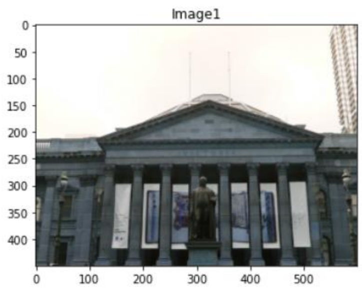
CMSC 426 – Computer Vision

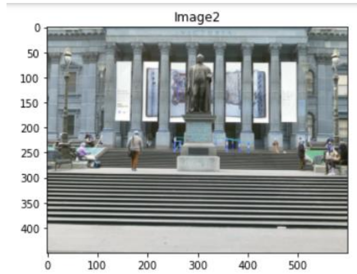
Project 4: Panoramic Stitching with RANSAC

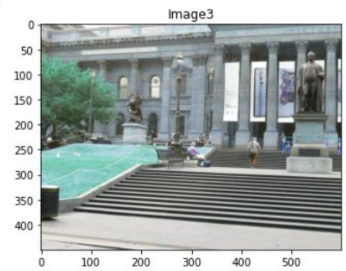
In this project I will form a panoramic image from a set of photos captured at different perspectives.

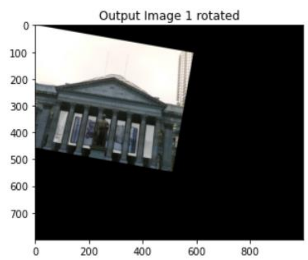
Alessandro Russo

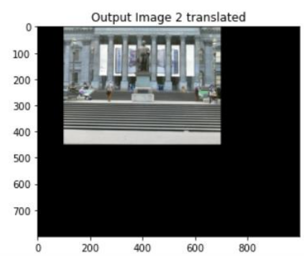
2 – Intro t o Homographies:

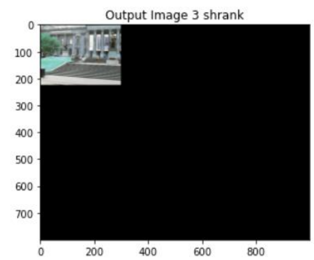












This part involved rotating the first picture 10 degrees, translating the second picture 100 pixels right and shrinking the third picture by ½.This part was simple as I followed how to make the transformation matrices from lecture. I then plugged that matrix into the warpPerspective function. Tx is this case was 100, ty was 0 (no shift on y axis). Theta was the radian value of 10. And lastly Sx and Sy were both 0.5.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

translation

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{s_x} & 0 & 0 \\ 0 & \boldsymbol{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

scaling

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

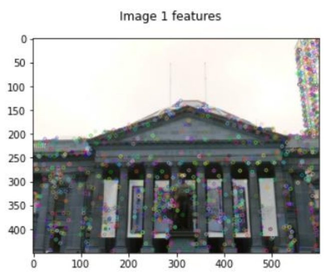
rotation

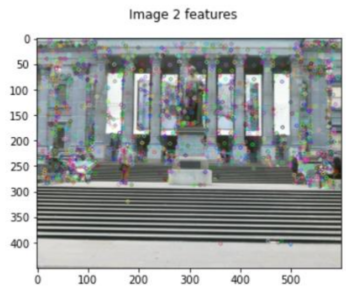
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \beta_x & 0 \\ \beta_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

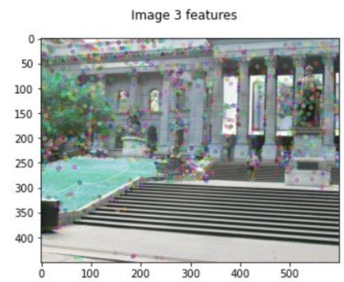
shearing

3.1 – Compute Sift features:

Similarly, to project 2, I used the siftDetectandCompute to calculate all the features in the 3 images. I used cv2.drawKeypoints to visually show the features as you see below.







3.2 – Match Features:

Firstly, I used the distance matrix command in scipy.spatial to find all the distances between image 2 and 1 and between image 2 and 3. I then flattened the matrices, sorted them, and took the first 100 values to preserve the 100 best matches. I then got all the key points of the indices that the values corresponded to and as a result was left with 2 sets of match features.

Set of Matches between 1 and 2

```
[[ 0.  0.  0. ...  4. 22. 16.]
 [ 82.  2.  0. ... 39.  9.  3.]
 [ 11.  5. 110. ...  5.  4.  1.]
 ...
 [ 17. 39. 25. ...  2.  0.  2.]
 [ 0.  2. 132. ...  1.  1.  0.]
 [ 25.  0.  0. ...  0.  0.  7.]]
[[ 0.  0.  0. ...  6. 20. 20.]
 [ 81.  4.  0. ... 50. 12.  3.]
 [ 7.  2. 102. ...  3.  5.  0.]
 ...
 [ 13. 47. 30. ...  1.  0.  1.]
 [ 0.  1. 99. ...  1.  0.  0.]
 [ 14.  0.  0. ...  4.  1. 16.]]
```

Set of matches between 2 and 3

```
[[ 1.  0.  0. ...  0.  0.  6.]
 [ 2.  0.  0. ...  0.  0.  1.]
 [ 3.  0.  0. ...  0.  0.  1.]
 ...
 [ 4.  2.  2. ...  3.  8.  1.]
 [ 4.  0.  0. ...  0.  0.  1.]
 [20. 13.  0. ...  0.  1.  3.]]
[[ 3.  0.  0. ...  0.  0.  3.]
 [ 3.  0.  0. ...  0.  0.  1.]
 [ 2.  0.  0. ...  0.  0.  3.]
 ...
 [ 8.  3.  3. ...  5.  7.  1.]
 [ 2.  1.  1. ...  0.  0.  0.]
```

3.3 – Estimate the Homographies:

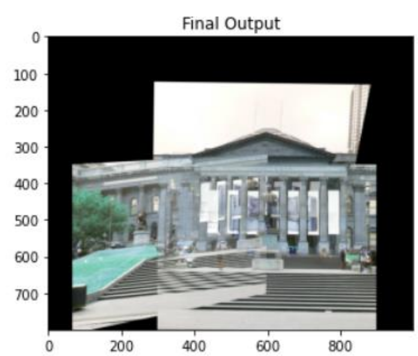
I used the cv2.findHomography command to apply RANSAC to find a homography mapping image 1's matched features to image 2's. Similarly, I did this with image 2 and image 3. When applying RANSAC, I declared a set of matched features inliers if the reprojection error is less than 2 pixels. I used the indices corresponding to the best matches as a basis to form the source points and the destination points. An example between image 2 and 1 can be seen below.

```
src_pts = np.float32([ m.pt for m in best_features_img1_21 ]).reshape(-1,1,2)
dst_pts = np.float32([ m.pt for m in best_features_img2_21 ]).reshape(-1,1,2)

M_21, mask_21 = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, ransacReprojThreshold = 2)
```

3.4 – Warp and translate Images:

Using the same logic as the first part I declare my translation matrix like so: translation_matrix = np.float32([[1, 0, 300], [0, 1, 350], 0, 0, 1]]). This signifies a shift of 300 pixels to the right and 350 pixels down. I multiplied this by my homography matrix between image 1 and image 2 returned above then fed this into the warpPerspective function. I repeated the same for image 2 and image 3. For image 2 I did not need the homography matrix, I just used the translation matrix. For the stitching I used np.maximum twice to fuse the images together. Below is the result.



As you can see the pictures are aligned along the roof and along the statue.