Image Warping and Mosaicing

Part1: Shoot the Picture

Part2: Recover Homographies

Here is my code.

```
def normalize(pts):

  c = pts.mean(axis=0)
  rms = np.sqrt(((pts - c)**2).sum(axis=1).mean())
  s = np.sqrt(2) / rms
  T = np.array([[s,0,-s*c[0]],[0,s,-s*c[1]],[0,0,1]])
  pts_h = np.c_[pts, np.ones(len(pts))].T

  return T, (T @ pts_h).T[:, :2]

def homography(src, dst):
  T, src_n = normalize(src)
  Td, dst_n = normalize(dst)

  A = []
  for (x,y),(xp,yp) in zip(src_n, dst_n):
    A.append([-x,-y,-1, 0, 0, 0, x*xp, y*xp, xp])
    A.append([ 0, 0, 0,-x,-y,-1, x*yp, y*yp, yp])

  A = np.asarray(A)

  U,S,Vt = np.linalg.svd(A)
  Hn = Vt[-1].reshape(3, 3)
  H = np.linalg.inv(Td) @ Hn @ T
  print(src)
  print(dst)

  print(A)
  print(H)

  return H / H[2,2]
```

I manually selected the corresponding feature points between the two images. The blue circles indicate the matched pairs used for homography estimation

Here is the matrix of A and matrix of H.

```
[[ 1.41636477  0.65612667 -1.          0.          0.          0.
   2.23469749  1.03521682 -1.57776975]
 [ 0.          0.          0.          1.41636477  0.65612667 -1.
   0.84996584  0.39374409 -0.60010377]
 [ 1.44033287  1.01564814 -1.          0.          0.          0.
   2.35777618  1.66258165 -1.63696618]
 [ 0.          0.          0.          1.44033287  1.01564814 -1.
   1.48676586  1.04839028 -1.03223768]
 [ 0.73327399  1.04860427 -1.          0.          0.          0.
   0.52753312  0.75438853 -0.71942157]]
```

```
[[ 6.96710112e-01  4.13044009e-02 -4.67861402e+02]
 [ 9.60437387e-02  5.74706140e-01 -9.52286652e+01]
 [ 2.05360235e-04 -3.02805080e-07  3.42065557e-01]]
```

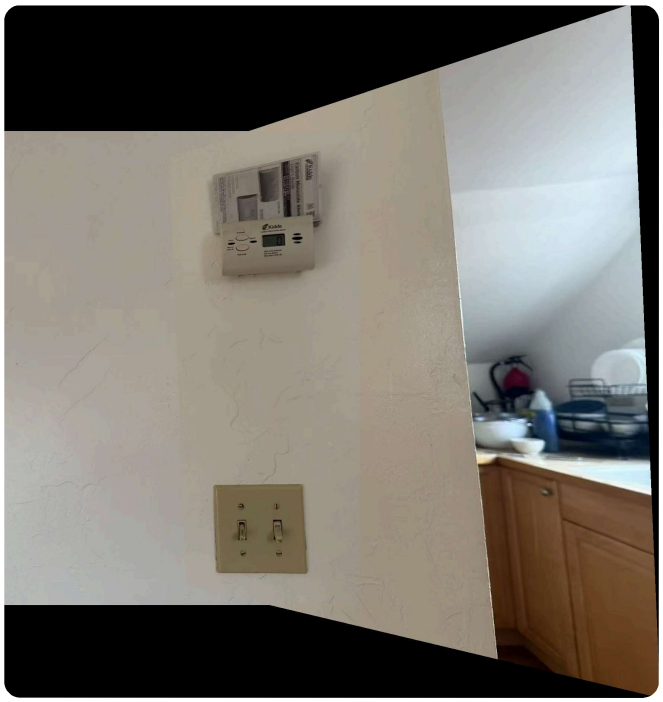
Part3: Wrap the Images

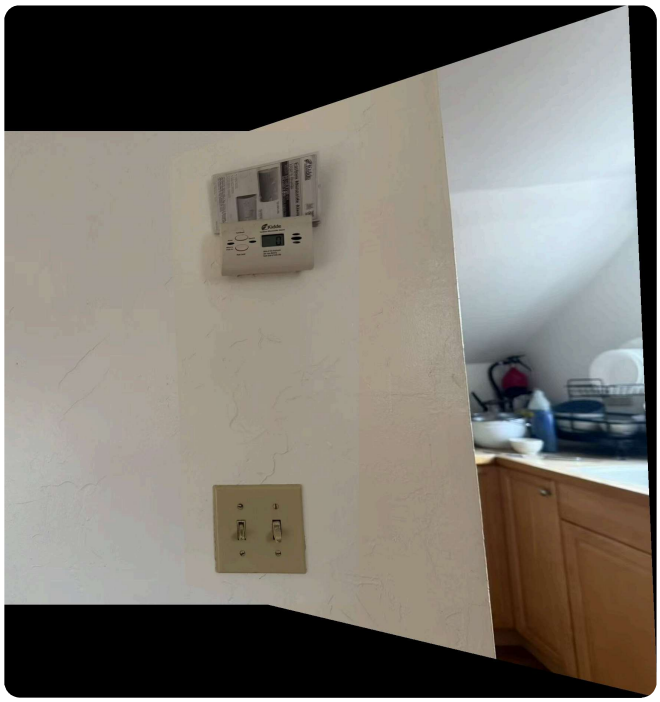
The left image was generated using warpImageNearestNeighbor, and the right image using warpImageBilinear. The result shows that the left image exhibits more noticeable discontinuities, while the right image appears smoother and more visually consistent.

Part4: Blend the Images into a Mosaic

```python
def blend_2imgs(img1, img2):

    assert img1.shape == img2.shape
    a = img1.astype(np.float64)
    b = img2.astype(np.float64)
    if img1.ndim ==2:
      m1 = (a != 0).astype(np.float64)
      m2 = (b != 0).astype(np.float64)

      wsum = m1 + m2
      wsum[wsum == 0] = 1.0

      out = (a * m1 + b * m2) / wsum
    else:
       m1_pix = (np.sum(a, axis=2) != 0).astype(np.float64)
       m2_pix = (np.sum(b, axis=2) != 0).astype(np.float64)
       m1c = m1_pix[..., None]
       m2c = m2_pix[..., None]

       wsum = m1c + m2c
       wsum[wsum == 0] = 1.0
       out = (a * m1c + b * m2c) / wsum


    if np.issubdtype(img1.dtype, np.integer):
        out = np.clip(out, 0, 255).astype(img1.dtype)
    else:
        out = out.astype(img1.dtype)
    return out
```

In this part, I aligned multiple images onto a common coordinate system using the computed homographies and blended them into a single mosaic through weighted averaging. Each image was assigned higher weights near the center that gradually decreased toward the edges, which helped reduce visible seams and brightness discontinuities. The final results produced three smooth and natural-looking mosaics.



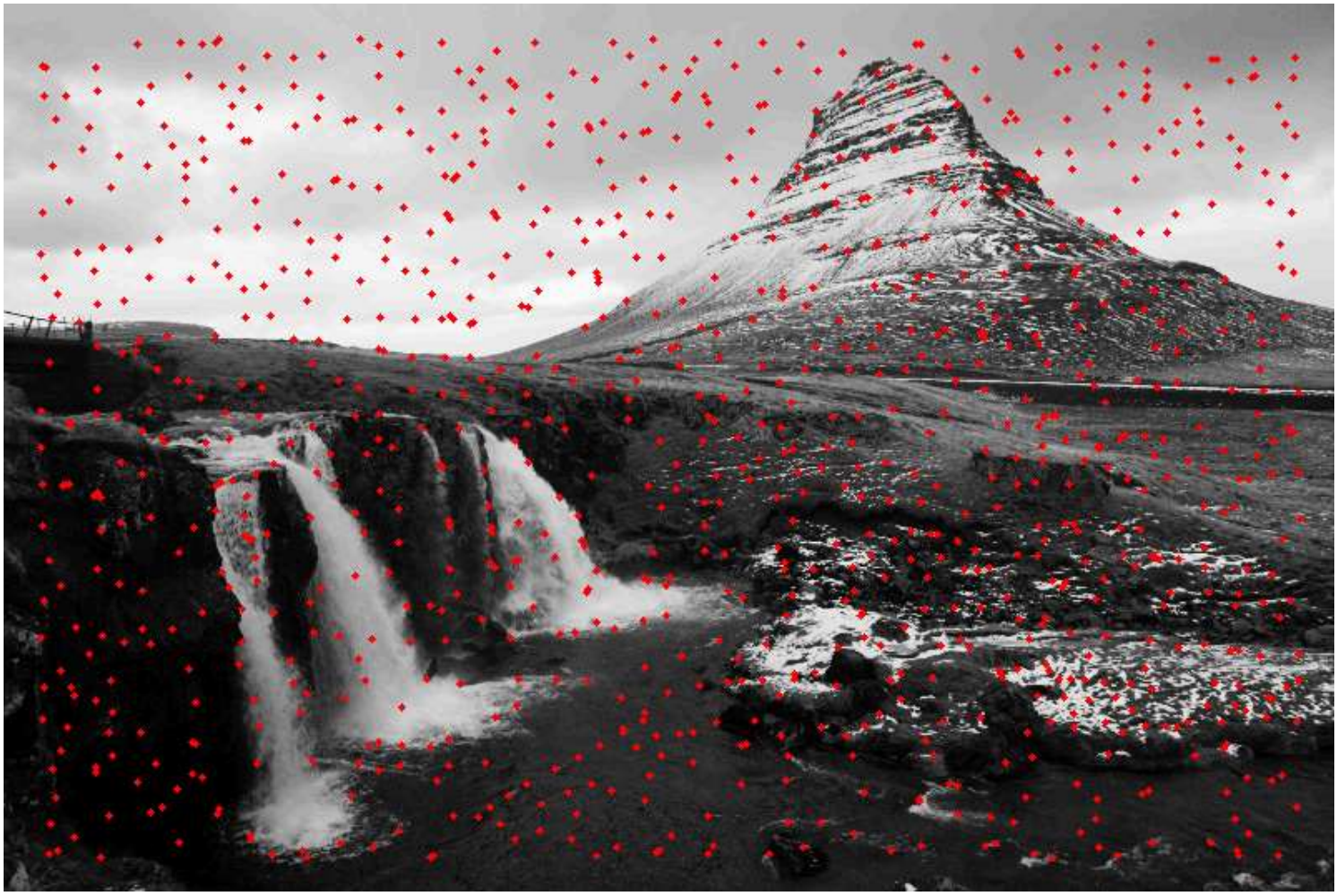


Part2.1: Harris Corner Detection

Here is the orginal image

Corner detection without ANMS

Corner detection with ANMS



Part2.2: Feature Descriptor Extraction

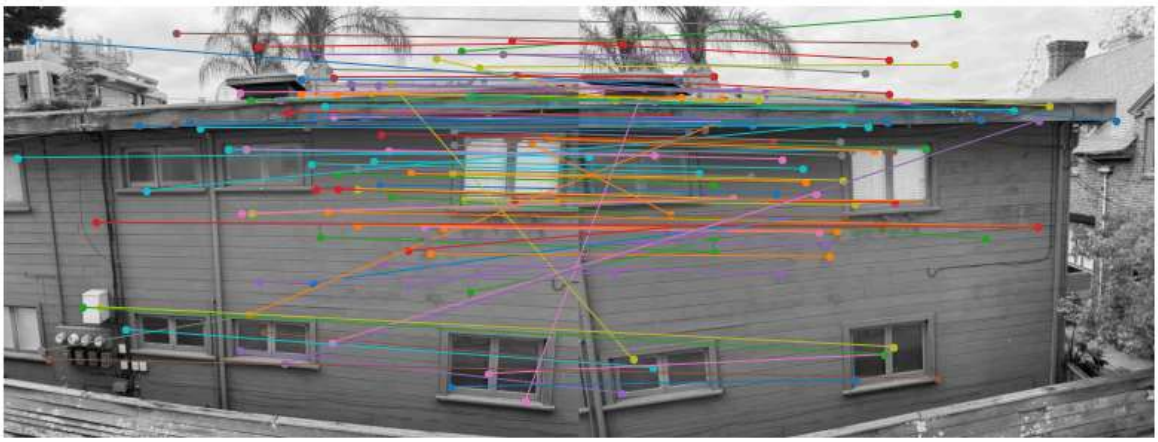
First two features of corner detection with ANMS

```
[[-0.49605295 -0.46528998 -0.34223807 -0.03460835  1.0728587   0.9805698
   1.4112514   1.2112921   0.765229    1.39587     1.4727774   1.0728587
   1.3189625   0.41145477  0.65755856  1.2112921  -0.14227876  0.6421771
   0.6267956   0.22687693  0.9959513   1.0420958   0.1499695   1.303581
  -0.8959716  -0.8036827  -0.8652086  -1.1574569  -1.1420754  -0.9882605
  -0.5883419   0.56526965 -1.2497458  -1.2036014  -1.0497864  -0.6652493
   0.8267549   0.0269176  -0.01922686  1.1805291  -1.3574163  -1.2189828
  -1.2651273  -0.434527    1.303581    1.5035404   1.4266329   1.3189625
  -1.2343643  -1.2958902  -1.2805088  -1.1420754  -0.2960936   0.5345067
   0.93442535  1.3651069  -1.2497458  -1.2651273  -1.2651273  -1.2958902
  -1.1728383  -0.5883419  -0.24994916 -0.23456767]
```

```
[ 1.4351043   1.8894533   0.41281876  1.2079297  -0.08696526 -0.56403184
 -0.45044455  0.27651402  2.162063    1.5941265  -0.24598745 -0.49587944
  0.52640605  1.0943425   1.1624948   0.41281876  0.00390456  1.0943425
  0.77629805  0.11749184  0.09477438  1.2079297   0.8898853   0.7990155
  0.4355362   0.29923147 -0.40500963 -0.29142237 -0.3141398   0.6854282
 -0.08696526 -0.24598745  1.821301   -0.13240017  0.4355362   1.5941265
 -0.22326998 -0.473162    1.2987995   1.525974   -0.723054   -0.6776191
 -0.6776191   0.6627108  -0.5185969  -0.6549016  -0.42772707  0.9353202
 -1.2909904  -1.1774031  -1.3818603  -1.1092508  -1.2682729  -1.1774031
 -1.2682729  -1.3818603  -0.54131436  0.14020929 -1.1546856  -1.518165
 -1.4272951  -1.4954475  -1.5408823  -1.5635998 ]
```

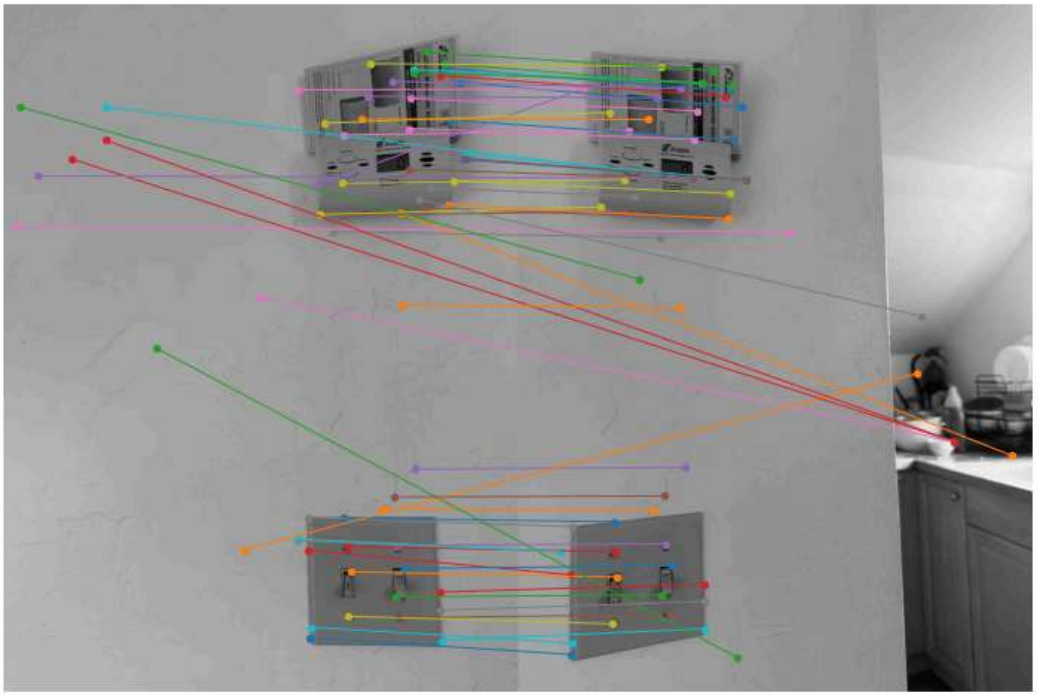
Part2.3: Feature Matching

Match result of house.



Match result of my desk.

Match result of my lobby.



Part2.4: RANSAC for Robust Homography

Left side is using the Bilinear manually, and right side is automatic stitching.


MARVIS
MARVIS
JASMIN MINT
MARVIS
护照
PASSPORT


MARVIS
MARVIS
JASMIN MINT
MARVIS
护照
PASSPORT