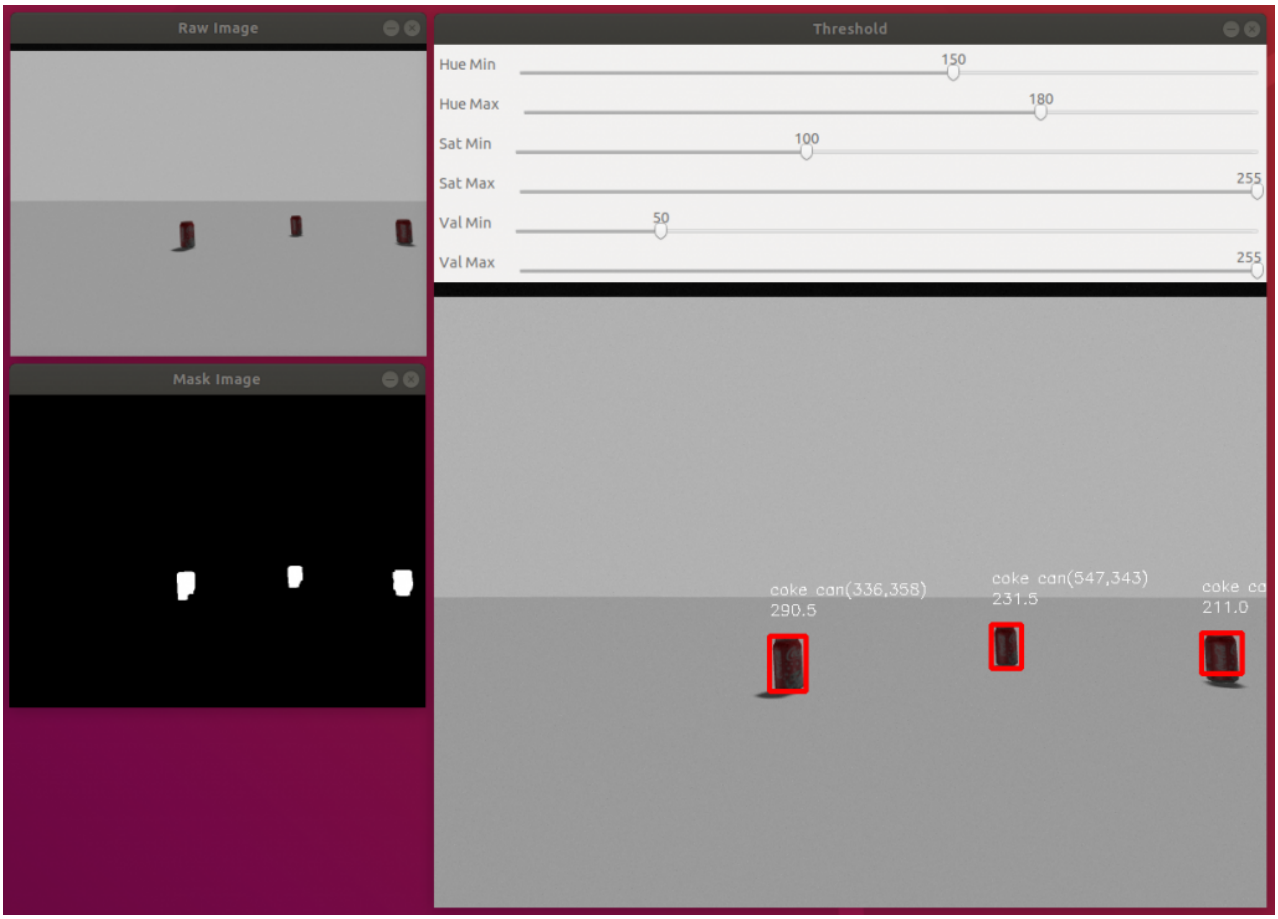
HARD2021：基礎的な物体検出器をPythonで作ろう！

demura.net/robot/hard/20204.html

2021.03.19



この記事はHARD2021（Home AI Robot Development)ワークショップ第３回用です。カメラから取得した画像データをOpenCVを使い画像処理することを学んだので、今回は物体検出のプログラムを作ってみましょう！　現在では、深層学習を使った物体検出の方法がよく使われていますが、ここではそれ以前の基礎的な手法を使います。

サンプルプログラム

このサンプルプログラムの概要を説明します。前回の[HARD2021：OpenCVとPythonプログラムで画像処理をしよう！]ではマスク画像を使い色情報で背景画像を物体を切り分けることで物体１個を検出するサンプルプログラムを学びました。

今回は、同じく色情報を使い複数の物体を検出し、それを面積で識別するサンプルプログラムを紹介します。複数の物体を検出するためには、まず、マスク画像を作り、対象物体の色領域だけの２値化画像を作ります。それに輪郭領域を検出するfindContours()を使い、その色領域を検出します。検出された各色領域に対して、その面積を計算し、ある値以下はノイズとみなすことにより対象物体を検出しています。検出した結果を上図にあるように赤い長方形（バウンディングボックス）で囲み、その上に物体名と位置、面積を表示しています。

さらに、マスク画像を生成するためにHSV表色系の閾値をマウス操作で変更できるトラックバーを実装しています。説明はこのぐらいにして、ソースコードを読んでみましょう

```
#!/usr/bin/env python
# -*- coding: utf-8 -*- # 日本語を使うためのおまじない。
import sys, traceback
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 ObjectDetection:
    # コンストラクタ
    def __init__(self, node_name):
        # ノード名
        self.node_name = node_name

        # 端末に文字列を表示
        rospy.loginfo("%s node", self.node_name)

        # 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、
        # 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)

        # マスク画像を作るためのHSV表色系の閾値
        self.hue_min = 150
        self.hue_max = 180
        self.sat_min = 100
        self.sat_max = 255
        self.val_min =  50
        self.val_max = 255

        # 各種表示用の画像
        self.info_image         = None # 検出情報表示画像
        self.result_image       = None # 検出結果画像
        self.half_raw_image     = None # 生画像のハーフサイズ
        self.half_mask_image    = None # マスク画像のハーフサイズ

        # ウインドウとトラックバーの生成
        self.create_windows_trackbars()

    # 物体検出
    def detection(self):
        # RGB表色系からHSV(Hue:色相,Saturation:彩度,value:明度)表色系に変換
        hsv_image = cv2.cvtColor(self.raw_image, cv2.COLOR_BGR2HSV)

        # しきい値の設定（ここでは赤を抽出）
        color_min = np.array([self.hue_min, self.sat_min, self.val_min])
        color_max = np.array([self.hue_max, self.sat_max, self.val_max])

        # マスク画像を生成
        color_mask = cv2.inRange(hsv_image, color_min, color_max)
```

```
        # 収縮
        kernel = np.ones((5,5), np.uint8) # 5x5 サイズのカ―ネル
        # color_mask = cv2.erode(color_mask, kernel, iterations = 1)

        # 膨張
        color_mask = cv2.dilate(color_mask, kernel, iterations = 2);
        self.half_mask_image = color_mask.copy()

        # 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。
        self.result_image  = cv2.bitwise_and(self.raw_image, self.raw_image, mask
= color_mask)

        # 画像モ―メント関数momentsを使うためグレ―スケ―ル画像へ変換
        gray_image = cv2.cvtColor(self.result_image, cv2.COLOR_BGR2GRAY)


        # 輪郭領域を検出するために２値化する。この例は大津の方法を使っているので閾値は自動計算される。
        # 1番目の引数は2値化する画像。グレ―スケ―ルでなければいけない。2番目の引数は閾値、
        # 3番目の引数は画素の最大値、4番目の引数は処理方法。1番目の出力は閾値、2番目の閾値は２値化画
像。
        rete, binary_image = cv2.threshold(gray_image, 0, 255,cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

        # 輪郭検出：１番目の引数は画像、２番目の引数は輪郭取得モ―ド、３番目は輪郭の概算方法
        # 出力は輪郭(リスト）
        contours = cv2.findContours(binary_image, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)[1]

        # 輪郭のバウンディングボックスを描画
        for i in range(len(contours)):
            # 領域の面積[ピクセル数]
            area = cv2.contourArea(contours[i])
            # print(i,":area=",area)
            if area > 50:
                # バウンディングボックスの取得
                x,y,width,height = cv2.boundingRect(contours[i])
                # バウンディングボックスの描画
                cv2.rectangle(self.result_image, (x,y), (x+width,y+height),
(0,0,255),4)
                cv2.rectangle(self.info_image, (x,y), (x+width,y+height),
(0,0,255),4)
                # 重心の計算
                M = cv2.moments(contours[i])
                cx = int(M['m10']/M['m00'])
                cy = int(M['m01']/M['m00'])
                text = 'coke can('+str(cx) + ',' + str(cy) + ')'
                # 物体名と重心の表示
                cv2.putText(self.info_image,text,(x, y-
40),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,255,255),1)

                # 面積の表示
                cv2.putText(self.info_image,str(area),(x, y-
20),cv2.FONT_HERSHEY_SIMPLEX,0.5, (255,255,255),1)


        # ウインドウサイズの変更
        self.half_raw_image    = cv2.resize(self.raw_image,
(0,0),fx=0.5,fy=0.5)
        self.half_mask_image = cv2.resize(self.half_mask_image,
```

```
(0,0),fx=0.5,fy=0.5)

        # キー入力待
        cv2.waitKey(1)

    # 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)

        if 'cv_image' in locals(): # locals: ローカル変数を辞書形式で返却
            self.raw_image = cv_image

        # 画像生成
        self.info_image = self.raw_image.copy()

        # 物体検出処理
        self.detection()

        # 物体検出結果の表示
        self.show_results()

        # キー入力待
        cv2.waitKey(1)

        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(self.result_image,
"bgr8"))
        except CvBridgeError as e:
            print(e)


    # 結果表示
    def show_results(self):
        if self.info_image is not None:
            cv2.imshow('Threshold', self.info_image)
            # cv2.moveWindow('Threshold',200,600);
        if self.half_raw_image is not None:
            cv2.imshow('Raw Image',self.half_raw_image)
            # cv2.moveWindow('Raw Image',50,50);
        if self.half_mask_image is not None:
            cv2.imshow('Mask Image', self.half_mask_image)
            # cv2.moveWindow('Mask Image',450,50);

    # ウインドウの生成
    def create_windows_trackbars(self):
        cv2.namedWindow('Threshold', cv2.WINDOW_AUTOSIZE)
        cv2.createTrackbar('Hue Min', 'Threshold', self.hue_min, 255,
self.hue_min_cb)
        cv2.createTrackbar('Hue Max', 'Threshold', self.hue_max, 255,
self.hue_max_cb)
        cv2.createTrackbar('Sat Min', 'Threshold', self.sat_min, 255,
self.sat_min_cb)
```

```
        cv2.createTrackbar('Sat Max', 'Threshold', self.sat_max, 255,
self.sat_max_cb)
        cv2.createTrackbar('Val Min', 'Threshold', self.val_min, 255,
self.val_min_cb)
        cv2.createTrackbar('Val Max', 'Threshold', self.val_max, 255,
self.val_max_cb)


    # トラックバーのコールバック関数
    def hue_min_cb(self,val):
        self.hue_min = val

    def hue_max_cb(self,val):
        self.hue_max = val

    def sat_min_cb(self,val):
        self.sat_min = val

    def sat_max_cb(self,val):
        self.sat_max = val

    def val_min_cb(self,val):
        self.val_min = val

    def val_max_cb(self,val):
        self.val_max = val

if __name__ == '__main__':
    try:
        node_name = 'object_detection'
        rospy.init_node(node_name, anonymous=True)
        ObjectDetection(node_name)
        rospy.spin()

    except KeyboardInterrupt:
        print("Shutting down")
        cv2.destroyAllWindows()
```

カメラの追加

create_autonomyのロボットモデルにカメラモデルを追加されている必要があります。
追加していない方は以下のリンクの「カメラの追加」を実施してください。

HARD2021：OpenCVとPythonプログラムでCameraを使ってみよう！

パッケージの作成

次のコマンドでobject_detectionパッケージを作りましょう！

- `$ cd ~/catkin_ws/src/hard2021`
- `$ catkin_create_pkg object_detection 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object_detection.pyというファイル名で~/catkin_ws/src/hard2021/object_detection/srcディレクトリの中に保存する。以下はgeditを使う場合。エディタはお好きなものをどうぞ。

- `$ cd ~/catkin_ws/src/hard2021/object_detection/src`
- `$ gedit object_detection.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 object_detection`
- rosrunコマンドで実行できるように実行権を与える。
  - `$ source ~/.bashrc`
  - `$ roscd object_detection/src`
  - `$ chmod u+x object_detection.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。object_detectionノードの起動

  `$ rosrun object_detection object_detection.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

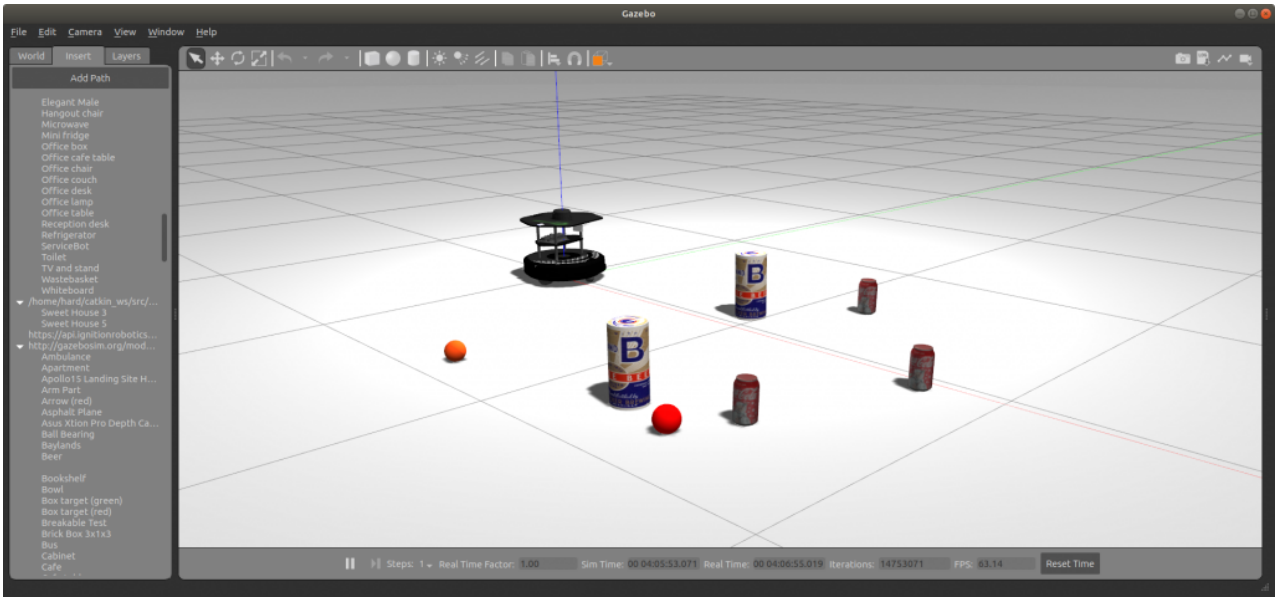
1. **サンプルの実行**
   - 上の説明に従ってサンプルプログラムを実行して動作を確認しよう！
   - 上の「カメラの追加」作業をしていない場合はする。
   - 「パッケージの作成」、「ソースコードの作成」、「ビルド」、「実行」を順番にする。
   - 立ち上げたシミュレータにコーラ缶(Coke Can)を3個、ロボットの前に挿入する。

     次の操作をGazeboで3回繰り返す。

     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. **閾値調整**
    - このサンプルプログラムではマスク画像を生成する閾値をトラックバーで操作できる。操作をして、Thresholdウインドウの画面に表示されているバウンディングボックスの面積[ピクセル数]の値を概ね最大になるように設定してみよう。
    - トラックバー操作で閾値を決める作業が半自動化されたが、これを完全に自動化したい。いろいろな方法が考えるが、マウス操作で対象物体領域の全画像を取得し、そのデータを元に閾値を決める方法がある。HSV表色系ではVは明度 (value, brightness)なので、明るさにロバスト (頑強)なアプリケーションを作るために、H (Heu、色相)とS (Saturation、彩度）だけを使う場合も多い。
3. **収縮･膨張**

    61行目と64行目に収縮と膨張の関数がある。一番最後の引数iterationsは繰り返し回数である。この値を変更したり、収縮と膨張の順番を変えるなどしてマスク画像がどう変化観察しよう。なお、収縮･膨張処理は環境と物体により多きく変わる。
4. **物体識別**
    - このサンプルプログラムではコーラ缶だけを検出した。下図のようにコーラ(Coke can)3缶の他に赤いクリケットボール(Cricket ball)2個、ビール(Beer)2缶を追加して、物体識別のプログラムに拡張してみよう。なお、各物体の追加はコーラ缶と同様である。
        - Insert タブ→http://gazebosim.org/models/→Beer
        - Insert タブ→http://gazebosim.org/models/→Cricket ball
    - ヒント
        - 下図を見るとわかるが、クリケットボールもコーラ缶も赤なので色での識別は難しそうである。形状が違うので画像モーメント、真円度やハフ変換などが使えるだろう。
        - 別の方法として、カラーヒストグラムにより識別もある。OpenCVのAPIを使うと簡単にできるので試してみてはいかがでしょう。



終わり