cartographer_ros ガイド

by nnn112358

About me

nnn112358@twitter

anoken 2017@twitter

https://anoken.jimdo.com/

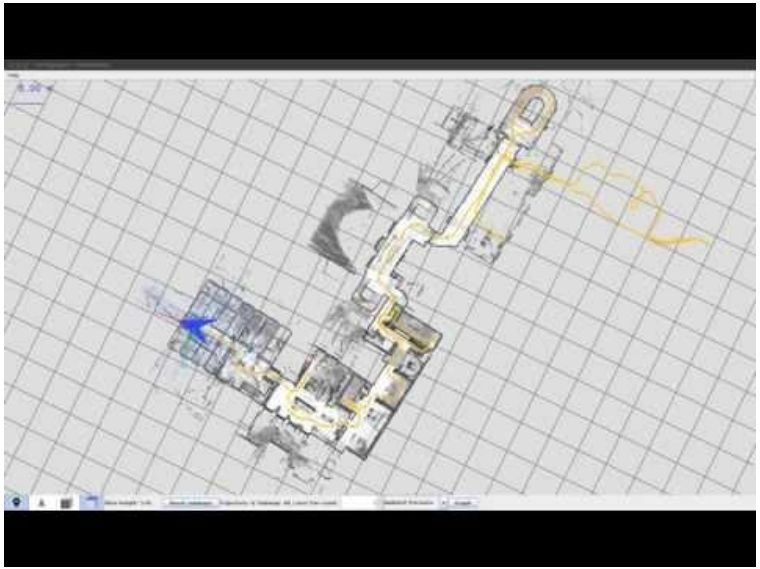
Developping sado robotics .
coming soon...!!!

Abstoraction

- cartographerとは？
- cartographerの仕組み
- cartographer_rosのインストール
- cartographerを動かす

cartographerとは？

googleが2016年10月にOpenSourceでリリースした、Realtime SLAM(simultaneous localization and mapping) software。LIDAR, or IMU ,or cameraを使って、地図を作る。
ROSのInterfaceが用意されている。



cartographer document
https://github.com/googlecartographer/cartographer

cartographer_ros document
https://github.com/googlecartographer/cartographer_ros

ROSConで発表があった。

ROSCon 2016 Lightning Talk Google Releases Cartographer
https://www.youtube.com/watch?v=OKRSmFHo2oU

ROSCon 2017 SLAM on Turtlebot2 using ROS2
https://vimeo.com/236172294

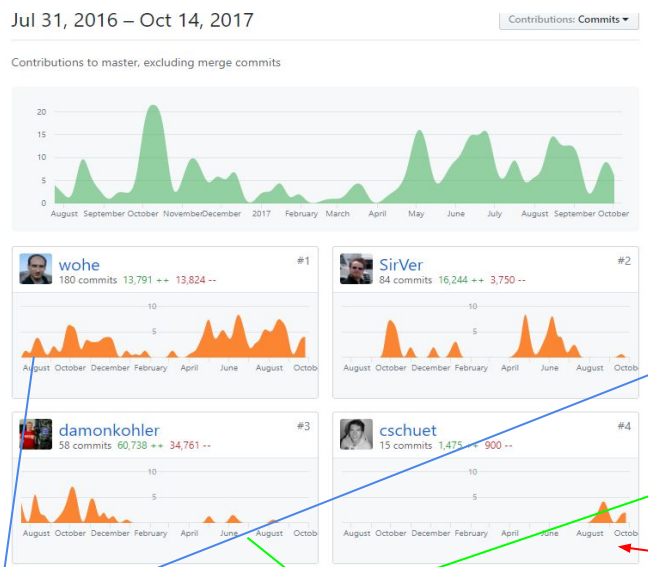
要求Specは少々高め。

- 64-bit, modern CPU (e.g. 3rd generation i7)
- 16 GB RAM
- Ubuntu 14.04 (Trusty) and 16.04 (Xenial)
- gcc version 4.8.4 and 5.4.0

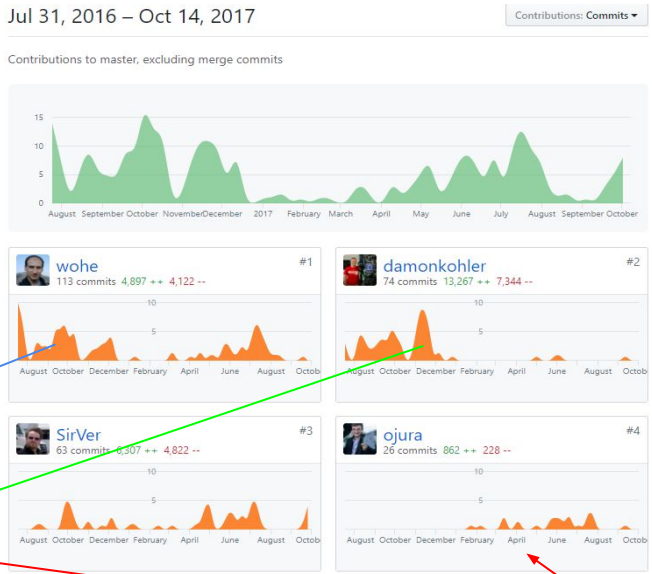
cartographerとは？

現在も頻繁にCommitされており、開発が進んでいる。（Updateすると同じ設定で動かないことが多々。。。）

github：cartographer



github：cartographer_ros

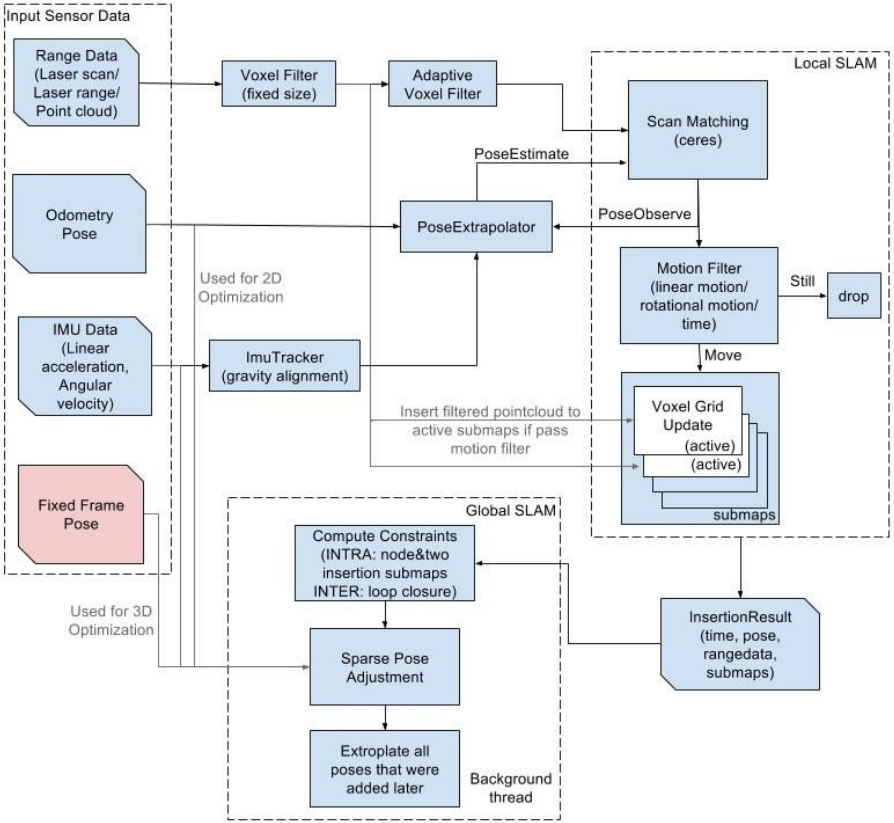


Wolfgang Hess Damon Kohler Holger Rapp Daniel Andor

Real-Time Loop Closure in 2D LIDAR SLAM
2016 IEEE International Conference on Robotics and Automation (ICRA), pp. 1271-1278

newface??

cartographerの仕組み

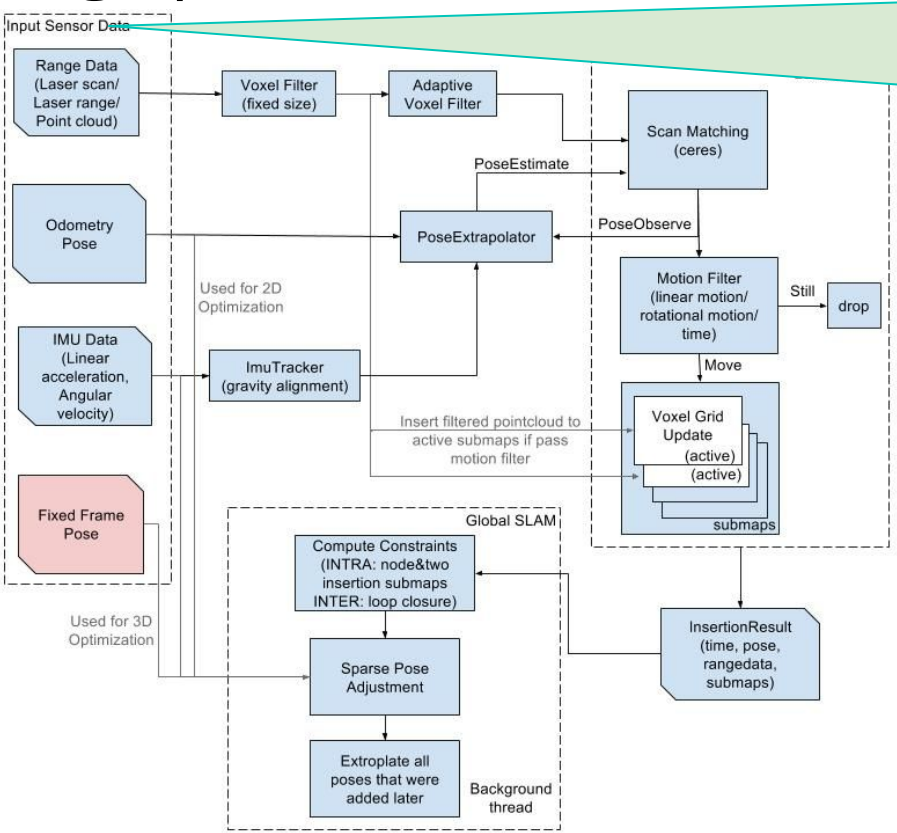


https://google-cartographer.readthedocs.io/en/latest/

cartographer document
https://github.com/googlecartographer/cartographer

cartographer_ros document
https://github.com/googlecartographer/cartographer_ros

cartographerの仕組み



https://google-cartographer.readthedocs.io/en/latest/

・Lidar
・Odometory（なくても良い）
・IMU（水平に設置されいるなら、なくても良い）
・センサの取り付け位置
をセンサーから入力する

センサーの種類・位置関係を設定する。
ROSからのInterfaceが用意されている。(scan/odom/tf/imu)

fetch robotics

Odom+Lidar

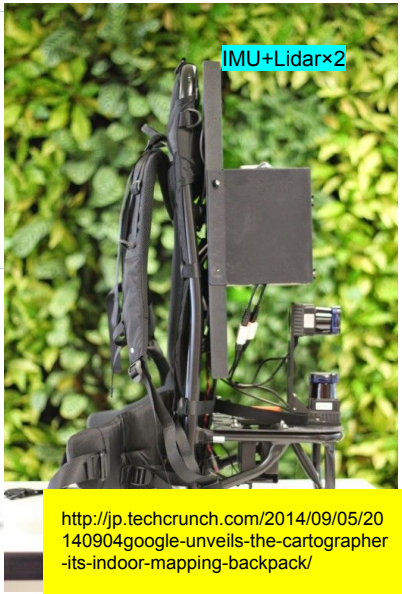
http://fetchrobotics.com/

TurtleBot2

Open Robotics platform designed for education & research on state of robotics.

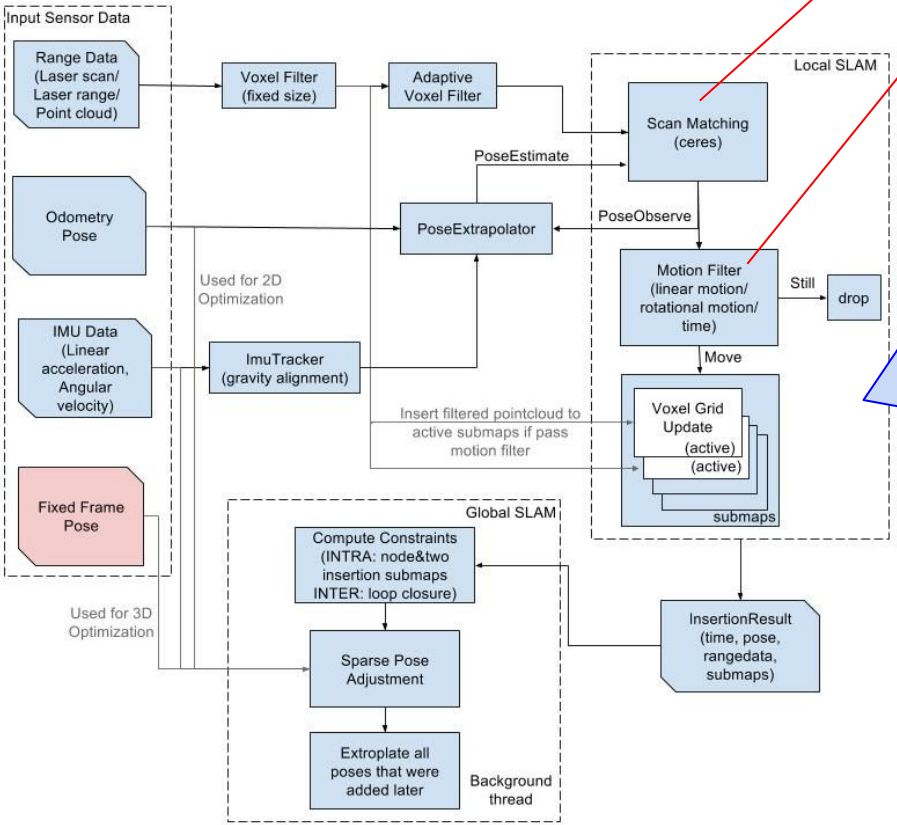
Odom+Kinect

http://www.nihonbinary.co.jp/Products/Robot/TurtleBot.html



http://jp.techcrunch.com/2014/09/05/20140904google-unveils-the-cartographer-its-indoor-mapping-backpack/

cartographerの仕組み



https://google-cartographer.readthedocs.io/en/latest/

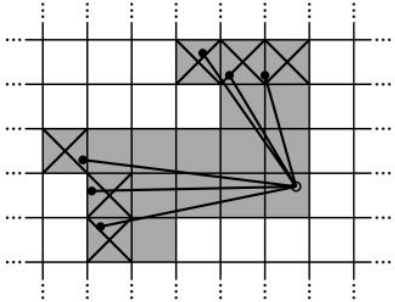
https://github.com/googlecartographer/cartographer/blob/master/cartographer/mapping_2d/scan_matching/ceres_scan_matcher.cc

https://github.com/googlecartographer/cartographer/blob/master/cartographer/mapping_3d/motion_filter.cc

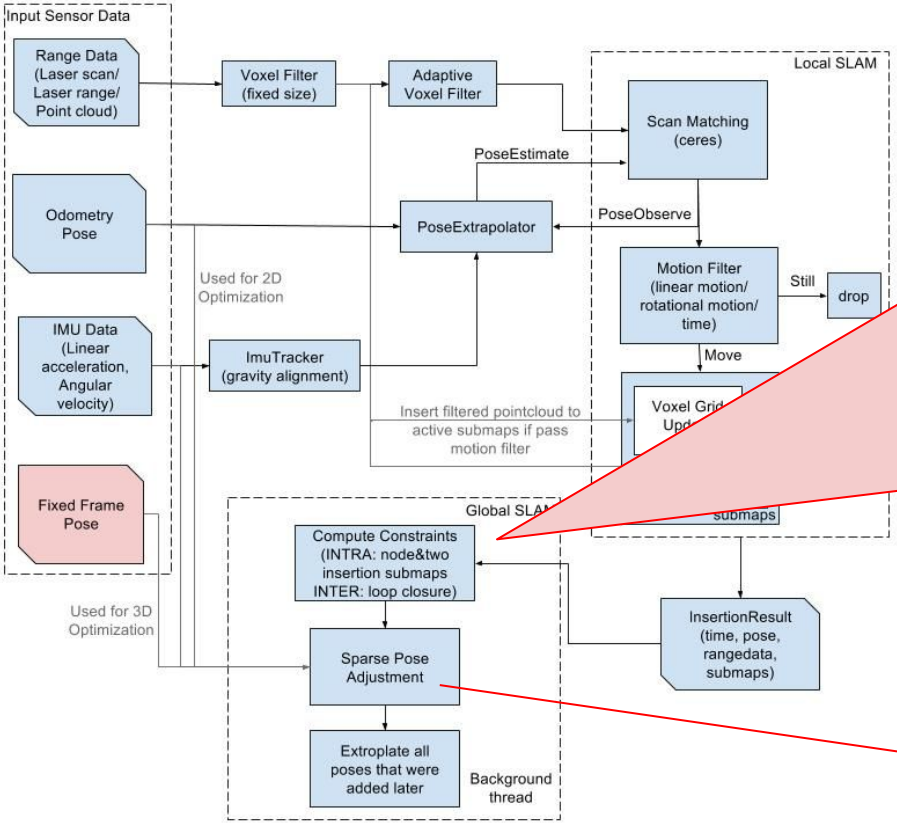
Local SLAM

CeresScanMatcherで、Lidarのスキャンとsubmapとが整合とれる最もよい場所を探索する。
submapと、scanのサブピクセル位置合わせと
GoogleのC++最適化ライブラリCeres Solverを利用。

MotionFilterは3D時のみ。



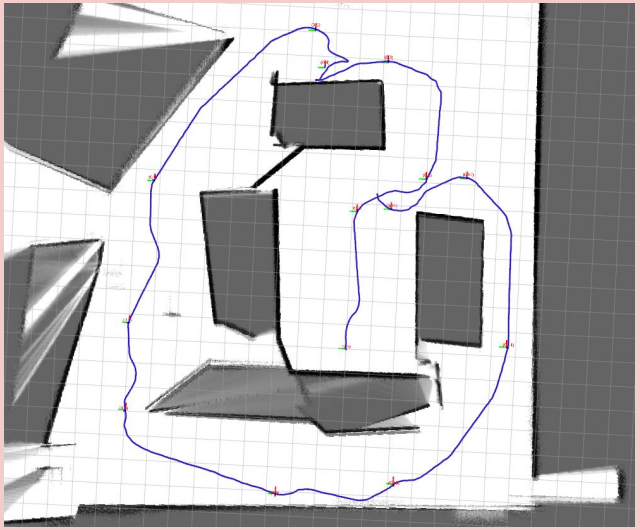
cartographerの仕組み



https://google-cartographer.readthedocs.io/en/latest/

Global SLAM

Loop Closureによる最適化。
こちらも、GoogleのC++最適化ライブラリCeres Solverを利用



https://github.com/googlecartographer/cartographer/blob/master/cartographer/mapping_2d/sparse_pose_graph.cc

cartographer_rosのインストール

cartographer_rosを、ドキュメントにしたがって、インストールする。

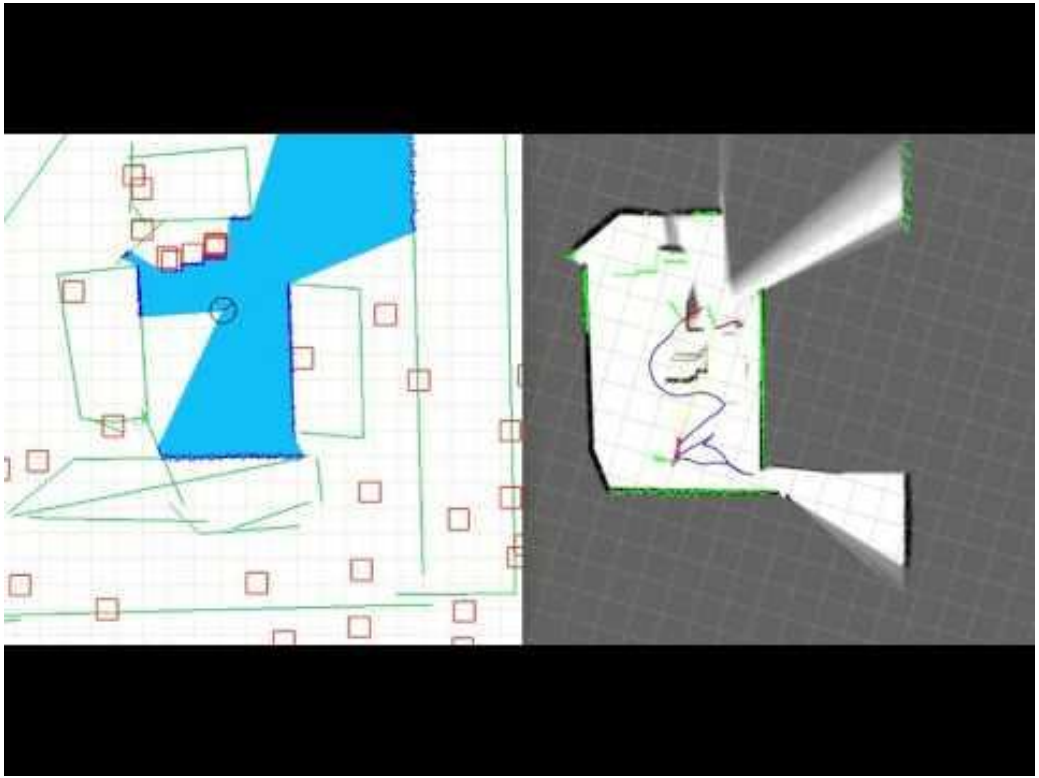
https://google-cartographer-ros.readthedocs.io/en/latest/

```
$ sudo apt-get update
$ sudo apt-get install -y python-wstool python-rosdep ninja-build

$ mkdir catkin_ws
$ cd catkin_ws
$ wstool init src

$ wstool merge -t src
https://raw.githubusercontent.com/googlecartographer/cartographer_ros/master/cartographer_ros.rosinstall
$ wstool update -t src

$ sudo rosdep init
$ rosdep update
$ rosdep install --from-paths src --ignore-src --rosdistro  =${ROS_DISTRO} -y

$ catkin_make_isolated --install --use-ninja
$ source install_isolated/setup.bash
```

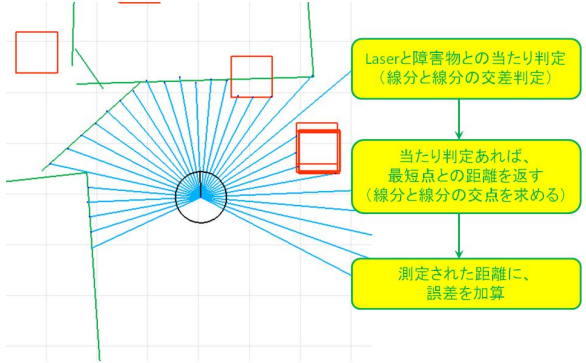
ROSBAGで動作確認する。

```
$ wget -P ~/Downloads https://storage.googleapis.com/cartographer-public-data/bags/revo_lds/cartographer_paper_revo_lds.bag
$ roslaunch cartographer_ros demo_revo_lds.launch bag_filename=${HOME}/Downloads/cartographer_paper_revo_lds.bag
```

cartographer_rosを動かす

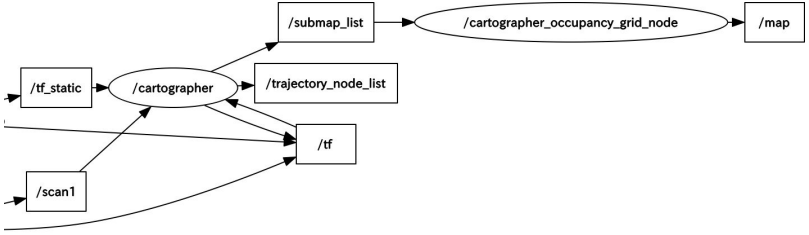


・移動障害物があっても、Mapが綺麗につくれている。
・Loop CloserでMapが最適化されていく。



Reference：
ROS JAPAN UG #11 シミュレーションで SLAMを試す
https://gitpitch.com/nnn112358/rosjp_170719

cartographer_rosを動かす



2Dで使う(移動ロボットで使う)のであれば、gmappingとほとんど同じ。
scan topic / tf (map->odom->base_link->base_footpoint)
を用意すればよい。

cartographerはlaunchファイルで起動する。
パラメータはluaファイルで設定する。

2D-Slamするなら、cartographer_fetchのfreightのサンプル
を参考がオススメ。
https://github.com/googlecartographer/cartographer_fetch

freight.launch

```
<launch>
  <node name="cartographer" pkg="cartographer_ros"
      type="cartographer_node" args="
          -configuration_directory $(find cartographer_fetch)/configuration_files
          -configuration_basename freight.lua"
      output="screen">
    <remap from="scan" to="base_scan" />
  </node>

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

freight.lua

```
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 = "odom",
  odom_frame = "odom",
  provide_odom_frame = false,
  use_odometry = true,
  num_laser_scans = 1,
  num_multi_echo_laser_scans = 0,
  num_subdivisions_per_laser_scan = 30,
  num_point_clouds = 0,
  lookup_transform_timeout_sec = 0.1,
  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.,
  imu_sampling_ratio = 1.,
}
```

cartographer_rosを動かす

freight.launch

```
<launch>
  <node name="cartographer" pkg="cartographer_ros"
      type="cartographer_node" args="
          -configuration_directory $(find cartographer_fetch)/configuration_files
          -configuration_basename freight.lua"
      output="screen">
    <remap from="scan" to="base_scan" />
  </node>

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

luaファイルディレクトリ
luaファイル名

ROS形式の MAPを発行

cartographer_rosを動かす

freight.lua

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

options = {
  map_builder = MAP_BUILDER,
  trajectory_builder = TRAJECTORY_BUILDER,
  map_frame = "map",                          //tfのmap
  tracking_frame = "base_link",               //tfのbase_link
  published_frame = "odom",                   //tfのodom
  odom_frame = "odom",                        //tfのodom
  provide_odom_frame = false,                 //odomを発行するか？
  use_odometry = true,                        //odometoryを使うか？
  num_laser_scans = 1,                        //Lidarの数(水平or水平＆垂直)
  num_multi_echo_laser_scans = 0,             //multi_echo対応のLidarの数
  num_subdivisions_per_laser_scan = 30        //Lidarのscanの分解能
  num_point_clouds = 0,                       //Lidarでない点群でデータ入力する場合
  lookup_transform_timeout_sec = 0.1,         //tfのタイムアウト時間
  submap_publish_period_sec = 0.3,            //地図を発行する周期
  pose_publish_period_sec = 5e-3,             //姿勢を発行する周期
  trajectory_publish_period_sec = 30e-3,      //経路を発行する周期
  rangefinder_sampling_ratio = 1.,            //Lidarの平滑化フィルタの係数
  odometry_sampling_ratio = 1.,               //odomtoryの平滑化フィルタの係数
  imu_sampling_ratio = 1.,                    //IMUの平滑化フィルタの係数
}
```

http://google-cartographer-ros.readthedocs.io/en/latest/configuration.html

```
MAP_BUILDER.use_trajectory_builder_2d = true
TRAJECTORY_BUILDER_2D.scans_per_accumulation = 30          //累積スキャン数
TRAJECTORY_BUILDER_2D.min_range = 0.                       //Lidarの最小距離
TRAJECTORY_BUILDER_2D.max_range = 20.                      //Lidarの最大距離
TRAJECTORY_BUILDER_2D.missing_data_ray_length = 5.         //空白を入れる距離
TRAJECTORY_BUILDER_2D.use_imu_data = true                  //IMUの有無

SPARSE_POSE_GRAPH.constraint_builder.min_score = 0.65      //sparse porseの閾値
SPARSE_POSE_GRAPH.optimization_problem.huber_scale = 3     //Huber損失関数の係数

SPARSE_POSE_GRAPH.optimization_problem.
consecutive_scan_translation_penalty_factor = 1e3          //並進運動ペナルティ係数

SPARSE_POSE_GRAPH.
optimization_problem.consecutive_scan_rotation_penalty_factor = 1e2 //回転運動ペナルティ係数

TRAJECTORY_BUILDER_2D.ceres_scan_matcher.occupied_space_weight = 10
                                                           //地図の更新比率
TRAJECTORY_BUILDER_2D.ceres_scan_matcher.rotation_weight = 40
                                                           //回転の更新比率
TRAJECTORY_BUILDER_2D.submaps.num_range_data = 120
                                                  //新規に追加する点群の数
TRAJECTORY_BUILDER_2D.motion_filter.max_distance_meters = 0.1
                                                  //更新するときの並進方向の閾値
TRAJECTORY_BUILDER_2D.motion_filter.max_angle_radians = math.rad(0.2)
                                                  //更新するときの回転方向の閾値

return options
```

https://google-cartographer.readthedocs.io/en/latest/configuration.html#

cartographer_rosを動かす

mapを保存する

1.**Assets writer**でmapを保存する（cartographer形式：pbstream）

2.map_saverでmapを保存する。(ROS標準：pgm/yaml)

1A.**Assets writerで保存する**：オンラインで保存

joystickなどで、robotを走行させた後で、別コンソールで以下を実行。

```
$ rosservice call /finish_trajectory 0                    //slamを終了する。
$ rosservice call /write_state your_filename.pbstream     //cartographer形式のMapを保存する。
```

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

1B.**Assets writerで保存する：オフライン** (rosbag）から保存。

```
$ roslaunch assets_writer_freight.launch \
      bag_filenames:=your_filename.bag \
      pose_graph_filename:=your_filename.pbstream
```

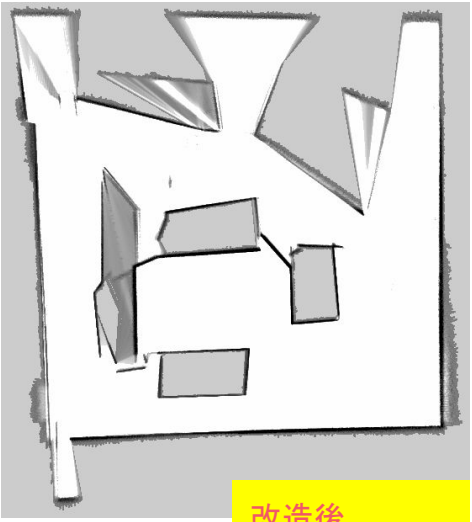
cartographer_rosを動かす

2.ROS形式(pgm/yaml)でmapを保存する。

$ rosrun map_server map_saver -f mymap

但し、cartographerとmap_saverで仕様の不一致があり、そのまま使うとMapデータが欠落する。
→map_saverを修正する。
amcl/move_baseで使いたいなら必須。

map_saverそのまま
→うまく保存できない



改造後

map_saver.cpp

```
for(unsigned int y = 0; y < map->info.height; y++) {
  for(unsigned int x = 0; x < map->info.width; x++) {
    unsigned int i = x + (map->info.height - y - 1) * map->info.width;
    if (map->data[i] == 0) { //occ [0,0.1)
      fputc(254, out);
    } else if (map->data[i] == +100) { //occ (0.65,1]
      fputc(000, out);
    } else { //occ [0.1,0.65]
      fputc(205, out);
    }
  }
}
```

ROS形式(0~100)とpgm(0~255)の変換を
map_saverは行っているが、
怪しい3値化になっている。

```
if (map->data[i] >= 0 && map->data[i] <=100) {
  unsigned int value = round((float)(100.0-map->data[i])*2.55);
  if (value == 128) {
    fputc(129, out);
  }
  else {
    fputc(value, out);
  }
}
else {
  fputc(128, out);
}
```

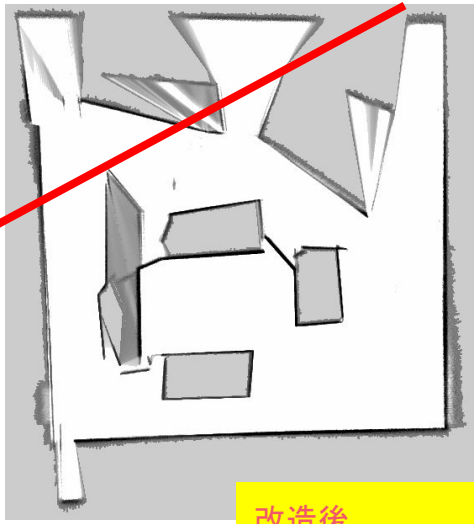
cartographer_rosを動かす

2.ROS形式(pgm/yaml)でmapを保存する。

$ rosrun map_server map_saver -f mymap

但し、cartographerとmap_saverで仕様の不一致があり、そのまま使うとMapデータが欠落する。
→map_saverを修正する。
amcl/move_baseで使いたいなら必須。

map_saverそのまま
→うまく保存できない



改造後

map_saver.cpp

```
for(unsigned int y = 0; y < map->info.height; y++) {
  for(unsigned int x = 0; x < map->info.width; x++) {
    unsigned int i = x + (map->info.height - y - 1) * map->info.width;
    if (map->data[i] == 0) { //occ [0,0.1)
      fputc(254, out);
    } else if (map->data[i] == +100) { //occ (0.65,1]
      fputc(000, out);
    } else { //occ [0.1,0.65]
      fputc(205, out);
    }
  }
}
```

ROS形式(0~100)とpgm(0~255)の変換を
map_saverは行っているが、
怪しい3値化になっている。

```
if (map->data[i] >= 0 && map->data[i] <=100) {
  unsigned int value = round((float)(100-0-map->data[i])*2.55);
  if (value == 128) {
    fputc(129, out);
  }
  else {
    fputc(value, out);
  }
}
else {
  fputc(128, out);
}
```

https://github.com/googlecartographer/cartographer_ros/issues/475
→ROS形式Map出力を検討している模様。いずれ解決しそう。

cartographer_rosを動かす

mapを保存する

1.**Assets writer**でmapを保存する（cartographer形式：pbstream）

2.map_saverでmapを保存する。(ROS標準：pgm/yaml)

3.今週Assets writerでmapを保存する(ROS標準：pgm/yaml)実装された!!

https://github.com/googlecartographer/cartographer_ros/issues/553

1B.**Assets writerで保存する：オフライン**（rosbag）から保存。

```
$ roslaunch assets_writer_freight.launch \
        bag_filenames:=your_filename.bag \
        pose_graph_filename:=your_filename.pbstream
```

assets_writer_freight.launch

```
<launch>
 <node name="cartographer_assets_writer" pkg="cartographer_ros"
required="true"
    type="cartographer_assets_writer" args="
       -configuration_directory
          $(find cartographer_fetch)/configuration_files
       -configuration_basename assets_test.lua
       -bag_filenames $(arg bag_filenames)
       -pose_graph_filename $(arg pose_graph_filename)"
    output="screen">
 </node>
</launch>
```

assets_test.lua

```
options = {
 tracking_frame = "base_laser_front_link",
 pipeline = {
  {     action = "min_max_range_filter",     min_range = 0.0,     max_range = 5.,   },
  {
    action = "write_ros_map",
    range_data_inserter = {
      insert_free_space = true,      hit_probability = 0.55,      miss_probability = 0.49,
    },
    filestem = "map",     resolution = 0.05,
  }
 }
}
```

cartographer_rosを動かす

mapを読み込む・自己位置推定(pure_localization)

mapを読み込んで、mapを追加できる。AMCLの代わりに自己位置推定に使える。

launchファイル

```
<launch>
 <param name="/use_sim_time" value="true" />
 <node name="cartographer" pkg="cartographer_ros"
    type="cartographer_node" args="
       -configuration_directory $(find cartographer_fetch)/configuration_files
         -configuration_basename freight_localization.lua
         -map_filename your_filename.pbstream
    output="screen">
   <remap from="scan" to="base_scan" />
 </node>
 <node name="rviz" pkg="rviz" type="rviz" required="true"
    args="-d $(find cartographer_fetch)/configuration_files/demo.rviz" />
 <node name="playbag" pkg="rosbag" type="play"
    args="--clock $(arg bag_filename)" />


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

LUAファイル

```
include "freight.lua"

TRAJECTORY_BUILDER.pure_localization = true
SPARSE_POSE_GRAPH.optimize_every_n_scans = 10
return options
```

https://github.com/googlecartographer/cartographer_fetch/tree/master/cartographer_fetch/configuration_files
を引用

まとめ

ROSでGoogleのSLAM ライブラリ cartographerを使う。
→世界最先端の SLAM・Localizationを、だれでも試すことができる。

難しいSLAM・Localizationが簡単に、できて当たり前に。

Thank you for listening.