Homework 3

Thiago Henrique Pincinato
Introduction to Signal and Image Processing

April 28, 2019

1 Ransac

1.1

The implementation of the fit model was based on the line equation $m = (y[1] - y[0])/(x[1] - x[0])$. When $x[1]$ and $x[0]$ were equal, a very small number were added in the denominator to avoid a division to zero.

1.2

The function *point_to_line_dist* was implemented by using the definition of the point to line distance from wikipedia ($distance = (ax + by + e = 0, (x0, y0)) = |ax0 + by0 + e|/sqrta^2 + b^2$.

1.3 and 1.4

The result of the Ransac algorithm depend on the edge map, which depends on the sigma value. The parameter sigma will influence in how big it will be the smooth factor of the image, and thus, high values of sigma will lead to a very blurred image that does not contain significant edges. That happens because the value of the pixels that define the line will be below the threshold of the canny edge detection. Without any edges, the edge map is empty, causing the failure of the Ransac algorithm.

In contrast to that, when choosing a very low value of the sigma, we obtain a higher amount of edges which leads to a longer computational time.

Therefore, there is a trade-off between the computational time and the precision of the algorithm in the range of acceptable values of sigma.

When sigma is 1 all noise present in the image will become edge, no smooth is applied, which leads to a wrong final result.

The figure 1 illustrates the different results in the *synthetic.jpg* image due to different sigma values.

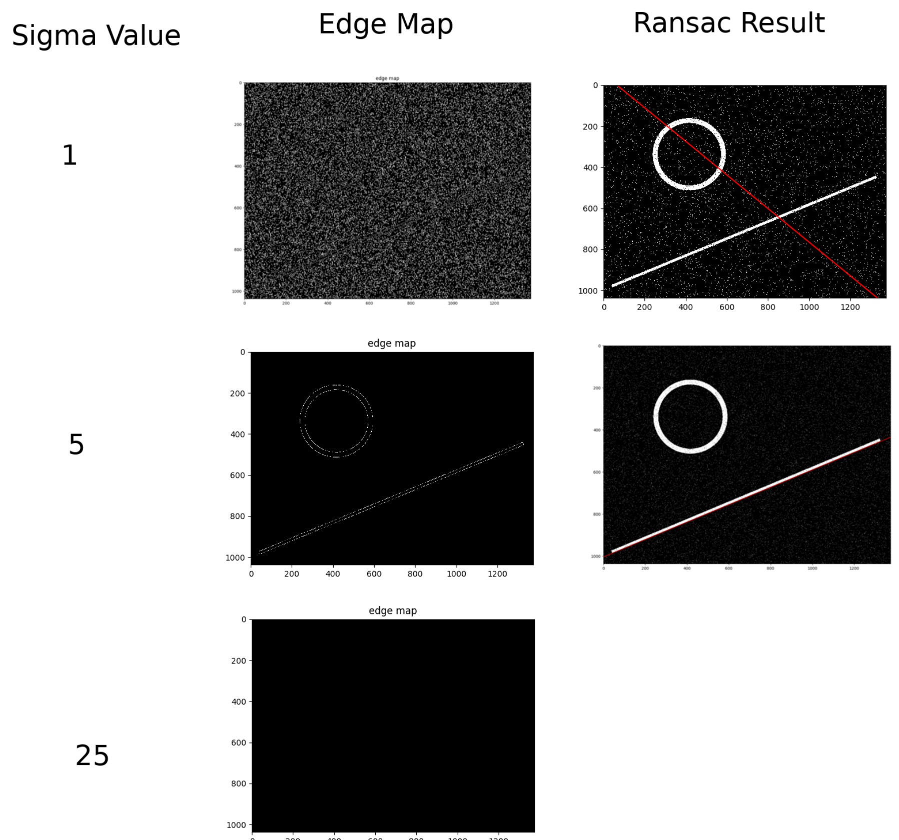

Figure 1: *Ransac Results due to sigma*

1.5

For the image *bridge.jpg* the sigma value was set to 5. Higher values of sigma would lead to a empty edge map and lower values would lead to unnecessary computational effort.

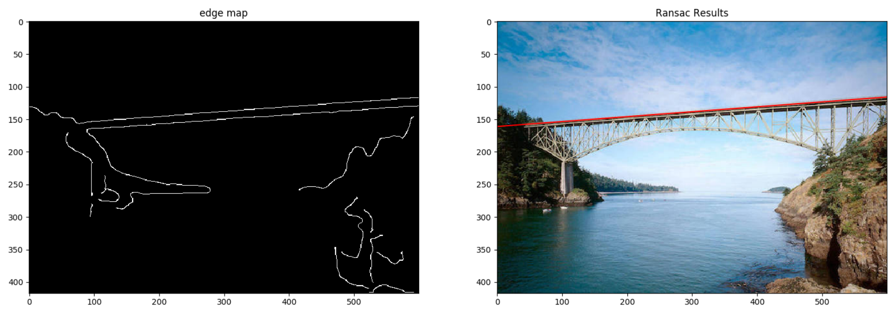

Figure 2: *Ransac Result for bridge.jpg*

The result can be seen in the figure 2

Regarding the image *pool.jpg*, the sigma values used was 5. A sigma value equal to 10 could be used to detect the line that pass to the first step of the stairs, which is not our aim. Higher values return a empty edge map, and as before, value smaller then 5 would lead to unnecessary computational effort. The figure 3 shows the edge map and the Ransac result for this image.

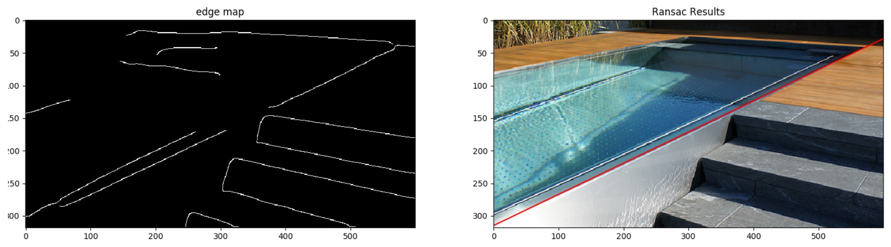

Figure 3: *Ransac Result for pool.jpg*

Concerning the last image (*tennis.jpg*), the value chosen for sigma was 10. As previously, the criteria computational effort and representative result was used. When using a lower value of sigma, the final result is similar, however more computational effort is needed. When using a higher value of sigma, the edge maps is empty. The result of Ransac for *tennis.jpg* can be seen in the figure 4

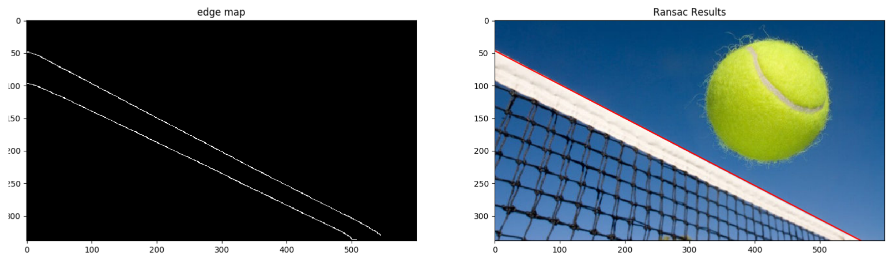

Figure 4: *Ransac Result for tennis.jpg*

2 Texture Synthesis

2.1

The *compute_ssd* function was implemented using a for loop with texture size iterations. In each iteration, the window in the texture is copied to a variable Sn. The Sn indexes in the mask region is set to zero, and then, a vector of *sdd* receives the sum of squared difference for this window. Before the for loop, the variables used in the for loop were prepared.

2.2

Using the index where the sum of square difference is minimum, the positions x and y were obtained. In the function *copypatch* the *Sn* with smaller *ssd* is initialized by coping the right window of the texture. The indexes where the mask has value 1 were set up to zero. Then, two for loop were used to attributed the value of the texture into the return variable (*res*).

2.3

The result of the texture synthesis approach can be seen in the figures 5, 7 and 8.

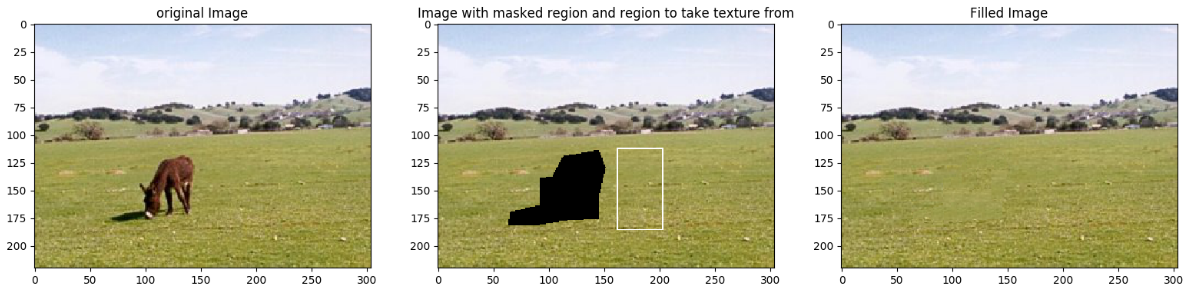

Figure 5: *Texture Synthesis Result for donkey.jpg*

In the figure 5, we have a good result because the texture selected is highly similar with the surround texture of the region of interest (region to be reconstructed). The filled image has some artifacts, which is due to the algorithm that use a whole region instead of a single pixel to be inserted into the masked region. In our case, the quality of the reconstructed image depends on the window size. A small window size would lead to as better fit of the image. In order to illustrate this effect, the value of *patch_half_size* was set to 3. The figure 6 shows the result.

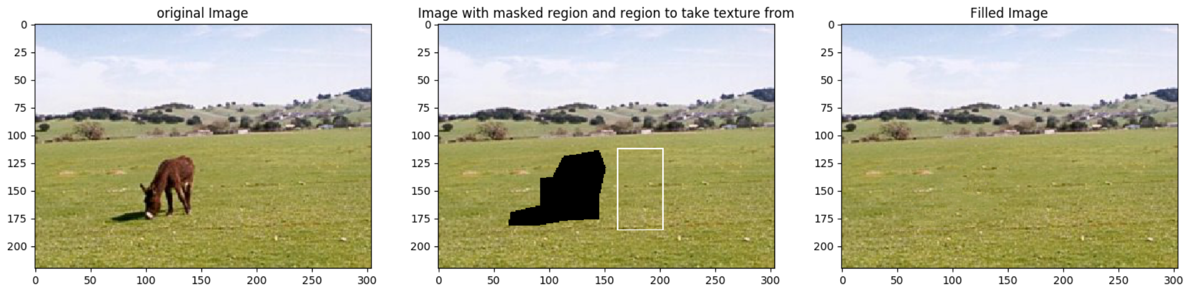

Figure 6: *Texture Synthesis Result for donkey.jpg* $patch\_half\_size = 3$

As we can notice, the quality of the texture synthesis approach was enhanced by simply reducing the window size.

In the figure 7, the result is not as good as before. In this case, we have the effect of Chernobyl Harvest. The texture of the image *tomato.jpg* is quite complex which causes the bad fit. The window size could be reduce, or even a single pixel could be used to improve the result. However, even when using a single pixel, the Chernobyl Harvest effect would happen.

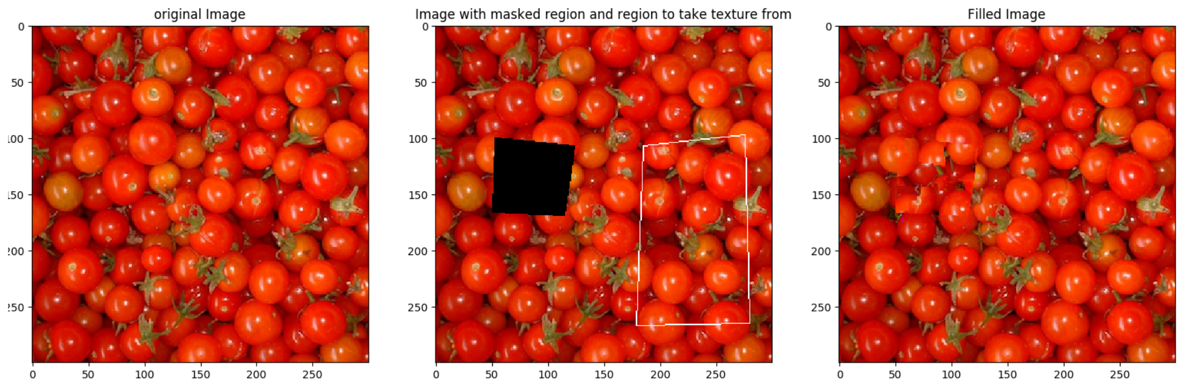

Figure 7: *Texture Synthesis Result for tomato.jpg*

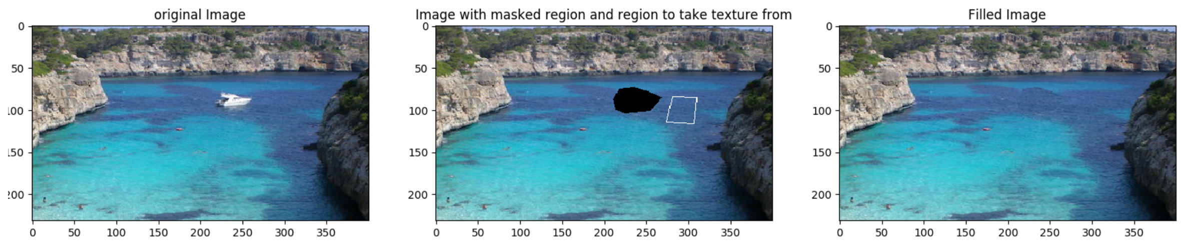

Figure 8: *Texture Synthesis Result for yacht.jpg*

In the figure 8, the result is also satisfactory, being difficult to recognize if the image was edited or not. That happens due to the similarity between the region to take the texture from and the surround of the mask region.

Finally, we can conclude that the algorithm depends not only on the window size, but also on the complexity of the texture to be replaced. In addition, there is a trade-off between the quality of the method and the computational effort, controlled by the window size. Small window size, a higher quality, but more computation effort.