筑波大学 知能ロボット研究室
University of Tsukuba, Intelligent Robot Laboratory

障害物の検知、停止（回避）

坂田 崚真

山彦セミナー　2023-05-08

1 本時の目標

- 目の前にある障害物を検知して停止する

2 復習

- これまででできるようになったこと
  - 任意の箇所で停止、直線に沿って走行（第4回）
  - 物体の検知（第11回）
  - Waypointに沿って走行（第12回）

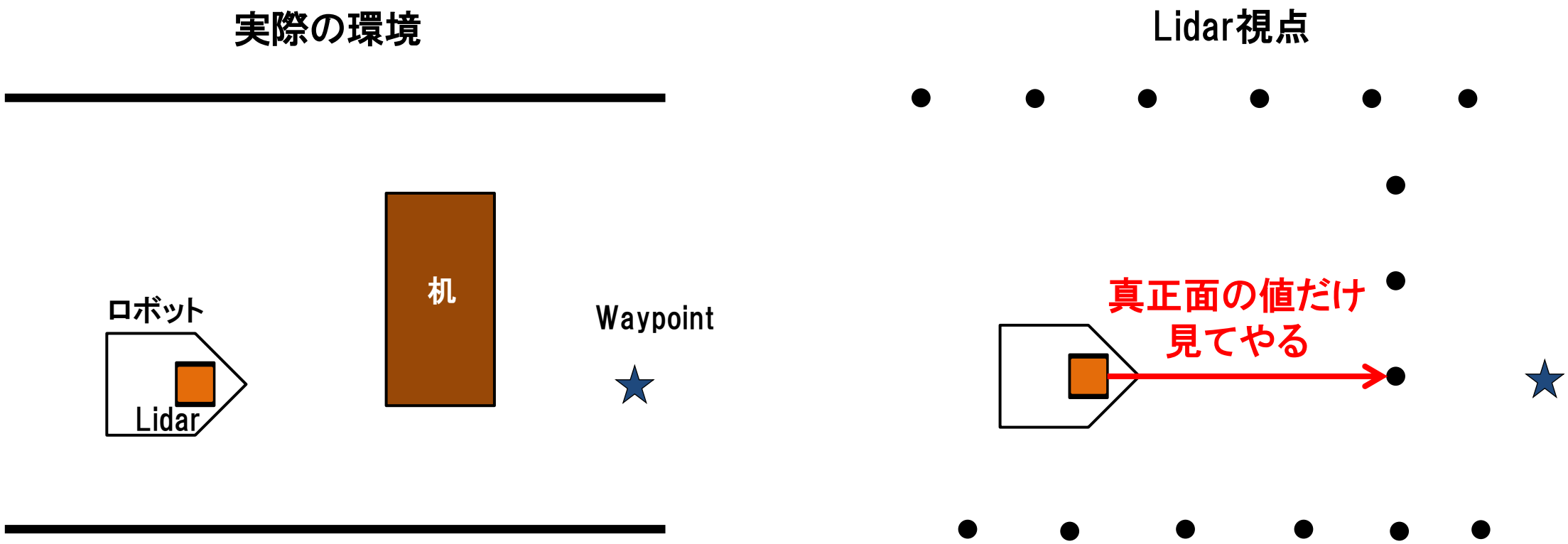
⇨ 理想的な（走行経路上に何も障害となるものが存在しない）状況では自律移動が可能！

- 実環境（つくばチャレンジ等）ではどうか
  - 目の前に人や花壇等が存在する

⇨ ロボットの走行上邪魔となる物体（障害物）を検知し、対処する必要あり

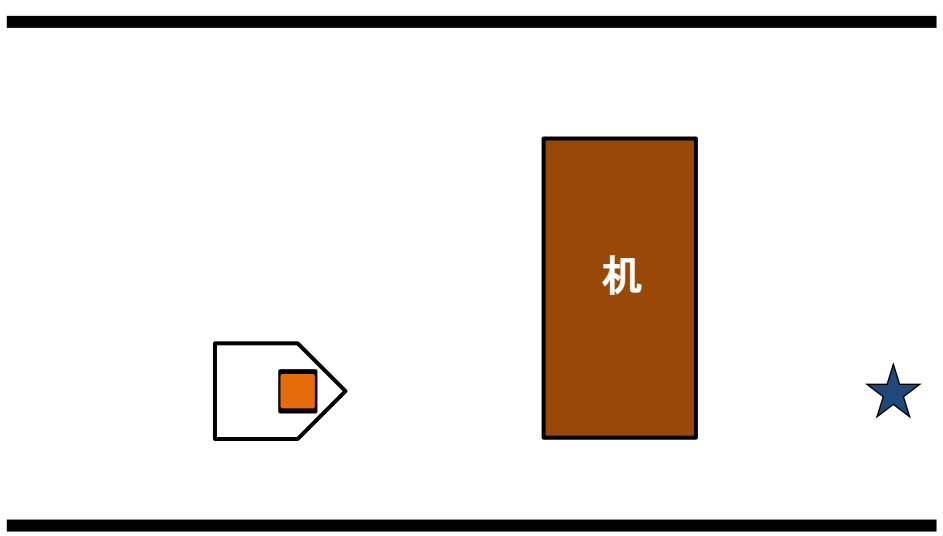
3 一番原始的な考え方

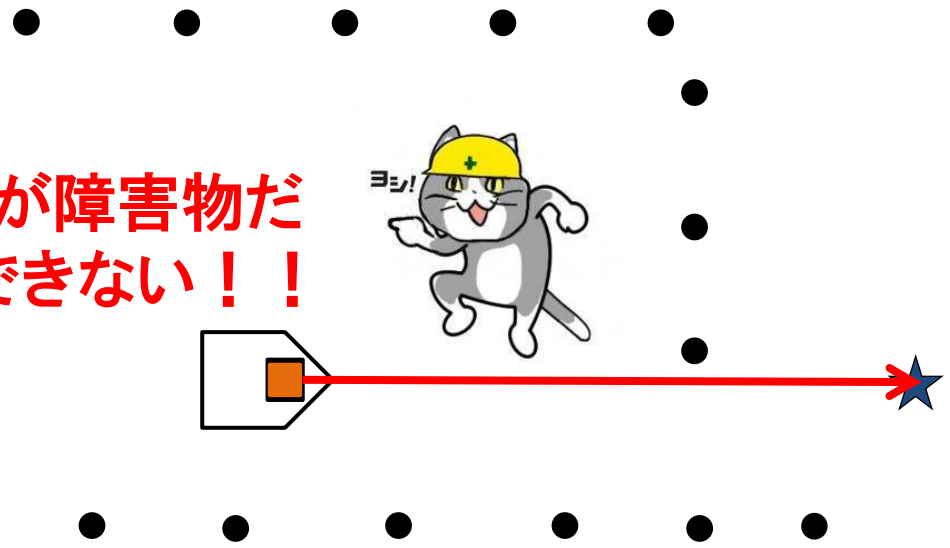
- 目の前に障害物がある
  - ロボットの正面の点だけ取り出してその点がある閾値(一定の値)を超えたら停止



4 一番原始的な考え方の問題点

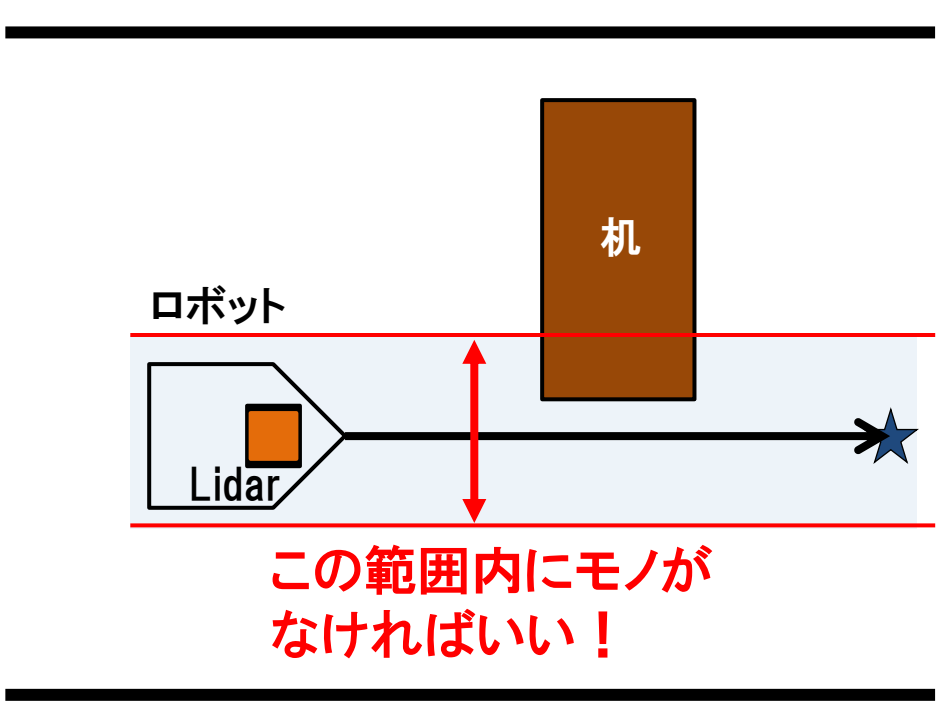
- ちょっとでも正面からずれている、一瞬外れ値をとってしまった時

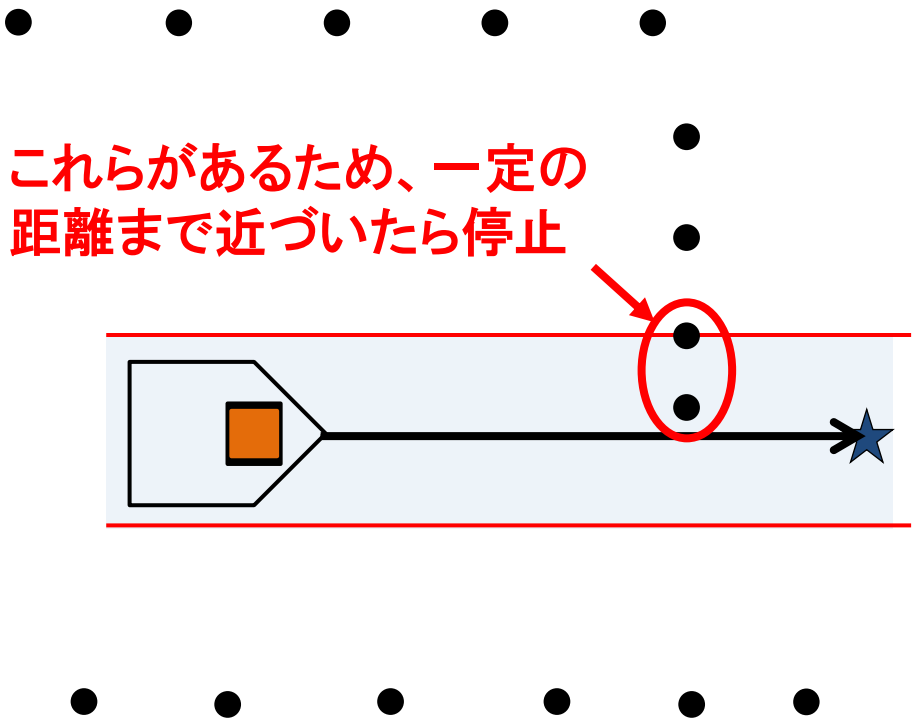




5 もう少し高尚な手法

- 走行経路上に障害物があると困る
  - 走行経路上からロボットの幅分の範囲かつ一定の距離以下の時に障害物があるときに停止





本当にそうしないとダメ？

- 2DLidarならせいぜい1000点だが、3DLidar(Velodyne VLP-16)になると300,000点
  - これらの点をリアルタイムですべて処理することは困難
  - ノイズに弱い(計測ミスが一点でもあり、それが走行経路上にあると判定した場合、停止してしまう)
- 1つ1つの点という認識ではなく点の集合(**点群**)として扱ってやる必要あり！
  - (PCLを用いた)ユークリッドクラスタリング
  - Obstacle_detector等の利用
- Obstacle_detector
  - こちらを参考に
  - /obstaclesのトピックに障害物の中心座標、障害物自体の半径、安全マージンを加味した障害物の半径が記述されている

Obstacle_detectorの使い方

- C++での実装
  - 移動するプログラムがあるパッケージ内のCmakelistに以下を追加

```
cmake_minimum_required(VERSION 3.0.2)
project(my_package)

find_package(catkin REQUIRED COMPONENTS
  roscpp
  obstacle_detector
)
include_directories(
  ${catkin_INCLUDE_DIRS}
  ${obstacle_detector_INCLUDE_DIRS}
)
add_executable(my_node src/my_node.cpp)
target_link_libraries(my_node
  ${catkin_LIBRARIES}
  ${obstacle_detector_LIBRARIES}
)
```

Obstacle_detectorの使い方

- 障害物検知をしたいプログラム内に以下のヘッダーファイルを追記

```
#include "obstacle_detector/obstacle_publisher.h"
```

- コールバック関数(main内でのsubscriberの記述は各自よろしく)

```
void obstacle_callback(const obstacle_detector::Obstacles::ConstPtr obs)
{
        //障害物の個数
        int size_num = obs->circles.size();
        for (int i = 0; i < size_num; i++)
        {
                //Lidarからみた各障害物の位置と半径
                obstacle_x[i] = obs->circles[i].center.x;
                obstacle_y[i] = obs->circles[i].center.y;
                obstacle_radius[i] = obs->circles[i].radius;

                //グローバル座標系での各障害物の位置
                global_obs_x[i] = obstacle_x[i] * cos(yaw) - obstacle_y[i] * sin(yaw) + robot_x;
                global_obs_y[i] = obstacle_x[i] * sin(yaw) + obstacle_y[i] * cos(yaw) + robot_y;
}
```

Obstacle_detectorの使い方

- roslaunch obstacle_detector input_scan.launch
- roslaunch <your pkg name> < your launchfile name>

より一般的な手法(経路計画)

- こちらのサイト参照。以下一部抜粋

- Geometric Analytic Approach
  - 幾何学的な関係を用いてロボットのコースを生成するアルゴリズム(Spline Plannning等)
- Graph Search Approach
  - 走行箇所をグラフやグリッドとして表現し、グラフ理論のアルゴリズム経路探索を行う(A*等)
- Dynamic Window Approach
  - 運動モデルを用いて、考えうる範囲を計算し、その時刻の中で最もコストが低い経路を選択する
- Randomized Approach
  - ランダムにサンプリングした点を元に、経路を徐々に探索していく経路計画アルゴリズム(RRT等)
- Model Predictive Control
  - ロボットのモデルと最適化技術を使って最適な各時刻の入力と軌跡を計算する経路計画手法

小技

- モータードライバー、Lidarを同時に立ち上げると、起動時の状況次第で順番が異なる場合がある(ttyACM0と1が固定されていないため)

- 以下の手順でttyACM0、1によらず固定で起動できる
  - 一度すべて起動した状態で以下のコマンドを打つ(第8回)

    ```
    ls -l /dev/serial/by-id/
    ```

  - その時に出てくる名前(usb-T-frog_project_T-frog_Driver-if00等)をlaunchファイルのportに直接書き込む

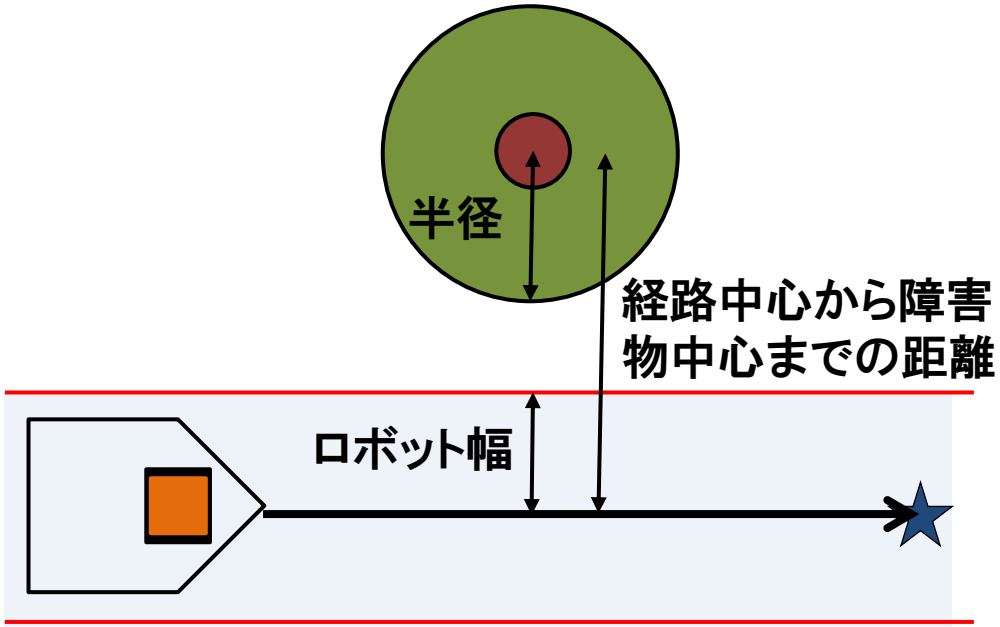
    ```
    <param name="port" value="/dev/serial/by-id/usb-T-frog_project_T-frog_Driver-if00"/>
    ```

課題

- スライド3で示したような手法でロボット正面に障害物がある状況下で、障害物の0.4m手前で停止してみよう
  - ヒント
    - 任意の場所で停止するプログラムをベースにして、停止する場所をLidar正面の座標で置き換えてやろう
    - Lidarの正面は何本目？
    - 点と点（グローバル座標系での障害物と自分の位置）の距離は？

オプション課題①

- スライド6で示したObstacle_detectorを用いて走行経路上に障害物がある状況下で、障害物の0.4m手前で停止してみよう
  - ヒント
    - 障害物の中心位置、障害物の幅を考慮し、各障害物が経路上に入っていなければいい
    - （経路中心と障害物中心の距離）-（障害物半径+ロボット幅/2）が負ならば経路上に障害物が存在



オプション課題②

- オプション課題①の状況で障害物を回避して任意の目的地まで自律走行させてみよう
  - 考えうる手法
    - 走行経路上に障害物が存在した場合、Waypointを障害物がいない領域に変更する
    - これまでの手法で一旦停止して、障害物が存在しない方向へ移動する(ウェイポイントを一つ先に送り、90°横を向いて前進し、もう一度方向転換をして前進)(昨年度のぱんじゃんはほぼこれ)