Computational Photography and Capture
Restoration of Old Film

Student ID:13028702 Student Name: Yifei Gao

Detection of Scene Cut

This solving this task involve two functions, detect_cuts.m and add_transition.m. The main idea here is to compare the sum of absolute difference between two frames and choose a threshold to determines it is a cut or not. As we assume the scene is continuous, where the difference between adjacent frames are alike, so when there is large jump in the sum of difference we can call it as a cut. Figure 1. shows the distribution of the sum absolute difference between frames in the given footages:

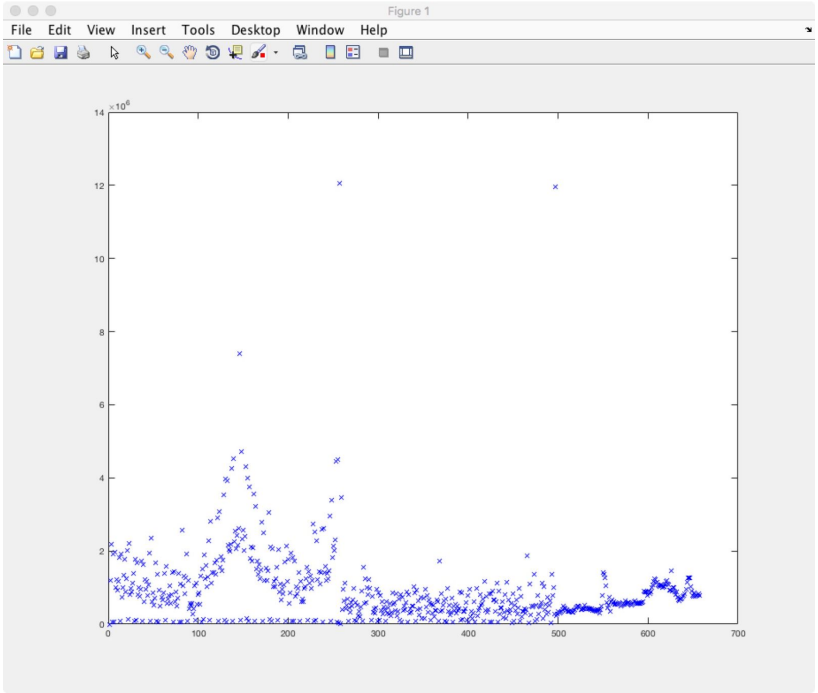

Figure 1. Distribution of the sum of difference

We can notice that there are 2 outliners with very high value of change that determines the index the index of the cut frame. A fixed threshold can be used to identify the cuts, but I choose to uses a percentage threshold instead, as a hard coded threshold might not works in other footage. This percentage threshold means how much of difference between frames is allowed, for example, threshold = 0.5 means that when the sum of difference is bigger than half of the sum of intensity of one frame, then there exist a cut.

This function then put the scene into different cells for further editing, in here we another function add_transition.m to add hints on the frames to notified viewer the next cut is coming. This function simply add the hint with given text and duration to the last few frame of each scene. Figure 2. is an example of transition frame with hint text:

Figure 2. Transition frame with hint text

Correction of Global Flicker

This task involved only 1 function, remove_flicker.m. The main idea of this algorithm is to use a sliding windows of given size to compute a average frame, and uses this average frame as a reference frame to perform histogram matching with current frame as the target. I've used a windows of 11 frames to compute the average frame, where it looks 5 frames forward and 5 frames backward to compute the average value of each pixel in the image, this process minimise the change of brightness in the image.

Then we compute the histogram of the reference frame and target frame, this can be done by allocated 255 bins for all possible intensity and count the number of pixel in the image that has the same intensity of the bins index. With the calculated histograms, we then cumulatively add all the bins up to compute a cumulative distribution of the histograms and normalise it for the Histogram matching step. There are many ways we can do the histogram matching, the implementation I've done is to use two pointer, one goes forward from 0 to 255 and another one goes backward from 255 to 0. We iterate the forward pointer by 1 each time to find the mapping of a bin and record its value in current frame's CDF, then we reset the backward pointer each time and iterate backward to find and store the index of the first bin that has larger value than

the bin we want to map in the average frame's CDF. Finally uses this new mapping to adjust the intensity of pixels in current frame, for example at Map(12) we have value of 14 means that we replace all pixel with intensity of 12 in current frame with 14.

Figure 3 shows the raw histogram, average histogram and the corrected histogram from top to bottom. Figure 4. Shows the raw frame and the corrected frame.

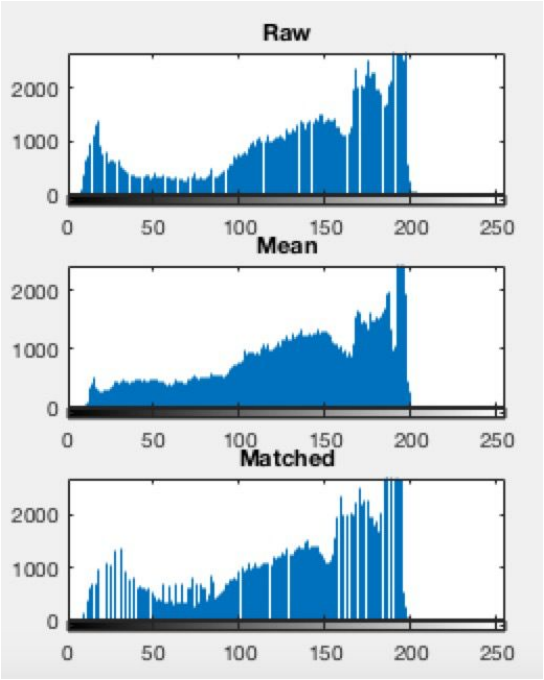

Figure 3. Resulting histograms. (From top to bottom: Raw, Average,corrected)

Figure 4. Histogram matching results.(Left: Raw frame, Right: Corrected frame)

This frame is one of the frames where the footage gets brighter, you can observe that the building in the corrected image is less bright, so this algorithm does adjust the histogram to matching the average frame with a given windows. But this method only works for global

change not local change, so it enhanced some local intensity changes in some small area, such as the sky in our example. This could be solved by applying an extra intensity filtering with edge detection, so when the area is smooth we average the change in intensity.

Correction of vertical artefacts

This part correspond to coorect_strip.m function in the code. The key idea of removing vertical strip here is to apply 1D median filtering to each row of the image, to minimise the large jump in the intensity. But as a result of median filtering, the image becomes blurry and some edges are lost, so we need to add the edges and details back to the result image.

My strategy is to apply median filtering repeatedly with kernel size from large to small, where in my example kernel size goes from 8 to 2 and skipping the odd kernel size, so it filter at kernel size 8,6,4 and 2. At each step compute and store a row-wise filtered image with given kernel size, then compute the absolute difference with between filtered image at current and last level (For the first level, we uses the raw frame as the last level). We can then uses this absolue difference to compute a mask with a given threshold, this allow us to create a mask of the vertical strips we detected(or in other words, the pixels that's been over-filtered by large kernel size). We use this mask to replace the pixel in the raw frame with the filtered pixel, so we only smoothing the area that's been identified as vertical artefacts. By repeating this procedure at different level, we are able to add the details back progressively, as the smaller kernel size median filtering trends to keep more feature of the edges. Figure 5. Shows how the resulting edge map changes by subtracting filtered image at different level.

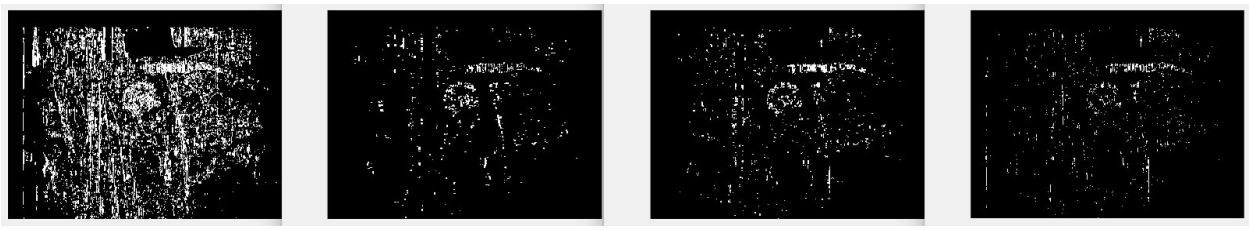

Figure 5. Edge map of filtered image between different levels.
(Form left to right: Raw-Lv8, Lv8-Lv6, Lv6-Lv4, Lv4-Lv2)

You can observe that the differences trends to get thinner as the level gets lower which is what we expected. To compared the result we show the raw frame, repeatedly filtered frame and our resulting frame in Figure 6:

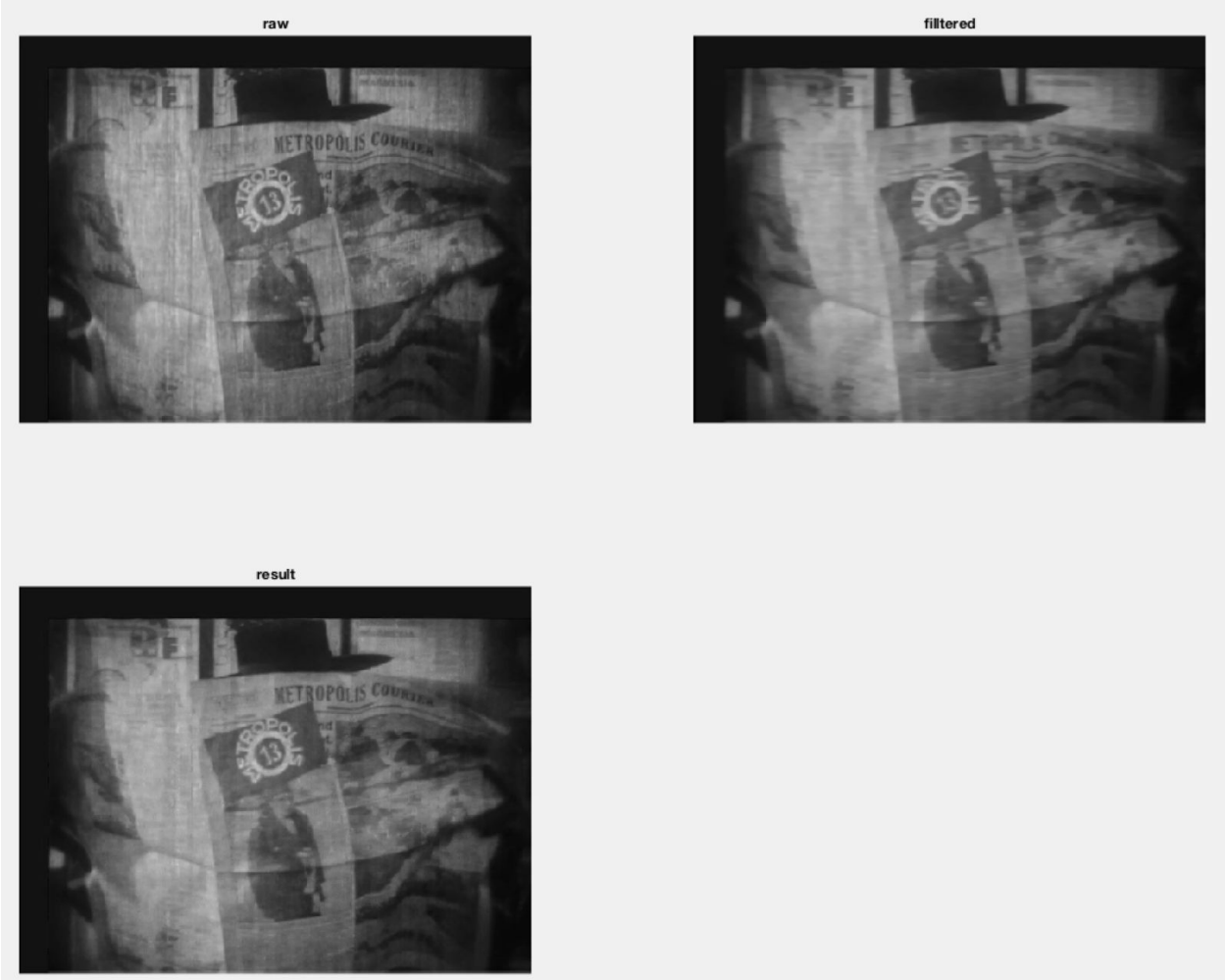

Figure 6. Result of vertical artefacts correction.
(Top Left: Raw frame; Top Right: Filtered frame
Bottom Left: Result frame of my algorithm)

You can observe that the filtered frame contains the least vertical strips, but the result image is very blur compare to our solution. The edge of the hat and the printed text on the newspaper is preserved in our solution, although there are still some remaining vertical stripes in our solution, but compare to the raw frame the effect of vertical strip removal is very good. But as we mentioned, there is always a tradeoff between smoother image and artefacts.

Correction of Camera Shake

This part correspond to remove_camera_shake.m function in the code. My solution is very simple, we need to identify the features in the features in each frame. This part corresponding to the compute_edge_map sub function in our code, we calculate the approximations of the horizontal and vertical derivatives (G_x and G_y)of the image by convoluting the vertical and

horizontal sobel operator to the images. Then combined G_y and G_x to give a gradient magnitude, where G = sqrt(G_x^2+G_y^2), we uses a threshold to kept the large magnitudes as our mask. Figure 7. Shows the gradient magnitude Mask with different thresholds.

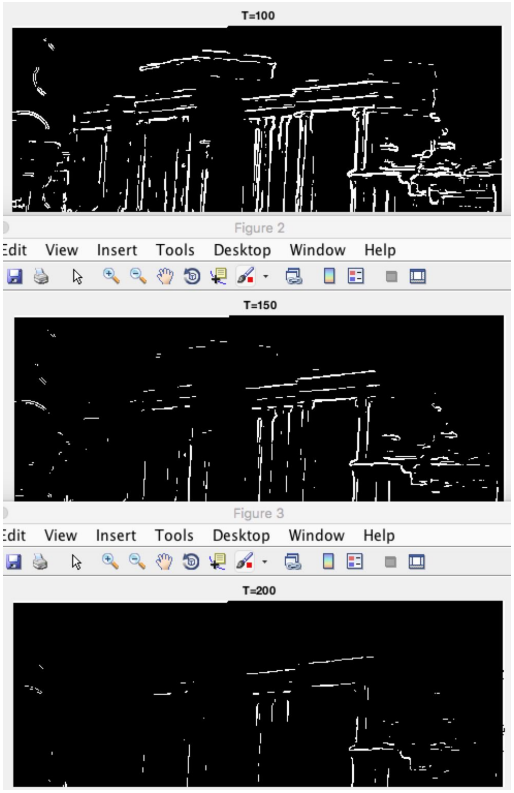

Figure.7 Mask with different Threshold.
(From top to bottom: Threshold = 100, Threshold =150, Threshold 200)

Now we have the mask of each frame, the problem left to solve is how to match the response between the mask so they align to each other. My strategy of this problem is to only kept the static part of the current frame and last frame, (the background buildings) and compute their cross correlation to find the translation that most resemble each other. There are many ways to identify the moving and static object in the sense (overkill for a simple solution), but from the footage we're provided, majority of the static objects appears on the top half the frame, hence we only take the top half of the image. This avoids the error due to moving objects and speed up the cross correlation calculation, as the size of the image is halved.

But this is not always true, as in the first cut of the footage, the moving vehicle covers most of the frames, the static background we used as guidance becomes less visible, the cross correlation end up matching the moving object features. This problem makes the peak response of the offset larger than they should be, a simple solution to fix this is to abandon the large jumps in the offsets values. Therefore I've used a max_dY and max_dX to specified the largest offset it could take, any offset larger then this will be see as tracking moving object and get reset to 0.

With this approximated offset we can translate the current frame to match last frame, but this creates some empty space in the image, hence we need to wrap the image to fill these empty space created by translation. My approach is to label these empty pixels with -1 and replaced them with the average pixel value of current and last frame in the same location. The performance of my solution works very well in the second cut, there are some artefacts in the first footage due to offset resetting to avoid tracking moving objects. This produce some extra shakes to the footage when the vehicles covers the frame, but perform well in the rest of the sequence. Figure 8. Shows the result of the camera shake correction algorithm.

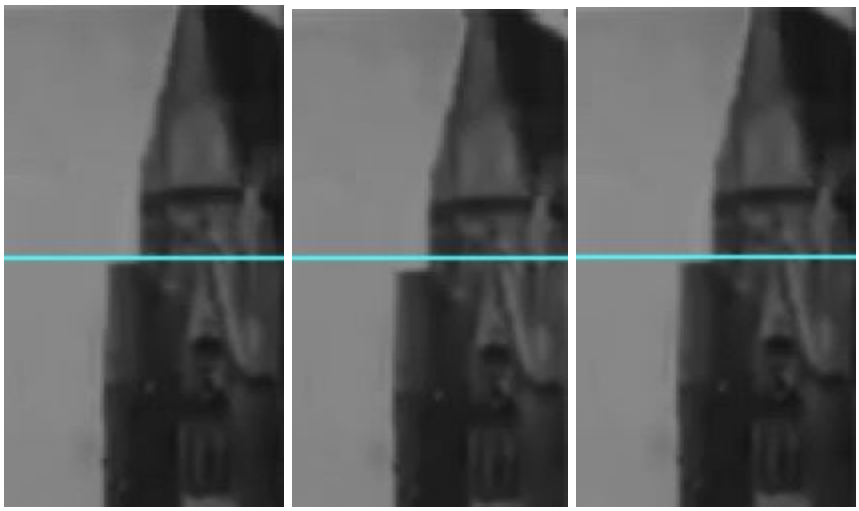

Figure 8. Result of camera shake correction.
(From left to right: Last frame; Raw Current frame; Corrected frame)

The limitation of this method is that it only assume the camera shake is only translation and the majority of the frame is statable, so it can not due with large change in the background and rotation of the camera.

Correction of Blotches

This part of the algorithm corresponding to remove_blothch.m function in the code. My solution of this problem is to calculate the difference of gradient magnitude of current frame with its adjacent frames to obtain a mask of changed feature in current frame. An alternative way to do this quick approximation of high intensity change is to measure patch similarity of within the same window of adjacent frame, this will identify the blotches better than just comparing the gradient magnitude changes. But in my example, I've only used the first approach.

This provides us a starting point for the next step, based on these estimated starting points, we uses its position and intensity to run the region growing algorithm to select the area with similar intensity, as we assume the blotches will have uniform colour. But it can also end up growing a

large smooth area that is not a blotch, to stop this from happening, we defined a growing_theshold that controls the range of intensity difference we accept in growing condition, and a maximum population that terminates the growing. The idea here is that if the area grows too large, then it's very unlikely to be a blotch, hence we do not include this area in our blotch mask. Figure 9. Shows the example of the starting position we used for region growing, and the resulting mask of region growing. You can observe that the bottom left starting points get discarded in the resulting mask as their region grows too large.

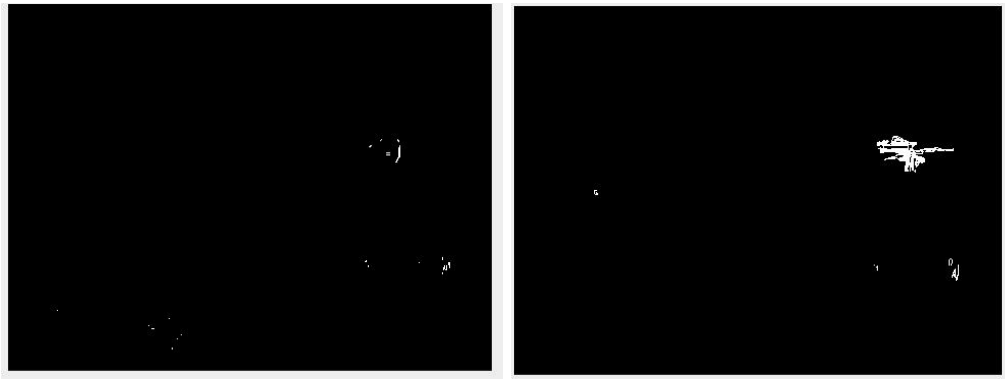

FIgure 9. Result of Region growing.
(Left: staring seeds of region growing; Right: Resulting mask of region growing)

At this point we have mask of blotches we've identified, the problem left to solve is how to replace these pixels in the mask so it looks alike to surrounding pixels. My implementation here is to uses the average of last three corrected frame to fill the blotch, some ghosting effect might appear due to the shake of the camera, so the order of applying different algorithm will change the quality of the blotch correction. There are also other factors that affects the performance of the algorithm, for example in the histogram matching process we might have map multiple intensity to one intensity, this will make the region growing selects larger area. Figure 10. Shows the result of this algorithm.

Figure 10. Result of blotch correction.(Left: Raw frame; RIght: Corrected frame)

In terms of performance, this algorithm is capable of identifying most of the large blotches in the footage, but not very good at identifying the small or thin blotches. The time performance is related to the number of starting position we defined, so in order to speed it up, we can apply some sampling to the starting positions. But there is a trade of between the speed and risk of missing some possible blotches.