Automatic Panoramic Image Stitching

大學部資工三 4110056030 鄭詠謙

使用語言: python

輸入測試圖片:圖片自攝



1.jpg



2.jpg



3.jpg

全景圖處理步驟:

* 建立Image\_Stitching()物件
  + 建構子部分定義

self.ratio：匹配時的特徵比率

self.min\_match：最小的匹配數量當匹配到的特徵點數量超過這個值時，才會執行圖像的拼接

self.sift：創建了一個 SIFT 特徵檢測器，是用於檢測圖像中關鍵特徵的演算法，能夠檢測到圖像中的旋轉、縮放和亮度變化不變的特徵點

self.smoothing\_window\_size：設置合併過程中的平滑窗口大小，指定了遮罩中平滑過渡的範圍大小

class Image\_Stitching():

    def \_\_init\_\_(self) :

        self.ratio=0.85

        self.min\_match=10

        self.sift=cv2.xfeatures2d.SIFT\_create()

        self.smoothing\_window\_size=800

* Registration function是用來找出圖片的對應關係
  + sift.detectAndCompute:SIFT 特徵檢測器檢測兩個圖像中的keypoints和descriptors
  + cv2.BFMatcher():使用Brute-Force Matcher
  + matcher.knnMatch：使用Brute-Force匹配器對兩個圖像的descriptors進行匹配，k=2 表示返回每個特徵點的兩個最佳匹配
  + cv2.drawMatchesKnn:繪製匹配點將之可視化
  + cv2.findHomography():計算兩個圖像之間的透視變換矩陣。使用了 RANSAC（Random Sample Consensus）

def registration(self,img1,img2,count):

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

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

        matcher = cv2.BFMatcher()

        raw\_matches = matcher.knnMatch(des1, des2, k=2)

        good\_points = []

        good\_matches=[]

        for m1, m2 in raw\_matches:

            if m1.distance < self.ratio \* m2.distance:

                good\_points.append((m1.trainIdx, m1.queryIdx))

                good\_matches.append([m1])

        img3 = cv2.drawMatchesKnn(img1, kp1, img2, kp2, good\_matches, None, flags=2)

        cv2.imwrite(f'museum/matching{count}.jpg', img3)

        if len(good\_points) > self.min\_match:

            image1\_kp = np.float32(

                [kp1[i].pt for (\_, i) in good\_points])

            image2\_kp = np.float32(

                [kp2[i].pt for (i, \_) in good\_points])

            H, status = cv2.findHomography(image2\_kp, image1\_kp, cv2.RANSAC,5.0)

        return H



matching1.jpg(第一張與第二張的特徵點)



matching2.jpg(第三張與第一&二張的特徵點)

* create\_mask是用來建立遮罩
  + barrier：計算第一張圖像的右邊界，也就是融合區域的起始位置。
  + mask[:, barrier + offset:] = 1：這一行程式將融合區域之外的部分設置為完全可見。
  + return cv2.merge([mask, mask, mask])：將遮罩合併

def create\_mask(self,img1,img2,version):

        height\_img1 = img1.shape[0]

        width\_img1 = img1.shape[1]

        width\_img2 = img2.shape[1]

        height\_panorama = height\_img1

        width\_panorama = width\_img1 +width\_img2

        offset = int(self.smoothing\_window\_size / 2)

        barrier = img1.shape[1] - int(self.smoothing\_window\_size / 2)

        mask = np.zeros((height\_panorama, width\_panorama))

        if version== 'left\_image':

            mask[:, barrier - offset:barrier + offset ] = np.tile(np.linspace(1, 0, 2 \* offset ).T, (height\_panorama, 1))

            mask[:, :barrier - offset] = 1

        else:

            mask[:, barrier - offset :barrier + offset ] = np.tile(np.linspace(0, 1, 2 \* offset ).T, (height\_panorama, 1))

            mask[:, barrier + offset:] = 1

        return cv2.merge([mask, mask, mask])

* blending function將兩個圖像進行融合，生成全景圖
  + cv2.warpPerspective: 將第二張圖像根據透視變換矩陣H進行透視變換，然後應用融合遮罩，再將不需要的部分設為透明
  + np.where(result[:, :, 0] != 0): 找到全景圖中非零pixel的行和列
  + final\_result = result[min\_row:max\_row, min\_col:max\_col, :]：根據計算出的最小和最大行列範圍，將全景圖剪裁，得到最終的全景圖

def blending(self,img1,img2,count):

        H = self.registration(img1,img2,count)

        height\_img1 = img1.shape[0]

        width\_img1 = img1.shape[1]

        width\_img2 = img2.shape[1]

        height\_panorama = height\_img1

        width\_panorama = width\_img1 +width\_img2

        panorama1 = np.zeros((height\_panorama, width\_panorama, 3))

        mask1 = self.create\_mask(img1,img2,version='left\_image')

        panorama1[0:img1.shape[0], 0:img1.shape[1], :] = img1

        panorama1 \*= mask1

        mask2 = self.create\_mask(img1,img2,version='right\_image')

        panorama2 = cv2.warpPerspective(img2, H, (width\_panorama, height\_panorama))\*mask2

        result=panorama1+panorama2

        rows, cols = np.where(result[:, :, 0] != 0)

        min\_row, max\_row = min(rows), max(rows) + 1

        min\_col, max\_col = min(cols), max(cols) + 1

        final\_result = result[min\_row:max\_row, min\_col:max\_col, :]

        return final\_result



panorama.jpg(最終全景圖)

* main即輸入輸出控制
  + 可以一次input 3張各自有部分重疊的圖片

def main(argv1,argv2,argv3):

    img1 = cv2.imread(argv1)

    img2 = cv2.imread(argv2)

    img3 = cv2.imread(argv3)

    count = 1

    temp=Image\_Stitching().blending(img1,img2,count)

    cv2.imwrite('museum/temp.jpg', temp)

    count += 1

    temp = cv2.imread('museum/temp.jpg')

    final=Image\_Stitching().blending(temp,img3,count)

    cv2.imwrite('museum/panorama.jpg', final)

main('museum/1.jpg','museum/2.jpg','museum/3.jpg')

完整程式碼:

import cv2

import numpy as np

import sys

class Image\_Stitching():

    def \_\_init\_\_(self) :

        self.ratio=0.85

        self.min\_match=10

        self.sift=cv2.xfeatures2d.SIFT\_create()

        self.smoothing\_window\_size=800

    def registration(self,img1,img2,count):

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

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

        matcher = cv2.BFMatcher()

        raw\_matches = matcher.knnMatch(des1, des2, k=2)

        good\_points = []

        good\_matches=[]

        for m1, m2 in raw\_matches:

            if m1.distance < self.ratio \* m2.distance:

                good\_points.append((m1.trainIdx, m1.queryIdx))

                good\_matches.append([m1])

        img3 = cv2.drawMatchesKnn(img1, kp1, img2, kp2, good\_matches, None, flags=2)

        cv2.imwrite(f'museum/matching{count}.jpg', img3)

        if len(good\_points) > self.min\_match:

            image1\_kp = np.float32(

                [kp1[i].pt for (\_, i) in good\_points])

            image2\_kp = np.float32(

                [kp2[i].pt for (i, \_) in good\_points])

            H, status = cv2.findHomography(image2\_kp, image1\_kp, cv2.RANSAC,5.0)

        return H

    def create\_mask(self,img1,img2,version):

        height\_img1 = img1.shape[0]

        width\_img1 = img1.shape[1]

        width\_img2 = img2.shape[1]

        height\_panorama = height\_img1

        width\_panorama = width\_img1 +width\_img2

        offset = int(self.smoothing\_window\_size / 2)

        barrier = img1.shape[1] - int(self.smoothing\_window\_size / 2)

        mask = np.zeros((height\_panorama, width\_panorama))

        if version== 'left\_image':

            mask[:, barrier - offset:barrier + offset ] = np.tile(np.linspace(1, 0, 2 \* offset ).T, (height\_panorama, 1))

            mask[:, :barrier - offset] = 1

        else:

            mask[:, barrier - offset :barrier + offset ] = np.tile(np.linspace(0, 1, 2 \* offset ).T, (height\_panorama, 1))

            mask[:, barrier + offset:] = 1

        return cv2.merge([mask, mask, mask])

    def blending(self,img1,img2,count):

        H = self.registration(img1,img2,count)

        height\_img1 = img1.shape[0]

        width\_img1 = img1.shape[1]

        width\_img2 = img2.shape[1]

        height\_panorama = height\_img1

        width\_panorama = width\_img1 +width\_img2

        panorama1 = np.zeros((height\_panorama, width\_panorama, 3))

        mask1 = self.create\_mask(img1,img2,version='left\_image')

        panorama1[0:img1.shape[0], 0:img1.shape[1], :] = img1

        panorama1 \*= mask1

        mask2 = self.create\_mask(img1,img2,version='right\_image')

        panorama2 = cv2.warpPerspective(img2, H, (width\_panorama, height\_panorama))\*mask2

        result=panorama1+panorama2

        rows, cols = np.where(result[:, :, 0] != 0)

        min\_row, max\_row = min(rows), max(rows) + 1

        min\_col, max\_col = min(cols), max(cols) + 1

        final\_result = result[min\_row:max\_row, min\_col:max\_col, :]

        return final\_result

def main(argv1,argv2,argv3):

    img1 = cv2.imread(argv1)

    img2 = cv2.imread(argv2)

    img3 = cv2.imread(argv3)

    count = 1

    temp=Image\_Stitching().blending(img1,img2,count)

    cv2.imwrite('museum/temp.jpg', temp)

    count += 1

    temp = cv2.imread('museum/temp.jpg')

    final=Image\_Stitching().blending(temp,img3,count)

    cv2.imwrite('museum/panorama.jpg', final)

main('museum/1.jpg','museum/2.jpg','museum/2.jpg')

Discuss:

* 本程式優點:
  + 參數調整：可以通過調整ratio、min\_match等參數來控制特徵匹配的精度和拼接效果
  + 速度快: 計算速度較其他演算法快
* 可以改進的地方:
  + 合併後的圖像可以再更加平滑
  + 範圍過大的圖片會有一些模糊的邊緣
  + 可以以openCV提供的imutils和createStitcher()建立更加精準的的拼接，但是其中演算法較為複雜，未來可繼續深入探究

以下是可改進程式範例:

import os

import cv2

import imutils

import numpy as np

img\_dir = 'museum/'

names = os.listdir(img\_dir)

images = []

for name in names:

    img\_path = os.path.join(img\_dir, name)

    image = cv2.imread(img\_path)

    images.append(image)

stitcher = cv2.createStitcher() if imutils.is\_cv3() else cv2.Stitcher\_create()

status, stitched = stitcher.stitch(images)

if status==0:

    cv2.imwrite('museum/stitch.jpg', stitched)

input圖片與本程式相同



stitch.jpg