

fuRo

Future Robotics Technology Center

千葉工業大学

ROSを用いた 自律移動ロボットの システム構築

千葉工業大学 未来ロボット技術研究センター

原 祥堯 (HARA, Yoshitaka)



@ystk_hara

<https://www.slideshare.net/hara-y/ros-nav-rsj-seminar>

つくばチャレンジでの自律走行の様子

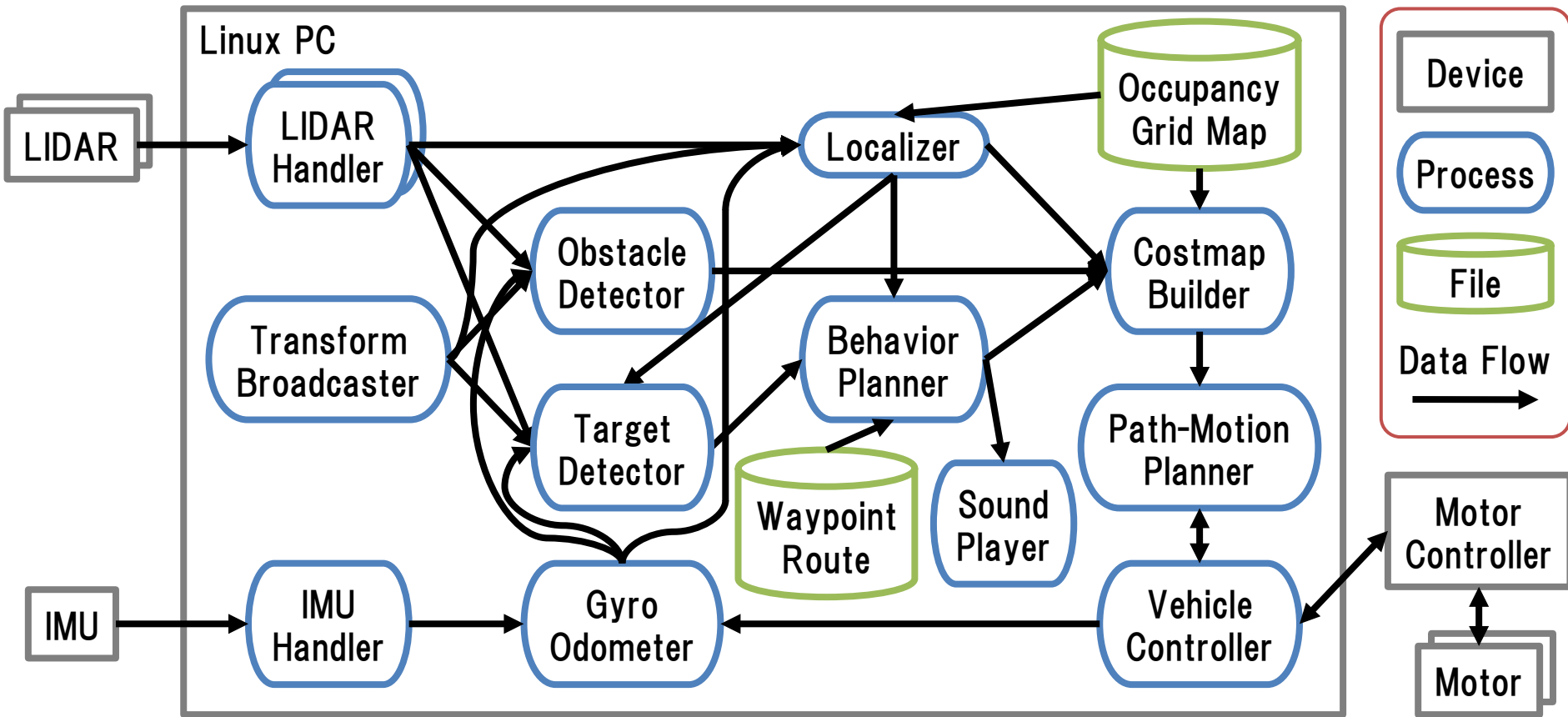
https://youtu.be/FzouCV4_Jnw



開発したロボットの外観と搭載センサ



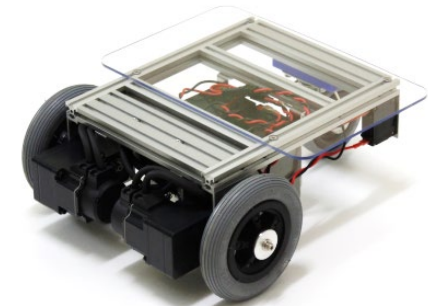
自律走行システムの構成図



1. 自律走行を実現するための ROS パッケージ
2. 自己位置推定と SLAM の基本
3. 各パッケージのアルゴリズム
 - a. 自己位置推定 (amcl)
 - b. SLAM、地図生成 (gmapping)
 - c. 大域的経路計画、局所的動作計画 (move_base)
4. ROS に関する情報の調べ方
5. ROS での開発に関する知見
6. まとめ

自律走行に使用できるデバイスとパッケージ

- センサ <http://wiki.ros.org/Sensors>
- 移動プラットフォーム台車 <http://wiki.ros.org/Robots>
 - T-frog i-Cart mini (T-frog モータコントローラ)
<http://t-frog.com/>
 - T-frog i-Cart edu (T-frog モータコントローラ)
<http://t-frog.com/>
- T-frog モータコントローラ ROS 対応パッケージ
 - ypspur_ros : 公式パッケージ (SEQSENSE 渡辺先生)
https://github.com/openspur/ypspur_ros
 - ypspur_ros_bridge : 旧パッケージ (筑波大 味研)
<http://www.roboken.iit.tsukuba.ac.jp/platform/gitweb/>
 - open-rdc icart_mini : Gazebo 対応 (千葉工大 林原研)
<https://github.com/open-rdc/icart>



ROS で使える3次元シミュレータ

自律走行に関するパッケージとその機能

□ slam_gmapping

- SLAM、地図生成

□ navigation

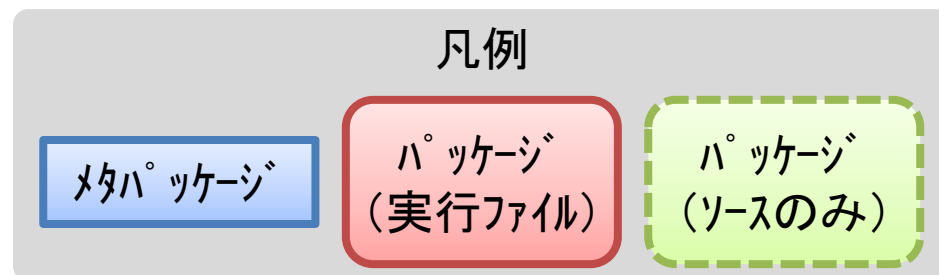
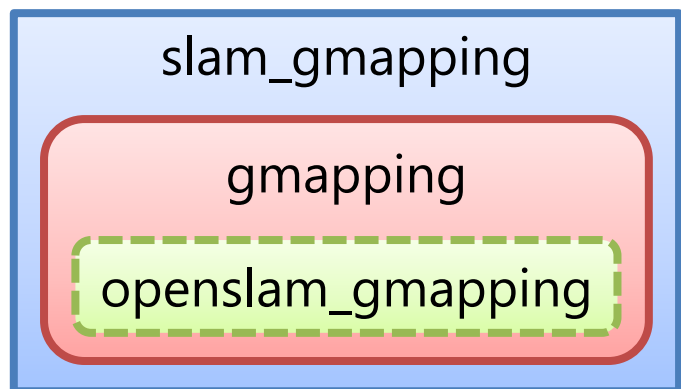
- 自己位置推定
- 占有格子地図生成、障害物情報の管理
- 大域的経路計画
- 局所的動作計画（障害物回避）

自律走行に必要な機能をひとつおりに提供、
C++で実装されている

自律走行を行う際のワークフロー

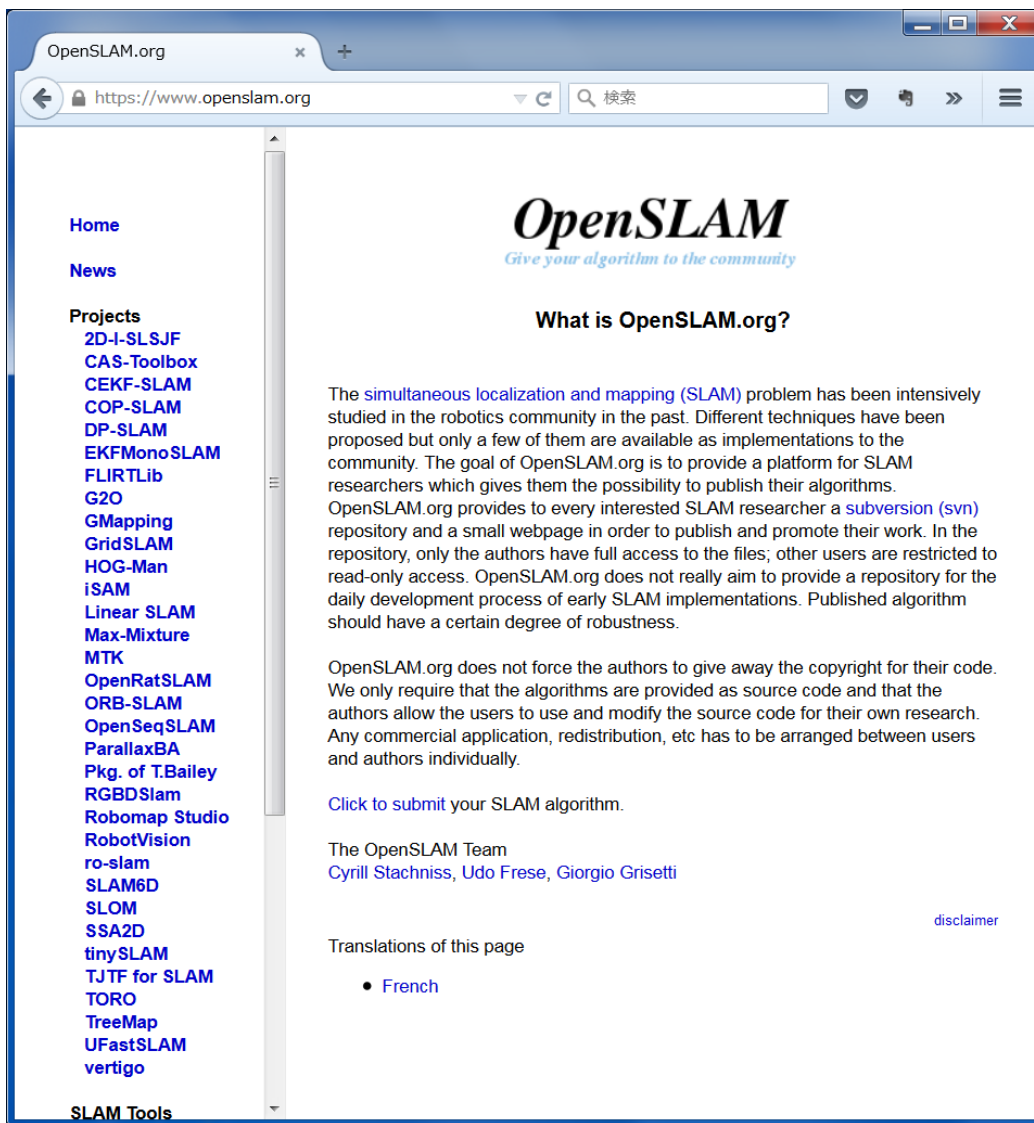
1. ロボットを**手動走行**させて環境のセンサーデータを取得、bag ファイルとして保存
2. bag ファイルを再生し、**slam_gmapping** パッケージを使用してオフラインで**地図生成**（オンラインも可能）
3. 生成した地図に基づき、**navigation** パッケージを使用して**自律走行**

slam_gmapping パッケージの構成



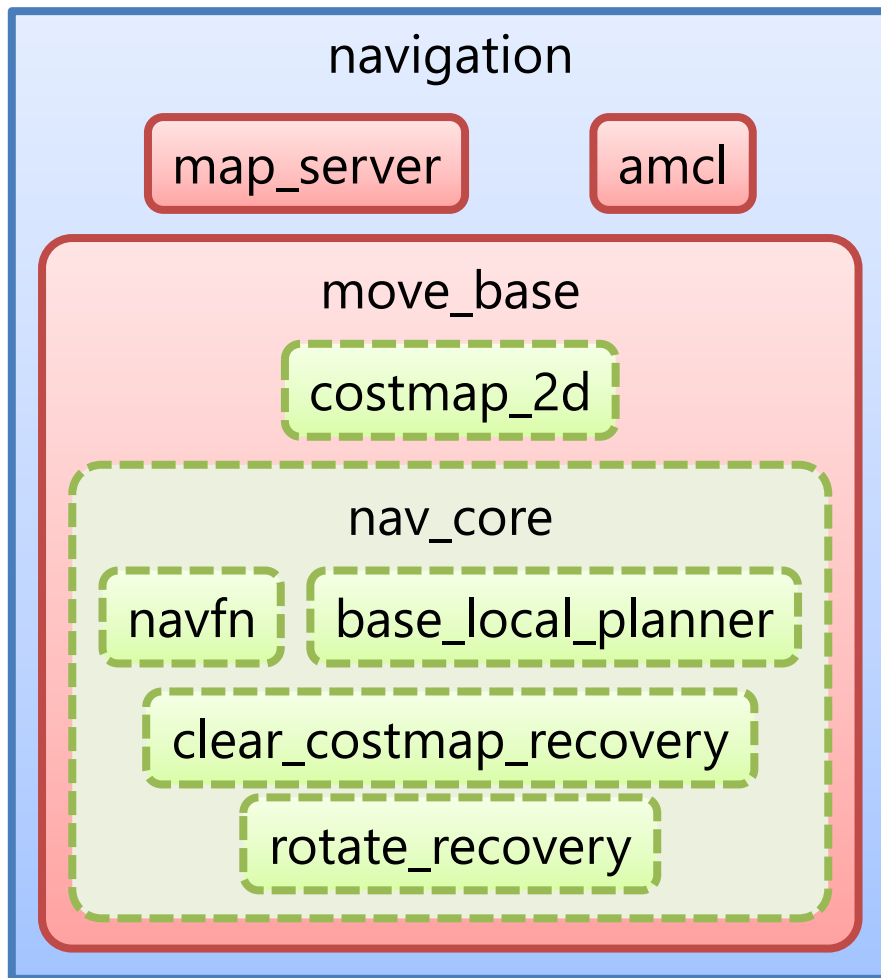
- **gmapping** : SLAM、地図生成の実行 (ROSラッパー)
- **openslam_gmapping** :
OpenSLAM で公開されている **Rao-Blackwellized Particle Filter** による Grid-based SLAM
(FastSLAM 2.0 での Grid Mapping)

OpenSLAM とは



- SLAM の各種アルゴリズムをオープンソースで公開する Web サイト
- EKF, RBPF, Graph-based SLAM, ICP マッチング など複数の手法の実装あり
- 研究者が各手法を比較するために公開
- ライブ リではない

navigation パッケージの構成



凡例

メタパッケージ

パッケージ
(実行ファイル)

パッケージ
(ソースのみ)

- **map_server** : 既存地図の配信
- **amcl** : 自己位置推定
- **move_base** : 移動タスクの実行
 - **costmap_2d** : コストマップの生成
 - **nav_core** : 行動インタフェース定義
 - **navfn** : 大域的経路計画
 - **base_local_planner** : 局所的動作計画
 - ***_recovery** : スタックした場合の復帰動作

nav_core によるパッケージの入れ替え

- ヘッダファイルでのインタフェース定義のみ（純粹仮想関数）
 - 大域的経路計画：BaseGlobalPlanner
 - 局所的動作計画：BaseLocalPlanner
 - スタック復帰動作：RecoveryBehavior
- 実装は move_base のパラメータで**実行時に変更可能**
- ROS の pluginlib という機能で実現されている

入れ替え可能なパッケージとそのアルゴリズム

□ 大域的経路計画 (BaseGlobalPlanner)

- navfn : ダイクストラ法
- global_planner : A* / ダイクストラ法
- carrot_planner : ゴールに向かって直進

□ 局所的動作計画 (BaseLocalPlanner)

- base_local_planner : Dynamic Window Approach
- dwa_local_planner : Dynamic Window Approach の別実装

□ スタック復帰動作 (RecoveryBehavior)

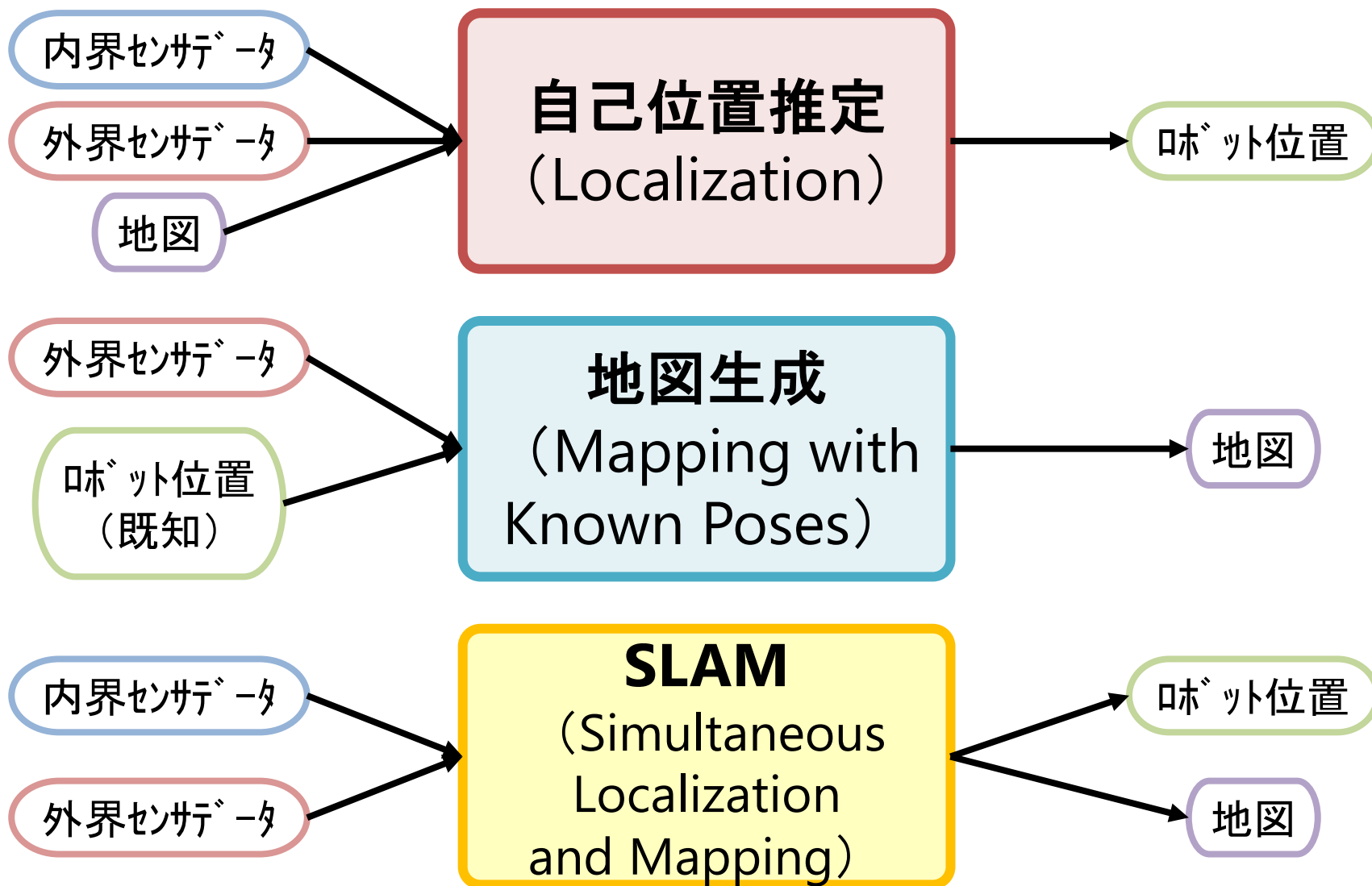
- clear_costmap_recovery : コストマップをクリア
- rotate_recovery : ロボットが旋回して障害物を測定
- move_slow_and_clear : コストマップをクリア、移動速度を制限

標準で使われるアルゴリズムのまとめ

- **SLAM、地図生成 (gmapping)**
Rao-Blackwellized Particle Filter による Grid-based SLAM (FastSLAM 2.0 での Grid Mapping)
- **自己位置推定 (amcl)**
Adaptive/Augmented Monte Carlo Localization
- **経路計画 (move_base)**
Global Dynamic Window Approach
 - 大域的経路計画 (navfn)
Navigation Function に基づくダイクストラ法
 - 局所的動作計画 (base_local_planner)
Dynamic Window Approach

1. 自律走行を実現するための ROS パッケージ
- 2. 自己位置推定と SLAM の基本**
3. 各パッケージのアルゴリズム
 - a. 自己位置推定 (amcl)
 - b. SLAM、地図生成 (gmapping)
 - c. 大域的経路計画、局所的動作計画 (move_base)
4. ROS に関する情報の調べ方
5. ROS での開発に関する知見
6. まとめ

自己位置推定、地図生成、SLAM とは



自己位置推定と SLAM の手法

	スキャンマッチング系		Bayes Filter 系	Graph-based SLAM 系
性質	逐次最適化、オンライン		フィルタリング、 オンライン	全体最適化、 オフライン
手法の概要	点群を最適化計算で位置合わせ		事前確率と尤度を 確率的に融合	味ット位置やランド マーク位置 (地図) を 表すグラフを最適化
	詳細マッチング	大域マッチング		
	初期位置あり、 ユークリッド空間で 対応付け	初期位置なし、 特徴量空間で 対応付け		
手法の例	ICP、NDT など	Spin Image、 FPFH、PPF、 SHOT など	EKF、HF、 PF 、 RBPF など	ポーズ調整、バンドル 調整、完全 SLAM

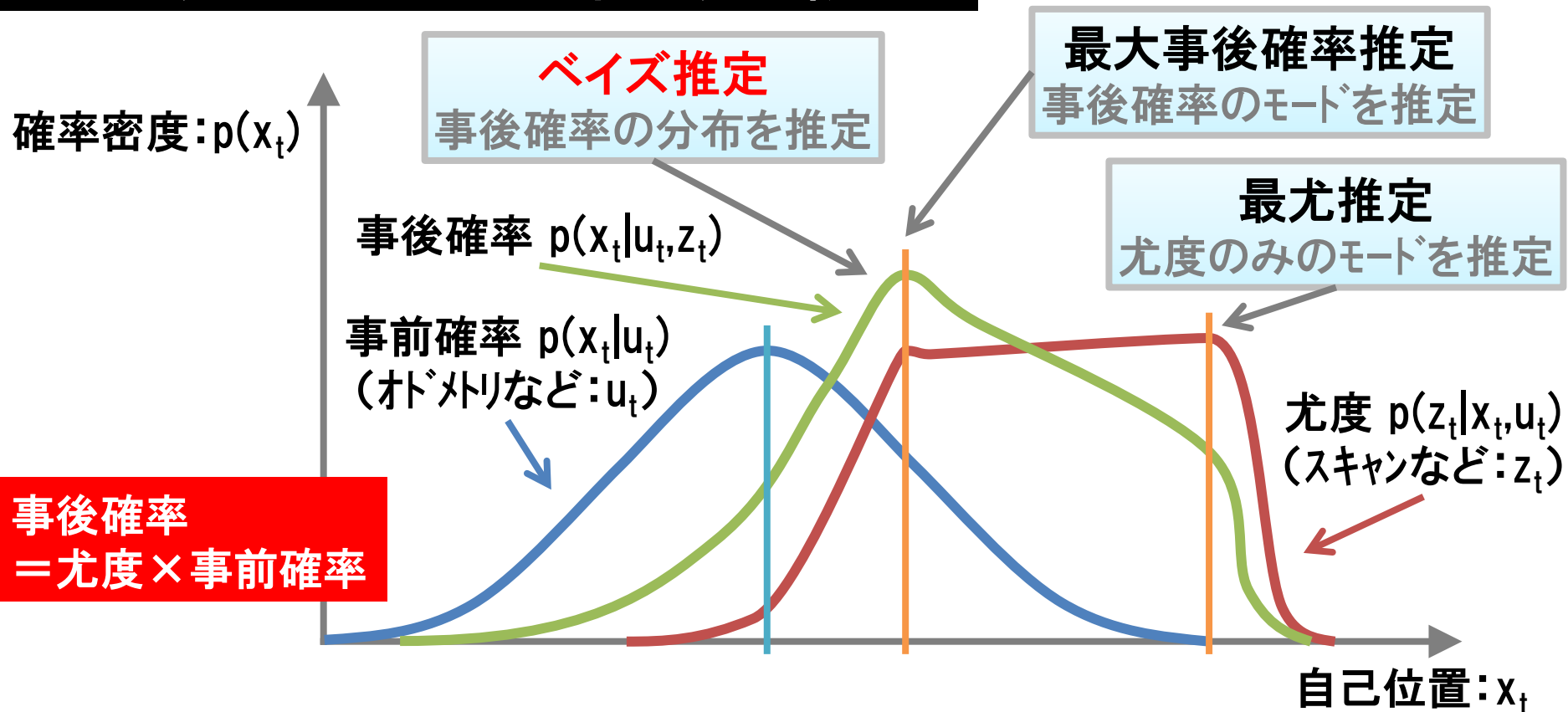
以下のパッケージは一部の手法を提供しているに過ぎない

- amcl : Particle Filter (PF) による Monte Carlo Localization
- gmapping : Rao-Blackwellized Particle Filter (RBPF)

SLAM の性能は屋外環境に対して不十分な場合も多い

最尤推定、最大事後確率推定、ベイズ推定

ベイズの定理による自己位置推定の概念図



Bayes Filter は、ベイズの定理に基づくベイズ推定によって自己位置の事後確率分布を求める

Bayes Filter による自己位置推定

- ロボットの自己位置の**確率分布**を求める
- 地図は事前に与えられている

平面図

地図: m

自己位置
(時刻 $t-1$): x_{t-1}

ドットリ動作(移動量): u_t

対応付け

スキャン計測: z_t

真の自己位置(時刻 t): x_t

ドットリによる自己位置
(時刻 t): $T(u_t)x_{t-1}$

確率的自己位置推定の隠れマルコフモデル

自己位置の時系列を1次マルコフ過程と仮定

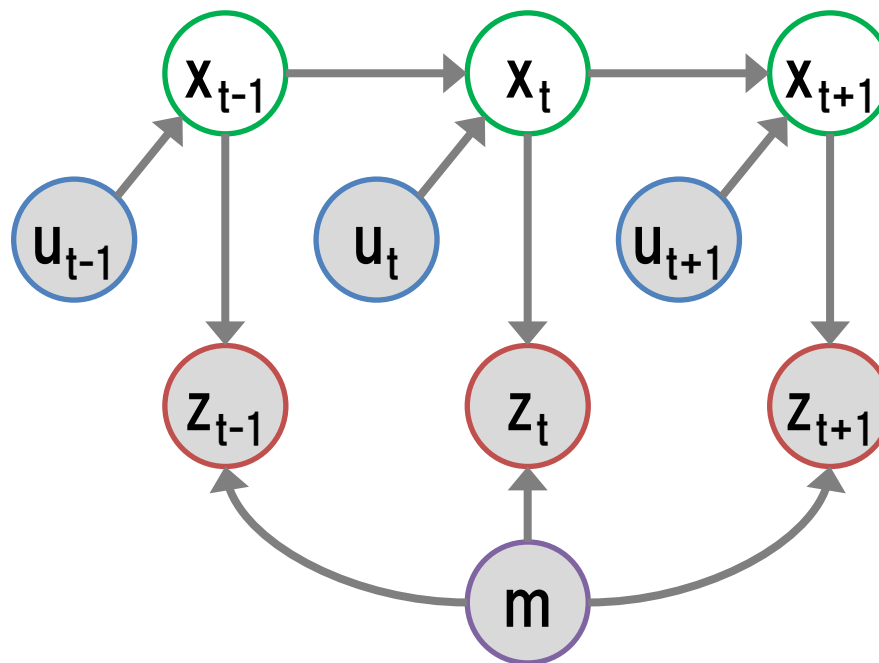
自己位置推定のグラフィカルモデル

robot pose: x
(hidden variable)

motion: u
(e.g. odometry)

measurement: z
(e.g. laser scan)

map: m



Bayes Filter の定式化

- 漸化式で時系列の確率過程を表現
- Extended Kalman Filter、Histogram Filter などはすべて Bayes Filter の理論に基づいた実装

$p(\mathbf{x}_t \mid \mathbf{u}_{1:t}, \mathbf{z}_{1:t}, \mathbf{m})$ ← 時刻 t の自己位置 \mathbf{x}_t の事後確率

$= \eta p(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{m})$ ← 尤度(計測モデル)

$$\cdot \int p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t) p(\mathbf{x}_{t-1} \mid \mathbf{u}_{1:t-1}, \mathbf{z}_{1:t-1}, \mathbf{m}) d\mathbf{x}_{t-1}$$

動作モデル

事前確率

時刻 $t-1$ の自己位置 \mathbf{x}_{t-1}

\mathbf{x} : ロボット位置

\mathbf{m} : 地図

\mathbf{u} : 制御動作(オドメトリなど)

\mathbf{z} : 環境計測(レーザスキャンなど)

Bayes Filter の数学的導出

$p(\mathbf{x}_t | \mathbf{u}_{1:t}, \mathbf{z}_{1:t}, \mathbf{m})$ ← 時刻 t の自己位置 \mathbf{x}_t の事後確率

↓ **ベイズの定理**

$$= \eta \underbrace{p(\mathbf{z}_t | \mathbf{x}_t, \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m})}_{\text{条件付き独立性 (過去には非依存)}} \underbrace{p(\mathbf{x}_t | \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m})}_{\text{全確率の定理}}$$

$$= \eta \underbrace{p(\mathbf{z}_t | \mathbf{x}_t, \mathbf{m})}_{\text{尤度 (計測モデル)}} \int \underbrace{p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m})}_{\text{条件付き独立性 (1次マルコフ性)}} \underbrace{p(\mathbf{x}_{t-1} | \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m})}_{\text{条件付き独立性}} d\mathbf{x}_{t-1}$$

$$= \eta \underbrace{p(\mathbf{z}_t | \mathbf{x}_t, \mathbf{m})}_{\text{尤度 (計測モデル)}} \int \underbrace{p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{u}_t)}_{\text{動作モデル}} \underbrace{p(\mathbf{x}_{t-1} | \mathbf{u}_{1:t-1}, \mathbf{z}_{1:t-1}, \mathbf{m})}_{\text{事前確率}} d\mathbf{x}_{t-1}$$

時刻 $t-1$ の自己位置 \mathbf{x}_{t-1}

漸化式として定式化

ベイズの定理の適用

□ ベイズの定理

$$p(A | B) = \frac{p(B | A) p(A)}{p(B)}$$

□ 条件付きベイズの定理

$$p(A | B, C) = \frac{p(B | A, C) p(A | C)}{p(B | C)}$$

□ 自己位置推定に適用

$$p(\mathbf{x}_t | \mathbf{u}_{1:t}, \mathbf{z}_{1:t}, \mathbf{m})$$

$$= p(\mathbf{x}_t | \mathbf{z}_t, \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m})$$

$$= \frac{p(\mathbf{z}_t | \mathbf{x}_t, \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m}) p(\mathbf{x}_t | \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m})}{p(\mathbf{z}_t | \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m})}$$

$$= \eta p(\mathbf{z}_t | \mathbf{x}_t, \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m}) p(\mathbf{x}_t | \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m})$$

\mathbf{x} : ロボット位置

\mathbf{m} : 地図

\mathbf{u} : 制御動作 (オドメトリ)

\mathbf{z} : 環境計測 (レーザースキャン)

マルコフ性の仮定と条件付き独立性の適用

1次マルコフ性を仮定

- 現在の状態は、1時刻前の状態にだけ依存
- それ以前の状態や他のデータとは独立

$$p(z_t \mid \mathbf{x}_t, \mathbf{u}_{1:t}, z_{1:t-1}, m) = p(z_t \mid \mathbf{x}_t, m)$$

現在の環境計測 z_t は、現在の状態である自己位置 \mathbf{x}_t と地図 m にのみ依存

$$p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_{1:t}, z_{1:t-1}, m) = p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t)$$

現在の状態である自己位置 \mathbf{x}_t は、1時刻前の状態 \mathbf{x}_{t-1} とそれ以後の制御動作 \mathbf{u}_t にのみ依存

全確率の定理の適用

□ 全確率の定理

$$p(A) = \int p(A | B) p(B) dB$$

□ 条件付き全確率の定理

$$p(A | C) = \int p(A | B, C) p(B | C) dB$$

□ 自己位置推定に適用

$$p(\mathbf{x}_t | \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m})$$

隠れ変数 \mathbf{x}_{t-1} を導入

$$= \int p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m}) p(\mathbf{x}_{t-1} | \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m}) d\mathbf{x}_{t-1}$$

\mathbf{x} : ロボット位置

\mathbf{m} : 地図

\mathbf{u} : 制御動作 (オドメトリ)

\mathbf{z} : 環境計測 (レーザーキャン)

1. 自律走行を実現するための ROS パッケージ
2. 自己位置推定と SLAM の基本
3. 各パッケージのアルゴリズム
 - a. 自己位置推定 (amcl)
 - b. SLAM、地図生成 (gmapping)
 - c. 大域的経路計画、局所的動作計画 (move_base)
4. ROS に関する情報の調べ方
5. ROS での開発に関する知見
6. まとめ

Particle Filter の定式化

- Bayes Filter の一種だが、定式化は若干異なる
- 各パーティクルは、時刻 t の自己位置だけでなく、**時刻1から時刻 t の走行軌跡を保持する**
- 動作モデルによる事前確率の計算に積分なし

$$\begin{aligned}
 & \underline{p(\mathbf{x}_{1:t} \mid \mathbf{u}_{1:t}, \mathbf{z}_{1:t}, \mathbf{m})} \longleftarrow \text{時刻1から}t\text{の走行軌跡}x_{t:t}\text{の事後確率} \\
 & = \underline{\eta p(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{m})} \longleftarrow \text{尤度(計測モデル)} \\
 & \quad \cdot \underline{p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t)} \quad \underline{p(\mathbf{x}_{1:t-1} \mid \mathbf{u}_{1:t-1}, \mathbf{z}_{1:t-1}, \mathbf{m})} \\
 & \quad \swarrow \quad \quad \quad \longleftarrow \\
 & \text{動作モデル} \quad \quad \quad \text{事前確率} \quad \quad \quad \text{時刻1から}t-1\text{の走行軌跡}x_{1:t-1}
 \end{aligned}$$

Particle Filter の数学的導出

$p(\mathbf{x}_{1:t} \mid \mathbf{u}_{1:t}, \mathbf{z}_{1:t}, \mathbf{m})$ ← 時刻1からtの走行軌跡 $\mathbf{x}_{t:t}$ の事後確率

↓ ベイズの定理

$$= \eta p(\mathbf{z}_t \mid \mathbf{x}_{1:t}, \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m}) p(\mathbf{x}_{1:t} \mid \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m})$$

条件付き独立性
(過去には非依存)

乗法定理

$$= \eta p(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{m}) p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}, \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m}) p(\mathbf{x}_{1:t-1} \mid \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m})$$

条件付き独立性
(1次マルコフ性)

条件付き独立性

$$= \eta p(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{m}) p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t) p(\mathbf{x}_{1:t-1} \mid \mathbf{u}_{1:t-1}, \mathbf{z}_{1:t-1}, \mathbf{m})$$

尤度
(計測モデル)

動作モデル

事前確率

時刻1からt-1の走行軌跡 $\mathbf{x}_{1:t-1}$

漸化式として定式化

乗法定理の適用 (Particle Filter の場合)

□ 乗法定理 (条件付き確率)

$$p(A, B) = p(A | B) p(B)$$

□ 条件付き乗法定理 (3変数での条件付き確率)

$$p(A, B | C) = p(A | B, C) p(B | C)$$

□ 自己位置推定に適用

$$p(\mathbf{x}_{1:t} | \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m})$$

$$= p(\mathbf{x}_t | \mathbf{x}_{1:t-1}, \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m}) p(\mathbf{x}_{1:t-1} | \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}, \mathbf{m})$$

x : ロボット位置

m : 地図

u : 制御動作 (オドメトリ)

z : 環境計測 (レーザースキャン)

Adaptive Monte Carlo Localization

- Particle Filter による2次元3自由度での確率的自己位置推定 (Monte Carlo Localization)
 - オドメトリ動作モデル
 - ビーム計測モデル / 尤度場計測モデル
- KLD-Sampling による Adaptive MCL
推定位置の不確かさに応じて、**パーティクル数を適応的に調整**
- Augmented MCL
真値周辺のパーティクルの喪失に対処するために、**ランダムパーティクルを挿入**

Adaptive Monte Carlo Localization

処理イメージ(平面図)

1. 予測

動作モデルにより
時刻 t の
ロボット位置を予測

パーティクルによって表される
ロボット位置の複数仮説

時刻: $t-1$

時刻: t

●→ :ロボット位置

Adaptive Monte Carlo Localization

処理イメージ(平面図)

2. 尤度計算

計測モデルにより
尤度を計算

スキャンと占有格子地図の
重なり度合いを評価

■ : 占有格子地図

●●●●● : 現時刻のスキャン計測

○ : 尤度

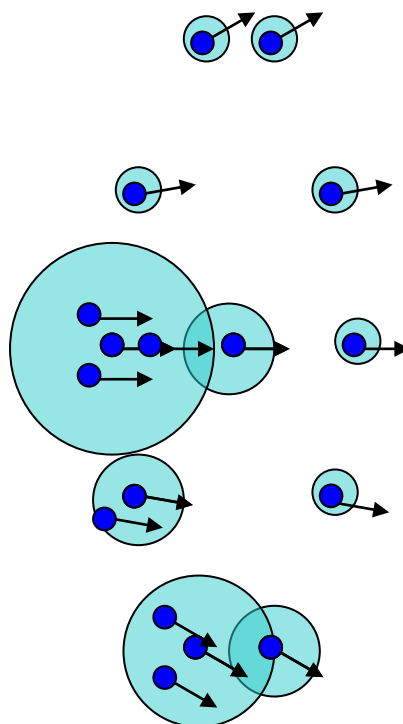
事前に与えられた1つの地図に対して、各パーティクルのスキャンを評価

Adaptive Monte Carlo Localization

処理イメージ(平面図)

3. リサンプリング

- 尤度の高いパーティクルは自身のコピーを多く残す
- 尤度の低いパーティクルは消滅
- **KLD-Sampling** により、推定位置の不確かさに応じて**パーティクル数を調整**



● : 尤度

1. 自律走行を実現するための ROS パッケージ
2. 自己位置推定と SLAM の基本
3. 各パッケージのアルゴリズム
 - a. 自己位置推定 (amcl)
 - b. SLAM、地図生成 (gmapping)
 - c. 大域的経路計画、局所的動作計画 (move_base)
4. ROS に関する情報の調べ方
5. ROS での開発に関する知見
6. まとめ

Rao-Blackwellized Particle Filter の定式化

ロボット走行軌跡と地図を求める完全 SLAM、
独立性の仮定により因子分解して解く

$p(\mathbf{x}_{1:t}, \mathbf{m} \mid \mathbf{u}_{1:t}, \mathbf{z}_{1:t})$ ← ロボット走行軌跡と地図の同時確率

$$= p(\mathbf{x}_{1:t} \mid \mathbf{u}_{1:t}, \mathbf{z}_{1:t}) \prod_{n=1}^N p(\mathbf{m}_n \mid \mathbf{x}_{1:t}, \mathbf{z}_{1:t})$$

ロボット走行軌跡 $\mathbf{x}_{1:t}$
Particle Filter で推定

地図 \mathbf{m}
Binary Bayes Filter で推定

\mathbf{x} : ロボット位置

\mathbf{u} : 制御動作 (オドメトリなど)

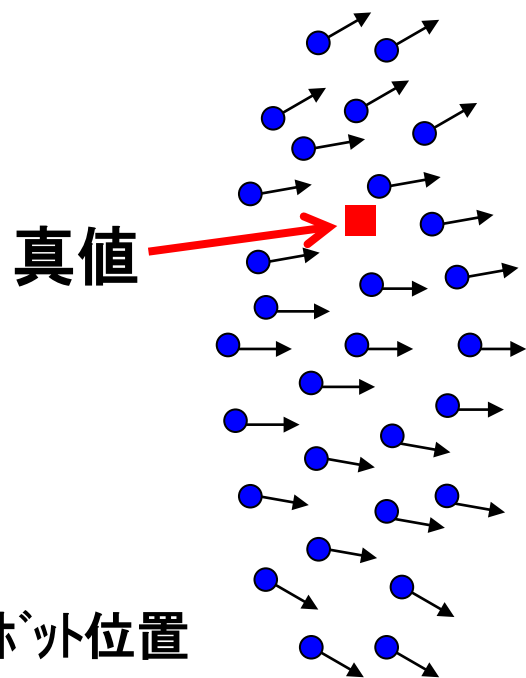
\mathbf{m} : 地図

\mathbf{z} : 環境計測 (レーザースキャンなど)

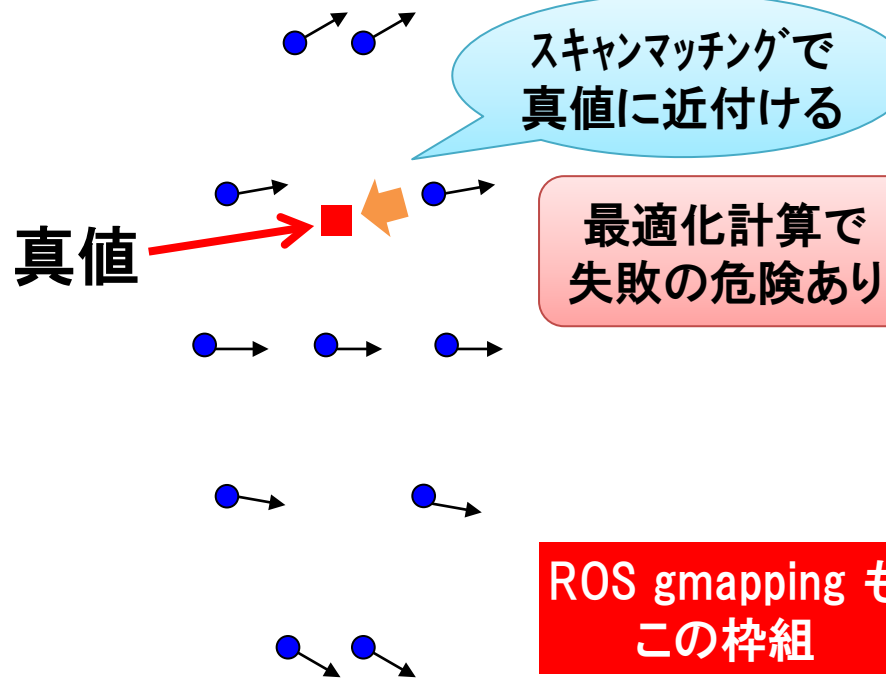
提案分布における最適化計算の利用

計算コストはパーティクル数に依存するので、
少数のパーティクルそれぞれで最適化計算を併用

多数のパーティクルによる提案分布
(高精度だが計算コスト大)



少数のパーティクルと最適化計算
による提案分布 (FastSLAM 2.0)

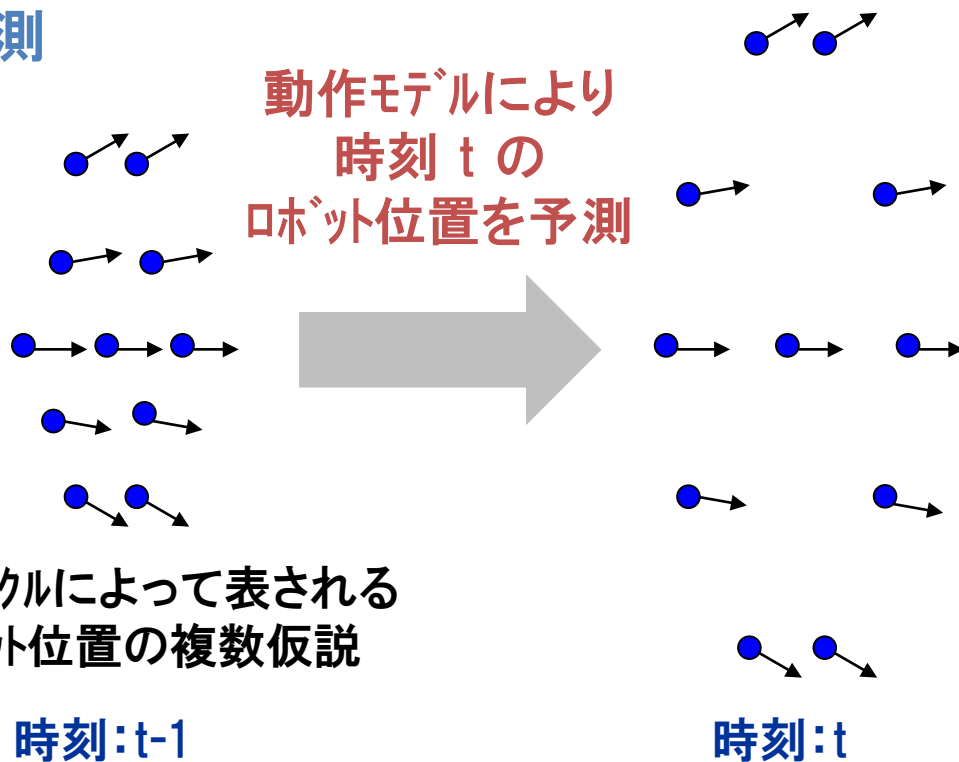


Rao-Blackwellized Particle Filter SLAM

処理イメージ(平面図)

1. 予測

動作モデルにより
時刻 t の
ロボット位置を予測



パーティクルによって表される
ロボット位置の複数仮説

●→ :ロボット位置

Rao-Blackwellized Particle Filter SLAM

処理イメージ(平面図)

2. 尤度計算

計測モデルにより
尤度を計算

スキャンを占有格子地図に
Greedy Scan Matching
(FastSLAM 2.0)

■ : 占有格子地図

●●●●● : 現時刻のスキャン計測

● : 尤度

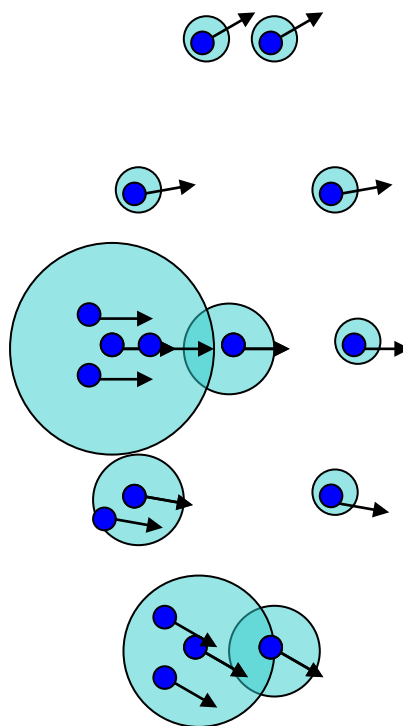
Rao-Blackwellization により各パーティクルごとに地図を持っている

Rao-Blackwellized Particle Filter SLAM

処理イメージ(平面図)

3. リサンプリング

- 尤度の高いパーティクルは自身のコピーを多く残す
- 尤度の低いパーティクルは消滅

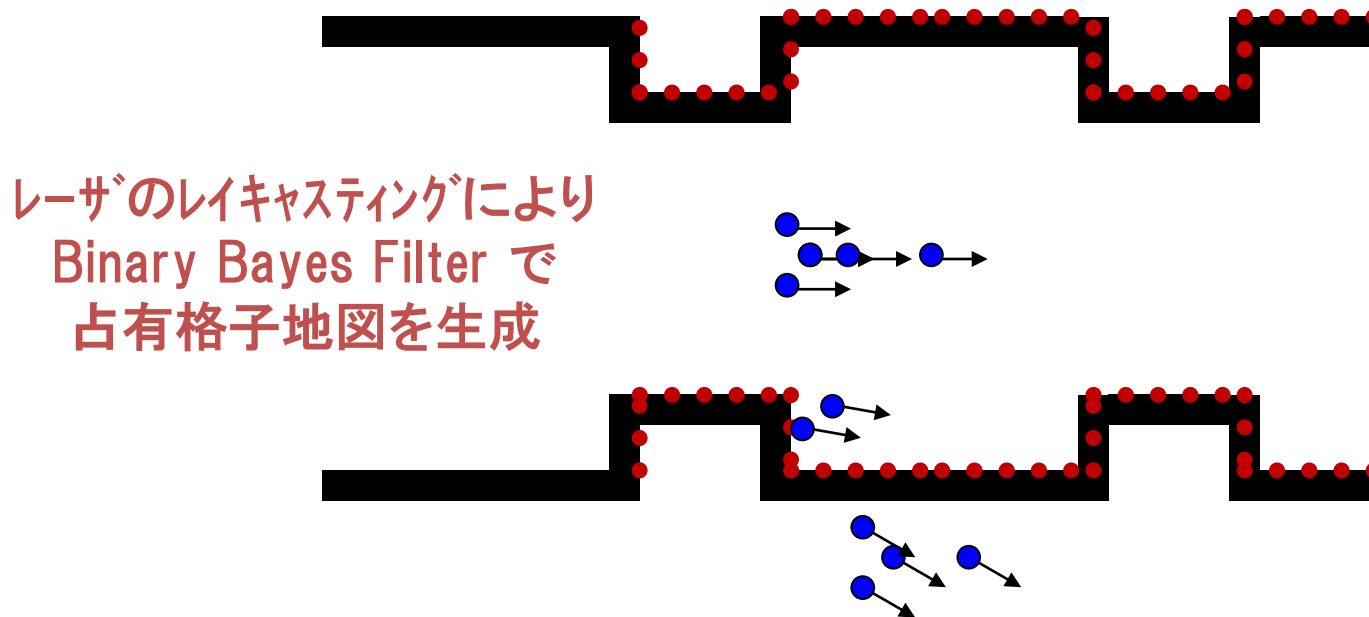


● : 尤度

Rao-Blackwellized Particle Filter SLAM

処理イメージ(平面図)

4. 地図生成



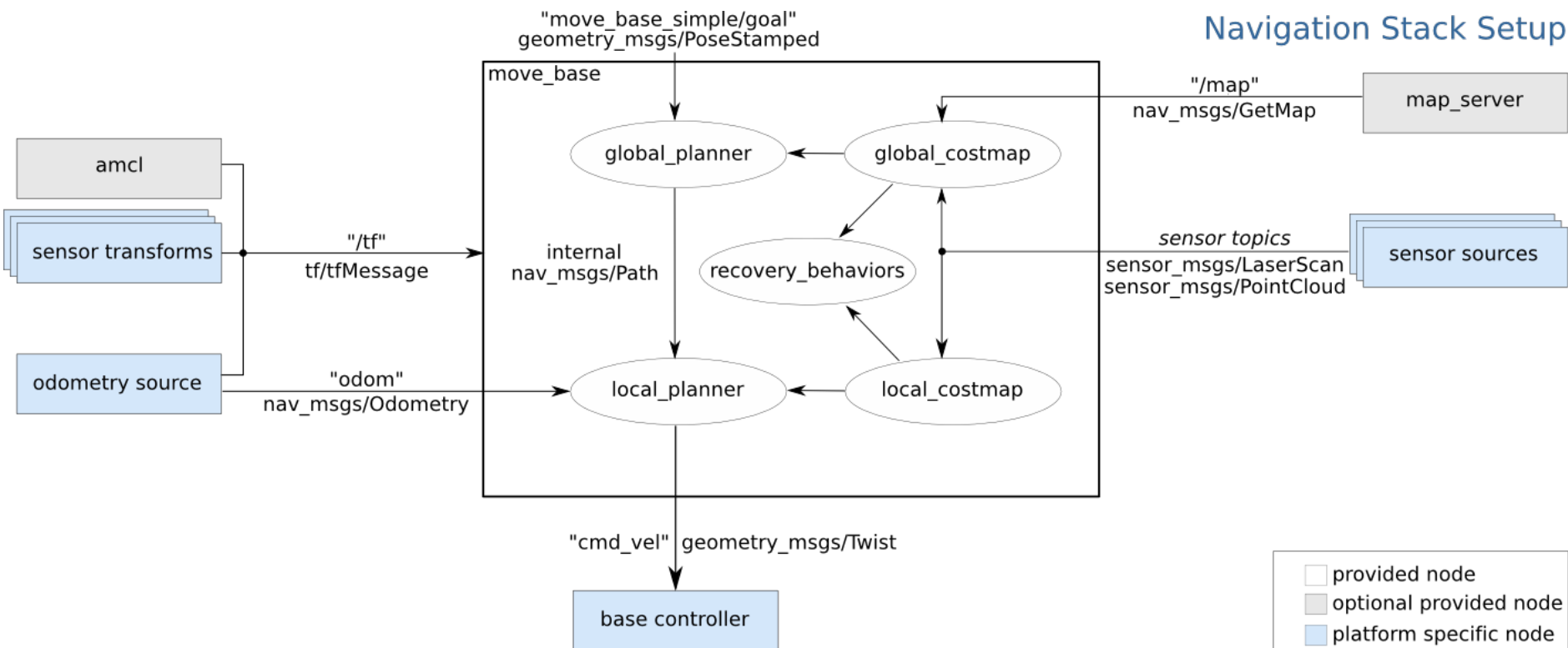
■■■■■ : 占有格子地図

●●●●● : 現時刻のスキャン計測

Rao-Blackwellization により各パーティクルごとにそれぞれの地図を生成する

1. 自律走行を実現するための ROS パッケージ
2. 自己位置推定と SLAM の基本
3. 各パッケージのアルゴリズム
 - a. 自己位置推定 (amcl)
 - b. SLAM、地図生成 (gmapping)
 - c. 大域的経路計画、局所的動作計画 (move_base)
4. ROS に関する情報の調べ方
5. ROS での開発に関する知見
6. まとめ

自律走行時のデータの流れ

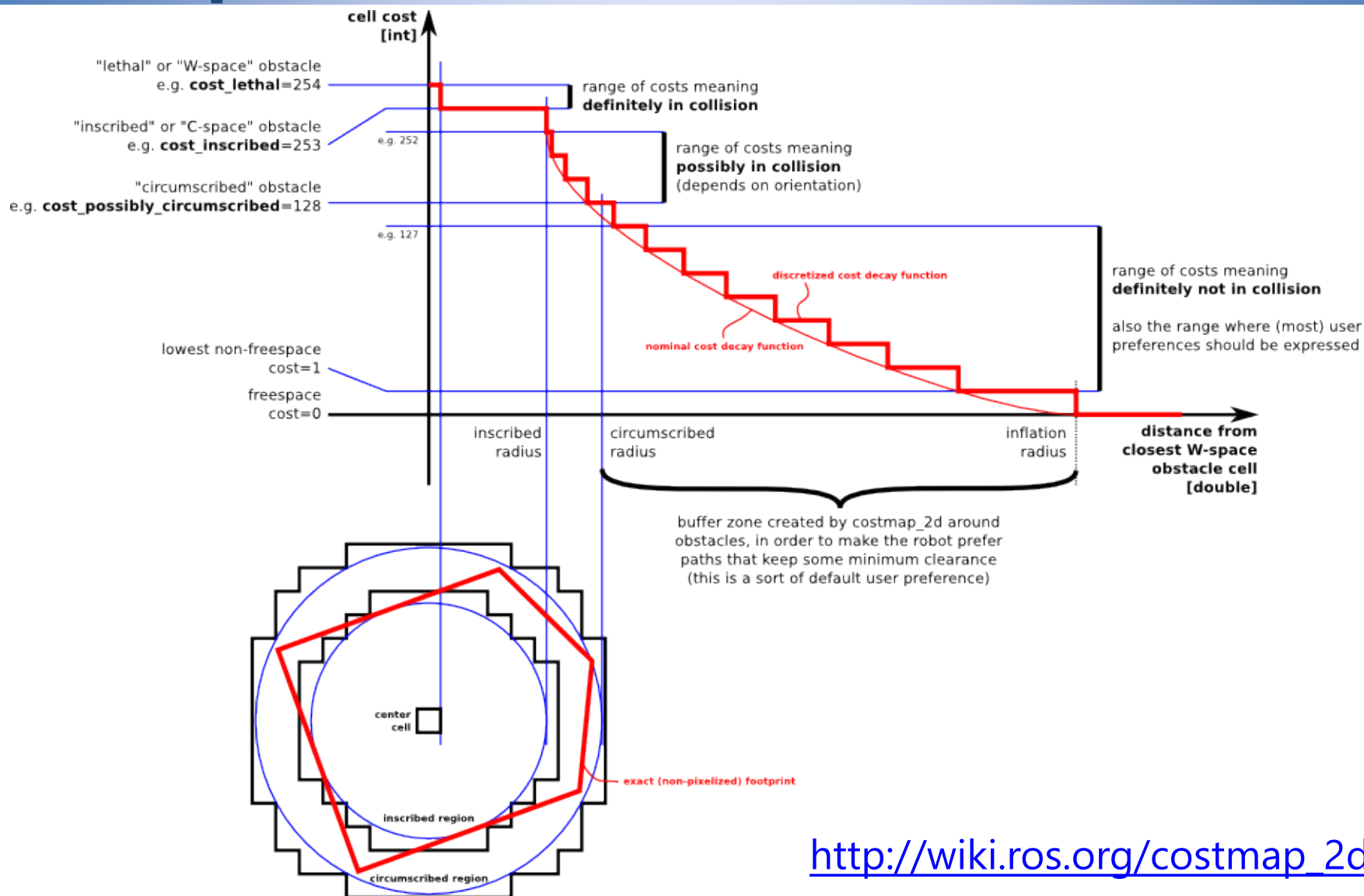


http://wiki.ros.org/move_base

costmap_2d によるコストマップの生成

- レーザースキャンまたは3次元点群のデータを用いて、**2次元／3次元の占有格子地図**を生成
- レイキャスティングにより、占有／フリー／未知の3状態で障害物情報を管理
- 占有格子地図の障害物を味ダットの内接円半径の大きさに膨張させ、 **x, y の2自由度コンフィギュレーション空間**として2次元コストマップを生成
- **グローバルコストマップ**と**ローカルコストマップ**の2種類を生成

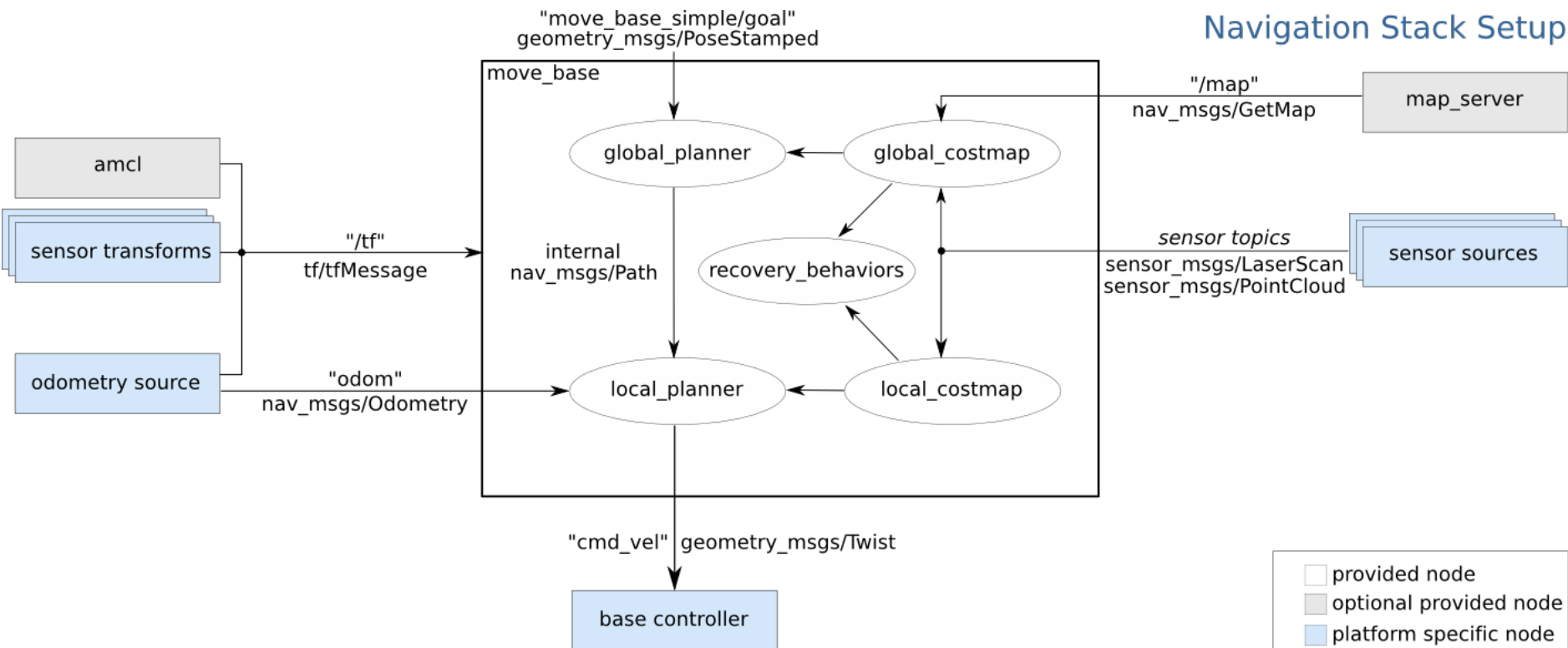
costmap_2d によるコストマップの生成



コストマップ° を生成する枠組み

- ROS Hydro 以降、costmap_2d の構成が変更
 - http://wiki.ros.org/costmap_2d
 - http://wiki.ros.org/costmap_2d/hydro
 - http://wiki.ros.org/hydro/Migration#Navigation_-_Costmap2D
- LayeredCostmap という仕組みが導入された
 - Static Map Layer : SLAM で生成した静的地図
 - Obstacle Map Layer : 検出した障害物の蓄積と消去 (ObstacleCostmapPlugin / VoxelCostmapPlugin)
 - Inflation Layer : 障害物を膨張したコンフィギュレーション空間
- ROS の pluginlib という機能で実現されている

自律走行時のデータの流れ

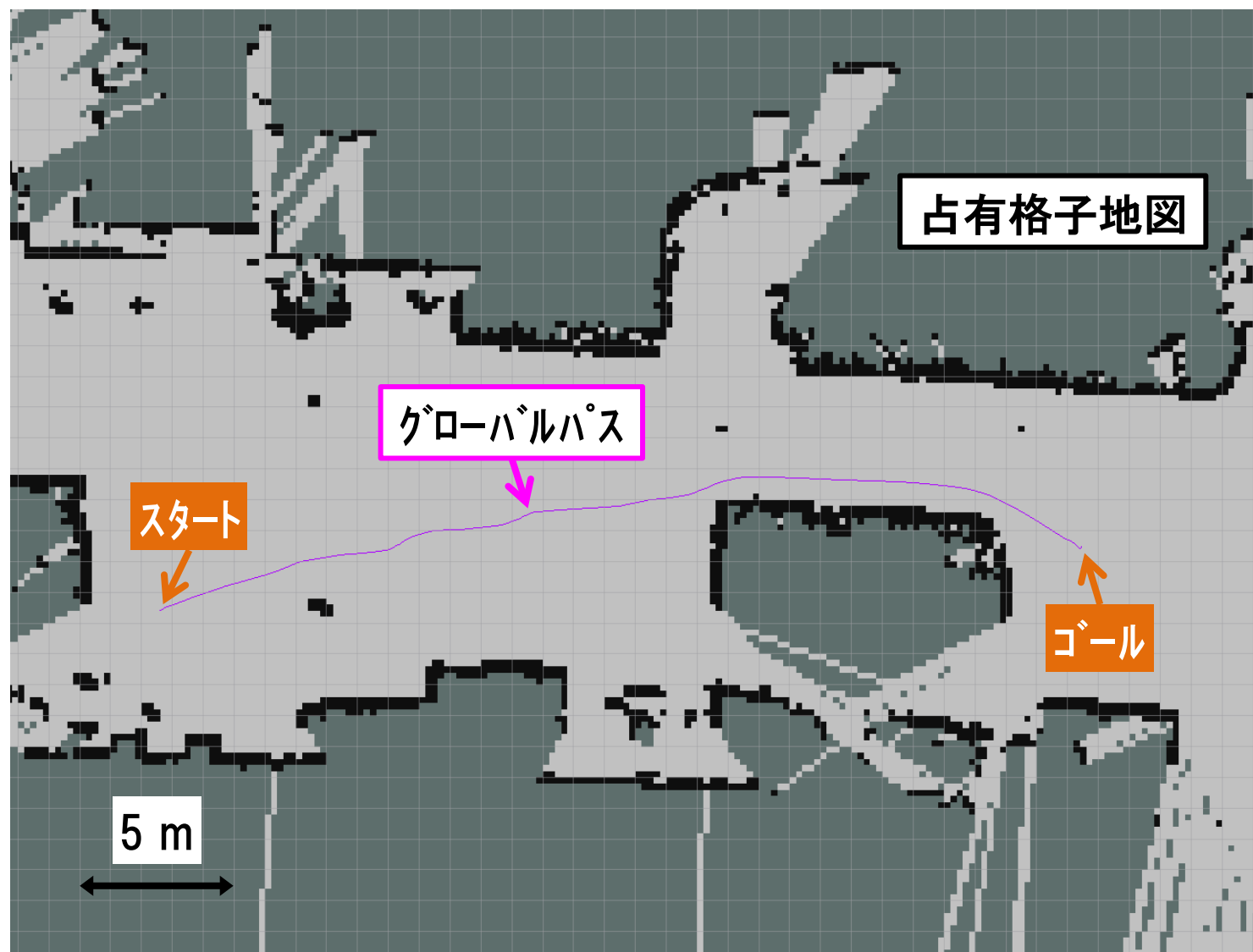


http://wiki.ros.org/move_base

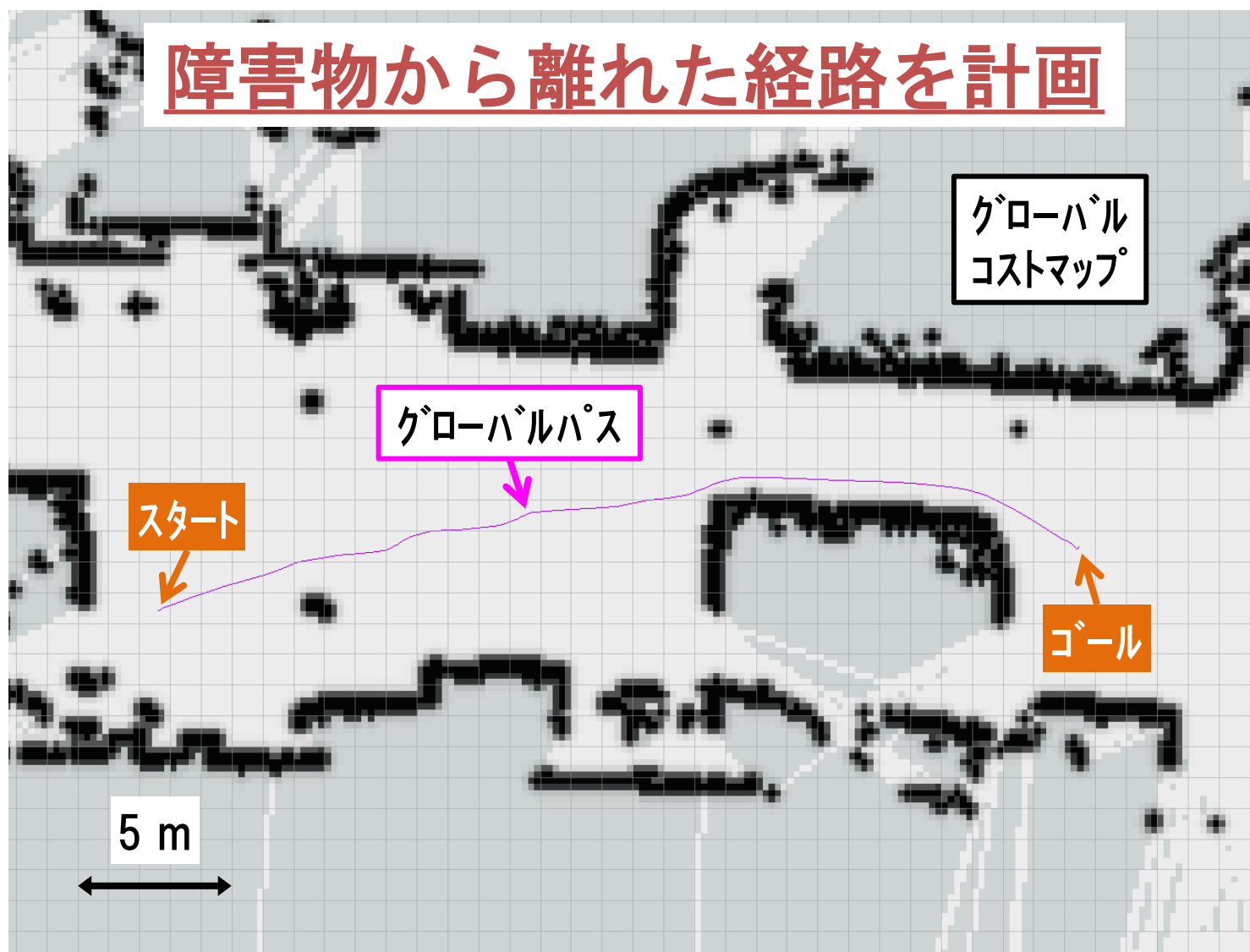
navfn の大域的経路計画

- **Navigation Function** と呼ばれるポテンシャル場で評価
 - **ゴールまでの距離**と**障害物からの距離**をコストとする
 - グローバルコストマップに基づいて処理
- 格子地図に対して、**ダイクストラ法**でポテンシャル場の最小コストとなる経路を探索
 - 各セルのポテンシャル場のコストを計算しながらダイクストラ法で探索し、ゴールまでのグローバルパスを計画
- **ロボットの方向は考慮しない**（円形と仮定）
 - 方向によっては障害物と衝突する経路が計画される

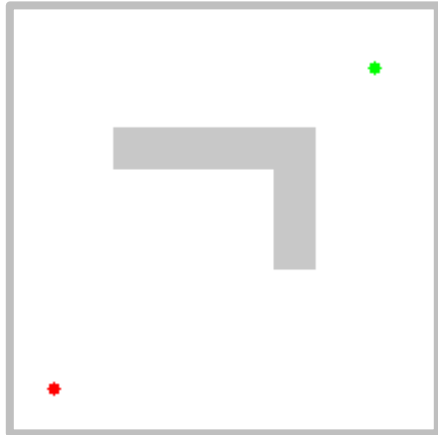
navfn による大域的経路計画の例



navfn による大域的経路計画の例

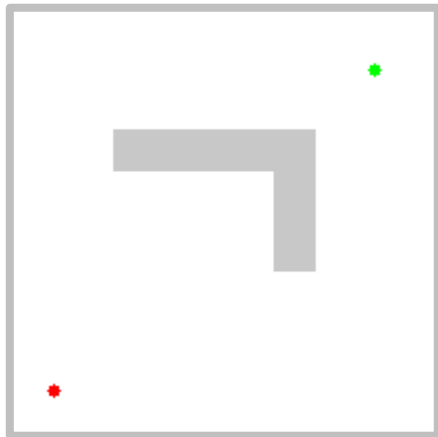


ダイクストラ法とA*の違い



□ ダイクストラ法

幅優先探索をコストを評価するように拡張した、最良優先探索と呼ばれる手法の一種



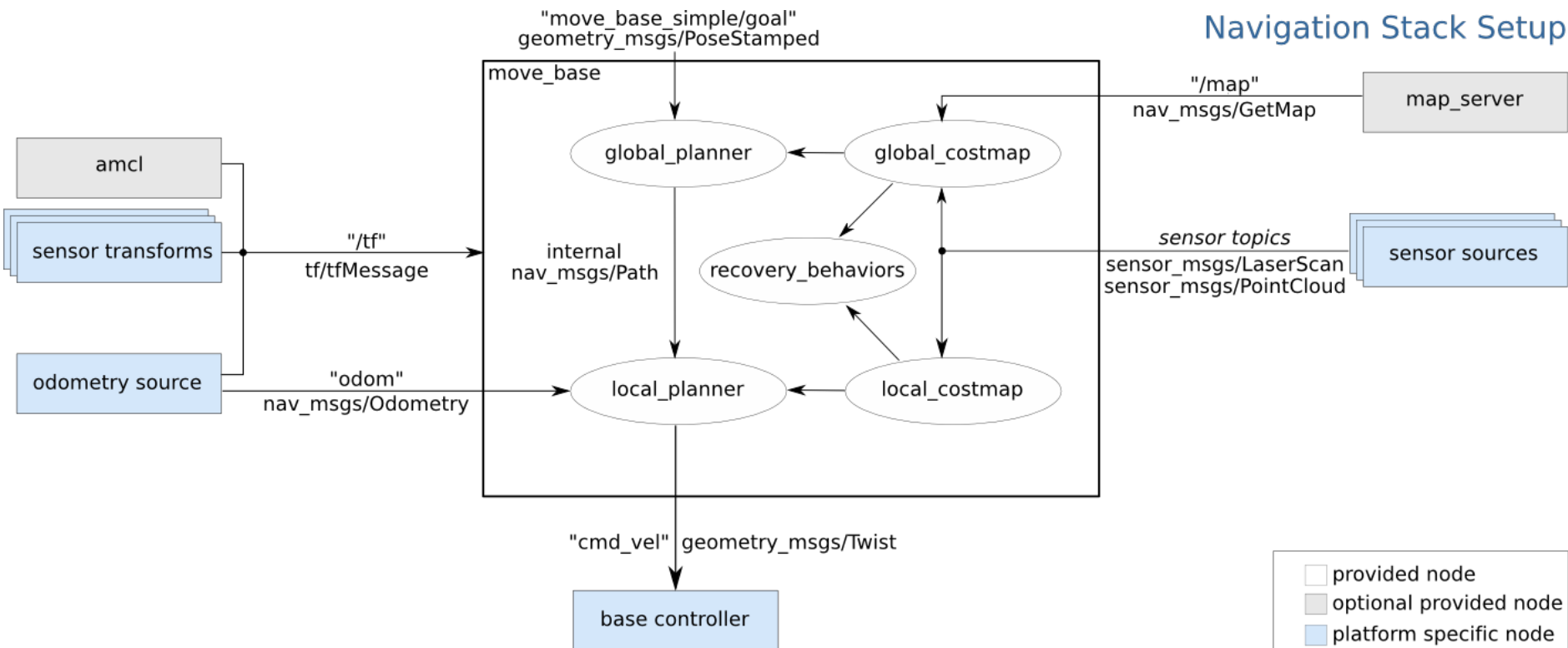
□ A* (A-Star)

ヒューリスティック関数でコストの予測値を追加した、ダイクストラ法の改良版

https://en.wikipedia.org/wiki/Dijkstra's_algorithm

https://en.wikipedia.org/wiki/A*_search_algorithm

自律走行時のデータの流れ



http://wiki.ros.org/move_base

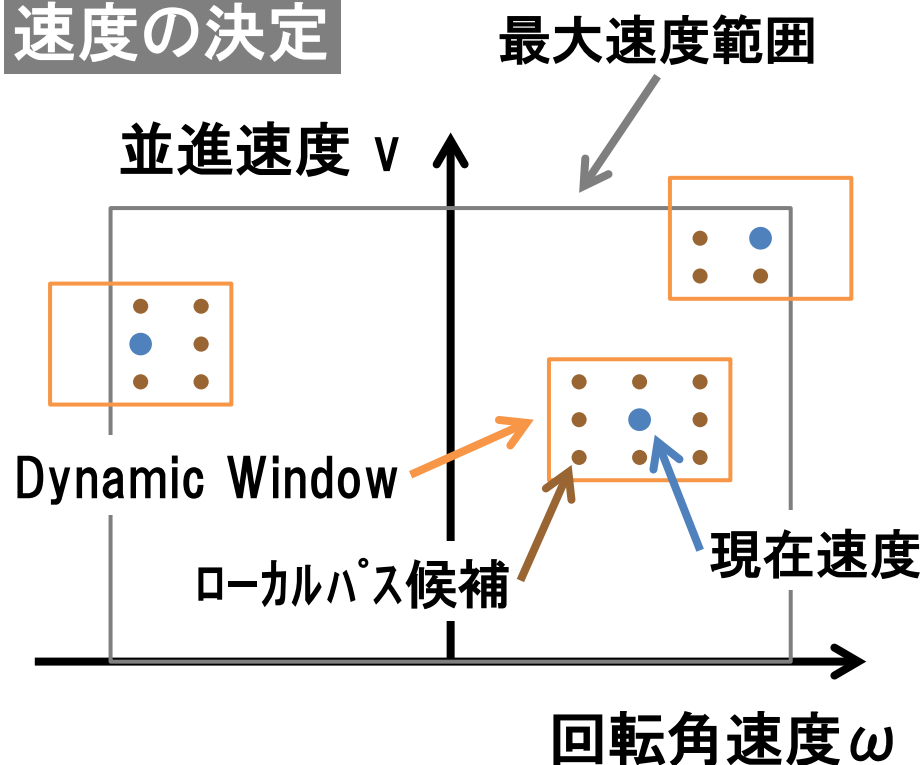
base_local_planner の局所的動作計画

- 大域的経路計画によるグローバルパスに追従しつつ、障害物を回避する動作（速度）を計画
- **Dynamic Window Approach** を使用
 - ロボットの**キネマティクス（運動モデル）**を考慮した軌跡
 - 現在の速度に基づき、**ダイナミクス**を考慮して実行可能な複数のローカルパス候補を生成
 - 評価関数により、グローバルパスに近く、かつ障害物から離れたローカルパスの動作を選択
 - ローカルコストマップに基づいて処理

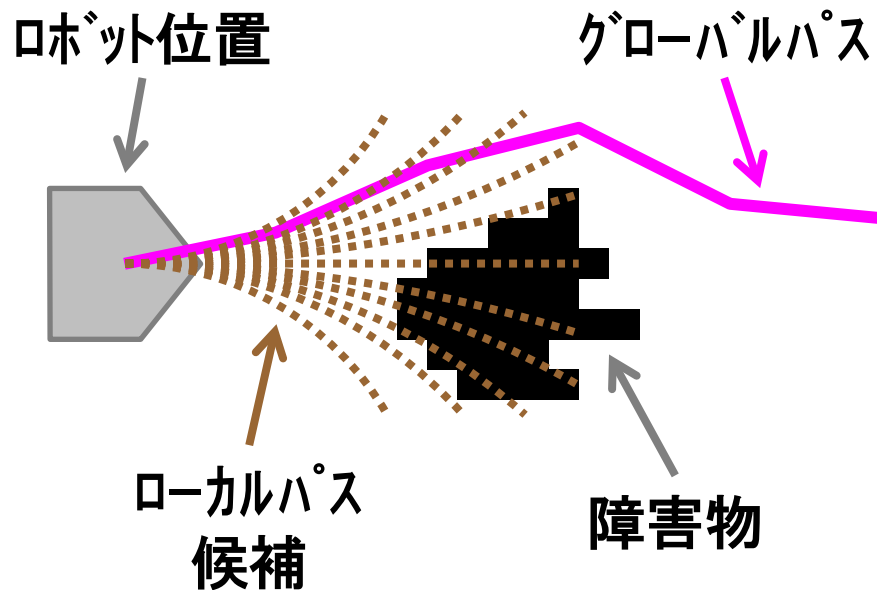
Dynamic Window Approach

キネマティクスとダイナミクスを考慮してローカルパス候補の軌跡を生成、離散化してコストを評価

速度の決定



平面図



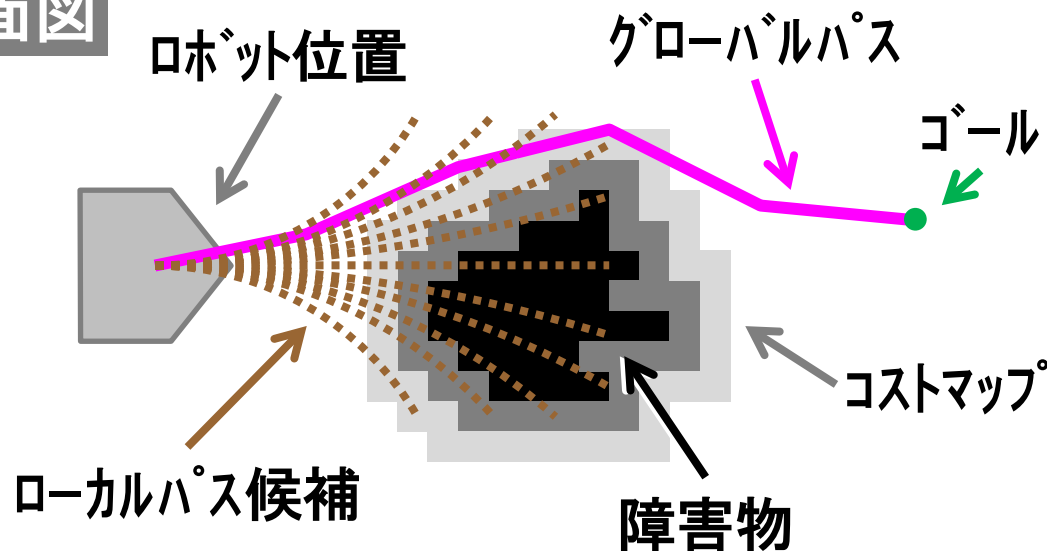
Dynamic Window Approach のコスト評価関数

軌跡コスト =

経路距離スケール × ローカルパス端点とグローバルパスの距離
 + ゴール距離スケール × ローカルパス端点とゴールの距離
 + 障害物スケール × ローカルパス上での最大障害物コスト

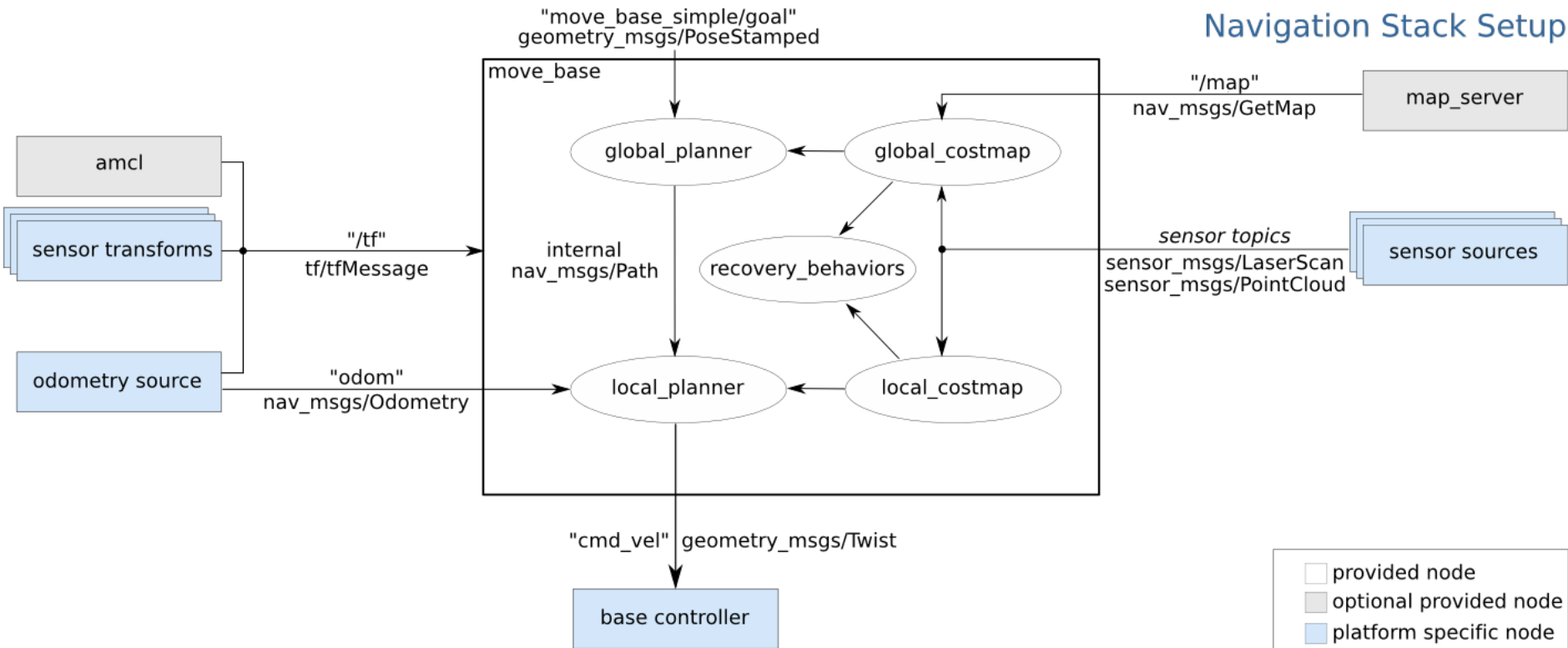
上記の ROS の実装は、オリジナル論文の評価関数とは異なる

平面図



- 距離は各セルごとに事前計算
- ローカルパスの振動を抑制するため、一定距離は逆符号の速度を無効化

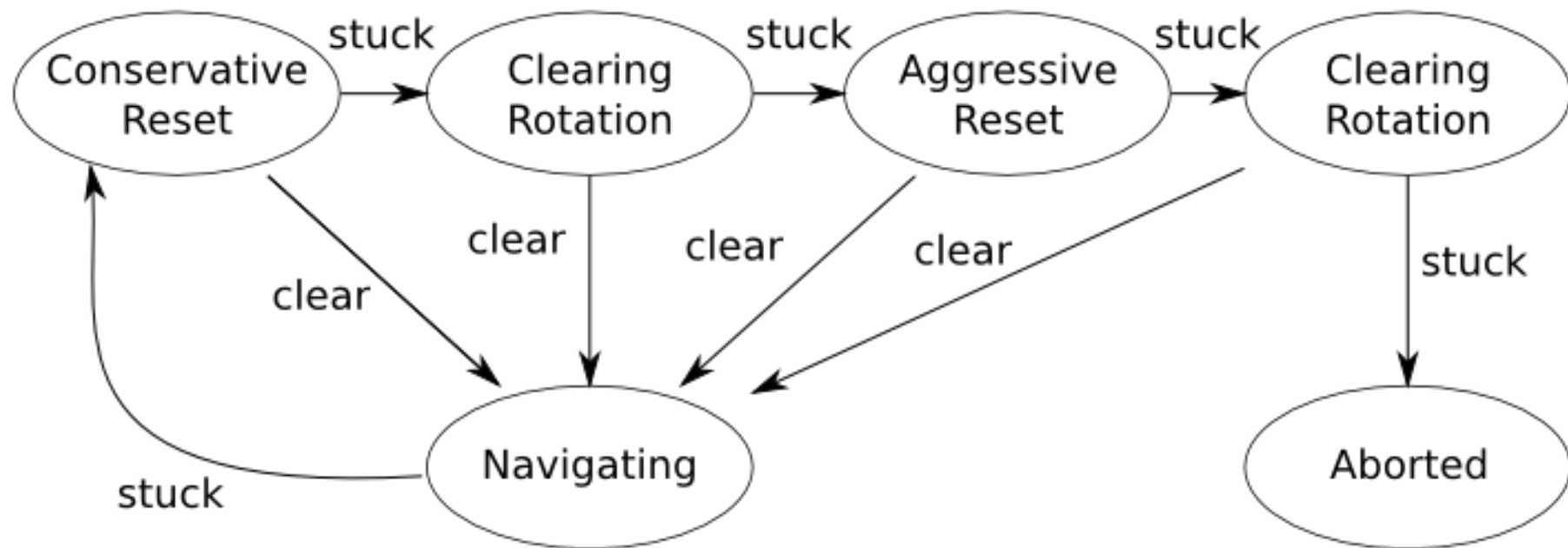
自律走行時のデータの流れ



http://wiki.ros.org/move_base

スタックした場合の復帰動作の状態遷移

move_base Default Recovery Behaviors



http://wiki.ros.org/move_base

1. 自律走行を実現するための ROS パッケージ
2. 自己位置推定と SLAM の基本
3. 各パッケージのアルゴリズム
 - a. 自己位置推定 (amcl)
 - b. SLAM、地図生成 (gmapping)
 - c. 大域的経路計画、局所的動作計画 (move_base)
- 4. ROS に関する情報の調べ方**
5. ROS での開発に関する知見
6. まとめ

ROSに関する情報源

- **ROS.org** (パッケージリスト、ニュースなど)
<https://www.ros.org/>
- **ROS Wiki** (共同編集文章)
<https://wiki.ros.org/>
- **ROS Answers** (質問掲示板)
<https://answers.ros.org/>
- **ROS Index** (パッケージ情報、2015年ローンチ)
<https://index.ros.org/>
- **GitHub Organizations** (ソースコード、Issue Trackers)
<https://github.com/ros-planning>
<https://github.com/ros-perception> など
- **C++ / Python API** (Doxygen / Sphinx 生成文章)
<https://docs.ros.org/en/api/> 以下の各ディレクトリ
- **ROS Discourse**
<https://discourse.ros.org/>

情報が実装と一致しない場合も多い
(特に日本語ページは古い)

ROS Index (パッケージ情報を集約)

README や package.xml からの自動生成文章

ROS Index BETA ABOUT INDEX DOC CONTRIBUTE STATS Search

Home > Packages > navigation

KINETIC JADE INDIGO HYDRO OLDER

navigation package from navigation repo

amcl base_local_planner carrot_planner clear_costmap_recovery costmap_2d dwa_local_planner fake_localization
global_planner map_server move_base move_slow_and_clear nav_core navfn navigation robot_pose_ekf
rotate_recovery voxel_grid

GITHUB-ROS-PLANNING-NAVIGATION

Overview 0 Assets 17 Dependencies 8 Tutorials

Package Summary

Tags No category tags.

Version 1.14.0

License BSD,LGPL,LGPL (amcl)

Buildtool CATKIN

Use RECOMMENDED

Repository Summary

Checkout URI `https://github.com/ros-planning/navigation`

Package Description

A 2D navigation stack that takes in information from odometry, sensor streams, and a goal pose and outputs safe velocity commands that are sent to a mobile base.

Additional Links

Website

Maintainers

David V. Lu!!
Michael Ferguson

API Docs
Browse Code
Get Help
Wiki
Eco

GitHub などのリンク

GitHub Organizations (コード、Issues)

パッケージの種類ごとに組織されている

The screenshot shows the GitHub repository page for 'ros-planning/navigation'. The repository is part of the 'ros-planning' organization. It has 65 watchers, 106 stars, and 312 forks. The repository description is 'ROS Navigation stack. Code for finding where the robot is and how it can get somewhere else.' The repository statistics show 1,620 commits, 10 branches, 100 releases, and 60 contributors. The current branch is 'jade-devel'. The repository contains several sub-packages, each with a commit message and a timestamp.

Package	Commit Message	Timestamp
amcl	Allow AMCL to run from bag file to allow very fast testing.	2 months ago
base_local_planner	1.13.1	8 months ago
carrot_planner	1.13.1	8 months ago
clear_costmap_recovery	1.13.1	8 months ago
costmap_2d	Fixed bug with inflation layer that caused underinflation	11 days ago
dwa_local_planner	1.13.1	8 months ago
fake_localization	1.13.1	8 months ago
global_planner	1.13.1	8 months ago
map_server	Corrections to alpha channel detection and usage.	4 months ago
move_base	1.13.1	8 months ago
move_slow_and_clear	1.13.1	8 months ago
nav_core	1.13.1	8 months ago

ROS C++/Python API (Doxygen/Sphinx)

ソースコードのコメントからの自動生成文章

The screenshot shows a web browser displaying the ROS documentation for the `slam_gmapping` package. The browser address bar shows `docs.ros.org/api/gmapping/html/`. The page title is `slam_gmapping`. The main content area is divided into sections: **slam_gmapping**, **ROS topics**, **services**, and **ROS parameters**. A sidebar on the right contains a **gmapping** section with a description and a link to the homepage.

slam_gmapping is a wrapper around the GMapping SLAM library. It reads laser scans and odometry and computes a map. This map can be written to a file using e.g.

```
"roslaunch map_server map_saver static_map:=dynamic_map"
```

ROS topics

Subscribes to (name/type):

- "scan"/ `sensor_msgs/LaserScan` : data from a laser range scanner
- "/tf": odometry from the robot

Publishes to (name/type):

- "/tf"/`tf/tfMessage`: position relative to the map

services

- "~/dynamic_map" : returns the map

ROS parameters

Reads the following parameters from the parameter server

Parameters used by our GMapping wrapper:

- "~/throttle_scans": [int] throw away every nth laser scan
- "~/base_frame": [string] the tf frame_id to use for the robot base pose
- "~/map_frame": [string] the tf frame_id where the robot pose on the map is published
- "~/odom_frame": [string] the tf frame_id from which odometry is read
- "~/map_update_interval": [double] time in seconds between two recalculations of the map

gmapping

This package contains a ROS wrapper for OpenSlam's Gmapping. The gmapping package provides laser-based SLAM (Simultaneous Localization and Mapping), as a ROS node called `slam_gmapping`. Using `slam_gmapping`, you can create a 2-D occupancy grid map (like a building floorplan) from laser and pose data collected by a mobile robot.

- Homepage: <http://wiki.ros.org/gmapping>

自律走行の実現に役立つチュートリアルページ

□ ROS 概要

<https://wiki.ros.org/ROS/StartGuide>

□ ROS 基本チュートリアル（プロセス間通信、開発ツールなど）

<https://wiki.ros.org/ROS/Tutorials>

https://wiki.ros.org/roscpp_tutorials/Tutorials

□ tf チュートリアル（座標変換）

<https://wiki.ros.org/tf/Tutorials>

<https://wiki.ros.org/tf2/Tutorials>

□ navigation チュートリアル（自律走行）

<https://wiki.ros.org/navigation/Tutorials/RobotSetup>

<https://wiki.ros.org/navigation/Tutorials/Navigation%20Tuning%20Guide>

□ slam_gmapping チュートリアル（SLAM、地図生成）

https://wiki.ros.org/slam_gmapping/Tutorials/MappingFromLoggedData

□ rviz チュートリアル（ビューア）

<https://wiki.ros.org/rviz/Tutorials>

<https://wiki.ros.org/navigation/Tutorials/Using%20rviz%20with%20the%20Navigation%20Stack>

1. 自律走行を実現するための ROS パッケージ
2. 自己位置推定と SLAM の基本
3. 各パッケージのアルゴリズム
 - a. 自己位置推定 (amcl)
 - b. SLAM、地図生成 (gmapping)
 - c. 大域的経路計画、局所的動作計画 (move_base)
4. ROS に関する情報の調べ方
5. ROS での開発に関する知見
6. まとめ

ROSパッケージの様々な使い方

□ ROS API で既存パッケージを launch するだけ

- ROS が想定している基本的な使い方
- **メッセージ型は比較的良く整理されている**

パラメータ調整には
アルゴリズムの
理解が必要

□ C++/Python API で使用 (ライブラリとして使用)

- アルゴリズムの中身を個別に使える
- ライブラリとしては整理が不十分なパッケージも多い

□ 既存パッケージの中身を改造

- ソースコードが整理されておらず、苦勞する場合もある

□ 新規パッケージを作成

- 標準的なメッセージ型に従えば、既存パッケージと連携可

slam_gmapping, navigation にない機能

- 3次元空間の SLAM、地図生成
- Graph-based SLAM
- PF による3次元6自由度での自己位置推定
(EKF は robot_pose_ekf パッケージで可能)
- 3次元点群を用いた SLAM / 自己位置推定
- x, y, yaw の3自由度コンフィギュレーション空間
- 3次元6自由度での経路計画
- 3次元点群を用いた段差などの障害物の検出
- 移動障害物の検出と追跡

ROS のビルドシステム

- **rosbuild** (2012年以前の古いシステム)
- **catkin** (**catkin_make** コマンド、**全パッケージ同時ビルド**)
<https://wiki.ros.org/catkin>
 ワークスペース src/ 直下にトップレベル CMakeLists.txt あり
 WS初期化 \$ catkin_init_workspace
 パッケージ生成 \$ catkin_create_pkg <pkg_name>
 ビルド \$ catkin_make
- **catkin_tools** (**catkin** コマンド、**各パッケージ個別ビルド**)
 catkin_make_isolated の後継、2015年ローンチ、ベータ版
<https://catkin-tools.readthedocs.io/>
 ワークスペース src/ 直下にトップレベル CMakeLists.txt なし
 WS初期化 \$ catkin init
 パッケージ生成 \$ catkin create <pkg_name>
 ビルド \$ catkin build

catkin ワークスペースの構成に関する Tips

□ ファイルツリーは階層的なディレクトリでも良い

catkin_ws/
src/

(catkin ワークスペース)

(src ディレクトリ)

stack_dir/

(**複数パッケージを集約**)

pkg_a/

(パッケージ a)

pkg_b/

(パッケージ b)

□ 複数のワークスペースをオーバーレイできる

https://wiki.ros.org/catkin/Tutorials/workspace_overlaying

- 同名のパッケージをオーバーレイ可能

既存パッケージの改造に便利

- 初回の catkin_make コマンド実行時に読み込んでいた設定が新しく生成される setup.bash に引き継がれる
- setup.bash は最後に読み込んだもので**上書き**される

Subscriber に登録できるコールバックの型

boost::function でサポートされた型を登録可能

<https://wiki.ros.org/roscpp/Overview/Publishers%20and%20Subscribers>

□ 通常の間数

□ メンバ 関数（クラスのメソッド）

https://wiki.ros.org/roscpp_tutorials/Tutorials/UsingClassMethodsAsCallbacks

□ 関数ポインタ

□ 関数オブジェクト（ファンクタ）

□ boost::bind

- コールバック関数に引数を渡す場合に使用
- C++11 の bind には ROS が未対応

Subscriber へのコールバック関数の登録方法

- 通常の間数、引数なし
- 通常の間数、引数あり ← boost::bind 使用
- メンバ関数、引数なし
(クラスのオブジェクト/ポインタ経由で登録)
- メンバ関数、引数あり ← boost::bind 使用
(クラスのオブジェクト/ポインタ経由で登録)
- メンバ関数、クラス内 Subscriber、引数なし
- メンバ関数、クラス内 Subscriber、引数あり
← boost::bind 使用



推奨

通常関数、引数なし

```
void cbFunc(const std_msgs::String::ConstPtr &msg) {  
    // コールバック処理  
}
```

```
int main(int argc, char **argv) {  
    ros::init(argc, argv, "subscriber_example");  
    ros::NodeHandle nh;  
    ros::Subscriber sub  
        = nh.subscribe("topic_name", 100, cbFunc);  
    ros::spin();  
    return 0;  
}
```

Subscriber のテンプレート型は
基本的には省略可能

通常の変数、引数あり（値渡し）

```
void cbFuncCopy(const std_msgs::String::ConstPtr &msg, int num) {  
    // コールバック処理  
}
```

```
int main(int argc, char **argv) {  
    ros::init(argc, argv, "subscriber_example");  
    ros::NodeHandle nh;  
    int num = 1234;  
    ros::Subscriber sub  
        = nh.subscribe<std_msgs::String>("topic_name", 100,  
            boost::bind(cbFuncCopy, _1, num));  
    ros::spin();  
    return 0;  
}
```

bind の場合は Subscriber の
テンプレート型を省略できない

C++11 の bind は未対応

通常の変数、引数あり（参照渡し）

```
void cbFuncRef(const std_msgs::String::ConstPtr &msg,
               std::vector<int> &obj) {
    // コールバック処理
}

int main(int argc, char **argv) {
    ros::init(argc, argv, "subscriber_example");
    ros::NodeHandle nh;
    std::vector<int> obj;
    ros::Subscriber sub
        = nh.subscribe<std_msgs::String>("topic_name", 100,
            boost::bind(cbFuncRef, _1, std::ref(obj)));
    ros::spin();
    return 0;
}
```

参照渡しは ref() / cref() が必要
(C++11 と boost のどちらも OK)

メンバ関数、引数なし (クラスのオブジェクト)

```
class Foo {
public:
    void cbMethod(const std_msgs::String::ConstPtr &msg) {
        // コールバック処理
    }
};

int main(int argc, char **argv) {
    ros::init(argc, argv, "subscriber_example");
    ros::NodeHandle nh;
    Foo foo_obj;
    ros::Subscriber sub
        = nh.subscribe("topic_name", 100, &Foo::cbMethod, &foo_obj);
    ros::spin();
    return 0;
}
```

メンバ関数、引数なし (クラスのポインタ経由)

```
class Foo {
public:
    void cbMethod(const std_msgs::String::ConstPtr &msg) {
        // コールバック処理
    }
};

int main(int argc, char **argv) {
    ros::init(argc, argv, "subscriber_example");
    ros::NodeHandle nh;
    auto foo_ptr = boost::make_shared<Foo>();
    ros::Subscriber sub
        = nh.subscribe("topic_name", 100, &Foo::cbMethod, foo_ptr);
    ros::spin();
    return 0;
}
```

C++11のshared_ptrやunique_ptrは未対応

メンバ関数、引数あり

```
class Foo {
public:
    void cbMethodArg(const std_msgs::String::ConstPtr &msg,
                    std::vector<int> &obj) {
        // コールバック処理
    }
};

int main(int argc, char **argv) {
    ros::init(argc, argv, "subscriber_example");
    ros::NodeHandle nh;
    Foo foo_obj;
    std::vector<int> obj;
    ros::Subscriber sub
        = nh.subscribe<std_msgs::String>("topic_name", 100,
            boost::bind(&Foo::cbMethodArg, &foo_obj, _1, std::ref(obj)));
    ros::spin();
    return 0;
}
```

メンバ関数、クラス内 Subscriber、引数なし

```
class BarNode {  
public:  
    BarNode() {  
        sub_ = nh_.subscribe("topic_name", 100, &BarNode::cbMethod, this);  
    }  
private:  
    void cbMethod(const std_msgs::String::ConstPtr &msg) {  
        // コールバック処理  
    }  
    ros::NodeHandle nh_;  
    ros::Subscriber sub_;  
};  
int main(int argc, char **argv) {  
    ros::init(argc, argv, "subscriber_example");  
    BarNode bar_obj;  
    ros::spin();  
    return 0;  
}
```

ROS ノード クラス

推奨

メンバ関数、クラス内 Subscriber、引数あり

```
class BarNode {  
public:  
    BarNode() {  
        std::vector<int> obj;  
        sub_ = nh_.subscribe<std_msgs::String>("topic_name", 100,  
            boost::bind(&BarNode::cbMethodArg, this, _1, obj));  
    }  
private:  
    void cbMethodArg(const std_msgs::String::ConstPtr &msg,  
        std::vector<int> obj) {  
        // コールバック処理  
    }  
    ros::NodeHandle nh_;  
    ros::Subscriber sub_;  
};  
int main(int argc, char **argv) {  
    ros::init(argc, argv, "subscriber_example");  
    BarNode bar_obj;  
    ros::spin();  
    return 0;  
}
```

ROS ノード クラス

参照渡しの場合は
obj が既に無効なので注意

統合開発環境 Qt Creator

hello_math.cpp - ymbc_tutorial - Qt Creator

ファイル(F) 編集(E) ビルド(B) デバッグ(D) 解析(A) ツール(T) ウィンドウ(W) ヘルプ(H)

プロジェクト hello_math.cpp <シンボルの選択> CMakeLists.txt 行番号: 1, 列位置: 1

```

1 #include <cstdio>
2 #include <cmath>
3 // #include <iostream>
4
5 int main(int argc, char **argv)
6 {
7     printf("Hello, world!\n");
8     // std::cout << "Hello, world with iostream!" << std::endl;
9
10    printf("sin(90 [deg]) = %f\n", sin(M_PI / 2.0));
11    printf("atan2(1.0, sqrt(3.0)) = %f [deg]\n", atan2(1.0,
12
13    return 0;
14 }
15

```

```

1 cmake_minimum_required(VERSION 2.8 FATAL_ERROR)
2
3 project(ymbc_tutorial)
4
5 # set the default path for built executables to the "bin" directory
6 set(EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
7 # set the default path for built libraries to the "lib" directory
8 set(LIBRARY_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/lib)
9
10 # set the build type
11 set(CMAKE_BUILD_TYPE Release)
12 # confirmation messages
13 message(STATUS "CMAKE_BUILD_TYPE: ${CMAKE_BUILD_TYPE}")
14 message(STATUS "CMAKE_C_FLAGS: ${CMAKE_C_FLAGS}")
15 message(STATUS "CMAKE_C_FLAGS_RELEASE: ${CMAKE_C_FLAGS_RELEASE}")
16 message(STATUS "CMAKE_CXX_FLAGS: ${CMAKE_CXX_FLAGS}")
17 message(STATUS "CMAKE_CXX_FLAGS_RELEASE: ${CMAKE_CXX_FLAGS_RELEASE}")
18
19 #include_directories(/usr/local/include/mylib1 /usr/local/include/mylib2)
20 #link_directories(/usr/local/lib/mylib1 /usr/local/lib/mylib2)
21 #add_definitions(-DMYDEFINITION1 -DMYDEFINITION2)
22
23 # set the executable name, sources, libraries
24 add_executable(hello_math src/hello_math.cpp)
25 target_link_libraries(hello_math m)
26

```

コンパイル出力

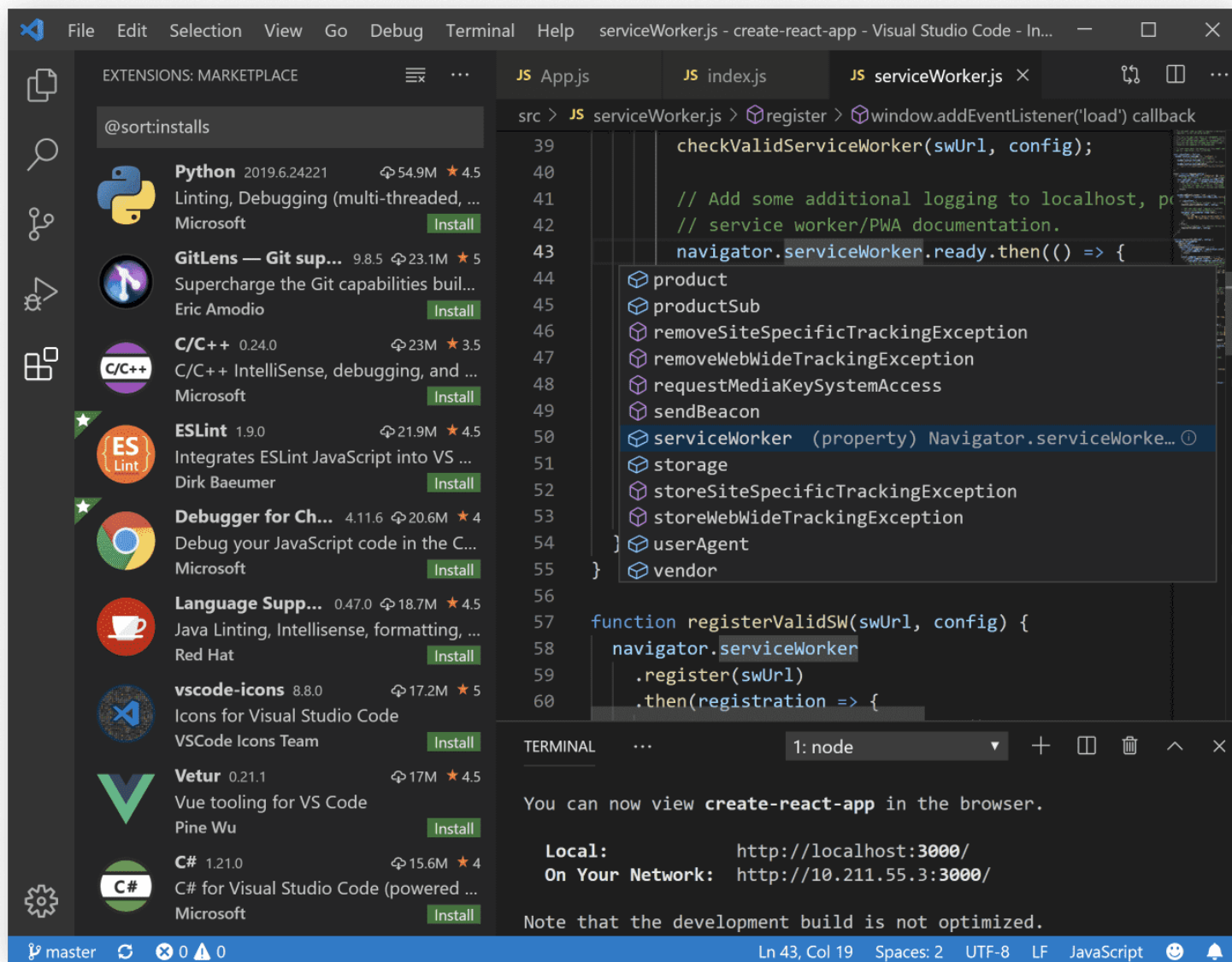
```

17:07:21: プロジェクト ymbc_tutorial のステップを実行中...
17:07:21: 起動中: "/usr/bin/make" clean
17:07:21: プロセス "/usr/bin/make" は正常に終了しました。
17:07:21: 起動中: "/usr/bin/make"
[100%] Building CXX object CMakeFiles/hello_math.dir/src/hello_math.cpp.o
Linking CXX executable ../bin/hello_math
[100%] Built target hello_math
17:07:22: プロセス "/usr/bin/make" は正常に終了しました。

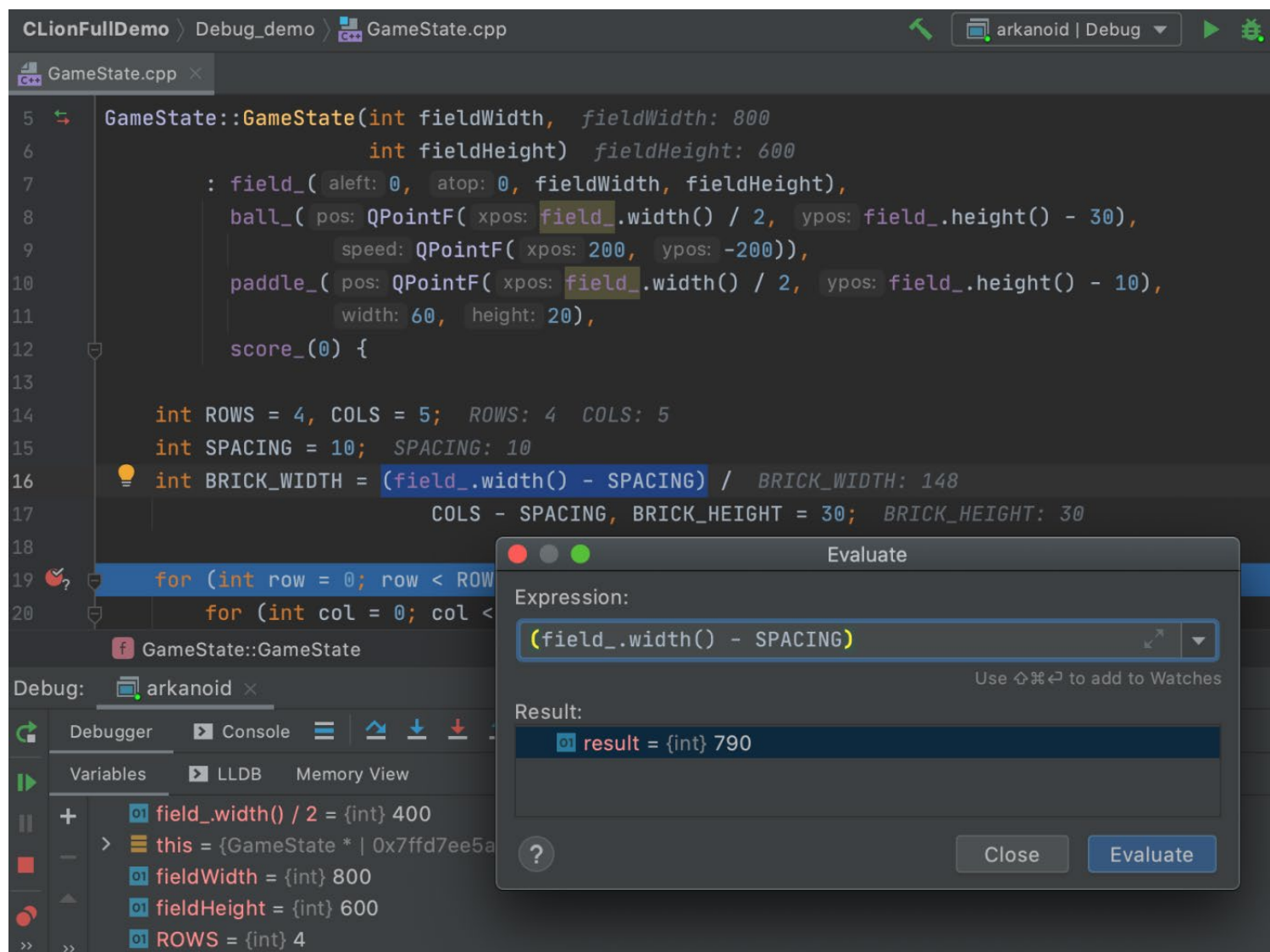
```

1 問題 2 検索結果 3 アプリケーション出力 4 コンパイル出力

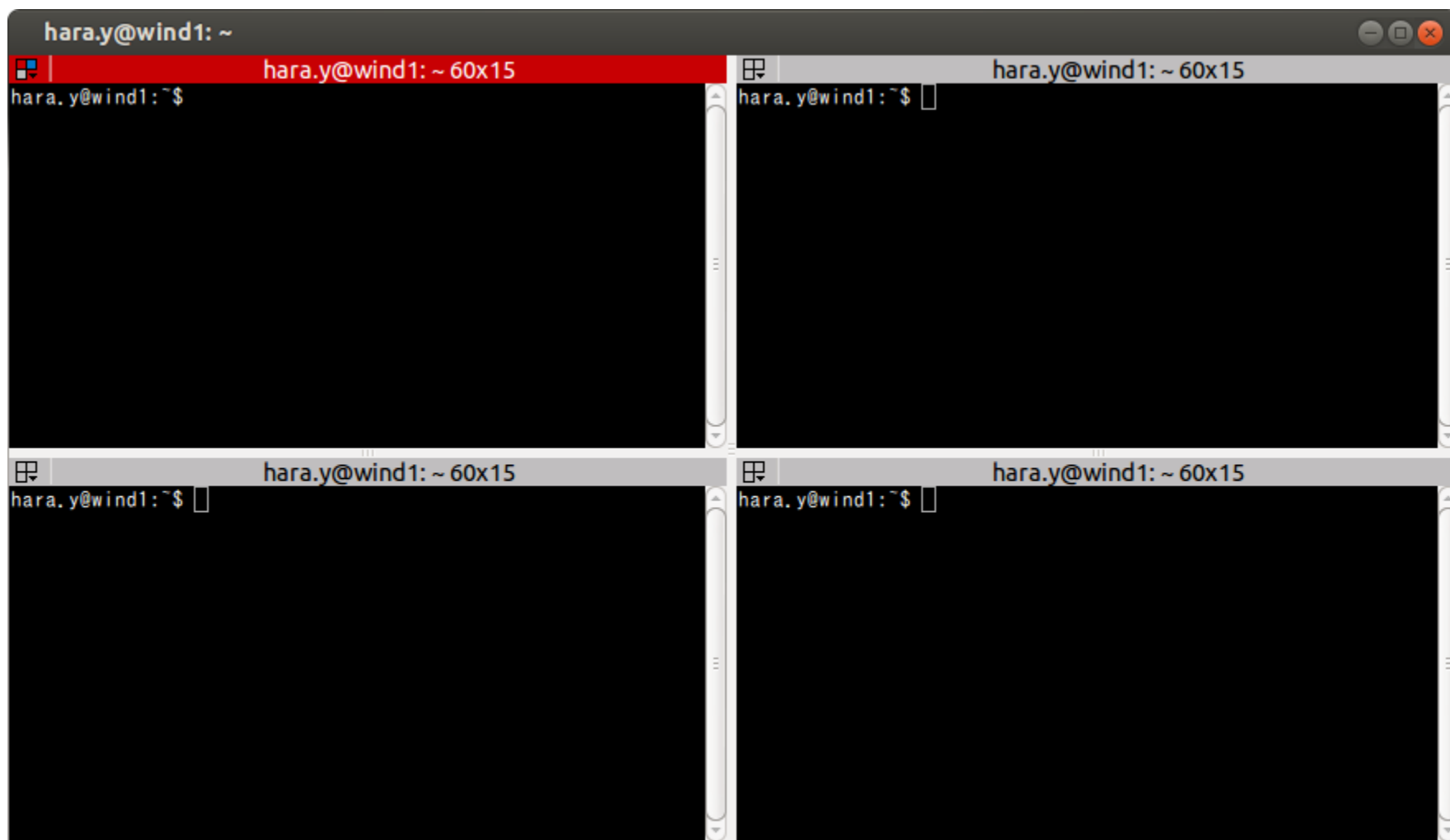
統合開発環境 Visual Studio Code



統合開発環境 CLion



画面分割できるターミナル Terminator



screen / tmux のような画面分割を簡単に利用可能

1. 自律走行を実現するための ROS パッケージ
2. 自己位置推定と SLAM の基本
3. 各パッケージのアルゴリズム
 - a. 自己位置推定 (amcl)
 - b. SLAM、地図生成 (gmapping)
 - c. 大域的経路計画、局所的動作計画 (move_base)
4. ROS に関する情報の調べ方
5. ROS での開発に関する知見
6. まとめ

まとめ

- **slam_gmapping, navigation の構成、アルゴリズム**
 - 自己位置推定 : Adaptive Monte Carlo Localization
 - SLAM、地図生成 : Rao-Blackwellized Particle Filter SLAM (FastSLAM 2.0 での Grid Mapping)
 - 大域的経路計画 : Navigation Function とダイクストラ法
 - 局所的動作計画 : Dynamic Window Approach
- **ROS に関する情報の調べ方 (ROS Index を活用)**
- **ROS での開発に関する知見**
 - catkin_make コマンド / catkin コマンド (新しいビルドシステム)
 - Subscriber へのコールバック関数の登録方法
 - Qt Creator : CMake を開ける統合開発環境
 - Terminator : 画面分割できるターミナル

参考文献

- **ROS Wiki**
<https://wiki.ros.org/>
- 友納 正裕: “移動ロボットのための確率的な自己位置推定と地図構築”, 日本ロボット学会誌, vol. 29, no. 5, pp. 423-426, 2011.
- 友納 正裕: “移動ロボットの自己位置推定と地図構築の基礎”, 第47回ロボット工学セミナー, 2008.
- Sebastian Thrun, Wolfram Burgard, and Dieter Fox: “**Probabilistic Robotics**”, The MIT Press, 2005.
(邦訳) 上田 隆一: “確率ロボティクス”, マイナビ, 2007.
- 坪内 孝司: “移動体の位置認識”, 計測自動制御学会 編, ビークル, コロナ社, 2003.
- Dieter Fox, Wolfram Burgard, and Sebastian Thrun: “**The Dynamic Window Approach to Collision Avoidance**”, IEEE Robotics & Automation Mag., vol. 4, no. 1, pp. 23-33, 1997.