Handson 5

CIE5141 Computer Vision in Construction

頭髮好柔順飄逸！羨慕！加**30**分！

b07501113_林庭瑄

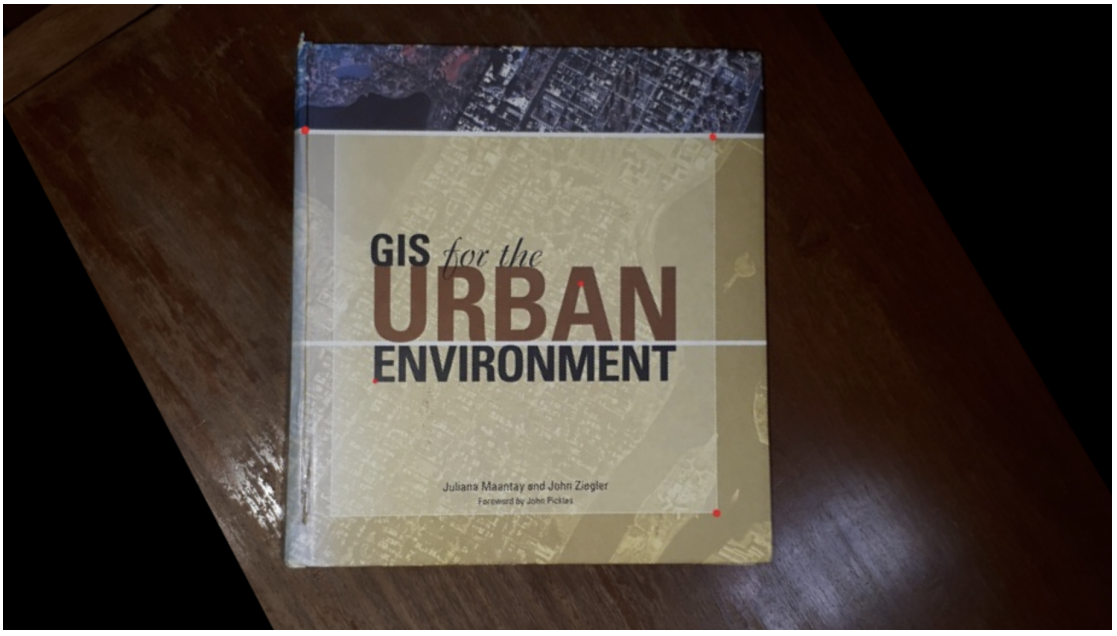
1. Perspective Projection Correction

```
# TODO: Read the source image(book1.jpg) and destination
image(book2.jpg)
filename1 = 'book1.jpg'
filename2 = 'book2.jpg'
img1 = cv2.imread(filename1)
img2 = cv2.imread(filename2)

# TODO: Display the images
cv2_imshow(img1)
cv2_imshow(img2)
```

```
# Identify the coordinates of the points marked in red color in
the source image
pts_src = np.array([[211, 327], [364, 94], [407, 213], [394,
368], [587, 210]])
# Identify the coordinates of the points marked in red color in
the destination image
pts_dst = np.array([[249, 105], [588, 110], [479, 230], [308,
311], [591, 421]])
# TODO: Calculate the homography
# hint: cv2.findHomography
h, status = cv2.findHomography(pts_src, pts_dst)

# TODO: Wrap the images
# hint: cv2.warpPerspective
im_out = cv2.warpPerspective(img1, h,
(img2.shape[1],img2.shape[0]))

# Display the corrected image
cv2_imshow(im_out)
```

```
# Identify the coordinates of the points marked in red color in
the source image
pts_src = np.array([[211, 327], [364, 94], [407, 213], [394,
368], [587, 210]])
# Identify the coordinates of the points marked in red color in
```

```
the destination image
pts_dst = np.array([[249, 105], [588, 110], [479, 230], [308,
311], [591, 421]])
# TODO: Calculate the homography
# hint: cv2.findHomography
h, status = cv2.findHomography(pts_src, pts_dst)

# TODO: Wrap the images
# hint: cv2.warpPerspective
im_out = cv2.warpPerspective(img1, h,
(img2.shape[1],img2.shape[0]))

# Display the corrected image
cv2_imshow(im_out)
```



2. Image Stitching

Stitch the first two images into one from the image dataset.

```
# Write a function named 'imagestitcher' that can stitch two
images
def imagestitcher (img1, img2):
  # TODO: Convert input images to grayscale in order to achieve
more accurate results
 img_gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
 img_gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
 # TODO: SIFT Feature detection
 sift = cv2.SIFT_create()
 # TODO: Find the key points and descriptors
```

```
 kp1, des1 = sift.detectAndCompute(img_gray1,None)
 kp2, des2 = sift.detectAndCompute(img_gray2,None)
  # Feature matching use cv2.BFMatcher(), bf.knnMatch
 # TODO: Create a BFMatcher object. It will find all of the
matching keypoints on two images
 bf = cv2.BFMatcher()
 # TODO: find matching points
 matches =  bf.knnMatch(des1,des2, 2)# K nearest neighbour
matching with k=2

 good = [] # Store good matches into a list if the distance
between corresponding matches lies between 0.5
 for m in matches:
   if (m[0].distance < 0.5*m[1].distance):
     good.append(m)
 matches = np.asarray(good)
  if (len(matches[:,0]) >= 4): # Make ture that we have enough
keypoints
     src = np.float32([ kp1[m.queryIdx].pt for m in matches[:,0]
]).reshape(-1,1,2)
     dst = np.float32([ kp2[m.trainIdx].pt for m in matches[:,0]
]).reshape(-1,1,2)
     # TODO: Calculate the homography
     # hint: cv2.findHomography, adding cv2.RANSAC and
reprojection theshold=5 as the third and forth parameter
     H, masked = cv2.findHomography(src, dst, cv2.RANSAC,5.0)
 else:
     raise AssertionError('Can’t find enough keypoints.')
  # TODO: Image Stitching
 # hint: cv2.warpPerspective((1st image, homography matrix, size
of the output image)
 dst = cv2.warpPerspective(img1, H,
(img2.shape[1]*2,img2.shape[0]))
 dst[0:img2.shape[0], 0:img2.shape[1]] = img2 #stitched image
 return dst
```

```
# TODO: Stitch the first two site images into one and display
stitched_img = imagestitcher (img_1, img_2)
cv2_imshow(stitched_img)
```

3. Panorama creation

Create a panorama combining all 6 images.

```
# TODO: Stitch all 6 images to make a Panorama
imgs = []
imgs.append(img_1)
imgs.append(img_2)
imgs.append(img_3)
imgs.append(img_4)
imgs.append(img_5)
imgs.append(img_6)

stitcher = cv2.Stitcher.create(cv2.Stitcher_PANORAMA)
_result, pano = stitcher.stitch(imgs)

# TODO: Display the panorama image
cv2_imshow(pano)
```