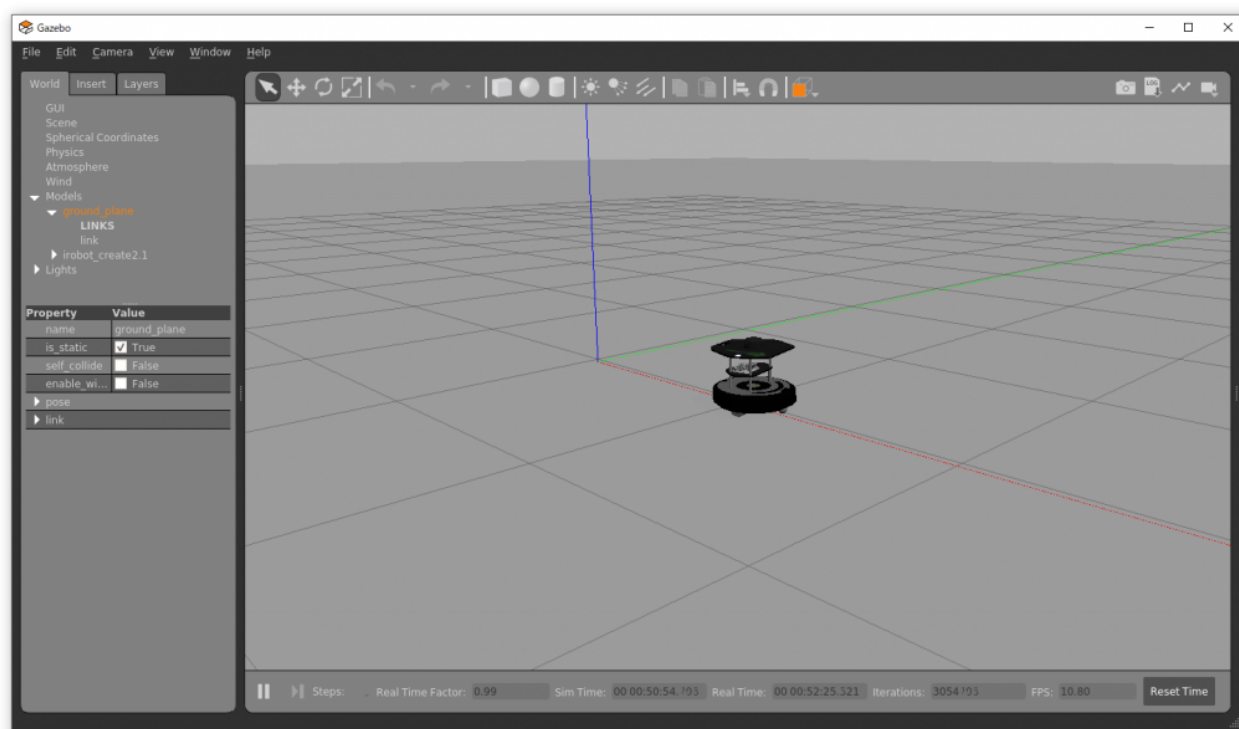


HARD2021: ルンバをPythonプログラムで動かそう！

 demura.net/robot/hard/20101.html

2021.03.11



Pythonプログラムでルンバを動かしてみましょ！まず、ソースコードを見てみましょう。今回もたったの39行です。PythonプログラムはC++でコーディングする場合と比較して圧倒的に短くなるので初心者にもとっつきやすいと思います。

ソースコード

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import rospy
from geometry_msgs.msg import Twist

def set_vel(vel_msg, lv, av):
    vel_msg.linear.x = lv
    vel_msg.linear.y = 0
    vel_msg.linear.z = 0
    vel_msg.angular.x = 0
    vel_msg.angular.y = 0
    vel_msg.angular.z = av

def main_loop():
    rospy.init_node('move')
    vel_publisher = rospy.Publisher('/create1/cmd_vel', Twist, queue_size=10)
    vel_msg = Twist()
    set_vel(vel_msg, 0, 0)
    print("Let's move your robot")
    linear_vel = input("Input linear velocity [m/s] :")
    angular_vel = input("Input angular velocity [rad/s] :")
    vel_msg.linear.x = linear_vel
    vel_msg.angular.z = angular_vel
    rate = rospy.Rate(10)

    while not rospy.is_shutdown():
        vel_publisher.publish(vel_msg)
        rospy.loginfo("Velocity: Linear=%s Angular=%s", vel_msg.linear.x,
vel_msg.angular.z)
        rate.sleep()

if __name__ == '__main__':
    main_loop()
```

ソースコードの説明

このプログラムではROSの速度指令トピック/create1/cmd_velをmove_baseノードへパブリッシュ(配信)することによりロボットを動かしています。通常速度指令のトピックは/cmd_velですが、このcreate_autonomyパッケージでは複数台のロボットを識別するために/create1を前に付けています。2代目のロボットなら/create2になります。なお、速度指令トピックの型はTwist型で、3次元の並進速度ベクトルと角速度ベクトルがメンバーになっています。

次に重要なことはROSではループの周期をrospy.Rate(周波数)で設定し、ループの中にrospy.sleep()を入れることで簡単に一定の周期でループを制御できることです。ロボットの制御においては一定周期でループを回して処理をすることがとても大切です。

それ以外の詳細については、ソースコードに説明を加えたものを以下示しますので参照してください。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*- # 日本語を使うためのおまじない。
import rospy # ROSでpython使う場合に必要
from geometry_msgs.msg import Twist # ロボットの速度を扱う場合は必要

# set_vel関数：速度の設定。
# 1番目の引数は速度メッセージ、2番目の引数は並進速度、3番目の引数は角速度
def set_vel(vel_msg, lv, av):
    vel_msg.linear.x = lv
    vel_msg.linear.y = 0
    vel_msg.linear.z = 0
    vel_msg.angular.x = 0
    vel_msg.angular.y = 0
    vel_msg.angular.z = av

# main関数
def main():
    # ノードの初期化。引数はノード名。ROSでは同じノード名のノードを複数作ることはできない。
    rospy.init_node('move')

    # パブリッシャーの生成。一番目の引数はトピック名、2番目の引数はメッセージの型、
    # 速度指令はTwist型。3番目の引数はメッセージバッファのサイズ。
    vel_publisher = rospy.Publisher('/create1/cmd_vel', Twist, queue_size=10)

    vel_msg = Twist() # vel_msgオブジェクトの生成
    set_vel(vel_msg, 0, 0) # 速度の初期化。安全のために0に設定。

    # キーボードからの入力
    linear_vel = input("Input linear velocity [m/s] :")
    angular_vel = input("Input angular velocity [rad/s] :")

    # 並進速度または角速度の設定
    rospy.is_shutdown():
    # ROS座標系は右手系、ロボットの前進方向がX軸、右方向がY軸、上方向がZ軸
    vel_msg.linear.x = linear_vel
    vel_msg.angular.z = angular_vel

    # ループの周期を設定。10Hz。つまり、1ループ100ms。
    # ロボットを一定の周期で動かすことはとても重要
    rate = rospy.Rate(10)

    # Ctrl-Cが押されるまで無限ループ
    while not rospy.is_shutdown():
        vel_publisher.publish(vel_msg) # メッセージの配信。ロボットに速度指令を送る。
        rospy.loginfo("V: Linear=%s Angular=%s", vel_msg.linear.x,
        vel_msg.angular.z) # 表示
        rospy.sleep() # rospy.Rate()で指定した時間になるように調整

# このプログラムをモジュールとしてimportできるようにするおまじない。
# なお、モジュールとは他のプログラムから再利用できるようにしたファイルのこと。
if __name__ == '__main__':
    main()
```

パッケージの作成

- パッケージは次のcatkin_create_pkgコマンドで作ります。依存パッケージはそのパッケージを作るために必要なパッケージです。
catkin_create_pkg <パッケージ名> [依存パッケージ1] [依存パッケージ2] [依存パッケージ3]

- では、次のコマンドでmoveパッケージを作りましょう！既にhard2021ディレクトリを作成している場合は、2番目のcdコマンドから実行してください。
 - `$ mkdir -p ~/catkin_ws/src/hard2021`
 - `$ cd ~/catkin_ws/src/hard2021`
 - `$ catkin_create_pkg move roscpp rospy std_msgs`
- 上手く作成できたか確認しましょう。
 - `$ cd ~/catkin_ws/src/hard2021/move`
 - `$ ls`
 - 次のファイルができていれば成功です。
CMakeLists.txt, include, package.xml, srcができていれば成功。

ソースコードの作成

エディタを開き、上の説明のない方のソースコードをコピーしてセーブします。以下のコマンドはエディタにgeditを使う場合です。

- `$ cd ~/catkin_ws/src/hard2021/move/src`
- `$ gedit move.py`

ビルド

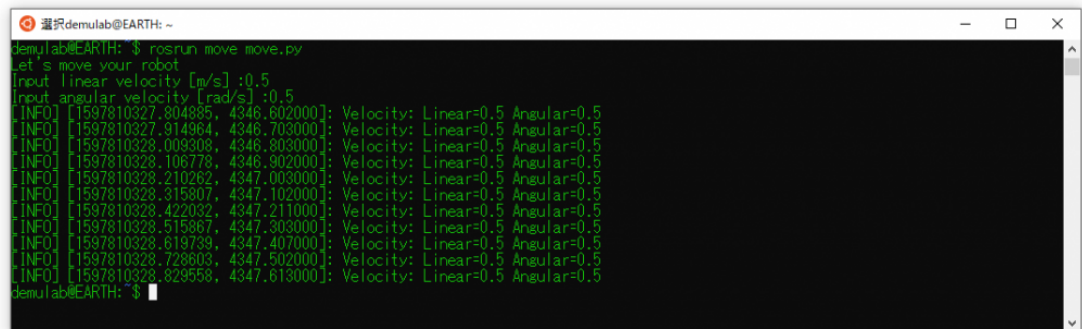
- Pythonはビルドする必要はないが、ROSで使用する場合は必要になる場合があるので、以下のコマンドを実行しましょう。
 - `$ cd ~/catkin_ws`
 - `$ catkin build move`
- 作成したpythonプログラムに実行権を与えましょう。実行権を与えないとroslaunchコマンドで実行できません。
 - `$ source ~/.bashrc`
 - `$ roscd move/src`
 - `$ chmod u+x move.py`

実行

- シミュレータの起動
 - 今回はrvizを使わないので、端末を開き、次のコマンドでGazeboと同時に起動しないようにする。
 - `$ export RVIZ=false`
 - 続けて、以下のコマンドを実行してシミュレータGazeboを起動する。一番上図のルンバをベースにしたロボットが現れる。
 - `$ roslaunch ca_gazebo create_empty_world.launch`

- move.pyノードの実行

- 端末をもう一つ開き、以下のコマンドを実行する。
 - `$ roscd move/src`
 - `$ rosrun move move.py`
- 以下のように並進速度と角速度を聞かれるので入力する。この例では、linear velocity (並進速度) : 0.5 [m/s]、Angular velocity (角速度) : 0.5 [rad/s]を入力している。



```
demulab@EARTH: ~$ rosrun move move.py
let's move your robot
input linear velocity [m/s]: 0.5
input angular velocity [rad/s]: 0.5
[1597810327.804885, 4346.602000]: Velocity: Linear=0.5 Angular=0.5
[1597810327.814964, 4346.703000]: Velocity: Linear=0.5 Angular=0.5
[1597810328.009308, 4346.803000]: Velocity: Linear=0.5 Angular=0.5
[1597810328.106778, 4346.902000]: Velocity: Linear=0.5 Angular=0.5
[1597810328.210262, 4347.003000]: Velocity: Linear=0.5 Angular=0.5
[1597810328.315807, 4347.102000]: Velocity: Linear=0.5 Angular=0.5
[1597810328.422032, 4347.211000]: Velocity: Linear=0.5 Angular=0.5
[1597810328.515867, 4347.303000]: Velocity: Linear=0.5 Angular=0.5
[1597810328.619739, 4347.407000]: Velocity: Linear=0.5 Angular=0.5
[1597810328.728603, 4347.502000]: Velocity: Linear=0.5 Angular=0.5
[1597810328.829558, 4347.615000]: Velocity: Linear=0.5 Angular=0.5
demulab@EARTH: $
```

- ロボットが円軌道上を動いたら成功。なお、角速度を0にすると直進するが、半径無限大の円周上を移動しているとも考えることもできる。
ロボットを停止する場合は、まず、rosrunを実行した端末でCtrl+cキーを押してプロセスを止める。次にもう一度rosrun move move.pyを実行して並進と角速度を0にする。並進速度に負の値を入れると後進し、角速度に負の値を入れるとロボットを上から見て時計周りに回転する。

終わり