
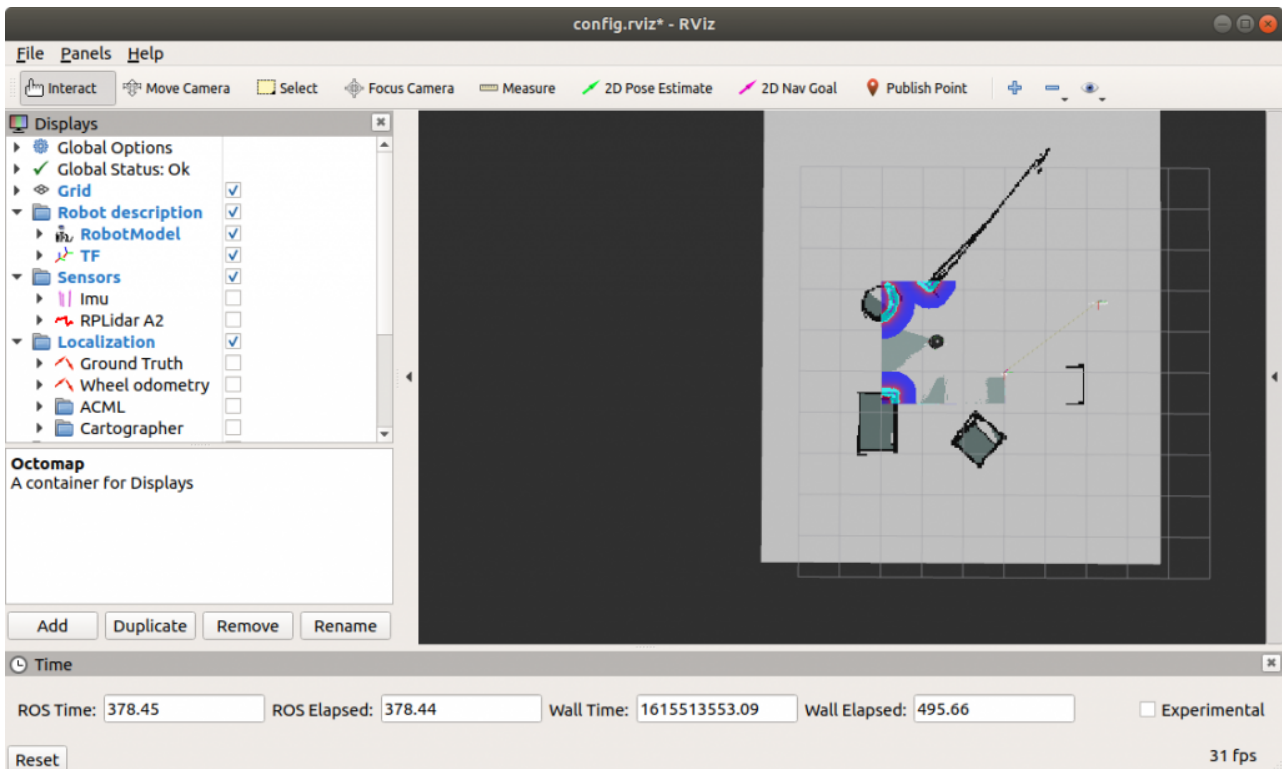


HARD2021：ルンバをPythonプログラムでナビゲーションさせよう！

 demura.net/robot/hard/20114.html

2021.03.12



この記事は**HARD2021**(Home AI Robot Development)スプリングワークショップ用です。今回はナビゲーション用のPythonプログラムを作りましょう。このナビゲーションはロボットが通過する地点(ウェイポイント)を人間が設定し、それを順番にたどるウェイポイントナビゲーションとよばれているもっとも基本的なものです。ROSのナビゲーションの機能も使っているので途中で障害物があれば、ロボットがそれを避けてウェイポイントまで行ってくれる優れたものです。

ナビゲーションのPythonプログラムを簡単にするためにROSの**Actionlib**を使います。ROSには非同期で片方向通信の**トピック通信**、同期で双方向通信の**サービス通信**があります。あるノード(クライアント)が別のノード(サーバー)にロボットのあるタスクを依頼する場合、メッセージをやり取りする必要があるため双方向通信のService通信の方が都合が良いわけですが一つ問題があります。それはサーバーの処理が遅い場合、クライアントがその処理が終わるまでブロック(待機)させられてしまいます。さらに、クライアントが途中でそのタスクをキャンセルしたい場合や途中結果などを聞きたい場合がありますが、Service通信では対応していません。それに対応するためにROSには、**アクション通信**があります。

アクション通信は サービス通信と同様にタスクを依頼する**アクションクライアント**(ActionClient)と処理をする**アクションサーバー**(ActionServer)があり、サービス通信と異なる点は**非同期通信**ということです。**actionlib**パッケージはアクション通信のためのツールを提供しています。

まずは、次のROS Wiki、ROSチュートリアルを読んでActionLibの詳細を理解しましょう。

- [actionlib\(日本語\)](#)
- [ActionLib Tutorial\(英語\)](#)

次のソースコードはこのC++プログラムのPython版です。プログラムのにはクラス、例外処理なども入ってきて高度になってきています。クラスを使うとグローバル変数なども使う必要がなくシンプルにコーディングできます。

ソース

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import rospy
import tf
import actionlib
from actionlib_msgs.msg import *
from geometry_msgs.msg import Pose, PoseWithCovarianceStamped, Point, Quaternion\
, Twist
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
from math import pi

# クラス
class WpNavi():
    def __init__(self): # コンストラクタ
        rospy.init_node('wp_navi') # ノードの初期化
        rospy.on_shutdown(self.shutdown) # シャットダウン時の処理

        # アクションクライアントの生成
        self.ac = actionlib.SimpleActionClient('move_base', MoveBaseAction)

        # アクションサーバーが起動するまで待つ。引数はタイムアウトの時間(秒)
        while not self.ac.wait_for_server(rospy.Duration(5)):
            rospy.loginfo("Waiting for the move_base action server to come up")

        rospy.loginfo("The server comes up")

        self.goal = MoveBaseGoal() # ゴールの生成
        self.goal.target_pose.header.frame_id = 'map' # 地図座標系
        self.goal.target_pose.header.stamp = rospy.Time.now() # 現在時刻

        way_point = [[-1.0, -1.0, 0.0 * pi], [0.0, -5.0, 0.5 * pi],
                     [3.0, -5.0, -1.0 * pi], [3.0, 3.0, -0.5 * pi],
                     [-2.0, 2.0, 0.0 * pi], [999, 999, 999]]

        # メインループ。ウェイポイントを順番に通過
        i = 0
        while not rospy.is_shutdown():
            # ROSではロボットの進行方向がx座標、左方向がy座標、上方向がz座標
            self.goal.target_pose.pose.position.x = way_point[i][0]
            self.goal.target_pose.pose.position.y = way_point[i][1]

            if way_point[i][2] == 999:
                break

            q = tf.transformations.quaternion_from_euler(0, 0, way_point[i][2])
            self.goal.target_pose.pose.orientation = Quaternion(
                q[0], q[1], q[2], q[3])
            rospy.loginfo("Sending goal: No" + str(i + 1))

            self.ac.send_goal(self.goal)
            # サーバーにgoalを送信

            # 結果が返ってくるまで30.0[s] 待つ。ここでブロックされる。
            succeeded = self.ac.wait_for_result(rospy.Duration(30))

            # 結果を見て、成功ならSucceeded、失敗ならFailedと表示
            state = self.ac.get_state()
            if succeeded:
                rospy.loginfo(
                    "Succeeded: No." + str(i + 1) + "(" + str(state) + ")")
            else:
                rospy.loginfo(
```

```

        "Failed: No." + str(i + 1) + "(" + str(state) + ")")

    i = i + 1

# シャットダウン時の処理
def shutdown(self):
    rospy.loginfo("The robot was terminated")
    self.ac.cancel_goal() # ゴールをキャンセル

if __name__ == '__main__':
    # 例外処理。rospy.ROSInterruptExceptionを捕まえる。
    # この例外はCtrl+cキーが押されるときに発生するので、
    # この例外処理によりこのノードが終了する。
    try:
        WpNavi()
        rospy.spin()
    except rospy.ROSInterruptException:
        rospy.loginfo("WP navigation finished.")

```

launchファイル

- ノードを実行するためにroslaunchコマンドを実行しました。複数のノードを立ち上げたり、複数の変数を設定するのをコマンドでいちいち実行するのは面倒です。それを解決してくれるのが**launchファイル**とよばれるファイルです。
- launchファイルを作りたい場合、パッケージディレクトリの中にlaunchというディレクトリを作り、そのなかに拡張子がlaunchというファイルを作ります。それがlaunchファイルです。以下のコードがその例です。launchファイルはXML形式で書きます。XML形式はいろいろなアプリでよく使われています。
 - launchタグ：launchファイルは<launch>で始まり</launch>で終わります。その中に書いていきます。
 - argタグ：引数を指定する。
 - name: 引数名。
 - default: 引数のデフォルト(規定)値
 - value: 引数の値
 - nodeタグ：ノードを設定する。
 - pkg: パッケージ名。この例ではwp_navi。
 - type: 実行ファイル名。この例ではwp_navi.py。
 - name: ノード名。この例ではwp_navi。パッケージ名とノード名は同じ場合が多い。
 - ns: 名前空間(name space)。create_autonomyパッケージでは複数台のロボットを扱うために名前空間を使っている。ns = \$(args ns)は3行目で設定した値 create1を名前空間としている意味。これがないとwp_navi.pyは動かない。
 - output: screenの場合はノードからの出力を端末に表示する。指定しないとログファイルだけに記録される。

```

<launch>
  <arg name="id" default="$(optenv ID 1)" doc="Unique identifier of the robot [1-Inf.)"/>
  <arg name="ns" value="create$(arg id)" doc="Namespace of the robot. By default: create1."/>

  <node pkg="wp_navi" type="wp_navi.py" name="wp_navi"
        ns="$(arg ns)" output="screen" />
</launch>

```

ハンズオン

では、さっそくwp_naviパッケージを作り、wp_navi.pyを起動するlaunchファイルを作って実行して、ロボットにナビゲーションをさせてみましょう！

- パッケージの作成
 - `$ cd ~/catkin_ws/src/hard2021`
 - `$ catkin_create_pkg wp_navi roscpp rospy std_msgs`
- ソースコードの作成
 - 次のコマンドでgeditを開き、上のソースコード(wp_navi.py)をコピーしてセーブする。
`$ gedit ~/catkin_ws/src/hard2021/wp_navi/src/wp_navi.py &`
 - wp_navi.pyに実行権を与える。
 - `$ cd ~/catkin_ws/src/hard2021/src`
 - `$ chmod u+x wp_navi.py`
- launchファイルの作成
 - `$ cd ~/catkin_ws/src/hard2021/wp_navi`
 - `$ mkdir launch`
 - `$ cd launch`
 - 次のコマンドでgeditを開き、上図のソースコード(wp_navi.launch)を打ち込んでwp_navi.launchというファイル名をつけてlaunchディレクトリの中にセーブする。全部打ち込むのがつらい場合は、
~/catkin_ws/src/create_autonomy/navigation/ca_move_base/launch/send_goal.launch
ファイルをコピーして改変すると楽できます。
`$ gedit ~/catkin_ws/src/hard2021/wp_navi/launch/wp_navi.launch &`
- ビルドと実行権の付与
 - `$ cd ~/catkin_ws`
 - `$ catkin build wp_navi`
- 実行
 - 端末を2つ開く
 - 1番目の端末
 - `$ export LOCALIZATION=amcl`
 - 新しいcreate_playgroundの3Dモデルをネットから読み込むので表示に数分かかる。
`$ roslaunch ca_gazebo create_playground.launch`
 - 2番目の端末
 - `$ source ~/.bashrc`
 - 上で作成したwp_navi.launchファイルを使います。
`$ roslaunch wp_navi wp_navi.launch`
 - 上で作成したlaunchファイルを使わない場合は、wp_navi.pyの22行目の'move_base'を'create1/move_base'に変更して次のコマンドを実行する。
`$ rosrunc wp_navi wp_navi.py`
 - 成功するとロボットがウェイポイント順番に通過し、ぐるっと1周してスタート地点に戻ります。

終わり

