

セミナー受講者限り：

**本資料の複製や二次利用、第三者への譲渡・開示等を
一切禁止します**



株式会社 日本テクノセンター

〒163-0722

東京都新宿区西新宿2-7-1

小田急第一生命ビル22F私書箱5043号

T E L (03)5322-5888 FAX(03)5322-5666



SLAMの基礎とROS による自律移動システム への応用

名城大学理工学部准教授 田崎豪

自己位置推定と地図作成 (SLAM):
Simultaneous Localization And Mapping

Tasaki Lab.



講師紹介

田崎 豪 (41)



Academic & Professional Experience

2004年から「ロボットビジョン」の研究に従事

Apr. 2000 - Mar. 2004 京都大学

Apr. 2004 - Mar. 2006 京都大学大学院情報学研究科

ヒューマンインターフェースロボットの研究

Apr. 2006 - Mar. 2018 株式会社東芝

移動ロボットの研究

2015年自動運転研究スタート

Apr. 2012 - Sep. 2013 京都大学大学院情報学研究科

博士 (情報学)

Apr. 2018 - today

名城大学

SLAMを使った
製品を発売

現在の研究テーマ

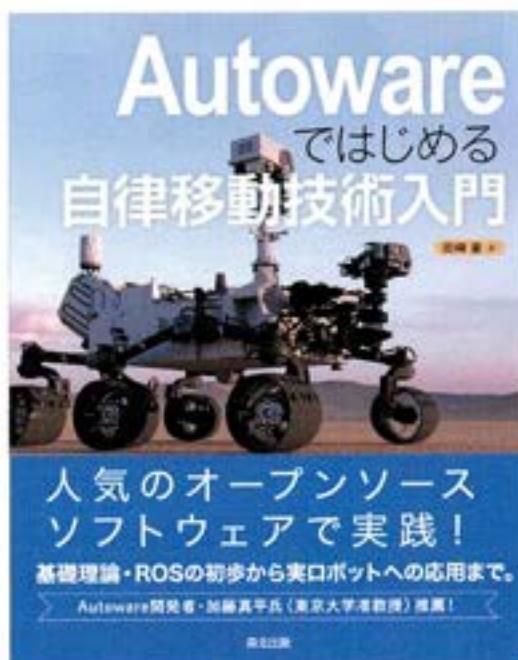
Tasaki Lab.



著書

4

Tasaki Lab.



自動運転技術入門
オーム社
(2021/4/7発売)

Autowareではじめる
自律移動技術入門
森北出版
(2021/7/1発売)



本日のスケジュール

Tasaki Lab.

| | | |
|----------|--------|-----------------------|
| 1 | 10:30- | ・自動運転/自律移動を実現するHW/SW |
| | 12:00 | ・LiDAR SLAMの原理 |
| 2 | 13:00- | ・障害物検出/経路生成/経路追従の原理 |
| | 13:50 | (Autowareで実装済みの手法の解説) |
| 3 | 14:00- | ・ROSの紹介 |
| | 14:50 | ・ROSのプログラミングデモ |
| 4 | 15:00- | ・Autowareの操作法の説明 |
| | 15:50 | ・Autowareによる自動運転デモ |
| 5 | 16:00- | ・AutowareによるSLAMデモ |
| | 17:00 | (・Autowareによる自律移動デモ) |

17:00以降は全体を通して質疑応答時間



本セミナーの目的と予備知識

セミナー
前半

■自動運転車や自律移動ロボットの動作原理の取得

- ・大学一年生程度の数学の知識（復習）

■移動ロボットを動作させるためのSWの実装方法の取得

- ・Linux/C++言語のプログラミング

セミナー
後半

SLAMの基礎とROS による自律移動システム への応用

1限目：SLAMの原理



1限目内容

8

Tasaki Lab.

■自動運転/自律移動のHWとSW

- SLAMの入出力を理解する

■SLAM原理の理解に必要な数学

- 最適化演算と座標変換に必要な数学を復習する

■LiDAR SLAMの仕組み



自動運転車と自律移動ロボット

動作原理は同じ

- 現在地から目的地まで障害物にぶつからないように移動する
- 違いは交通ルールくらい

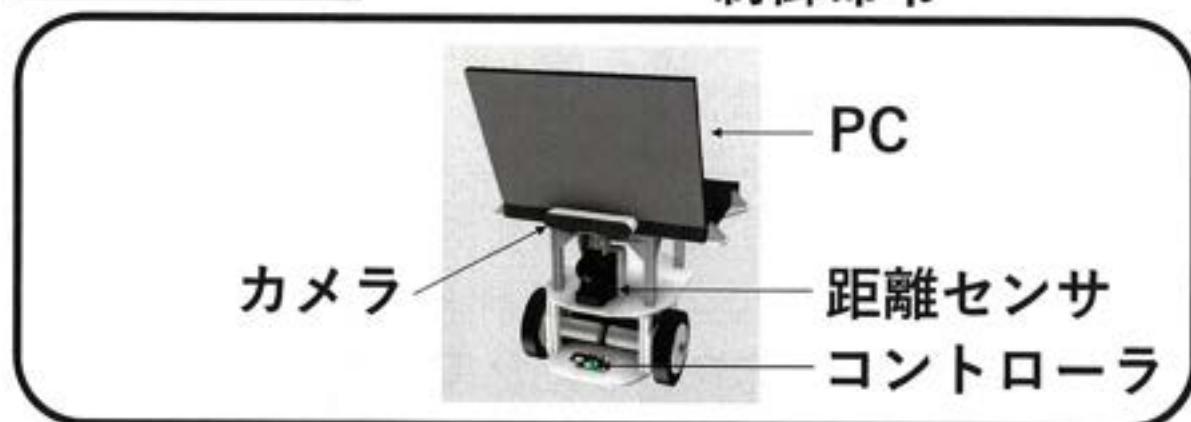
ハードウェアもソフトウェアも
ほぼ同じ構成にできる

以降自動運転車も含めて
移動ロボットと記載



移動ロボットに使用するHW

三次元点群



オドメトリ:
車輪回転量から計算できる大まかな自己位置

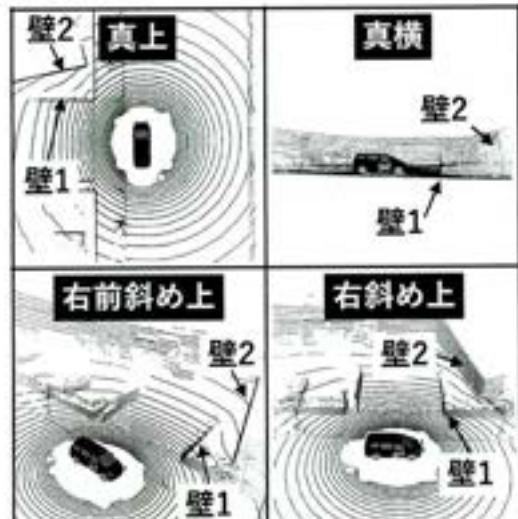
距離センサ

■SLAMの要となるセンサ

■センサ周辺の三次元空間情報を点群の座標として取得可能

3次元LiDARの例

LiDAR

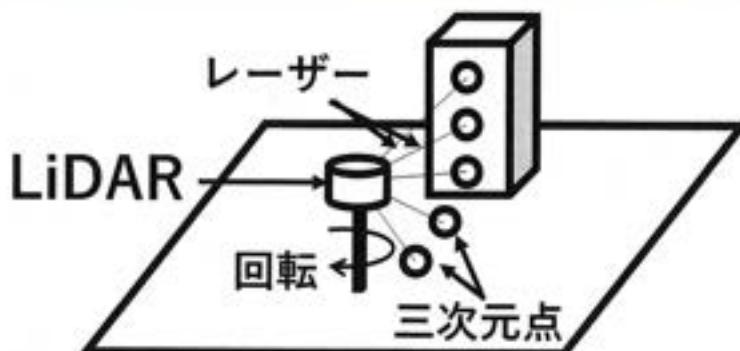


代表的な距離センサ

| センサ名 | 原理 |
|------------------------|--|
| 超音波 | 1.スピーカーで超音波発射 2.マイク収音までの時間で距離計測 |
| LiDAR (2次元/ 3次元) | 1.レーザー発射 2.レーザー反射光受光までの時間で 距離計測 |
| RGBDカメラ | 1.特殊なパターンの赤外光を発光 2.カメラ画像上のパターンの位置 変化から距離計測 |
| ステレオカメラ | 1.二つのカメラで画像取得 2.各カメラの物体の見え方の違いか ら距離計測 |



回転式LiDAR (Velodyne製)

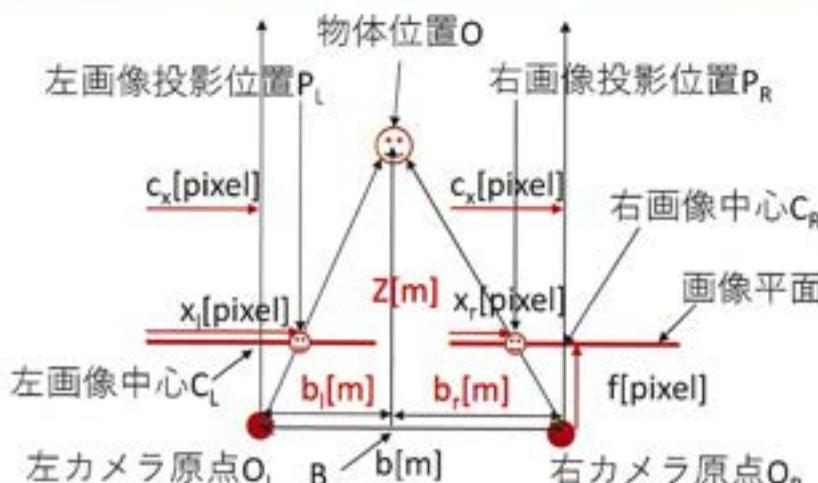


- ・高精度な距離計測 (100mまで±3cmの誤差)
- ・黒色物体/ガラス等は検出不可能

2次元LiDAR: レーザーを一定の高さだけに照射
 3次元LiDAR: レーザーを照射する高さが変化



ステレオカメラ



$$\begin{aligned} \triangle O_L P_L C_L &\sim \triangle O O_L B \\ \rightarrow b_i &= Z (x_i - c_x) / f \\ \triangle O_R P_R C_R &\sim \triangle O O_R B \\ \rightarrow b_r &= Z (c_x - x_r) / f \\ \downarrow \\ b &= b_i + b_r = Z (x_i - x_r) / f \\ \rightarrow Z &= bf / (x_i - x_r) \end{aligned}$$



取得できる点群の違い

LiDAR:
高価/高精度

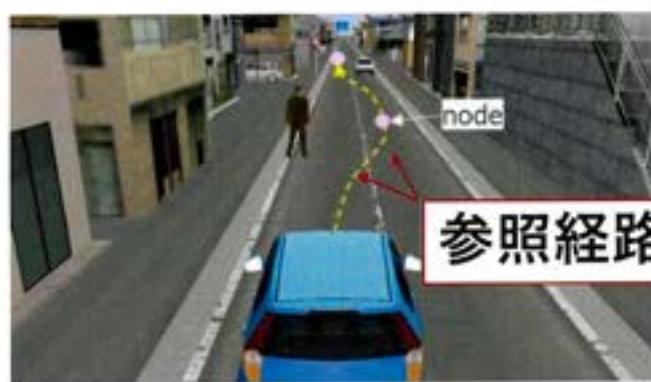
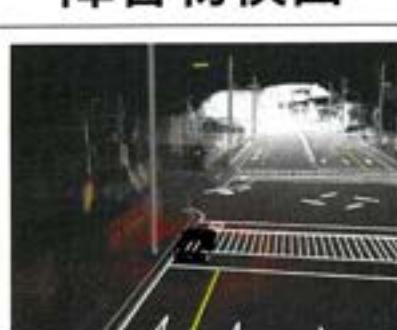
ステレオカメラ:
安価/低精度



精度が高い点群が多く取得できる
≒SLAMの精度高くなる

三次元LiDAR > 二次元LiDAR > ステレオカメラ

移動ロボットを構成するSW



Autowareの動作例

Tasaki Lab.



HW/SWのまとめ

Tasaki Lab.

■移動ロボットの入力には距離センサを用いる

- 距離センサは精度の良いLiDARを使うとよい

■自動運転のSWは自己位置推定、障害物検出、経路生成、経路追従で構成される



SLAMの概要と 必要な数学知識の復習

三次元地図作成の流れ

20

Tasaki Lab.

1. 自律移動したい空間を走行

- ・走行時にオドメトリ*と距離センサデータを同期取得

2. 走行時データからSLAMで三次元地図を作成

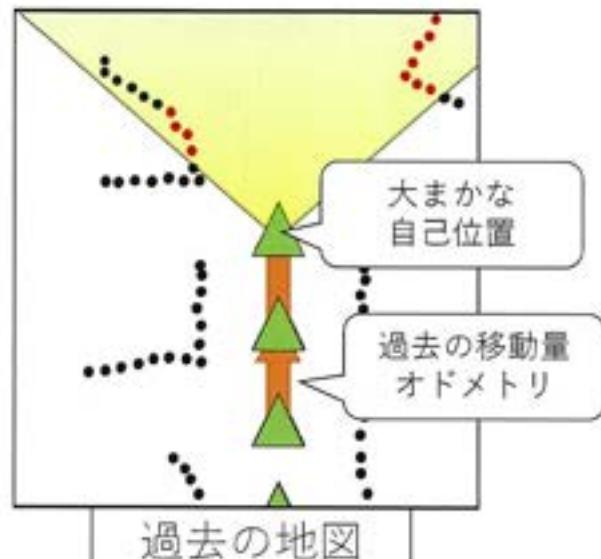
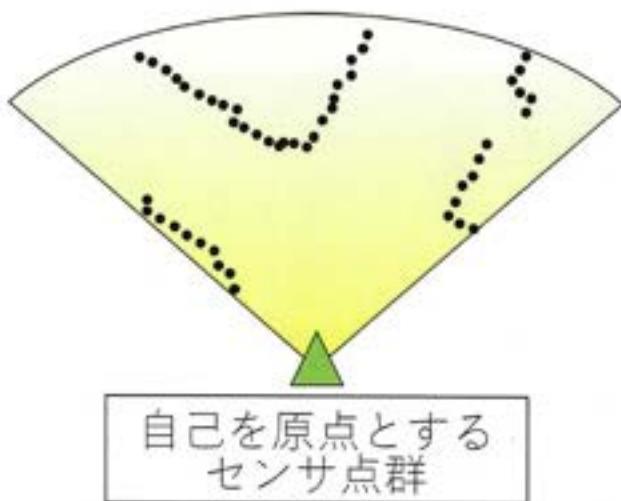
*オドメトリ：

車輪回転量から計算できる大まかな自己位置
スリップなどの影響で誤差が徐々に大きくなる

オドメトリが取得できない場合は過去
の移動量から大まかな自己位置を得る

SLAMの概要

時系列で相対的に自己位置を計算しつつ
三次元地図を作成



現在のセンサ点群と過去の地図点群を照合
→正確な位置を取得&地図拡大

SLAMで行う計算

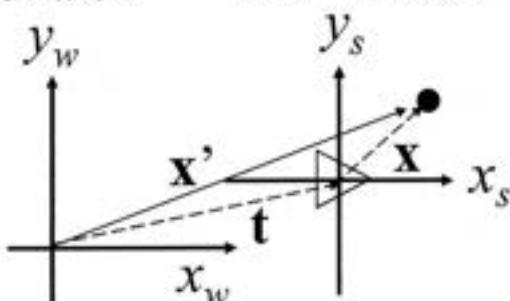
- ロボット座標系の点群を地図の世界座標系に変換する座標変換
- 地図点群座標とセンサ点群座標の誤差を示す関数の最小化
(ニュートン法)



座標変換（並進移動）

ロボットが世界に対して傾いていない場合

世界座標系 W ロボット座標系 S



x : ロボットから見た計測点の座標

t : 世界座標から見たロボットの位置

世界座標から見た計測点の座標

$$x' = x + t$$

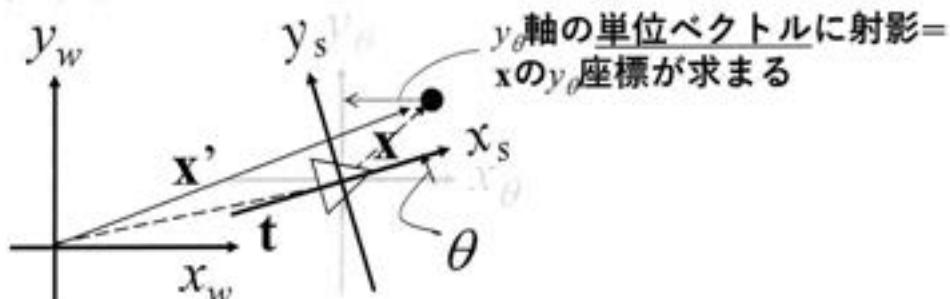


座標変換（回転移動）1/3

ロボットが世界に対して θ 傾いた場合

世界座標系 W

回転補正座標系 θ



回転していない座標系に変換（射影）
してから並進移動させる

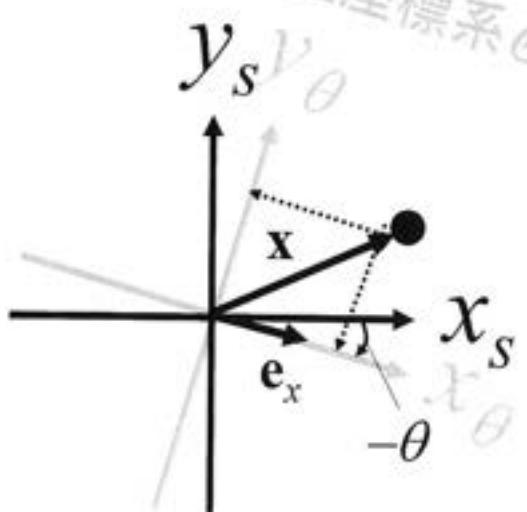


座標変換（回転移動）2/3

Tasaki Lab.

e_x : ロボット座標系からみた
回転補正座標系の x 軸
単位ベクトル

$$\begin{aligned} e_x &= (\cos(-\theta) \quad \sin(-\theta))^T \\ &= (\cos \theta \quad -\sin \theta)^T \end{aligned}$$



回転補正座標系から
見た計測点の x 座標 $x^{(\theta)}$

$$x^{(\theta)} = e_x \cdot x = (\cos \theta \quad -\sin \theta)x$$

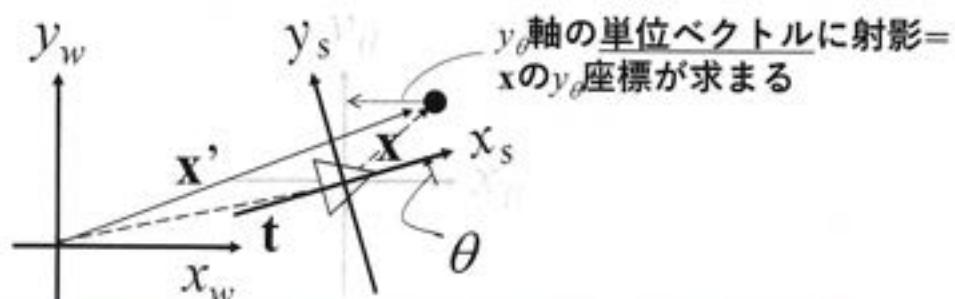
回転補正座標系から
見た計測点の y 座標 $y^{(\theta)}$

$$y^{(\theta)} = (\sin \theta \quad \cos \theta)x$$

座標変換（回転移動）3/3

Tasaki Lab.

ロボットが世界に対して θ 傾いた場合

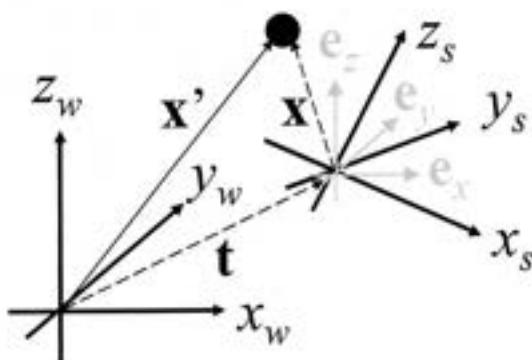
世界座標系 W 回転補正座標系 S 

世界座標から見た計測点の座標

$$x' = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} x + t = \begin{pmatrix} e_x^T \\ e_y^T \end{pmatrix} x + t = Rx + t$$

回転を示す行列Rと位置を示すベクトルtを
求めるのが自己位置推定

三次元座標変換



考え方は二次元と同じ

- ・回転補正座標系の軸の単位ベクトルを求めて回転補正

- ・位置ベクトルtを加算 $x' = \begin{pmatrix} e_x^T \\ e_y^T \\ e_z^T \end{pmatrix} x + t = Rx + t$

座標変換式

三次元座標変換(例題)

前提：ロボットが世界座標で(2,1,1)の位置で左に90度回転している

例題：ロボットが計測した点(0.5,1,1)の世界座標は？

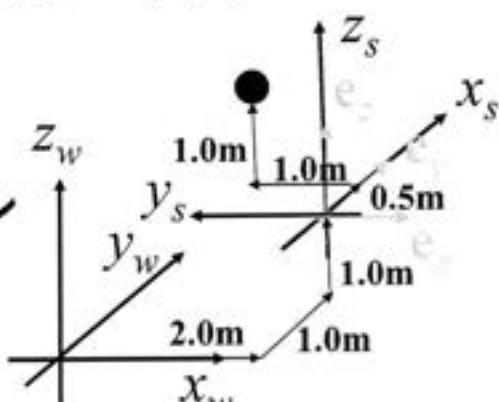
解答

回転補正座標系単位ベクトル

$$e_x = (0 \ -1 \ 0)^T$$

$$e_y = (1 \ 0 \ 0)^T$$

$$e_z = (0 \ 0 \ 1)^T$$



$$\text{座標変換式 } x' = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0.5 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1.5 \\ 2 \end{pmatrix}$$



関数の最小化(ベクトルの偏微分)

自己位置推定

- 真の自己位置t、姿勢Rに近づくほど小さくなる関数を定義する

- 関数を最小にするtとRを求める

関数の最小化→偏微分=0の解を求める

スカラー関数fをn次元ベクトルxで偏微分

$$\frac{\partial f}{\partial \mathbf{x}} = \left(\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \quad \dots \quad \frac{\partial f}{\partial x_n} \right)^T$$

n次元ベクトルxをスカラーuで偏微分

$$\frac{\partial \mathbf{x}}{\partial u} = \left(\frac{\partial x_1}{\partial u} \quad \frac{\partial x_2}{\partial u} \quad \dots \quad \frac{\partial x_n}{\partial u} \right)^T$$



関数の最小化(ニュートン法)

関数の最小化→偏微分=0の解を求める

難しい方程式は直接解けない

大まかな解(初期値)から正確な解へ近づける

ニュートン法

大まかな解 x_k を更新して正確な解へ近づける

更新式

$$\begin{cases} \mathbf{x}_{k+1} = \mathbf{x}_k + \Delta \mathbf{x}_k \\ \Delta \mathbf{x}_k = -H_k^{-1} J_k \end{cases}$$

$$J_k = \left. \frac{\partial f}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_k} \quad H_k = \left. \left(\frac{\partial J_k}{\partial x_1} \quad \dots \quad \frac{\partial J_k}{\partial x_n} \right) \right|_{\mathbf{x}=\mathbf{x}_k}$$

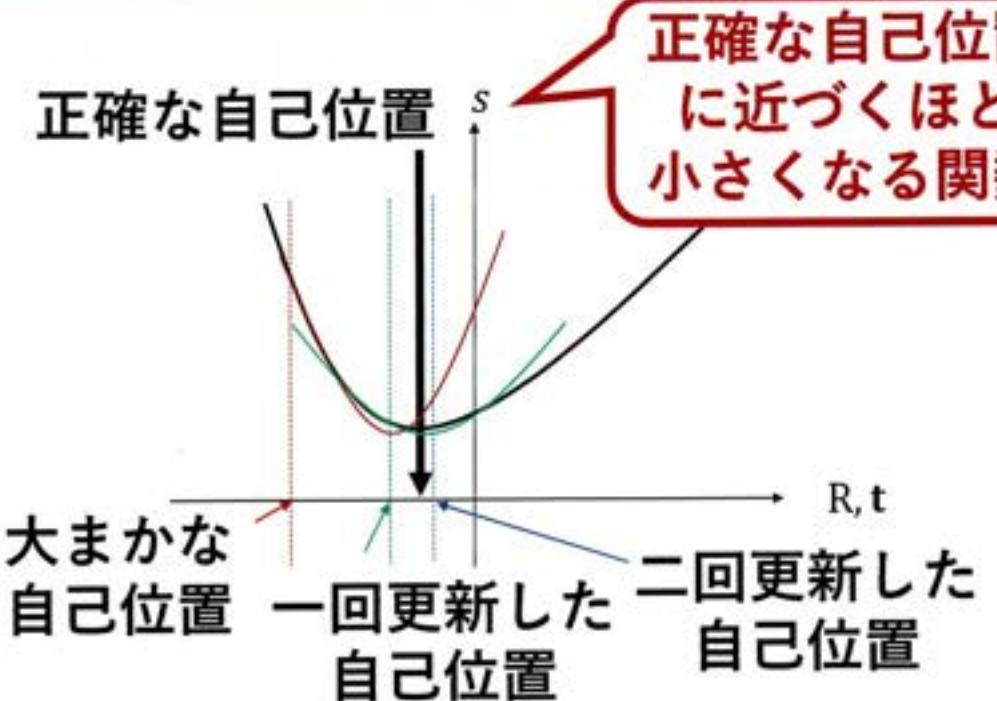
勾配ベクトル

ヘッセ行列



ニュートン法応用イメージ

Tasaki Lab.



更新量が閾値以下になるまで更新する



例題

Tasaki Lab.

関数 $f(\mathbf{x}) = x_1^4 + x_2^4$ を初期値 $\mathbf{x}_0 = (2, 2)$ から
ニュートン法で $f(\mathbf{x}) = 0$ の解に近づけよ

$$\mathbf{J}_0 = \left(\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \right)^T \Bigg|_{\mathbf{x}=\mathbf{x}_0} = (4x_1^3 \quad 4x_2^3)^T \Bigg|_{\mathbf{x}=\mathbf{x}_0} = \begin{pmatrix} 32 \\ 32 \end{pmatrix}$$

$$\mathbf{H}_0 = \begin{pmatrix} 12x_1^2 & 0 \\ 0 & 12x_2^2 \end{pmatrix} \Bigg|_{\mathbf{x}=\mathbf{x}_0} = \begin{pmatrix} 48 & 0 \\ 0 & 48 \end{pmatrix}$$

$$\Delta \mathbf{x}_0 = -\mathbf{H}_0^{-1} \mathbf{J}_0 = -\begin{pmatrix} 1/48 & 0 \\ 0 & 1/48 \end{pmatrix} \begin{pmatrix} 32 \\ 32 \end{pmatrix} = -\begin{pmatrix} 2/3 \\ 2/3 \end{pmatrix}$$

$$\mathbf{x}_1 = \mathbf{x}_0 + \Delta \mathbf{x}_0 = \begin{pmatrix} 4/3 \\ 4/3 \end{pmatrix} \rightarrow f(\mathbf{x}_0) = 32 > f(\mathbf{x}_1) = 512/81$$

となり最小解へ近づく



■SLAM概要

- SLAMとは三次元点群を時系列で重ね合わせて地図を作る処理

■SLAMに必要な数学

- 正確な自己位置姿勢ほど値が小さくなる関数を定義する
- 関数を最小にする自己位置姿勢は直接求められず大まかな解を更新して求める



LiDAR SLAMの仕組み

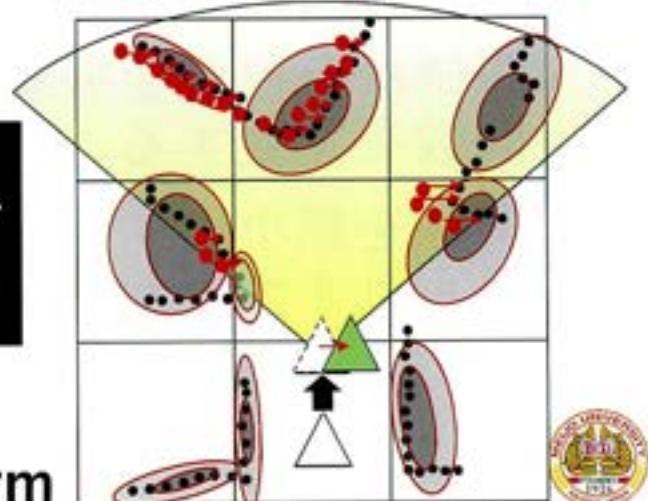
NDTマッチング

- 地図を均等な箱に分割
- 箱内の地図点群の正規分布の平均にセンサ点群の平均が近づくようにマッチング

特徴

地図を点で管理せず
正規分布で管理

NDT: Normal
Distributions Transform

**正規分布を使う利点**

- 点そのもので管理することに比べて省メモリになる
- 地図の点が増えても処理速度があまり落ちない
- 2階微分できる

- ニュートン法のような高速な最適化理論が使える

正規分布 $f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^3 \det V}} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T V^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)$

共分散行列
平均位置



NDTマッチングの計算式

以下のスコア S を最小にする R と t を求める

$$S = - \sum_{m=1}^N \exp \left(-\frac{1}{2} (\mathbf{x}'_m - \boldsymbol{\mu})^T V^{-1} (\mathbf{x}'_m - \boldsymbol{\mu}) \right)$$

q_mとおく
点が平均に近いほど小さくなる

$$\mathbf{x}'_m = R\mathbf{x}_m + \mathbf{t}$$

自己位置姿勢をRとtとしたとき
の点の位置を求めている

$\boldsymbol{\mu}$: ある箱内の正規分布の平均

V : ある箱内の正規分布の共分散行列

\mathbf{t} : 自己位置を示す並進ベクトル

R : 自己姿勢を示す回転行列

\mathbf{x}_m : 点 m の座標



スコア最小化(ニュートン法)

求めたい位置姿勢をベクトル p とおく

平面を移動する場合の位置姿勢:

xy座標と姿勢 θ で三次元 $\rightarrow p = (p_1, p_2, p_3)^T$

k 回目更新時

$$\text{勾配ベクトル } J_{ki} = \sum_{m=1}^N \exp \left(-\frac{1}{2} \mathbf{q}_m^T V^{-1} \mathbf{q}_m \right) \mathbf{q}_m^T V^{-1} \frac{\partial \mathbf{q}_m}{\partial p_i}$$

ヘッセ行列

$$H_{kij} = \sum_{m=1}^N \left\{ -\exp \left(-\frac{1}{2} \mathbf{q}_m^T V^{-1} \mathbf{q}_m \right) \mathbf{q}_m^T V^{-1} \frac{\partial \mathbf{q}_m}{\partial p_j} \mathbf{q}_m^T V^{-1} \frac{\partial \mathbf{q}_m}{\partial p_i} \right. \\ \left. + \exp \left(-\frac{1}{2} \mathbf{q}_m^T V^{-1} \mathbf{q}_m \right) \frac{\partial \mathbf{q}_m^T}{\partial p_j} V^{-1} \frac{\partial \mathbf{q}_m}{\partial p_i} + \exp \left(-\frac{1}{2} \mathbf{q}_m^T V^{-1} \mathbf{q}_m \right) \mathbf{q}_m^T V^{-1} \frac{\partial^2 \mathbf{q}_m}{\partial p_i \partial p_j} \right\}$$

偏微分は消せるか???



勾配ベクトルとヘッセ行列の偏微分

Tasaki Lab.

平面移動の場合

$$\begin{aligned}\mathbf{q}_m &= \begin{pmatrix} \cos p_3 & -\sin p_3 \\ \sin p_3 & \cos p_3 \end{pmatrix} \mathbf{x}_m + \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} - \boldsymbol{\mu} \\ &= \begin{pmatrix} x_{m1} \cos p_3 - x_{m2} \sin p_3 + p_1 - \mu_1 \\ x_{m1} \sin p_3 + x_{m2} \cos p_3 + p_2 - \mu_2 \end{pmatrix}\end{aligned}$$

勾配ベクトル用

$$\frac{\partial \mathbf{q}_m}{\partial p_1} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \frac{\partial \mathbf{q}_m}{\partial p_2} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad \frac{\partial \mathbf{q}_m}{\partial p_3} = \begin{pmatrix} -x_{m1} \sin p_3 - x_{m2} \cos p_3 \\ x_{m1} \cos p_3 - x_{m2} \sin p_3 \end{pmatrix}$$

ヘッセ行列用

$$\begin{aligned}\frac{\partial^2 \mathbf{q}_m}{\partial p_1 \partial p_j} &= \frac{\partial}{\partial p_j} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \frac{\partial^2 \mathbf{q}_m}{\partial p_2 \partial p_j} = \frac{\partial}{\partial p_j} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\ \frac{\partial^2 \mathbf{q}_m}{\partial p_3 \partial p_j} &= \begin{cases} \begin{pmatrix} 0 \\ 0 \end{pmatrix} & j = 1, 2 \\ \begin{pmatrix} -x_{m1} \cos p_3 + x_{m2} \sin p_3 \\ -x_{m1} \sin p_3 - x_{m2} \cos p_3 \end{pmatrix} & j = 3 \end{cases}\end{aligned}$$

NDTの弱点（大まかな自己位置）

Tasaki Lab.

大まかな自己位置が大きくずれていると
最適値にたどり着けない



NDTの弱点 (単純な地図)

Tasaki Lab.

廊下のような
まっすぐな道の
地図の点

自己位置推定

得られた点

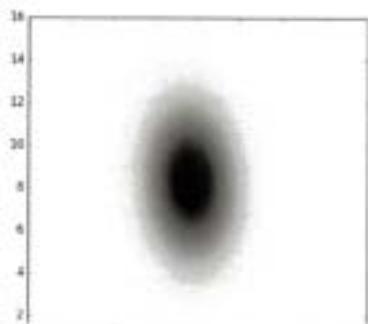
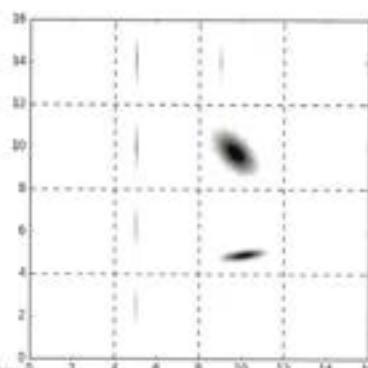
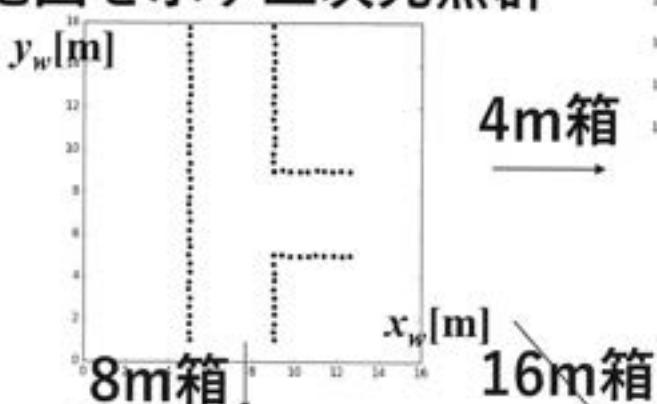
自己位置姿勢が
一つに定まらない



NDTの弱点 (箱のサイズ大)

Tasaki Lab.

地図を示す三次元点群

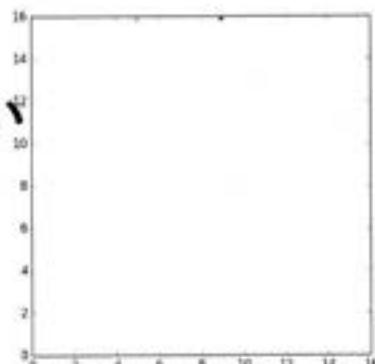
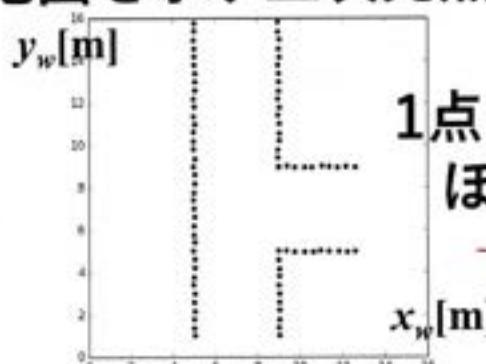


箱のサイズを大きくすると形状を表せない



NDTの弱点（箱のサイズ小）

地図を示す三次元点群



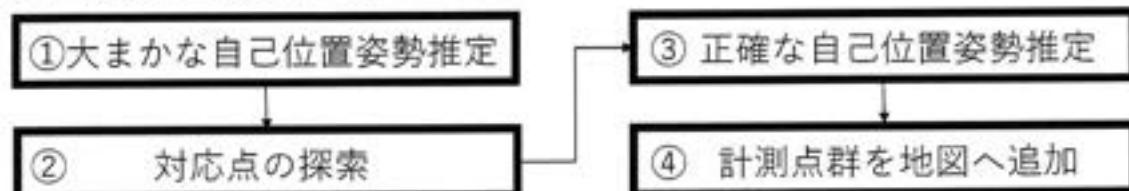
分布が取れない
→ニュートン法が使えない



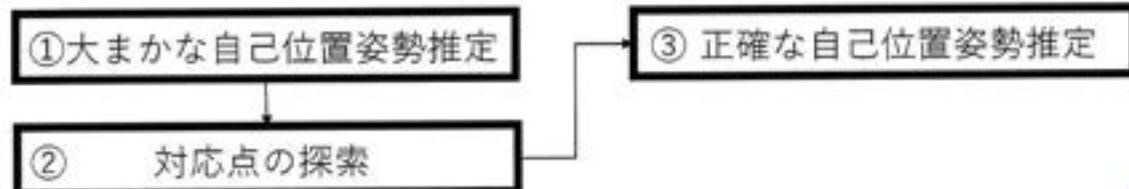
自己位置推定

SLAMの地図作成処理を除去する

SLAMの処理フロー



自己位置推定の処理フロー



LiDAR SLAMの仕組み

LOAM

SLAMのベンチマークテスト

46

Tasaki Lab.

■SLAMの評価

位置が不正確だと
地図はできない

- 地図作成時の自己位置推定の性能を用いる

KITTIデータセット：100ms周期の
ステレオ画像、LiDAR点群、自己位置を提供

2022年11月のランキング

| 順位 | 手法名 | センサ | 並進誤差 |
|----|--------|-----------|-------|
| 1 | SOFT2 | カメラ | 0.53% |
| 2 | V-LOAM | カメラ+LiDAR | 0.54% |
| 3 | LOAM | LiDAR | 0.55% |

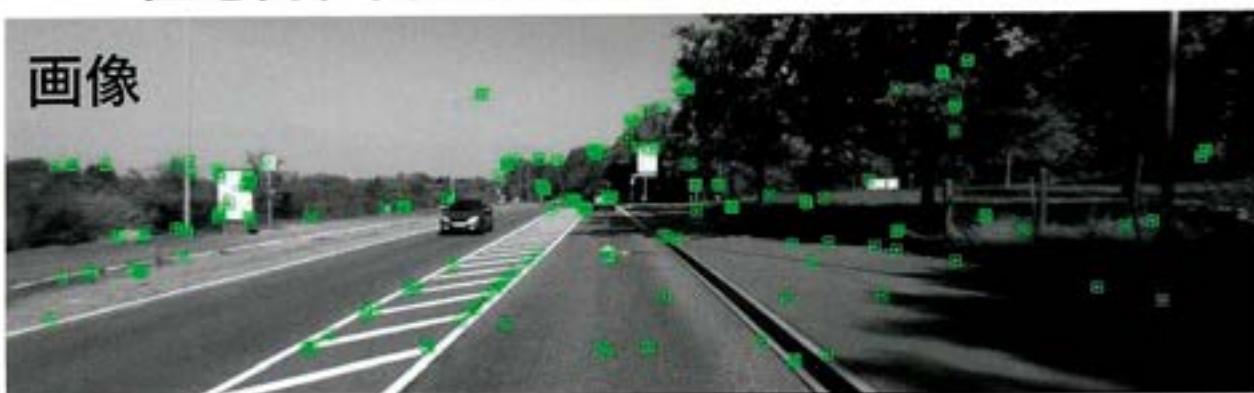


KITTIオドメトリデータセット

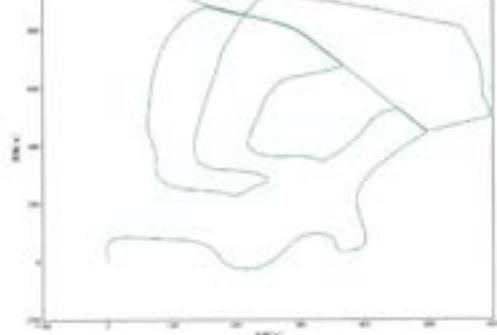
Tasaki Lab.

住宅街、高速道路など11シーン提供

画像



http://www.cvlibs.net/datasets/kitti/eval_odometry.php



走行軌跡
(上空から見た車両の軌跡)



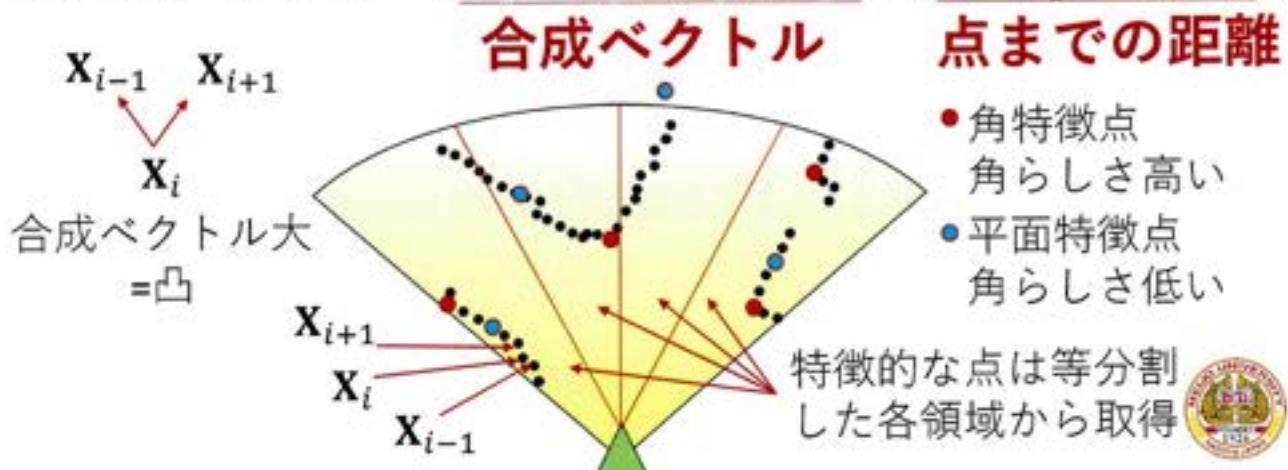
LOAMの動作原理

Tasaki Lab.

■特徴的な点だけを用いて照合

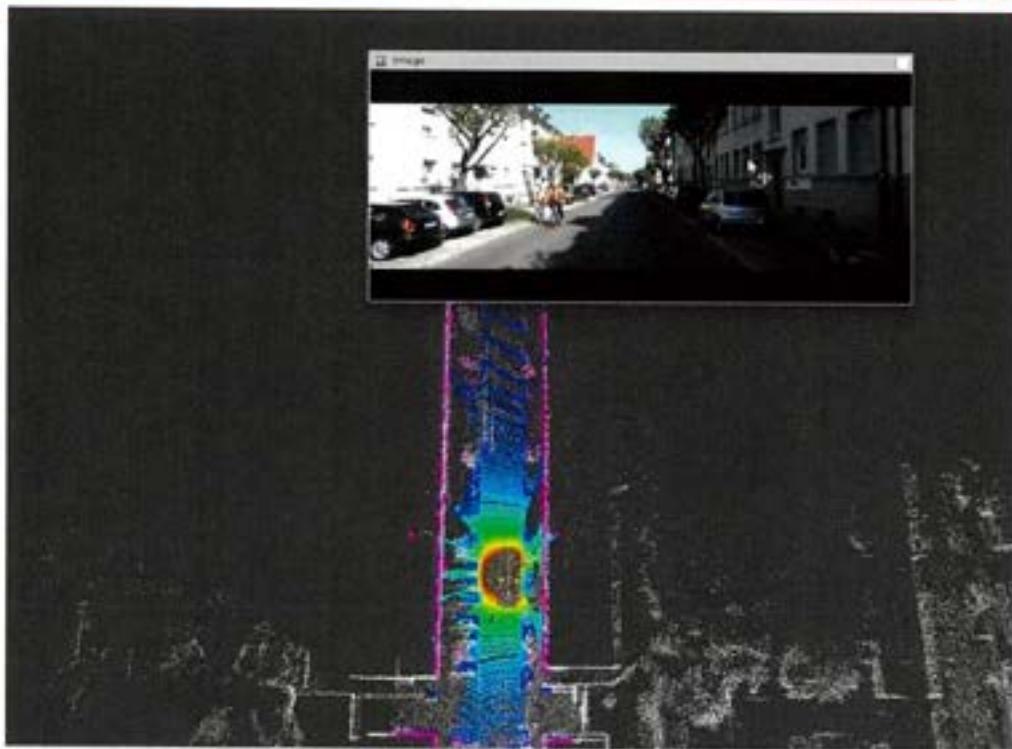
LiDAR視野を等分割した領域で
角らしさが高い点と低い点を使用

$$\text{点 } i \text{ の角らしさ} = ||\mathbf{X}_{i+1} - \mathbf{X}_i + \mathbf{X}_{i-1} - \mathbf{X}_i|| / \left\{ \left(\frac{||\mathbf{X}_{i+1} + \mathbf{X}_{i-1}||}{2} \right) ||\mathbf{X}_i|| \right\}$$



LOAM動作例

Tasaki Lab.



ROSで動作して精度も高いが計算量が多い



LiDAR SLAMまとめ

Tasaki Lab.

- LiDAR SLAMは地図と計測点のマッチング方法で精度や処理速度が変わる
- NDTは地図を箱で区切って箱内の正規分布と計測点群の分布をマッチングする



SLAMの基礎とROS による自律移動システム への応用

2限目：障害物検出/経路生成/経路追従の原理



2限目内容

2

Tasaki Lab.

■障害物検出

- ・障害物を管理する地図（コストマップ）の作成

■経路生成

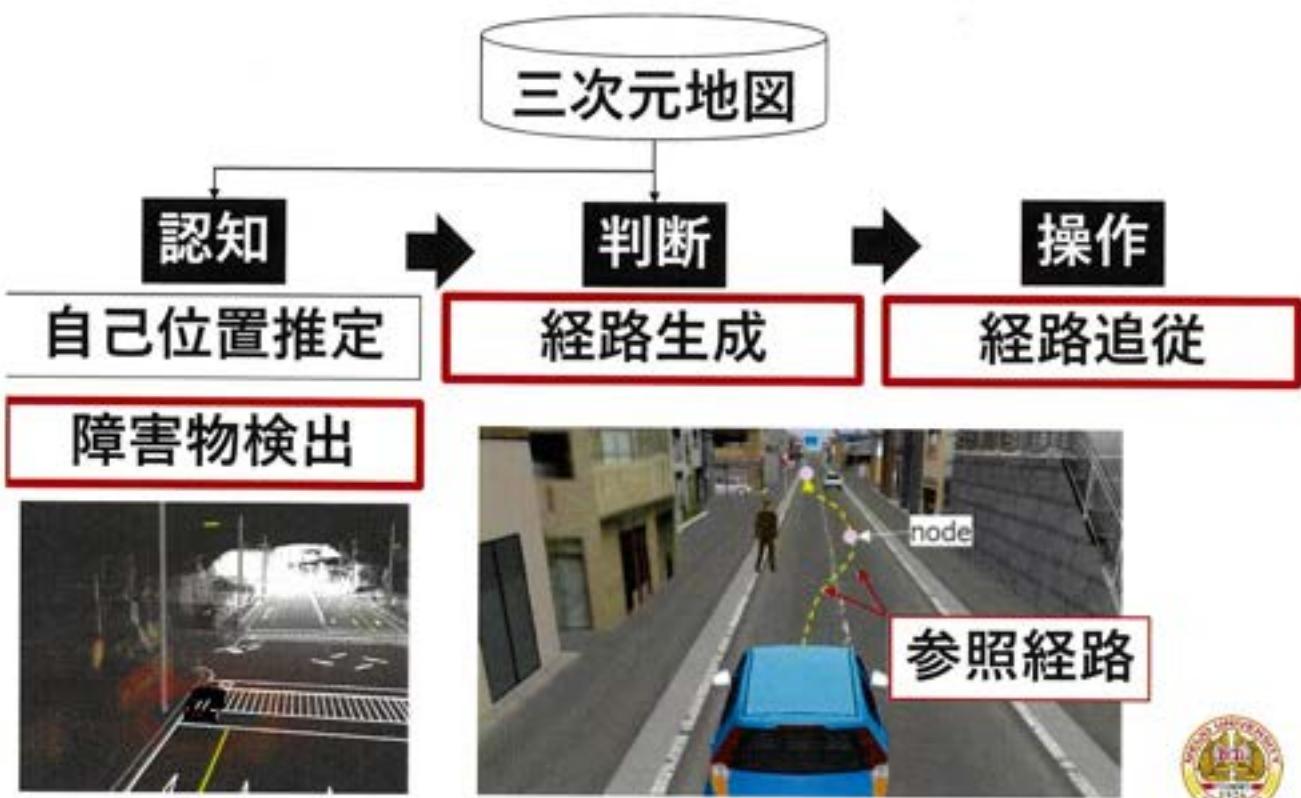
- ・コストマップ上での障害物回避経路の探索法

■経路追従

- ・車輪型移動ロボットの制御法

Autowareで使用している手法を紹介





障害物検出

障害物の検出とコストマップ

障害物検出の流れ

Tasaki Lab.

入力: LiDAR

(センサ誤差が小さいため障害物検出容易)

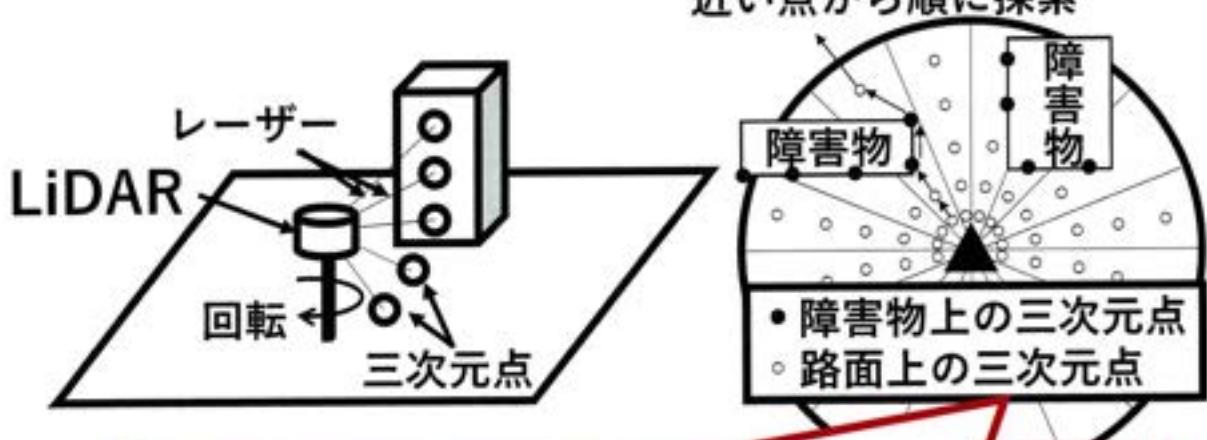


分割領域での障害物検出

Tasaki Lab.

ロボット周辺の上面図

近い点から順に探索



1. ロボットを中心とした円を等分割
2. 分割領域ごとに障害物点を検出

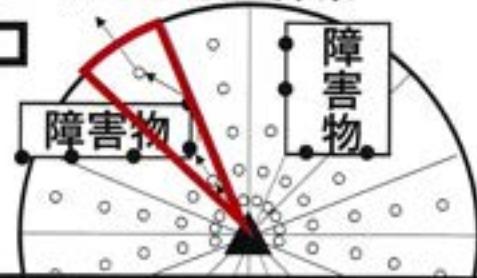


障害物検出概要

一定仰角以上の高さにある点を障害物点とする

1領域を横から見る

近い点から順に探索



高さ

直前点から見た
高さ閾値

直前点から見た
角度閾値

車輪

距離

障害物判定
ライン

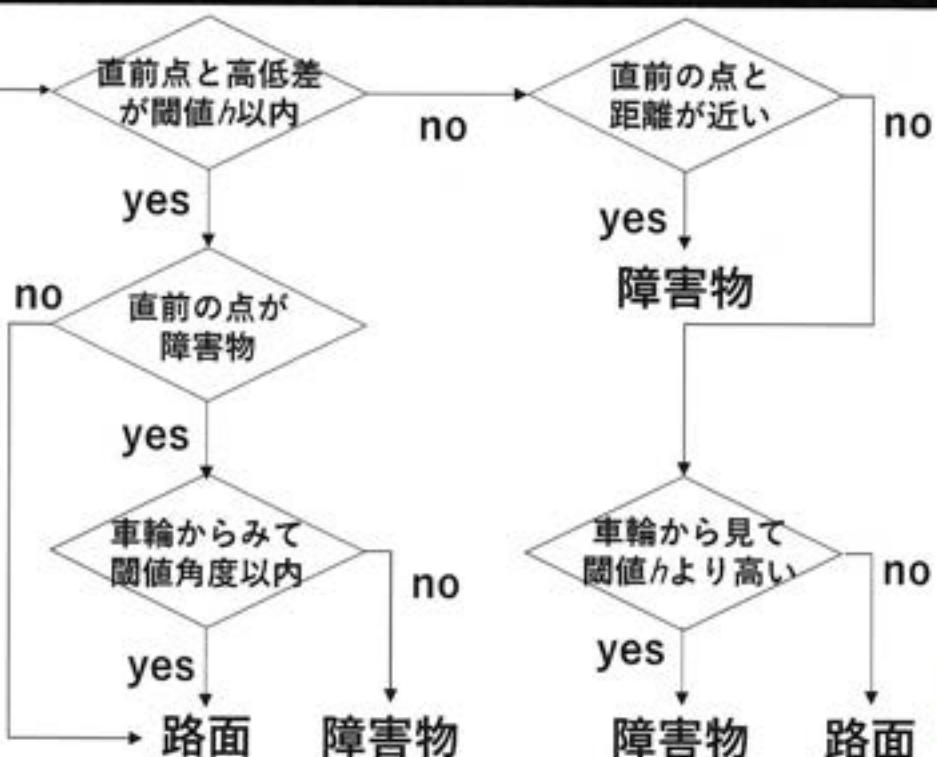
仰角

詳細な障害物判定フロー

*1点目は直前点が障害物で高さ0mとして判定

ノイズ除去のため直前の判定結果も利用

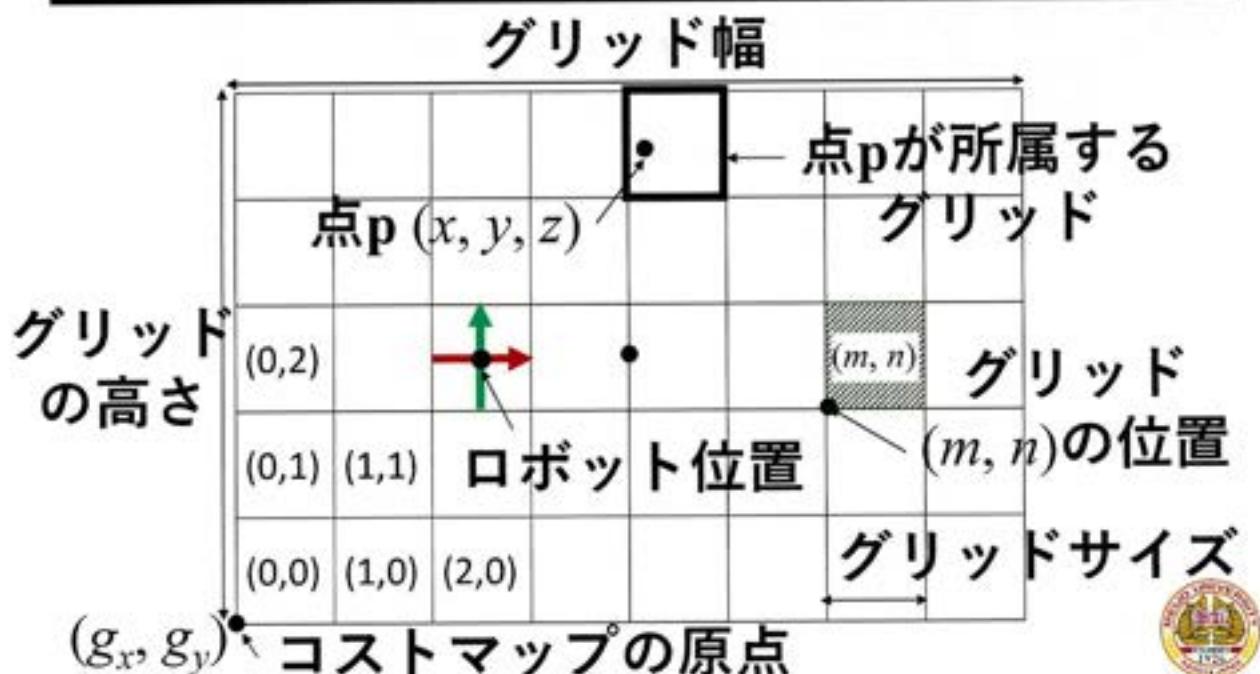
判定開始



コストマップでの障害物管理

Tasaki Lab.

グリッドを使用することで探索空間を離散化
→経路探索しやすくなる



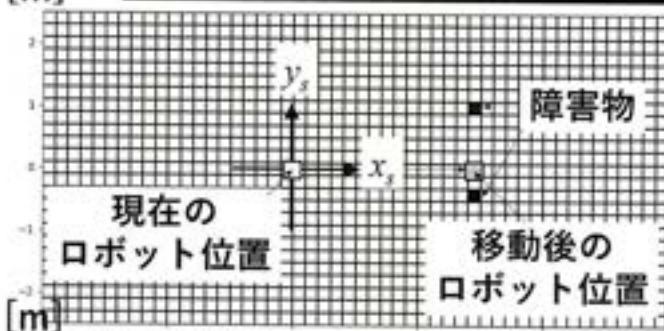
Autowareのコストマップ

Tasaki Lab.



グリッドサイズはロボットサイズ等を考慮して適切に設定する

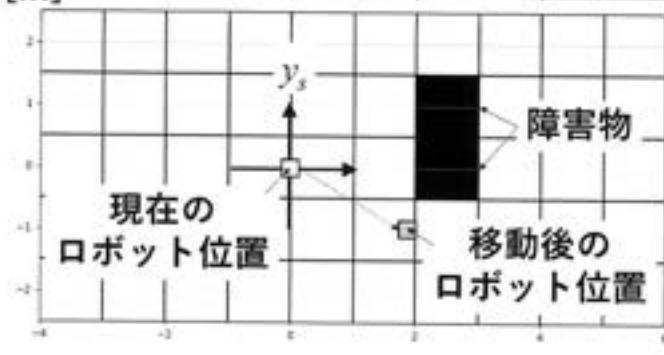
[m]



グリッドサイズ小

- ・障害物を適切に回避
- ・不安定な経路が出る可能性がある

[m]



グリッドサイズ大

- ・経路が生成されない可能性がある



障害物検出まとめ

- LiDARを使用すると三次元点群の高さベースの単純な手法で障害物点を抽出できる
- 障害物は点ではなく格子状に区切られたコストマップで管理する



経路生成

Hybrid A* 探索

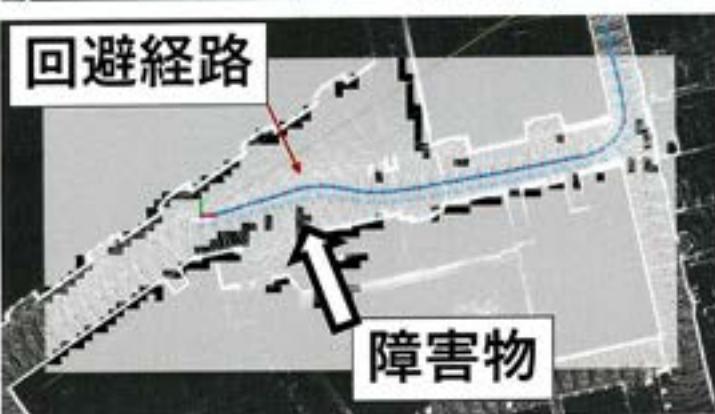
Autowareの経路生成例

14

Tasaki Lab.



障害物がない場合
・ 参照経路を出力



障害物がある場合
・ 回避経路を出力



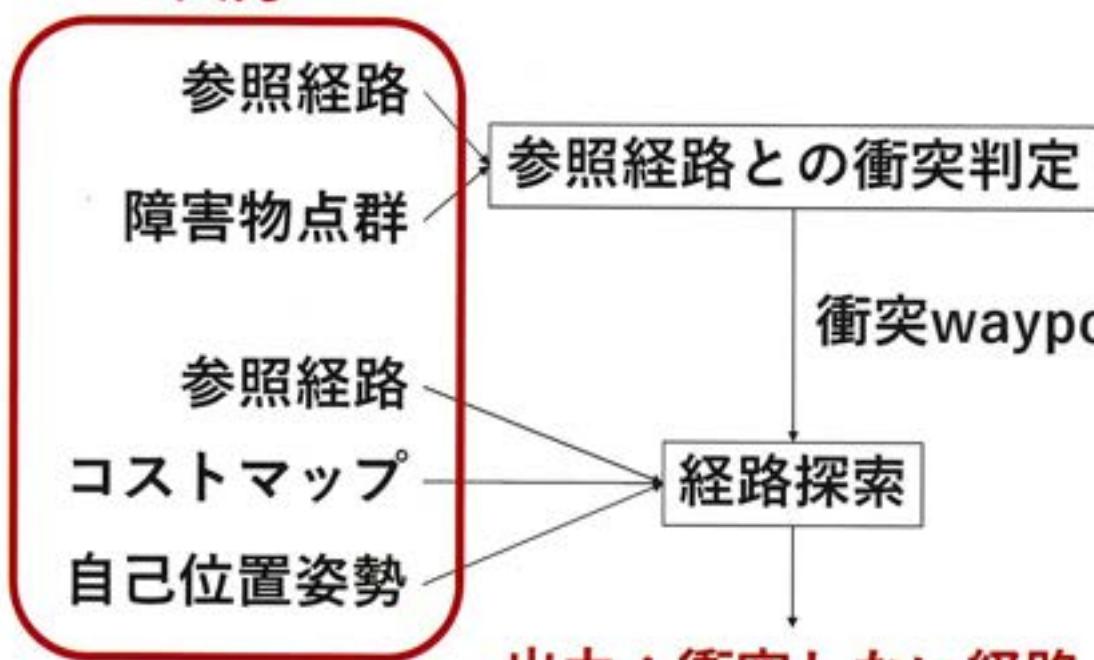
経路の表現

経路は線ではなく点列で構成される



経路生成の流れ

入力



参照経路との衝突判定

コストマップは
使わない

waypointから
衝突判定距離内の
障害物点数で判定

経路探索のゴールとして使用

衝突判定距離

障害物点

参照経路点列



障害物点数閾値0の場合



衝突 衝突

障害物点数閾値1の場合



衝突

Hybrid A*による経路探索手順

1. コストマップ各グリッドに姿勢を示すノードを配置
2. スタートからゴールまでコストが低いノードを順に選択して経路を生成

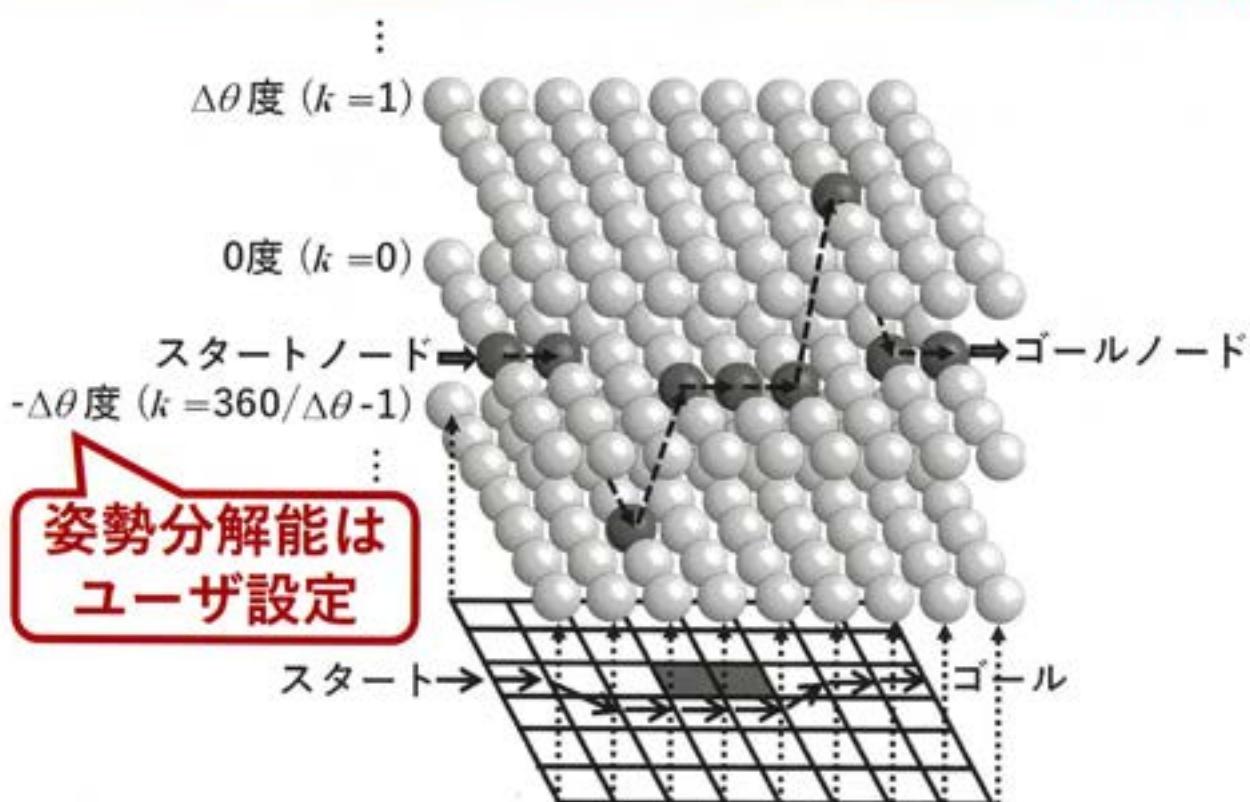
ゴールまで衝突なく移動

コスト(以下の合計) :

- ・ノードからゴールまでの距離
- ・スタートからノードまでの移動距離
- ・ノードが位置するコストマップの値



ノード探索のイメージ



Hybrid A* 探索の特徴

探索する(コスト計算する)ノードはロボットが移動できる可能性があるノードに限定

自動車などの車輪移動ロボット
は真横には移動できない



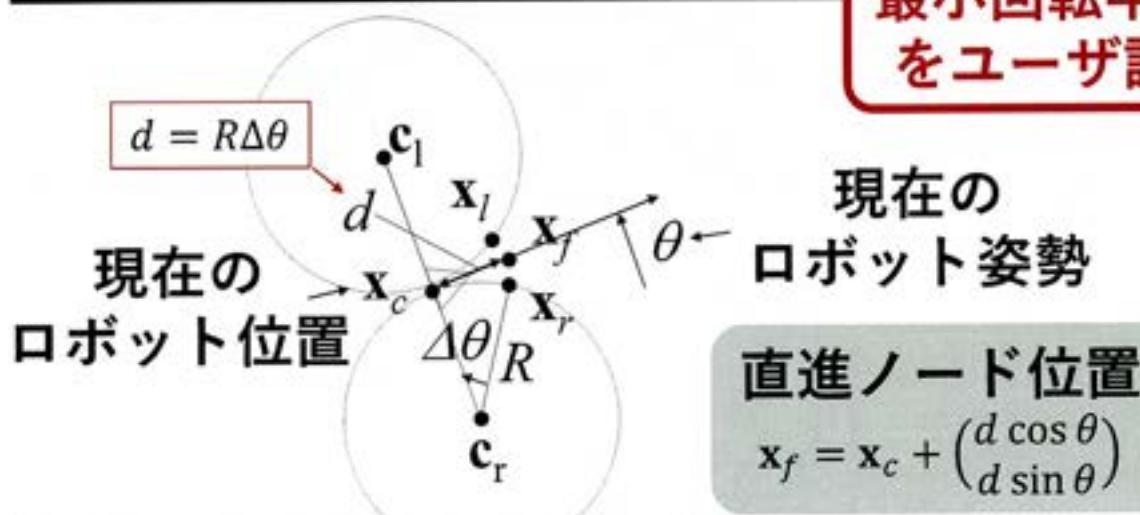
Autowareでは3ノード
(直進・左円弧・右円弧)に絞って探索



3ノードの具体的な位置(直進)

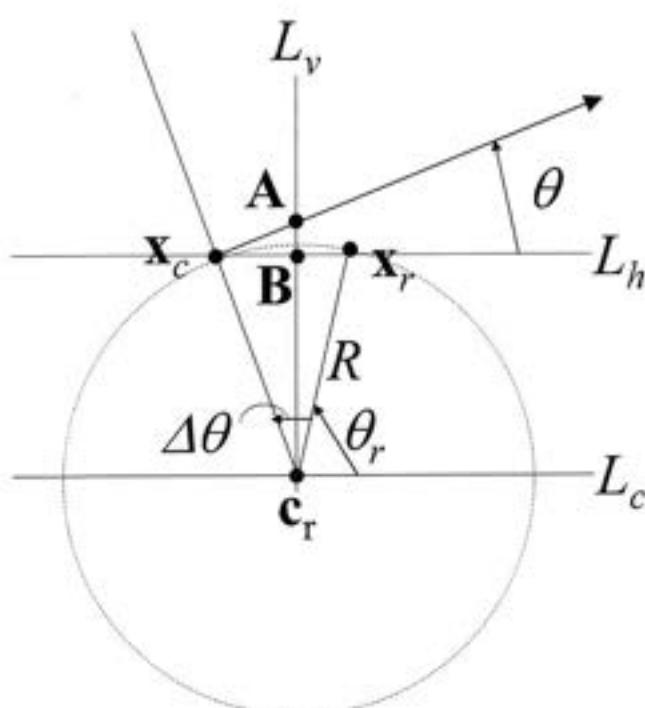
車輪に移動拘束があるロボットは
円軌道ならば安定して追従可能

**最小回転半径R
をユーザ設定**



3ノードとも移動距離は
設定した半径と姿勢分解能で決まるdとする

3ノードの具体的な位置(円弧)



$$c_r = x_c + \begin{pmatrix} R \sin \theta \\ -R \cos \theta \end{pmatrix}$$

$$\theta_r = \frac{\pi}{2} + \theta - \Delta\theta$$



右ノード位置

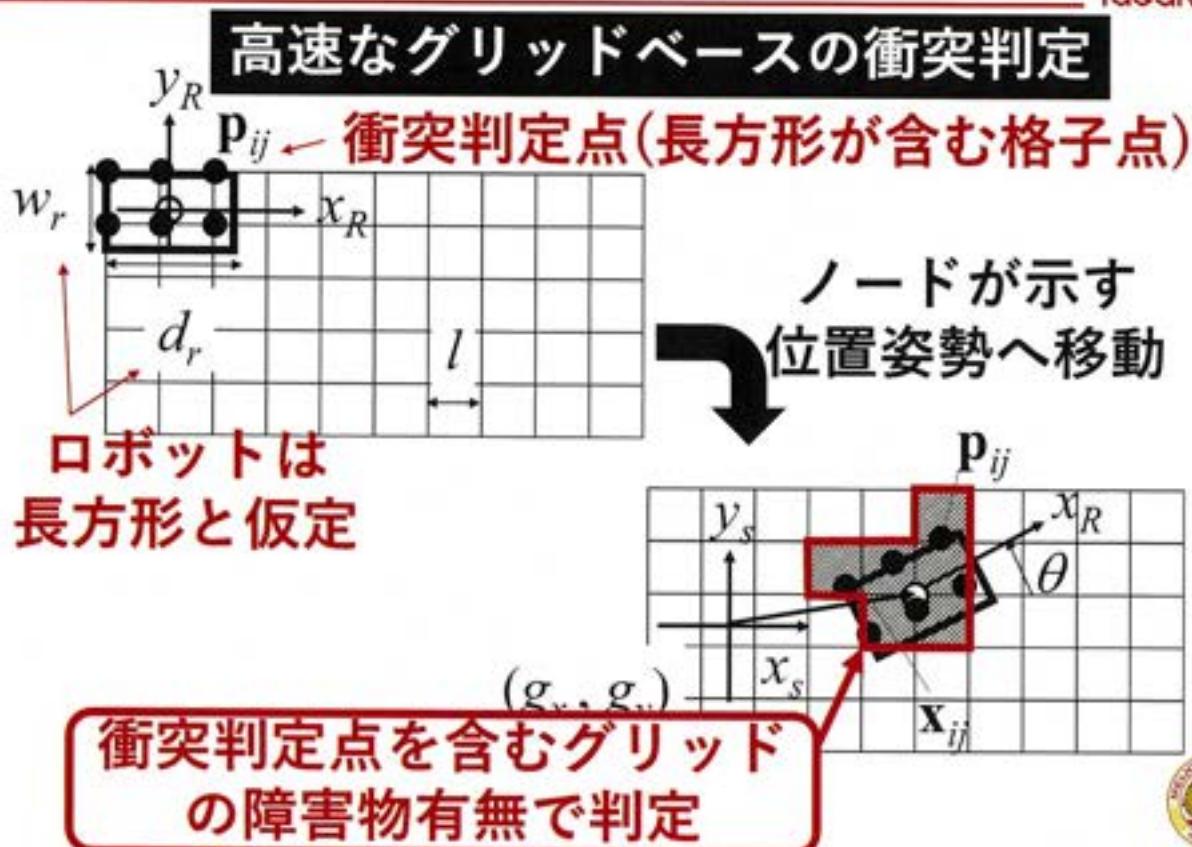
$$x_r = c_r + \begin{pmatrix} R \cos \theta_r \\ R \sin \theta_r \end{pmatrix}$$

右ノード姿勢

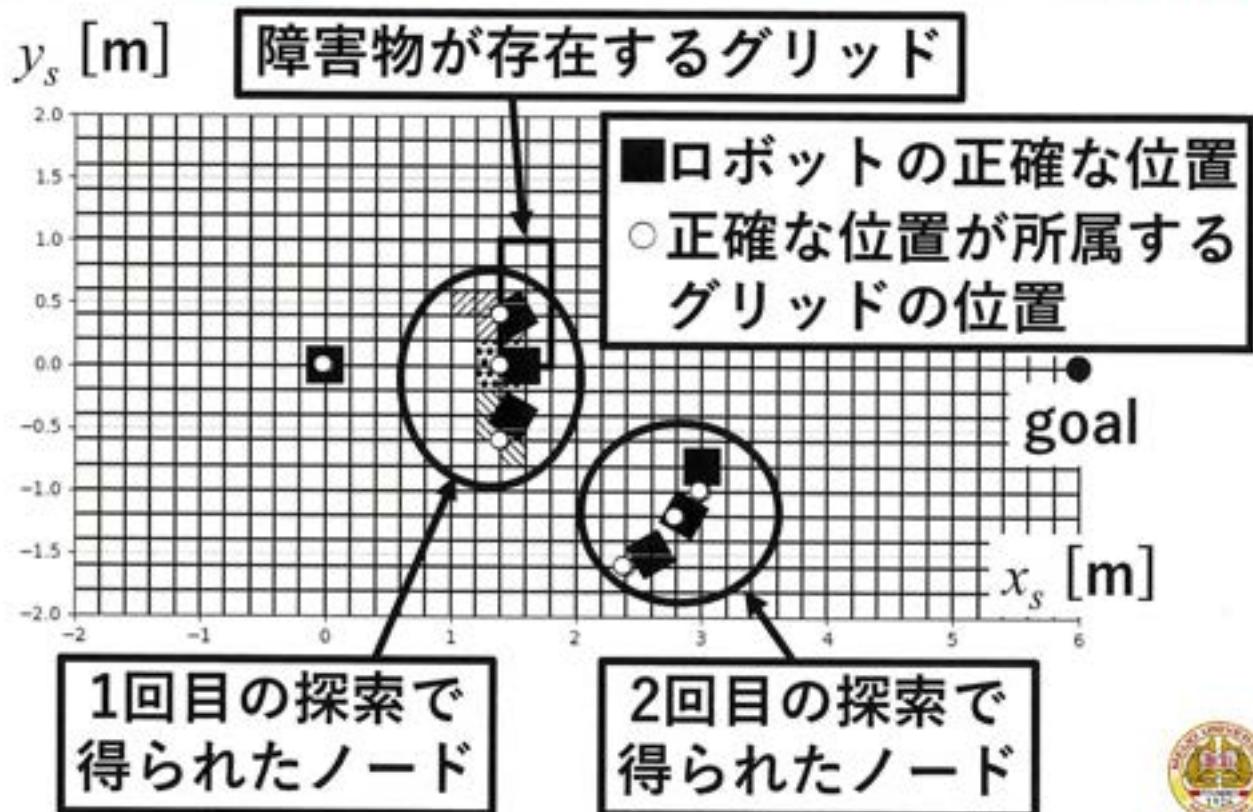
$$\theta - \Delta\theta$$



探索中の衝突判定

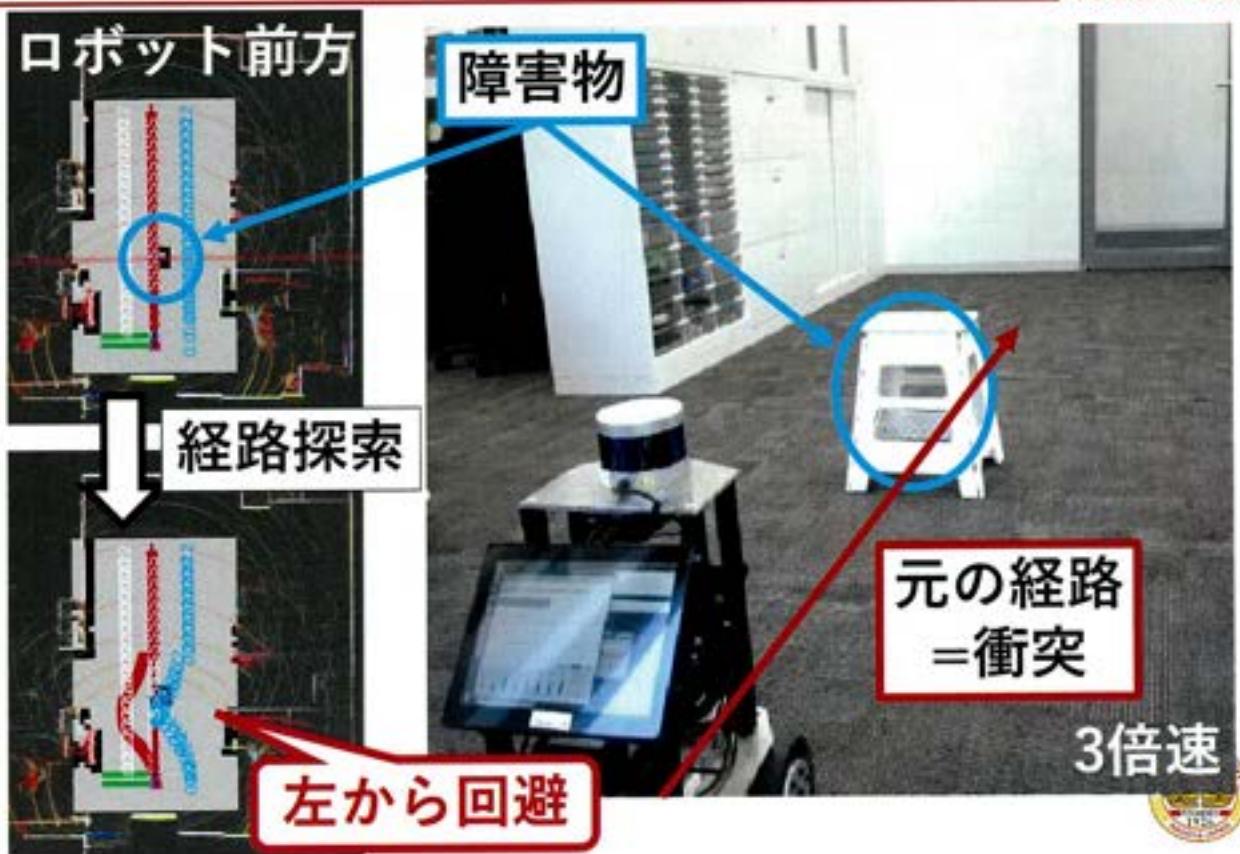


経路探索例



Autowareでの回避動作

Tasaki Lab.



経路生成利用時の注意

Tasaki Lab.

ロボットサイズは安全性を考慮して設定する



■ 衝突判定に用いるグリッド

ロボットサイズは通路幅を考慮しつつ
少し大きめに設定する



■経路生成法は2種類

- ・参照経路上に障害物がない場合
→参照経路をそのまま出力
- ・障害物がある場合
→Hybrid A*で探索した経路を出力

■Hybrid A*はコストマップのグリッドベースで回避経路を探索



経路追従

Pure Pursuit

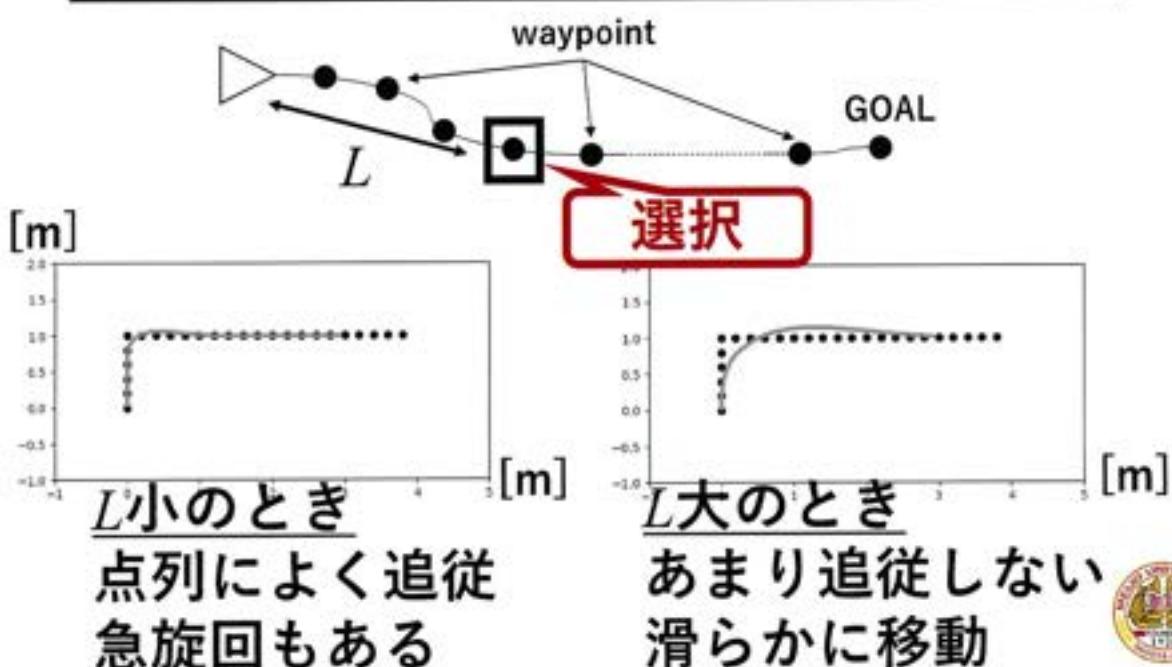
経路追従 (Pure Pursuit) の流れ

1. 経路の中から目標点を決める
2. ロボットと目標点をつなぐ円弧の半径を計算
3. 円弧軌道を描く角速度を計算

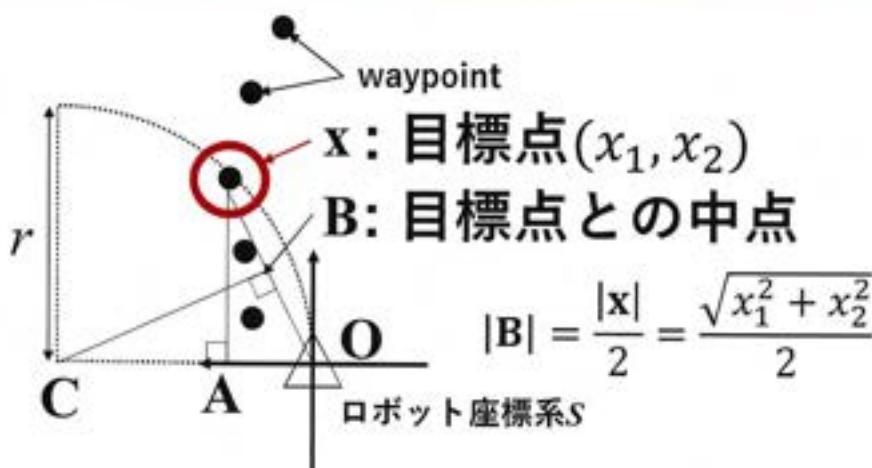


1. 目標点を決める

現在位置からユークリッド距離 L 以上で
最短のwaypointを目標点として選択



2. 円弧の半径 r を計算



$\triangle OxA$ と $\triangle OCB$ は相似なので

$$r = \frac{|\mathbf{x}|}{|\mathbf{A}|} |\mathbf{B}| = \frac{\sqrt{x_1^2 + x_2^2}}{|x_2|} \frac{\sqrt{x_1^2 + x_2^2}}{2} = \frac{x_1^2 + x_2^2}{2|x_2|}$$



3. 角速度を計算

移動ロボット(モーター)の制御

→ 座標ではなく速度を与える

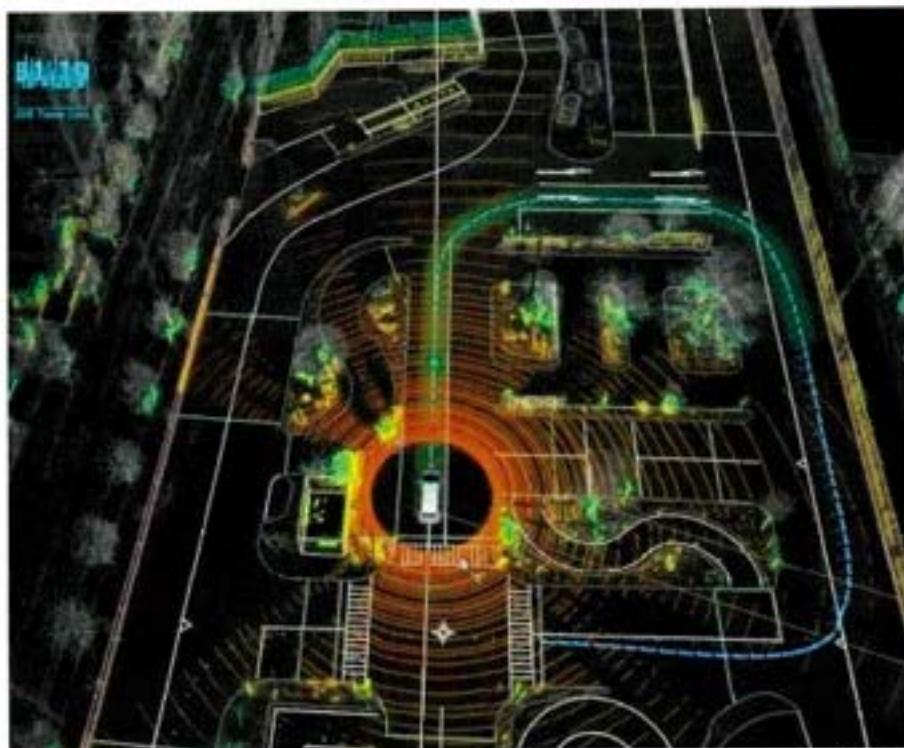
並進速度・角速度

並進速度 v ：waypointが保持している速度を利用
角速度 ω ：半径×角速度=並進速度の式を利用

$$\omega = \frac{v}{r} = \frac{2x_2 v}{x_1^2 + x_2^2}$$

x_2 の絶対値は回転方向の正負を表すため削除





制御を安定させるため速度に応じて L を変更

$L=kv$

- 経路はあくまで参考であり実際は円弧軌道で経路に追従する
- 円弧軌道を描くための目標点の選び方で軌道の滑らかさや追従性能が変わる



SLAMの基礎とROS による自律移動システム への応用

3限目：Robot Operating System (ROS)

Tasaki Lab.



3限目内容

2

Tasaki Lab.

■ROSの紹介

- ROSの構造
- ROSの開発手順

■ROSプログラミングデモ

- ROSでソフトウェアを作成する
- ロボットシミュレータを起動する



ロボットソフトウェアの構造

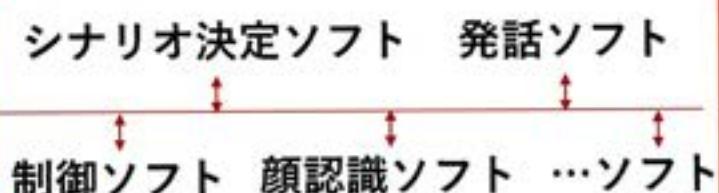
■スター型

- P2P



■バス型

- ブロードキャスト



ROS登場以前
ソフト間で独自のメッセージをソケット通信

通信の標準化

OpenRTM-aist :

日本で開発したロボット用SW開発方式
ロボットソフト部品をRTCという共通形式で作成



- 独自のメッセージはすたれる
- ソフトウェアは使いまわし構想があるも世界標準にならず



**Robot Operating System (ROS)の登場
～ロボット開発が容易に～**



ROSとは

- ロボットシステム開発を支援する
プラットフォームで以下を提供

- ロボットソフト部品の開発枠組み
枠組みに従えば使いまわしが楽
- ロボットソフト部品（パッケージ）
SLAM、逆運動学、…
- 開発ツール
可視化ツール、デバッグツール、…

ロボット開発に必要な
SW部品はそろっている



ROSの勝因

Tasaki Lab.

■オープンイノベーションマインド

- ・サンプルソフトをたくさん公開

■ROS対応のハードも提供

- ・アーム付き移動ロボットを無償提供
- ・センサ、ロボットも続々対応

一例



ZEDステレオカメラ Denso COBOTTA



ROSのインストール

Tasaki Lab.

ubuntu18の場合

ROS インストール

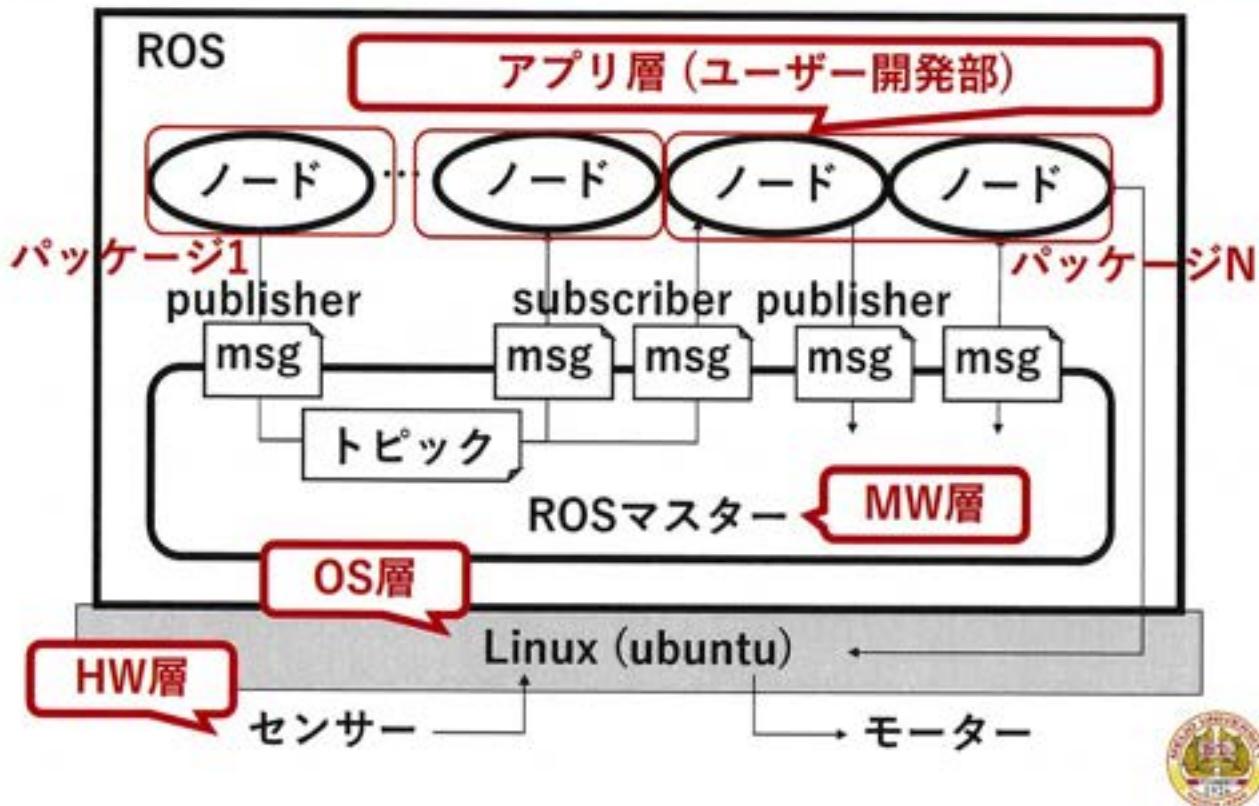


```
$ sudo sh -c 'echo "deb ¥
http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > \
/etc/apt/sources.list.d/ros-latest.list'
$ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' \
--recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
$ sudo apt update
$ sudo apt install ros-melodic-desktop-full
$ echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

数回コマンドを打つだけでインストール可能
(20分程度)



ROSの構成



ROS動作確認

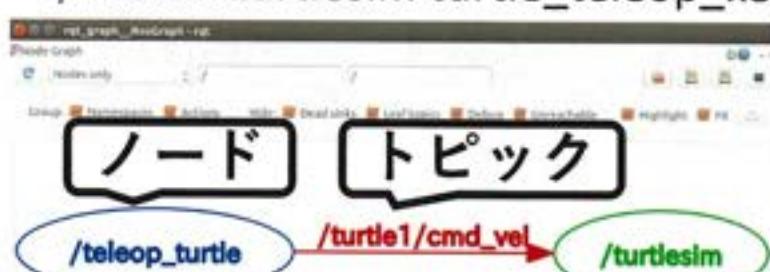
亀を動かす

\$ roscore

\$ rosrun turtlesim turtlesim_node

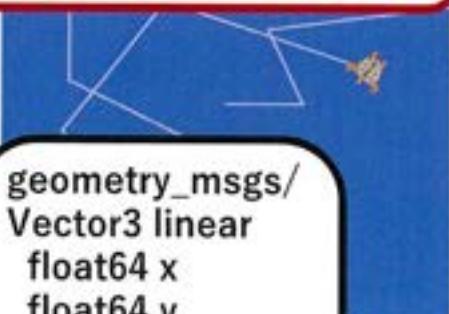
\$ rosrun turtlesim turtle_teleop_key

rosrun パッケージ名 ノード名
パッケージに属するノード実行



msgの型
`geometry_msgs/Twist`

`geometry_msgs/Vector3 linear`
`float64 x`
`float64 y`
`float64 z`
`geometry_msgs/Vector3 angular`
`float64 x`
`float64 y`
`float64 z`



turtlesimノードを実機用ノードにすれば実機も動く

複数ノード同時起動(roslaunch)

\$ rosrun (パッケージ名) launch ファイル

roscoreの起動も自動化

launch ファイル例

```

1 <launch>
2 <!-- launch node -->
3 <node pkg="turtlesim" type="turtlesim_node" name="turtle">
4 </node>
5 <node pkg="turtlesim" type="turtle_teleop_key" name="key" />
6 </launch>

```

type: 実行ノード名

name: 登録名

- ・ <node>は</node>で閉じることもできる
- ・ <node ...><remap from="a" to="b"/></node>で使用するトピックをaからbにできる



ROSデバッグツール

rqt_graph
ノードとトピックの可視化

\$ rqt_graph



ROSデバッグツール

Tasaki Lab.

rviz

データ、結果の可視化、I/Fの提供

\$ rviz



ROSデバッグツール

Tasaki Lab.

rosbag

時系列でトピックを保存&後から再生

全てのトピックを<バグファイル名>に保存する

\$ rosbag record -a -O <バグファイル名>

特定の<トピック名>だけ保存する

\$ rosbag record <トピック名> -O <バグファイル名>

保存した<バグファイル名>を再生する

\$ rosbag play <バグファイル名>

取得した時間と同じタイミングで
保存したトピックを出力

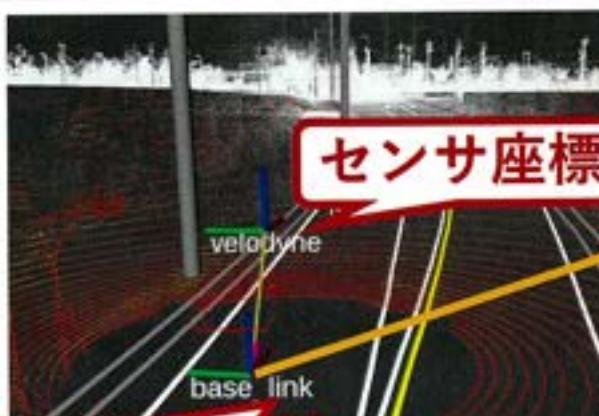


ROSの重要パッケージ: TF

座標変換を行うライブラリ

launchファイル等で変換したい座標系の関係を記述しておくだけで任意の座標変換を実現

Autowareの座標系の例



地図座標系

map

世界座標系

world

ロボット座標系

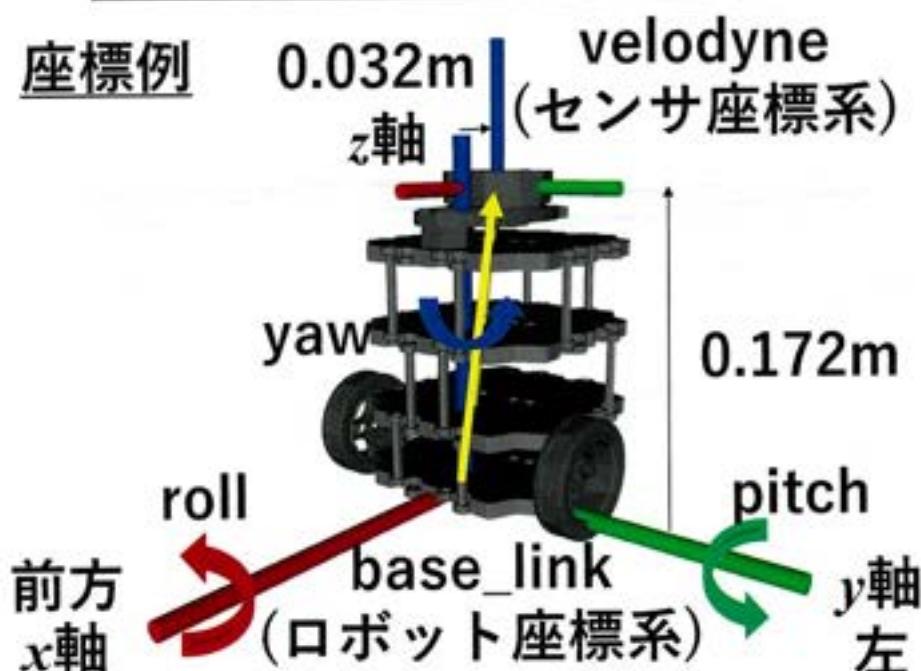
mapとworldは屋内移動なら一致でよい

ロボット座標系とセンサ座標系の例

センサ位置はロボット座標系(両輪中心)から見たセンサの位置姿勢を入力

座標例

0.032m velodyne
z軸 (センサ座標系)



| |
|----------|
| x=-0.032 |
| y=0 |
| z=0.172 |
| roll=0 |
| pitch=0 |
| yaw=0 |



座標系を記述するlaunchファイル例

4行目:/base_linkからみた/velodyneの位置姿勢をargsで指定

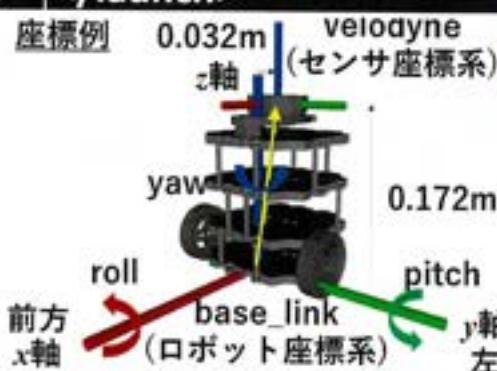
x, y, z, [m]

roll, pitch, yaw [rad]の順

```

1 <launch>
2   <node pkg="tf" type="static_transform_publisher"
3     name="base_link_to_localizer"
4     args="-0.032 0 0.172 0 0 0 /base_link /velodyne 10" />
5 </launch>

```



座標更新周期[ms]



ROSノード開発手順

1. ワークスペースを作成
2. パッケージを作成
3. ソースコードを作成
4. CMakeLists.txt修正
5. ビルド

ROSは開発物を管理するディレクトリ構成が重要
→ディレクトリ構成の作法を守る



ROSディレクトリ構成例

```
/autoware_ws ワークスペース
|- /build
|- /devel パッケージ管理
|- /src ディレクトリ
  |- CMakeLists.txt
  |- /パッケージ名 自作ディレクトリ
    |- CMakeLists.txt
    |- package.xml
    |- /launch
    |- /include
    |- /src 自動生成
      |- /ノード名 ディレクトリ
        |- /launch
        |- /include
        |- /src ノードのソース
          |- /node_name コードを保存
```

launch
ファイル
を保存



ROS プログラミングデモ

自作ノードの作成

ワークスペース/パッケージ作成

21

手順1,2

```
$ mkdir -p ~/shared_dir/autoware_ws/src  
$ cd ~/shared_dir/autoware_ws/src  
$ catkin_init_workspace  
$ catkin_create_pkg パッケージ名 必要パッケージ  
$ cd ..  
$ catkin_make
```

ソースコード変更のたびに必要

パッケージ名 : test_pkg

文字列msgを扱う

必要パッケージ : rosccpp std_msgs の場合

catkin_create_pkg test_pkg rosccpp std_msgs

msg用パッケージの例: sensor_msgs,
geometry_msgs, autoware_msgs

ディレクトリ構成確認

22

手順1,2,3

```
~/shared_dir/autoware_ws  
|- /build  
|- /devel  
|- /src  
  |- CMakeLists.txt  
  |- /test_pkg  
    |- CMakeLists.txt  
    |- package.xml  
    |- /include  
    |- /src
```

ノード作成のため
srcに移動

```
$ cd ~/shared_dir/autoware_ws/src/test_pkg/src  
$ gedit talker.cpp
```

talker.cpp (publisher) の作成

手順3

```

1 #include "ros/ros.h"
2 #include "std_msgs/String.h"
3 #include <iostream>
4
5 int main(int argc, char **argv){
6     ros::init(argc, argv, "talker");
7     ros::NodeHandle nh_pub;
8     ros::Publisher chatter_pub = nh_pub.advertise<std_msgs::String>("chatter", 1000);
9
10    int count = 0;
11    ros::Rate loop_rate(10);
12    while (ros::ok()){
13        std_msgs::String msg;
14        std::stringstream ss;
15        ss << count;
16        msg.data = ss.str();
17        chatter_pub.publish(msg);
18        loop_rate.sleep();
19        ++count;
20    }
21    return 0;
22 }
```

ノード設定部
publisher作成

ROS talker

msg送信部
loop_rate周期

保存(S)

gedit右上の保存ボタン
を押してから終了

listener.cpp (subscriber) の作成

24

手順3

\$ gedit listener.cpp

msg受信部

msg受信のたびに呼ばれる関数

```

1 #include "ros/ros.h"
2 #include "std_msgs/String.h"
3
4 //コールバック関数実装
5 void chatterCallback(const std_msgs::String::ConstPtr& msg){
6     ROS_INFO("I heard: [%s]", msg->data.c_str());
7 }
8
9 int main(int argc, char **argv){
10    ros::init(argc, argv, "listener");
11    ros::NodeHandle n;
12    ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);
13
14    //メッセージ待ち
15    ros::spin();
16    return 0;
17 }
```

ノード設定部
subscriber作成

msg待ち部
ループ+spinOnce
でもよい



```

~/shared_dir/autoware_ws
|- /build
|- /devel
|- /src
  |- CMakeLists.txt
  |- /test_pkg
    |- CMakeLists.txt
    |- package.xml
    |- /include
    |- /src
      |- talker.cpp
      |- listener.cpp

```

ビルドのため編集



```

$ cd ~/shared_dir/autoware_ws/src/test_pkg
$ gedit CMakeLists.txt &

```

末尾(206~209行目くらい)に以下を追加

```

206 add_executable(talker src/talker.cpp)
207 target_link_libraries(talker ${catkin_LIBRARIES})
208 add_executable(listener src/listener.cpp)
209 target_link_libraries(listener ${catkin_LIBRARIES})

```

add_executable

ビルドするソースコードとノード名を記述

target_link_libraries

指定したノードが使用するライブラリを記述

■catkin_make

- ・ワークスペース内のROSパッケージをビルド
- ・ワークスペースのトップディレクトリで実行

```
$ cd ~/shared_dir/autoware_ws
$ catkin_make
```



0. roscoreを起動

- ・他で起動している場合は起動しない
(2重起動はエラーの元)

1. 環境変数設定

2. rosrunもしくはroslaunch



talker/listener実行手順例

ターミナルを3つ起動 (A, B, C) し、以下を実行

ターミナルA

```
$ roscore
```

ターミナルB

sourceで環境変数設定

```
$ source ~/shared_dir/autoware_ws/devel/setup.bash  
$ rosrun test_pkg talker
```

ターミナルC

ターミナルごとに設定

```
$ source ~/shared_dir/autoware_ws/devel/setup.bash  
$ rosrun test_pkg listener
```

ROS プログラミングデモ

シミュレータの起動

■Turtlebot3 burger

- ROS対応ロボットとして有名
- 10万円以下で購入可能
- センサは2D LiDAR

Autoware対応ではなく高精度ではない

Autoware対応でない
ロボットも使えるようになる



シミュレータの導入

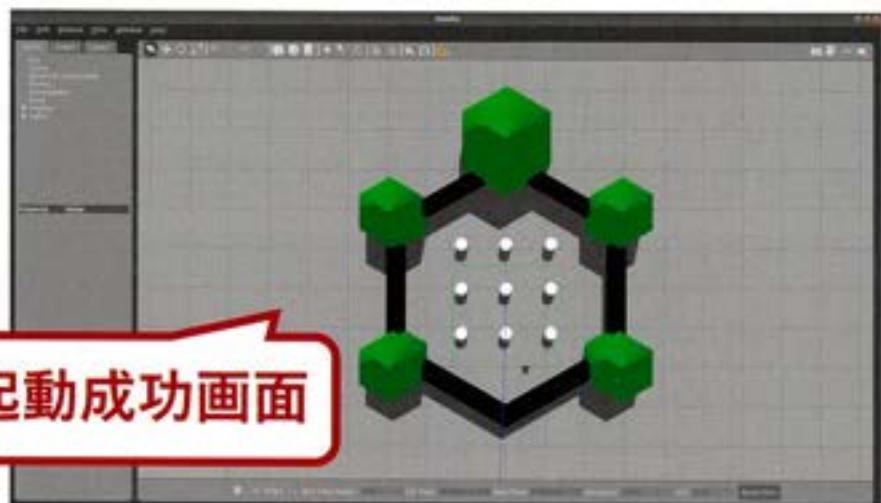
```
$ mkdir -p ~/shared_dir/turtlebot_ws/src
$ cd ~/shared_dir/turtlebot_ws/src
$ catkin_init_workspace
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git
$ git clone
  https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
$ git clone
  https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
$ git clone
  https://github.com/ROBOTIS-GIT/turtlebot3_gazebo_plugin.git
$ cd ..
$ catkin_make
```

時間がかかるので今回
は省略(事前に実施)

シミュレータ起動確認

```
$ cd ~/shared_dir/turtlebot_ws
$ source devel/setup.bash
$ export TURTLEBOT3_MODEL=burger
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

ROSに
関係ない
シミュレータ
固有の設定



起動成功画面



シミュレータ動作確認

シミュレータ起動したまま別のターミナルを起動

```
$ cd ~/shared_dir/turtlebot_ws
$ source devel/setup.bash
$ export TURTLEBOT3_MODEL=burger
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Tasaki Lab.

ロボットをキーボードで操作

w: 前進

a: 左旋回 s: 停止 d: 右旋回

x: 後進



ロボットの設定ファイル(xacro)

センシング範囲を広げる場合

```
$ cd ~/shared_dir/turtlebot_ws/src/turtlebot3
$ cd turtlebot3_description/urdf
$ gedit turtlebot3_burger.gazebo.xacro &
```

.xacroはシミュレータの設定ファイル

119行目あたりの最大センシング距離を10.0mに

```
117 <range>
118 <min>0.120</min>
119 <max>10.0</max>
120 <resolution>0.015</resolution>
121 </range>
```

**base_scanで検索
→2つのbase_scanの間にある**

ROSまとめ

■ROS

- ・プログラムを分散実行
- ・便利なツール (roslaunch, rviz, rosbag)

■ROSの実装

- ・作成するのはワークスペース、パッケージ、ソースコード
(ディレクトリ構成に注意)
- ・CMakeLists.txtを修正してビルド



SLAMの基礎とROS による自律移動システム への応用

4限目：Autoware



4限目内容

2

Tasaki Lab.

■Autowareの紹介

- Autowareの概要/操作法

■Autowareによる自動運転デモ

- bagファイルによる自動運転動作確認



Autoware

Tasaki Lab.

■オープンソースで世界初の自動運転統合開発SW

- ・認知/判断/制御すべてを提供
- ・ソースコードも公開

■ROSで構築

- ・改良が容易

■自動車もロボットも操作可能



Autowareのインストール

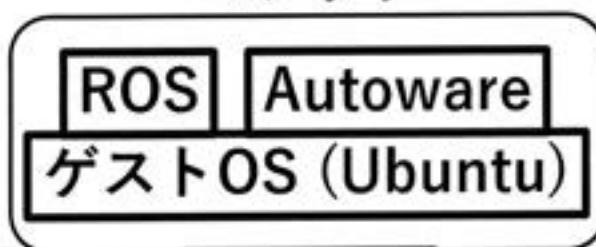
Tasaki Lab.

■必要なライブラリが多い

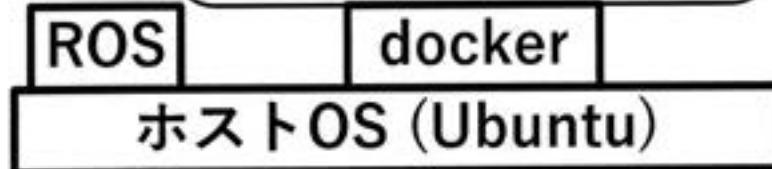
■ライブラリの指定が厳しい

➡ dockerでのインストール推奨

コンテナ



仮想環境
(ゲストPC)



実PC
(ホストPC)



dockerの利点

■PC内に仮想的なPCを構築

- ・ホストPCの環境が破壊されない
- ・不要になったら仮想PCごと破棄できる
- ・ライブラリがインストールされたPCごと配布されるため環境設定がいらない

■他の仮想環境に比べて高速

- ・全てのハードウェアをエミュレートしていない

ゲストOSとのやり取り

docker終了後
も保存したい
ファイル

共有ディレクトリ
~/shared_dir

ファイル

ホストOS
(ホストPC)

ゲストOS
(ゲストPC)

一般的なROS

Autoware用のROS

トピック通信
(一般的なROSのmsgだけ)

リアルタイムデータ

dockerのインストール準備

必要なソフトのインストール

```
$ sudo apt update
$ sudo apt install apt-transport-https ca-certificates
curl software-properties-common
```

リポジトリの設定

```
$ curl -fsSL
https://download.docker.com/linux/ubuntu/gpg
| sudo apt-key add -
$ sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable"
```



dockerのインストールと設定

dockerのインストール

```
$ sudo apt update
$ sudo apt install docker-ce
```

dockerの動作確認

\$ docker -v

**Docker version…のように
バージョンが表示されればOK**

sudoなしでdockerを使う設定

```
$ sudo usermod -aG docker $USER
```



nvidia-docker2のインストール

Tasaki Lab.

リポジトリの設定

参考：nvidiaGPU利用時

```
$ curl -sL https://nvidia.github.io/nvidia-docker/gpgkey
| sudo apt-key add -
$ distribution=$(./etc/os-release;echo $ID$VERSION_ID)
$ curl -sL
https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list
| sudo tee /etc/apt/sources.list.d/nvidia-docker.list
$ sudo apt-get update
```

インストールと動作確認

```
$ sudo apt-get install nvidia-docker2
$ sudo pkill -SIGHUP dockerd
$ sudo docker run --runtime=nvidia --rm
nvidia/cuda nvidia-smi
```

Autoware関連ファイルの取得

Tasaki Lab.

Autoware起動スクリプトの取得

```
$ mkdir -p ~/shared_dir/autoware
$ cd ~/shared_dir/autoware
$ git clone
https://gitlab.com/autowarefoundation/autoware.ai/docker.git
```

Autoware起動練習データの取得

```
$ cd ~/shared_dir
$ wget
https://autoware-ai.s3.us-east-2.amazonaws.com/sample_moriyama_150324.tar.gz
$ tar -xvzf sample_moriyama_150324.tar.gz
$ wget
https://autoware-ai.s3.us-east-2.amazonaws.com/sample_moriyama_data.tar.gz
$ tar -xvzf sample_moriyama_data.tar.gz
```

Autowareの起動

nvidia ドライバによる描画支援がない場合

```
$ cd ~/shared_dir/autoware/docker/generic
$ ./run.sh -c off -r melodic -t 1.14.0
$ roslaunch runtime_manager runtime_manager.launch
```

nvidia ドライバによる描画支援がある場合

```
$ cd ~/shared_dir/autoware/docker/generic
$ ./run.sh -r melodic -t 1.14.0
$ roslaunch runtime_manager runtime_manager.launch
```



ランタイムマネージャー



クリック操作で自動運転機能/ROS機能を起動

CPU0 CPU1 CPU2 CPU3 CPU4 CPU5 CPU6 CPU7 CPU8 CPU9 CPU10 CPU11 CPU12 CPU13 CPU14 CPU15 CPU16 CPU17

AutoWare



ランタイムマネージャーのタブ

| タブ | ツールの内容 |
|------------|-----------------------|
| Setup | ロボット座標系から見たセンサ位置姿勢を設定 |
| Map | 三次元地図を読み込み |
| Sensing | センサの起動/取得した点群のフィルタリング |
| Computing | 自律移動各機能のROSノードを起動 |
| Interface | 自律移動を開始 |
| Simulation | bagファイルの操作をサポート |

Autowareの基本操作

- 使いたいツールに関連するタブを選択
- 画面内にあるチェックボックス・ボタンのクリックでツールを利用



静的環境での自動運転手順

■準備

- 地図作成、参照経路作成

■手順

まずは地図/参照経路あり & 仮データで手順2以降を練習

- センサ接続
- 自己位置推定
- 経路生成
- 経路追従

静的環境：地図にない障害物が存在しない環境
 → 経路生成は参照経路を出力するだけ

1. センサ接続

仮データ (bagファイル)版

仮センサデータの再生と一時停止

16

Tasaki Lab.



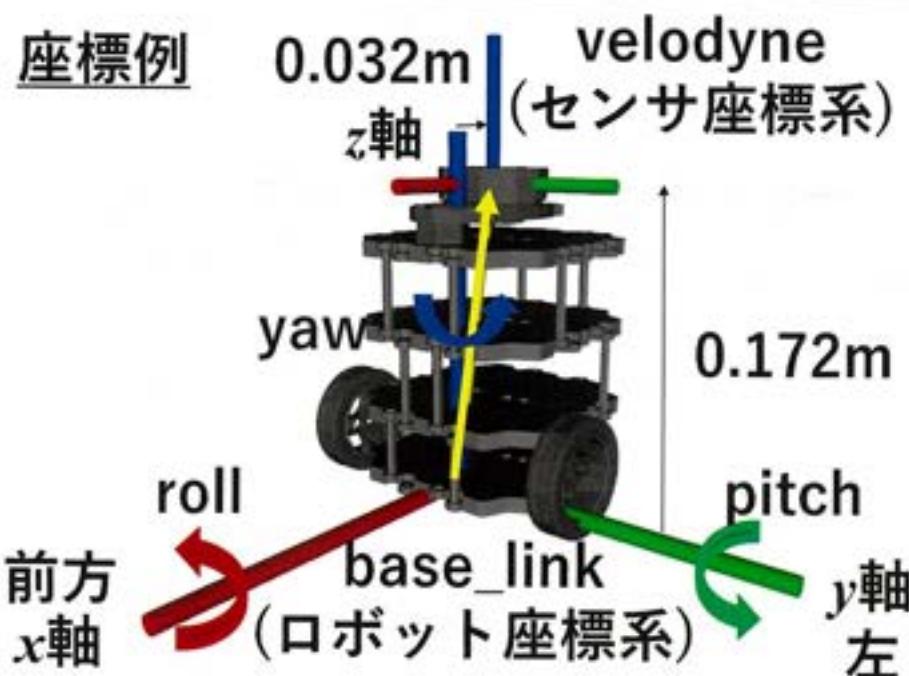
- ①クリック (ファイル選択) → バーに表記のbagファイルを選択
- ②クリック (データの再生)
- ③Playingになることを確認
- ④クリック (データの一時停止)

自律移動に必要な機能を起動終えたら再再生



ロボット座標系とセンサ座標系

センサ位置はロボット座標系(両輪中心)
から見たセンサの位置姿勢を入力



| |
|----------|
| x=-0.032 |
| y=0 |
| z=0.172 |
| roll=0 |
| pitch=0 |
| yaw=0 |



2. 自己位置推定起動

自己位置推定起動手順

20

Tasaki Lab.

1. 地図読み込みと座標設定

2. 自己位置推定関連ROSノードを 起動

自己位置推定関連ROSノード

| 起動名 | タブ | 役割 |
|-------------------|-----------|----------------|
| voxel_grid_filter | Sensing | 三次元点群フィルタリング |
| nmea2tfpose | Computing | GNSSのデータを取得 |
| ndt_matching | Computing | 自己位置推定 |
| vel_pose_connect | Computing | 自己位置推定トピック名を変更 |

nmea2tfposeは屋内移動ロボットでは不要





- ①クリック（表記pcdファイルを全選択）
 - ②クリック（地図読み込み）
 - ③クリック（表記tf.launchを選択）
 - ④TFで世界座標と地図座標の関係を設定

三次元点フィルタリング (Sensingタブ)²²



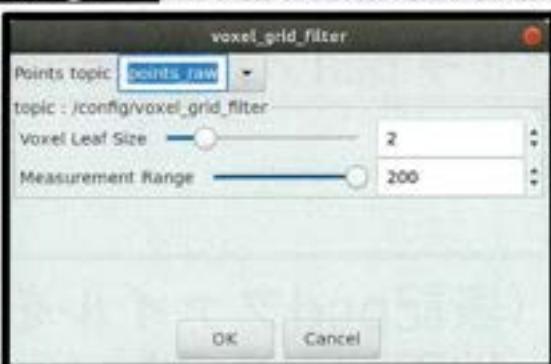
- ①appクリック(voxel_grid_filter設定画面表示)
 - ②OKクリック(表示通りの設定か確認)
 - ③チェック(voxel grid filter起動)

voxel_grid_filterのパラメータ

Tasaki Lab.

別口ポート応用にはvoxel_grid_filterの設定必要

| パラメータ | 役割 |
|-------------------|------------------|
| Voxel Leaf Size | フィルタサイズ[m]を設定する。 |
| Measurement Range | センシング距離[m]を設定する。 |



AutowareのROSノード起動基本操作

1. appで設定画面を開いてパラメータ設定
2. ROSノード起動用チェックボタンをチェック

自己位置推定起動 (Computingタブ)

Tasaki Lab.



ndt_matchingのパラメータ

Tasaki Lab.

別口ロボット応用にはndt_matchingの設定必要

| パラメータ | 役割 |
|--------------------|--|
| Initial Pos / GNSS | 自己位置推定ノード起動時の大まかな自己位置姿勢を与える。 屋内移動ロボットではGNSSを使わないのでInitial Posを選択する。位置[m]と姿勢[rad]はmap座標系とする。 |
| Resolution | NDTマッチングで使用する箱の大きさ[m]を設定する。 |
| Use Odometry | NDTマッチングの大まかな自己位置姿勢に車輪回転量等から計算できるオドメトリを使用する場合チェックする。 |



データ再再生/rviz起動(Simulationタブ)

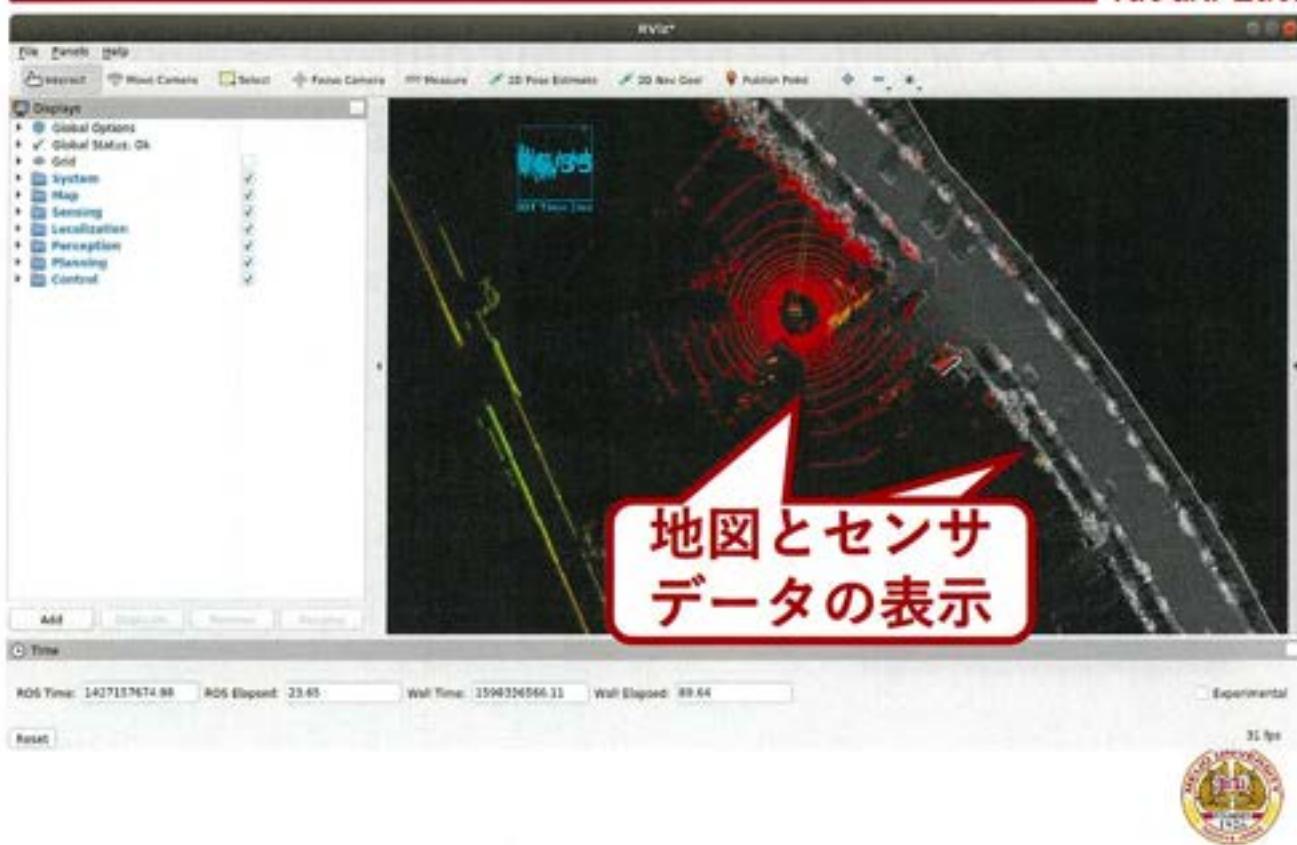
Tasaki Lab.

*再再生 = 一時停止解除



自己位置推定動作確認

Tasaki Lab.



起動ノード位置の指定方法

Tasaki Lab.

起動ノードの位置は階層構造で指定可能



タブ
画面左or右 (L/R)
ノードグループ名1
ノードグループ名2
起動名(と app)

階層
上位
↓
下位

ndt_matchingの位置指示例

Simulation/L/Localization/lidar_localizer/ndt_matching

上位 → 下位



3. 経路生成起動

静的環境

経路生成ノードの起動

30

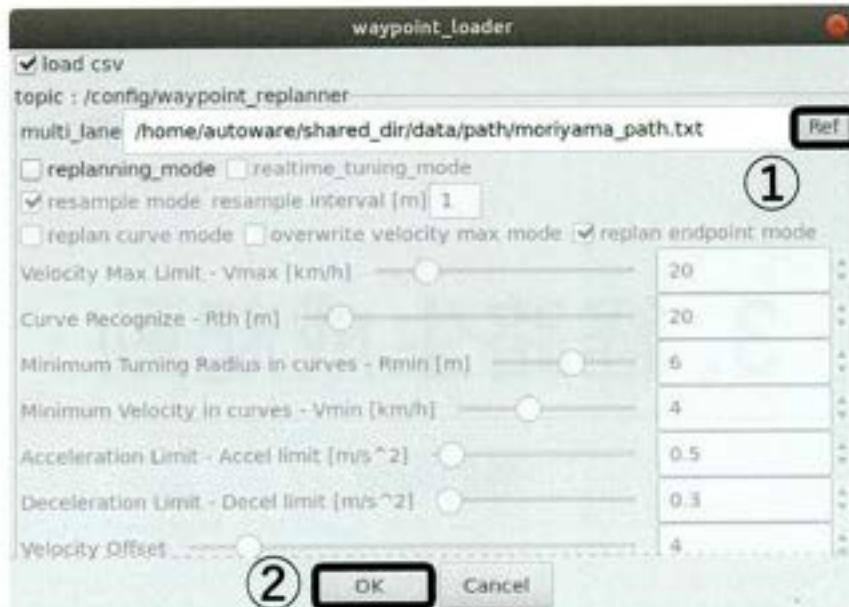
Tasaki Lab.

練習でもwaypoint_loaderの設定は必要

| ノード名 | タブ | 役割 |
|-----------------|-----------|-------------------------|
| lane_rule | Computing | 参照経路を調整する。 |
| lane_select | Computing | 現在位置に最も近いwaypointを選択する。 |
| waypoint_loader | Computing | 参照経路を読み込む。 |
| astar_avoid | Computing | 経路を生成する。 |
| velocity_set | Computing | 参照経路と障害物との衝突判定をする。 |

| ノード名 | 位置 |
|-----------------|--|
| lane_rule | Computing/R/Mission Planning/lane_planner |
| lane_select | Computing/R/Mission Planning/lane_planner |
| waypoint_loader | Computing/R/Motion Planning/waypoint_marker |
| astar_avoid | Computing/R/Motion Planning/waypoint_planner |
| velocity_set | Computing/R/Motion Planning/waypoint_planner |

waypoint_loaderのパラメータ



- ①クリック(経路ファイル選択)
②クリック(閉じる)



4. 経路追従起動

経路追従ノードの起動

Tasaki Lab.

別口ボット応用にはpure_pursuitの設定必要

| ノード名 | タブ | 役割 |
|--------------|-----------|-----------------------|
| pure_pursuit | Computing | 経路追従する。 |
| twist_filter | Computing | 追従時の異常な並進速度、角速度を抑制する。 |

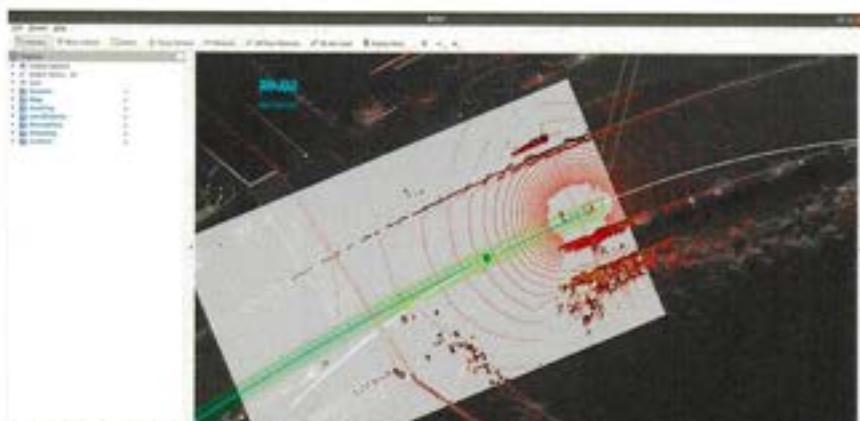
| ノード名 | 位置 |
|--------------|---|
| pure_pursuit | Computing/R/Motion Planning/waypoint_follower |
| twist_filter | Computing/R/Motion Planning/waypoint_follower |



pure_pursuitのパラメータ

Tasaki Lab.

| パラメータ | 役割 |
|----------------------------|-------------------------------|
| lookahead_ratio | 速度に応じて waypoint を選ぶ際の係数を設定する。 |
| minimum_lookahead_distance | waypoint を選ぶ最小距離[m]を設定する。 |



経路追従機能が起動されれば自律移動可能



■Autoware

- dockerで簡単インストール
- 自動運転の各機能をROSで実装して提供

■自動運転手順

1. センサ接続
2. 自己位置推定
3. 経路生成
4. 経路追従



SLAMの基礎とROS による自律移動システム への応用

5限目：SLAMデモ

Tasaki Lab.



5限目内容

2

Tasaki Lab.

■SLAMデモ

- Autowareとロボットの接続
- 地図作成

■自律移動デモ

- 自作地図を用いた自律移動の実施



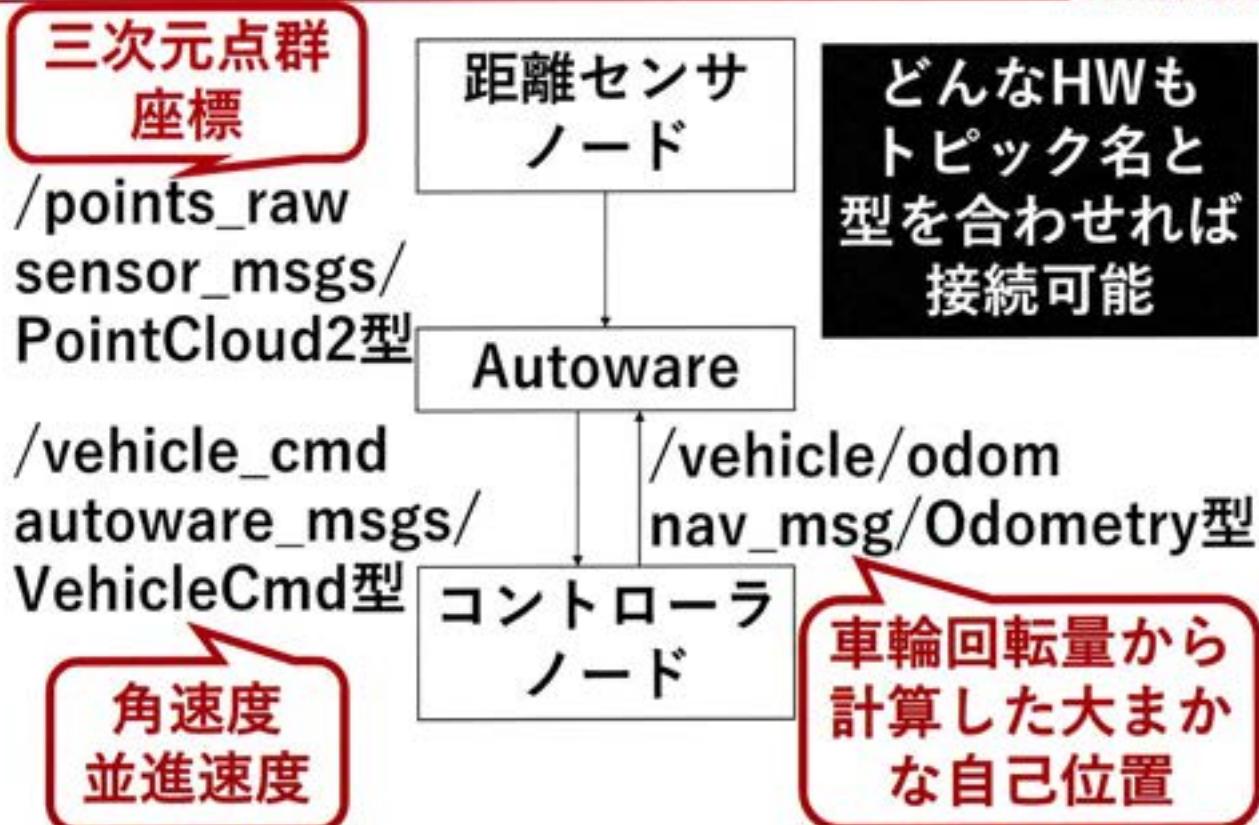
Autowareと ロボットの接続



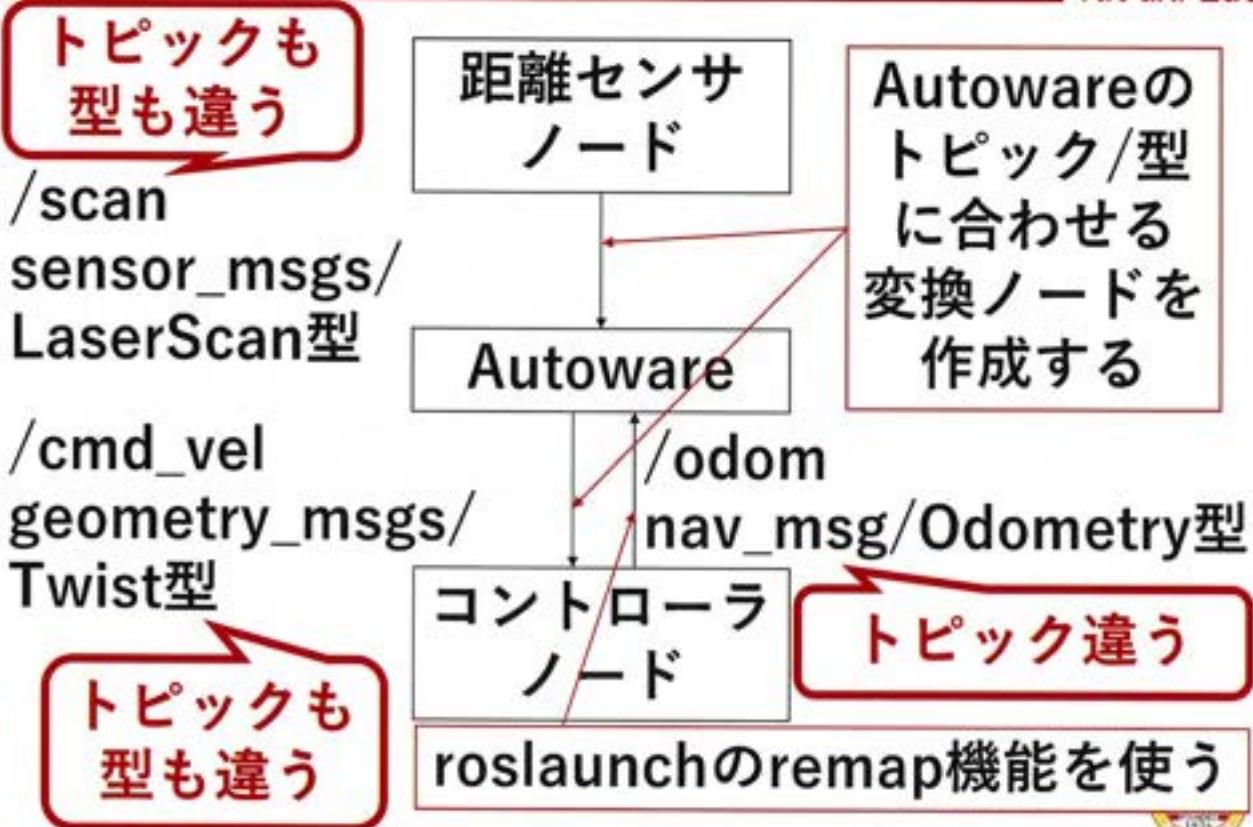
接続用のトピック名/メッセージ型

4

Tasaki Lab.



Turtlebot3の場合



ノードを作成するPC



ファイルは共有ディレクトリで一元管理すると楽



ヘッダ宣言

ros::Publisherを大域で宣言

コールバック関数{

受信したデータを送信したいmsgに変換
大域のPublisherで送信

}

main関数{

ノード設定

ros::spin()

}

- ・大域宣言された
Publisherの設定
- ・Subscriberの設定



変換ノードサンプル

<https://github.com/Meijo-TasakiLab/autoware-book>

```
$ cd  
$ git clone https://github.com/Meijo-TasakiLab/autoware-  
book.git  
$ mv autoware-book a-b
```

以降の資料で簡略化のため
a-b
という名前に変更



Turtlebotで接続用に準備するもの

Tasaki Lab.

■ センサ接続関連 変換ノード型

- ・変換ノード用ソースコード
- ・変換ノード用launchファイル
- ・CMakeLists.txt

ホスト
PCで
実行

■ コントローラ接続関連 remap型

- ・トピック変換用launchファイル

■ 移動命令関連 変換ノード型

- ・変換ノード用ソースコード
- ・CMakeLists.txt

ゲスト
PCで
実行



センサ接続関連準備

変換ノード型 10

Tasaki Lab.

1. ホストPCでsensor_pkgを作成
2. srcにソースコード保存
3. launchファイルを保存
4. ノードビルド
 - ・CMakeLists.txt修正
 - ・catkin_make



センサ接続関連準備コマンド

```
$ mkdir -p ~/shared_dir/sensor_ws/src
$ cd ~/shared_dir/sensor_ws/src
$ catkin_init_workspace
$ catkin_create_pkg sensor_pkg roscpp sensor_msgs
$ mkdir -p sensor_pkg/launch
$ cp ~/a-b/src/trans_sensor.cpp sensor_pkg/src/
$ cp ~/a-b/launch/mysensor.launch sensor_pkg/launch/ ←
$ cp ~/a-b/launch/mysetup.launch sensor_pkg/launch/ ←
$ gedit sensor_pkg/CMakeLists.txt
$ cd ..
$ catkin_make
```

実データ用

仮データ(rosbag)用

CMakeLists.txtの末尾に追記

```
add_executable(trans_sensor src/trans_sensor.cpp)
target_link_libraries(trans_sensor ${catkin_LIBRARIES})
```

コントローラ接続準備

Autowareで扱えるようにトピック名をremap

```
$ cd /opt/ros/melodic/share/gazebo_ros/launch
$ sudo gedit empty_world.launch
```

empty_world.launchを以下のように変更

```
46 <node name="gazebo" pkg="gazebo_ros"
      type="$(arg script_type)" respawn="$(arg respawn_gazebo)"
      output="$(arg output)"
47 args="$(arg command_arg1) $(arg command_arg2)
      $(arg command_arg3) -e $(arg physics) $(arg extra_gazebo_args)
      $(arg world_name)"
48 required="$(arg server_required)" > /を削除
49 <remap from="/odom" to="/vehicle/odom" /> 追記
50 </node>
```

移動命令関連準備

1. ゲストPCでsim_pkgを作成
2. ホストPCでsrcにソースコード保存
3. ホストPCでCMakeLists.txtを修正
4. ゲストPCでノードビルト

- catkin_make

ゲストPCの起動

```
$ cd ~/shared_dir/autoware/docker/generic
$ ./run.sh -r melodic -t 1.14.0
```

移動命令関連準備コマンド

```
$ mkdir -p ~/shared_dir/autoware_docker_ws/src
```

```
$ cd ~/shared_dir/autoware_docker_ws/src
```

```
$ catkin_init_workspace
```

ゲストPC

```
$ catkin_create_pkg sim_pkg roscpp sensor_msgs \
geometry_msgs autoware_msgs
```

```
$ cd ~/shared_dir/autoware_docker_ws/src/
```

```
$ cp ~/a-b/src/trans_cmd.cpp sim_pkg/src/
```

ホストPC

```
$ gedit sim_pkg/CMakeLists.txt
```

CMakeLists.txtの末尾に追記

```
add_executable(trans_cmd src/trans_cmd.cpp)
```

```
target_link_libraries(trans_cmd ${catkin_LIBRARIES})
```

```
$ cd ..
```

```
$ catkin_make
```

ゲストPC

SLAMデモ

Autowareを使用した三次元地図作成



三次元地図作成手順

16

1. 地図を作成したい空間で
ロボットを移動
→bagファイルを保存



2. 保存したbagファイルを使用し
て地図を作成

パラメータを変えて何度も地図を
作成しなおせるようにbagファイルを使用



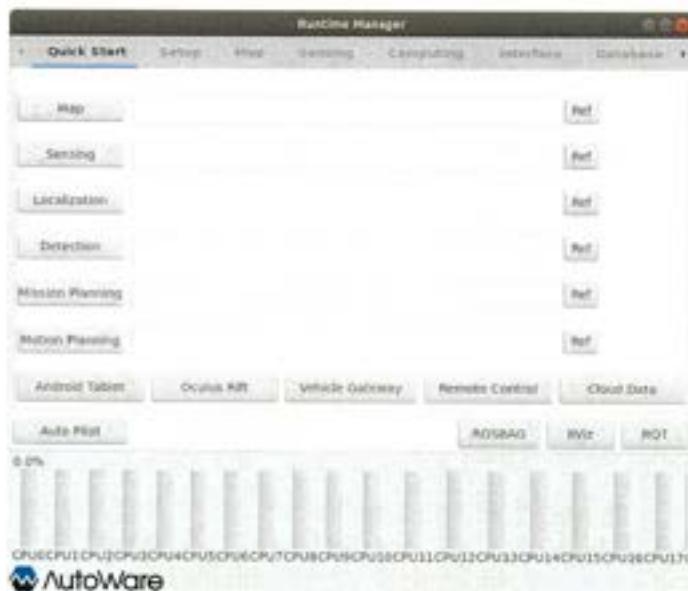
1. bagファイル保存

- ランタイムマネージャー起動
- センサ接続
- オドメトリ取得開始
- bagファイル保存開始
- ロボット移動
- bagファイル保存終了



a. ランタイムマネージャー起動

```
$ cd ~/shared_dir/autoware/docker/generic ] ホストPC
$ ./run.sh -r melodic -t 1.14.0
$ roslaunch runtime_manager runtime_manager.launch ] ゲストPC
```



b. センサ接続 (コマンド)

```
$ cd ~/shared_dir/sensor_ws
$ source devel/setup.bash
$ roslaunch sensor_pkg mysensor.launch
```

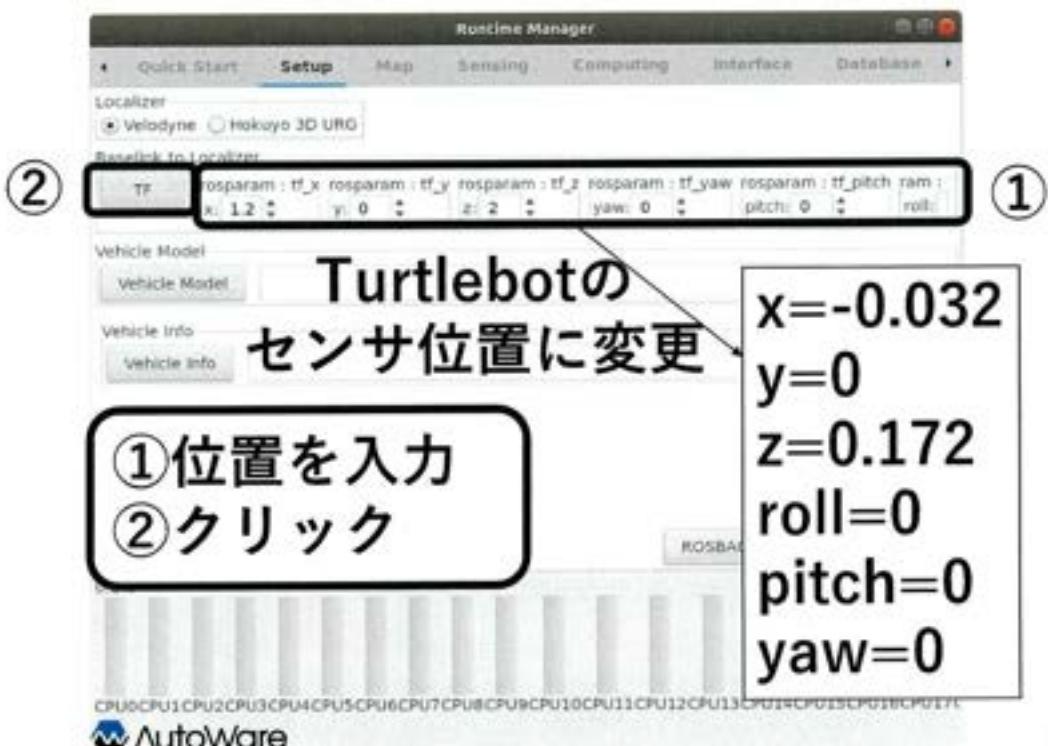
mysensor.launch:

センサ座標系の名前を変えて接続ノードを起動

```
1 <launch>
2   <node pkg="tf" type="static_transform_publisher"
3     name="velodyne_to_localizer"
4     args="0 0 0 0 0 /velodyne/base_scan 10" />
5     <node name="trans_sensor" pkg="sensor_pkg"
6       type="trans_sensor" output="screen" />
7 </launch>
```

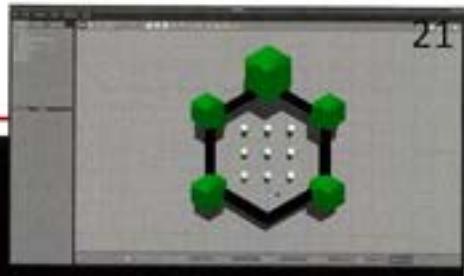
Turtlebotのセンサ座標名base_scanを
Autowareのセンサ座標名velodyneに変更

b. センサ接続 (Setupタブ)



c. オドメトリ取得開始

```
$ cd ~/shared_dir/turtlebot_ws  
$ source devel/setup.bash  
$ export TURTLEBOT3_MODEL=burger  
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```



21

別ターミナルで手順eの準備

```
$ cd ~/shared_dir/turtlebot_ws  
$ source devel/setup.bash  
$ export TURTLEBOT3_MODEL=burger  
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

*bagファイル保存開始までロボットを動かさない



d~f. bagファイル保存開始/終了

22

Tasaki Lab.



大まかな自己位置/vehicle/odomと
三次元点群/points_rawを同期保存する

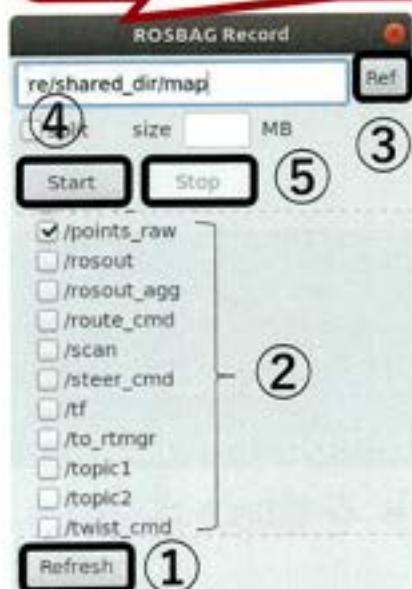


d~f. bagファイル保存開始/終了

Tasaki Lab.

共有ディレクトリ内に
map.bagという名前で保存

まずは低速直進で
柱三つ半ほど



- ①全トピック表示
- ②保存トピック選択
(/points_rawと
/vehicle/odomを✓)
- ③保存位置と名前を指定
- ④保存開始→Turtlebot移動
- ⑤Turtlebot移動後に終了

bagファイルが作成できたらいったんすべて終了

2. 地図作成

24

Tasaki Lab.

- a. bagファイルの再生と一時停止
- b. センサ位置を設定
- c. 地図作成ノード起動
- d. bagファイル再再生
- e. 地図を保存

準備：ランタイムマネージャー再起動

```
$ rosrun runtime_manager runtime_manager.launch
```

a. bagファイル再生と一時停止



Autoware



b. センサ位置設定（コマンド）

```
$ cd ~/shared_dir/sensor_ws
$ source devel/setup.bash
$ roslaunch sensor_pkg mysetup.launch
```

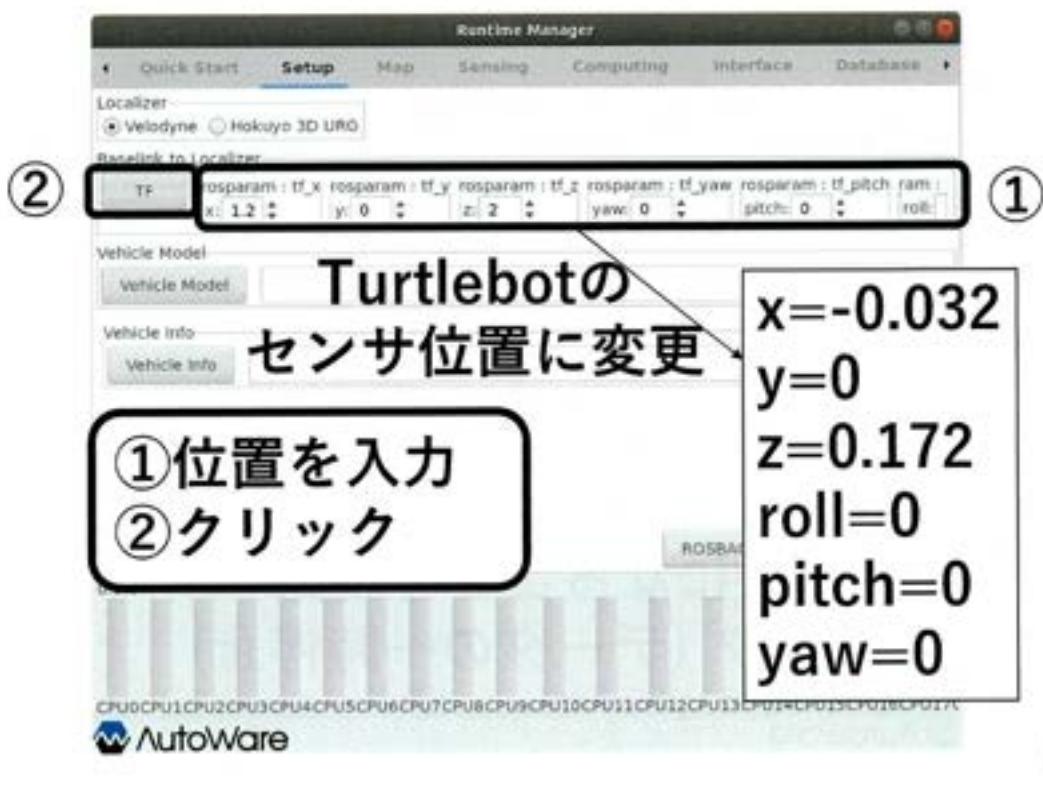
mysetup.launch

```
1 <launch>
2   <node pkg="tf" type="static_transform_publisher"
3     name="velodyne_to_localizer"
4     args="0 0 0 0 /velodyne/base_scan 10" />
5 </launch>
```

bagファイルからセンサデータを取得するので
trans_sensorを起動しない



b. センサ位置設定 (Setupタブ)



c. 地図作成ノード起動 (app)

Computing/L/Localization/lidar_localizer/ndt_mapping



注意：パラメータ設定について

Tasaki Lab.

センシング距離などのパラメータ
→センサ設定から一意に決まる

一意に決まらないパラメータは
パラメータの意味を考えロボットサイズや
移動する環境の規模をベースに決定する

Autowareの初期パラメータ=自動車用
Turtlebot3は自動車の1/10程度の大きさ



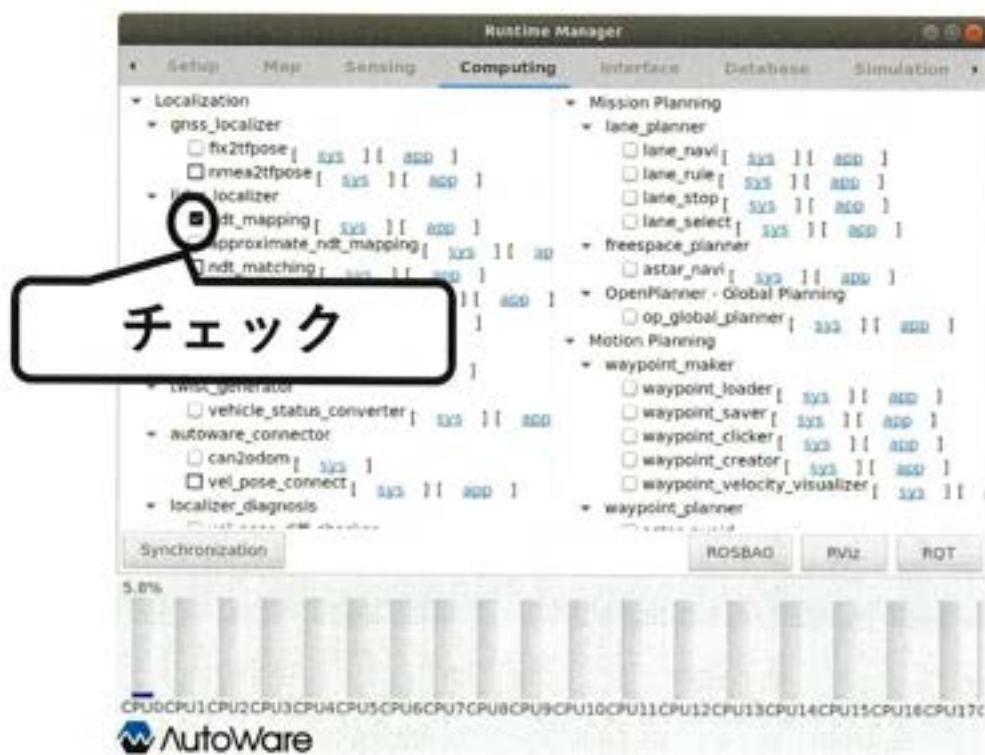
Autowareの初期パラメータの1/10を目安に
してうまくいかなければ少しづつ値を変える



c. 地図作成ノード起動 (✓)

Tasaki Lab.

Computing/L/Localization/lidar_localizer/ndt_mapping



d. bagファイル再再生



地図作成中の画面

```
/home/autoware/Autoware/install/points_downsampler/share/points_downsampler/launch... ● ◻ ◌
```

File Edit View Search Terminal Help

```
(Processed/Input): (64 / 64)
Failed to find match for field 'intensity'.
Failed to find match for field 'intensity'.

Sequence number: 531
Number of scan points: 360 points.
Number of filtered scan points: 160 points.
transformed_scan_ptr: 360 points.
map: 360 points.
NDT has converged: 1
Fitness score: 0.022776
Number of iteration: 2
(x,y,z,roll,pitch,yaw):
(0.352255, 0.0178452, 0, -5.90673e-17, 1.3294e-16, -0.00549513)
Transformation Matrix:
0.999985 0.0054961 1.92362e-16 0.320256
-0.0054961 0.999985 5.80109e-17 0.0180211
-1.9204e-16 -5.90673e-17 1 0.172
0 0 0 1
shift: 0.352707

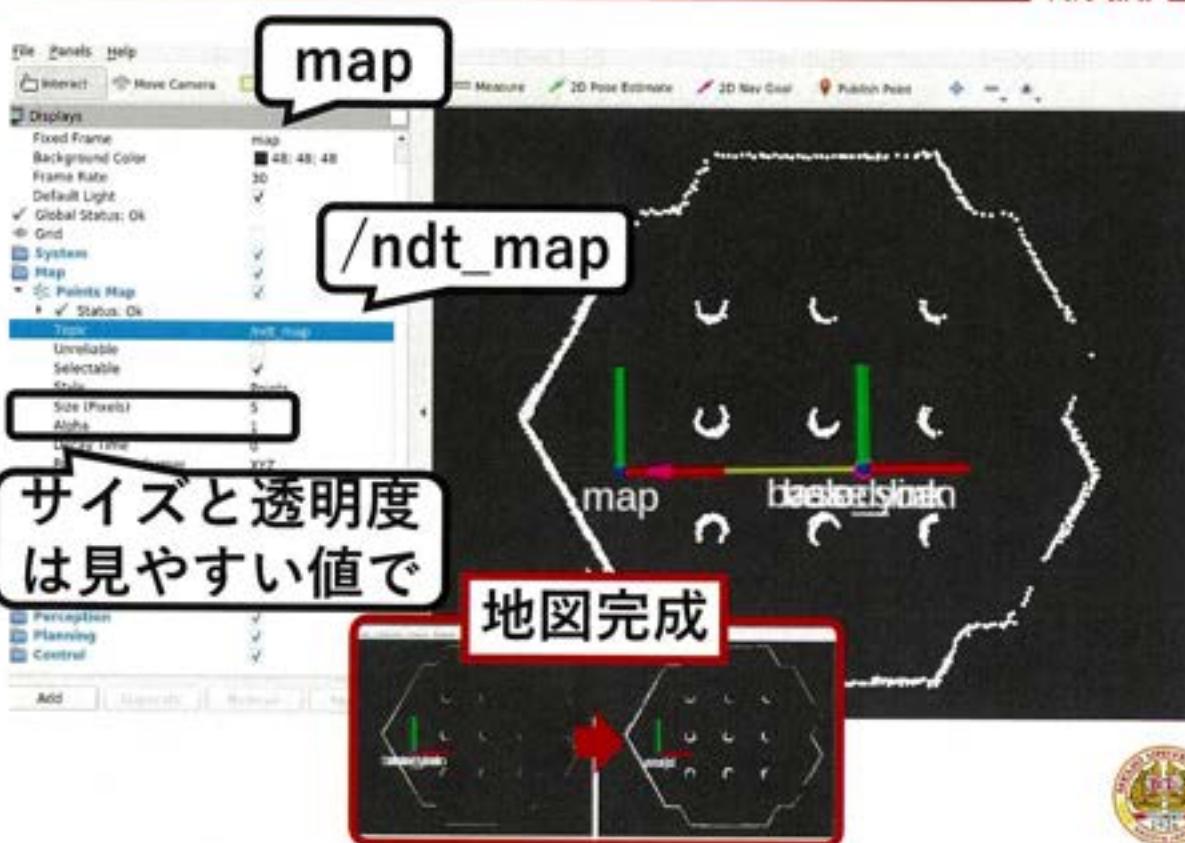
(Processed/Input): (65 / 65)
```

自己位置姿勢



参考：地図作成過程の表示

Tasaki Lab.



e. 地図を保存(app)

Computing/L/Localization/lidar_localizer/ndt_mapping

Tasaki Lab.



自律移動デモ

静的環境自律移動



静的環境での自動運転手順

36

Tasaki Lab.

■準備

完了

- 地図作成、参照経路作成

表計算ソフト
で作成

■手順

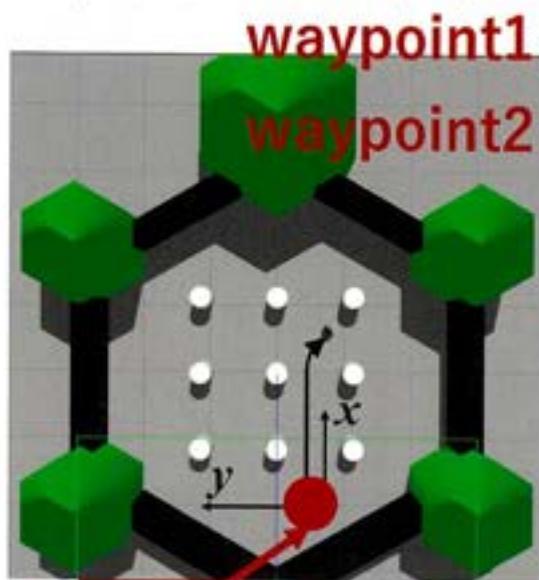
- センサ接続
- 自己位置推定
- 経路生成
- 経路追従

bagファイル
ではなくなる
パラメータ変更
するだけ

基本的な手順はAutoware起動練習と変わらない

参照経路

=単なるcsvファイル



saved_waypoints.csv

| | X | y | z | yaw | velocity | change_flag |
|----|--------|---------|---|---------|----------|-------------|
| 2 | 0.0067 | -0.0089 | 0 | 0.015 | 0 | 0 |
| 3 | 0.2081 | -0.0007 | 0 | 0.0017 | 0.0598 | 0 |
| 4 | 0.4137 | 0.012 | 0 | 0.0015 | 0.144 | 0 |
| 5 | 0.6162 | 0.0225 | 0 | 0.0061 | 0.241 | 0 |
| 6 | 0.822 | 0.01 | 0 | 0.0096 | 0.1797 | 0 |
| 7 | 1.0233 | 0.0248 | 0 | 0.0105 | 0.2301 | 0 |
| 8 | 1.2239 | 0.0359 | 0 | 0.01 | 0.2162 | 0 |
| 9 | 1.4307 | 0.0345 | 0 | 0.0162 | 0.1692 | 0 |
| 10 | 1.6308 | 0.0283 | 0 | 0.0218 | 0.1823 | 0 |
| 11 | 1.8308 | 0.0322 | 0 | 0.0268 | 0.3213 | 0 |
| 12 | 2.0408 | 0.0416 | 0 | 0.0331 | 0.2939 | 0 |
| 13 | 2.2522 | 0.0163 | 0 | -0.3386 | 0.3656 | 0 |
| 14 | 2.4285 | -0.0948 | 0 | -0.7842 | 0.2149 | 0 |
| 15 | 2.5561 | -0.2543 | 0 | -0.9475 | 0.1343 | 0 |

原点

位置[m]

姿勢[rad]

速度[km/h]

準備その1(ファイルコピー)

- ①参照経路を共有ディレクトリにコピー

```
$ cd ~/shared_dir
$ cp ~/a-b/other/saved_waypoints.csv ./
```

- ②mytf.launchをコピー

```
$ cp ~/a-b/launch/mytf.launch ./
```

```
1 <launch>
2   <node pkg="tf" type="static_transform_publisher"
3     name="world_to_map"
4     args="0 0 0 0 0 /world /map 10" />
5 </launch>
```

狭い世界なので
地図座標系と世界座標系
を同じにする



準備その2 (Turtlebot接続準備)

ランタイムマネージャーと子ターミナルを起動

```
$ cd ~/shared_dir/autoware/docker/generic
$ ./run.sh -r melodic -t 1.14.0
$ gnome-terminal &
$ rosrun runtime_manager runtime_manager.launch
```

子ターミナル：ランタイムマネージャー起動後
ゲストOSでランタイムマネージャー以外
(trans_cmd)を起動するために使用

```
$ cd ~/shared_dir/autoware_docker_ws
$ source devel/setup.bash
$ rosrun sim_pkg trans_cmd
```

子ターミナルで実行

準備その3 (Turtlebot起動)

```
$ cd ~/shared_dir/turtlebot_ws
$ source devel/setup.bash
$ export TURTLEBOT3_MODEL=burger
$ rosrun turtlebot3_gazebo turtlebot3_world.launch
```



1. センサ接続 (コマンド)

再掲

41

Tasaki Lab.

```
$ cd ~/shared_dir/sensor_ws  
$ source devel/setup.bash  
$ roslaunch sensor_pkg mysensor.launch  
mysensor.launch:
```

センサ座標系の名前を変えて接続ノードを起動

```
1 <launch>  
2   <node pkg="tf" type="static_transform_publisher"  
3     name="velodyne_to_localizer"  
4     args="0 0 0 0 /velodyne/base_scan 10" />  
5   <node name="trans_sensor" pkg="sensor_pkg"  
6     type="trans_sensor" output="screen" />  
7 </launch>
```

Turtlebotのセンサ座標名base_scanを
Autowareのセンサ座標名velodyneに変更

1. センサ接続 (Setupタブ)

再掲

42

Tasaki Lab.



2. 自己位置推定起動

自己位置推定起動手順

44

Tasaki Lab.

1. 地図読み込みと座標設定
2. 自己位置推定関連ROSノードを起動

自己位置推定関連ROSノード

| 起動名 | タブ | 役割 |
|-------------------|-----------|----------------|
| voxel_grid_filter | Sensing | 三次元点群フィルタリング |
| nmea2tfpose | Computing | GNSSのデータを取得 |
| ndt_matching | Computing | 自己位置推定 |
| vel_pose_connect | Computing | 自己位置推定トピック名を変更 |

nmea2tfposeは屋内移動ロボットでは不要



地図読み込み/座標変換 (Mapタブ)



- ①クリック (map.pcdを選択)
- ②クリック (地図読み込み)
- ③クリック (mytf.launchを選択)
- ④クリック (地図座標を設定)

三次元点フィルタリング (Sensingタブ)⁴⁶



- ①appクリック (voxel_grid_filter設定画面表示)
- ②パラメータ変更 (地図作成時と同様)
- ③OKクリック
- ④チェック (voxel grid filter起動)

自己位置推定起動 (Computingタブ)



3. 経路生成起動

静的の環境

経路生成ノードの起動

Tasaki Lab.

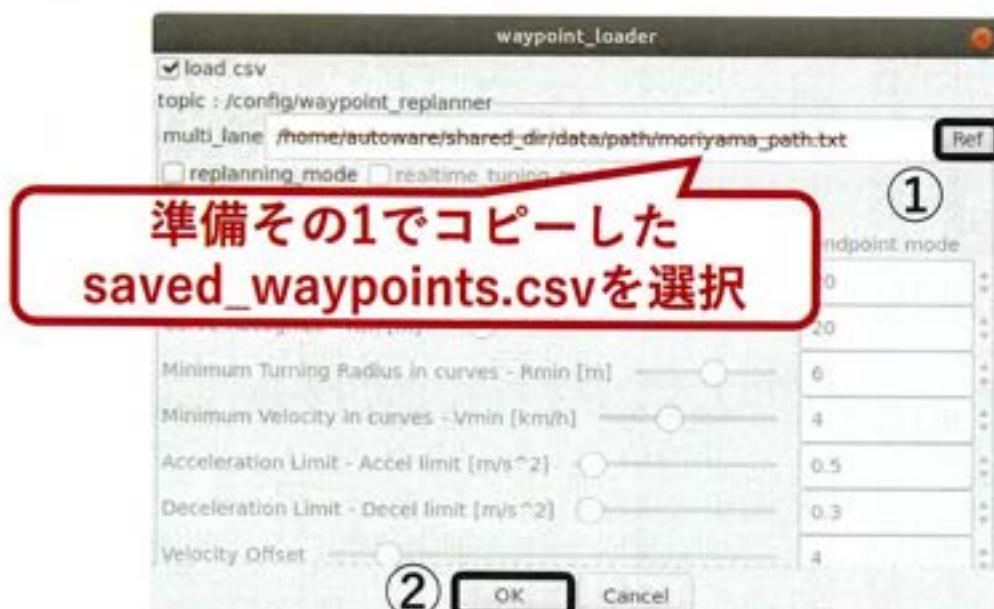
waypoint_loaderは起動 ✓ 前に設定必要

| ノード名 | タブ | 役割 |
|-----------------|-----------|-------------------------|
| lane_rule | Computing | 参照経路を調整する。 |
| lane_select | Computing | 現在位置に最も近いwaypointを選択する。 |
| waypoint_loader | Computing | 参照経路を読み込む。 |
| astar_avoid | Computing | 経路を生成する。 |
| velocity_set | Computing | 参照経路と障害物との衝突判定をする。 |

| ノード名 | 位置 |
|-----------------|--|
| lane_rule | Computing/R/Mission Planning/lane_planner |
| lane_select | Computing/R/Mission Planning/lane_planner |
| waypoint_loader | Computing/R/Motion Planning/waypoint_marker |
| astar_avoid | Computing/R/Motion Planning/waypoint_planner |
| velocity_set | Computing/R/Motion Planning/waypoint_planner |

waypoint_loaderのパラメータ

Tasaki Lab.



準備その1でコピーした
saved_waypoints.csvを選択

①クリック(経路ファイル選択)

②クリック(閉じる)



4. 経路追従起動

52

経路追従ノードの起動

Tasaki Lab.

pure_pursuitは起動 ✓ 前に設定必要

| ノード名 | タブ | 役割 |
|--------------|-----------|-----------------------|
| pure_pursuit | Computing | 経路追従する。 |
| twist_filter | Computing | 追従時の異常な並進速度、角速度を抑制する。 |

| ノード名 | 位置 |
|--------------|---|
| pure_pursuit | Computing/R/Motion Planning/waypoint_follower |
| twist_filter | Computing/R/Motion Planning/waypoint_follower |



pure_pursuitのパラメータ

| パラメータ | 役割 |
|----------------------------|---|
| lookahead_ratio | 速度に応じて waypoint を選ぶ際の係数を設定する。 |
| minimum_lookahead_distance | waypoint を選ぶ最小距離[m]を設定する。 0.2に設定 |

minimum_lookahead_distanceの設定:

- ・経路間隔に合わせて0.2mを設定
→経路間隔はAutoware標準の1mの1/10だと
ロボットサイズ以下になって細かすぎたため
ロボットサイズ程度まで大きくした



自律移動開始 (Interfaceタブ)



安全のためtrans_cmd.cpp(17~24行目)で
クリックされるまで停止命令を出している



■実施できるようになったこと

- ROS対応ロボットを静的環境で自律移動させる

■本セミナーで扱えなかったこと

- 参照経路上に障害物が存在する環境での自律移動
→ Hybrid A*などの手法は説明済みなので挑戦してもらいたい

