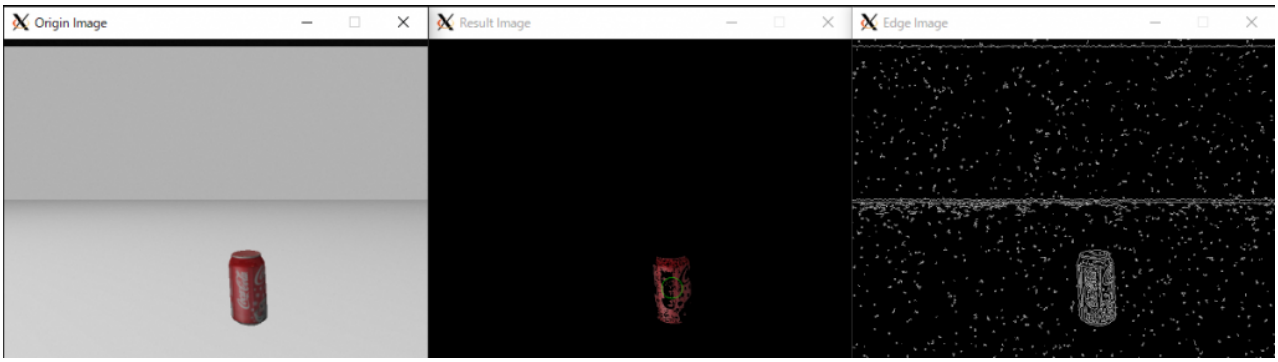
HARD2021：OpenCVとPythonプログラムで画像処理をしよう！

demura.net/robot/hard/20145.html

March 14, 2021



この記事はHARD2021（Home AI Robot Development)ワークショップ用です。今回は、コンピュータビジョンライブラリの定番OpenCVをROSで使うためにcv_bridgeを用いたPythonプログラムを勉強します。サンプルのPythonプログラムではカメラから取得した画像のエッジを検出したり、指定した色領域を抽出しています。

なお、この記事は、以下のサンプルプログラムはROSチュートリアルと「ROSで始めるロボットプログラミング、著者：小倉崇、出版社：工学社」を参考にしました。

> Converting between ROS images and OpenCV images (Python)

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。まず、カメラから取得したデータが/create1/camera/image_rawトピックにあるので、それをサブスクライブします。次に、callback関数では、トピックの画像の生データdataはsensor_msgs::Image型なので、それをOpenCVで処理できるようにcv2:Mat型に変換します。OpenCVでの処理は、指定した色領域を抽出するためにマスク画像を生成し、それと画像のビット毎の論理積を計算したり、画像のエッジを検出するために、カラー画像からグレースケール画像に変換し、その結果をウインドウで表示しています。詳細については、以下のソースコードをご覧ください。

```
#!/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を使う場合に必要
import cv2   # OpenCVをpytonで使う場合に必要
import numpy as np
from std_msgs.msg import String
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError # OpenCVをROSで使う場合に必要

class image_converter:
    def __init__(self):
        # 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が、
        # anonymous=Trueを付けると自動的にノード名を変換してくれる。
        rospy.init_node('camera', anonymous=True)

        # 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、
        # 3番目の引数はメッセージバッファーのサイズ。
        self.image_pub = rospy.Publisher("image_topic", Image, queue_size=1)
        self.bridge    = CvBridge()

        # サブスクライバーの生成。一番目の引数はトピック名、２番目の引数はメッセージ型、
        # 3番目の引数はコールバック関数。ここではカメラの画像生データを含んでいる
        # /create1/camera/image_rawトピックをサブスクライブする。
        self.image_sub =
rospy.Subscriber("/create1/camera/image_raw",Image,self.callback)

    # コールバック関数
    def callback(self,data):
        try:
            # img_to_cv2メソッド(メンバ関数)でROSのデータ形式sensor_msgs::Image をOpenCVの
形式cv:Matに変換する。
            cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
        except CvBridgeError as e:
            print(e)

        # RGB表色系からHSV表色系に変換
        hsv_image = cv2.cvtColor(cv_image, cv2.COLOR_BGR2HSV)

        # しきい値の設定（ここでは赤を抽出）
        color_min = np.array([150,100, 50])
        color_max = np.array([180,255,255])

        # マスク画像を生成
        color_mask = cv2.inRange(hsv_image, color_min, color_max);
        # 画像配列のビット毎の倫理積。マスク画像だけが抽出される。
        cv_image2  = cv2.bitwise_and(cv_image, cv_image, mask = color_mask)

        # 重心を求める。moments関数を使うためグレースケール画像へ変換。
        gray_image2 = cv2.cvtColor(cv_image2, cv2.COLOR_BGR2GRAY)
        mu = cv2.moments(gray_image2, True)
        x,y= int(mu["m10"]/mu["m00"]) , int(mu["m01"]/mu["m00"])

        # 重心を中心として半径20ピックセルの円を描画
        color  = (0, 255, 0)
        center = (x, y)
        radius = 20
        cv2.circle(cv_image2, center, radius, color)
```

```
        # エッジを検出する。Canny関数を使うためRGBからグレースケールへ変換
        gray_image = cv2.cvtColor(cv_image, cv2.COLOR_BGR2GRAY)
        cv_image3  = cv2.Canny(gray_image, 15.0, 30.0);

        # ウインドウのサイズを変更
        cv_half_image  = cv2.resize(cv_image,  (0,0),fx=0.5, fy=0.5)
        cv_half_image2 = cv2.resize(cv_image2, (0,0),fx=0.5,fy=0.5);
        cv_half_image3 = cv2.resize(cv_image3, (0,0),fx=0.5,fy=0.5);

        # ウインドウ表示
        cv2.imshow("Origin Image", cv_half_image)
        cv2.imshow("Result Image", cv_half_image2)
        cv2.imshow("Edge Image",   cv_half_image3)
        cv2.waitKey(3)

        try:
            # cv2_to_imgmsgメソッドでOpenCVのcv::Mat型をROSのsensor_msgs::Imageメッセージに変換。
            self.image_pub.publish(self.bridge.cv2_to_imgmsg(cv_image2, "bgr8"))
        except CvBridgeError as e:
            print(e)

# 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。
# ROS pythonで推奨されている形式
def main():
    try:
        ic = image_converter()
        rospy.spin()
    except KeyboardInterrupt:
        print("Shutting down")
        cv2.destroyAllWindows()
```

カメラの追加

現在のロボットモデルにはカメラが搭載されいないので追加する。ロボットモデルについては~/catkin_ws/src/create_autonomy/ca_description/urdfの中でいろいろ記述されている。カメラを追加するために以下の２つのファイルに追記する。なお、既に追加済みのファイルがあるので、次のファイル編集作業をやる必要はありません。

ファイル編集

create_base.xacro：110行目 <!– Simulation sensors –>の上に以下を挿入する。

```
<!-- Camera追加  ここから-->
<xacro:property name="camera_link" value="0.05" />
<!-- Size of square 'camera' box -->
<joint name="camera_joint" type="fixed">
  <axis xyz="1 0 0" />
  <origin xyz="0.1 0 0.19" rpy="0 0 0"/>  <!-- change -->
  <parent link="base_link"/>
  <child link="camera_link"/>
</joint>

<!-- Camera -->
<link name="camera_link">
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="${camera_link} ${camera_link} 0.03"/>
    </geometry>
  </collision>

  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="${camera_link} ${camera_link} 0.03"/>
    </geometry>
    <material name="black"/>
  </visual>

  <inertial>
    <mass value="1e-5" />
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6" />
  </inertial>
</link>
<!-- Camera追加 ここまで -->


<!-- Simulation sensors -->
<xacro:sim_create_base/>
```

create_base_gazebo.xacro：下から３行目に以下を挿入する。

```
    <!-- camera追加　ここから-->
    <!-- from http://gazebosim.org/tutorials?tut=ros_gzplugins -->
    <gazebo reference="camera_link">
      <sensor type="camera" name="camera">
        <update_rate>30.0</update_rate>
        <camera name="head">
          <horizontal_fov>1.3962634</horizontal_fov>
          <image>
            <width>800</width>
            <height>600</height>
            <format>R8G8B8</format>
          </image>
          <clip>
            <near>0.02</near>
            <far>300</far>
          </clip>
          <noise>
            <type>gaussian</type>
            <!-- Noise is sampled independently per pixel on each frame.
                 That pixel's noise value is added to each of its color
                 channels, which at that point lie in the range [0,1]. -->
            <mean>0.0</mean>
            <stddev>0.007</stddev>
          </noise>
        </camera>
        <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
          <alwaysOn>true</alwaysOn>
          <updateRate>0.0</updateRate>
          <cameraName>/camera</cameraName>
          <imageTopicName>image_raw</imageTopicName>
          <cameraInfoTopicName>camera_info</cameraInfoTopicName>
          <frameName>camera_link</frameName>
          <hackBaseline>0.07</hackBaseline>
          <distortionK1>0.0</distortionK1>
          <distortionK2>0.0</distortionK2>
          <distortionK3>0.0</distortionK3>
          <distortionT1>0.0</distortionT1>
          <distortionT2>0.0</distortionT2>
        </plugin>
      </sensor>
    </gazebo>
    <!-- camera追加　ここまで-->


  </xacro:macro>
</robot>
```

追加済みファイルのダウンロードとコピー

- 上の変更を追加済みのurdf_camera.tarファイル(下リンク)をクリックして、~/Downloadsに保存する。
  urdf_camera.tar
- 端末を開いて、次のコマンドで~/catkin_ws/src/create_autonomy/ca_description/tmpフォルダを作成し、urdr_camera.tarをその下にコピーする。
  - `$ mkdir -p ~/catkin_ws/src/create_autonomy/ca_description/tmp`
  - `$ cp ~/Downloads/urdf_camera.tar ~/catkin_ws/src/create_autonomy/ca_description/tmp`

端末で以下のコマンドを実行して、元のファイルを別名で保存し、追加済みファイルへ変更する。

- `$ roscd ca_description/tmp`
- `$ tar xvf urdf_camera.tar`
- `$ cp ../urdf/create_base.xacro ../urdf/create_base.xacro.org`
- `$ cp ../urdf/create_base_gazebo.xacro ../urdf/create_base_gazebo.xacro.org`
- `$ cp ./urdf/create_base.xacro  ../urdf/create_base.xacro`
- `$ cp ./urdf/create_base_gazebo.xacro ../urdf/create_base_gazebo.xacro`

パッケージの作成

次のコマンドでcameraパッケージを作りましょう！

- `$ cd ~/catkin_ws/src/hard2021`
- `$ catkin_create_pkg camera sensor_msgs opencv2 cv_bridge rospy std_msgs`

ソースコードの作成

エディタを開き、上のプログラムをコピペしてcamera.pyというファイル名で~/catkin_ws/src/hard2021/camera/srcディレクトリの中に保存する。以下はgeditを使う場合。

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

ビルド

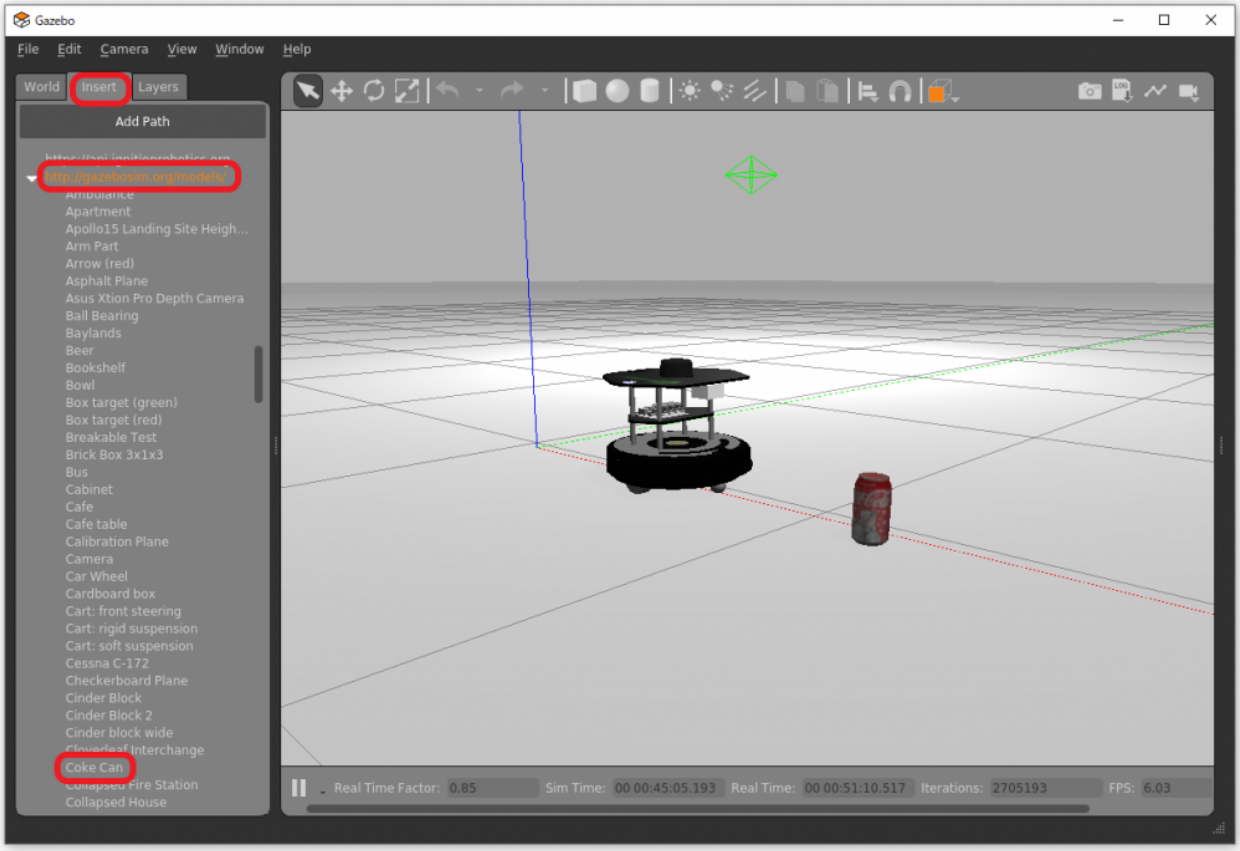
- ROSにパッケージとして認識されるために、以下のコマンドでビルドする。
  - `$ cd ~/catkin_ws`
  - `$ catkin build camera`
- rosrunコマンドで実行できるように実行権を与える。
  - `$ source ~/.bashrc`
  - `$ roscd camera/src`
  - `$ chmod u+x camera.py`

実　行

- 端末を３つ開き、以下のコマンドを実行する。
- １番目の端末：シミュレータの起動
  - rvizを使わないので次のコマンドを実行
    ```
    $ export RVIZ=false
    ```
  - Gazeboの起動
    ```
    $ roslaunch ca_gazebo create_empty_world.launch
    ```
- ２番目の端末。cameraノードの起動
  ```
  $ rosrun camera  camera.py
  ```
- ３番目の端末。遠隔操作
  ```
  $ roslaunch ca_tools keyboard_teleop.launch
  ```

ハンズオン＆ホームワーク

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！
   - 上の「カメラの追加」作業をする。「追加済みファイルのダウンロードとコピー」から行う。
   - 「パッケージの作成」、「ソースコードの作成」、「ビルド」、「実行」を順番にする。
   - 立ち上げたシミュレータにコーラ缶(Coke Can)をロボットの前に挿入する。
     Insert タブ→http://gazbosim.org/models/→Coke Can
   - ロボットをキーボードで操作してコーラ缶の方向に進んだり、その場回転して画像処理が行われていることを確認する。
2. コーラ缶の手前で停止するプログラムを作ろう。なお、カメラでは物体の距離が直接わからないので、コーラ缶の輪郭を検出し、その面積や外接矩形のサイズ[pixel]から推定する必要がある。コーラ缶のサイズはgazeboのGUIで簡単に調べることができる。
3. コーラ缶を任意の位置において、それをロボットが探して近づき、手前で停止するプログラムを作ろう。なお、カメラの位置が比較的高いので近づいてコーラ缶を見失う場合には、urdfファイルを変更して高さを変えるなど工夫しよう。



プチプロジェクト

- 勾配検出フィルタ
  サンプルプログラムではエッジ検出にcanny関数を使用した。その他にも勾配検出フィルタとして、Sobel関数、Laplacian関数などがある。参考リンクのOpenCV-Pythonチュートリアルで調べて、サンプルプログラムを改変して違いを見てみよう。
- 物体の輪郭抽出
  サンプルプログラムではコーラ缶の位置を推定するのに赤領域の重心を使った。より正確にするには物体の輪郭を検出するfindContours関数がある。参考リンクで調べて輪郭を使って物体の位置を推定してみよう。

参考リンク

OpenCV-Pythonチュートリアル

終わり