demo

July 7, 2020

[1]:
```
import sys
sys.path.append("/home/arijin/dataset/KITTI/kitti_tool")

import os
DATASET_ROOT_PATH = '/home/arijin/dataset/KITTI'
CALIB_PATH = os.path.join(DATASET_ROOT_PATH, 'object', 'training', 'calib')
IMG_LEFT_PATH = os.path.join(DATASET_ROOT_PATH, 'object', 'training', 'image_2')
IMG_RIGHT_PATH = os.path.join(DATASET_ROOT_PATH, 'object', 'training',␣
 ↪'image_3')

import time
import numpy as np
import cv2
from matplotlib import pyplot as plt
import kitti_util_copy as kitti
```

[2]:
```
#    sift            bbox      bbox bbox detect
# arg kp1, kp2, goods   sift.sift output
#      detection1, detection2     detect   list
# output new_goods
#         cost     bbox       cost    M*N M  detect  bbox  M  detect  bbox
#         good_in_detection    cost
def isgoodinbbox(kp1, kp2, goods, detection1, detection2):
    new_goods = []
    N = len(detection1)
    M = len(detection2)
    cost = np.zeros(shape=(N, M))   # Cost matrix
    good_in_detection = []
    for i in range(N):
        temp = []
        for j in range(M):
            temp.append([])
        good_in_detection.append(temp)
    for good in goods:
        left, right = -1, -1
        for i, detection in enumerate(detection1):
```

```
            xmin, ymin, xmax, ymax = detection[2][0], detection[2][1],␣
↪detection[2][2], detection[2][3]
            if kp1[good[0].queryIdx].pt[0]>xmin and \
               kp1[good[0].queryIdx].pt[0]<xmax and \
               kp1[good[0].queryIdx].pt[1]>ymin and \
               kp1[good[0].queryIdx].pt[1]<ymax:
                left = i
                break
        for j, detection in enumerate(detection2):
            xmin, ymin, xmax, ymax = detection[2][0], detection[2][1],␣
↪detection[2][2], detection[2][3]
            if kp2[good[0].trainIdx].pt[0]>xmin and \
               kp2[good[0].trainIdx].pt[0]<xmax and \
               kp2[good[0].trainIdx].pt[1]>ymin and \
               kp2[good[0].trainIdx].pt[1]<ymax:
                right = j
                break
        if left>=0 and right>=0:
            new_goods.append(good)
            good_in_detection[left][right].append(good[0])
            cost[left][right] += 1
    return new_goods, cost, good_in_detection

#
# kp1 kp2 cv::keypoint
# P1, P2            3*4
def Triangulate(kp1, kp2, P1, P2):
    P1_row0 = np.array(calib.P[0])
    P1_row1 = np.array(calib.P[1])
    P1_row2 = np.array(calib.P[2])
    P2_row0 = np.array(calib.P2[0])
    P2_row1 = np.array(calib.P2[1])
    P2_row2 = np.array(calib.P2[2])
    A_row0 = kp1.pt[0] * P1_row2 - P1_row0
    A_row1 = kp1.pt[1] * P1_row2 - P1_row1
    A_row2 = kp2.pt[0] * P2_row2 - P2_row0
    A_row3 = kp2.pt[1] * P2_row2 - P2_row1
    A = np.vstack([A_row0, A_row1, A_row2, A_row3])
    U, S, Vh = np.linalg.svd(A)
    P = Vh[-1,:]
    P = P/P[3]
    return P[:3]
```

[3]:
```
import darknet.darknet as darknet
import detect
import sift
```

```
cv2.destroyAllWindows()

#
frame = 8
img_left_file = '{:0>6d}.png'.format(frame)
img1 = cv2.imread(os.path.join(IMG_LEFT_PATH, img_left_file))  # queryImage
img_right_file = '{:0>6d}.png'.format(frame)
img2 = cv2.imread(os.path.join(IMG_RIGHT_PATH, img_right_file))  # trainImage

# yolo
print('loading...')
netMain, metaMain = detect.yolo_initialize()
print('finish.')

prev_time = time.time()
#  opencv sift          sift.py
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
#  sift.py
kp1, kp2, goods = sift.sift(gray1, gray2)
pts1, pts2 = cv2.KeyPoint_convert(kp1), cv2.KeyPoint_convert(kp2)
print(time.time()-prev_time)

# detect yolo
# Create an image we reuse for each detect
darknet_image = darknet.make_image(darknet.network_width(netMain),
                                   darknet.network_height(netMain),3)
# detect1  detect bbox          detections1 bbox
frame_read = img1.copy()
frame_rgb = cv2.cvtColor(frame_read, cv2.COLOR_BGR2RGB)
frame_resized = cv2.resize(frame_rgb,
                           (darknet.network_width(netMain),
                            darknet.network_height(netMain)),
                           interpolation=cv2.INTER_LINEAR)

darknet.copy_image_from_bytes(darknet_image,frame_resized.tobytes())

detections = darknet.detect_image(netMain, metaMain, darknet_image, thresh=0.45)
shape1 = frame_read.shape
shape2 = (darknet.network_width(netMain), darknet.network_height(netMain), 3)
detections1 = detect.convertBack(detections, shape2, shape1)

# detect2  detect bbox
frame_read = img2.copy()
frame_rgb = cv2.cvtColor(frame_read, cv2.COLOR_BGR2RGB)
frame_resized = cv2.resize(frame_rgb,
                           (darknet.network_width(netMain),
```

```
                                darknet.network_height(netMain)),
                               interpolation=cv2.INTER_LINEAR)

darknet.copy_image_from_bytes(darknet_image,frame_resized.tobytes())

detections = darknet.detect_image(netMain, metaMain, darknet_image, thresh=0.45)
shape1 = frame_read.shape
shape2 = (darknet.network_width(netMain), darknet.network_height(netMain), 3)
detections2 = detect.convertBack(detections, shape2, shape1)

# filter
new_goods, cost, match_in_detection = isgoodinbbox(kp1, kp2, goods,␣
 ↪detections1, detections2)
print(cost)
print(len(kp1), len(kp2), len(goods), len(new_goods))

# distance measurement
#         bbox
#  bbox
#
calib_file = '{:0>6d}.txt'.format(frame)
calib = kitti.Calibration(os.path.join(CALIB_PATH, calib_file))
Pos_for_detections = []
for i in range(cost.shape[0]):
    j = np.argmax(cost[i])  #     bbox           bbox
    goods_in_detection = match_in_detection[i][j]
    pts_Pos = []
    for good_in_detection in goods_in_detection:
        pt_Pos = Triangulate(kp1[good_in_detection.queryIdx],␣
 ↪kp2[good_in_detection.trainIdx], calib.P, calib.P2)
        pts_Pos.append(pt_Pos)
    pts_Pos = np.array(pts_Pos)
    Pos = np.mean(pts_Pos, axis=0)
    print(Pos)
    Pos_for_detections.append(Pos)

print(time.time()-prev_time)

# visualize
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
image = detect.cvDrawBoxes_on_origin_img(detections1, Pos_for_detections, img1)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
cv2.imshow('Demo1', image)
cv2.waitKey(2000)

# img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,new_goods[:], None, flags=2)
# plt.imshow(img3),plt.show()
```

```
# cv2.imshow("MatchDemo", img3)
# cv2.waitKey(10000)
# cv2.destroyAllWindows()
```

```
loading…
finish.
0.5999557971954346
[[79.  3.  0.  0.  3.  0.  0.  0.  0.  0.  0.]
 [ 0. 30.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 1.  0.  0.  0. 21.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. 17.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. 50.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. 19.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  3.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  5.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  3.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  4.  0.]]
4500 4147 1054 238
[-1.33922072  0.7777092   7.50109797]
[ 1.22150064  0.86190286 13.81768853]
[ 1.0743857   0.18483196 -1.19765894]
[ 7.28444849  0.91495633 33.93589222]
[3.97484733 1.06493194 5.58433077]
[ 8.09458954  1.10579471 19.50097075]
[23.7874704   0.29805053 83.78048633]
[10.70812233  1.28158053 41.71651452]
[24.69148952  0.72290541 70.37359817]
[ 34.78831572  -0.32630121 110.2948883 ]
0.8379285335540771
```

[3]: -1