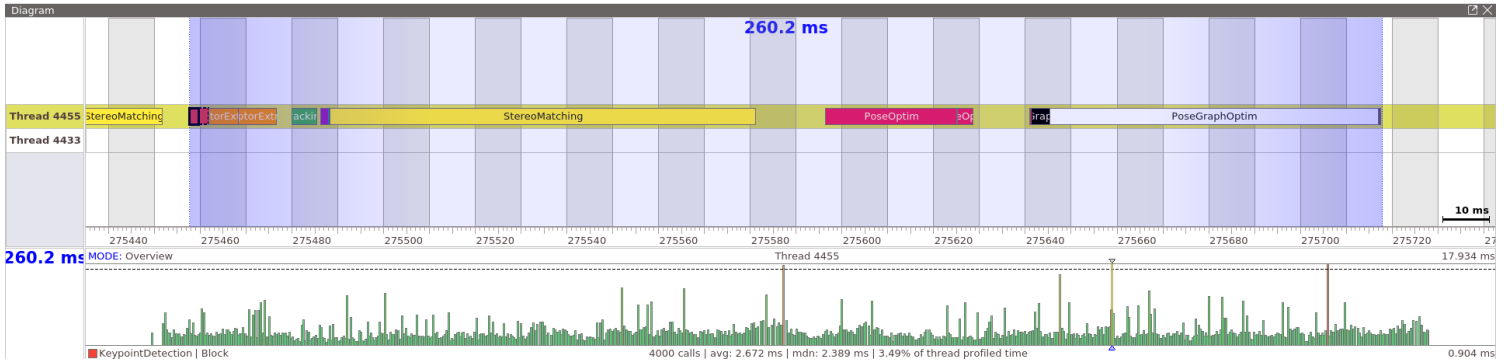
Aligner Type

ICP

Optimization Type

Gauss-Newton





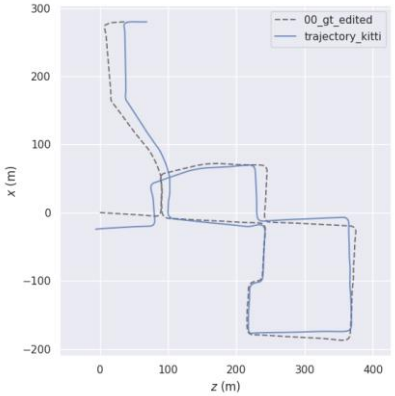
모듈러 아키텍처 (2)

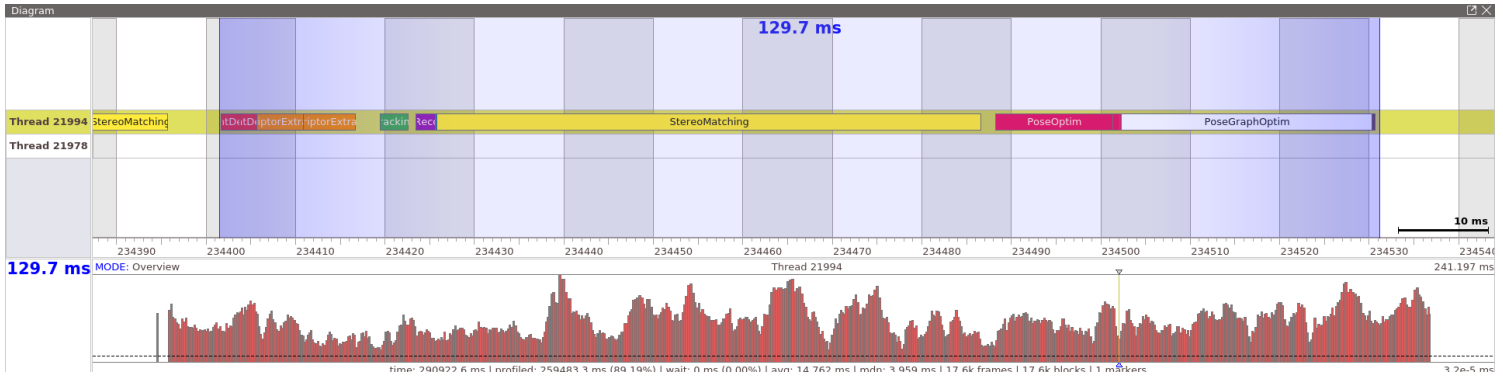
Aligner Type

FAST ICP

Optimization Type

DOGLEG





신뢰도는 높지 않을 것

다른 Open-Source SLAM 빌드하고 성능 비교하기
(feat. 제발 빌드 되어 주세요)

	ORB SLAM2	LSD SLAM	Rtap Map
특징	■ 자료가 많고, 주석이 잘 기술됨 ■ 다른 팀들이 모두 ORB 계열 사용 ■ Local Mapping과 Tracking, Loop Closure가 각 Thread에서	■ Tracking, Mapping, Constraint Search Optimization이 각 Thread에서 실행 ■ Tracking을 Lost할 경우 모든 Thread를 수동으로 재시작	■ 모듈화 측면에서 강사님의 추천을 받음 ■ IMU 등 다른 기법을 지원하는 SLAM ■ 실외(KITTI) 데이터를 '지원'은 함 ↳ 억지로 입력할 경우 에러가 큼
Build & Docker Image	O	O	O
속도 측정	O	O	X
정확도 측정	O	X	X

Pose 정보 수동 저장

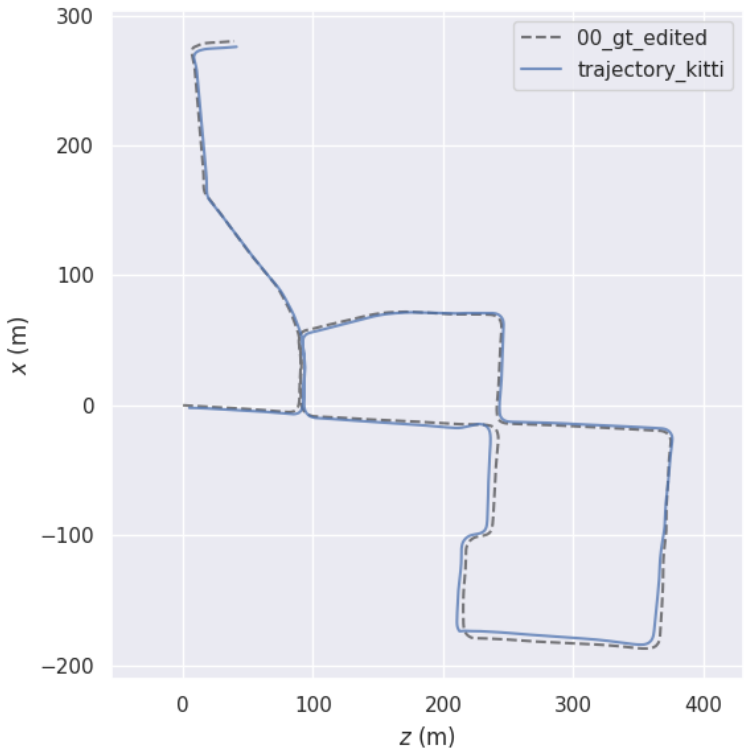
시간 부족 & 구조 복잡

다른 Open-Source SLAM 빌드하고 성능 비교하기
(feat. 제발 빌드 되어 주세요)

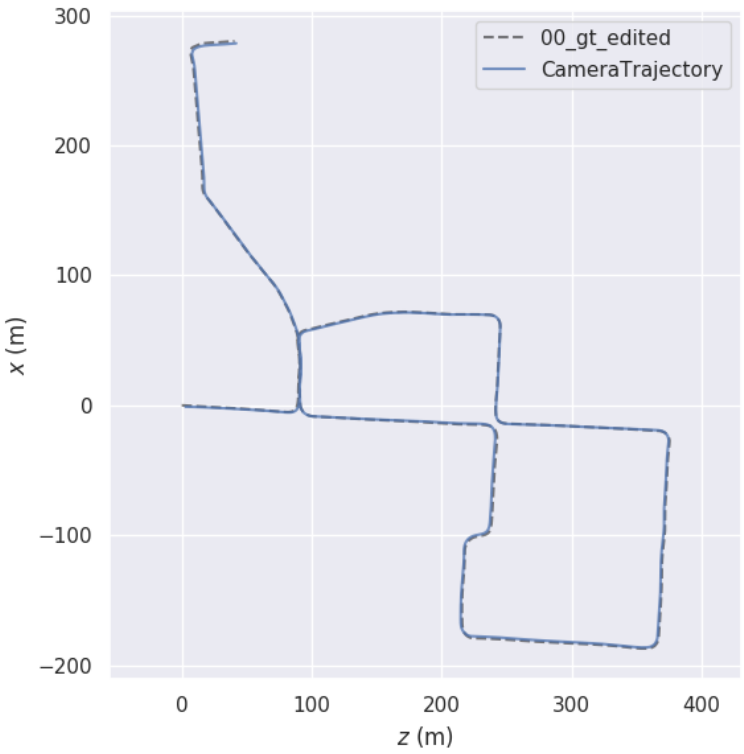
모듈러 아키텍처 이용

Feature detector	FAST
Matcher	Flann based
Homograpy	Use
Homograpy Type	LMEDS

Best 조합은 아니므로
더 성능이 좋아질 수도 있음



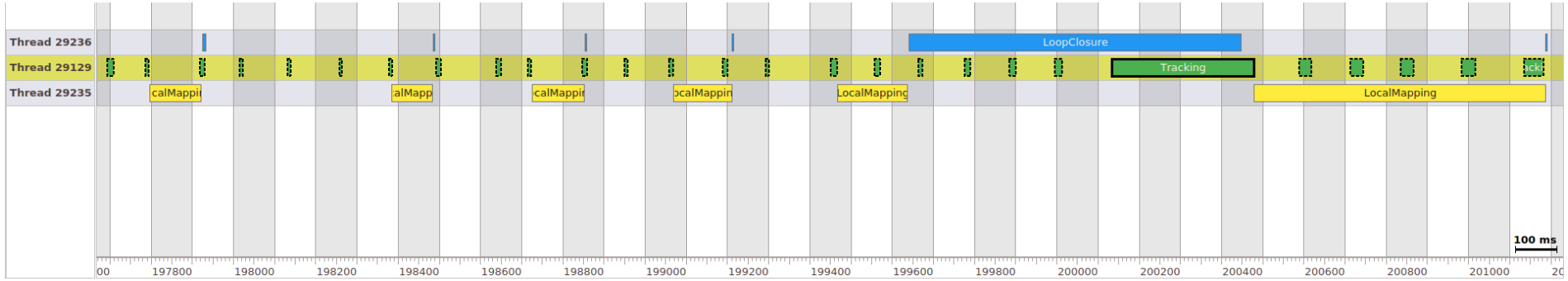
ORB SLAM2



다른 Open-Source SLAM 빌드하고 성능 비교하기
(feat. 제발 빌드 되어 주세요)

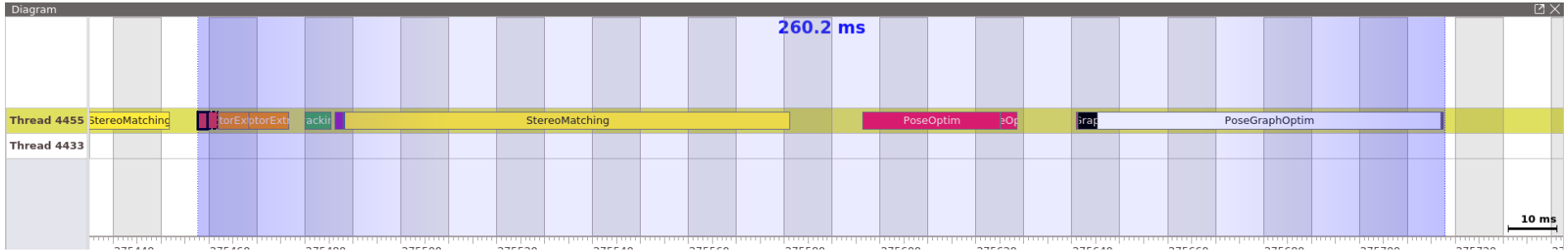
ORB SLAM2

3개의 Thread 생성해
Local Mapping, Tracking,
Loop Closure



모듈러 아키텍처 이용

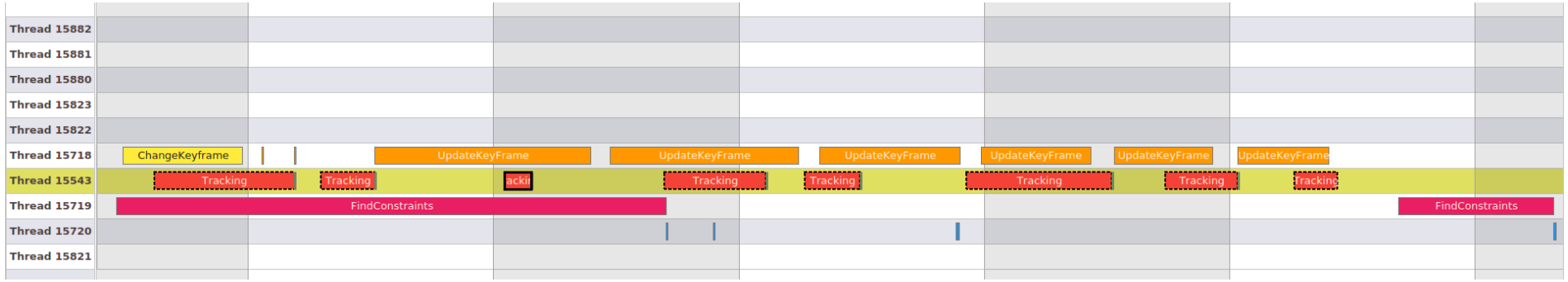
- Feature detector ··· FAST
- Matcher ····· Flann based
- Homograpy ····· Use
- Homograpy Type ··· LMEDS



다른 Open-Source SLAM 빌드하고 성능 비교하기
(feat. 제발 빌드 되어 주세요)

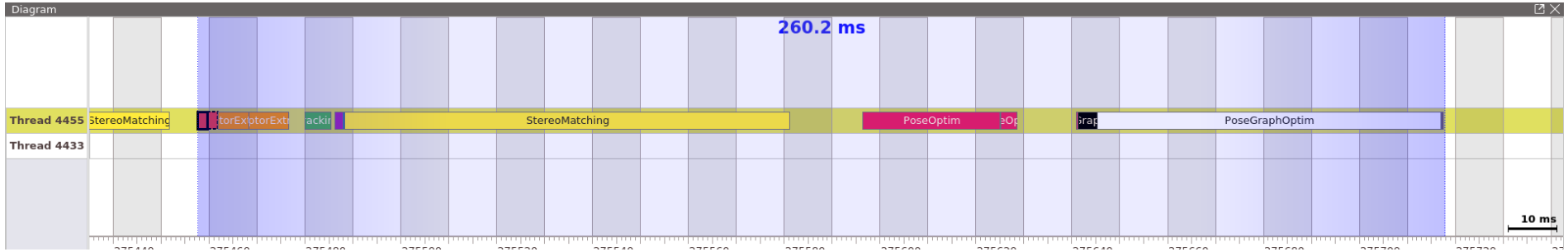
LSD SLAM

Tracking을 Lost할 때마다
재시작 명령을 내리면
Thread 재생성해 시작



모듈러 아키텍처 이용

- Feature detector ··· FAST
- Matcher ····· Flann based
- Homograpy ····· Use
- Homograpy Type ··· LMEDS



다른 Open-Source SLAM 빌드하고 성능 비교하기
(feat. 제발 빌드 되어 주세요)

Feature detector ··· FAST

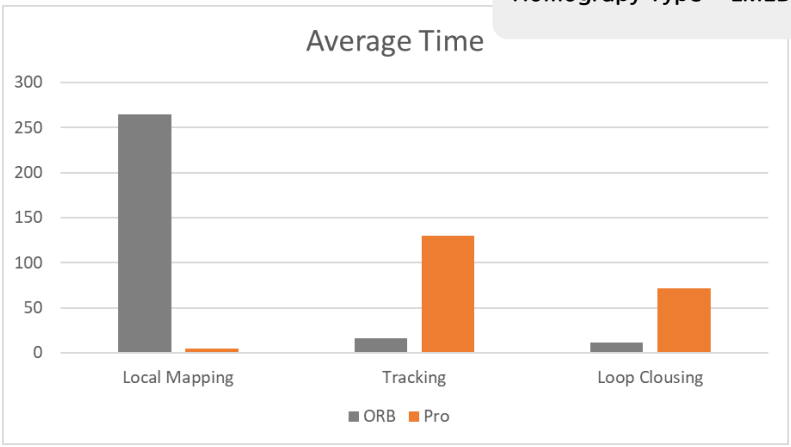
Matcher ··· Flann based

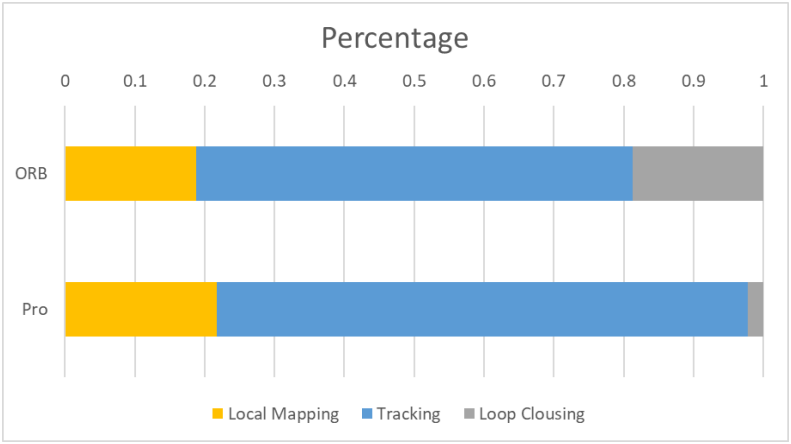
Homograpy ··· Use

Homograpy Type ··· LMEDS

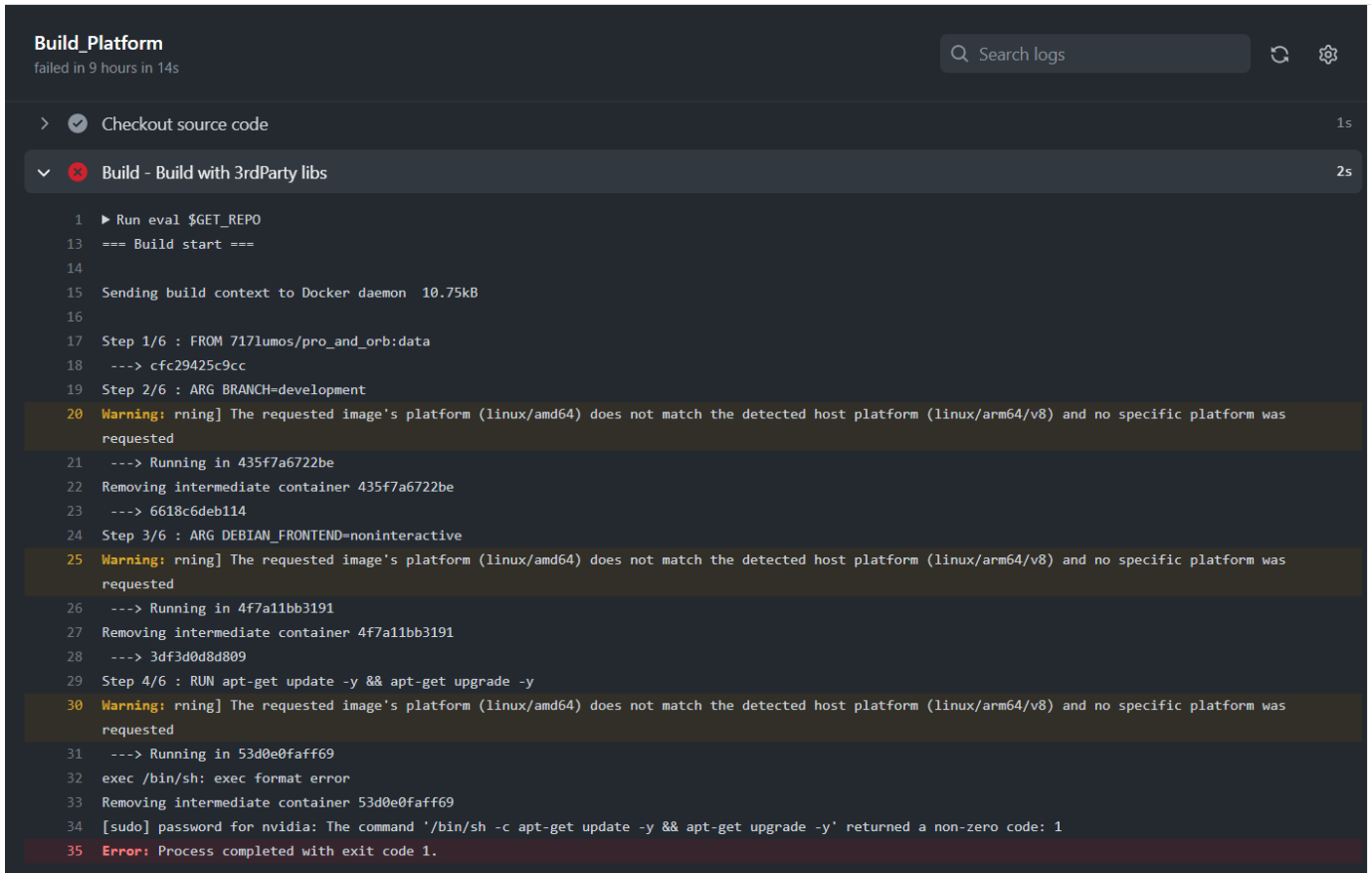
SLAM 별 모듈의 구분

	ProSLAM	ORB SLAM2	LSD SLAM
Triangulation	KeyPoint Detection Descriptor Extraction Stereo Matching	Tracking	Tracking
Incremental Motion Estimation	Tracking		
Map Management	Pose Recovery Landmark Optimization	Local Mapping	Map Estimation Update KeyFrame
Relocalization	Pose Optimization Pose Graph Add Pose Graph Optimization Landmark Merge	Loop Closure	Optimization Find Constraints Change KeyFrames





Xycar에서 4개 SLAM CI 자동화 & 실행시키기
(feat. 눼…? 정말 갑작스럽군요 😱)



기존 개발 플랫폼 ⟷ Xycar 플랫폼

AMD 기반 | ARM 64 / v8

Docker image 작동 X

- [ ] Xycar에서 라이브러리 & 소스코드 직접 빌드
- [ ] ARM 전용으로 Docker image 재빌드

⇨ 시간 부족으로 시도하지 못함

Contents

Team 스리슬램 SLAM PoC Project 회고

Project Result (Score)

분명 하는 데까지만 하자면서요…?

최종
1200점 달성

기본 목표

프로젝트 종료 시기에 맞춰 정해진 개발 목표 달성시키기 (기술개발)	✓	150
프로젝트 종료 시기에 맞춰 개발 내용 및 사용한 프레임워크의 알고리즘에 대해 발표 (이론 이해)	✓	150
중간 점검 시기에 맞춰 개발 중인 내용 보고 (프로젝트 매니징)	✓	150

보너스 포인트

KITTI 데이터셋에서 돌 수 있게 프레임워크 개량	✓	50
PC 웹캠/리얼센스 카메라로 실시간 데모가 가능하게 함		100
자이카에서 돌 수 있게 프레임워크 개량		100
정확도 개선		50
속도 개선	✓	50
오프라인 시각화 가능	✓	50
실시간 시각화 가능	✓	100

아키텍처/알고리즘 재사용성 개선	✓	100
안정성 확보: CI/CD 유닛테스트	✓	100
다른 팀에게도 도움이 될 수 있는 자료 정리 및 공유	✓	50
오픈소스를 참고해 직접 VSLAM 파이프라인을 설계 및 구현(최소 2 모듈 이상 변경)	✓	150
고객의 갑작스러운 요구사항1 달성	✓	100
고객의 갑작스러운 요구사항2 달성		200

스리슬쩍 결과물을 내고 싶었으나
매일매일이 다이나믹했던 2주간의 시간들

이창준

Image Processing Engineer

능력이 출중한 분들과 **협업**하면서 **다양하게 많이 배울** 수 있었다.

SLAM 프로젝트를 진행하기에 **준비가 덜 되어있다는 생각**이 들었고 실제로도 많이 어려워했다. 팀원분들이 잘 도와준 덕분에 잘 끝낸 것 같다. **다들 정말 감사드립니다!**

이론을 먼저 빠삭하게 공부한 뒤 코드를 분석했으면 훨씬 이해가 잘 되지 않았을까 아쉽다. 그래서 다른 SLAM을 분석할 때 다소 비효율적이었던 것 같다.

이현진

Frontend Engineer

다른 프로젝트와 다르게, 레퍼런스 코드가 없어 **스스로 기준 모델을 선정해 레퍼런스를 찾아야** 했다.

처음에는 힘들지만, **다른 프로젝트보다 많이 연구**했다.

자연히 **몰입감** 있게 프로젝트를 진행 할 수 있었다.

반면 **코드 분석의 시간이 상당히 소요**되었다는 점은 아쉽다.

빨리 결과물을 내야 하는데, **결과가 나오지 않아 답답한 마음**을 무시할 수가 없었다.

짧은 프로젝트 기간을 하면서 절반만 가자는 마인드였는데, 팀원들의 캐리로 인해서 **뭔가 결과가 좋게 나와서 기분이 좋았다.**

스리슬쩍 결과물을 내고 싶었으나
매일매일이 다이나믹했던 2주간의 시간들

유희평

Backend Engineer

논문이나 코드 둘 중 하나만 보면 SLAM 파이프라인을 이해하기 힘든 것 같다. 처음엔 논문을 보고 어느정도 이해를 쌓고 그 다음에 코드를 보았다. **규모가 있는 C++코드는 처음 분석**해봐서 며칠동안 끙끙대며 코드를 봤다.

Catkin으로 빌드하다보니 IDE에서 정의로 이동이 제대로 작동하지 않아서 더 힘들었다. 마치 **네비 없이 표지판만 보고 목적지를 찾아가는 느낌**이랄까.

그래도 하루하루 지날수록 어느정도 이해가 되고 코드에서 바꿀 만한 부분이 어디인지 알게 되었다.

적용할 만한 알고리즘을 다 구현해보고 싶었지만, 그 중 FAST-ICP부터 적용해보았다. 오류 없이 작동은 하지만, **이름은 FAST인데 FAST하지 않은 알고리즘**이 완성되었다.

수학적인 백그라운드가 없다보니 정확하게 동작하는 알고리즘을 만드는데 어려움이 있었다. **코드단에서 더 깊은 이해**를 했다면, **시간**이 더 주어졌다면, 알고리즘 속도 개선이 우리의 주목적이었다면 개선을 했을 수도 있을 것이다.

약간의 리펙터링을 통해 **ICP기법의 변경에 열려 있는 코드**를 만들었다.

빌드 지옥을 경험하면서 코드가 돌아가게 만든 것도 의미가 있다.

다음에는 **더 괜찮은 디버깅 툴**이 있나 찾아봐야겠다. 머리를 안 쓰면 몸이 고생한다.

스리슬쩍 결과물을 내고 싶었으나
매일매일이 다이나믹했던 2주간의 시간들

한은기

PM & Architecture

처음 팀빌딩이 되었을 때 '어벤저스 팀'이라 불렸다. 누가 되지 않도록 방학 기간동안 SLAM 뿐만 아니라 **CI/CD, Unit Test, Docker, Modern C++ 등을 치열하게 공부**했다.

공부한 내용을 **대부분 제대로 써보지 못해** 큰 아쉬움이 남지만, 분명 **엔지니어의 역량을 향상**시킬 좋은 시간이었다 생각한다.

초반에는 '**처음부터 아키텍처를 만들고** 살을 붙이자'고 했으나, 워낙 SLAM의 구조가 복잡하고 개념도 어려워 **기존의 모델을 수정**해야 했다.

그래서 **기능 구현이나 분석에 많은 제약**이 있었다. 더 풍부한 이론적 배경과 시간적 여유가 있었다면 **바닥부터 구현해 더욱 간단하고 유연한 프레임워크**를 만들 수 있지 않았을까?

막내가 팀을 이끄는 PM을 맡으며 부담감도 많았지만, 팀원들이 **적극적으로 의견을 개진하고 서로의 뜻을 존중**하는 분위기를 만들어준 덕분에 여유를 가지고 각자의 역할에 집중할 수 있었다.

후반부로 갈수록 컨디션이 나빠져 **팀원들을 걱정시켜 미안했다.** 프로젝트에 지장이 가지 않도록 **페이스를 조절해야 함**을 절실히 느꼈다.

Sharing Useful Contents

피땀눈물의 결과… 그대들에게 도움이 되었나요?
Notion, Docker, GitHub는 당분간 그만 보고 싶어요…

- ProSLAM 논문 정리
  - ProSLAM-Graph SLAM from a Programmer's Perspective.pdf 2363.7KB
- 이슈 해결
  - 템플릿 빌드 시 이슈 해결
  - easy profiler 이슈
  - proSLAM 빌드 이슈
  - PTAM 빌드 이슈
  - ORB-SLAM2 빌드 이슈
  - evo 활용법
- Docker
  - (Old) docker image 받아 사용하기
  - ProSLAM 데이터 포함 이미지 사용하기
  - LSD-SLAM 데이터 포함 이미지 사용하기
  - RTAB-Map 데이터 포함 이미지 사용하기
  - Docker 명령어 정리
- 참고할 코드
  - ProSLAM 외 구조 분석

programmers_slam_project_template

ProSLAM : https://gitlab.com/srrg-software/srrg_proslam

KITTI Data: https://drive.google.com/file/d/0ByaBRAPfmgEqdXhJRmktQ2lsM

LSD-SLAM: https://github.com/tum-vision/lsd_slam

Rtab-map: https://github.com/introlab/rtabmap

ORB SLAM2: https://github.com/raulmur/ORB_SLAM2

(Old) docker image 받아 사용하기

Docker Hub
https://hub.docker.com/r/717lumos/slam

1. 도커 계정을 만든다.
   a. 링크를 접속해 본인의 계정을 생성하고, User id와 비밀번호를 잘 기억해주세요.
2. 도커를 설치한다.
   a. 이전 버전이 있는지 확인 → 끝에 [installed]가 되어 있으면 `sudo apt-get remove <이름>` 으로 제거함

```
su -
apt list docker{,-engine,.io} containerd runc
```

ProSLAM 데이터 포함 이미지 사용하기

1. 도커 설치 & 로그인까지 다 된 이후라고 가정합니다.
2. 도커 이미지를 다운받는다.
   a. 이미지 pull하기

```
docker pull 717lumos/proslam:data
```

   b. 이미지 확인하기

```
docker images
```

3. 도커 컨테이너를 만든다
   a. 컨테이너 생성 명령어 입력하기 (--name 태그 뒤에 있는 proslam_data_devel 은 컨테이너의 이름으로, 본인이 변경하셔도 상관 없고, [--name <이름>] 부분은 쓰지 않으셔도 됩니다.) → 실행을 하고 나면 사용자가 `root@b6d38b3729c8:/#` 식으로 바뀔 겁니다.

Docker 명령어 정리

- 로그인하기: `docker login`

상태 확인

- 현재 컴퓨터에 저장된 이미지 목록 출력: `docker images`
  → 이미지 이름과 태그, ID 파악 가능
- 현재 실행중인 컨테이너 목록 출력 : `docker ps`
  → 컨테이너 이름과 ID, 사용하고 있는 image 파악 가능
- 컴퓨터에 저장된 컨테이너 목록을 모두 출력: `docker ps -a`
  → 컨테이너 이름과 ID, 사용하고 있는 image, 종료/진행 상태 파악 가능

Container

- 컨테이너 실행시키기: `docker run -it <이미지_이름>:<태그>`
  - 예: ubuntu 20.04를 실행할 컨테이너 만들기

```
docker run -it ubuntu:focal
```

  - 다양한 옵션 사용 가능
    (예) 컨테이너 이름을 부여하며 시작
    : `docker run -it ubuntu:focal --name my_ubuntu`
- 컨테이너 종료: `Ctrl + D`
- 기존에 생성되었던 컨테이너 재시작:
  `docker start <컨테이너_이름 or 컨테이너_ID>`
- 로컬 디렉터리에 있는 파일을 컨테이너 안으로 옮기기 (도커 밖에서 실행):
  `docker cp <로컬주소> <컨테이너_이름 or ID>:<컨테이너 내부에 복사할 경로>`

programmers
자율주행 데브코스 3기

Team 스리슬램

고생 많았습니다! 감사합니다!

2022.06.30.목

발표자 한은기(PM)