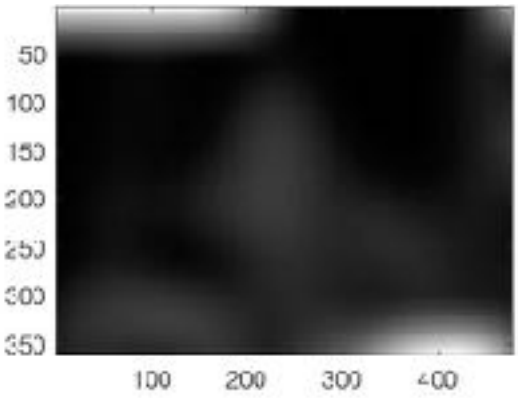
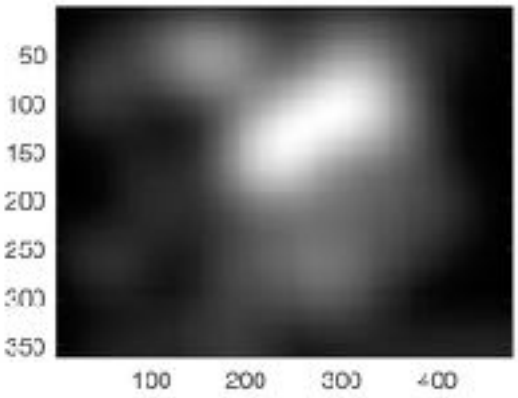
Computational photography and capture
Restoration of old films

Task 1 : **Detection of scene cut** (detectSceneCut.m)

To detect scene cut I used a edge detector combined with a gaussian filter. Indeed, as the frames in a same sequence don't move much, by comparing the edge maps with a sum squared differences between 2 frames, I could detect the big changes which are scene cuts. However, as the images move, I must blur the edges to compensate those movements.
So by computing the edge for each frame and the previous one, by blurring them a with gaussian filter and by comparing them with a SSD, I obtained a value for each frame. Higher is value is, the most chances I have to have found a scene cut. Here are my results for a scene cut for example :

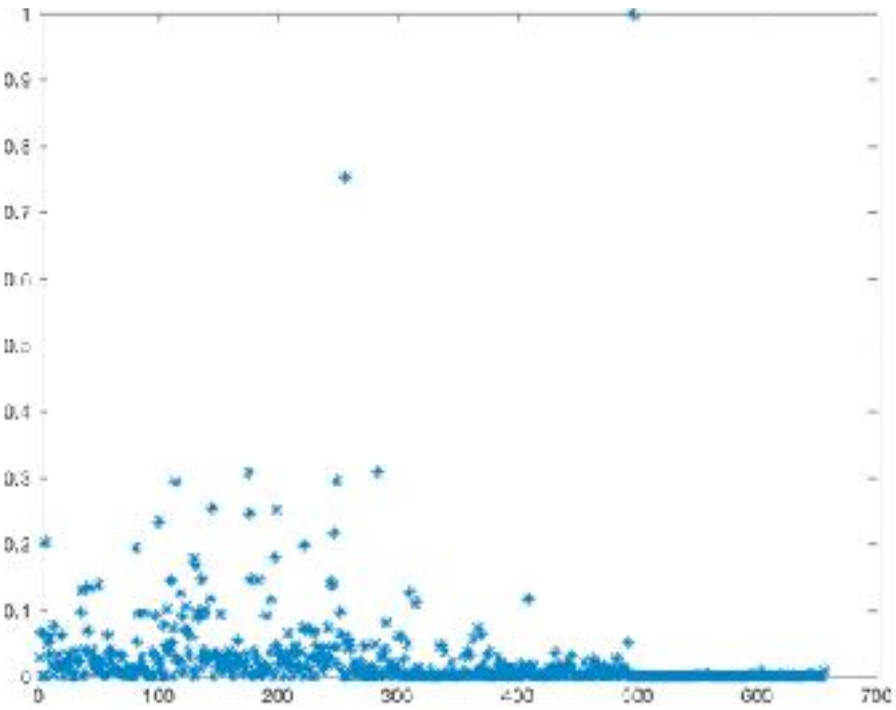


Edge maps



Filtered edge maps

Here we can see the edge maps for 2 frames not in the same sequence and the blur obtained by applying a gaussian filter with a high variance (30), in order to compensate object movements and camera shake. Then, I compute the SSD value between those frames, normalized the values over all the video, and here is what I obtained :



Normalized SSD of the filtered edge maps along the film

We can see 2 values above 0.5, which I arbitrarily choose as my threshold to distinguish scene cuts. The 2 values correspond to frames where a scene cut happened.

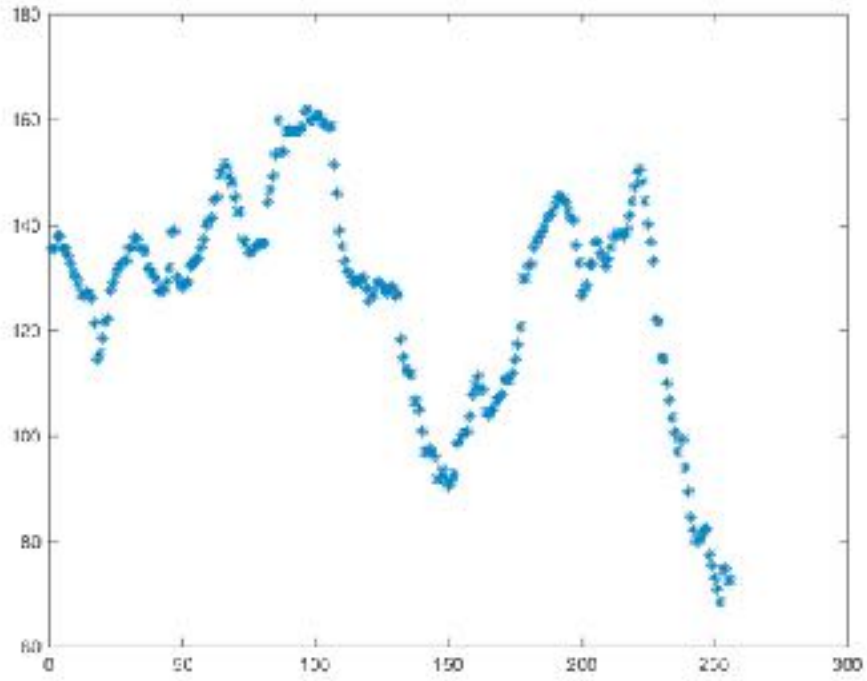
Then, I could compute a new video adding a written information on the frame which start a new sequence like above.

I also returned the frame positions where the scene cuts arrive to use them in future questions.

Task 2 : **Correction of global flicker** (reduceFlicker.m)

For the question, I applied a correction using histogram matching and a computation of the mean and variance of the intensity for each frame in a sequence.
For each sequence, I computed the mean and the variance of the intensity in each frame. Then, I went through the sequence and, if the mean of this frame is above the mean plus the variance of the sequence or below the mean minus variance of the sequence, I corrected it. To correct it, I computed an histogram with the 2 frames before and 2 frames after the corrupted frame and applied an histogram matching method.

First, we can observe how the means of the intensity are displaced in the first sequence.

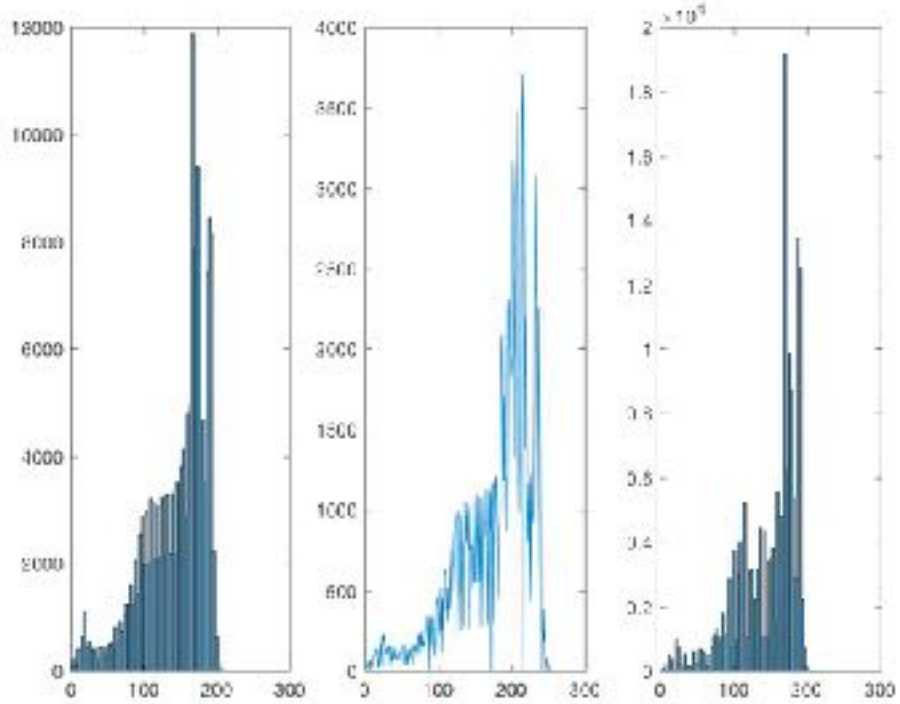


Mean of the intensity of each each frame along the sequence

To know if a frame has to be corrected or not, I computed the mean and the variance for this distribution and if the mean intensity of the frame isn't in [mean-var, mean+var] I corrected it .

To correct it, I use an histogram matching technique. I computed a target histogram composed of the mean of histogram of 2 frames before and 2 frames after. Then, I matched the mean histograms to the flickered frame. To match it I computed a mapping function to go from the source image to the target one. To do so, we needed to first calculate the normalized CDF of the source image histogram and of the target image histogram. The mapping function is then obtained by comparing the values between the cdf source and the cdf target. We had to map both cdf for each value in [0,255] to obtain a out value in [0,255]. To do so, we find the first cdf source value that is greater than a cdf target value, and that value will be the mapping from the source value to the out value.

Here we can visualize from left to right, the previous histogram, the target histogram and the corrected histogram.



Old histogram Target histogram New histogram

Here we can observe the result with the corrected image on the left and the old one on the left.The sky and the building seems less bright now.

Corrected image Original image

However, with this technique, only global changes are corrected and not local changes. In some frames, there are a high increase in intensity in a small area which won't be resolved using this technic. Moreover I don't find the result very impressive. I think it's because we look at the entire image and not a parts of it.

I could also have used an algorithm computing the mean of intensity in all the frames, the variance and then used this distribution to remap the intensities of all the frames.

Task 4 : **Correction of vertical artifacts** (verticalArtefact.m)

In this part we are trying to correct vertical artifacts we can see in the last sequence. Here is an example :

To correct them, I applied a median filter on each row to « wipe » the artifacts away. However, if I only used this technique lots of details are lost. Here is an example where I applied 3 filters with different orders : from right to left 3, 7 and 12.

We can see that artifacts really disappear on the last one but the quality of the image is really damaged.

Order = 3

Order = 7

Order =12

To keep those details, I wanted to keep the original image pixel values where the gradient was high. To do so, i computed the gradient of the image and for each pixel in the image I computed the mean gradient within a small window. If the mean gradient is above a certain threshold, I kept the original image value. If it's below, I put the value from the filtered image.
To find the best combination of values between the threshold, the order of the filter and the window size I computed some test (see the test function). Here are my result for a filter order of 12 :

Lundi 6 février 2017

Maud Buffier

	WindowSize = 3	WindowSize = 5	WindowSize = 7
Threshold = 100			
Threshold = 70			
Threshold = 150			

We can see that if the threshold is to high details are lost because we don't take enough of this original image. Moreover, if the window size is too big artifacts appear on the top of the letters for example. After analyzing the results I thought the best values were a window of 5*5 and a threshold to 70 for the gradient. With those values here is my last result :

Original image

Corrected image

We can see that vertical artifacts are mostly gone and details like « metropolis » are still there. However, some artifacts have appeared on the newspaper name for example.

Task 5 : **Correction of camera shake** (reduceShake.m)

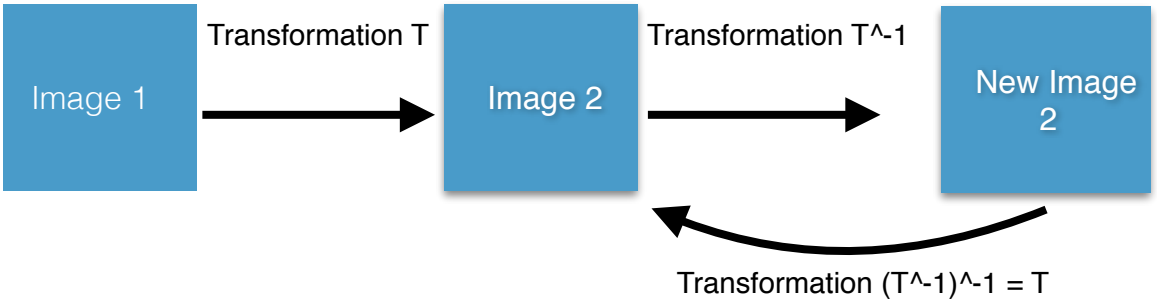
In this part, we were asked to correct the shake of the camera between frames. To do so, I found characteristic points in 2 successive frames using surf matching function into matlab. Then, I applied a threshold to take only matching points which don't move much. Indeed, if a matching point appears on a moving object the result of the movement of my background plan will be false.
Here we can see the first set of matching point and then the second one with the threshold applied :

First set of matching points

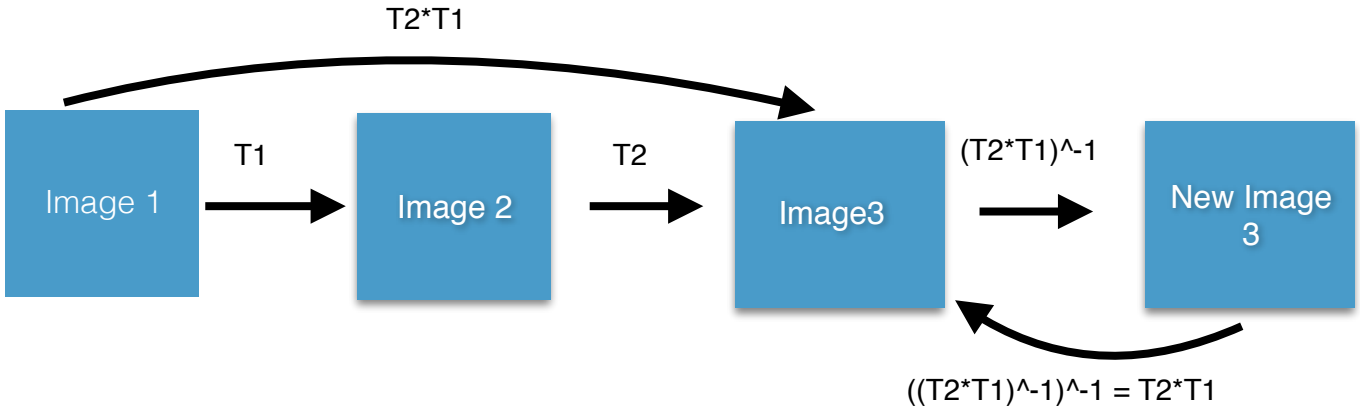
Set of matching points after applying the threshold

We can see that we erased some wrong matching point in the image and most of them are now in the background.
To find the first set of matching points I computed the sum of square differences between 1 descriptors in the first image and all the descriptors in the second image. If the difference between the maximum value in all the SSD and the second maximum value is above a certain threshold, the point associated with the maximum value is a match for the point in the first image.

Then, I estimated a transformation between my first and my second image using the 2 sets of points. To undo the transformation I needed to apply the inverse transform to the second image. However, to avoid losing information I use an inverse process to find my new image as explained here :



To fill the new image 2, I go through all the pixel in new image2, applied the T transform to the pixel position and use an interpolation between the four pixel intensity I landed into the image 2.
To be able to stabilize the camera, I needed to fix all the images with respect to the first one. Here is how I did that :



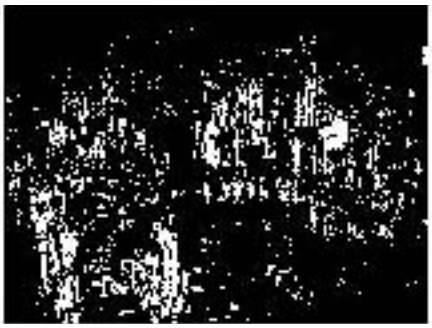
First, I try to fit a non reflective similarity using matlab function fitgeotrans. However, I wasn't really satisfied with the result.

Hence, I decided to fit a better non reflective similarity using RANSAC algorithm. To do so, I extracted 3 corresponding points from both sets to compute a perfect fitting transformation. Using this transformation, I computed the distance between the set of point in the second image in the transformed set of points in the first image. Then, computed the number of corresponding features point with a distance below 1 between them and their matching point with the transformation.
After doing this operation 100 times, I looked at the transformation which gives me the biggest number of inliers. Then, I computed the final transformation using all the inlier of the best transformation of the previous step. Hence, I computed the transformation which best fitted all the points also taking into account the possibility of having outliers in the set. I choose the number 20 because it's gives me always a transformation with a high number of inliers.
I also tried to fit an affine and an homography transformation but allowing too many degres of freedom in the image distorted too much the resulting image.
However, I thought the image would have been more stabilized using the Ransac algorithm. I think as the images are sometimes entirely covered by an object it could explain why it doesn't work well I think the correspondence points are also sometimes false because of the noise and as the distortions are completely random from frames to frames errors accumulate.

Task 6 : **Correction of blotches** (reduceShake.m)

In this section we have to correct the blotches which appear in the video. To do so, I compare each pixel in a frame with the 6 pixels around the same positions in the previous and in following frame. If the value of intensity for this pixel is above the maximum of the 6 pixels around or below the minimum it gives me a measure of distance for this pixel. Hence, higher this distance is, more chances there is to have a bloch in this region. Then, I applied a threshold to have a mask of possible blotches.
To consider the possibility of the detected blotches being noise, I computed 2 thresholds : one very high and one to zero to have all the possible blotched detected.
Here are my result :

Original frame



Threshold at 0

Threshold at 35

Then, I decided not to treat blotches as individual pixel but as object. Moreover, I tried to reduce the noise in distance frames images to better detect blotches. To do so, I started by appling dilation and erosion operations to my images to bring together pixels. Then, with the function «bwconncomp » I computed connected components. For each components, I computed a probability to be noise. Indeed, for each components in the frame I can find the probability to be noise according to the distance value in the distance frame. The probability is given by a gaussian distribution with the mean as the mean of pixel distances for the blotch and a variance of 9 (estimation for the noise variance). Then, I computed the probability total for the blotch to be noise (as the mean of all the pixel probabilities). If this probability is above a certain value of risk, the component is deleted and treated as noise.
For the threshold to zero, I gave a risk of 1000 (high chance to be noise) and for the higher threshold a probability of 10^5.
Here are my result :

Original frame

Risk at 1000

Risk at 10^5

Then I used an threshold hysteresis algorithm with the function « bwselect » of matlab to obtain my last result for the mask :

Final mask

We can see only the blotch stays.

However, if I stay with the same values other blotches aren't detect but, if I decreased them, lots of non-blotches are detected. Indeed, all the movement of the car are often taken as blotches. I tried to remove moving connected components with an optical flow technique of matlab but blotches were also detected as movement so I let it that way.
To fill in the holes I use a mean of the previous and following frames.
For this example here is my result :

Original Image

Corrected image

However, it works really good for this frame because I could chose the values I wanted but those values doesn't fit other blotches, maybe smaller, less distinctive or « lost in the movement ». I couldn't put values which corrected all blotches because if I did, some moving objects in the frames were thought to be blotches as well which gives a weird result at the end.

Here is some other examples where big blotches were detected and corrected :

We can see that the car is also thought to be blotches so it gives artifacts and the image losses in quality around those areas.

Moreover, if the blotch appears in several frames in a row it is not detected by the algorithm which only detect blotches which are not in only 1 frame. Indeed, the way I computed the distances function only authorizes 1 frame blotches (if the blotch stays, the distance will be set to zero because the pixel has a correspondence in the next frame..)

General coursework information :

To prepare this coursework, I discussed some results with Hugo Germain from CGVI but the whole content of the submission is my own work. Moreover, I was inspired by « Restoration and Storage of Film and Video Archive Material » from IOS Press, 1999 given in the coursework files.