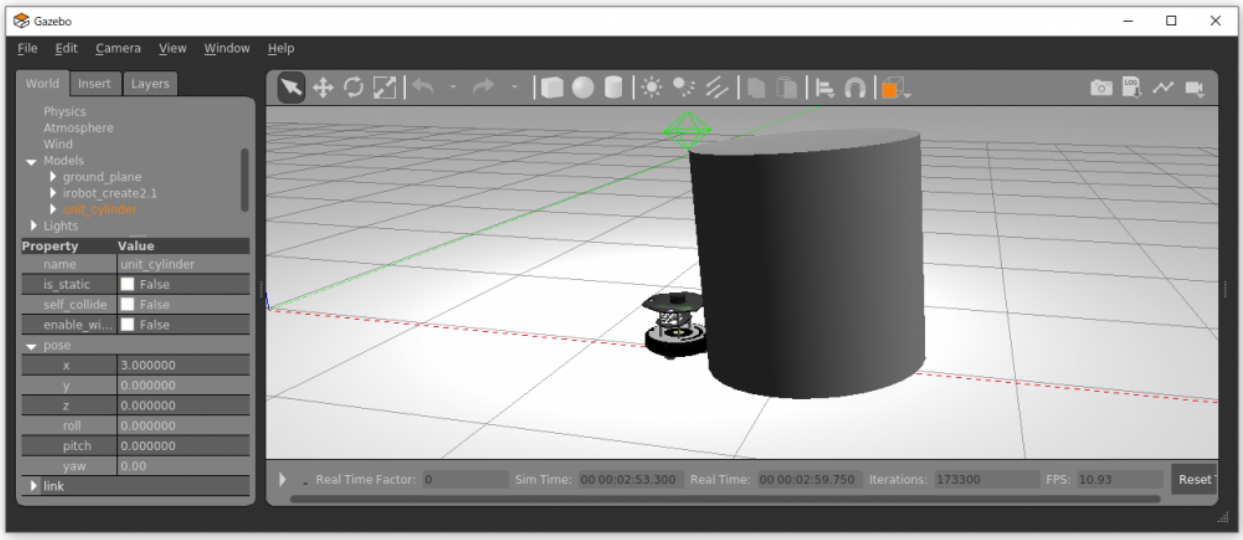
HARD2021：PythonプログラムでLIDARを使おう！

demura.net/robot/hard/20141.html

March 14, 2021



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**LIDAR(Laser Imaging Detection and Ranging)**の情報を取得するPythonプログラムを作りましょう。LIDARは日本語では**レーザ式測域センサ**ともよびますが、LIDARとそのままよばれる方が多いです。SLAMや自己位置推定になくてはならないもので、自動運転車の重要なセンサとして急速に開発が進められています。

本ワークショップのシミュレータで使用しているLIDARはSLAMTECが開発したRPLIDAR A2をモデルにしたもので、検出距離が0.15[m]から12[m]、検出範囲は360[°]、分解能は0.5[°]となっています。これを今回説明するPythonプログラムでは端末に表示します。

ROSではLIDARのトピック名は**/scan**が一般的ですが、このシミュレータではトピック名は**/create1/rplidar/scan**となっています。メッセージの型は**sensor_msgs/LaserScan**で、次のリンクで定義されています。メンバーは、計測距離データranges[m]、反射強度データintensitiesの他に、角度分解能angle_increment[rad]などがあります。詳細は次のリンクをご覧くだあい。

sensor_msgs/LaserScan Message

では、LIDARの計測データを取得するPythonプログラムを見ていきましょう。見覚えありませんか？前回学んだPythonプログラムとほぼ同じです。トピック名とメッセージ型名を変えただけです。

サンプルプログラム（説明付き）

このサンプルプログラムの概要を説明します。サブスクライバでは、LIDARからのデータがある/create1/rplidar/scanトピックをサブスクライブします。LIDARのデータはself.rangesに格納されているので、それを使ってハンズオンのミッションをクリアしてください。
パブリッシャは、ルンバを動かすために、キーボードから読みんだ並進と角速度をset_vel関数でセットして、/create1/cmd_velトピックをパブリッシュしています。

```
#!/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
from sensor_msgs.msg import LaserScan # LIDARを使うときは必要
import math   # 数学関数モジュール

# RoombaRobotクラス
class RoombaRobot():
    def __init__(self):  # コンストラクタ
        #  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。
        #  2番目の引数anonymous=Trueを付けると、同じノード名がある場合に自動的にユニークなIDを
        #  つけたノード名に変換してくれる。
        rospy.init_node('lidar', anonymous=True)

        # サブスクライバ（購読者)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、
        # オドメトリのメッセージはLaserScan型(正確にはsensor_msgs/LaserScan型だが
        # from sensor_msgs.msg import LaserScanでインポートしているのでここではLaserScanだ
けでよい)。
        # ３番目の引数はコールバック関数。
        self.odom_sub = rospy.Subscriber('/create1/rplidar/scan', LaserScan,
self.lidar_callback)

        # パブリッシャの生成。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ーのサイズ。
        self.vel_pub = rospy.Publisher('/create1/cmd_vel', Twist, queue_size=10)
        self.set_vel(0, 0)

    # set_vel関数：速度の設定。
    def set_vel(self, lv, av):
        vel = Twist()
        vel.linear.x  = lv  # 並進速度
        vel.linear.y  = 0
        vel.linear.z  = 0
        vel.angular.x = 0
        vel.angular.y = 0
        vel.angular.z = av  # 角速度

        self.vel_pub.publish(vel)

        # 表示
        rospy.loginfo("Velocity: Linear=%s Angular=%s", vel.linear.x,
vel.angular.z)
        del vel

    # lidar_callback関数
    # /scanのMessage型は
http://docs.ros.org/melodic/api/sensor_msgs/html/msg/LaserScan.htmlを参照。
    def lidar_callback(self, msg):
        self.ranges          = msg.ranges
        self.angle_min       = msg.angle_min # スキャンの開始角度[rad]
        self.angle_max       = msg.angle_max # スキャンの終了角度[rad]
        self.angle_increment = msg.angle_increment # 角度分解能[rad]
        self.range_min       = msg.range_min # 最小検出距離[m]
        self.range_max       = msg.range_max # 最大検出距離[m]

    # main関数
    def main(self):
        # ループの周期を設定。100Hz。つまり、1ループ10ms。
```

```
        rate = rospy.Rate(100)

        # キーボードからの入力
        print("Let's move your robot")
        linear_vel  = input("Input linear velocity [m/s] :")
        angular_vel = input("Input angular velocity [rad/s] :")
        self.set_vel(linear_vel, angular_vel)

        # Ctrl-Cが押されるまで無限ループ
        while not rospy.is_shutdown():

            rospy.loginfo("Number of Rays=%d", len(self.ranges))  # レーザの本数

            # シミュレーションのPRDLIARは全周360[°] 計測可能。計測角度は-180[°]から180[°]。
            # ROSは右手系、進行方向x軸、左方向y軸、上方向がz軸（反時計まわりが正)。
            rospy.loginfo("Angle [rad] min=%f max=%f", self.angle_min,
self.angle_max)

            # ROSの角度は[rad]。math.degresss関数でラジアンから°に変換している。
            rospy.loginfo("Angle [deg] increment=%.3f",
math.degrees(self.angle_increment))
            rospy.loginfo("Range [m] min=%.3f max=%.3f", self.range_min,
self.range_max)
            rospy.loginfo("  0[deg]=%.3f[m]  90[deg]=%.3f[m]",
self.ranges[0],self.ranges[180])
            rospy.loginfo("180[deg]=%.3f[m] -90[deg]=%.3f[m]",
self.ranges[360],self.ranges[540])

            #  ハンズオン１：ここに、何かに0.3[m]以内に近づいたら停止するコードを書く


            # rospy.Rate()で指定した時間になるように調整してくれる。
            rate.sleep()


# 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__':
    try:
        robot = RoombaRobot()
        robot.main()
    except rospy.ROSInterruptException: pass
```

パッケージの作成

次のコマンドでlidarパッケージを作りましょう！

- `$ cd ~/catkin_ws/src/hard2021`
- `$ catkin_create_pkg lidar roscpp rospy std_msgs`

ソースコードの作成

エディタを開き、上のソースコードをコピペしてlidar.pyというファイル名で保存する。以下のコマンドはエディタにgeditを使う場合。

- `$ cd ~/catkin_ws/src/hard2021/lidar/src`
- 端末を何個も起動しなくてよいようにコマンドの最後に&をつけてバックグラウンドでgeditを起動する。

```
$ gedit lidar.py &
```

ビルド

- Pythonはビルドする必要はないが、ROSにパッケージとして認識されるために、以下のコマンドでパッケージをビルドする。
  - `$ cd ~/catkin_ws`
  - `$ catkin build  lidar`
- 作成したpythonプログラムに実行権を与える。実行権を与えないとrosrunコマンドで実行できない。
  - `$ source ~/.bashrc`
  - `$ roscd lidar/src`
  - `$ chmod u+x  lidar.py`

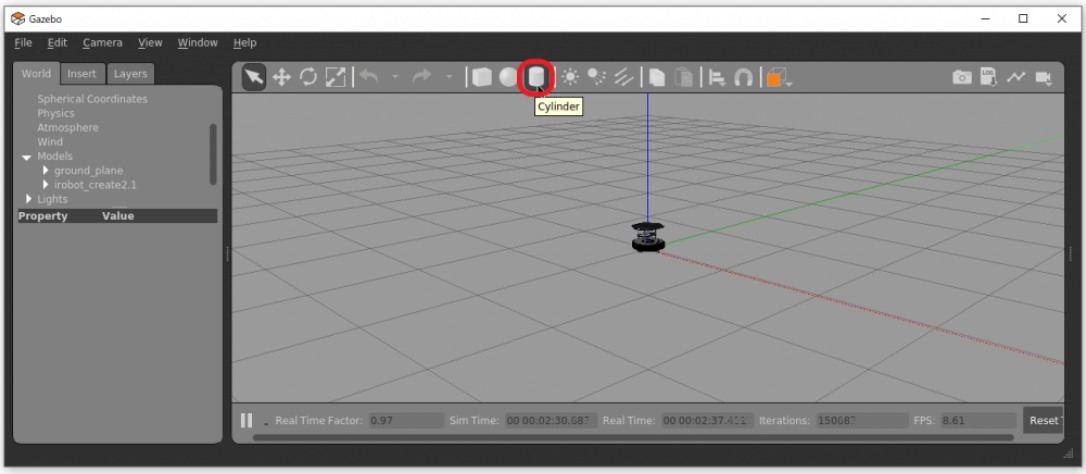
実　行

- シミュレータの起動

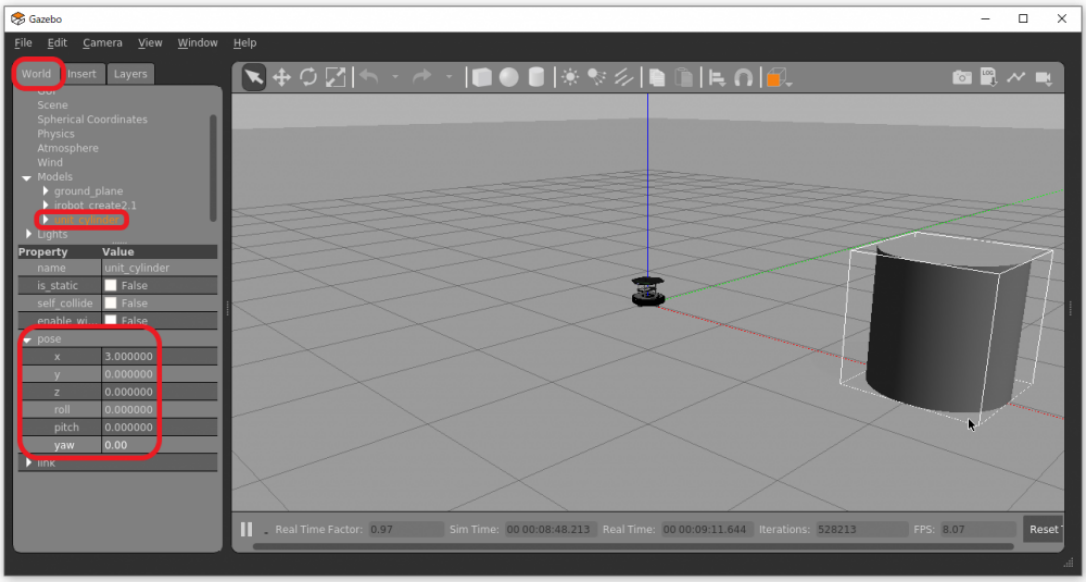
  端末を開き、以下のコマンドを実行してシミュレータGazeboを起動する。一番上図のルンバをベースにしたロボットが現れる。

```
$ roslaunch ca_gazebo create_empty_world.launch
```

- オブジェクトの挿入
  - 円柱の挿入：ロボットの上あるバーに、Box(立方体)、Sphere(球)、Cylinder(円柱）のアイコンがある。これをクリックして、シミュレータ上の好きな位置で放すと物体が取り込まれる。ここでは、Cylinder(円柱)を選択して、赤線で示されたx軸上の数m地点に置く。シミュレータの格子は各辺1[m]となっている。



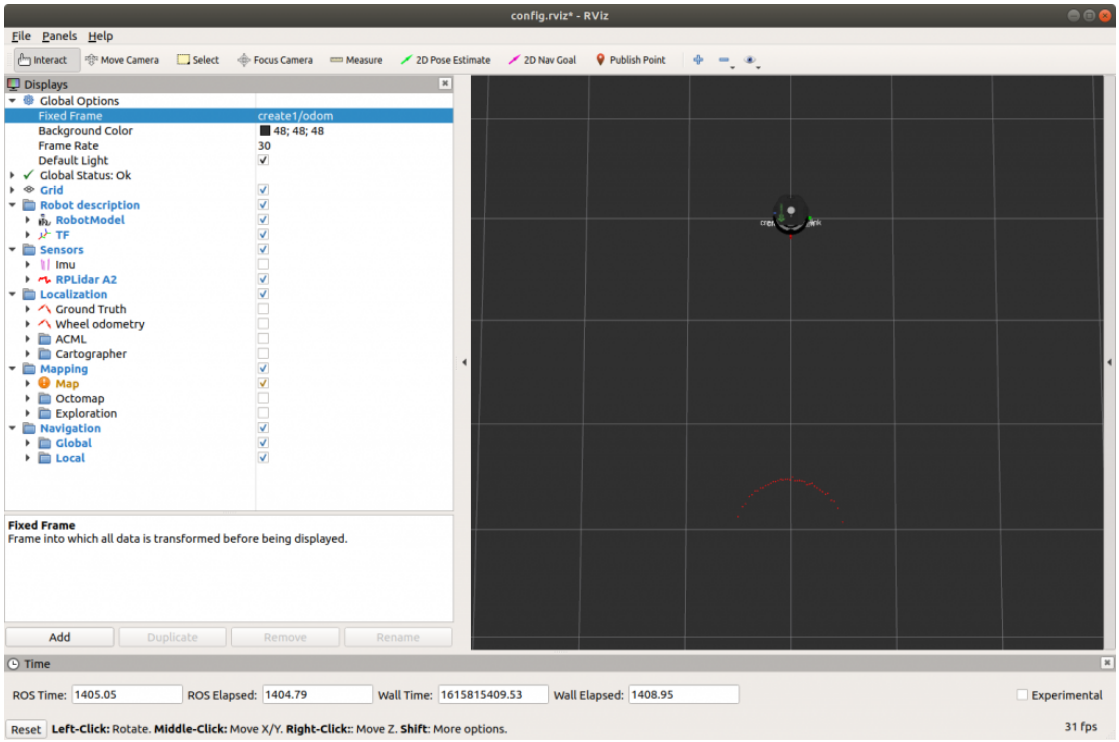
  - 位置の設定：取り込んだ円柱をクリックして、左メニューの[World]タブをクリックして、[Models]の[unit_cylinder]をクリックすると下図のように[Property]を設定できる。ここでは[Pose]のxを3.0、yを0、zを0.5にする。これで円柱は座標(3,0,0.5)の位置に移動する。



  - なお、ここでは円柱、立方体、球などの基本形状の取り込みを説明したが、[World]タブの右にある[Insert]タブから机などの什器、車やロボットなどいろいろな3Dモデルを取り込める。

- Rvizの設定
  - 次にRvizを設定しよう。デフォルト設定が地図座標系mapになっている。現在、mapサーバーが動いていないので、Rvizは[Global Status: Error]と表示されている。このエラーを解除するため、[Global Option]をクリックして次のように設定する。これは座標系をオドメトリで自己位置を計算するodom(オドメトリ)座標系にするという意味。今回のempty_worldはロボット以外に何も構造物がないので地図を作成できない。つまり、map座標系は使えない。

    Fixed Frame: create1/odom
  - 次に、[Sensors]の下に[RPLidar A2]という項目があるのでチェックを入れる。そうすると下図のようにLIDARの計測が視覚化される。レーザ光は物体を透過しないので、ロボット側だけ半円に並ぶ計測点が赤く表示さえる。



- 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 lidar/src`
    - `$ rosrun lidar lidar.py`
  - 実行すると並進速度と角速度が以下ように聞かれるので0.2と0を入力する。
    - Input linear velocity [m/s]: **0.2**
    - Input angular velocity [rad/s]: **0**
  - ロボットが入力された速度で直進し、円柱に衝突してもそのまま動き続ける。端末にはLIDARに関する検出距離、検出範囲、分解能などの情報が表示される。

ハンズオン

1. 上の説明に従ってサンプルプログラムを実行して動作を確認しよう！
2. ロボットが進行方向の円柱の0.3[m]以内に近づいたら停止するようなPythonプログラムを作ろう！
3. シミュレータに球を挿入して、その0.3[m]手前でロボットが停止するプログラムを作ろう！ただし、球は初期位置を任意に変えても動作するプログラムにすること。

4. シミュレータに立方体と球を挿入して、球形状の物体だけを追うプログラムを作ろう！完成するとロボットが玉転がし続けます。ただし、立方体と球の初期位置を任意に変えても動作するプログラムにすること。

終わり