Cartographer @ 오로카

(Real-Time Loop Closure in 2D LIDAR SLAM)

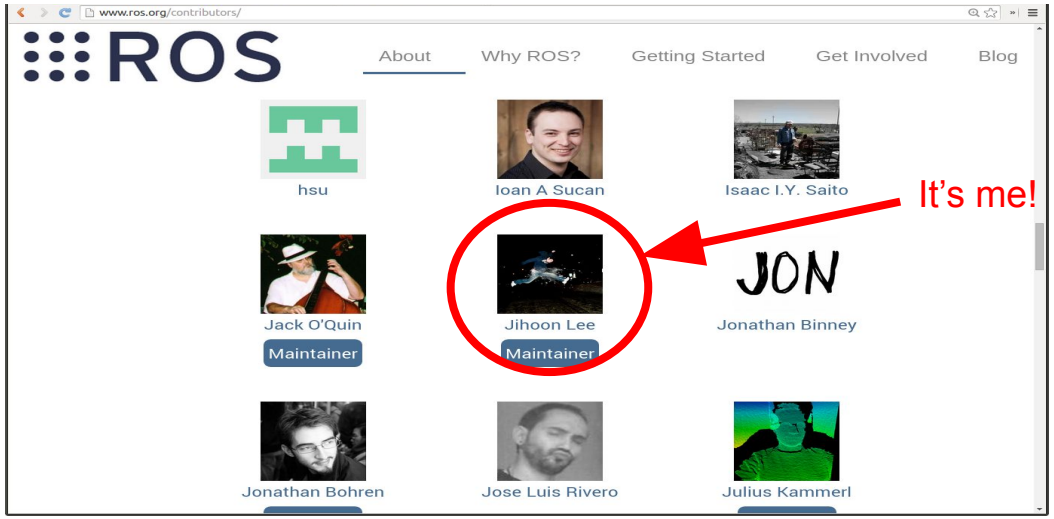
2020.12.01

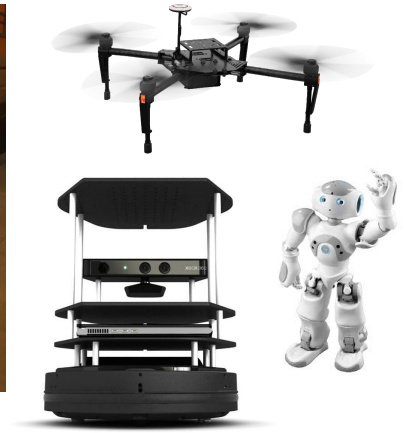
Jihoon Lee

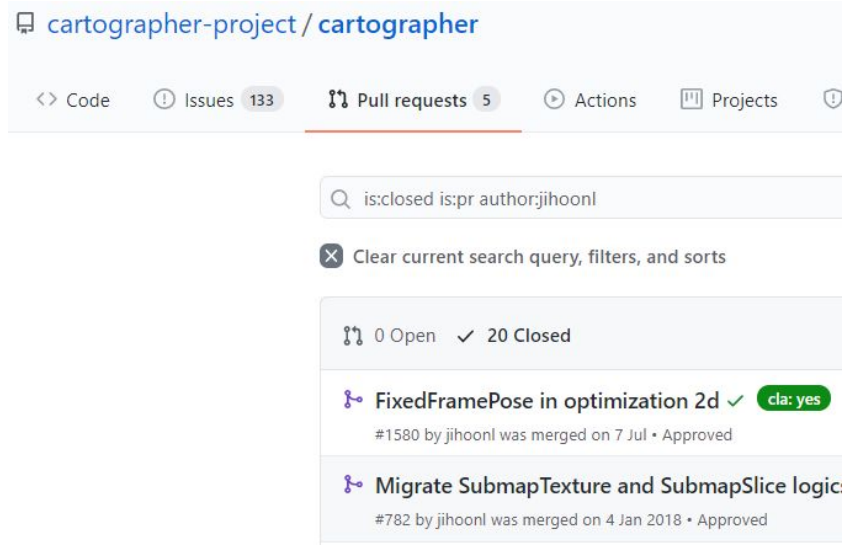
Speaker

이지훈

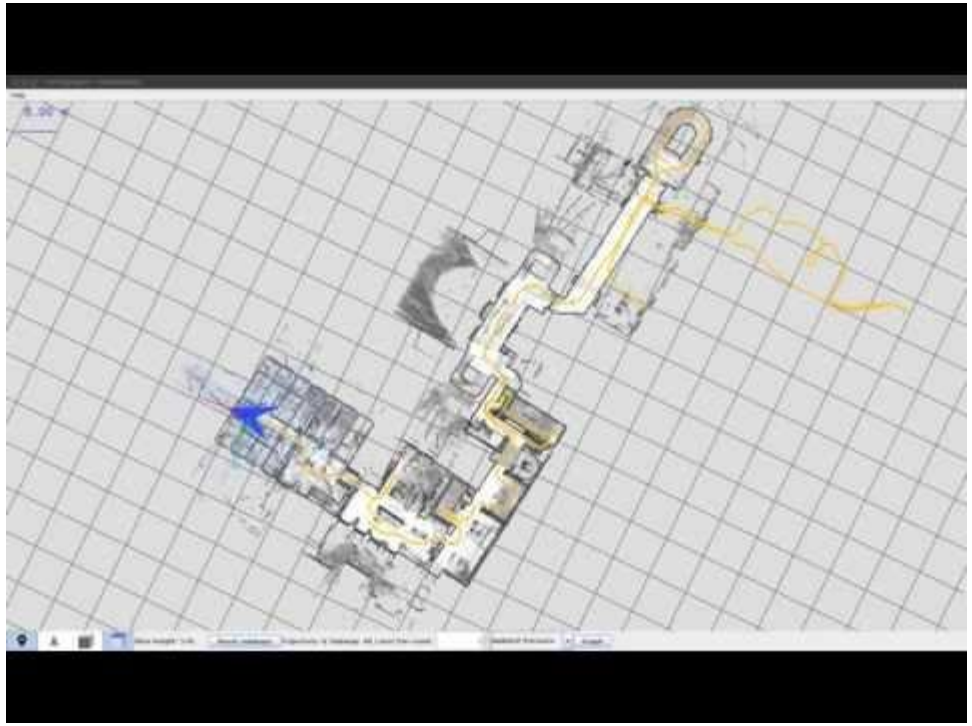
- 현 Kakao Brain
- Github: https://github.com/jihoonl







Cartographer



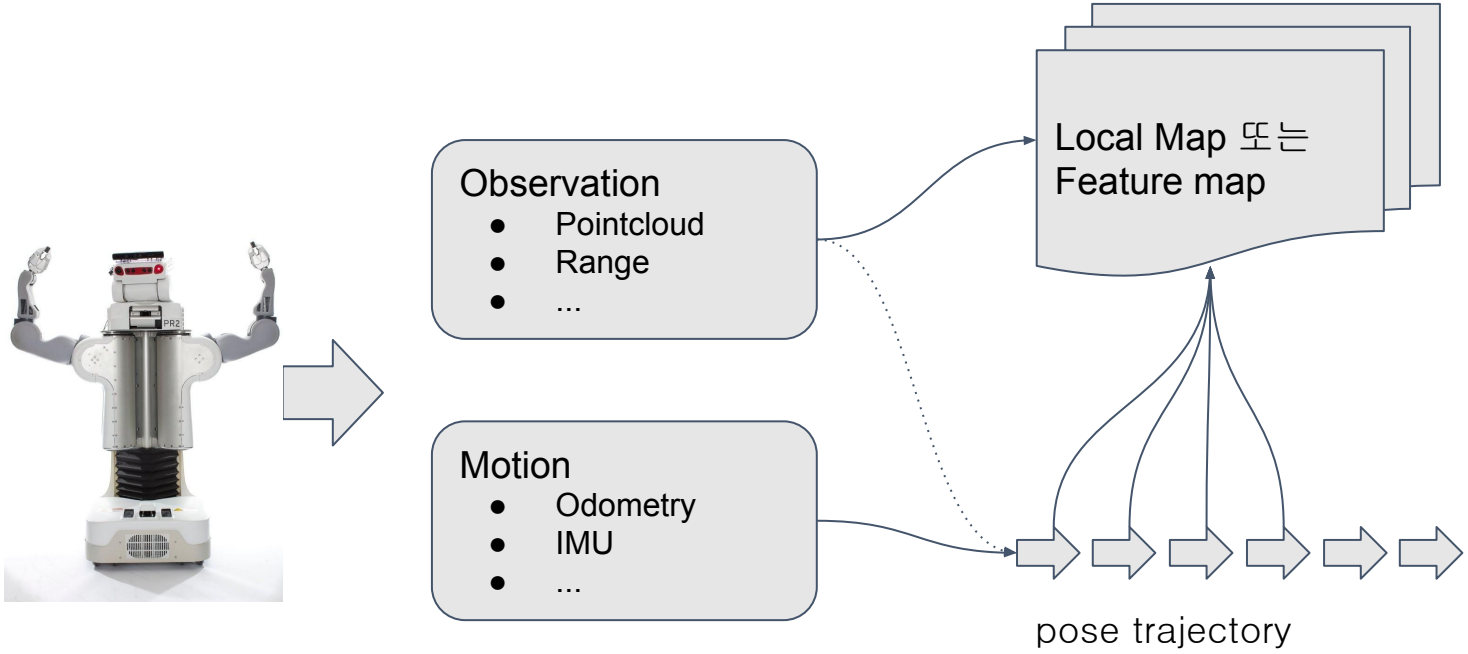
- General Graph SLAM “system” Recap
- Paper Review - “Real-Time Loop Closure in 2D LIDAR SLAM”
- Cartographer as open source
  - Beyond the paper

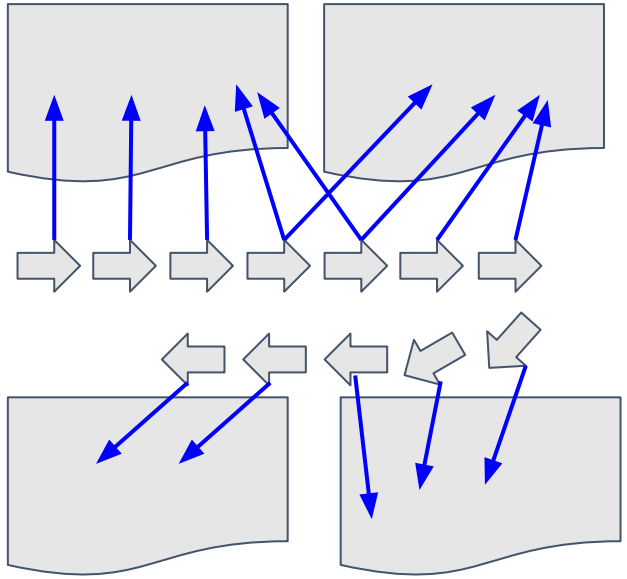
일반적인 Graph SLAM 구조?

Observation
- Pointcloud
- Range
- ...

Motion
- Odometry
- IMU
- ...

일반적인 Graph SLAM 구조?





일반적인 Graph SLAM 구조?

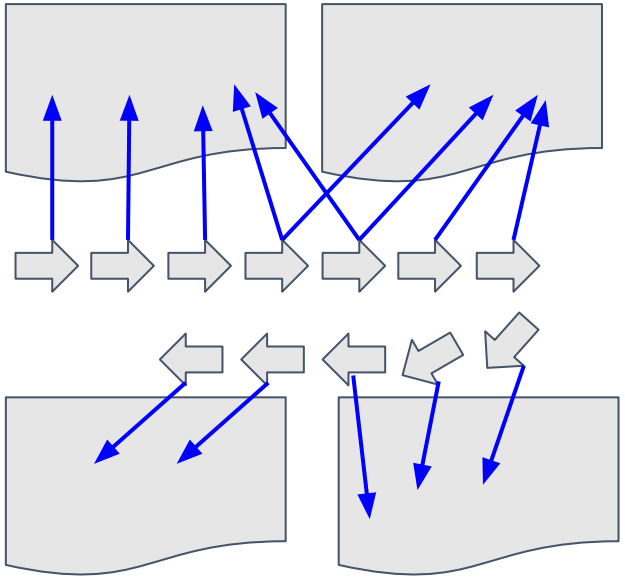
Local SLAM
또는
Frontend

Observation
- Pointcloud
- Range
- ...

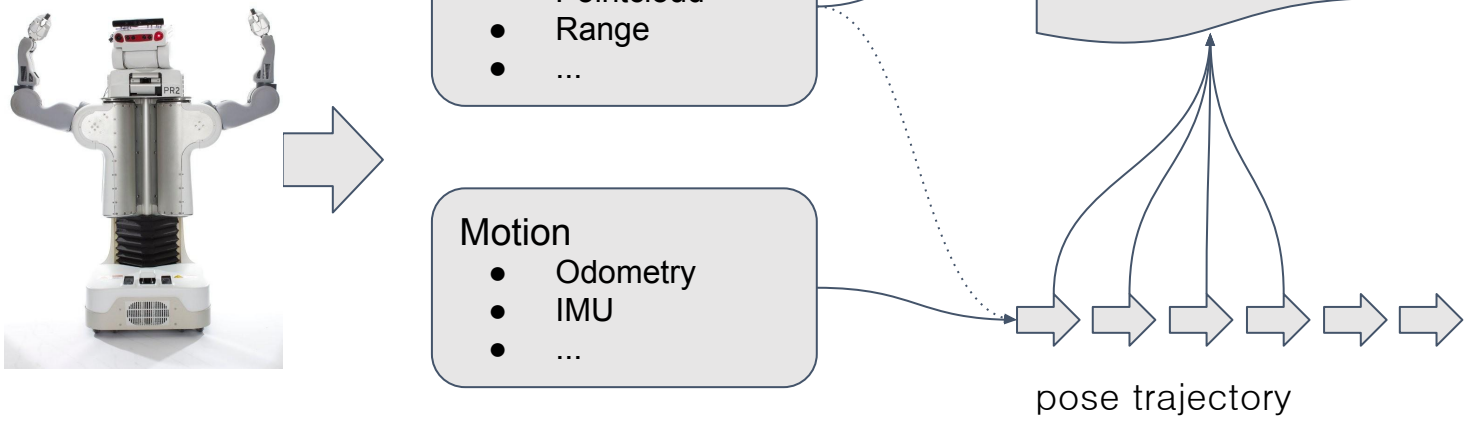
Motion
- Odometry
- IMU
- ...

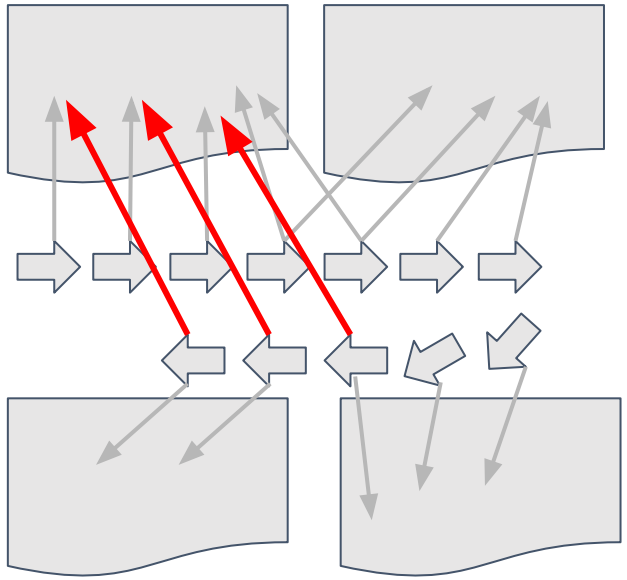
Local Map 또는
Feature map

pose trajectory



일반적인 Graph SLAM 구조?





Loop closure detection

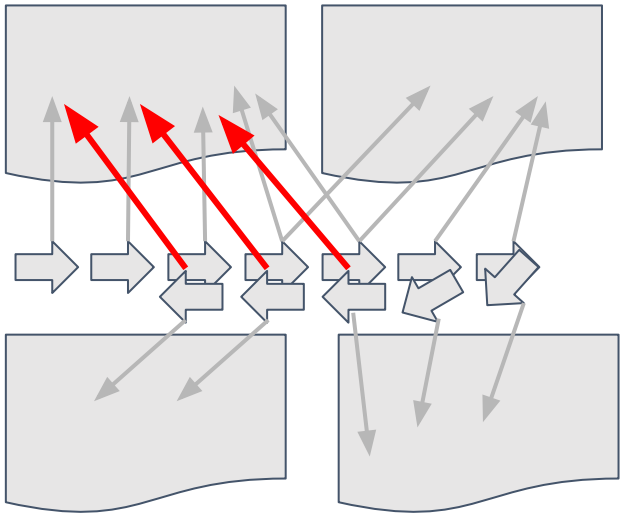
일반적인 Graph SLAM 구조?

Observation
- Pointcloud
- Range
- ...

Motion
- Odometry
- IMU
- ...

Local Map 또는
Feature map

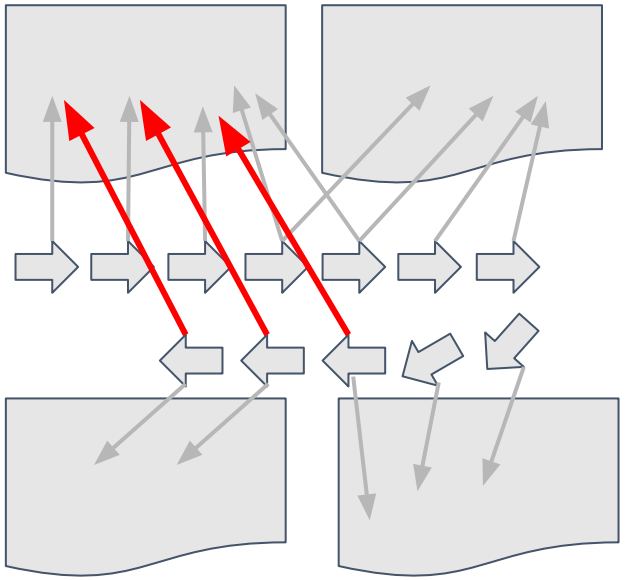
pose trajectory



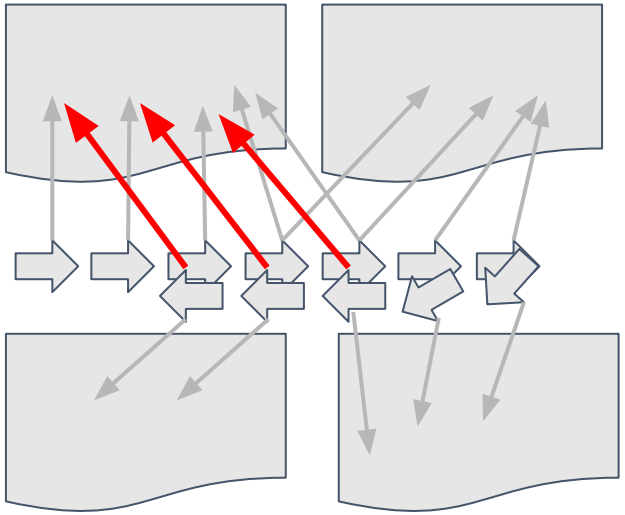
Optimization

일반적인 Graph SLAM 구조?

Global SLAM
또는
Backend

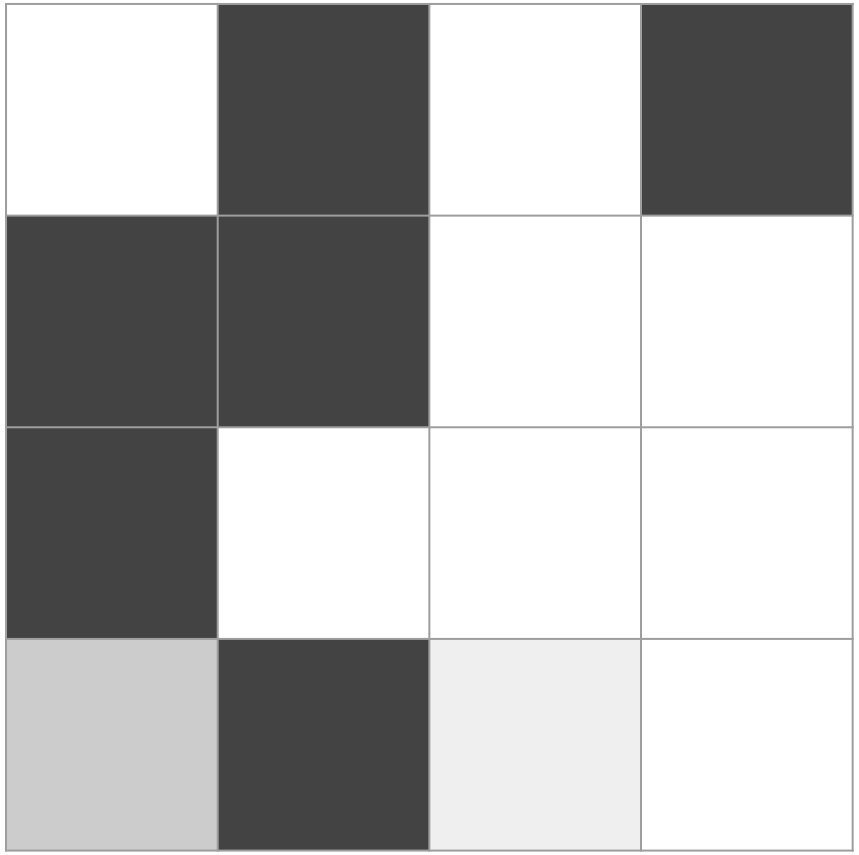


Loop closure
detection

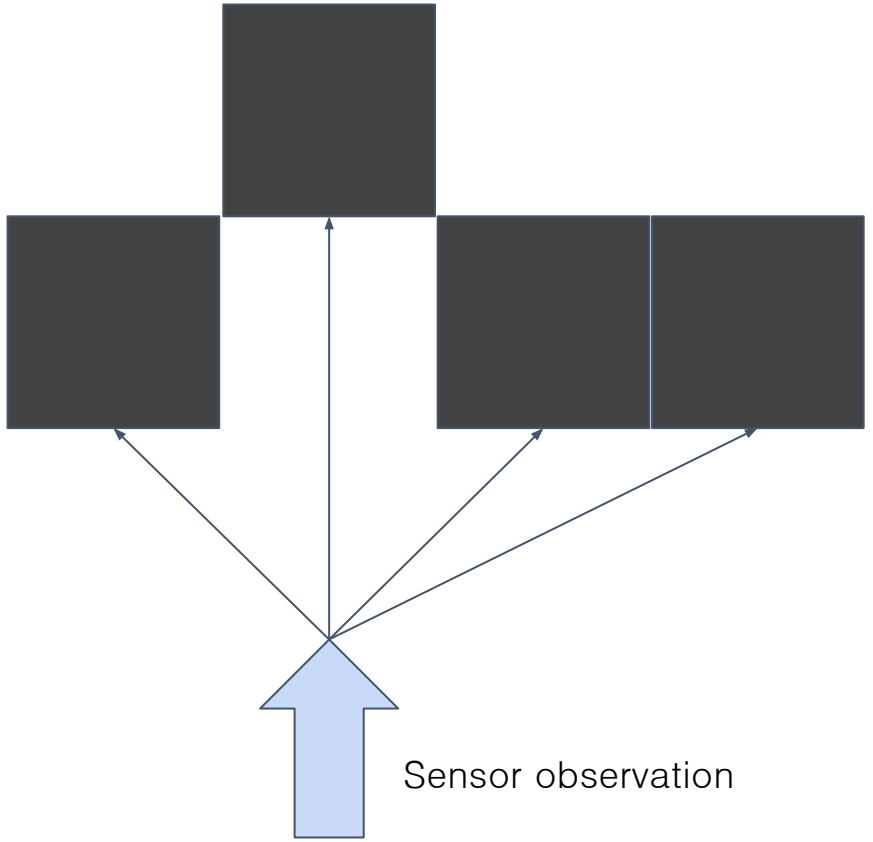


Optimization

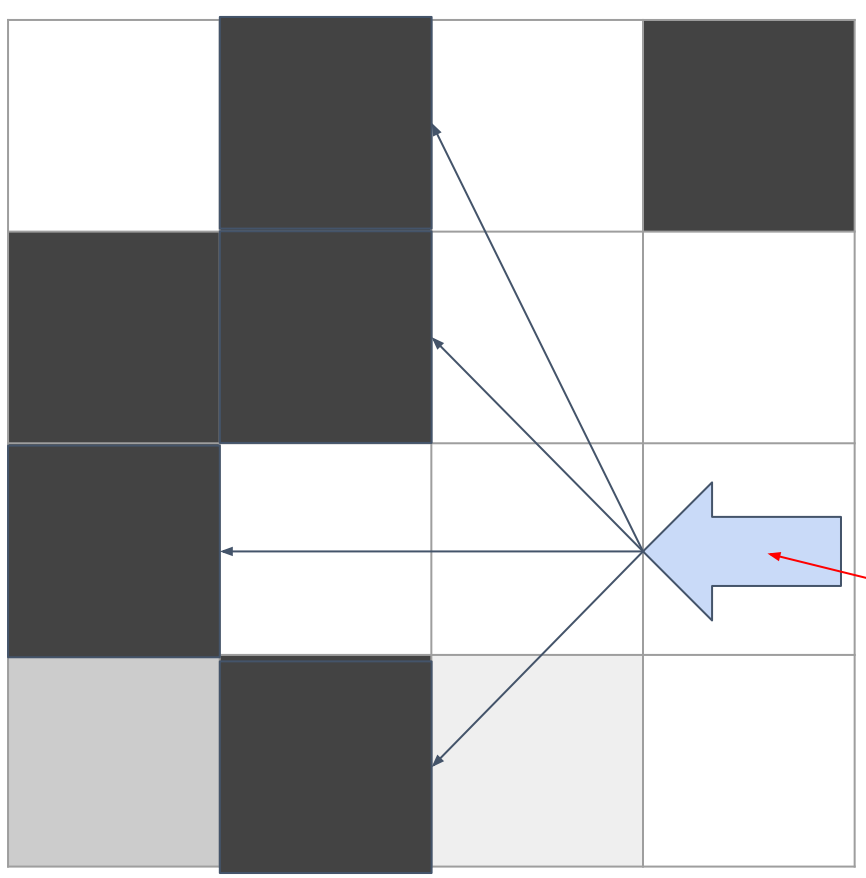
Loop Closure Detection - Revisit



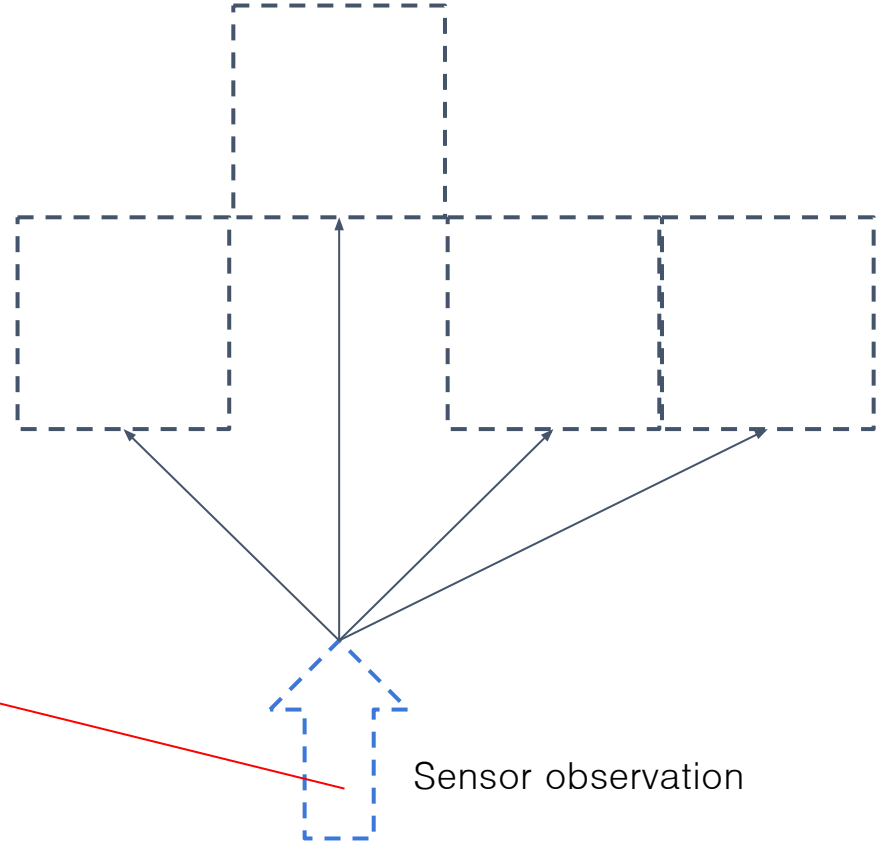
Map



Loop Closure Detection - Revisit



Map



다른 시간대에 만들어진 다른 modality간의 연결점을 찾아주는 것

- Map Resolution ↑ -> search space ↑↑
- Map size ↑ -> search space ↑↑

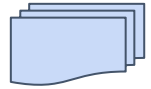
Real-Time Loop Closure in 2D LIDAR SLAM

Real-Time Loop Closure in 2D LIDAR SLAM

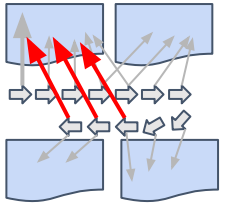
Abstract— Portable laser range-finders, further referred to as LIDAR, and simultaneous localization and mapping (SLAM) are an efficient method of acquiring as-built floor plans. Generating and visualizing floor plans in real-time helps the operator assess the quality and coverage of capture data. Building a portable capture platform necessitates operating under limited computational resources. **We present the approach used in our backpack mapping platform which achieves real-time mapping and loop closure at a 5 cm resolution. To achieve realtime loop closure, we use a branch-and-bound approach for computing scan-to-submap matches as constraints.** We provide experimental results and comparisons to other well known approaches which show that, in terms of quality, our approach is competitive with established techniques.

쉽게 말하면 **5cm 수준의 고화질의 Map**에서 Global(Loop closure) Constraint을 ~~실시간~~**실시간 수준**으로 계산할 수 있는 Branch-and-bound방식을 제안한다는 것
논문 구성

- Local map(named as Submap in cartographer) 표현과 생성 방법



- Branch-and-Bound 알고리즘 설명 - Loop closure detection



- 실험 결과 비교

Local SLAM - Submap

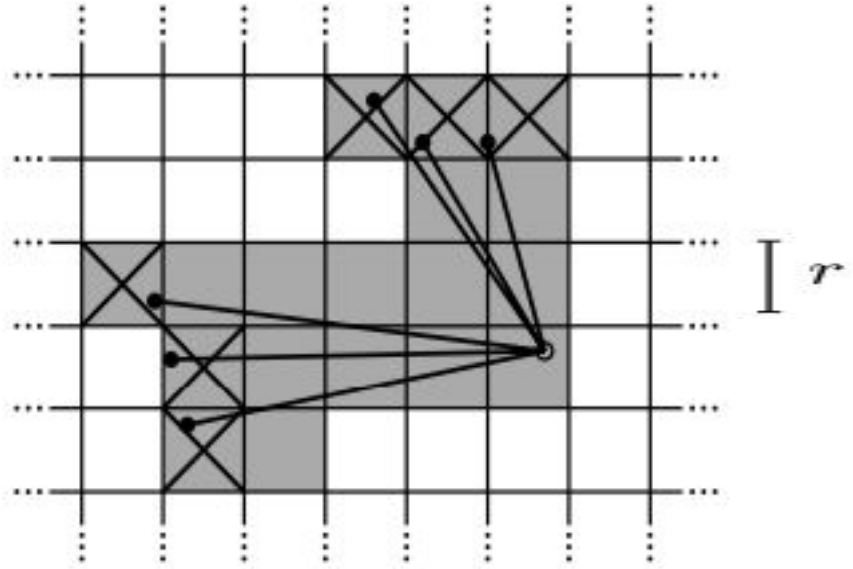


Fig. 2. A scan and pixels associated with *hits* (shaded and crossed out) and *misses* (shaded only).

$$\text{odds}(p) = \frac{p}{1-p}, \tag{2}$$

$$M_{\text{new}}(x) = \text{clamp}(\text{odds}^{-1}(\text{odds}(M_{\text{old}}(x)) \cdot \text{odds}(p_{\text{hit}}))) \tag{3}$$

- Probability Grid 기반
- 각 셀은 logodd로 p_miss, p_hit을 둘다 기록
- 각 셀의 크기가 맵의 resolution을 결정
  - 5cm resolution 맵이면 셀의 크기가 5cm
- N개의 Consecutive scan이 모여서 한개의 submap 구성
  - 로봇의 주행과 scan의 거리에 따라서 submap이 클수도 작을 수도 있음
- 각각의 submap들은 생성시점에서는 서로 독립적
  - N이 클 경우 모션 모델의 영향을 많이 받음.

- noise 보정을 위해 두개의 추가적 알고리즘
  - real-time correlated scan matcher – olson2009icra
  - ceres scan matcher – 수학적 최적화

$$\underset{\xi}{\text{argmin}} \sum_{k=1}^{K} \left(1 - M_{\text{smooth}}(T_\xi h_k)\right)^2 \qquad \text{(CS)}$$

Local SLAM - Submap

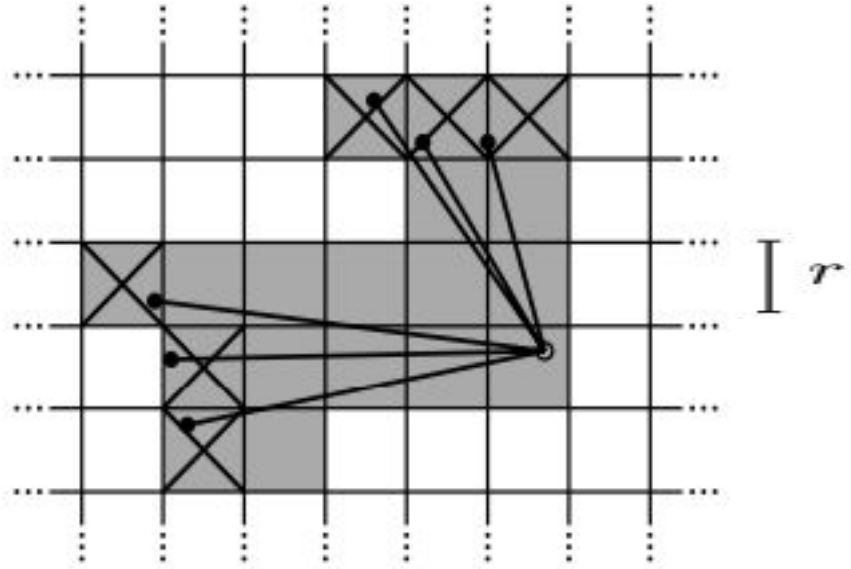


Fig. 2. A scan and pixels associated with *hits* (shaded and crossed out) and *misses* (shaded only).

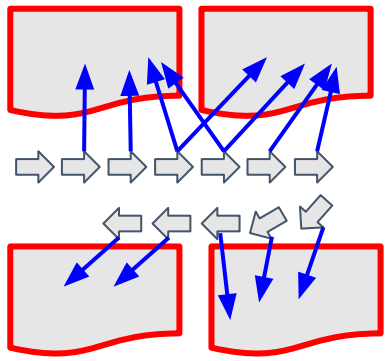
$$\text{odds}(p) = \frac{p}{1-p}, \tag{2}$$

$$M_{\text{new}}(x) = \text{clamp}(\text{odds}^{-1}(\text{odds}(M_{\text{old}}(x)) \cdot \text{odds}(p_{\text{hit}}))) \tag{3}$$

- Probability Grid 기반
- 각 셀은 logodd로 p_miss, p_hit을 둘다 기록
- 각 셀의 크기가 맵의 resolution을 결정
  - 5cm resolution 맵이면 셀의 크기가 5cm
- N개의 Consecutive scan이 모여서 한개의 submap 구성
  - 로봇의 주행과 scan의 거리에 따라서 submap이 클수도 작을 수도 있음
- 각각의 submap들은 생성시점에서는 서로 독립적
  - N이 클 경우 모션 모델의 영향을 많이 받음.

- noise 보정을 위해 두개의 추가적 알고리즘
  - real-time correlated scan matcher – olson2009icra
  - ceres scan matcher – 수학적 최적화

$$\underset{\xi}{\text{argmin}} \sum_{k=1}^{K} \left(1 - M_{\text{smooth}}(T_\xi h_k)\right)^2 \tag{CS}$$



Branch-and-Bound

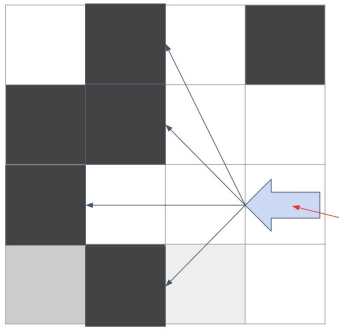
Algorithm 3 DFS branch and bound scan matcher for (BBS)

$best\_score \leftarrow score\_threshold$
Compute and memorize a score for each element in $\mathcal{C}_0$.
Initialize a stack $\mathcal{C}$ with $\mathcal{C}_0$ sorted by score, the maximum score at the top.
while $\mathcal{C}$ is not empty **do**
  Pop c from the stack $\mathcal{C}$.
  if $score(c) > best\_score$ **then**
    if c is a leaf node **then**
      $match \leftarrow \xi_c$
      $best\_score \leftarrow score(c)$
    else
      Branch: Split c into nodes $\mathcal{C}_c$.
      Compute and memorize a score for each element in $\mathcal{C}_c$.
      Push $\mathcal{C}_c$ onto the stack $\mathcal{C}$, sorted by score, the maximum score last.
    end if
  end if
end while
return $best\_score$ and $match$ when set.

특정 submap내 최적해를 가진 위치를 찾기 위한 트리구조의 탐색 알고리즘

- Node: Candidate pose
- Height: Higher resolution
- 저해상도의 Map에서 시작해서 pose를 추론해서 점점 고해상도내의 pose를 찾아낸다.

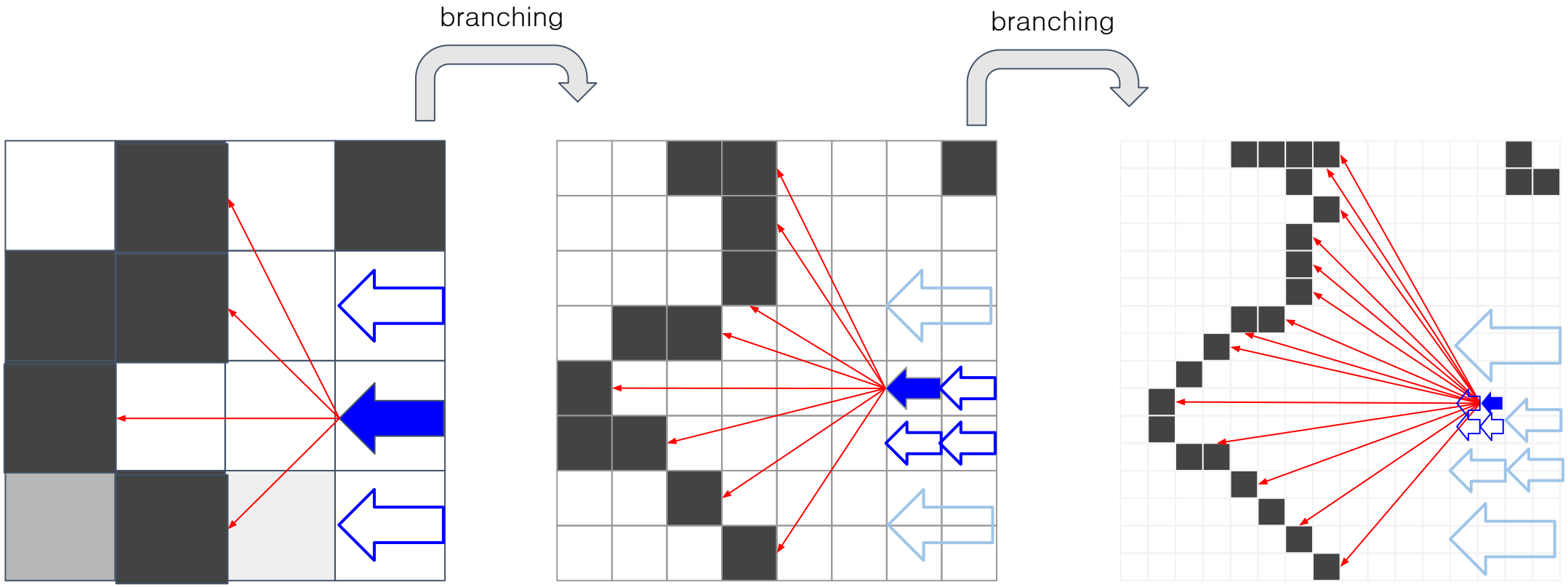
- DFS, matching score 기반
- Computing Upper bounds
  - 각 node들의 matching score 계산 방식



Best matching pose candidate

$$score(c) = \sum_{k=1}^{K} \max_{j \in \overline{\mathcal{W}}_c} M_{\text{nearest}}(T_{\xi_j} h_k)$$
$$\geq \sum_{k=1}^{K} \max_{j \in \overline{\mathcal{W}}_c} M_{\text{nearest}}(T_{\xi_j} h_k) \qquad (15)$$
$$\geq \max_{j \in \overline{\mathcal{W}}_c} \sum_{k=1}^{K} M_{\text{nearest}}(T_{\xi_j} h_k).$$

Branch-and-Bound



Height 0

Height 1

Height 2

- Leaf nodes
- Highest resolution

Experiments

TABLE II

QUANTITATIVE COMPARISON OF ERROR WITH [21]

	Cartographer	GM
Aces		
Absolute translational	0.0375 ± 0.0426	0.044 ± 0.044
Squared translational	0.0032 ± 0.0285	0.004 ± 0.009
Absolute rotational	0.373 ± 0.469	0.4 ± 0.4
Squared rotational	0.359 ± 3.696	0.3 ± 0.8
Intel		
Absolute translational	0.0229 ± 0.0239	0.031 ± 0.026
Squared translational	0.0011 ± 0.0040	0.002 ± 0.004
Absolute rotational	0.453 ± 1.335	1.3 ± 4.7
Squared rotational	1.986 ± 23.988	24.0 ± 166.1
MIT Killian Court		
Absolute translational	0.0395 ± 0.0488	0.050 ± 0.056
Squared translational	0.0039 ± 0.0144	0.006 ± 0.029
Absolute rotational	0.352 ± 0.353	0.5 ± 0.5
Squared rotational	0.248 ± 0.610	0.9 ± 0.9
MIT CSAIL		
Absolute translational	0.0319 ± 0.0363	0.004 ± 0.009
Squared translational	0.0023 ± 0.0099	0.0001 ± 0.0005
Absolute rotational	0.369 ± 0.365	0.05 ± 0.08
Squared rotational	0.270 ± 0.637	0.01 ± 0.04
Freiburg bldg 79		
Absolute translational	0.0452 ± 0.0354	0.056 ± 0.042
Squared translational	0.0033 ± 0.0055	0.005 ± 0.011
Absolute rotational	0.538 ± 0.718	0.6 ± 0.6
Squared rotational	0.804 ± 3.627	0.7 ± 1.7
Freiburg hospital (local)		
Absolute translational	0.1078 ± 0.1943	0.143 ± 0.180
Squared translational	0.0494 ± 0.2831	0.053 ± 0.272
Absolute rotational	0.747 ± 2.047	0.9 ± 2.2
Squared rotational	4.745 ± 40.081	5.5 ± 46.2
Freiburg hospital (global)		
Absolute translational	5.2242 ± 6.6230	11.6 ± 11.9
Squared translational	71.0288 ± 267.7715	276.1 ± 516.5
Absolute rotational	3.341 ± 4.797	6.3 ± 6.2
Squared rotational	34.107 ± 127.227	77.2 ± 154.8

TABLE III

QUANTITATIVE COMPARISON OF ERROR WITH [9]

	Cartographer	Graph FLIRT
Intel		
Absolute translational	0.0229 ± 0.0239	0.02 ± 0.02
Absolute rotational	0.453 ± 1.335	0.3 ± 0.3
Freiburg bldg 79		
Absolute translational	0.0452 ± 0.0354	0.06 ± 0.09
Absolute rotational	0.538 ± 0.718	0.8 ± 1.1
Freiburg hospital (local)		
Absolute translational	0.1078 ± 0.1943	0.18 ± 0.27
Absolute rotational	0.747 ± 2.047	0.9 ± 2.0
Freiburg hospital (global)		
Absolute translational	5.2242 ± 6.6230	8.3 ± 8.6
Absolute rotational	3.341 ± 4.797	5.0 ± 5.3

TABLE IV

LOOP CLOSURE PRECISION

Test case	No. of constraints	Precision
Aces	971	98.1 %
Intel	5786	97.2 %
MIT Killian Court	916	93.4 %
MIT CSAIL	1857	94.1 %
Freiburg bldg 79	412	99.8 %
Freiburg hospital	554	77.3 %

TABLE V

PERFORMANCE

Test case	Data duration (s)	Wall clock (s)
Aces	1366	41
Intel	2691	179
MIT Killian Court	7678	190
MIT CSAIL	424	35
Freiburg bldg 79	1061	62
Freiburg hospital	4820	10

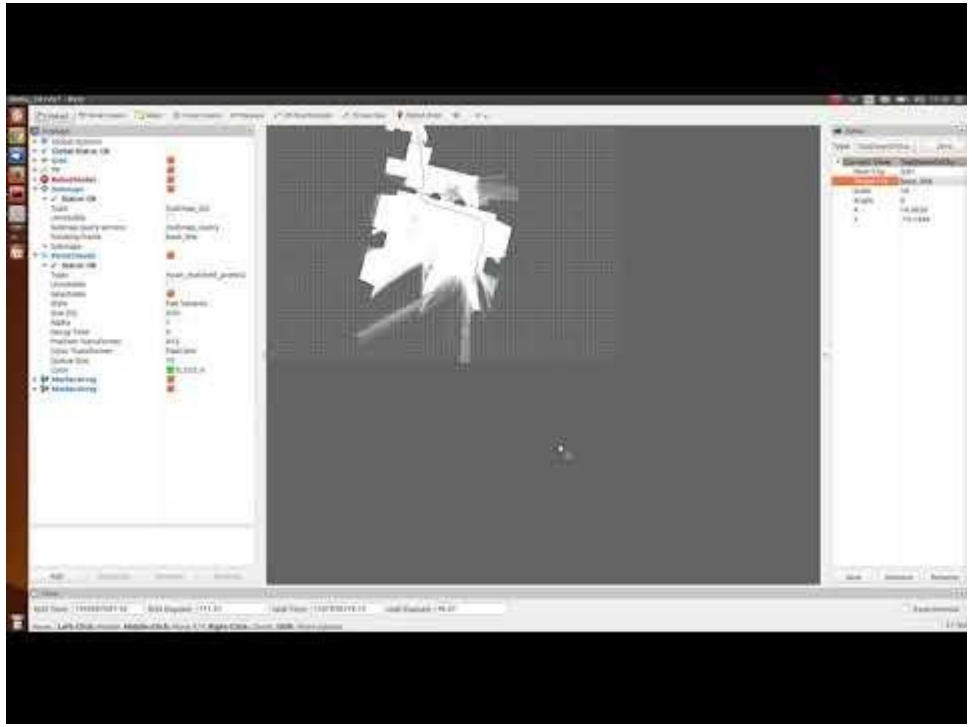
Experiments

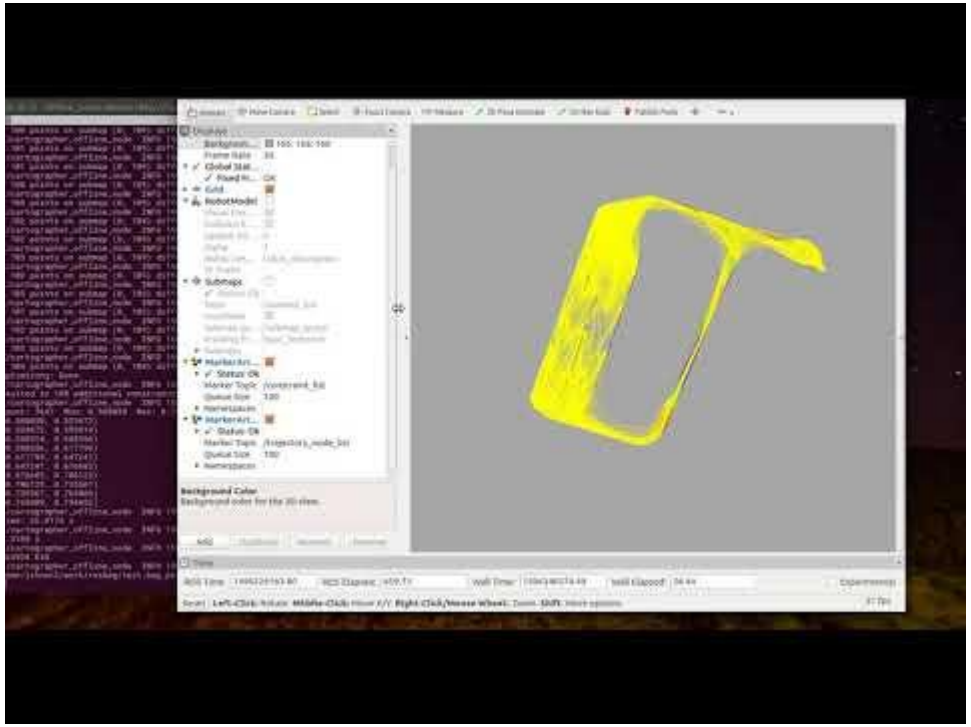
- 도이치박물관 매핑
  - 약 30분, 2.2km의 trajectory 분량의 데이타
  - 6분이내 최적화된 맵 생성 가능.
  - 실시간보다 약 5.3배 빠르다고 주장
  - Intel Xeon E5-1650 at 3.2GHz, 2.2G mem
- Graph FLIRT[9]와 Graph Mapping[21] 와 비교
  - Table 2, 3
- 다른 데이터셋내에서 결과 공개
  - Table 4: Loop closure detection의 갯수및 정확도
  - Table 5: 데이터의 시간길이와 Cartographer로 최적화할때 걸리는 시간

[9] G.D.Tipaldi,M.Braun,andK.O.Arras,"FLIRT:Interestregionsfor 2D range data with applications to robot navigation," in *Experimental Robotics*. Springer, 2014, pp. 695–710

[21] R. Ku¨mmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner, "On measuring the accuracy of SLAM algorithms," *Autonomous Robots*, vol. 27, no. 4, pp. 387–407, 2009.

Demo - 도이치박물관, 물류창고





https://www.facebook.com/jihoonlee.in/posts/10159483687145578

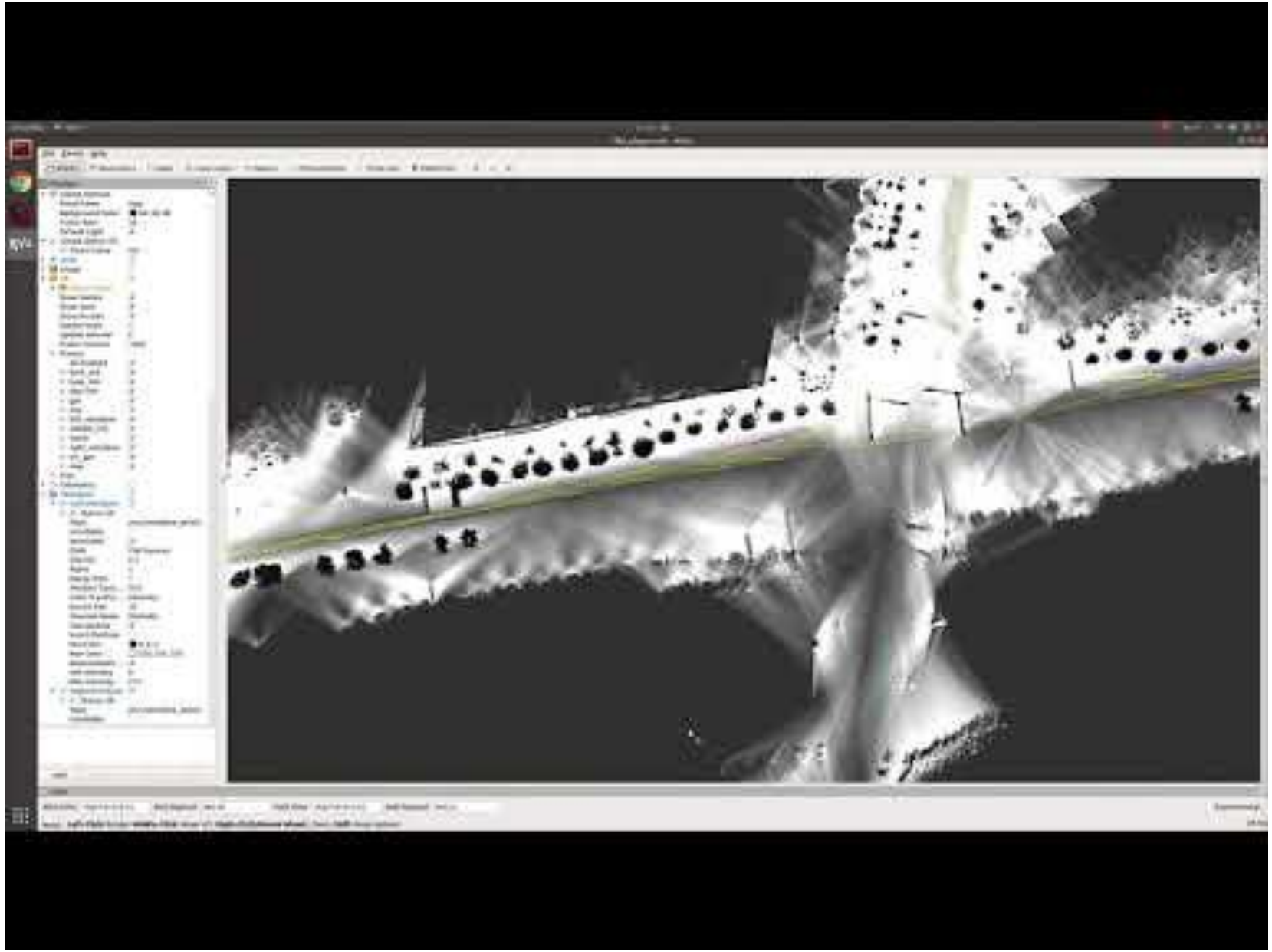
Demo

ROSCon 2018 Madrid: Cloud-based Mapping and Localization in Dynamic Warehouse Environments

https://vimeo.com/293260413#t=690s

Demo - KAIST Urban Dataset

https://irap.kaist.ac.kr/dataset/example.html



Cartographer as open source

Beyond the paper - Open source Developments

Cartographer as open source

Beyond the paper - Open source Developments

- ROS independent
  - 모듈 자체는 독립적이다. 하지만 *cartographer_ros*와 같은 인터페이스 구현을 직접 해줘야한다.
- 단일로봇의 다중 센서 지원
  - N개의 Pointcloud, N개의 Rangefinder, N개의 LaserScan 지원
  - Odometry, IMU, GPS, Landmark observation지원
- 다중 로봇 지원
- 구글 레벨의 잘 정리되고 최적화된 코드
- 인도어 로봇부터 자율주행차까지 사용 가능.

- 3D Mapping: 단 Mobile 로봇 한정.
- Localization only mode

--- 여기까지는 확실하게 직접 테스트해본 부분.

- Life-long mapping and localization
- Cloud Mapping
- TSDF(Truncated Signed Distance Function) Map support

--- 여기는 구글 cartographer 개발팀과 전직장동료에게 들은 내용 및 간단한 테스트 정도만 진행해봄.

Cartographer as open source

Beyond the paper - Open source Developments

- ROS independent
  - 모듈 자체는 독립적이다. 하지만 *cartographer_ros*와 같은 인터페이스 구현을 직접 해줘야한다.
- 단일로봇의 다중 센서 지원
  - N개의 Pointcloud, N개의 Rangefinder, N개의 LaserScan 지원
  - Odometry, IMU, GPS, Landmark observation지원
- 다중 로봇 지원
- 구글 레벨의 잘 정리되고 최적화된 코드
- 인도어 로봇부터 자율주행차까지 사용 가능.

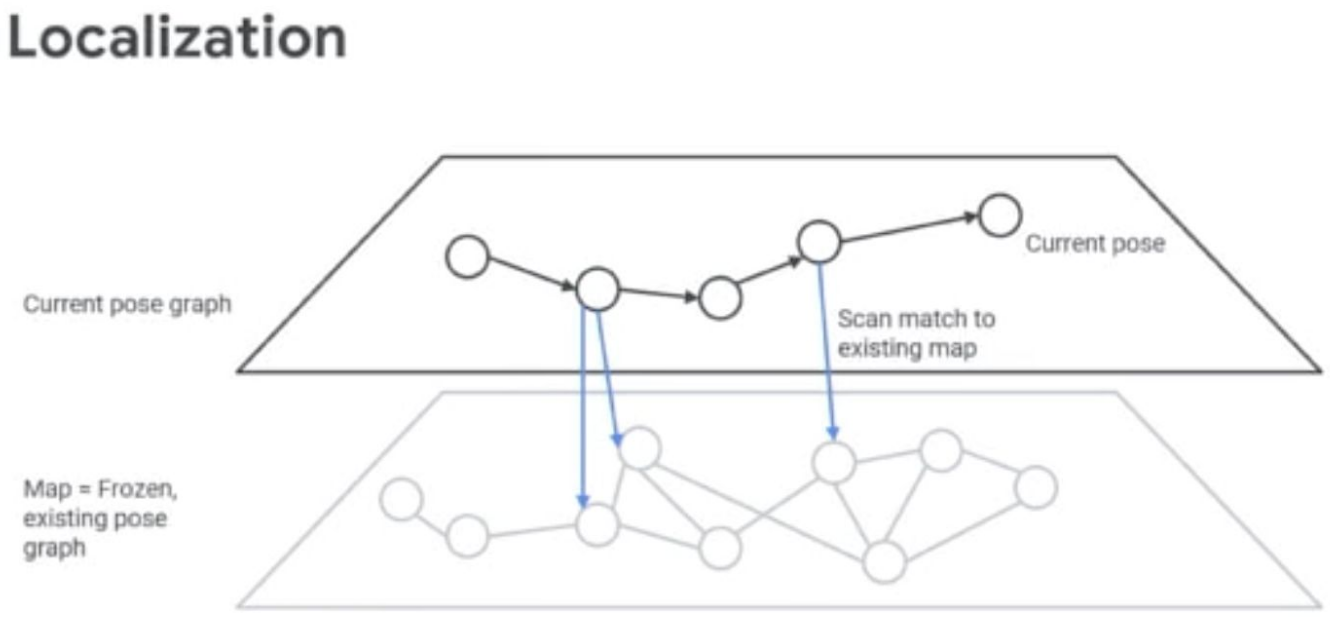
- 3D Mapping: 단 Mobile 로봇 한정.
- Localization only mode

--- 여기까지는 확실하게 직접 테스트해본 부분.

- Life-long mapping and localization
- Cloud Mapping
- TSDF(Truncated Signed Distance Function) Map support

--- 여기는 구글 cartographer 개발팀과 전직장동료에게 들은 내용 및 간단한 테스트 정도만 진행해봄.

Localization



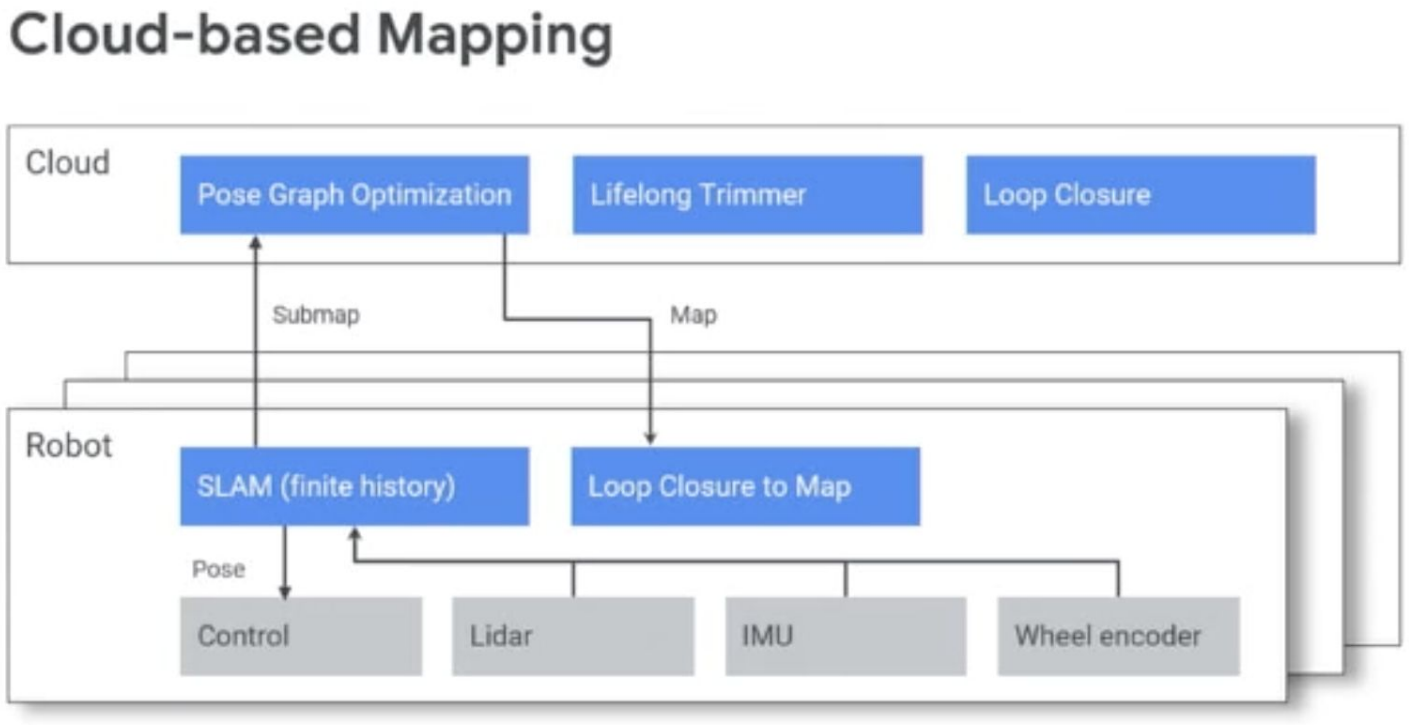
- realtime pose graph optimization을 통한 localization
- Frozen Pose graph(Map)와 현재 로봇이 만들고있는 Pose graph간의 inter trajectory constraint(또는 global constraint)를 지속적으로 찾아주는 방식
- 현재 생성되는 pose trajectory의 길이를 windowing방식으로 제한을 두어 실시간 최적화가 가능

```
TRAJECTORY_BUILDER = {
  trajectory_builder_2d = TRAJECTORY_BUILDER_2D,
  trajectory_builder_3d = TRAJECTORY_BUILDER_3D,
--  pure_localization_trimmer = {
--    max_submaps_to_keep = 3,
--  },
  collate_fixed_frame = true,
  collate_landmarks = false,
}
```

trajectory_builder.lua

ROSCon 2018 Madrid: Cloud-based Mapping and Localization in Dynamic Warehouse Environments

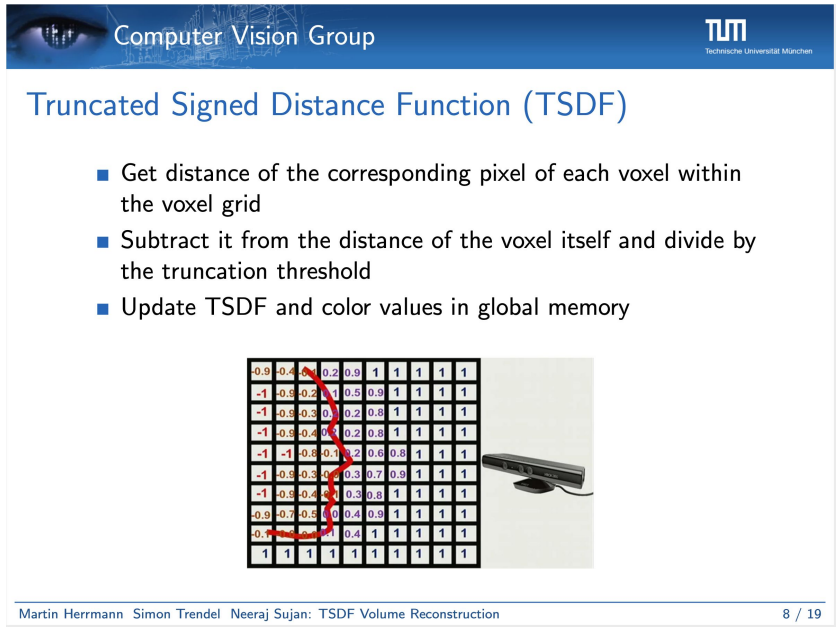
Cloud Mapping



- GRPC 기반
- 로봇은 localization mode처럼 한정된 trajectory만 유지
- Cloud에서 Map(Frozen trajectory)를 로봇에게 지속적으로 업데이트해주는 방식
- Lifelong submap Trimmer를 통해 overlap이 되는 submap은 지워줌
  -> Map의 전체 크기는 pose trajectory에 비례하는 것이
  아니라 공간의 크기에 비례

ROSCon 2018 Madrid: Cloud-based Mapping and Localization in Dynamic Warehouse Environments

TSDF Support



- Truncated Signed Distance Function기반 submap 생성 지원
- 3D Volume Reconstrunction할때 사용
- Probability Grid 방식보다 느리지만 더 정밀한 맵을 만들 수 있다.
- 단 튜닝이 까다롭다.

https://vision.in.tum.de/_media/teaching/ss2015/gpucourse_ss2015/kinectfusion.pdf
https://github.com/googlecartographer/cartographer_ros/issues/1058

Cartographer Demo 실행방법

Note

- Deb release 기준입니다.
- Master branch의 API는 다를수 있습니다.

Cartographer Demo 실행방법

https://google-cartographer-ros.readthedocs.io/en/latest/demos.html

Deutsches Museum

Download and launch the 2D backpack demo:

```
wget -P ~/Downloads https://storage.googleapis.com/cartographer-public-data/bags/backpack_2d/ca
roslaunch cartographer_ros demo_backpack_2d.launch bag_filename:=${HOME}/Downloads/cartographer
```

Download and launch the 3D backpack demo:

```
wget -P ~/Downloads https://storage.googleapis.com/cartographer-public-data/bags/backpack_3d/wi
roslaunch cartographer_ros demo_backpack_3d.launch bag_filename:=${HOME}/Downloads/b3-2016-04-0
```

Static landmarks

```
# Download the landmarks example bag.
wget -P ~/Downloads https://storage.googleapis.com/cartographer-public-data/bags/mir/landmar

# Launch the landmarks demo.
roslaunch cartographer_mir offline_mir_100_rviz.launch bag_filename:=${HOME}/Downloads/landm
```

Pure localization

Pure localization uses 2 different bags. The first one is used to generate the map, the second to run pure localization.

Download the 2D bags from the Deutsche Museum:

```
wget -P ~/Downloads https://storage.googleapis.com/cartographer-public-data/bags/backpack_2d/b2
wget -P ~/Downloads https://storage.googleapis.com/cartographer-public-data/bags/backpack_2d/b2
```

Generate the map (wait until cartographer_offline_node finishes) and then run pure localization:

```
roslaunch cartographer_ros offline_backpack_2d.launch bag_filenames:=${HOME}/Downloads/b2-2016-
roslaunch cartographer_ros demo_backpack_2d_localization.launch \
   load_state_filename:=${HOME}/Downloads/b2-2016-04-05-14-44-52.bag.pbstream \
   bag_filename:=${HOME}/Downloads/b2-2016-04-27-12-31-41.bag
```

Download the 3D bags from the Deutsche Museum:

```
wget -P ~/Downloads https://storage.googleapis.com/cartographer-public-data/bags/backpack_3d/b3
wget -P ~/Downloads https://storage.googleapis.com/cartographer-public-data/bags/backpack_3d/b3
```

Generate the map (wait until cartographer_offline_node finishes) and then run pure localization:

```
roslaunch cartographer_ros offline_backpack_3d.launch bag_filenames:=${HOME}/Downloads/b3-2016-
roslaunch cartographer_ros demo_backpack_3d_localization.launch \
   load_state_filename:=${HOME}/Downloads/b3-2016-04-05-13-54-42.bag.pbstream \
   bag_filename:=${HOME}/Downloads/b3-2016-04-05-15-52-20.bag
```

Cartographer Demo 실행방법 - Turtlebot 3

https://emanual.robotis.com/docs/en/platform/turtlebot3/slam/#run-slam-node

- **Cartographer** (ROS WIKI, Github)

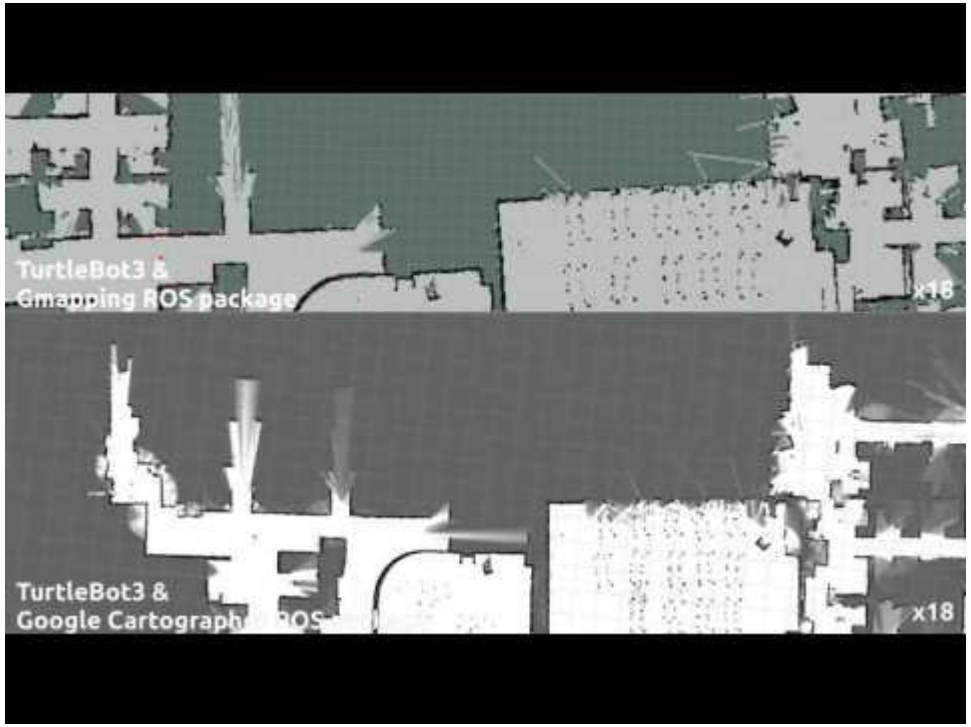
1. Download and build packages on PC.

   The Cartographer package developed by Google supports ROS1 Kinetic with 0.2.0 version. So if you need to use Cartogrpher on Kinetic, you should download and build the source code as follows instead of installing with the binary packages. Please refer to official wiki page for more detailed installation instructions.

```
$ sudo apt-get install ninja-build libceres-dev libprotobuf-dev protobuf-compiler libprotoc-dev
$ cd ~/catkin_ws/src
$ git clone https://github.com/googlecartographer/cartographer.git
$ git clone https://github.com/googlecartographer/cartographer_ros.git
$ cd ~/catkin_ws
$ src/cartographer/scripts/install_proto3.sh
$ rm -rf protobuf/
$ rosdep install --from-paths src --ignore-src -r -y --os=ubuntu:xenial
$ catkin_make_isolated --install --use-ninja
```

2. Launch the Cartographer SLAM node.

```
$ source ~/catkin_ws/install_isolated/setup.bash
$ roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=cartographer
```



Cartographer - Mapping launch 파일 설정

```
<launch>
  <!-- Urdf 설정 -->
  <param name="robot_description"
    textfile="$(find cartographer_ros)/urdf/backpack_2d.urdf" />

  <!-- /tf 추가 설정 -->
  <node name="robot_state_publisher" pkg="robot_state_publisher"
    type="robot_state_publisher" />


  <!-- cartographer node -->
  <node name="cartographer_node" pkg="cartographer_ros"
      type="cartographer_node" args="
          -configuration_directory $(find cartographer_ros)/configuration_files
          -configuration_basename backpack_2d.lua"
      output="screen">
    <!-- Multi echo Laser Scan 1개일때 -->
    <remap from="echoes" to="horizontal_laser_2d" />

    <!-- Multi echo laser scan 1개 이상일때
    <remap from="echoes_1" to="echo_laser_2d_1" />
    <remap from="echoes_2" to="echo_laser_2d_2" />
    -->

    <!-- 만약 1개 이상의 pointcloud일때
    <remap from="points2_1" to="horizontal_laser_3d" />
    <remap from="points2_2" to="vertical_laser_3d" />
    -->
  </node>

  <!-- Cartographer에서 나오는 map 포맷을 nav_msgs/OccupancyGrid 로 바꿔주는 노드 -->
  <node name="cartographer_occupancy_grid_node" pkg="cartographer_ros"
      type="cartographer_occupancy_grid_node" args="-resolution 0.05" />
</launch>
```

```
<launch>
  <param name="robot_description"
    textfile="$(find cartographer_ros)/urdf/backpack_3d.urdf" />

  <node name="robot_state_publisher" pkg="robot_state_publisher"
    type="robot_state_publisher" />

  <node name="cartographer_node" pkg="cartographer_ros"
      type="cartographer_node" args="
          -configuration_directory $(find cartographer_ros)/configuration_files
          -configuration_basename backpack_3d.lua"
      output="screen">
    <remap from="points2_1" to="horizontal_laser_3d" />
    <remap from="points2_2" to="vertical_laser_3d" />
  </node>

  <node name="cartographer_occupancy_grid_node" pkg="cartographer_ros"
      type="cartographer_occupancy_grid_node" args="-resolution 0.05" />
</launch>
```

타겟 로봇과 센서 설정없이는 모든걸 설명하기가 어렵기에 큰 틀에서만 설명할게요.

launch파일에서 설정해줘야할 중요한 것들

(로봇에 필요한 정보들 /tf, URDF등은 이미 있다고 가정했을때)

- configuration_directory + basename
  - cartographer를 설정해주는 lua파일 위치

- 각 센서 <remap> 들
  - _<N>이 붙으면 여러개 사용한다는 것
  - (odom, imu 없어도 가능하지만 있다면)
  - odom
  - imu

- cartographer_occupance_grid_node
  - Cartographer는 ros map대신 submap이라는 다른 포맷을 줍니다. 기본 navigation stack을 사용하기 위해 submap을 nav_msgs/OccupancyGrid로 바꿔주는 노드

- 3D모드에선 imu가 필수

Cartographer - Configuration lua 파일 설정

[터틀봇 depth camera]

```
include "map_builder.lua"
include "trajectory_builder.lua"

options = {
  map_builder = MAP_BUILDER,
  trajectory_builder = TRAJECTORY_BUILDER,
  map_frame = "map",
  tracking_frame = "gyro_link",
  published_frame = "odom",
  odom_frame = "odom",
  provide_odom_frame = false,
  publish_frame_projected_to_2d = false,
  use_odometry = true,
  use_nav_sat = false,
  use_landmarks = false,
  num_laser_scans = 1,
  num_multi_echo_laser_scans = 0,
  num_subdivisions_per_laser_scan = 1,
  num_point_clouds = 0,
  lookup_transform_timeout_sec = 0.2,
  submap_publish_period_sec = 0.3,
  pose_publish_period_sec = 5e-3,
  trajectory_publish_period_sec = 30e-3,
  rangefinder_sampling_ratio = 1.,
  odometry_sampling_ratio = 1.,
  fixed_frame_pose_sampling_ratio = 1.,
  imu_sampling_ratio = 1.,
  landmarks_sampling_ratio = 1.,
}

MAP_BUILDER.use_trajectory_builder_2d = true

TRAJECTORY_BUILDER_2D.min_range = 0.1
TRAJECTORY_BUILDER_2D.max_range = 4.0
TRAJECTORY_BUILDER_2D.missing_data_ray_length = 2.0
TRAJECTORY_BUILDER_2D.use_imu_data = true
TRAJECTORY_BUILDER_2D.motion_filter.max_time_seconds
TRAJECTORY_BUILDER_2D.motion_filter.max_distance_mete
TRAJECTORY_BUILDER_2D.motion_filter.max_angle_radians
TRAJECTORY_BUILDER_2D.ceres_scan_matcher.translation_
TRAJECTORY_BUILDER_2D.ceres_scan_matcher.rotation_wei
```

[백팩 데모 3D]

```
include "map_builder.lua"
include "trajectory_builder.lua"

options = {
  map_builder = MAP_BUILDER,
  trajectory_builder = TRAJECTORY_BUILDER,
  map_frame = "map",
  tracking_frame = "base_link",
  published_frame = "base_link",
  odom_frame = "odom",
  provide_odom_frame = true,
  publish_frame_projected_to_2d = false,
  use_odometry = false,
  use_nav_sat = false,
  use_landmarks = false,
  num_laser_scans = 0,
  num_multi_echo_laser_scans = 0,
  num_subdivisions_per_laser_scan = 1,
  num_point_clouds = 2,
  lookup_transform_timeout_sec = 0.2,
  submap_publish_period_sec = 0.3,
  pose_publish_period_sec = 5e-3,
  trajectory_publish_period_sec = 30e-3,
  rangefinder_sampling_ratio = 1.,
  odometry_sampling_ratio = 1.,
  fixed_frame_pose_sampling_ratio = 1.,
  imu_sampling_ratio = 1.,
  landmarks_sampling_ratio = 1.,
}

TRAJECTORY_BUILDER_3D.num_accumulated_range_data = 160

MAP_BUILDER.use_trajectory_builder_3d = true
MAP_BUILDER.num_background_threads = 7
POSE_GRAPH.optimization_problem.huber_scale = 5e2
POSE_GRAPH.optimize_every_n_nodes = 320
POSE_GRAPH.constraint_builder.sampling_ratio = 0.03
POSE_GRAPH.optimization_problem.ceres_solver_options.max_num_iterations = 10
POSE_GRAPH.constraint_builder.min_score = 0.62
POSE_GRAPH.constraint_builder.global_localization_min_score = 0.66

return options
```

launch 파일이상으로 중요한 설정파일.

- 로봇의 odometry의 유무에 따라서 설정이 달라지고
- imu의 유무에 따라서 달라짐

- 여기서 설정한 센서의 갯수만큼 launch 파일의 remap이 적용됨.
  - num_laser_scans
  - num_multi_echo….
  - num_point_clouds
- PC의 성능과 로봇의 센서 성능에 따라서 설정을 조화롭게 맞춰줘야 제대로된 결과를 만들수있음.
- 가장 가장 가장 중요한건 로봇 센서 데이터의 정확성. 이게 제대로 안되있으면 아래 파라메터들 백날 건드려도 무의미합니다.

options.num_subdiviions_per_laser_scan:
laser scan 센서 frequency가 떨어지고 로봇의 속도가 빠를때 키워주면 좋음.
너무 잘게 쪼개도 무의미함. 그리고 늘릴수록 속도 느려짐.

TRAJECTORY_BUILDER_2D.scans_per_accumulation:
options.num_laser_scan * options.num_subdivisions_per_laser_scan.

TRAJECTORY_BUILDER_2D.use_online_correlative_scan_matching
속도가 느려지는대신 local map의 정확도가 좋아짐. imu나 odom이 없을때는 필수적으로
사용해야하며 사용한게 항상 결과는 좋음.

TRAJECTORY_BUILDER_2D.real_time_correlative_scan_matcher parameters:
local map의 정확도를 제어하고 싶을때 변경하는 파라메터들

TRAJECTORY_BUILDER_2D.submaps.num_range_data:
local map들에 들어가는 scan의 갯수를 정하는 파라메터. odometry error가 크다면 줄이면
이득

SPARSE_POSE_GRAPH.optimize_every_n_scans:
backend 최적화를 얼마나 자주할지 결정. PC 속도에 따라서 결정하는게 좋음.

SPARSE_POSE_GRAPH.constraint_builder.fast_correlative_scan_matcher:
backend 최적화할때 loop closure 탐색을 좀더 잘하게 만들고 싶을때 건드는 파라메터.

Cartographer - Localization Mode 설정

```
<launch>
  <param name="/use_sim_time" value="true" />

  <param name="robot_description"
    textfile="$(find cartographer_ros)/urdf/backpack_2d.urdf" />

  <node name="robot_state_publisher" pkg="robot_state_publisher"
    type="robot_state_publisher" />

  <node name="cartographer_node" pkg="cartographer_ros"
      type="cartographer_node" args="
          -configuration_directory $(find cartographer_ros)/configuration_files
          -configuration_basename backpack_2d_localization.lua
          -load_state_filename $(arg load_state_filename)"
      output="screen">
    <remap from="echoes" to="horizontal_laser_2d" />
  </node>

  <node name="cartographer_occupancy_grid_node" pkg="cartographer_ros"
      type="cartographer_occupancy_grid_node" args="-resolution 0.05" />

  <node name="rviz" pkg="rviz" type="rviz" required="true"
      args="-d $(find cartographer_ros)/configuration_files/demo_2d.rviz" />
  <node name="playbag" pkg="rosbag" type="play"
      args="--clock $(arg bag_filename)" />
</launch>
```

- 일반 설정과 거의 동일
- 대신 시작할때 localization설정이 들어간 lua파일 사용과 이전에 매핑해놓은 state를 load_state_filename 파라메터로 불러와야합니다.
- state file은 이전 실행에서 매핑을 완료후에 /write_state 서비스로 생기는 pbstream 파일입니다.

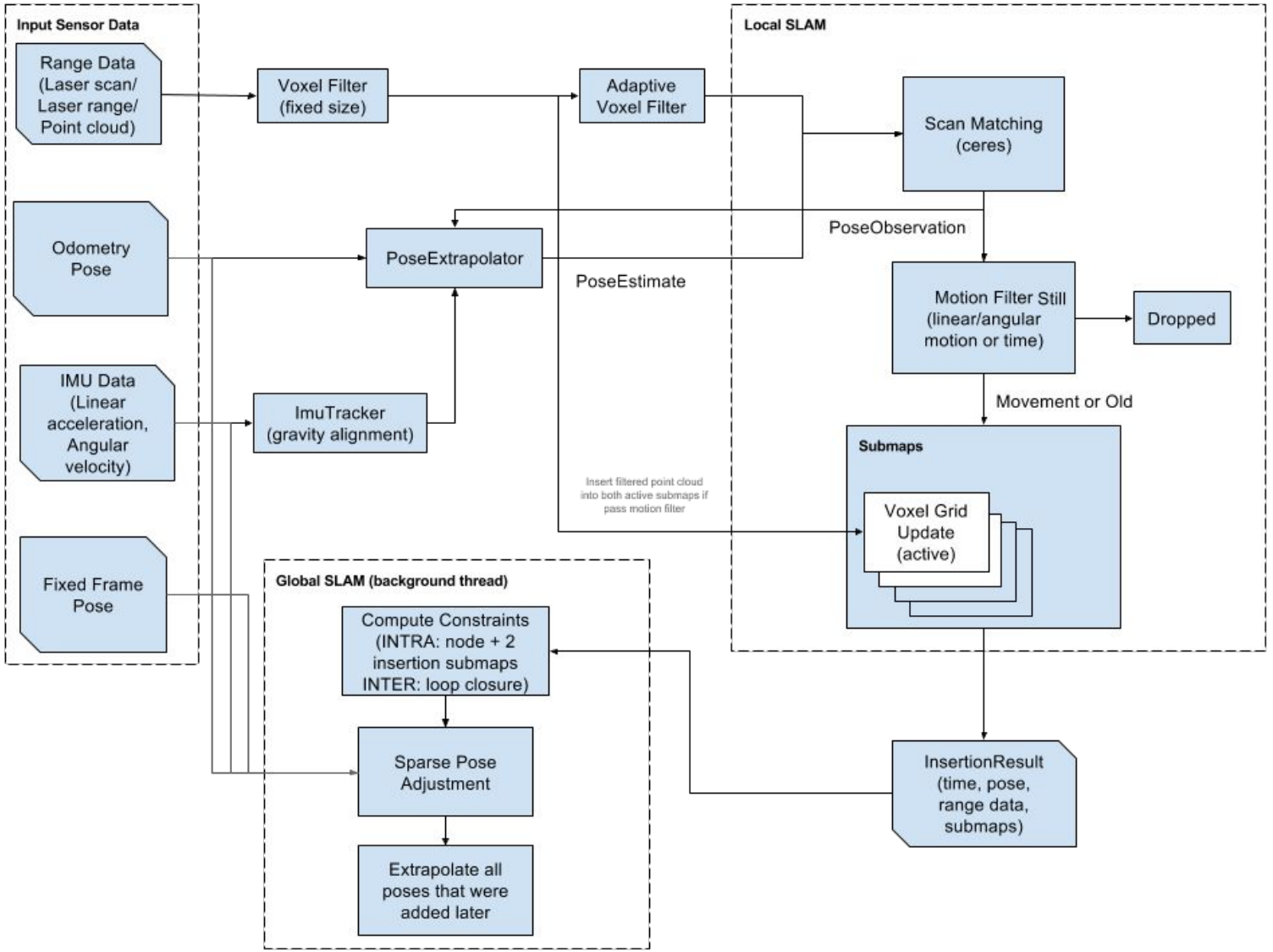
```
include "backpack_2d.lua"

TRAJECTORY_BUILDER.pure_localization = true
POSE_GRAPH.optimize_every_n_nodes = 20

return options
```

- Mapping때와 거의 동일하지만 puer_localization=true 맞춰주고
- optimize_envery_n_nodes를 mapping때보다 더 낮은 값을 줍니다.

Cartographer - 이외 실행 가능한 노드들

- cartographer_node: 이전 슬라이드에서 설명한 노드
- cartographer_offline_node: cartographer를 rosbag에서 실행할때 실시간이 아닌 훨씬 빠른 속도로 실행을 할때 쓰는 노드
- cartographer_assets_writer: rosbag과 생성된 state로 3차원 맵과 같은 포맷을 만들때 쓰는 노드
- cartographer_rosbag_validate: rosbag 데이터를 훑어서 cartographer에서 실행해도 문제가 없는 rosbag인지 체크해주는 노드

Q & A


Input Sensor Data
Range Data (Laser scan/ Laser range/ Point cloud)
Odometry Pose
IMU Data (Linear acceleration, Angular velocity)
Fixed Frame Pose
Voxel Filter (fixed size)
Adaptive Voxel Filter
PoseExtrapolator
ImuTracker (gravity alignment)
PoseEstimate
Insert filtered point cloud into both active submaps if pass motion filter
Local SLAM
Scan Matching (ceres)
PoseObservation
Motion Filter Still (linear/angular motion or time)
Dropped
Movement or Old
Submaps
Voxel Grid Update (active)
Global SLAM (background thread)
Compute Constraints (INTRA: node + 2 insertion submaps INTER: loop closure)
Sparse Pose Adjustment
Extrapolate all poses that were added later
InsertionResult (time, pose, range data, submaps)