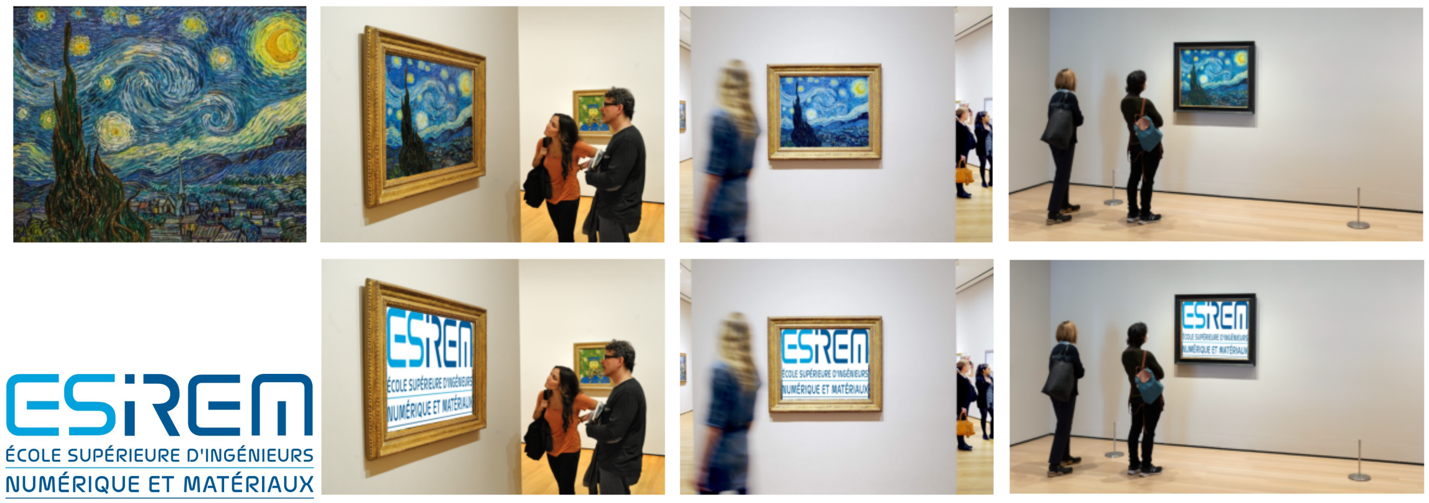
Computer Vision – Visual Perception
MSCV VIBOT & ESIREM Robotique
Renato Martins - Universite de Bourgogne, January 2023

Assignment II – Object Recognition & Augmented Reality with Homographies

In this assignment, we will practice object recognition and robust model fitting with RANSAC for homography estimation. We want to locate a given reference object in another image of a scene which contains that object (and eventually many other objects). Applications are on augmented reality or in a robotics context of we want to make a robot containing a camera to recognize and locate a given object in space to grasp it. We will start considering planar objects, and very salient objects to find, such as a painting. For instance, you will try to locate the painting "Nuit étoilée" of Van Gogh (the reference image, shown in the left most picture) in other publicly available image views in the MoMA museum (first row). We will then locate and fit an homography transformation between the reference view of the painting and its new locations in the other views and then replace it with another picture...such as the logo of ESIREM (second row of images) as follows:



For doing that automatically with the concepts seen in the class, we first will detect keypoints, extract features and then match the points between the different images. Then, from these candidate point correspondences, you will use RANSAC (**and implement it**) to estimate a robust transformation between the reference painting view (left most image) and the different target views, such as to be robust to wrong matches. Many of the steps of this assignment can be performed with tensors in PyTorch (for you to practice). Please notice you are NOT allowed to use any high level OpenCV implementation during this exercise (unless stated). However, you can have use them for debug purposes (verification of results), as well as the tips from OpenCV tutorials on image matching[1] and homography estimation[2].

[1] `https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html`
[2] `https://docs.opencv.org/3.4/d1/de0/tutorial_py_feature_homography.html`

Part I - Environment Setup & Useful Commands

Please create a notebook on Colab or follow these steps to configure your conda environment if you are worning in your local machine:

```
>> conda create -n cv_recognition
>> conda activate cv_recognition
>> conda install jupyter
# Go your workspace directory of the course and run jupyter
>> jupyter-notebook
```

We will use mostly PyTorch tensors and numpy arrays during this exercise. So do not hesitate to check some definitions on the tutorials on PyTorch[3] and for Numpy. There are also some helpful "Cheat Sheets" of basic commands for Numpy and Scipy (Linear algebra in Python)[4]. One important operation we will for visualization is to make tensor conversions between Numpy and Tensors in PyTorch. One useful operation we will often make is to convert from Numpy arrays to torch tensors (and vice versa) as well as sending tensors between the CPU to GPU (if available):

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
import torch

# Determine device to run on (GPU vs CPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Running tensors in", device)

# numpy array to torch tensor
def to_torch_tensor(x, device = "cpu", dtype = torch.float32, requires_grad = False):
    #return torch.from_numpy(x).to(device)
    return torch.tensor(x, dtype=dtype, requires_grad=requires_grad).to(device)

# torch tensor to numpy
def to_numpy_array(x):
    return x.detach().cpu().numpy()
```

We are proving the images in the folder **data/**, where the reference painting view is named *img_ref.jpg*.

Part II - Extracting Features and Matching using OpenCV

To establish correspondences between the objects in the images, we will detect and match keypoints. Keypoints are points in the image which are expected to be reliably matched, such as corner points or blobs (as seen in the class). In this first part of the exercise, you ARE ALLOWED to use OpenCV functions (detectors and descriptors). There are many different keypoint detectors implemented in OpenCV. We will test with SIFT detector and descriptor:

1. Write a function *extract_features* that receives an image and returns a list of detected keypoints and the extracted features with SIFT on each image using SIFT default parameters.
2. Display the detected keypoints on the images. Hint: You can use for that the function *cv.drawKeypoints* with *flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS*
3. How many points do you get in each image?

[3] `https://pytorch.org/tutorials/`

[4] Scipy: `http://datacamp-community-prod.s3.amazonaws.com/dfdb6d58-e044-4b38-bab3-5de0b825909b`
Numpy: `http://datacamp-community-prod.s3.amazonaws.com/ba1fe95a-8b70-4d2f-95b0-bc954e9071b0`

4. Is it a way of selecting the "best" keypoints? (hint: please look the fields in cv.Keypoint) Modify your algorithm to return the N best points and then display the best 1000 points for each image.

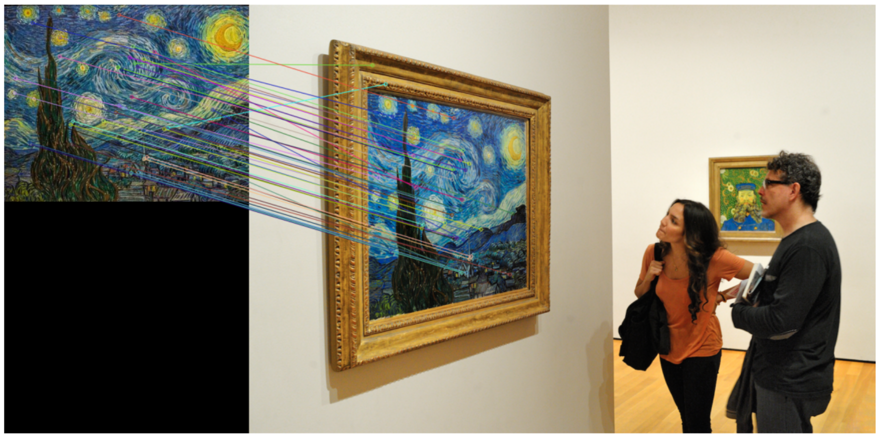
5. Please now change the parameters from SIFT to *contrastThreshold=0.02, nfeatures=1000.* Are there differences regarding the number of detected keypoints using the default parameters?

Once we have detected keypoints and extracted features related to them, we will now perform the matching to find corresponding visual features. For each detector/descriptor:

1. Please write a function *find_matches* that receives a pair of images, their corresponding keypoints and descriptors and that returns the found matches. For finding the correspondences between the features using brute force matching (one to all). The distances between the descriptors should be the Euclidean distance between the descriptions with SIFT features;

2. Implement yourself the 1NN/2NN ratio test for removing some ambiguous correspondences inside the function *find_matches* (check the slides of the course) and OpenCV matching tutorial for possible implementations[5];

3. Please display the found correspondences between each pair of images inside the function *find_matches*. Hint: For displaying keypoints we can use *cv.drawMatchesKnn* with *flags = cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS*. Your result should be something similar to the following visualization between the *img_ref* and *img_2*:



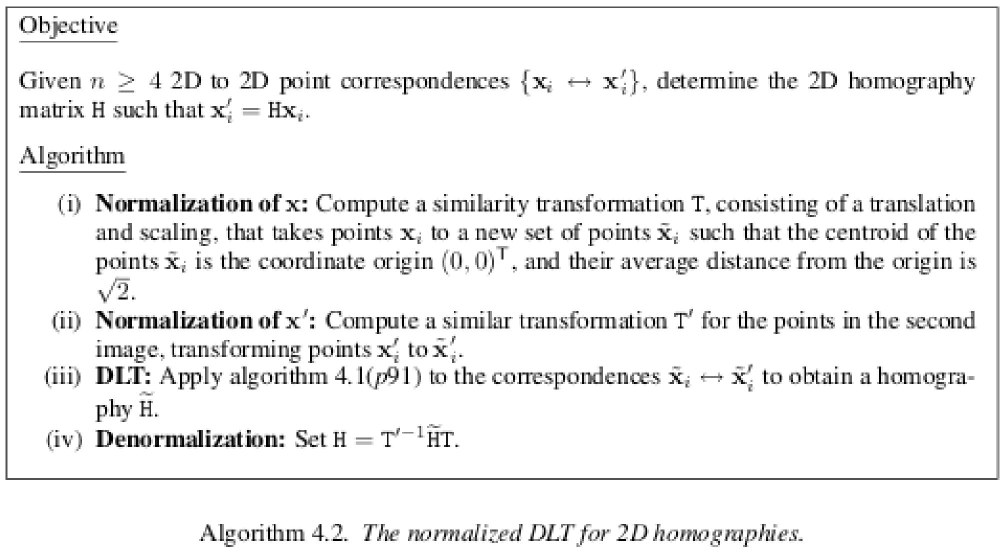
Part III - Robust 2D Homography Model Fitting

We will now estimate a transformation between each two images based on the found matches in the previous step. Since we are observing a planar object this transformation can be represented by an homography. From the corresponding features between the image of the object and the image of the scene, and based on the matched features, we will then be capable of finding the object's position in the different views.

1. Check in the course slides the model fitting constraints for fitting a 2D Homography transformation model. How many corresponding 2D points are required?

[5]`https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html`

2. The homography model fitting estimates a 3x3 matrix that handles correspondences between planar points such as from two images (2D–2D). We will adopt again the linear DLT algorithm (similar to the one used for camera Calibration). Implement the algorithm DLT following the recipe of the following algorithm shown in Figure 1. What is the minimal number of correspondences to find the homography transformation?



Objective

Given $n \geq 4$ 2D to 2D point correspondences $\{\mathbf{x}_i \leftrightarrow \mathbf{x}'_i\}$, determine the 2D homography matrix H such that $\mathbf{x}'_i = \mathtt{H}\mathbf{x}_i$.

Algorithm

(i) **Normalization of x:** Compute a similarity transformation T, consisting of a translation and scaling, that takes points $\mathbf{x}_i$ to a new set of points $\tilde{\mathbf{x}}_i$ such that the centroid of the points $\tilde{\mathbf{x}}_i$ is the coordinate origin $(0,0)^\mathsf{T}$, and their average distance from the origin is $\sqrt{2}$.

(ii) **Normalization of x′:** Compute a similar transformation T′ for the points in the second image, transforming points $\mathbf{x}'_i$ to $\tilde{\mathbf{x}}'_i$.

(iii) **DLT:** Apply algorithm 4.1(*p*91) to the correspondences $\tilde{\mathbf{x}}_i \leftrightarrow \tilde{\mathbf{x}}'_i$ to obtain a homography $\tilde{\mathtt{H}}$.

(iv) **Denormalization:** Set $\mathtt{H} = \mathtt{T}'^{-1}\tilde{\mathtt{H}}\mathtt{T}$.

Algorithm 4.2. *The normalized DLT for 2D homographies.*

Figure 1: Homography estimation using DLT recipe.

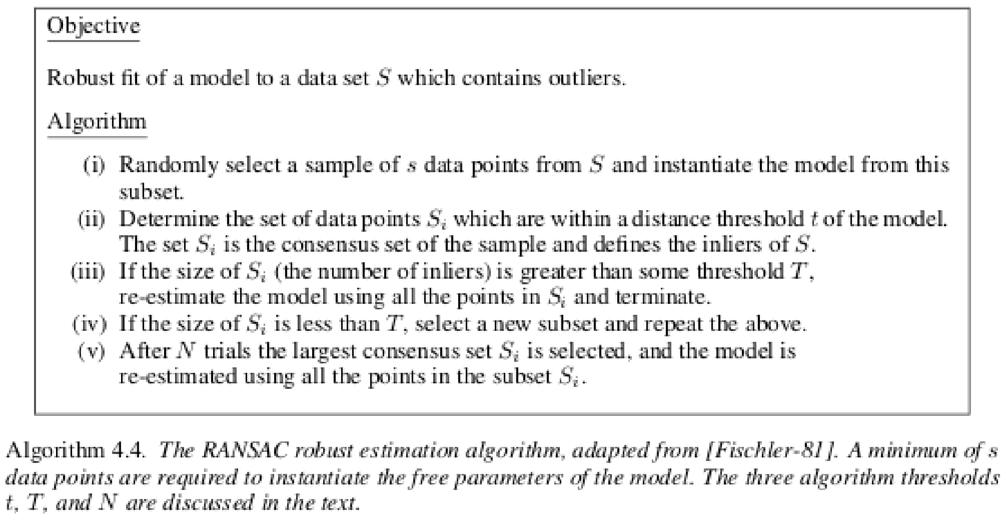
3. Implement your own RANSAC algorithm for estimating the homography matrix following the RANSAC recipe algorithm shown in Figure 2.

4. Discuss and indicate the parameters you need to select in order to compute one iteration of RANSAC to fit your descriptors matches. How many iterations would you select for finding the model with either 95% and 99% confidence?

5. Replace the painting on each target image with the ESIREM logo using the estimated homographies from the previous step Hint: For warping the image with the homography and then blending the images you can use/adjust the following function:

```python
# Function to warp and replace texture into new image given an homography transformation
def render_warped_texture(H, img_ref, img_tgt, patch_texture):

    ## warp patch texture that will be placed in the selected region with
    # the estimated homography transformation H
    h, w, _ = img_ref.shape
    patch_texture_res = cv.resize(patch_texture,(w,h))
    warped_patch = cv.warpPerspective(patch_texture_res, H, (target.shape[1],target.shape[0]))
    ## remove the pixels of the foreground that will be replaced (we keep only background)
    mask = (255*np.ones((h,w))).astype(np.uint8)
    mask_warped = cv.warpPerspective(mask, H, (img_tgt.shape[1],img_tgt.shape[0]), flags=cv.INTER_NEAREST)
    mask_background = (1 - mask_warped/255).astype(np.uint8)
    mask_background = cv.merge((mask_background, mask_background, mask_background))
```



Objective

Robust fit of a model to a data set S which contains outliers.

Algorithm

(i) Randomly select a sample of s data points from S and instantiate the model from this subset.
(ii) Determine the set of data points S_i which are within a distance threshold t of the model. The set S_i is the consensus set of the sample and defines the inliers of S.
(iii) If the size of S_i (the number of inliers) is greater than some threshold T, re-estimate the model using all the points in S_i and terminate.
(iv) If the size of S_i is less than T, select a new subset and repeat the above.
(v) After N trials the largest consensus set S_i is selected, and the model is re-estimated using all the points in the subset S_i.

Algorithm 4.4. *The RANSAC robust estimation algorithm, adapted from [Fischler-81]. A minimum of s data points are required to instantiate the free parameters of the model. The three algorithm thresholds t, T, and N are discussed in the text.*

Figure 2: RANSAC algorithm recipe.

```
    background = cv.multiply(mask_background, img_tgt)

    # blend images to a single frame and display
    blend_img = cv.add(background,warped_patch)
    plt.figure(figsize=(20,10))
    plt.imshow(cv.cvtColor(blend_img, cv.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()
```

6. Compare your homography solution with the OpenCV implementation *cv.findHomography* (with the RANSAC flag deactivated), and using your own RANSAC routine in the estimation.
7. **(EXTRA POINTS)** Perform the previous items with FAST detector and ORB descriptor. What can you observe of the performance between SIFT wrt ORB features for these images?

Submission

Please return a PDF report explaining your reasoning and equation developments. You should also submit your python notebook (commented) with the corresponding implementations, together inside a [name]_assign2_recognition.zip file (replacing [name] with your name ;-)). The submission should be done via the teams channel of the course using teams assignment.
Deadline: 08/02/2023 at 23:59pm.

Note: This assignment is individual. Plagiarism such as copying the work from another source (student, internet, etc.) will be awarded a 0 mark. In case there are multiple submissions with the same work (or partial), each one will receive a 0.

References

[1] Hartley, Richard, and Andrew Zisserman. "Multiple view geometry in computer vision." (2003).