Project 4: Report and Code Output

Alejandro J. Rigau

April 28, 2021

Abstract:

In this project, I present my Panoramic Stitching with Homographies results. I start by computing the SIFT features, then matching the features between images, estimating the homographies, and finally warping and translating the images. In the end, this results in a panoramic image.

2 Intro to Homographies

In this part I created the homography matrices by hand using numpy. I based them on the matrices shown in class. The matrices are as follow:

$$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

translation(t_x,t_y) rotation(θ) scale(s,s)

I used ***cv2.warpperspective()*** to apply these matrices to the images. First, I used the rotation matrix on image 1 to create a 10 degree angle. The translation matrix was then applied to image 2, which was moved 100 pixels to the right. Finally, I used the scaling matrix to reduce image 3 to half its original size. The images it produced are as follows:

Figure 1: 1.jpg rotated 10 degrees clockwise

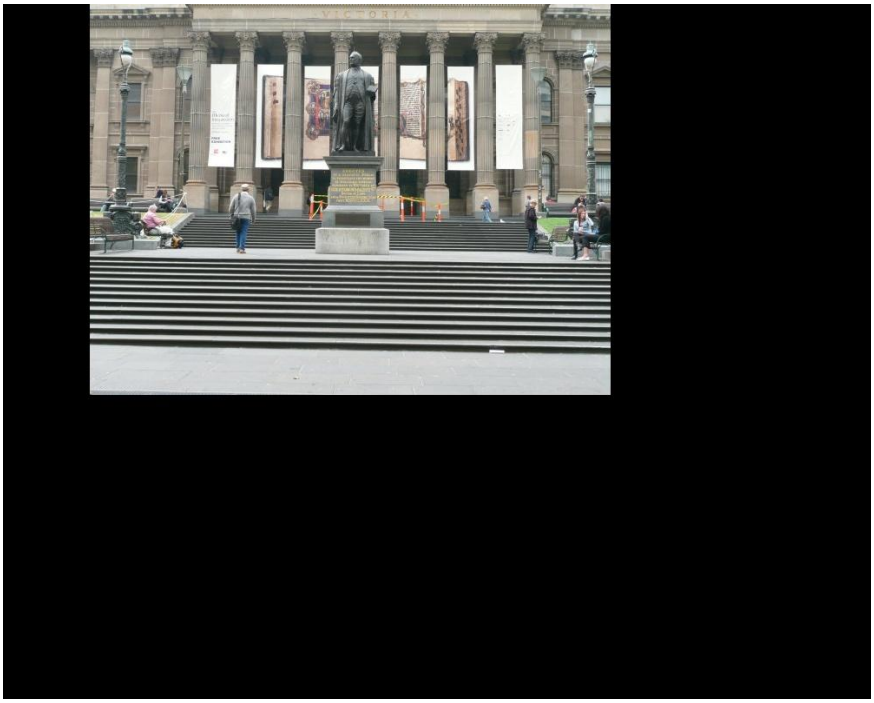

Figure 2: 2.jpg translated 100 pixels to the right

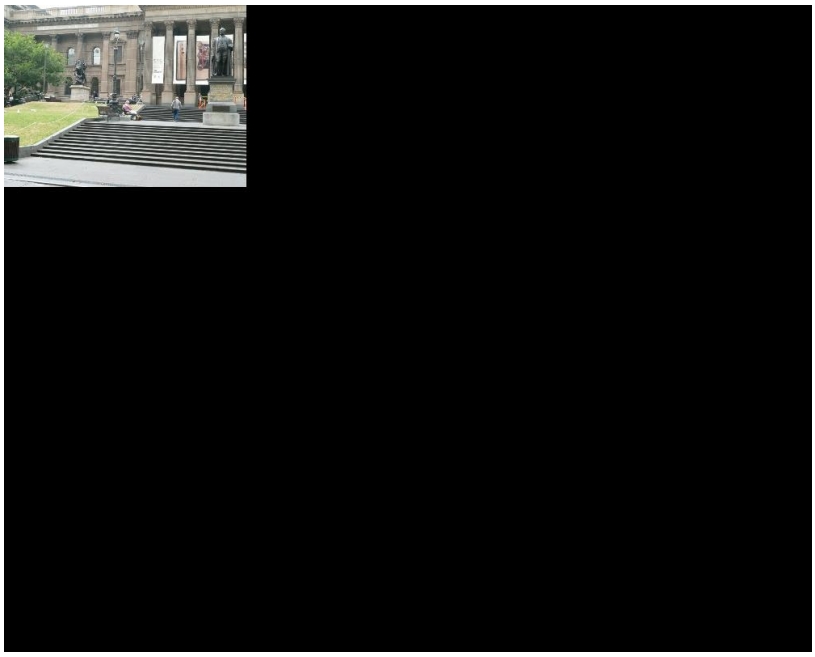

Figure 3: 3.jpg scaled to half its size

3 Panoramic Stitching

Now that I know how to use the ***cv2.warpperspective()*** function, I can work on the steps to create a panoramic image.

3.1 Compute SIFT features

In this section, I used the ***cv2.SIFT_create()*** function to initiate the SIFT detector which I then applied on the 3 images to obtain their respective keypoints and descriptors. Each of these were saved on separate variables.

3.2 Match Features

To match the features, I wrote a function called computeMatrix(kp1,des1,kp2,des2) that took in two sets of keypoints and descriptors and returned the best matching features for those images. This function was run twice: once to find the 100 best-matching features between images 1 and 2, and then again to find the best-matching features between images 2 and 3. The function calculates the distance matrix of all the descriptors that were passed as arguments. To select the best matching descriptors, I flattened the distance matrix, sorted it, and then saved the value in index 100. This would be my threshold. I would save the indexes of the original positions and

the values that were below this threshold. Once I collected the indices that I needed, I extracted those index values from the original keypoint and descriptor variables that contained all the features. In the end, this would extract the 100 best-matching features from the images.

3.3 Estimate the Homographies

Now that I have the 100 best matching features, I transformed all the keypoints into the proper format and passed them to the ***cv2.findHomography*** function that estimated the homography using the ***RANSAC*** algorithm. This found a homography mapping image 1's matched features to image 2's. Similarly this was done with image 3's matching features to image 2.

3.4 Warp and translate images

Finally, I'll warp the images into proper alignment using the homography matrices from the previous section. However, before doing so, the homography matrices were multiplied by a translation matrix to move the images to the correct location. Image 2 only required the translation matrix to be transformed, whereas images 1 and 3 required both the homography matrix calculated in the previous part and the translation matrix.

Figure 4: Final result after warping