**การจับคู่คุณลักษณะ**

เป้าหมาย

ในบทนี้

* เราจะดูวิธีการจับคู่คุณลักษณะในภาพเดียวกับคนอื่น ๆ
* เราจะใช้ตัวจับคู่แบบ Brute-Force และ FLANN Matcher ใน OpenCV

พื้นฐานของ Brute-Force Matcher

Matcher Brute-Force เป็นเรื่องง่าย ใช้คำอธิบายคุณลักษณะหนึ่งชุดในชุดแรกและจับคู่กับคุณลักษณะอื่น ๆ ทั้งหมดในชุดที่สองโดยใช้การคำนวณระยะทางบางอย่าง และที่ใกล้เคียงที่สุดจะถูกส่งกลับ

สำหรับ BF จับคู่แรกที่เราต้องสร้างวัตถุ BFMatcher โดยใช้cv2.BFMatcher () ใช้พารามิเตอร์ params สองตัว คนแรกคือnormType ระบุการวัดระยะทางที่จะใช้ โดยค่าเริ่มต้นมันเป็นcv2.NORM\_L2 เป็นเรื่องที่ดีสำหรับ SIFT, SURF เป็นต้น ( cv2.NORM\_L1ยังมี) สำหรับตัวบอกสตริงแบบไบนารีเช่น ORB BRIEF BRISK ฯลฯcv2.NORM\_HAMMINGควรใช้ซึ่งใช้ระยะทาง Hamming เป็นค่า หากลูกโลกใช้WTA\_K == 3หรือ 4 , cv2.NORM\_HAMMING2ควรจะใช้

ค่าที่สองคือตัวแปรแบบบูลีนcrossCheckซึ่งเป็นค่าเท็จโดยค่าเริ่มต้น ถ้าเป็นจริง Matcher จะส่งคืนเฉพาะค่าที่ตรงกับค่า (i, j) เช่น descriptor i-th ในชุด A มีตัวอธิบาย j-th ในชุด B เป็นคำที่ตรงที่สุดและในทางกลับกัน นั่นคือทั้งสองคุณสมบัติในทั้งสองชุดควรจับคู่กัน ให้ผลสม่ำเสมอและเป็นทางเลือกที่ดีในการทดสอบอัตราส่วนที่เสนอโดย D.Lowe ในกระดาษ SIFT

เมื่อมันถูกสร้างขึ้นสองวิธีที่สำคัญคือBFMatcher.match ()และBFMatcher.knnMatch () คนแรกให้ผลลัพธ์ที่ดีที่สุด วิธีที่สองผลตอบแทนkแข่งขันที่ดีที่สุดที่ k จะถูกระบุโดยผู้ใช้ มันอาจเป็นประโยชน์เมื่อเราจำเป็นต้องทำงานเพิ่มเติมเกี่ยวกับที่

เช่นเดียวกับที่เราใช้ cv2.drawKeypoints () เพื่อวาดคีย์พอยน์เตอร์cv2.drawMatches ()ช่วยให้เราสามารถจับคู่ จัดเรียงภาพสองภาพในแนวนอนและวาดเส้นจากภาพแรกไปยังภาพที่สองซึ่งแสดงถึงการจับคู่ที่ดีที่สุด นอกจากนี้ยังมีcv2.drawMatchesKnnที่ใช้จับคู่ที่ดีที่สุด k ถ้า k = 2 จะวาดเส้นตรง 2 เส้นสำหรับแต่ละคีย์พอยต์ ดังนั้นเราจะต้องผ่านหน้ากากถ้าเราต้องการที่จะเลือกดึงมัน

ลองดูตัวอย่างสำหรับแต่ละ SURF และ ORB (ทั้งสองใช้การวัดระยะทางที่ต่างกัน)

Brute-Force การจับคู่กับ ORB Descriptors

ที่นี่เราจะเห็นตัวอย่างง่ายๆในการจับคู่คุณลักษณะระหว่างสองรูป ในกรณีนี้ฉันมี QueryImage และ TrainImage เราจะพยายามค้นหา queryImage ใน trainImage โดยใช้การจับคู่คุณลักษณะ (ภาพเป็น/samples/c/box.pngและ/samples/c/box\_in\_scene.png )

เราใช้ SIFT descriptors เพื่อให้ตรงกับคุณลักษณะ เริ่มต้นด้วยการโหลดภาพค้นหา descriptors เป็นต้น

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

**import** **cv2**

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

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

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

*# Initiate SIFT detector*

orb = cv2.ORB()

*# find the keypoints and descriptors with SIFT*

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

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

ต่อไปเราจะสร้างวัตถุ BFMatcher ด้วยการวัดระยะทางcv2.NORM\_HAMMING (เนื่องจากเราใช้ ORB) และcrossCheckจะเปิดขึ้นเพื่อผลลัพธ์ที่ดียิ่งขึ้น จากนั้นเราใช้เมธอด Matcher.match () เพื่อให้ได้ผลลัพธ์ที่ดีที่สุดในสองภาพ เราจัดเรียงลำดับตามระยะทางที่ห่างออกไปเพื่อให้การจับคู่ที่ดีที่สุด (โดยมีระยะทางต่ำ) มาถึงด้านหน้า จากนั้นเราวาดเฉพาะ 10 คำแรกเท่านั้น (เพื่อความชัดเจนคุณสามารถเพิ่มได้ตามต้องการ)

*# create BFMatcher object*

bf = cv2.BFMatcher(cv2.NORM\_HAMMING, crossCheck=True)

*# Match descriptors.*

matches = bf.match(des1,des2)

*# Sort them in the order of their distance.*

matches = sorted(matches, key = **lambda** x:x.distance)

*# Draw first 10 matches.*

img3 = cv2.drawMatches(img1,kp1,img2,kp2,matches[:10], flags=2)

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

ด้านล่างนี้เป็นผลที่ฉันได้รับ:



Matcher Object คืออะไร?

ผลลัพธ์ของการจับคู่ = bf.match (des1, des2)คือรายการของออบเจกต์ DMatch ออบเจ็กต์ DMatch นี้มีแอตทริบิวต์ต่อไปนี้

* DMatch.distance - ระยะห่างระหว่าง descriptor ที่ต่ำกว่าที่ดีกว่าก็คือ
* DMatch.trainIdx - ดัชนีของ descriptor ใน descriptors รถไฟ
* DMatch.queryIdx - ดัชนีของ descriptor ใน descriptor แบบสอบถาม
* DMatch.imgIdx - ดัชนีภาพของรถไฟ

Brute-Force การจับคู่กับ SIFT Descriptors และ Ratio Test

ในครั้งนี้เราจะใช้BFMatcher.knnMatch ()เพื่อให้ได้ผลลัพธ์ที่ดีที่สุด ในตัวอย่างนี้เราจะใช้ k = 2 เพื่อให้เราสามารถใช้การทดสอบอัตราส่วนที่ D.Lowe อธิบายไว้ในเอกสารของเขา

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

**import** **cv2**

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

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)

*# BFMatcher with default params*

bf = cv2.BFMatcher()

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

*# Apply ratio test*

good = []

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

**if** m.distance < 0.75\*n.distance:

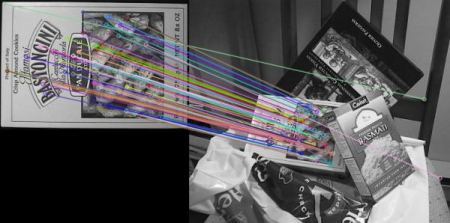
good.append([m])

*# cv2.drawMatchesKnn expects list of lists as matches.*

img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,good,flags=2)

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

ดูผลลัพธ์ด้านล่าง:



FLANN ตาม Matcher

FLANN ย่อมาจาก Fast Library สำหรับเพื่อนบ้านใกล้เคียงที่สุด มีชุดของอัลกอริทึมที่เหมาะสำหรับการค้นหาเพื่อนบ้านที่รวดเร็วในชุดข้อมูลขนาดใหญ่และสำหรับคุณลักษณะด้านมิติสูง ทำงานได้เร็วกว่า BFMatcher สำหรับชุดข้อมูลขนาดใหญ่ เราจะเห็นตัวอย่างที่สองกับ matcher แบบ FLANN

สำหรับ matcher ตาม FLANN เราจำเป็นต้องผ่านพจนานุกรมสองตัวที่ระบุถึงอัลกอริทึมที่จะใช้พารามิเตอร์ที่เกี่ยวข้องอื่น ๆ ตัวอย่างแรกคือ IndexParams สำหรับอัลกอริทึมต่างๆข้อมูลที่จะส่งจะอธิบายในเอกสาร FLANN ในฐานะสรุปสำหรับอัลกอริทึมเช่น SIFT, SURF เป็นต้นคุณสามารถส่งต่อไปนี้:

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

ขณะใช้ ORB คุณสามารถส่งต่อไปนี้ได้ ค่าแนะนำที่แนะนำตามเอกสาร แต่ไม่ได้ให้ผลลัพธ์ที่จำเป็นในบางกรณี ค่าอื่น ๆ ทำงานได้ดี:

index\_params= dict(algorithm = FLANN\_INDEX\_LSH,

table\_number = 6, *# 12*

key\_size = 12, *# 20*

multi\_probe\_level = 1) *#2*

พจนานุกรมที่สองคือ SearchParams ระบุจำนวนครั้งที่ต้นไม้ในดัชนีควรผ่านการสำรวจซ้ำ ค่าที่สูงขึ้นจะให้ความแม่นยำที่ดีขึ้น แต่ต้องใช้เวลามากขึ้น หากคุณต้องการที่จะเปลี่ยนค่าผ่านsearch\_params = Dict (ตรวจสอบ = 100)

ด้วยข้อมูลเหล่านี้เราจะไปได้ดี

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

**import** **cv2**

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

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 parameters*

FLANN\_INDEX\_KDTREE = 0

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

search\_params = dict(checks=50) *# or pass empty dictionary*

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

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

*# Need to draw only good matches, so create a mask*

matchesMask = [[0,0] **for** i **in** xrange(len(matches))]

*# ratio test as per Lowe's paper*

**for** i,(m,n) **in** enumerate(matches):

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

matchesMask[i]=[1,0]

draw\_params = dict(matchColor = (0,255,0),

singlePointColor = (255,0,0),

matchesMask = matchesMask,

flags = 0)

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

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

ดูผลลัพธ์ด้านล่าง:

