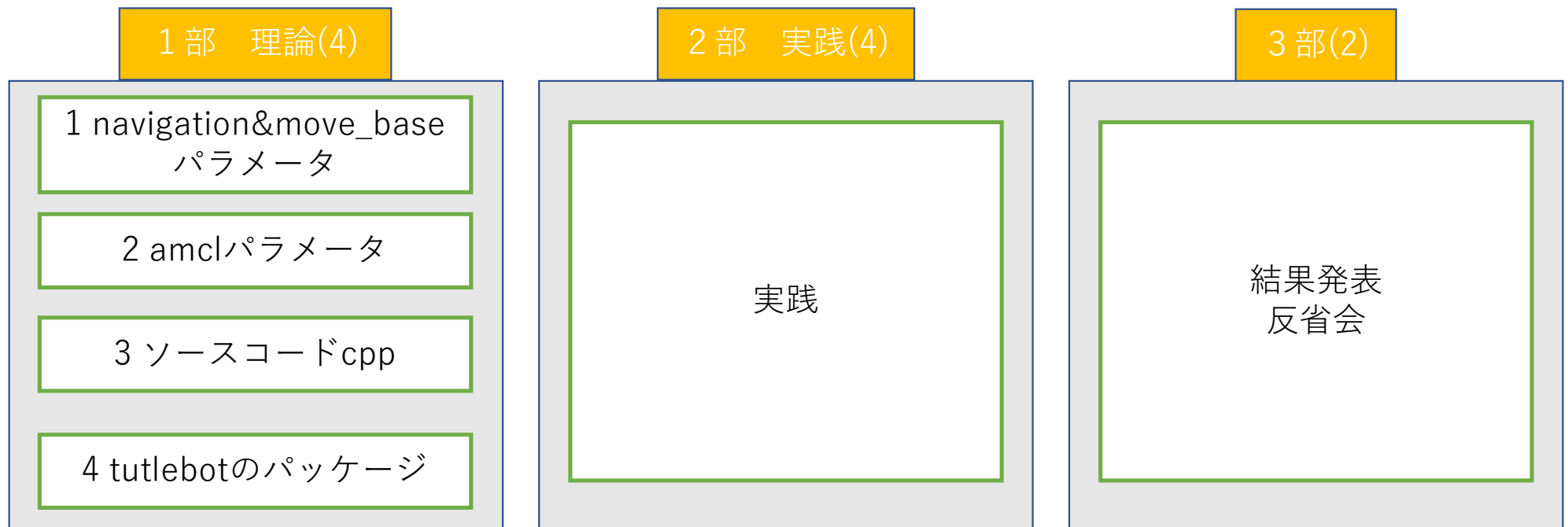


# CIT自律移動勉強会(navigation) 第一回 navigation & move\_base

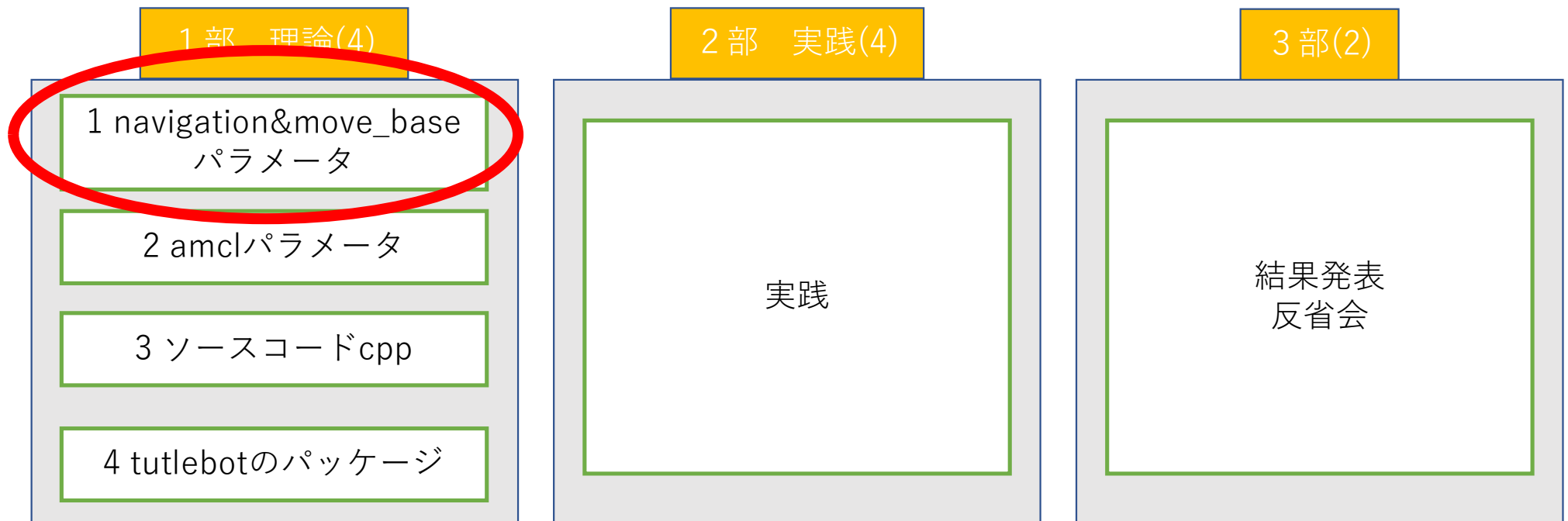
# 目次

- 勉強会スケジュール
- Navigationパッケージとは?
- Movebaseについて
  - 概要
  - パラメータ内容

# スケジュール



# スケジュール



# ROS navigationパッケージって何. . . ?

- 地図ベースで

自己位置推定

大域的(全体),局所的(近傍)な経路計画

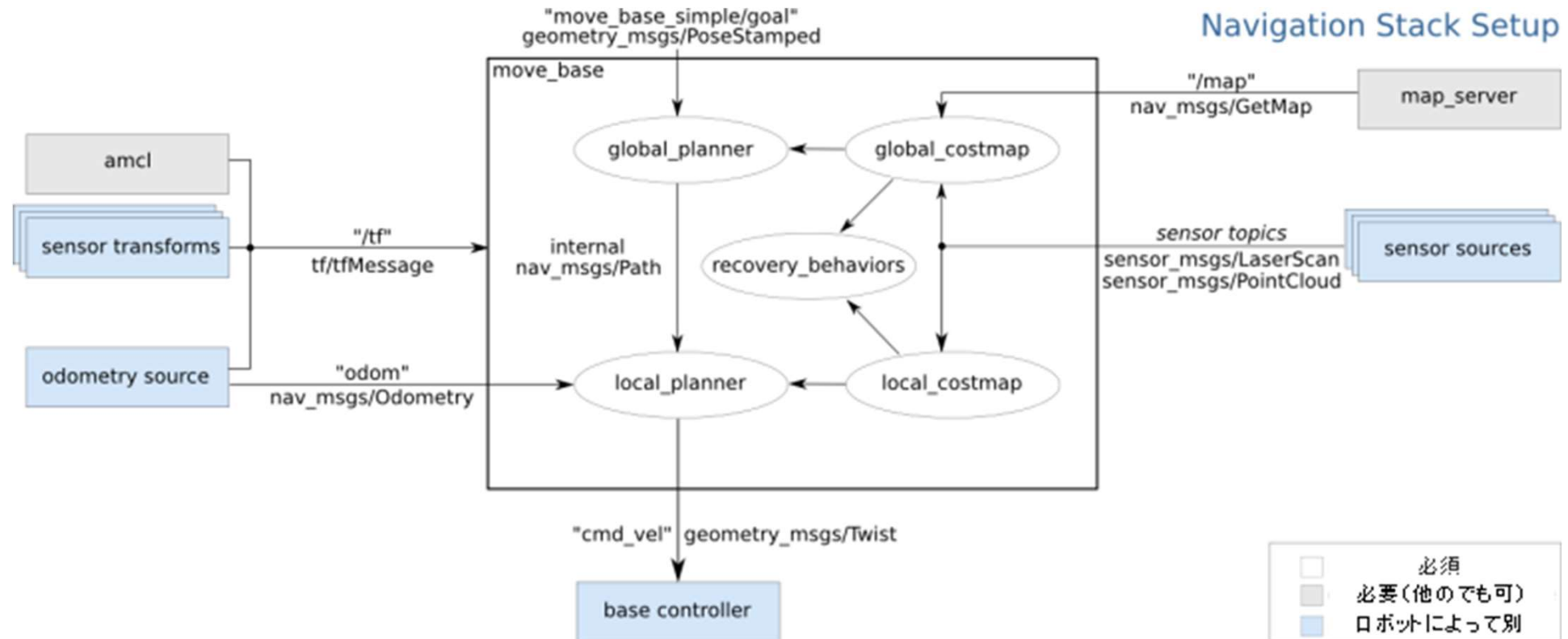
をするための機能をもつROSのパッケージ群

※地図作成機能は別パッケージ



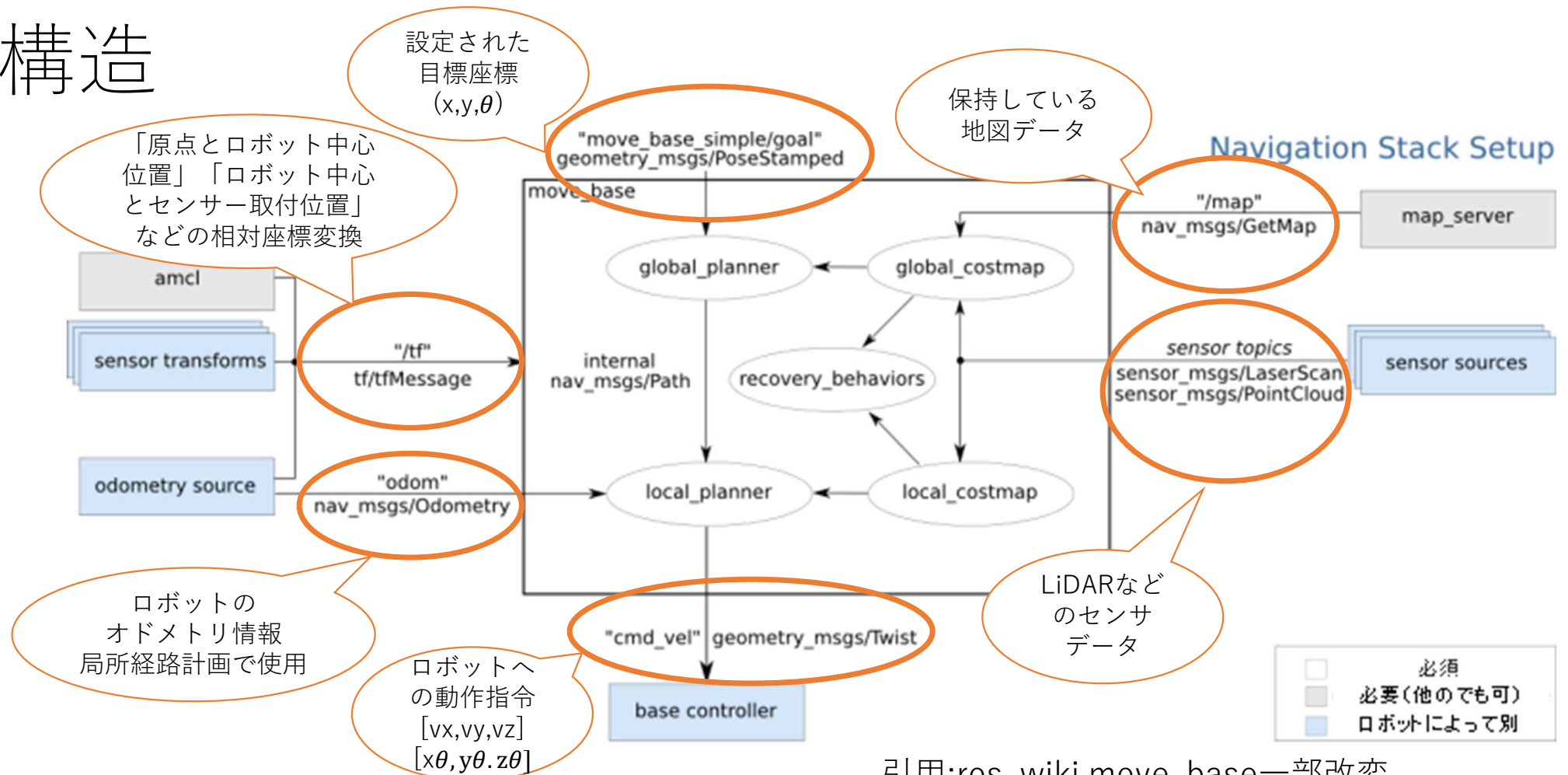
→地図上で指定された場所まで移動するために必要なもののセット

# 構造



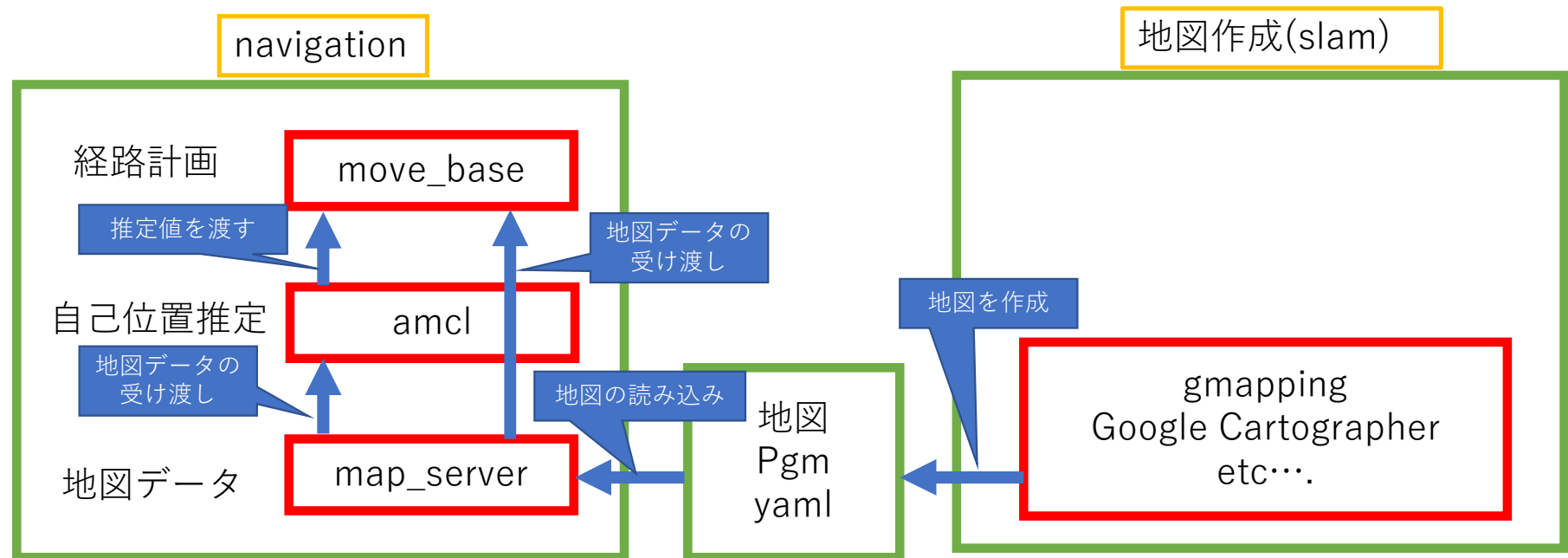
引用:ros\_wiki move\_base一部改変

# 構造



引用:ros\_wiki move\_base一部改変

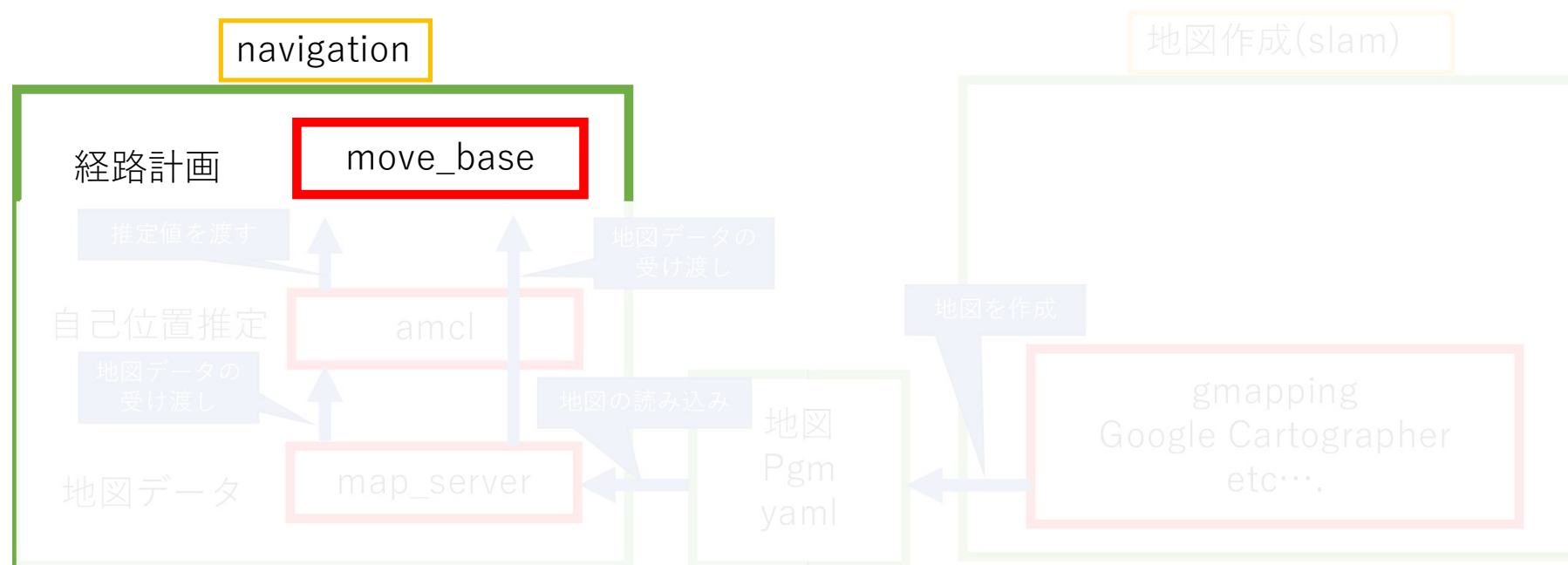
もっとわかりやすく////////



参考 Navigation Stack を理解する [@MoriKen](#)



もっとわかりやすく////////



参考 Navigation Stack を理解する [@MoriKen](#)

# move\_base

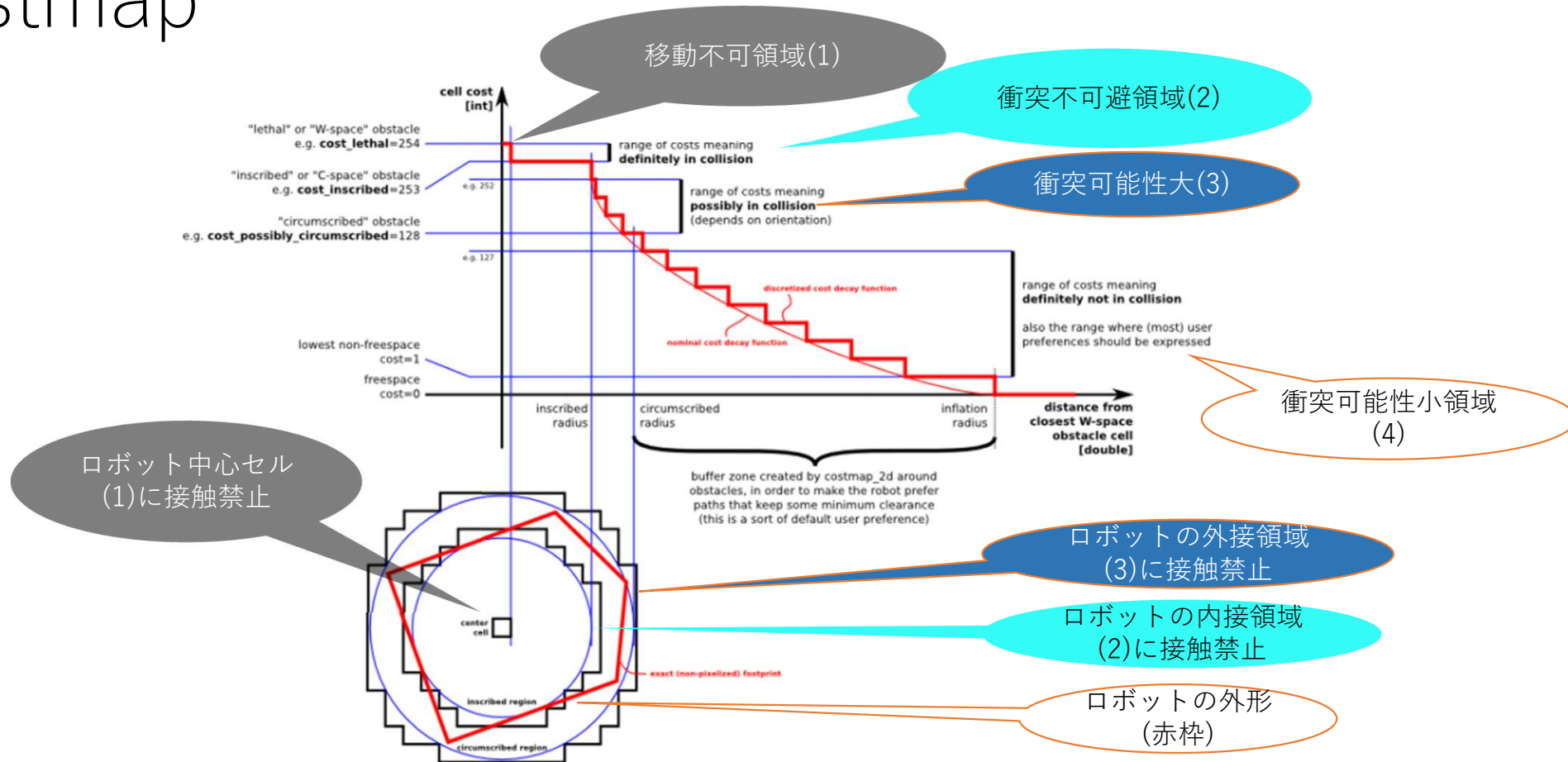
- Navigationの中心部（親玉）  
様々な情報を加味して、最終的に動作を決める  
後述するlaunchのみでも動作可能  
(自己位置推定なし!?)
- コストマップ→通れるか？障害物？を表したmap
  - Localcostmap(局所的狭い範囲)
  - Globalcostmap(大域的全域)
- 経路計画→現在地からゴールまでや障害物の回避経路
  - Localplanner(局所的、障害物回避など)
  - globalplanner(大域的、最終目的地までの全体)






# costmap

- 占有格子地図で表す通れそうな領域、障害物のある領域を数値で表現した地図(Costmap2D 2次元に限定)
- 空間をグリッド(四角)で区切り、グリッドにおける障害物っぽさを0~255で表現

# costmap



# Costmap例

-  → 障害物領域
  -  → 障害物を膨張させた  
衝突不可避領域
  -  → 衝突可能性領域  
なるべく避けたい…
- 色はRviz上で変更可能



# 経路計画(local\_planner)

動的ウィンドウアプローチ (DWA) アルゴリズム

- ①ロボットの制御空間 (dx、dy、dtheta) で個別にサンプリング
- ②サンプリングされた速度ごとに、ロボットの現在の状態から前方シミュレーションを実行して、サンプリングされた速度が一定の時間適用された場合に何が起きるかを予測
- ③障害物への近接性、ゴールへの近接性、グローバルパスへの近接性、速度などを盛り込んだ前方シミュレーションから得られた各軌道を下記の式で 評価 (スコアリング)、障害物と衝突する軌道を破棄。

$$G(v,\omega) = \sigma(\alpha \times \text{heading}(v,\omega) + \beta \times \text{dist}(v,\omega) + \gamma \times \text{velocity}(v,\omega))$$

$\text{heading}(v,\omega)$ :制御入力の際のロボットの方位とゴール方向の差の角度を180度から引いた値

$\text{dist}(v,\omega)$ :制御入力の際の、最近傍の障害物までの距離

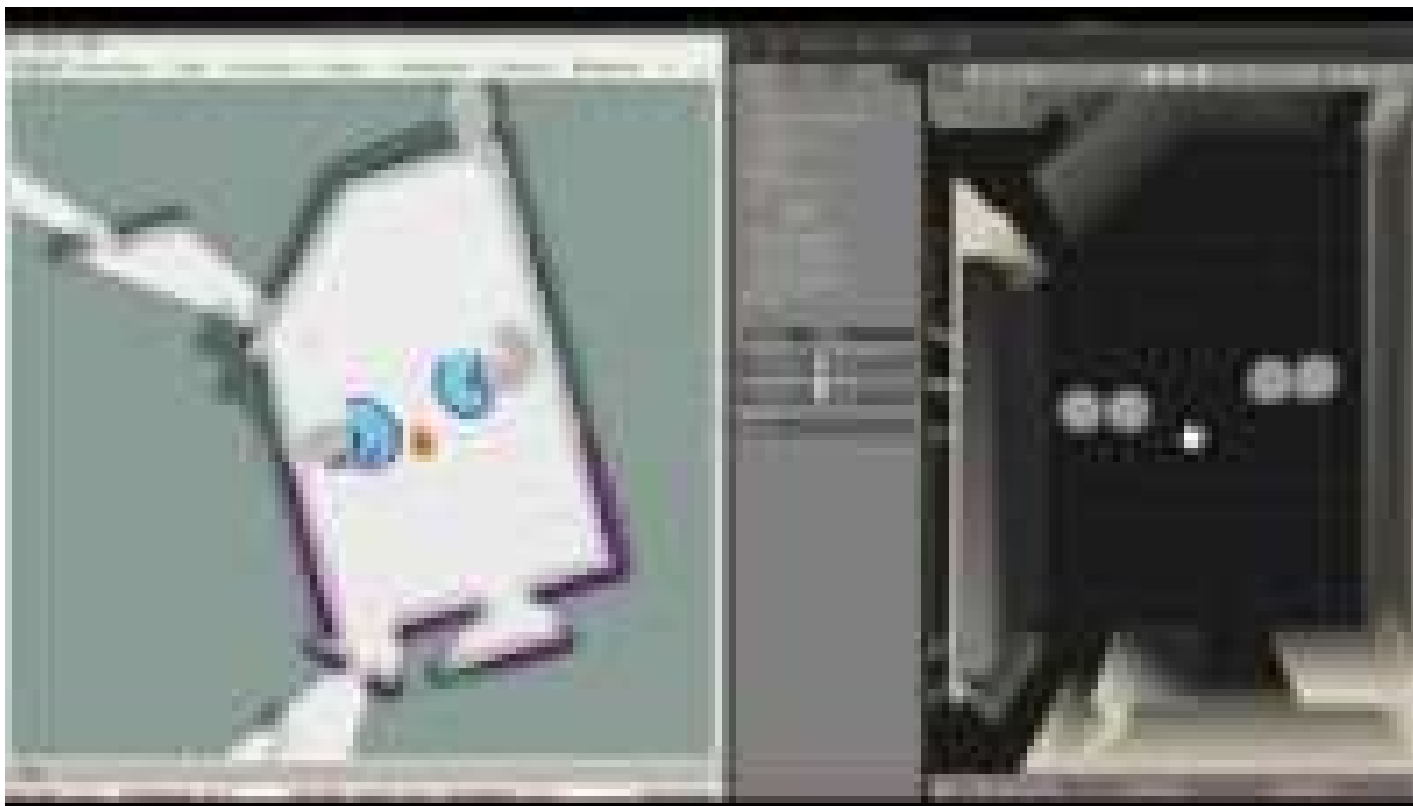
$\text{velocity}(v,\omega)$ :制御入力の際の速度

- ④最高スコアの軌道を選択し、関連する速度を出力。
- ⑤繰り返し

# 書き方失敗すると…



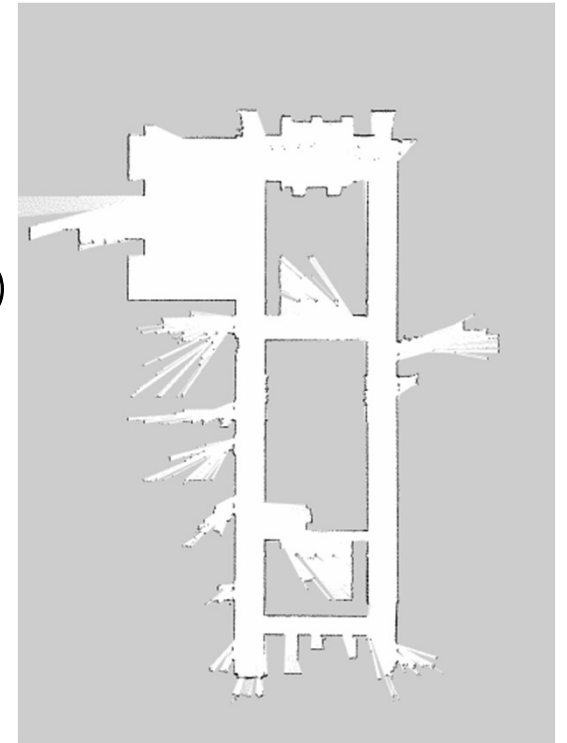
# 成功例





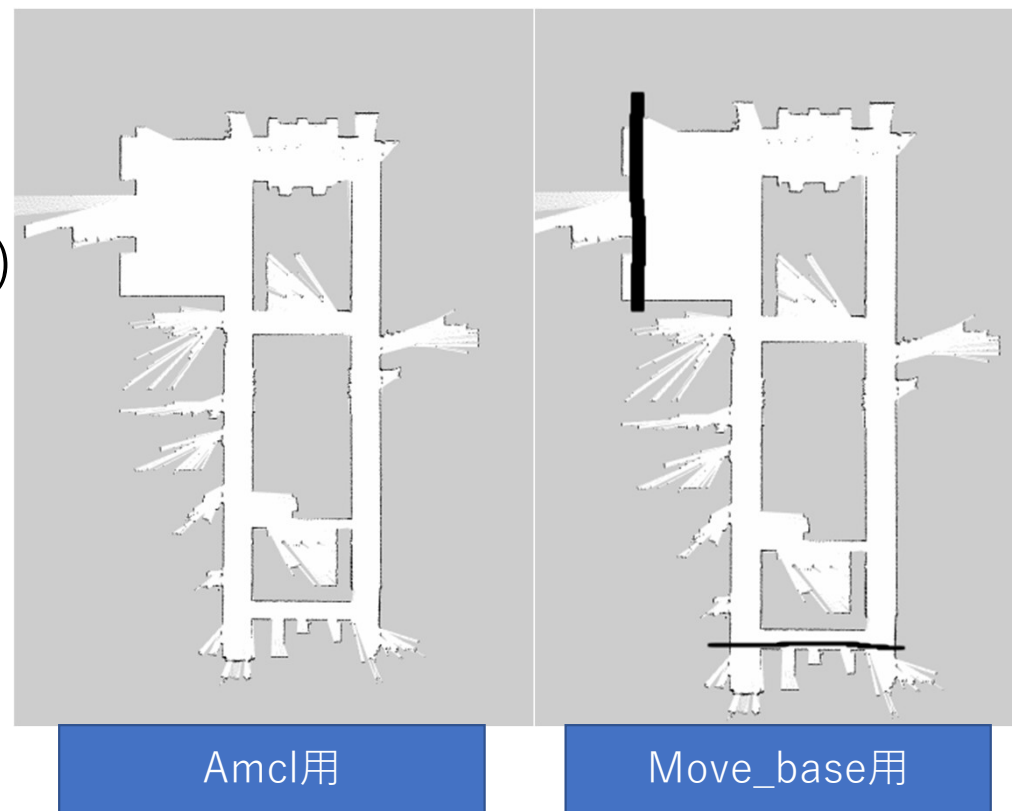
# 地図について

- Pgm形式の画像、yamlでパスなどの情報が記載されたファイルの組
  - Map上の黒い箇所は障害物として認識(立入禁止区域)
  - Gimp2などで保存されたmapを編集することが可能
- 障害物がないところを黒とした場合に  
amclなどの自己位置推定へ影響がでることが…



# 地図について

map\_serverを2つ立ち上げ  
move\_base用とamcl用で分ける  
(launchファイルでのmapは2つ)



# move\_base.launch書き方(1)

- <launch>
- <!-- Arguments -->
- <arg name="model" default="ロボットモデルネーム"/>
- <arg name="cmd\_vel\_topic" default="/cmd\_vel" />
- <arg name="odom\_topic" default="odom" />
- <arg name="move\_forward\_only" default="false"/>

# move\_base.launch書き方(2)

- <!-- move\_base -->
- <node pkg="move\_base" type="move\_base" respawn="false" name="move\_base" output="screen">
- <param name="base\_local\_planner" value="dwa\_local\_planner/DWAPlanerROS" />
- <rosparam file="\$(find raspicat\_navigation)/config/param/costmap\_common\_params\_\$(arg model).yaml" command="load" ns="global\_costmap" />
- <rosparam file="\$(find raspicat\_navigation)/config/param/costmap\_common\_params\_\$(arg model).yaml" command="load" ns="local\_costmap" />
- <rosparam file="\$(find raspicat\_navigation)/config/param/local\_costmap\_params.yaml" command="load" />
- <rosparam file="\$(find raspicat\_navigation)/config/param/global\_costmap\_params.yaml" command="load" />
- <rosparam file="\$(find raspicat\_navigation)/config/param/move\_base\_params.yaml" command="load" />
- <rosparam file="\$(find raspicat\_navigation)/config/param/dwa\_local\_planner\_params\_\$(arg model).yaml" command="load" />
- <remap from="cmd\_vel" to="\$(arg cmd\_vel\_topic)" />
- <remap from="odom" to="\$(arg odom\_topic)" />
- <param name="DWAPlanerROS/min\_vel\_x" value="0.0" if="\$(arg move\_forward\_only)" />
- </node>
- </launch>

# パラメータ

- パラメータはyamlファイルとして記述
  - 基本設定（更新速度など）
    - move\_base\_params.yaml
  - コストマップ関連
    - costmap\_common\_params.yaml
    - local\_costmap\_params.yaml
    - global\_costmap\_params.yaml
  - 動作（速度など）の設定
    - base\_local\_planner.yaml
    - dwa\_local\_planner.yaml etc...
  - Global\_plannerに関してはパラメータ設定なし（デフォルト）で良

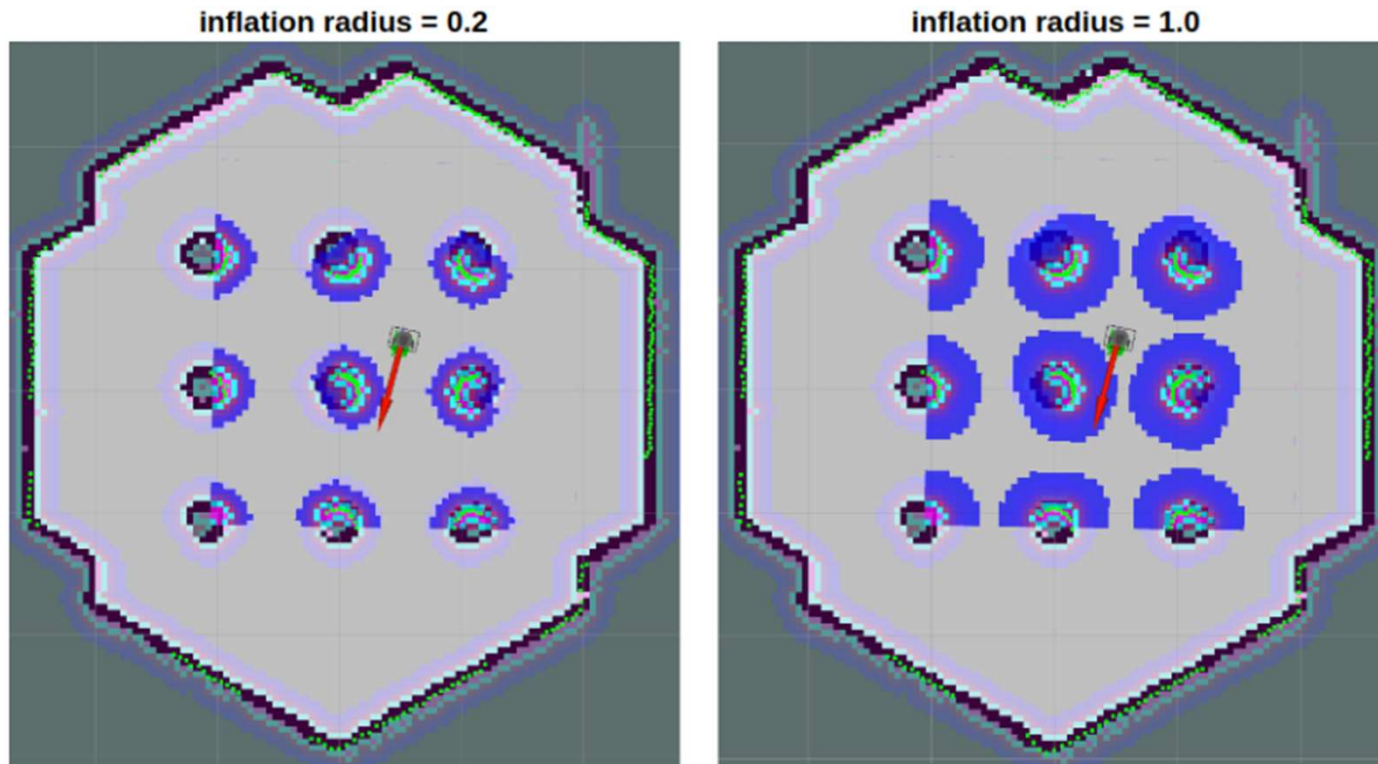
# move\_base\_params.yaml

- Shutdown\_costmap:false
  - >move\_baseが止まった時にcostmapノードの停止の是非
- Controller\_frequency:(double)[Hz]
  - >ロボットへの速度指令(cmd\_vel)を出す周期
- Controller\_partience::(double)[s]
  - >停止状態が継続した場合に待機する時間
- Planner\_frequency::(double)[Hz]
  - >全域に渡る経路計画の動作周期
- Planner\_partience::(double)[sec]
  - >有効なパスが見つからない場合、リカバリー動作を行うまでのロボットへの許容時間
- Oscillation\_timeout::(double)[s]
  - >リカバリーをするまでにロボットが発振するを許可する時間
- Oscillation\_distance::(double)[m]
  - >ロボットが発振じゃないと判定される距離、Oscillation\_timeoutが初期化される
- Conservative\_reset\_dist::(double)[m]
  - >リカバリー後のmapとcostmapの初期化時に削除する障害物の距離

# costmap\_common\_params.yaml

- `obstacle_range: (double)[m]`
    - >物体の障害物判定を行い、コストマップに判定する距離(meter)センサー性能に合わせる
  - `raytrace_range: (double)[m]`
    - >検出したデータをクリアし、フリーにする距離。obstacle\_rangeよりは大きく(+1?)
  - `footprint: [[x0, y0], [x1, y1], [x2, y2], [x3, y3]]`
    - >足跡、ロボットの外形表現 中心は[0.0]ロボットの大きさに合わせて調整
  - `#robot_radius: (double)[m]`
    - >ロボットが円形の場合に半径を記載
  - `inflation_radius: (double)[m]`
    - >障害物から膨張させる半径、検出された障害物へ接近不可にする範囲
  - `cost_scaling_factor: (double)`
    - >コストマップの計算時に用いる動作安定用のスケーリング変数  
コストファクタを低く設定しすぎたり、高く設定しすぎたりすると、パスの質が低下
- 
- `map_type: costmap`
    - >使用するコストマップの形式を選択voxel or costmap(cost\_map2d)
  - `observation_sources: scan`
    - >次に宣言するセンサを指定(名前)
  - `scan: {sensor_frame: base_scan, data_type: LaserScan, topic: scan, marking: true, clearing: true}`
    - >センサの座標系の指定、センサから送られてくるデータタイプ(LaserScan、PointCloud、PointCloud2など)、使用するtopic  
コストマップの反映の有無、センサデータを障害物のクリアに用いる(内側の領域は障害物が無しとするか)※3次元を2次元にするため注意

# inflation\_radius



引用:ROBOTIS turtlebot3 e-manual



# cost\_scaling\_factor

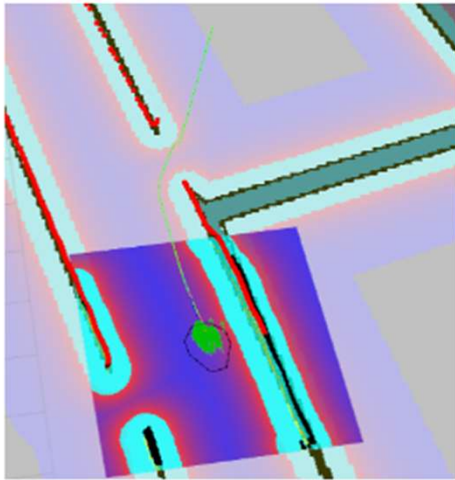


Figure 5: `cost_factor` = 0.01

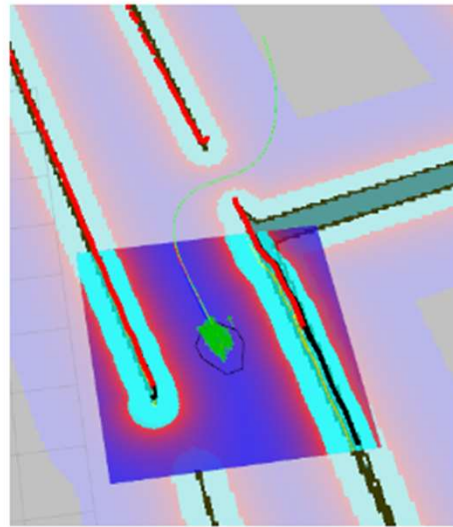


Figure 6: `cost_factor` = 0.55

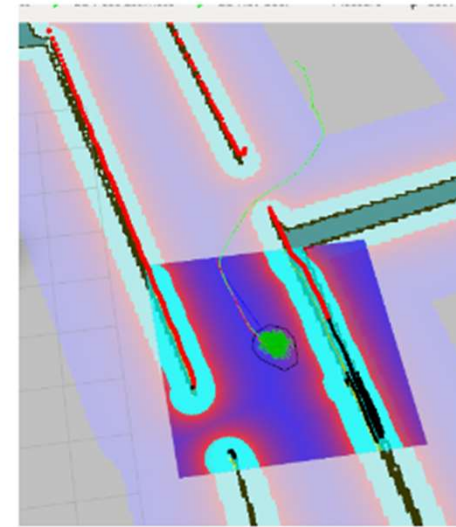


Figure 7: `cost_factor` = 3.55

引用:ROS Navigation Tuning Guide Kaiyu Zheng September 2, 2016\*

# local\_costmap\_params.yaml(2)

- local\_costmap:
- global\_frame: odom
  - >地図のフレーム(基にするもの)をオドメトリに設定
- robot\_base\_frame: base\_footprint
  - >使用するロボットのフレームをfootprintに指定
- update\_frequency: :(double)[Hz]
  - > costmapの更新周期、globalのものと同じが吉。  
早くしすぎると計算量が多くなるため注意  
rosvbagの量がえぐいことに
- publish\_frequency: :(double)[Hz]
  - >視覚情報(Rviz??)パブリッシュ (出力する)周期
- transform\_tolerance: :(double)[s]
  - >座標及びフレームの変換の許容時間  
変換エラーの対処の場合要調整



# local\_costmap\_params.yaml(2)

- static\_map: false
  - >存在する地図(map)を利用するかどうか
- rolling\_window: true
  - >自分の周りのだけのcostmapを使う(切り取り)オプション(局所)?
- width: :(double)[m]
  - >コストマップを計算する幅(x軸)を定義(局所)
- height: :(double)[m]
  - >コストマップを計算する高さ(y軸)
- resolution::(double)[m/cel]
  - >コストマップの解像度(細かさ)を定義.精度に影響



static map  
と同じが吉

# global\_costmap\_params.yaml

- global\_costmap:
- global\_frame: map
  - >地図のフレーム(基にするもの)をmapに設定
- robot\_base\_frame: base\_footprint
  - >使用するロボットのフレームをfootprintに指定
- update\_frequency: (double)[Hz]
  - >costmapの更新周期()
- publish\_frequency: (double)[Hz]
  - >パブリッシュ (出力する)周期
- transform\_tolerance: (double)[s]
  - >座標及びフレームの変換の許容時間
- static\_map: true
  - >保持している静的地図(map)を利用するか



# dwa\_local\_planner\_params.yaml(1)

- DWAPlannerROS:

- # Robot Configuration Parameters
- max\_vel\_x(double)[m/s]X軸での速度の最大値
- min\_vel\_x: (double)[m/s]X軸での速度の最小値
- max\_vel\_y: (double)[m/s]y軸での速度の最大値  
全方向移動でない場合0
- min\_vel\_y: (double)[m/s] y軸での速度の最小値

機体性能の  
ギリギリ?

## dwa\_local\_planner\_params.yaml(2)

- # The velocity when robot is moving in a straight line
- max\_vel\_trans: (double)[m/s]最大並進速度(実質の速度)
- min\_vel\_trans: (double)[m/s]最小並進速度,-でバック可能?
- max\_vel\_theta: (double)[rad/s]最大回転速度
- min\_vel\_theta: (double)[rad/s]最小回転速度
- acc\_lim\_x: (double)[m/s<sup>2</sup>]x軸の加速度制限
- acc\_lim\_y: (double)[m/s<sup>2</sup>]y軸の加速度制限
- acc\_lim\_theta: [rad/s<sup>2</sup>] theta(z?)軸の加速度制限

# dwa\_local\_planner\_params.yaml(3)

布団と1

- # Goal Tolerance Parametes
- xy\_goal\_tolerance: (double)[m]xyのゴールに対する許容距離誤差
- yaw\_goal\_tolerance: (double)[rad]ゴールに対する許容角度誤差
- latch\_xy\_goal\_tolerance: false 角度判定を入れるか入れないか

- # Forward Simulation Parameters
- sim\_time: (double)[sec]経路のシミュレーション時間  
(値小 = 障害物への反応向上, 値大 = 急速な回転不可)
- vx\_samples: x方向へとる並進速度サンプル数
- vy\_samples: y方向へとる並進速度サンプル数大体0?
- vth\_samples: 回転速度のサンプル数、vx,vyよりも多い方が吉
- controller\_frequency:[Hz] 制御周期

上げすぎると厳密  
になりすぎるので  
注意





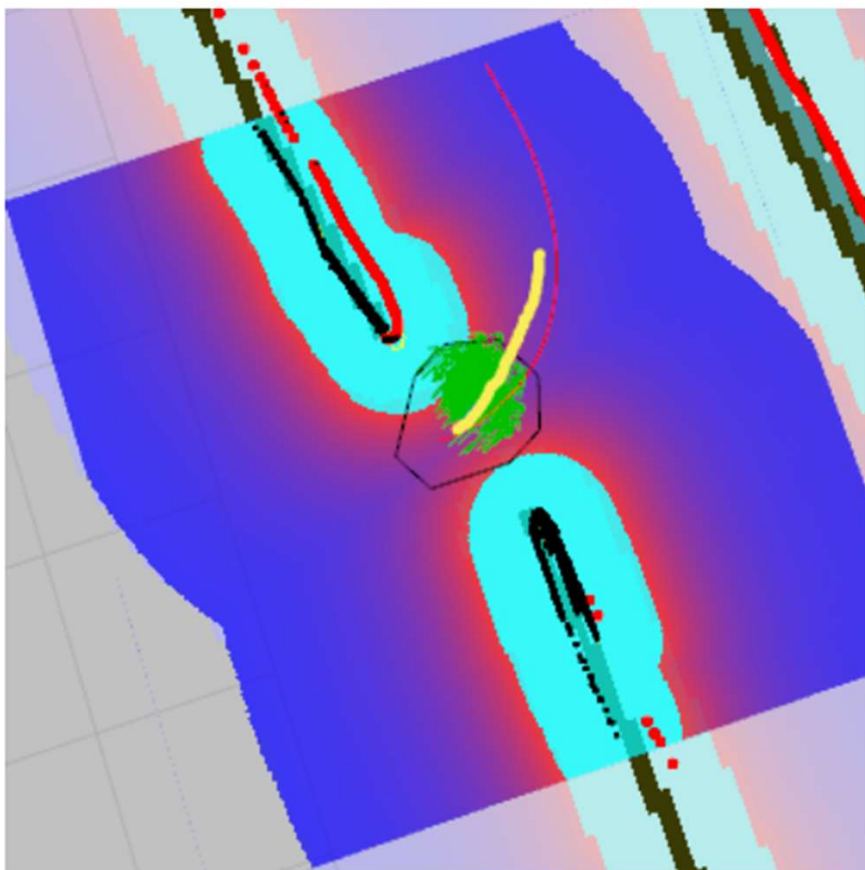


Figure 11: `sim_time = 1.5`

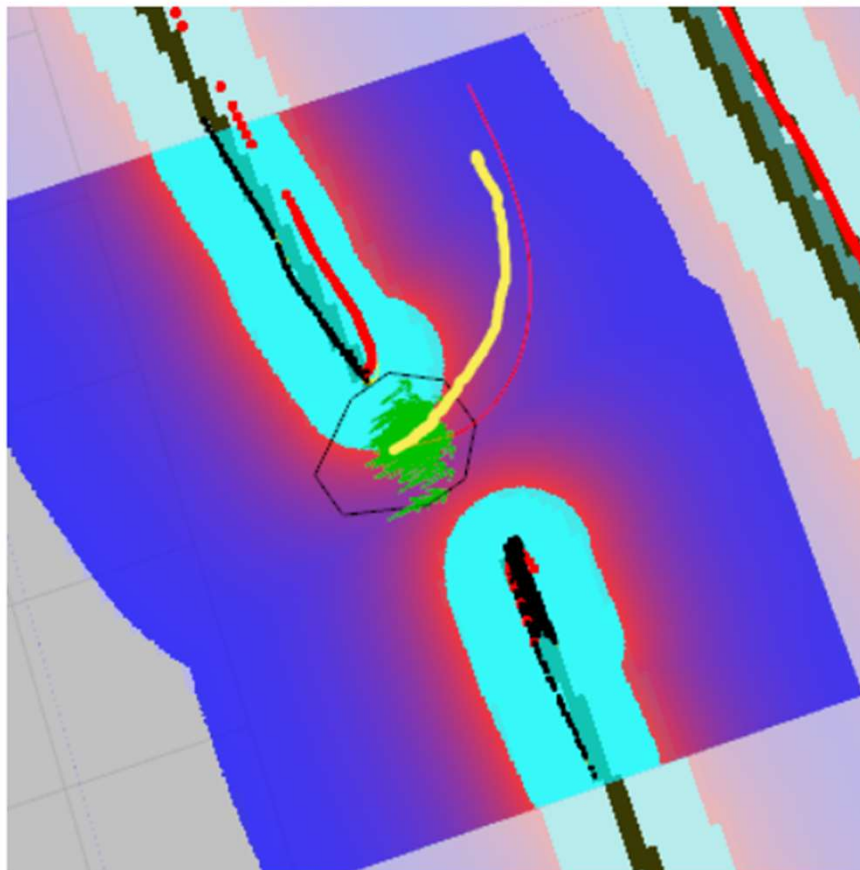


Figure 12: `sim_time = 4.0`

引用:ROS Navigation Tuning Guide Kaiyu Zheng September 2, 2016\*

# dwa\_local\_planner\_params.yaml(4)

```
cost = path_distance_bias * (distance(m) to path from the endpoint of the trajectory)
      + goal_distance_bias * (distance(m) to local goal from the endpoint of the trajectory)
      + occdist_scale * (maximum obstacle cost along the trajectory in obstacle cost (0-254))
```

- # Trajectory Scoring Parameters

- path\_distance\_bias: (double)
  - >ローカルプランナーがどれだけグローバルパスに近づくべきかの重み
- goal\_distance\_bias: (double)
  - >目標地点にどれだけ近づこうとするかの重み
- occdist\_scale: (double)
  - >どれだけ障害物回避を行うかに関する重み
- forward\_point\_distance: (double)[meter]
  - >ロボットとゴールの計算の頻度

ゴールに向かいつつ、  
障害物を避けながら、  
高速に走ることが  
できる？

# dwa\_local\_planner\_params.yaml(5)

- stop\_time\_buffer: (double)[meter] ロボットが停止するまでの距離
- scaling\_speed: (double)[meter/sec] スケーリング速度
- max\_scaling\_factor: (double) 最大スケーリング要素
- # Oscillation Prevention Parameters
- oscillation\_reset\_dist: 0.05 発振防止パラメータ
- # Debugging
- publish\_traj\_pc : true 移動軌跡のデバッグ要素
- publish\_cost\_grid\_pc: true costmapのデバッグ要素

# 質問

- Globalはダイクストラ法。
- Localのpassの変更はdwa ???
- rolling\_windowとは結果なに ??

# まとめ

- Navigationとは
  - 地図ベースで自律移動を行うためのパッケージ群
- move\_base
  - navigationパッケージの中心(経路計画)
  - パラメータはロボットや目的に合わせて後述の参考文献を参考に要調整(ゝ^ゝ^ゝ)

# 参考文献

- ROSロボットプログラミングバイブル
- ROS wiki
- ROBOTIS Turtlebot3 e-manual
- ROS Navigation Tuning Guide

来週の内容

# 第二回自律移動勉強会

## amclとは？

## パラメータ調整