Exercise 2: Video processing

Multimedia systems

2018/2019

Create a folder `exercise2` that you will use during this exercise. Unpack the content of `exercise2.zip` that you can download from the course webpage to the folder. Save the solutions for the assignments as the *Matlab/Octave* scripts to `exercise2` folder. In order to complete the exercise you have to present these files to the teaching assistant. Some assignments contain questions that require sketching, writing or manual calculation. Write these answers down and bring them to the presentation as well. The tasks that are marked with ★ are optional. Without completing them you can get at most 75 points for the exercise (the total number of points is 100 and results in grade 10). Each optional task has the amout of additional points written next to it, sometimes there are more optional exercises and you do not have to complete all of them.

Introduction

In the course of this exercise you will familiarize yourself with four common video manipulation tasks. In the first assignment you will perform stabilization of a shaky video. In the second assignment you will detect the most salient part of the video and center on it, the third assignmentw will deal with shot transition detections. You will implement a video summarization method in the fourth assignment.

Assignment 1: Video stabilization

Video stabilization is the process of estimating the motion between consecutive frames, and applying an image transformation that either cancels out the motion or diminishes its effects. It is important to note that video stabilization reduces the effective image size, so one should not consider it to be a substitution for a proper video capture procedure.

The goal of this assignment will be the compensation of translational vibrations, which cause a video frame from time t to be shifted according to vector $v_t = (x_t, y_t)$. The underlying idea is as follows:

(a) First, you need to select a stable point in the image, preferably one that can be tracked throughout the whole length of the video.

(b) Having chosen the point to track, you need to estimate its motion trajectory throughout the video, relative to its position in the first frame.

(c) Compensate for motion in each frame by translating the image by the negative value of estimated relative position of the tracked point, and gather the translated frames in new, stabilized, video.

The tracking of the selected point can be achieved by using the *normalized cross-corelation* (NCC) method. Using this method, you will compare the patch around the selected point with patches in the subsequent images, and find the best match. The normalized cross-correlation between the *template* patch and a given image patch is computed using the following formula:

$$\frac{1}{n}\sum_{x,y}\frac{(f(x,y)-\overline{f})(t(x,y)-\overline{t})}{\sigma_f\sigma_t}, \tag{1}$$

where $t(x,y)$ denotes the template, and $f(x,y)$ denotes the image patch that we are comparing to the template. n denotes the number of pixels in the patch (both template and image patch are of same dimensions); $\overline{f}$ and $\overline{t}$ denote the mean, while σ_f and σ_t denote the standard deviation of the image and the template patch, respectively. This way one can determine the position of the template inside a larger image by computing the *similarity scores* between the template and all patches inside the image. The assignment consists of the following steps:

(a) As the first part of the assignment, familiarize yourself with computation of similarity score via a simpler, non-normalized, cross-correlation:

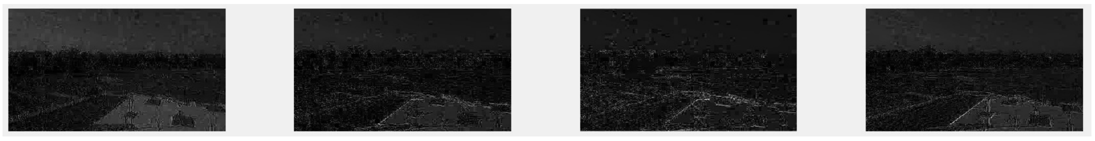
$$\sum_{x,y} f(x,y)t(x,y). \tag{2}$$

Given the template A, we wish to find the best matching patch inside the matrix B. Compute the similarity scores (cross-correlation) for all patches inside B, and find the best match.

$$A = \begin{array}{|c|c|}\hline 1 & 3 \\ \hline 7 & 1 \\ \hline\end{array} \quad B = \begin{array}{|c|c|c|c|}\hline 2 & 0 & 1 & 2 \\ \hline 7 & 1 & 4 & 6 \\ \hline 5 & 2 & 0 & 4 \\ \hline\end{array}$$

(b) Create a script that reads the video from a sequence of images using a `read_video` function that is available in the supplementary material and displays it on screen in frame-by-frame manner.

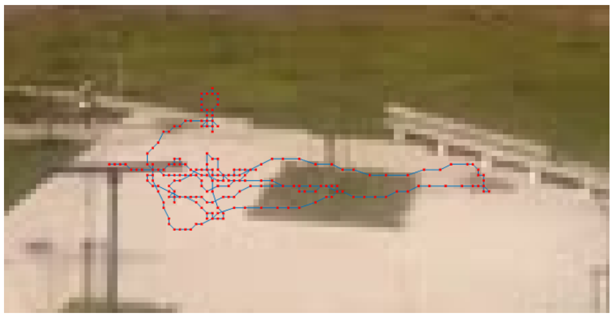
Visualize the video in another manner using function `imshowpair` that allows you to compare two images, in our case we will visualize the difference between two consecutive images. The function supports many visualization modes, if you select mode `diff` it will display the per-pixel differences for two images. Can you use this visualization to determine which regions will be good candidates for tracking? Motivate your answer.



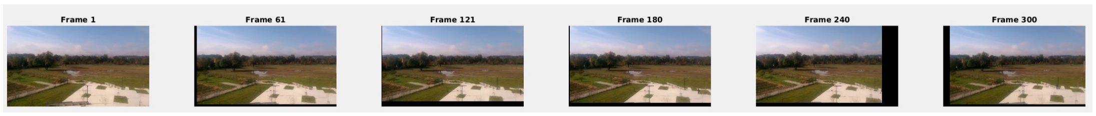
(c) Write a function `track_point`, which, given the video data, coordinates of the initial point, size of the surrounding template patch, and the size of the search neighborhood, computes and returns the trajectory that corresponds to motion of the point throughout the whole video sequence. To find the best template match in subsequent frames, use the built-in function `normxcorr2` that calculates a normalized cross-correlation in 2D matrices. You can use the following code skeleton:

```
function [trackX, trackY] = track_point(frames, startX, startY, patchSize, searchSize)

[height, width, ~, n] = size(frames);

image = rgb2gray(frames(:, :, :, 1)); % convert image to grayscale
x1 = max(1, startX - patchSize / 2);
y1 = max(1, startY - patchSize / 2);
x2 = min(width, startX + patchSize / 2 - 1);
y2 = min(height, startY + patchSize / 2 - 1);
patch = image(y1:y2, x1:x2); % cut the reference patch (template) from the first frame

% initialize the output
trackX = zeros(1, n);
trackY = zeros(1, n);
trackX(1) = startX;
trackY(1) = startY;

for i = 2:n
    % TODO: cut a region of seachSize times searchSize with the center in the point's previous ↩
        position (i-1).
    % TODO: convert the region to grayscale
    % TODO: compare the region to template using normxcorr2
    % TODO: select best match (maximum) and determine its position
end
```

Create a script that reads the video `shaky`, displays the first frame, and asks the user to pick the center point. Afterwards, use the previously-written function `track_point` to obtain the point's trajectory and display it over the last frame of the video sequence. Use the built-in function `ginput` to let the user pick the initial point. Experimentally determine the template patch size and the search-neighborhood size that result in stable tracking of the point.

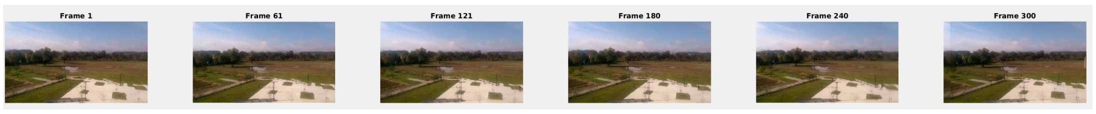


(d) Create a new script, in which you make use of the trajectory obtained by `track_point`. The script should create a new video sequence in which the original images are translated by offsets that correspond to negative estimated motion vectors. In other words, for each frame from original video, create an image of the same size and copy the original pixel elements with an offset that you obtain from the computed trajectory. If no pixel from the source image maps to a pixel in the destination image, that pixel should be left black. Implement the image transformation via matrix operations (avoid using explicit loops).



The result should be a sequence of images in which the selected point remains static. Write the video sequence to disk frame by frame using function `imwrite`. Then use one of the video processing tools[1] to convert the sequence to video for easy playback. Bring the video with you to the presentation of the exercise.

(e) ★ (5 points) Currently the stabilized video is filled with a lot of black area around the borders which is the result of the fact that we have to shift the images in order to compensate for the motion. But since the scene in our test video is more or less stationary, at least some of the information about these regions can be recovered from the overall video. Implement and demonstrate an algorithm that fills these black pixels with information obtained from a mean image obtained by stabilizing the entire video and computing a mean over pixel values for all images (it is a good idea to assign value `NaN` to all pixels that are not available in a certain stabilized image and then use function `nanmean` that ignores these values when calculating mean so that these values do not influence the estimate).



(f) The kind of stabilization that we have obtained so far is not always desirable; depending on how unstable the source video is, we may end up discarding significant parts of images. In addition, completely static video sequences tend to look unnatural. As our main goal is compensation of sudden movements, we will upgrade our stabilization with smoothing. Modify the script from the previous step so that it will use function `smooth_trajectory` to smooth the trajectory before using it for stabilization. Save the resulting video file to `smooth.avi`. Explain how function `smooth_trajectory` performs smoothing. Use both *Matlab/Octave* documentation, and experiment with the function itself — modify the algorithm, use different inputs, and so on.

(g) ★ (10 points) In a lot of cases tracking only a single point may not be desirable, the tracking can fail or can be unreliable. It is better to look at more points and calculate their consistency for each frame to estimate motion. Implement and demonstrate an algorithm that tracks more (three or more) points and at every frame selects the ones that are agreeing on translation of the image (point in the same direction) and computes their average estimate. Then render a stabilized video using these more robust estimates.

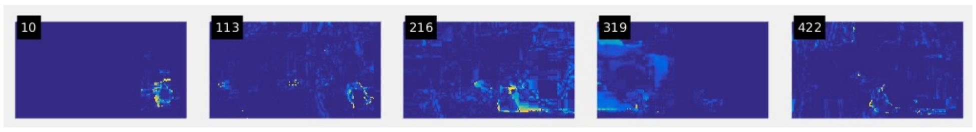
Assignment 2: Video centering

The goal of this assignment is to crop the input video sequence, centering it on the currently most salient (interesting) region. This *region of interest* will be determined by a

[1]Package `video` in Octave or class `VideoWriter` in MATLAB

simple mechanism of computing the difference of corresponding pixels in two consecutive frames. In order to avoid drifts due to illumination changes, we will look for the differences in the hue part of the HSV color space.

(a) Create a script that reads the video from directory `bigbuck`. Afterwards, it should iterate over the video, convert pairs of consecutive frames into HSV color space, extract their *hue* components, and compute the absolute differences of all value pairs. Display the result using the `imagesc` function.

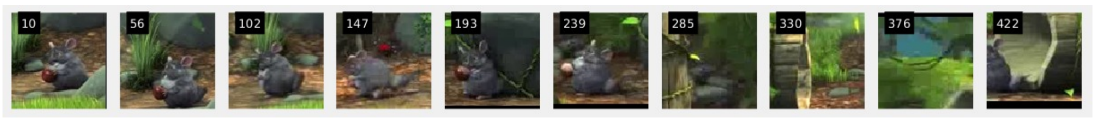


(b) Create a script that iterates over the video and smooths the absolute differences with the built-in `imfilter` function.

In each smoothed absolute difference of *hue* components for two consecutive frames, determine the point with the largest difference value. Store the corresponding coordinates in two vectors, which determine the trajectory. Experiment with different kernel size and determine the value for which the obtained trajectory most closely follows the happening in the video.

Question: Why is it a good idea to smooth the response before trying to find the maximum? Try finding the maximum both with and without smoothing, and write down your explanation on why smoothing is useful.

(c) Write a new script that makes use of the obtained trajectory; it should crop each frame to a fixed-size window of 100 pixels, centered around the current point on the trajectory. Store the resulting sequence and convert it to a video file and bring it with you to the presentation of the exercise.



(d) Use the function `smooth_trajectory` to smooth the trajectory and improve the results qualitatively, and repeat the video cropping procedure.

(e) ★ (10 points) Improve the change detection procedure with an idea of your own. You can increase the robustness by combining multiple visual cues (e.g. changes in different color spaces and channels) or enforcing constraints on movement between consecutive frames. Your algorithm should noticeably improve the performance of the baseline method on the test video, but you can also demonstrate its behavior on some other video of your own. Convert the resulting sequence to a video file and bring it with you to the presentation of the exercise.

Assignment 3: Detection of shot transitions

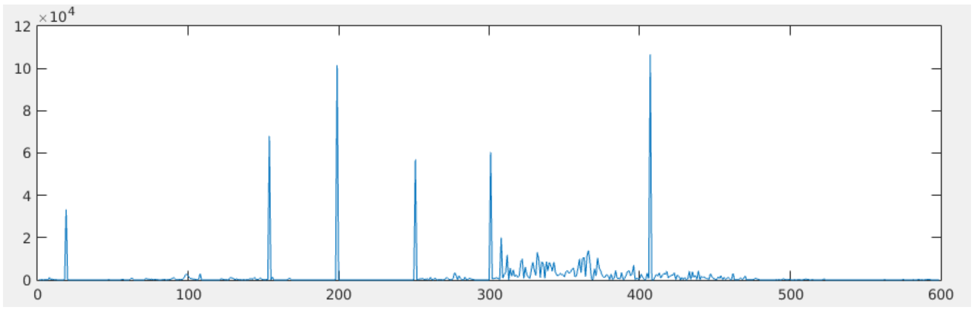
Sudden changes in shots can be detected by observing the differences between two consecutive images in the video sequence. If the difference is small, it can most likely be attributed to changes within the shot; if the difference is large, it is most likely due to the transition between two shots. You will implement a simple detector of shot transitions based on histogram comparison. Note that this is a very simple algorithm, and if you wanted to use it in real-life applications, you would likely need to combine it with other modalities.

To compute the distance between two histograms you will use Hellinger distance that is defined as:

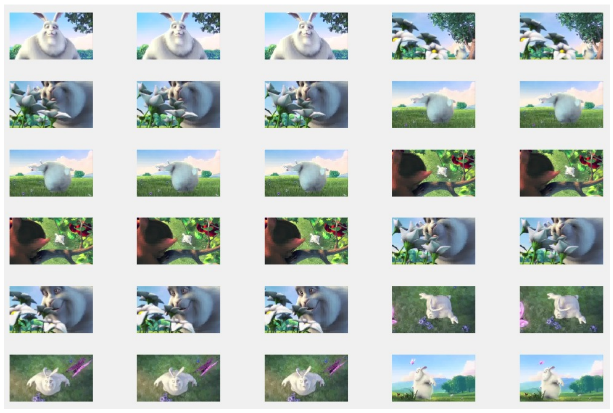
$$H(\mathbf{h}_1, \mathbf{h}_2) = \sqrt{\frac{1}{2}\sum_{i=0}^{N-1}\left(\sqrt{h_1(i)} - \sqrt{h_2(i)}\right)^2}. \tag{3}$$

Note that low values of Hellinger distances signify high similarity and high values signify low similarity.

(a) Use the function `read_video` to load the video sequence `bigbuck` into the memory (note that you need to do this only once when testing your code, as long as the resulting matrix of video frames remains loaded in your workspace). For each image in the video sequence, compute separate histograms for red, green, and blue channel, and then concatenate them into a single vector. Start with 8-bin histograms, and later experiment with different number of bins. Use the function `imhist` if it is available in your *Matlab/Octave* installation to speed up the computation. Use one of the histogram-distance measures (e.g., Hellinger or χ^2 distance) to compute similarities between histograms of consecutive image pairs. Store the results for the entire sequence to a vector, and visualize them using the `plot` function.



(b) Based on the obtained inter-frame histogram distances, estimate a threshold that will allow you to detect shot transitions. A straightforward automatic way of determining this threshold is to set the threshold to a certain portion (e.g., 20%) of the maximum value. This approach has several limitations (can you think of some?), but illustrates the general idea. Using the estimated threshold value, determine the positions of shot transitions, and visualize them by displaying five consecutive frames around each transition. Display all transitions in rows of a single figure, with each row corresponding to a single transition.

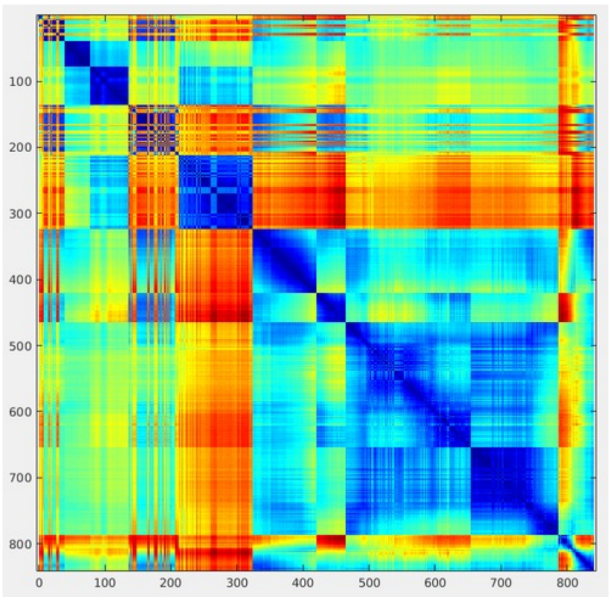


(c) ★ (10 points) Replace the global threshold with a more robust alternative. The actual implementation is left to you, however, here are some hints: you can implement non-maxima suppression to eliminate double peaks, and use median filter to obtain a dynamic threshold. To complete this task, you have to test your method on another video sequence with multiple shots (not part of the material) and demonstrate that your proposed method can detect at least some of the transitions that the baseline method fails to detect. Note that you do not need to load the entire video at once; you can load and process two consecutive images at a time to avoid running into memory or storage issues.

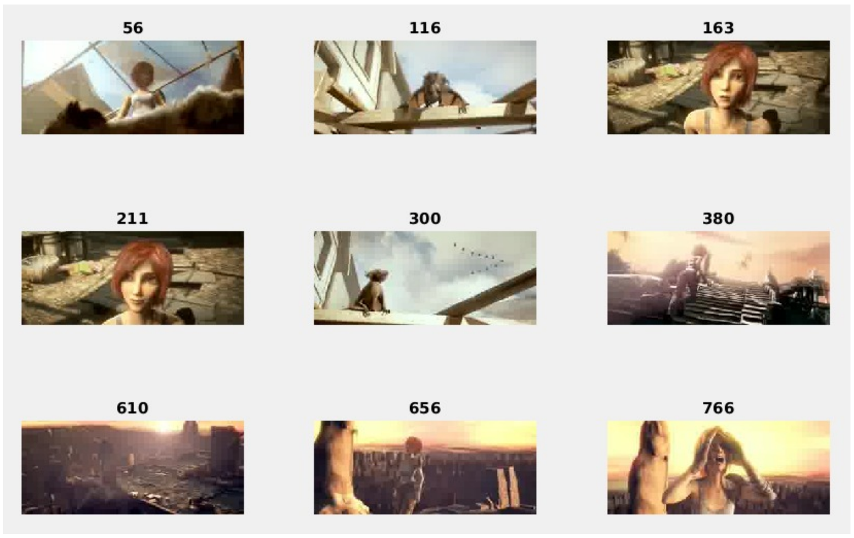
Assignment 4: Video summarization using color histograms

In this assignment, you will implement a video summarization system based on clustering of color histograms. The motivation for such system is to create a quick video summary using a set of static images that still reflect the structure of the video and its story.

(a) Load the video sequence `sintel` into the memory; convert each image from the RGB to the HSV color space, and obtain its combined histogram by computing histograms for individual channels and concatenating them into a single vector (use 8-bin histograms). Obtain a pair-wise distance matrix by computing the distances between all images in the video sequence (using a distance metric of your choice). As this can be quite slow, try to optimize the code; at least take into the account the symmetric nature of the metric, which saves you a half of the operations (the distance between histograms i and j and is the same as the one between j and i). Visualize the resulting distance matrix using the function `imagesc`.



(b) Having obtained the distance matrix, we can use the *affinity propagation* algorithm to automatically group images into clusters based on their mutual distances. The affinity propagation algorithm is available as a function `apcluster` in the supplementary material. The algorithm expects an input in form of a similarity matrix[2] and a preference that a given frame is selected as a representative for a cluster. As we do not have any preference for specific frames, we can give the algorithm a single value, which will be used for all frames. A typical choice of the preference value is either the median of all distances, or the minimum distance; however, it sometimes needs to be further fine-tuned, as it influences the number of resulting clusters. Find a value that gives you a reasonable number of clusters (for the test video, it is in the range between 5 and 10). Display the images that have been selected as representative samples in a single figure using the function `subplot`; use function `title` to label each image with its index in the video.



[2]Note that similarity has the inverse properties of distance; for similar elements, the distance will be small, and the other way around. As the affinity propagation algorithm also accepts negative values, it is safe to simply invert the values in your distance matrix to make it suitable for the algorithm.

(c) Upgrade your method so that it will export a video (or a sequence of images that you later convert to a video) that contains a representative thumbnail for each frame's cluster in the top-left corner. You can use function `imresize` to resize the representative image, and then insert it into the current frame using matrix indexing operations.

(d) ★ (5 points) When comparing images using histograms, we are actually comparing vectors in N-dimensional space, where N is the number of cells in a histogram. Experiment with an idea of inserting an additional element into this vector, the one that denotes the normalized position of a frame in the sequence (e.g., 0 for the first image, and 1 for the last one). Are the results different in this case? Compare the difference on an additional sequence of your choice (its length should be around 700–1000 frames, and it should contain multiple shots).