**คุณลักษณะการจับคู่ + การเขียนโฮมเวิร์คเพื่อค้นหาวัตถุ**

เป้าหมาย

ในบทนี้,

* เราจะผสมผสานการจับคู่คุณลักษณะและ findHomography จากโมดูล calib3d เพื่อค้นหาวัตถุที่เป็นที่รู้จักในภาพที่ซับซ้อน

พื้นฐาน

ดังนั้นสิ่งที่เราทำในเซสชั่นล่าสุด? เราใช้ queryImage พบจุดที่น่าสนใจบางส่วนในส่วนนี้เราได้นำภาพ TrainImage อื่นมาค้นพบคุณลักษณะในภาพนั้นด้วยและเราพบการจับคู่ที่ดีที่สุดในหมู่พวกเขา ในระยะสั้นเราพบตำแหน่งของบางส่วนของวัตถุในภาพที่รกอีก ข้อมูลนี้เพียงพอสำหรับการค้นหาวัตถุบนรถไฟ INImage

สำหรับการที่เราสามารถใช้ฟังก์ชั่นจากโมดูล calib3d คือcv2.findHomography () ถ้าเราผ่านชุดของจุดจากทั้งสองภาพจะได้พบกับการเปลี่ยนแปลงอย่างต่อเนื่องของวัตถุนั้น จากนั้นเราจะใช้cv2.perspectiveTransform ()เพื่อค้นหาวัตถุ ต้องมีจุดแก้ไข 4 จุดเพื่อหาการเปลี่ยนแปลง

เราได้เห็นว่าอาจมีข้อผิดพลาดที่เป็นไปได้ในขณะที่จับคู่ซึ่งอาจส่งผลต่อผลลัพธ์ เพื่อแก้ปัญหานี้อัลกอริทึมจะใช้ RANSAC หรือ LEAST\_MEDIAN (ซึ่งสามารถตัดสินใจได้จากธง) การจับคู่ที่ดีเพื่อให้การประมาณค่าที่ถูกต้องเรียกว่า inliers และส่วนที่เหลือเรียกว่าค่าผิดปกติ cv2.findHomography ()จะคืนค่ามาสก์ที่ระบุจุดเริ่มต้นและค่าผิดปกติ

ลองมาทำกันเถอะ !!!

รหัส

ขั้นแรกให้ตามปกติลองหาคุณลักษณะ SIFT ในรูปภาพและใช้การทดสอบอัตราส่วนเพื่อค้นหาการจับคู่ที่ดีที่สุด

**import** **numpy** **as** **np**

**import** **cv2**

**from** **matplotlib** **import** pyplot **as** plt

MIN\_MATCH\_COUNT = 10

img1 = cv2.imread('box.png',0) *# queryImage*

img2 = cv2.imread('box\_in\_scene.png',0) *# trainImage*

*# Initiate SIFT detector*

sift = cv2.SIFT()

*# find the keypoints and descriptors with SIFT*

kp1, des1 = sift.detectAndCompute(img1,None)

kp2, des2 = sift.detectAndCompute(img2,None)

FLANN\_INDEX\_KDTREE = 0

index\_params = dict(algorithm = FLANN\_INDEX\_KDTREE, trees = 5)

search\_params = dict(checks = 50)

flann = cv2.FlannBasedMatcher(index\_params, search\_params)

matches = flann.knnMatch(des1,des2,k=2)

*# store all the good matches as per Lowe's ratio test.*

good = []

**for** m,n **in** matches:

**if** m.distance < 0.7\*n.distance:

good.append(m)

ตอนนี้เราตั้งเงื่อนไขว่าจะมีการจับคู่อย่างน้อย 10 รายการ (กำหนดโดย MIN\_MATCH\_COUNT) เพื่อหาวัตถุ มิเช่นนั้นจะแสดงข้อความว่ามีการแข่งขันไม่เพียงพอ

หากพบการจับคู่ที่เพียงพอเราจะแยกตำแหน่งของคีย์พอยต์ที่จับคู่กันในทั้งสองภาพ พวกเขาจะถูกส่งผ่านไปหาการเปลี่ยนแปลงอย่างต่อเนื่อง เมื่อเราได้รับเมทริกซ์การเปลี่ยนแปลง 3x3 นี้แล้วเราจะใช้มันเพื่อเปลี่ยนมุมของ queryImage ไปยังจุดที่สอดคล้องกันใน trainImage จากนั้นเราวาดมัน

**if** len(good)>MIN\_MATCH\_COUNT:

src\_pts = np.float32([ kp1[m.queryIdx].pt **for** m **in** good ]).reshape(-1,1,2)

dst\_pts = np.float32([ kp2[m.trainIdx].pt **for** m **in** good ]).reshape(-1,1,2)

M, mask = cv2.findHomography(src\_pts, dst\_pts, cv2.RANSAC,5.0)

matchesMask = mask.ravel().tolist()

h,w = img1.shape

pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape(-1,1,2)

dst = cv2.perspectiveTransform(pts,M)

img2 = cv2.polylines(img2,[np.int32(dst)],True,255,3, cv2.LINE\_AA)

**else**:

**print** "Not enough matches are found - *%d*/*%d*" % (len(good),MIN\_MATCH\_COUNT)

matchesMask = None

สุดท้ายเราวาด inliers ของเรา (ถ้าประสบความสำเร็จพบวัตถุ) หรือจับคู่ keypoints (ถ้าล้มเหลว)

draw\_params = dict(matchColor = (0,255,0), *# draw matches in green color*

singlePointColor = None,

matchesMask = matchesMask, *# draw only inliers*

flags = 2)

img3 = cv2.drawMatches(img1,kp1,img2,kp2,good,None,\*\*draw\_params)

plt.imshow(img3, 'gray'),plt.show()

ดูผลลัพธ์ด้านล่าง วัตถุถูกทำเครื่องหมายด้วยสีขาวในภาพรก:

