SEED Solutions
Smart RT System for Service Robots

NEDO講座
第３部
地図生成・ナビゲーション

THK株式会社
産業機器統括本部 技術本部
事業開発統括部 永塚BU

第３部の概要

地図の生成と地図を用いたナビゲーションを行う．
ここでは，シミュレーション環境で演習を行う．
PCのスペックによっては，処理が重いので，気長にまってください．

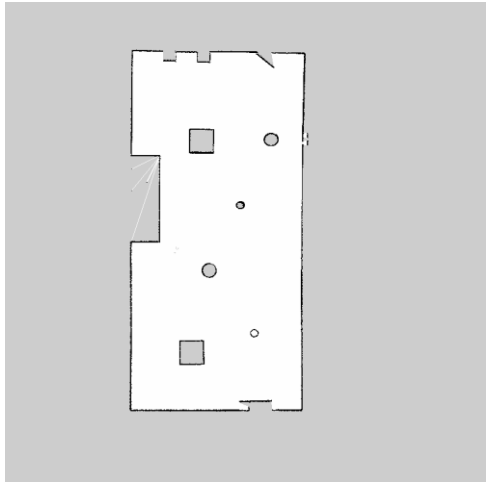
移動ロボットシステムの運用に関する
一連の流れを機能を体験することができます．

第３部ではゲームパッドの利用を推奨します．
ない場合はキーボードでも実施できます．
（ただし，操作になれが必要です．）

移動ロボットの運用

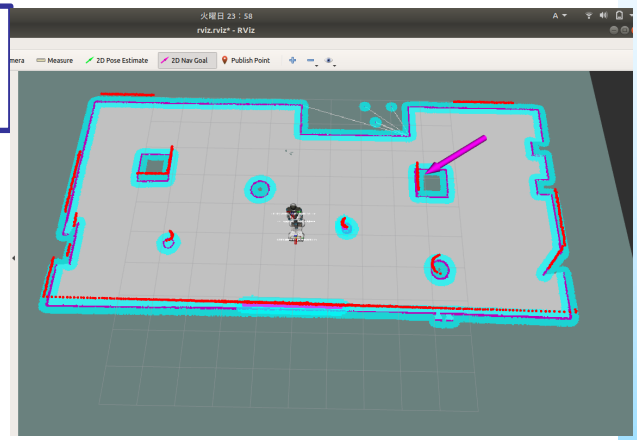
移動ロボットによる自律走行を実現するためには，以下の2つのステップを踏む必要がある．

地図生成



ロボットが移動する環境のデータを取得する．自己位置同定と地図生成を同時に行うSLAMが広く利用される．
(ROSではgmappingというアルゴリズムがよく利用される．)

自律走行



生成した地図データを用いて，初期位置から，目的地までの経路誘導を実施する．
(今回の講義では，Rviz上で行う．)

それぞれのステップは別々に独立して行われることが多く，
第３部では2つのステップに分けて実施する．

想定するゲームパッド

Dualshock 3

JC-U4113SBK

Dualshock3の場合は修正は必要ないですが，JC-U4113SBKを利用する場合は，configを書き換える必要があります．
それ以外の場合は，同じ場所にあるconfigファイルを適宜編集して合わせてください．
(付録にゲームパッドに合わせた設定確認方法を示します．)

環境地図の生成

Gazebo内の環境における地図の生成を行う．
起動するlaunchは2つ

- seed_r7_gazebo/seed_r7_example_world.launch

Gazebo内での環境やロボットのデータの読み込み

- seed_r7_navigation/wheel_with_making_map.launch

JoystickのノードやSLAM系のノード(gmappingなど)の起動

これら2つに加えて，地図の生成状況を確認するために，RVizを用いる．

全部で４つのターミナルを起動する．
（キーボードで行う場合は５つ起動します．）

Launchによるノードの起動

ターミナル１

```
roslaunch seed_r7_gazebo seed_r7_example_world.launch
```

Gazeboが起動するのに少し時間がかかるので，Mapが出てくるまで待つ．

ターミナル2

起動する段階で，JoystickはPCに接続しておくこと．

```
roslaunch seed_r7_navigation wheel_with_making_map.launch
```

ターミナル3

MAPの生成はGazeboではなく，rvizの画面を見ながら行う．

```
rosrun rviz rviz -d `rospack find seed_r7_bringup`/rviz.rviz
```

Gazeboが重たいですが，気長にお付き合いください．

キーボードを用いる場合

キーボードを用いる場合は，以下のようにキーボード入力を速度，回転角速度に変換するノードの起動を行います．

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

キーボードの操作は以下のようになる．（Holonomicの系なので，SHIFTキーを押しながらの操作になります．）

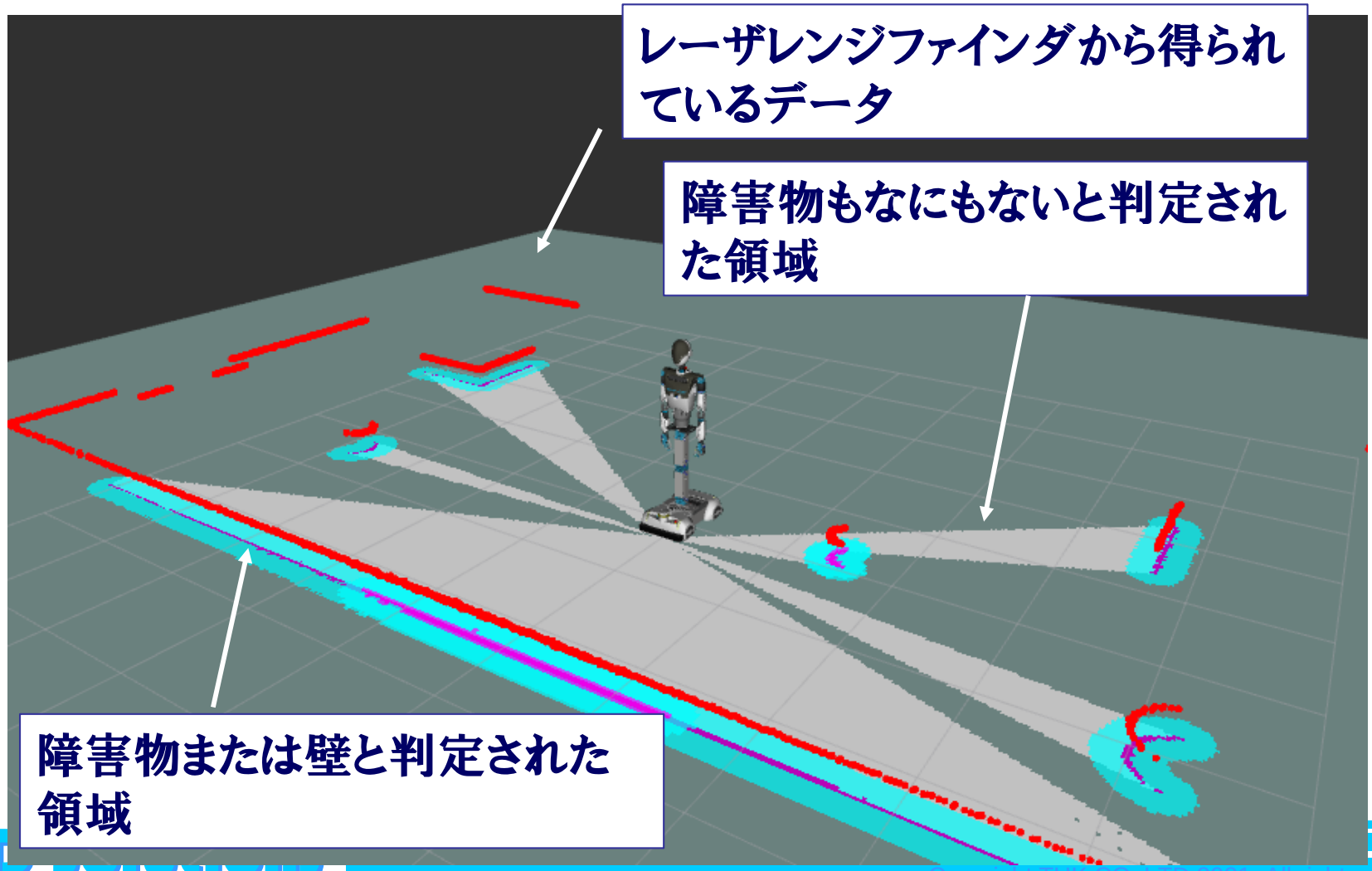
```
For Holonomic mode (strafing), hold down the shift key:
---------------------------
   U    I    O
   J    K    L
   M    <    >

t : up (+z)
b : down (-z)

anything else : stop

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%

CTRL-C to quit

currently:      speed 0.5       turn 1.0
```

上を進行方向とした場合のキーマッピング
「I」で前進，「<」で後進
「L」と「J」で左右の回転
キーを押すと動き続けるため，適宜「K」で停止をして下さい．

デフォルトだと早すぎるため，「z」で減速しておくと良い．

地図の生成

地図を作成する際，可能な限りMAP中を走行させるように心がける．
（MAP内の未知領域がないように走行させる．）
GazeboとRVizの両方を見るのが好ましいが，地図生成についてはRViz中心でもよい．



地図の保存

地図の生成が完了したら，新しいターミナルを起動し，MAP保存用のlaunchを実行する．

ターミナル4

```
roslaunch seed_r7_navigation map_saver.launch
```

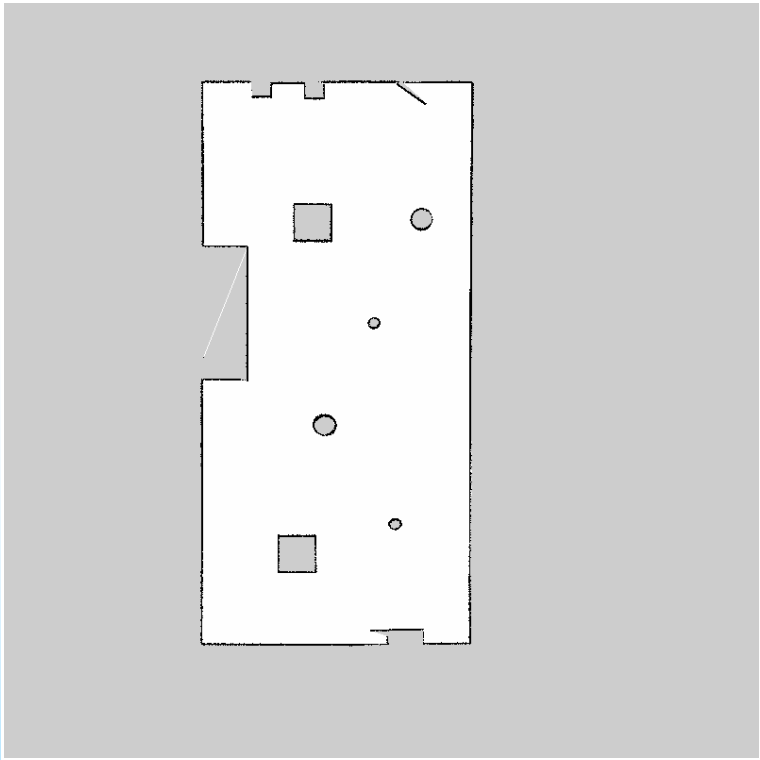
このlaunchを実行すると，下記の2つのファイルが生成・保存される．

- map.pgm(生成した地図の画像データ)
- map.yaml(画像データにおける縮尺や原点の情報を保管)

これらのファイルは以下に保存される．

```
~/ros/melodic/src/seed_r7_ros_pkg/seed_r7_navigation/maps/map.pgm
~/ros/melodic/src seed_r7_ros_pkg/seed_r7_navigation/maps/map.yaml
```

保存された地図データについて



- 黒い点は障害物
- 白い部分は障害物がない場所で通行可能領域
- 灰色の部分は未知領域

地図にノイズが入ったりしたときは，ペイントソフトなどで画像を修正する．

左の画像だけでは，実際の寸法や，原点の位置がわからないため，map.yamlにスケールや原点の情報が格納されている．

```
1 image: /home/seed/ros/melodic/src/seed_r7_ros_pkg/seed_r7_navigation/maps/map.pgm
2 resolution: 0.025000
3 origin: [-10.000000, -10.000000, 0.000000]
4 negate: 0
5 occupied_thresh: 0.65
6 free_thresh: 0.196
```

生成した地図を用いたナビゲーション

生成した地図を用いて，目的地までのナビゲーションを行う．
今回は，RViz上でナビゲーションを行う．
ナビゲーションを行うために，下記の3つのターミナルでlaunchを起動する．

ターミナル１

できれば起動したままで．
閉じてしまった場合，下記で再起動．

```
roslaunch seed_r7_gazebo seed_r7_example_world.launch
```

ターミナル2

自己位置同定とナビゲーションのためのノードを起動する．

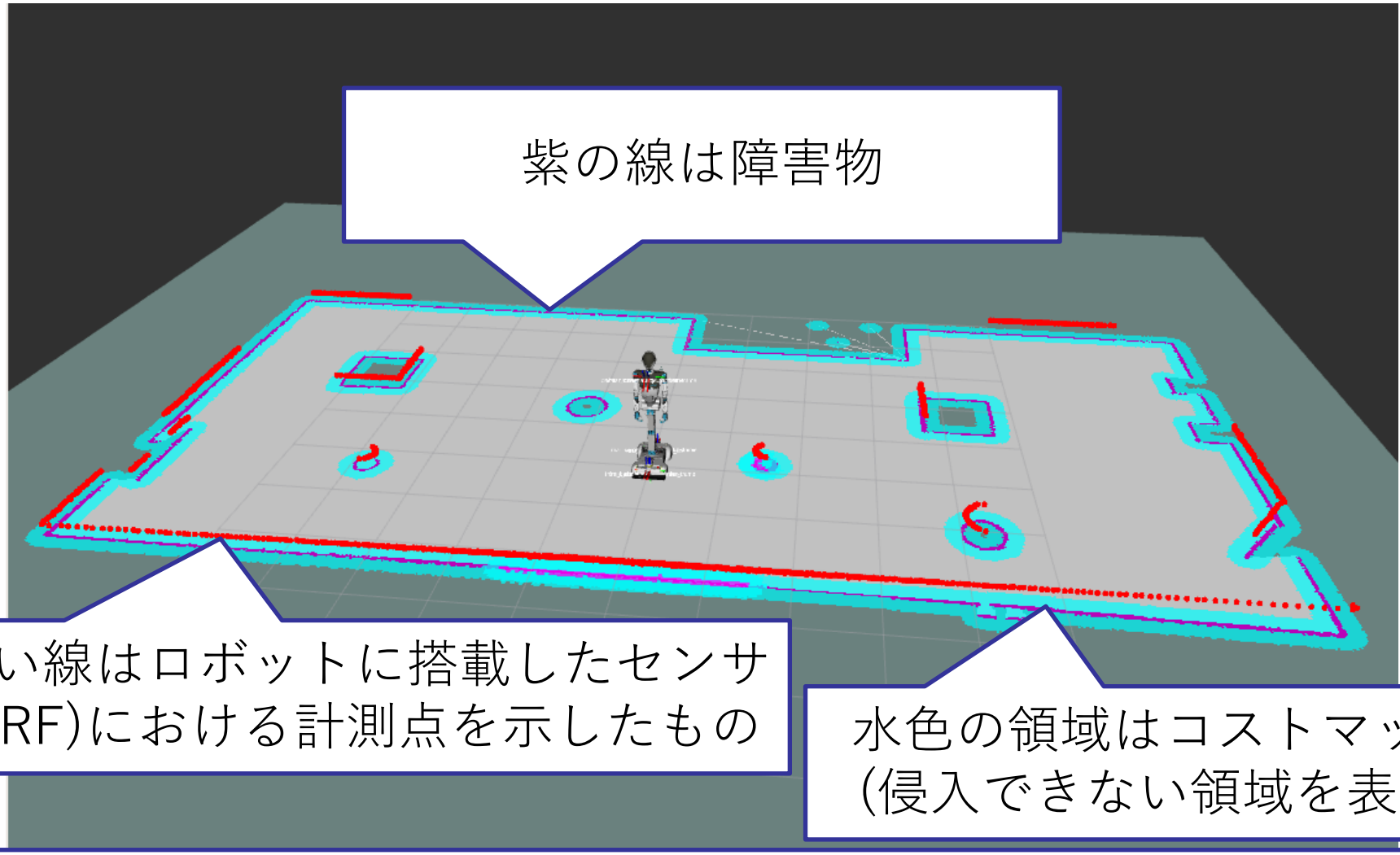
```
roslaunch seed_r7_navigation wheel_with_static_map.launch
```

ターミナル3

```
rosrun rviz rviz -d `rospack find seed_r7_bringup`/rviz.rviz
```

Rvizの表示の説明

rvizでは，以下の様に表示されているものに意味付けされている．



赤い線が紫の線と大体合っていることを確認すること
（自己位置がきちんと同定できていることを確認．）

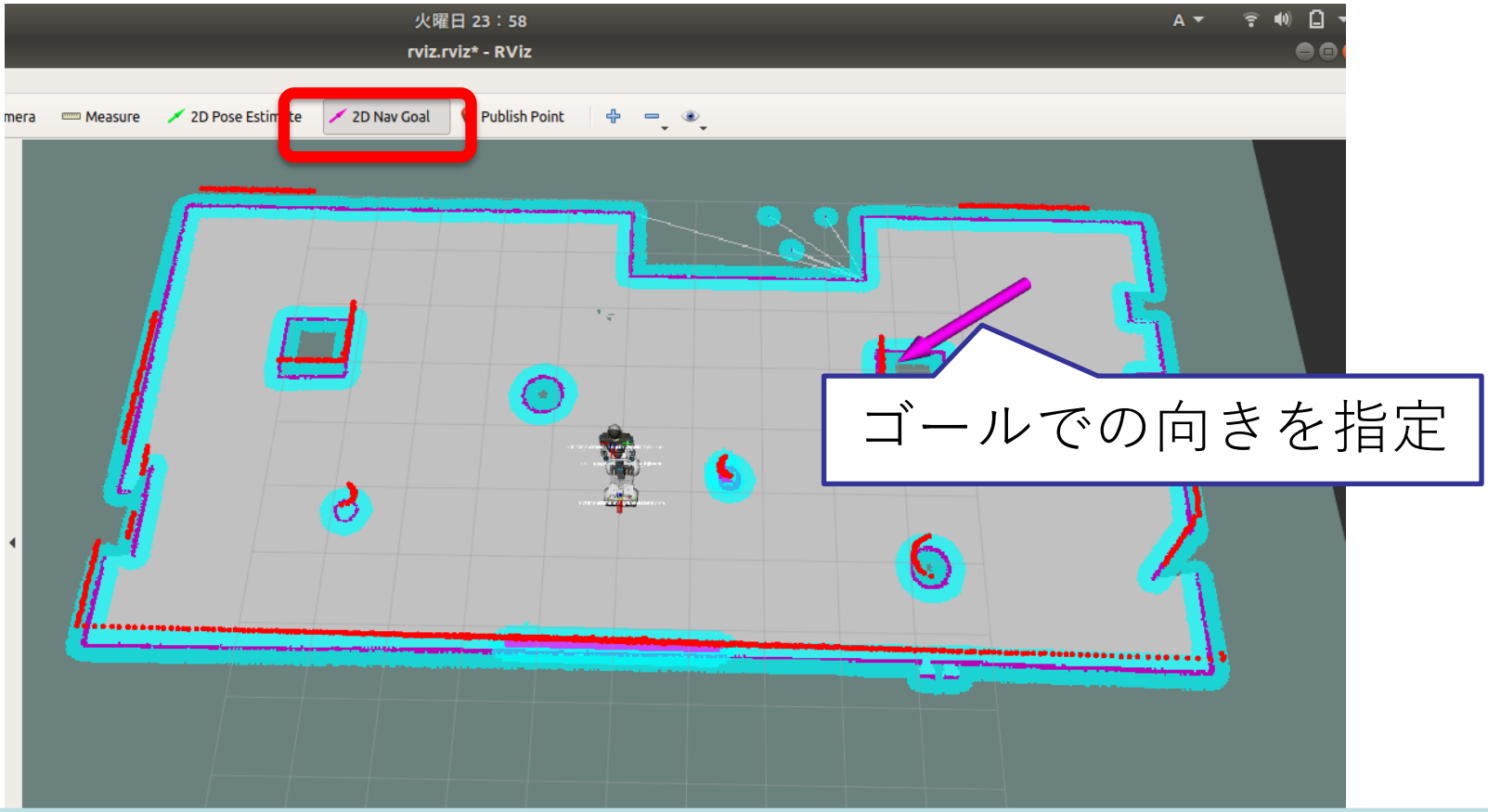
自己位置の修正

万が一自己位置がずれていたら，自己位置の修正を行う．
自己位置の修正では，位置と方向をマウスで操作しながら指定する．

「2D Pose Estimation」を選択して，位置と方向を修正する．

移動させたい位置をクリックすると矢印が出てくるので，クリックした状態を維持して，矢印の向きを真値に合わせて，離すと自己位置推定が行われる．

ナビゲーション



2D Nav Goalを選択肢，任意の目標位置をクリック．クリック状態を維持したまま出てきた矢印の方向を合わせ，マウスを放すと，経路計画・ナビゲーションが行われる．

Goal地点の記録

スクリプトを用いて自律走行させるための地点の取得のため，Goalについた状態で，以下のようにして，位置と姿勢を取得してください．

左のツリーから「RobotModel 」→「Links」→「base link」を選択し，PositionとOrientationをそれぞれメモする．値が小さい場合は0としてください．
（あまり下の桁までメモしなくても良いです．）

一旦すべてのノードを停止してください．
(Ctrl + Cを押すか，ターミナルを閉じるかをしてください．)

スクリプトを用いた自律走行

RVizを用いることで，任意の地点までナビゲーションできることを確認した.

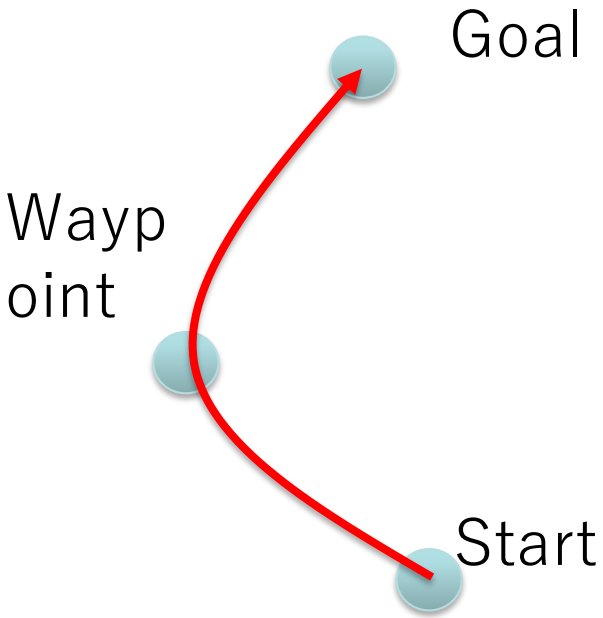
実際のアプリケーションでは複数の任意の地点を移動したり，決まった経路を移動させたりなど，アプリケーションに合わせて柔軟に運用したい.

スクリプトを書くことで
柔軟なナビゲーションを実現！

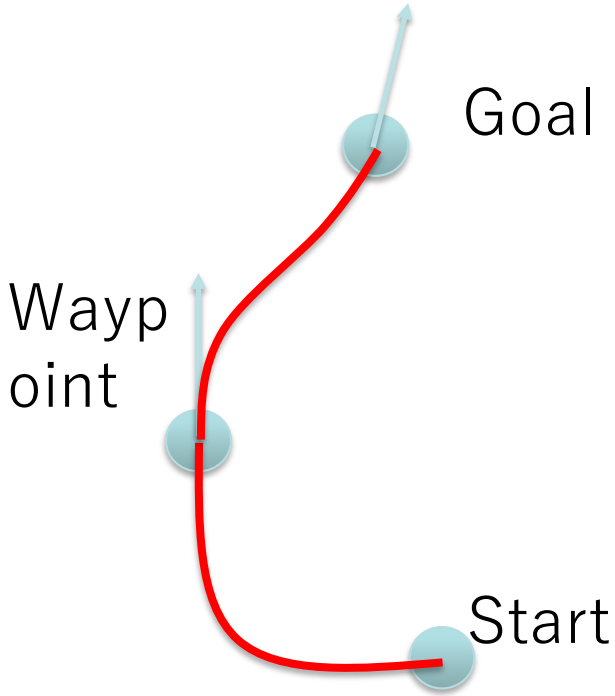
スクリプト作成のための基礎知識

経由点としてのWaypointを設定していく必要がある.

一般的に移動ロボットではWaypointとは経由点を示すが，ROSのnavigation stackのwaypointはショートゴールのイメージ



一般的な経由点のイメージ



ROSの経由点のイメージ

Waypointの決め方

広く行われている方法は，ゲームパッドなどで操作しながら，
　Waypoint候補の場所まで移動し，その位置・姿勢を記録.

Waypointをリストとして準備し，そのリストを読み込み，１つ１つの点をショートゴールと設定して，目的地まで移動させる.

実際に手順を踏んで確認してみましょう！

ナビゲーション用のスクリプトの作成

ナビゲーションを行うためのスクリプトを作成します.

スクリプトは以下に作成します.

```
~/ros/melodic/src/seed_r7_ros_pkg/seed_r7_navigation/scripts
```

以下のようにファイルを開いてコードを作成してください.

```
$cd ~/ros/melodic/src/seed_r7_ros_pkg/seed_r7_navigation/scripts
$gedit test_nav.py
```

ソースコードが完成したら以下のようにスクリプトに実行権限を与えてください.（スクリプトを作成した場所で実行します.）

```
$chmod +x test_nav.py
```

現在地から初期位置・姿勢への復帰

```python
#!/usr/bin/env python

import rospy
import actionlib

from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal

waypoints = [
    [(0.0, 0.0, 0.0), (0.0, 0.0, 0.0, 1.0)],
    [(-3.605, -3.653, 0), (0.0, 0.0, -0.734, 0.679)],
    [(0.0, 0.0, 0.0), (0.0, 0.0, 0.0, 1.0)]
]

def goal_pose(pose):
    goal_pose = MoveBaseGoal()
    goal_pose.target_pose.header.frame_id = 'map'
    goal_pose.target_pose.pose.position.x = pose[0][0]
    goal_pose.target_pose.pose.position.y = pose[0][1]
    goal_pose.target_pose.pose.position.z = pose[0][2]
    goal_pose.target_pose.pose.orientation.x = pose[1][0]
    goal_pose.target_pose.pose.orientation.y = pose[1][1]
    goal_pose.target_pose.pose.orientation.z = pose[1][2]
    goal_pose.target_pose.pose.orientation.w = pose[1][3]
    return goal_pose

if __name__ == '__main__':
    rospy.init_node('test_nav')

    client = actionlib.SimpleActionClient('move_base', MoveBaseAction)
    client.wait_for_server()

    for pose in waypoints:
        goal = goal_pose(pose)
        client.send_goal(goal)
        client.wait for result()
```

スタート地点から，メモした地点まで移動し，スタートまで戻ってくるスクリプト．

この例では，目標値を以下のようにしている．

(x, y, z)
＝(-3.605, -3.653，0)
(x, y, z, w)
=(0, 0, -0.734, 0.679)

次のスライドで詳細を説明しますので，まず作成してみて下さい．

スクリプト解説

```
1 #!/usr/bin/env python
2
3 import rospy
4 import actionlib
5
6 from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
7
8 waypoints = [
9     [(0.0, 0.0, 0.0), (0.0, 0.0, 0.0, 1.0)],
10    [(-3.605, -3.653, 0), (0.0, 0.0, -0.734, 0.679)]
11    [(0.0, 0.0, 0.0), (0.0, 0.0, 0.0, 1.0)]
12 ]
13
14 def goal_pose(pose):
15     goal_pose = MoveBaseGoal()
16     goal_pose.target_pose.header.frame_id = 'map'
17     goal_pose.target_pose.pose.position.x = pose[0][0]
18     goal_pose.target_pose.pose.position.y = pose[0][1]
19     goal_pose.target_pose.pose.position.z = pose[0][2]
20     goal_pose.target_pose.pose.orientation.x = pose[1][0]
21     goal_pose.target_pose.pose.orientation.y = pose[1][1]
22     goal_pose.target_pose.pose.orientation.z = pose[1][2]
23     goal_pose.target_pose.pose.orientation.w = pose[1][3]
24     return goal_pose
25
26 if __name__ == '__main__':
27     rospy.init_node('test_nav')
28
29     client = actionlib.SimpleActionClient('move_base', MoveBaseAction)
30     client.wait_for_server()
31
32     for pose in waypoints:
33         goal = goal_pose(pose)
34         client.send_goal(goal)
35         client.wait_for_result()
```

このプログラムは，すべての設定したwaypointへの移動が完了したら終了します．Waypointを追加すれば複数点移動できます．

２つの要素から構成されている
[(x,y,z), (x,y,z,w)]
前は位置，後ろは姿勢(四元数)

Waypointで指定した
位置・姿勢の組みを，送信用のデータにコピー

これはおまじない．
Actionを用いるよという宣言

Waypointに設定した位置・姿勢の組みの数だけ目標位置を送信．送信した目標位置に到達したら，次のWaypointを送信

Goal到達の判定基準

目標位置を(0,0,0)と指定しても，厳密にこの位置に到達することは難しいため，許容誤差が設定できる．

以下のファイルを開いて確認する．

~/ros/melodic/src/seed_r7_ros_pkg/seed_r7_navigation/config/TebLocalPlanner.yaml

以下の２つのパラメータが目標位置に対しての許容誤差を表している(単位は[m])．

```
# GoalTolerance
   xy_goal_tolerance: 0.03
   yaw_goal_tolerance: 0.05
```

これだけでなく，ナビゲーションのために設定するパラメータの意味やそのチューニング方法については以下の資料を参照すること．

http://kaiyuzheng.me/documents/navguide.pdf

ナビゲーションのスクリプトの実行

作成したスクリプトを用いて，自律ナビゲーションを行います．
以下のように４つのターミナルを起動してください．

ターミナル１ Gazeboの起動

```
roslaunch seed_r7_gazebo seed_r7_example_world.launch
```

ターミナル2 自己位置同定とナビゲーションのためのノード

```
roslaunch seed_r7_navigation wheel_with_static_map.launch
```

ターミナル3 Rvizの起動

```
rosrun rviz rviz -d `rospack find seed_r7_bringup`/rviz.rviz
```

ターミナル4 作成したスクリプトの起動．
ターミナル３までのものがすべて起動してから実行してください．

```
rosrun seed_r7_navigation test_nav.py
```

指定座標まで走行し，初期位置まで戻ってくる様子が確認できる

第３部のまとめ

SEED-Noidを用いて，Rviz上で地図生成とナビゲーションの機能について手順を概説した.

地図生成や，ナビゲーションの手順は，SEED-Noidだけでなく，他の移動ロボットについても同様であるため，今回学んだ事を利用すれば，他のロボットにも応用が可能である.

付録

THK株式会社
産業機器統括本部 技術本部
事業開発統括部 永塚BU

ゲームパッドの設定について(1)

未登録のゲームパッドを利用する場合，各ボタンの番号を把握する必要がある．
ゲームパッドの各ボタンとjoyノードの間の対応関係についての調べ方を紹介する．

ここでは，JC-U4113SBKを例に示す．
(後ろのスイッチをDの方に合わせる．)

Joyノードの起動

下記のようにjoyノードを起動する．

```
roslaunch  teleop_twist_joy teleop.launch
```

ゲームパッドの設定について(2)

ボタンとjoyノードの対応を調べるために，直接トピックをモニタリングできる「rostopic echo」を用いる．別のターミナルを起動して，下記のコマンドを打つ

```
$rostopic echo /joy
```

このコマンドを実行することで，ゲームパッドの状態が表示される．

```
header:
  seq: 3877
  stamp:
    secs: 1595949879
    nsecs: 628703425
  frame_id: ''
axes: [-0.0, -0.0, -0.0, -0.0, -0.0, -0.0]
buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

ゲームパッドの設定について(3)

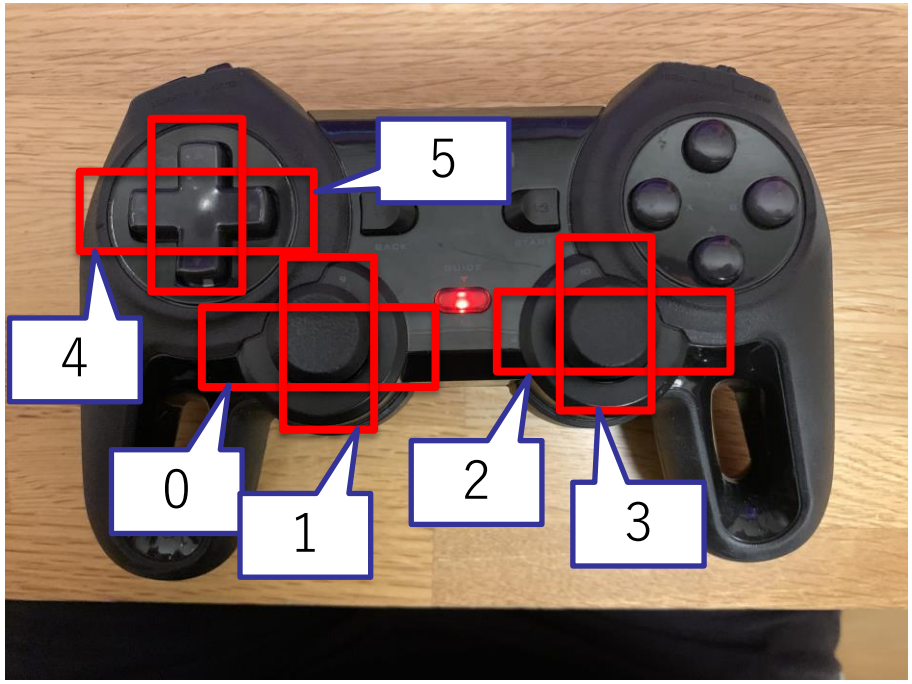
```
header:
  seq: 3877
  stamp:
    secs: 1595949879
    nsecs: 628703425
  frame_id: ''
axes: [-0.0, -0.0, -0.0, -0.0, -0.0, -0.0]
buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

「axes」は十字キーやジョイスティックの状態が格納.
axesは以下の様に表示される.
↑方向：＋　↓方向：－
←方向：－　→方向：＋
左右と上下はセットで，同じ配列で表現される.
「buttons」はボタンのON(1)/OFF(2)を取得
配列なので，一番左を0として順番に数える.

ゲームパッドの設定について(4)

前のスライドの事前知識を踏まえた上で，キーを押しながら，対応関係を調べていく．
例として， JC-U4113SBKでは以下の様になっている

axisについて

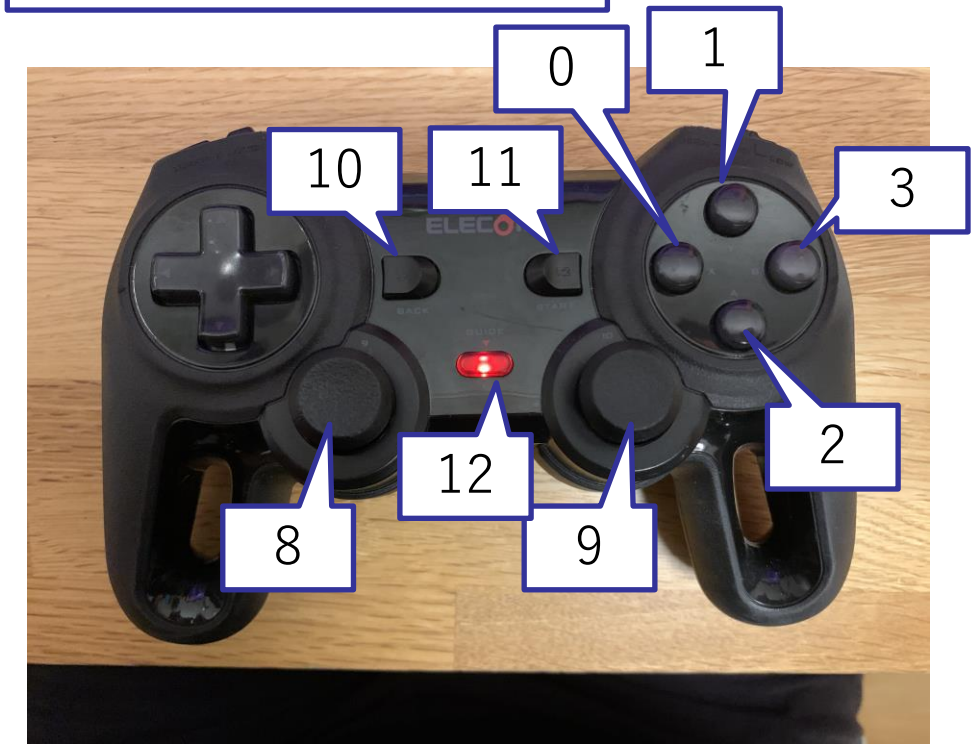


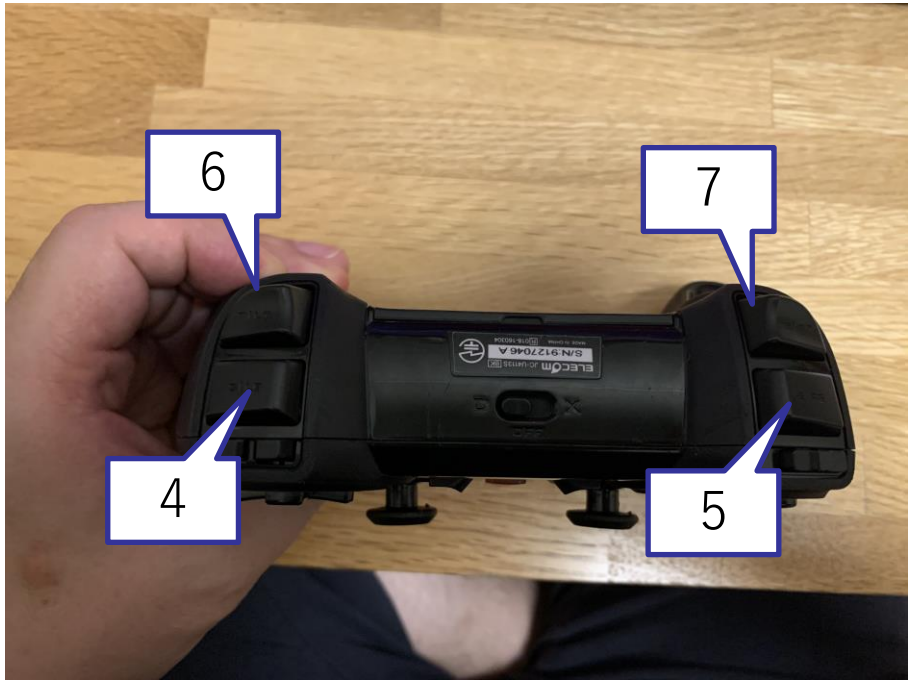
左の図のように対応している．
今回の演習では主に左ジョイスティックを前後左右の移動に，右ジョイスティックを回転に利用する．

ゲームパッドの設定について(5)

JC-U4113SBKには番号が印字されている.
番号-1がbuttonの配列と対応している.

buttonについて





ゲームパッドの設定について(6)

seed_r7_ros_pkgにおいて，ゲームパッドの設定ファイル(config.yaml)は下記に置かれている．

~/ros/melodic/src/seed_r7_ros_pkg/seed_r7_bringup/config

JC-U4113SBKの設定ファイルは以下にある．

elecom-holonomic.config.yaml

```
axis_linear:
  x: 1      #left axis up/down
  y: 0    #left axis left/right
scale_linear:
  x: 0.3
  y: 0.2
scale_linear_turbo:
  x: 0.6
  y: 0.4

axis_angular:
  yaw: 3      #right axis left/right
scale_angular:
  yaw: 0.6
scale_angular_turbo:
  yaw: 1.0

enable_button: 4       #L1 button
enable_turbo_button: 6  #L2 button
```

X方向の制御に左ジョイスティックの左右(0)，
Y方向の制御に上下(1）を設定
通常時とターボ時のスケールを倍率を設定

左回転，右回転の制御に，右ジョイスティックの左右(3)を利用．
通常時とターボ時のスケールを倍率を設定

4番（L1ボタン）
高速動作の「enable」として，
6番（L2）ボタンがアサイン

ゲームパッドの設定について(7)

このように，調べていくことで，任意のゲームパッドに対応させていくことができる．

あらたにファイルを作成した場合は，他のconfig.yamlと同じように下記に格納しておくことにする．

```
~/ros/melodic/src/seed_r7_ros_pkg/seed_r7_bringup/config
```

使いたいゲームパッドの設定を読み込ませるためには，下記に読み込ませるファイル名を設定しておく．

```
~/ros/melodic/src/seed_r7_ros_pkg/seed_r7_navigation/launch/wheel_bringup.launch
```

このファイルの下記の場所を書き換えれば良い．

```
19  <!-- DualShock3 Settings -->
20  <arg name="joy_config" default="elecom-holonomic" />
21  <arg name="joy_dev" default="/dev/input/js0" />
22  <arg name="config_filepath" default="$(find seed_r7_bringup)/config/$(arg joy_config).config.yaml" />
```

ここを書き換える．
（.config.yamlを抜いた形で．）