Basic Stitching Pipeline

Stitching Module: Color Correction + OpenCV Stitching

In this stage, we are going to go through the stitching pipeline that provided by OpenCV with additional color correction proposed in Xiong's paper[1] as pre-processing. As shown in Figure.1. the image stitching algorithm consists of steps: color correction, image registration(feature detection, feature matching), image matching, bundle adjustment, wave correction, surface warping, seam finding, exposure compensation and image blending, among which color correction, bundle adjustment, wave correction, exposure compensation and seam finding are optional, but the quality of the stitched image will be affected without them. The results w/wo color correction, w/wo wave correction, w/wo exposure compensation, w/wo seam finding will be compared and discussed.

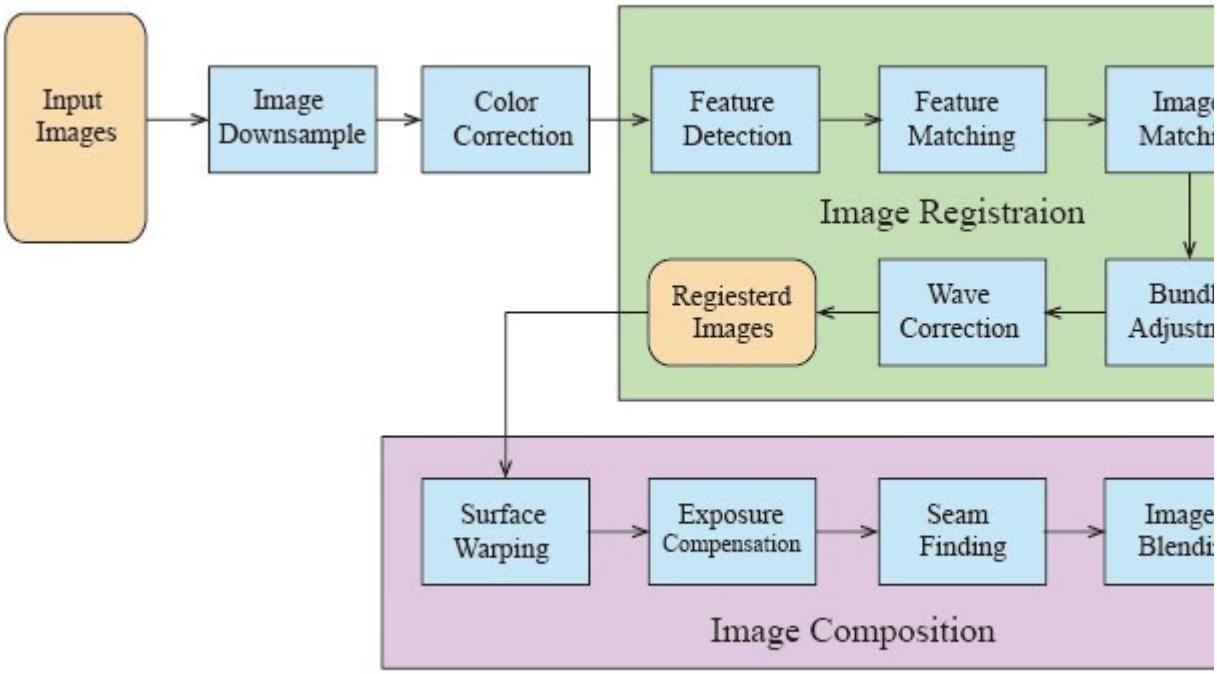


Figure. 2.

Stitching Pipeline

Input Images

In this step, input image sequence is loaded and down-sampled to proper size to reduce the stitching time. Figure. 2 shows the test data set we use, which consists of 5 natural scene images, and each image is down-sampled to 600 x 450.

Figure. 3. Test Dataset[2]

Step 1. Color Correction(Implementation of Xiong's paper[1])

The iPhone camera has a good performance in color rendering. The color differences might not be notable for common outside scene. Even so, color correction is very important for both visual effects and following-up stitching steps. Also, if we use auto-white balance and exposure, the color will be changing between different scenes. Equation (1)-(4) shows the adjustment factor calculation. It first use gamma-corrected RGB value to calculate the sum of the pixel value ($\Sigma P_{c,i}(p)^{\gamma}$) of each image i, for each channel c. Then calculate the difference factor α between images. A global adjustment factor is calculated by solving the least square function that the overall adjustment factor is as close to 1 as possible. (Equation (2)(3)). The final adjustment factor is the gamma-uncorrected overall factor $(g_c\alpha_{c,i})^{1/\gamma}$. The α factors are based on one "best image", which could be chosen automatically by the gray world assumption. [1]

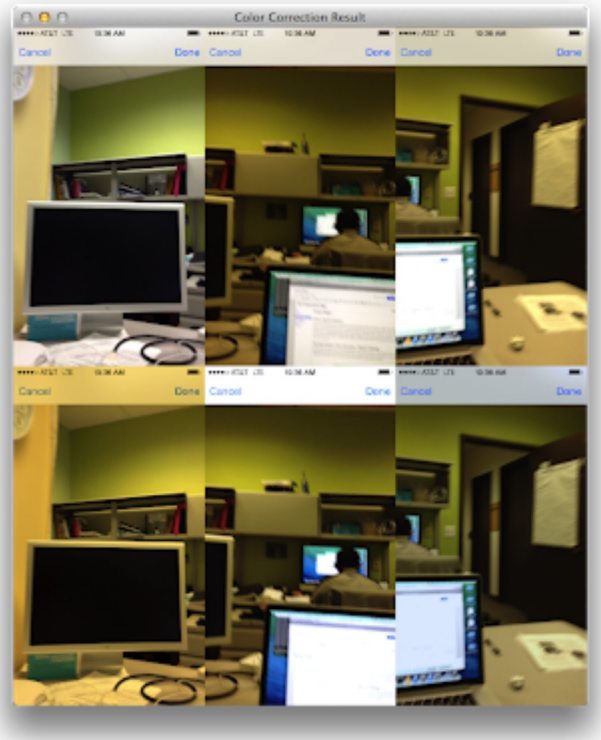
$$\alpha_{c,i} = \frac{\sum_{p}(P_{c,i-1}(p))^{\gamma}}{\sum_{p}(P_{c,i}(p))^{\gamma}} \quad c \in \{R,G,B\}\ (i = 1,2,3,\cdots,n), \tag{1}$$

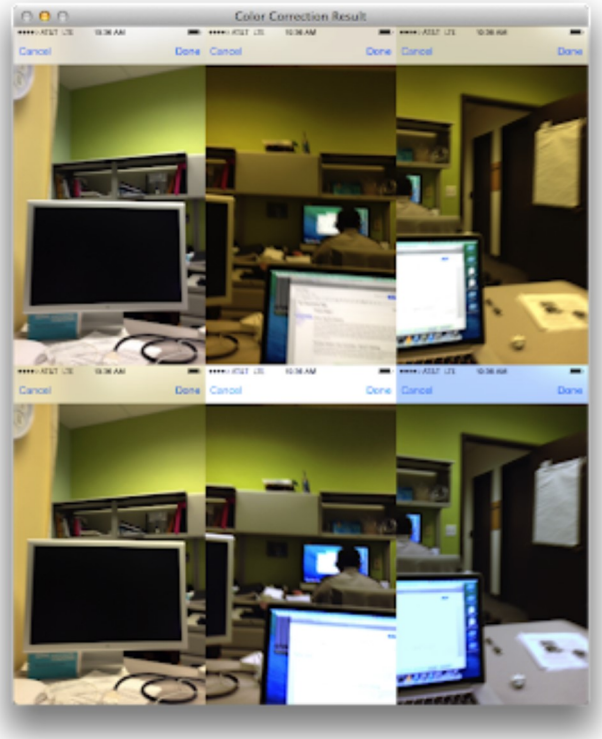
$$\min_{g_c} \sum_{i=0}^{n} (g_c \alpha_{c,i} - 1)^2 \quad c \in \{R,G,B\} \tag{2}$$

$$g_c = \frac{\sum_{i=0}^{n} \alpha_{c,i}}{\sum_{i=0}^{n} \alpha_{c,i}^2} \quad c \in \{R,G,B\}\ (i = 0,1,\ldots,n) \tag{3}$$

$$P_{c,i}(p) \leftarrow (g_c \alpha_{c,i})^{1/\gamma} P_{c,i}(p), c \in \{R,G,B\}\ (i = 0,1,\ldots,n) \tag{4}$$

Figure. 4. shows some demo results of the color correction result. For both subfigures, the first rows are the original image. The second rows show the corresponding color correction result, based on the first image. For (a), the global adjustment is implemented as in the paper. We can see that image 2 and 3 was too dark and yellowish before correction. They are adjusted a little bit but since the "yellowish style" is dominant in the series, image 1 has also been adjusted to darker and yellowish because of the global adjustment factor, even though we choose image 1 as the basis. For (b), the effect of global adjustment is reduced by applying of 0.2th power of the original factor. We can see that the results are less yellowish and brighter, more like image 1, whose color is more realistic. The iOS software "title bar" are kept only for this demo, so that you can actually make sure how the color is changed (less yellowish, more bluish, etc.) from the color change of the title bar, since the demo of a color correction might be less visible on a projector or worse screen. (The title bar is small and has a minor effect on the overall color calculation of the images.) Figure. 5 shows our results using the test set from the original paper. It use the first image as the basis using the automatic best image selection procedure. The "best image" choice remains an open question since the "mathematically" best image is not necessarily the "visual" best image. Figure. 6 shows the results of the test set we use in other steps, there are only slight differences between corrected and un-corrected images. However, from the 4th image we can see that the exposure has also been corrected. The average running time for color correction is 0.25s





(a) with global adjustment (b) with reduced global adjustment

Figure. 4. Color Correction Results Comparison (click to enlarge)

Figure. 5. Color Correction Results of the Dataset from Xiong's paper[1]

Figure. 6. Color Correction Results of the Test Dataset

Step 2. Feature Detection

The first step in panorama stitching is to extract and match features. Here we use OpenCV's implementation of the SURF (Speeded Up Robust Features)[3] detector. SURF is faster and more robust than SIFT detector. Figure.7 shows the feature detection results of the test data. The average running time for feature detection is 5.425s.

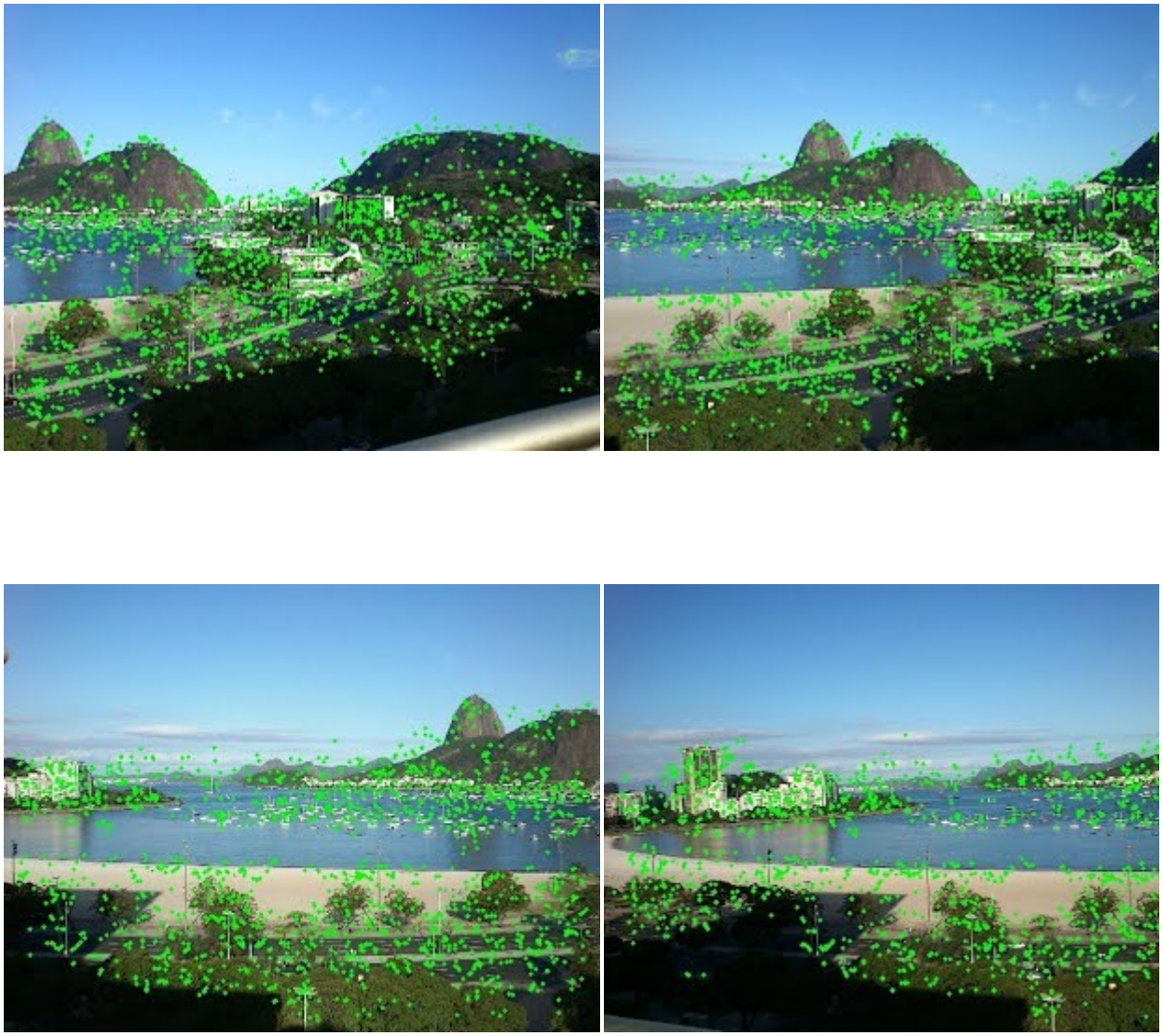


Figure. 7 Feature Detection Results

Step 3. Feature Matching

Once features have been extracted from all images, each feature is matched to its k-nearest neighbors in feature space. This is usually done by using a k-d tree to find approximate nearest neighbors. Here we use OpenCV's FLANN(Fast Library for Approximate Nearest Neighbor)[4], which is a fast implementation of K-D tree. The goal of this step is to find all matching images, since each image could potentially match every other one, the output of this step are 20 pairwise matches.Then, we identify images that have matches whose confidence is larger than the threshold(Here we choose 1.0 as threshold) as the best matched image pairs, following process will only performed on this subset. As is shown in Figure. 8, four best pairwise matches are determined for our test dataset.The average running time for feature matching is 32.464s

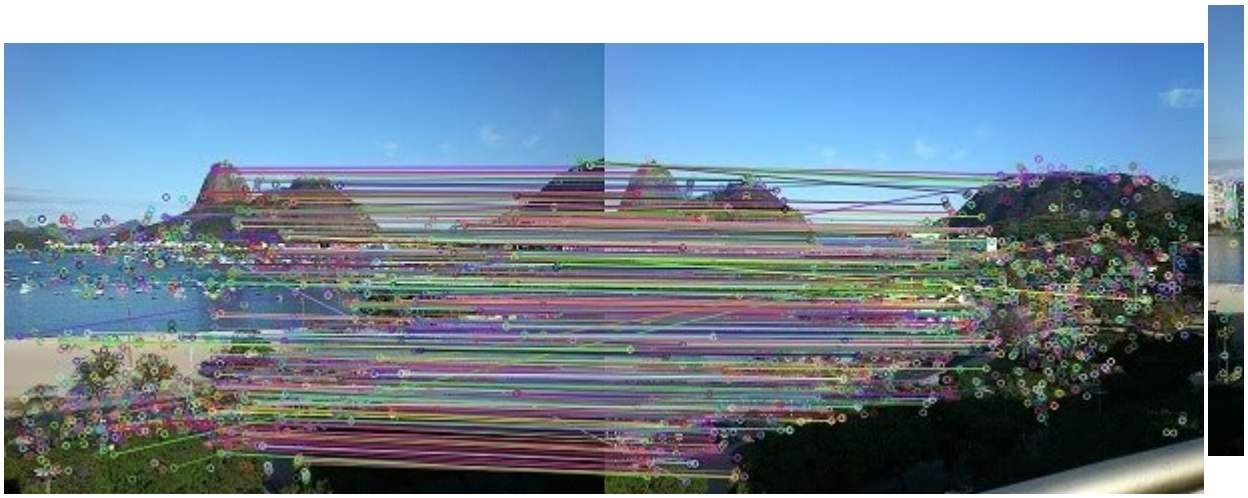


Image 1 <==> Image 2

Image 1 <==> Image 4
Image 5 <==> Image 2

Figure. 8. Feature Matching Results

Step 4. Image Matching

In this step, we use OpenCV's RANSAC(Random Sample Consensus) method to estimate the Homographies of the best matched pairs, with these homographies, images can be aligned. And images aligned this way is called local registration. The average running time for image matching is 0.001s.

Step 5. Bundle Adjustment

Local registration assumes that the rotation matrices of the camera used for capturing each image are the same unity matrics, this will lead to accumulative error for registration. To solve this problem, bundle adjustment is then employed to estimate the rotation matrices by minimizing that error. Figure. 9 shows the comparison of panorama result with and without bundle adjustment. Clearly the images are aligned better with bundle adjustment.The average running time for bundle adjustment is 2.177s

Figure. 9. Comparison of Panorama Result w/wo Bundle Adjustment

Step 6. Wave Correction

Image registration using previous steps however, only gives the relative rotation between each best matched pair, the 3D rotation to a chosen world coordinate is assumed to be the unity matrices. This assumption will lead to wavy effect in final panorama, especially when the number of input images is large. We correct this wavy effect using OpenCV's wave correction. The idea of wave correction is
based on the fact that although people are likely to tilt and rotate the camera when capturing the image, they seldom twist camera relative to the horizontal plane. Therefore, by finding the null space of covariance of camera's horizontal plane, the global rotation matrix can be found to remove the wavy effects.

Figure. 10. Comparison of Panorama Result w/wo Wave Correction

Step 7. Surface Warping

Once images are well aligned, they are mapped to a stitching surface. Here we use OpenCV's implementation of the plane, cylindrical and spherical surface warping method. Figure. 9. shows the surface warping on the first image. The average running time for surface warping is 0.157s

Figure. 11. Surface warp result using different type of surface warper

Once surface mapping is done, the overlapping pixels between each pair can be calculated. And following two steps are based on these overlapping pixels.

Step 9. Exposure Compensation

In previous sections, we have found the geometric parameters of each camera, in order to achieve visual appealing panorama result, we also need to find the photometric parameter of each camera, in other word, the overall gain of each pairs. Here we use OpenCV's exposure compensation method. The idea is to minimize the sum of gain normalized intensity errors for all overlapping pixels. The average running time for feature matching is 0.565s

Step 8. Seam Finding

Object motion and spatial alignment errors will cause ghosting artifacts if the images are composite sequentially as shown in Figure. 10(b). Therefore, an optimal seam finding method is required to find seams in the overlapping area of source images, creating labels for all pixels in the composite image, and merge source images along the optimal seam as shown in Figure. 10(a). Here we used OpenCV's Graph Cut seam finding algorithm. The average running time for seam finding is 17.94s

(a)

(b)

(c)

Figure. 12. Surface warp result using different type of surface warper

Step 10. Image Blending

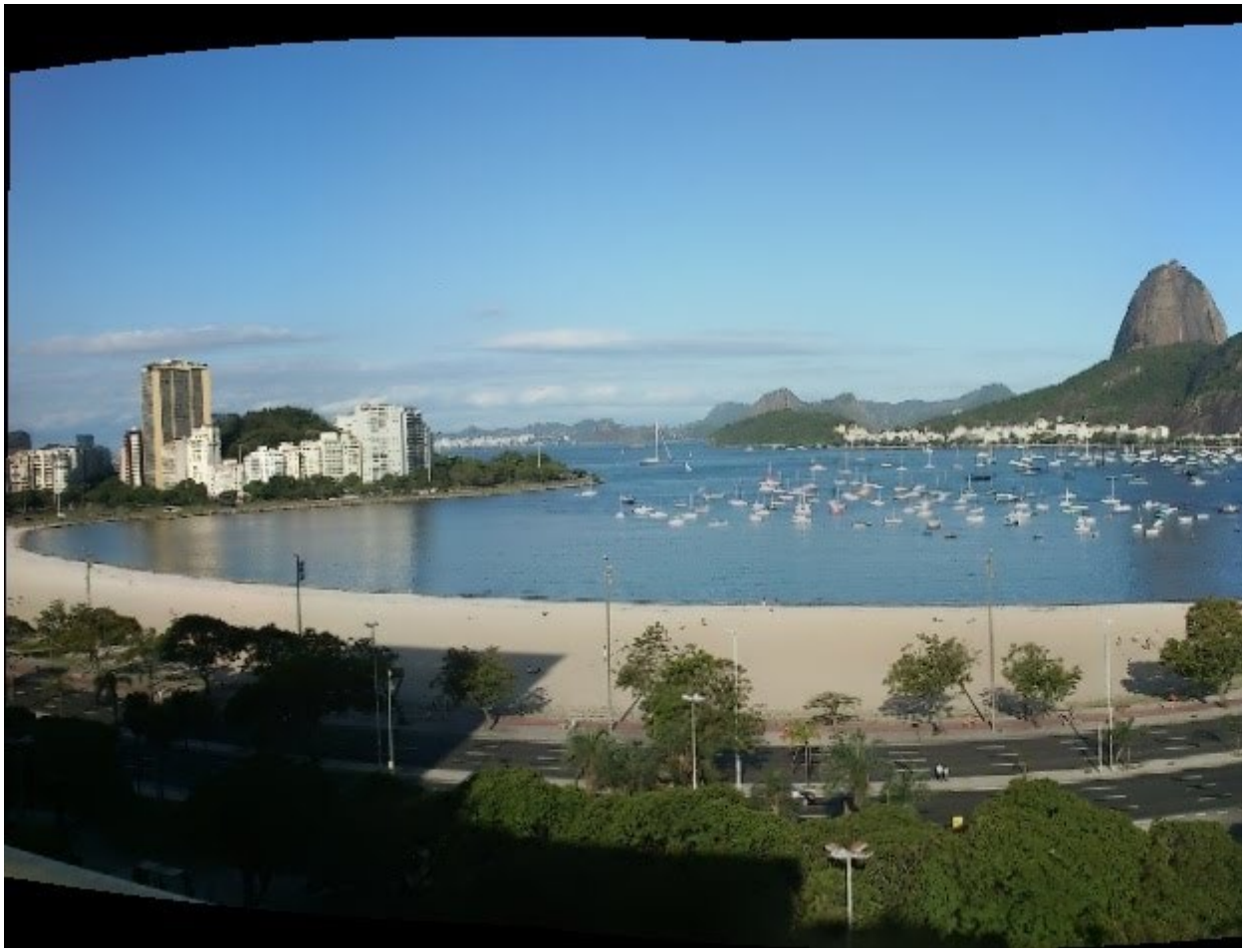
Once the optimal seam is found, OpenCV's Multibind blending method is used for final image composition. The average running time for image blending is 0.938s

The average total running time for a 600*450 image is 52.343s.

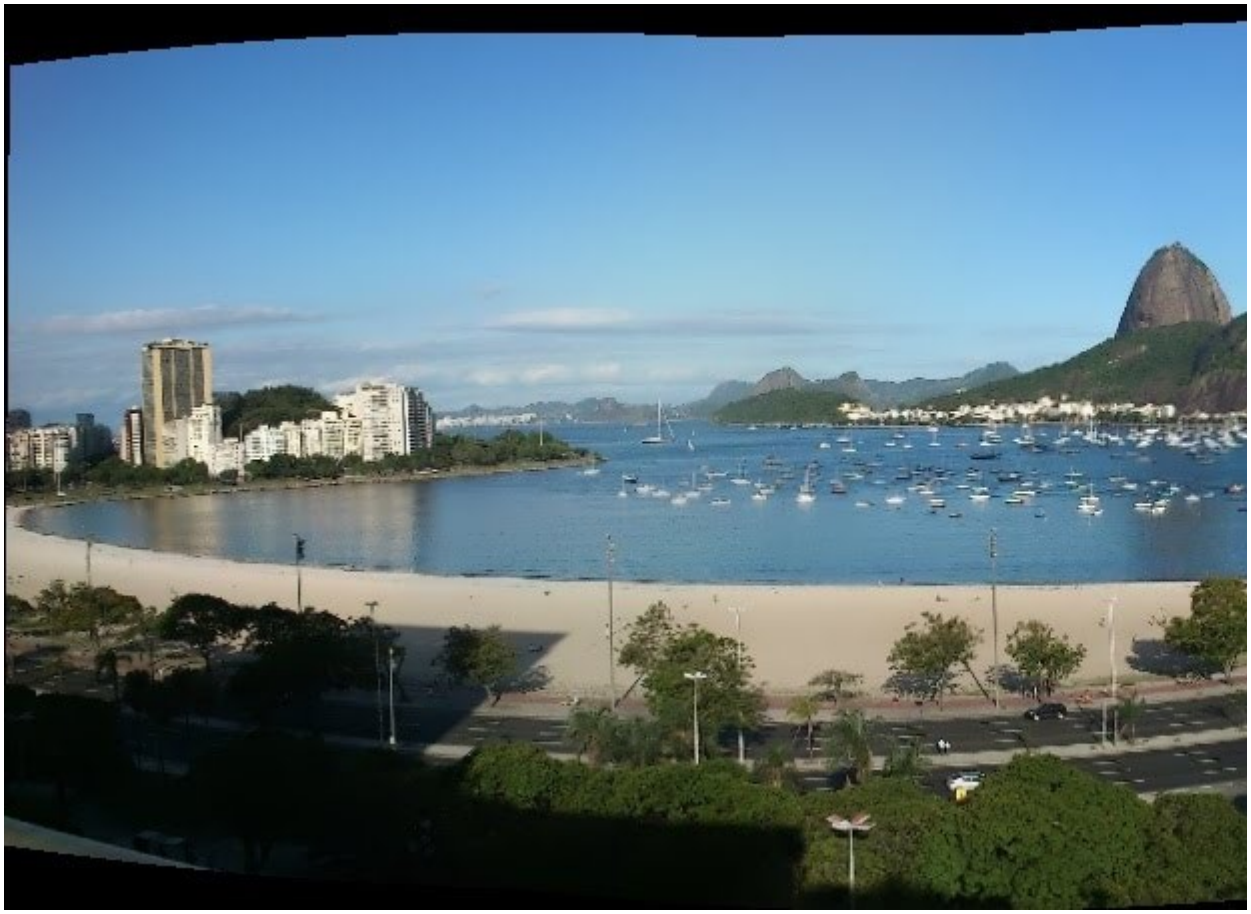
Results and Discussions

1. Panorama using different type of Surface

(a) Plane Surface



(a) Cylindrical Surface



(a) Spherical Surface

Figure. 13. Panorama Result using different surface mapping

2. Panorama Result W/WO Color Correction

Figure. 14. Panorama Result with and without color correction

3. Panorama Result W/WO Exposure Compensation

Figure. 15. Panorama Result with and without Exposure Compensation

Comments

You do not have permission to add comments.