Homework 3

Prof. Raphael Sznitman
Introduction to Signal and Image Processing

Handout: April 10, 2019
Handin: April 24, 2019, 14:15

Instructions

Your hand-in will consist of a `.zip` archive named `hw3_firstName_lastName.zip` containing the following:

- Python source files named `hw3_exY_firstName_lastName.py`, where Y is the exercise number.
- All necessary files to run the above.
- Your report named `hw3_firstName_lastName.pdf`.

Your archive must be uploaded on ILIAS before the deadline.

Code [7 points]

Code templates are provided to help you get started in solving the exercises. In the typical case, you will fill-in the missing function bodies and code blocks. Note that you may also make your own code from scratch.

IMPORTANT: In general, if you are not able to produce a functional code (i.e. your script crashes), comment out the concerned part, and if necessary give a short analysis in your report. Scripts that do not run will be penalized.

Report [3 points]

Along with the code, you are asked to provide a short report. In general, try to be concise. The questions will give you indications on its content.

IMPORTANT: The report is not only for the case where your code fails. It is graded in any case. So, if your code runs great but there is no report, you will only get 7 out of 10.

1 RANSAC [3 points]

In this exercise you will implement the RANSAC method to find lines in a given image.

The script will show two images, one contains the intermediary step from exercise 1.1 and one the final result with the fitted line. Since the program is missing some parts that you are supposed to fill in, the intermediate image shows a black image with some random noise and the resulting image shows the input image without the fitted line. Once you're done with the exercise, you will get results as those shown in figure 1. Template code `hw3_ex1_template.py` and sample images are provided.

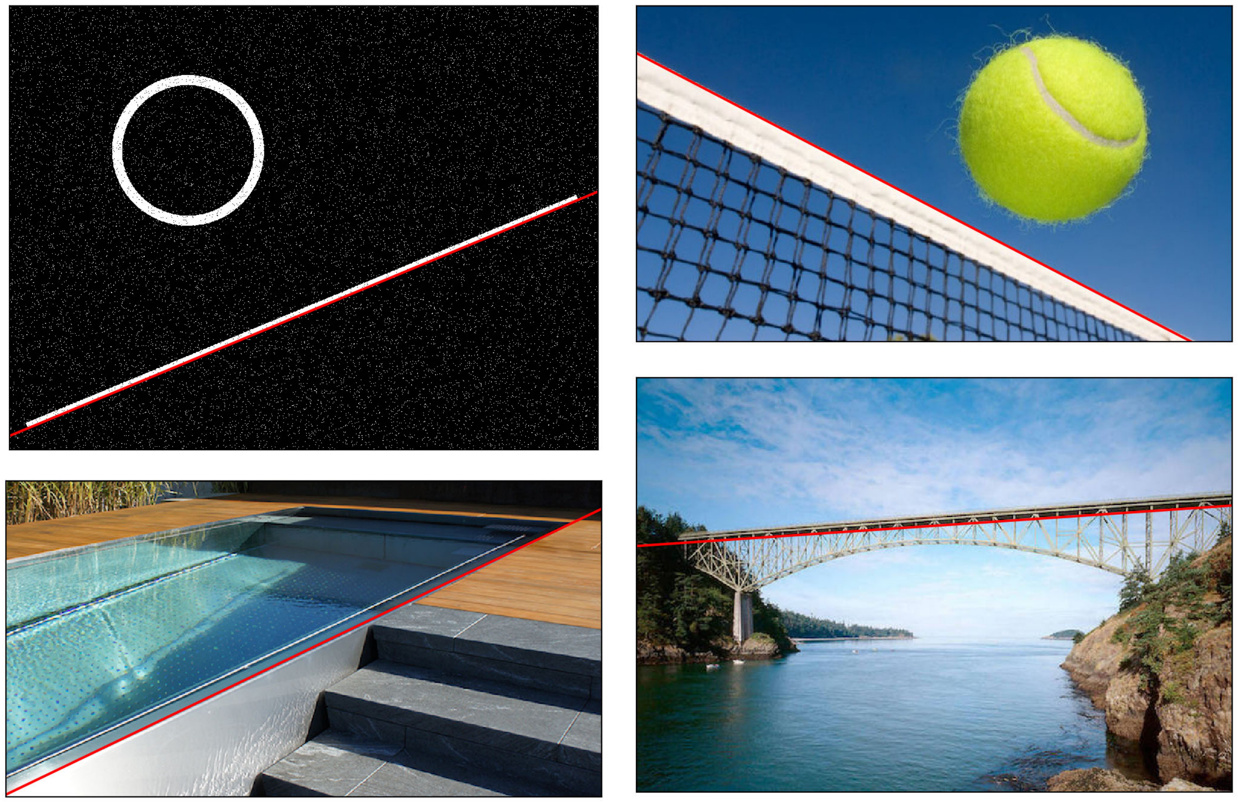

Figure 1: Results of correct RANSAC implementation of provided example images.

1.1 Implement a function `fit_line` that fits a line $y = mx + c$ through two given points (x_0, y_0) and (x_1, y_1).

1.2 As loss function, we use the distance of all points to the line. Implement a function `point_to_line_dist` that computes the minimal distance between point (x_0, y_0) and a line defined by $y = mx + c$.

1.3 As a preprocessing step, implement the function `edge_map` that computes the edge map on an image. You are allowed to use built-in functions like `skimage.feature.canny` and `skimage.color.rgb2gray`. Try different values for the smoothing parameter `sigma` (e.g., 1, 3, 5, 10, 25). Select a value appropriate for RANSAC. Justify your choice.

1.4 Iterate through steps 1 and 2 500 times. For each iteration, the 2 points from which the line is generated are chosen randomly among all possible points on the edge map. The script then chooses the best fitting line and plots it on the image. As a sanity check, run your RANSAC algorithm on sample image `synthetic.jpg`. Plot and comment your results.

1.5 Run your script with sample images `bridge.jpg`, `pool.jpg` and `tennis.jpg`. Plot and comment your results.

2 Texture Synthesis [4 points]

In this exercise, you will implement a simplified version of the texture synthesis approach of Efros and Leung [1]. Template code `hw3_ex2_template.py` and sample images are provided.

To each image corresponds a `*_fill.jpg` and a `*_texture.jpg` image. These images are binary masks corresponding to the regions to fill (also refered to as masked region) and the regions where the texture will be sampled from, respectively. Your script will produce a figure similar to fig. 2.

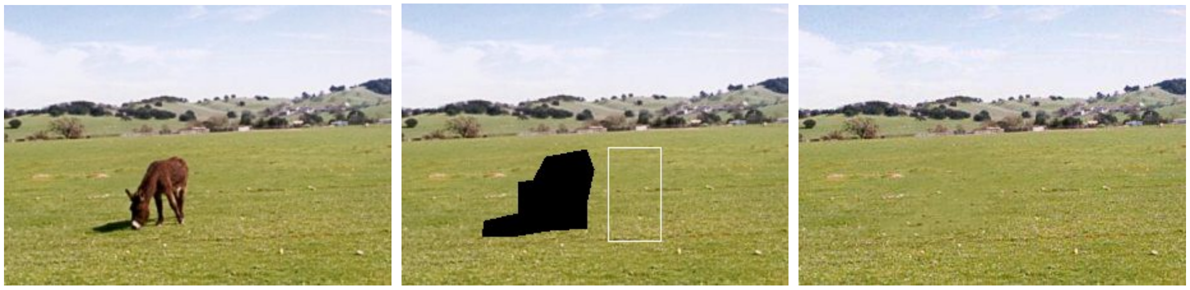

Figure 2: Left: original image `donkey.jpg`. Middle: image with region to be filled (black region) and sample texture region (white rectangle). Right: resulting image you will get at the end of this exercise.

The general idea of the algorithm is given here:

- Randomly select a point $\boldsymbol{r}$ on the boundary of the masked region.
- Let $P_{\boldsymbol{r}}$ a patch centered on $\boldsymbol{r}$ of size $W \times W$. Find another patch $Q_{\boldsymbol{r}'}$, of equal size, centered on $\boldsymbol{r}'$, in the texture region, such that a dissimilarity measure $d(P_{\boldsymbol{r}}, Q_{\boldsymbol{r}'})$ is minimized.
- Let $P_{masked,\boldsymbol{r}}$ the set of pixels in $P_{\boldsymbol{r}}$ that are masked. Replace each pixel $P_{masked,\boldsymbol{r}}(i,j)$ with pixel $Q_{\boldsymbol{r}'}(i,j)$. With $(i,j) \in [0,W]^2$
- Repeat until there are no more masked pixels.

[1]This technique of copying a whole patch is much faster than the center pixel as suggested in [1]. We are also omitting Gaussian weighted windowing. The results are therefore of inferior quality.

To do that, you may follow these steps:

2.1 Write a function `compute_ssd` that computes the Sum of Squared Difference (SSD) between a patch and a texture region, for each possible location of the patch within the texture region. It will ignore the masked pixels of `*_fill.jpg`. Please note that:

- `patch`, has size `[2 * patch_half_size + 1, 2 * patch_half_size + 1, 3]`.
- The texture region `texture`, has size `[tex_rows, tex_cols, 3]`.
- `mask` specifies which elements in `patch` will be filled in. It contains a 1 for each masked pixel. Its width and height are the same as `patch`. When computing the result, ignore the masked pixels.
- The output array `ssd`, is of size `[tex_rows - 2 * patch_half_width, tex_cols - 2 * patch_half_width]`, i.e. it is only defined where the patch completely overlaps the texture.

2.2 We now take the SSD image created above and chose the best match. This best match will be used to replace values in the masked region. Write a function `copy_patch` which copies this selected patch into the final image at the appropriate location. Replace pixel values only in the masked region of the image, i.e. unmasked pixels should be left as is.

2.3 Plot the image after texture synthesis. The original image, the masked image, and the sample texture region, as in fig. 2). Apply your texture synthesis script to `yacht.jpg` and `tomato.jpg`. Show your results and give a short analysis.

References

[1] Alexei A Efros and Thomas K Leung. Texture synthesis by non-parametric sampling. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1033–1038. IEEE, 1999.