Coursework 1: Restoration of Old Film

Due date: 8th Feb. 2018

1. Introduction

This report will give a description of algorithms implemented for old film restoration. There are 5 tasks included, which are scene cuts detection, global flicker correction, blotch correction, vertical artefacts correction and camera shake correction. Their corresponded algorithm explanation, results as well as limitations will be analyzed.

2. Tasks

2.1. Detection of scene cuts

Scene cuts represents the moment when scene changes. We aim to detect hard transitions that occur when scene switches to another.

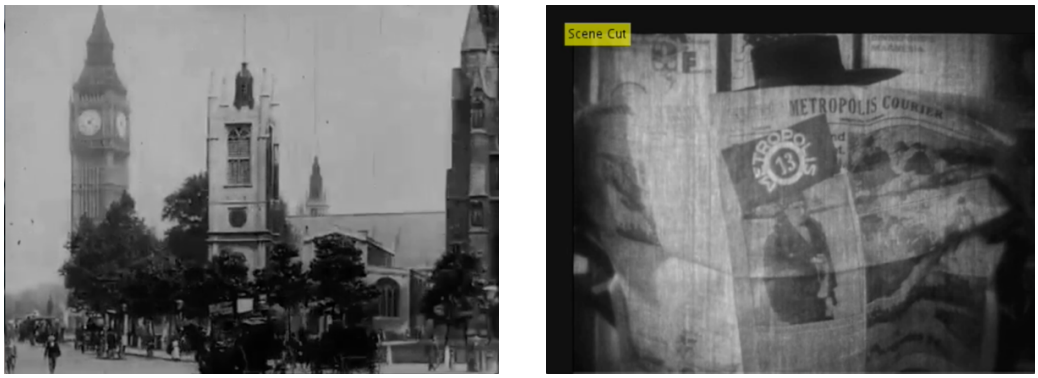
In this task, template matching was used in hard transitions detection within given image sequence. The specific function could be seen as Figure 1.

```
%% function for scene cuts detection
% Input:
% mov: image sequence
% v: output video
% frame_num: image number included in image sequence
function [cut_index] = sceneCuts_detection(mov, v, frame_num)

    cut_index = [];

    % write the first frame into the video
    writeVideo(v, mov(:,:,1));

    for n = 2 : frame_num
        % get the pixel sum of current frame and next frame
        sum1 = sum(sum(mov(:,:,n-1)));
        sum2 = sum(sum(mov(:,:,n)));

        % get normalize frame difference
        frame_diff = sum(sum(abs(mov(:,:,n) - mov(:,:,n-1))));
        frame_diff = frame_diff / ( (sum1+sum2)/2 );

        % set a threshold, if difference between frames is bigger than
        % threshold, scene cut occurs.
        current_frame = mov(:,:,n);
        if (frame_diff > 0.70)
            cut_index = [cut_index, n];
            current_frame = insertText(current_frame, [20, 20], 'Scene Cut');
        end

        % write the current frame into the video
        writeVideo(v, current_frame);
    end
end
```

Figure 1: Function for scene cuts detection

The algorithm calculates the total pixel sum of two consecutive frames firstly and then get the difference by subtracting the previous frame from the current one and normalize the result. A threshold is set according to multiple measurements (in this case the threshold is set as 0.70). If the normalized difference is larger than the threshold, it means that a scene cut occur between the current and previous frame and text said 'Scene Cuts' will be displayed on the left top corner of the current frame; Otherwise, no scene cut occurs between two frames. There are two scene cuts detected in given image sequence, which occur between 256th and 257th frames and 496th and 497th frames (Figure 2). The output video could be seen as video called 'Task1.avi'.

(a) 256th and 257th frames



(b) 496th and 497th frames

Figure 2: Detected Scene cuts

However, this algorithm may produce wrong detection when any objects have obvious movements within frames. Template matching method is very sensitive to camera movement and object motion within frames because its calculation is based on this point-to-point matching.

2.2. Correction of global flicker

Intensity flicker occurs as a kind of unnatural temporal fluctuation in image intensity, which won't have a constant global luminosity over time.

In this task, I implemented an algorithm which was able to make use of average grayscale value of neighborhood frames to adjust intensity histogram of current frame. The number of frames within neighborhood is predefined according to multiple tests and comparison. Meanwhile, the neighborhood frames should be in the same scene to avoid the wrong correction due to scene cuts. The function to predefine frame indexes within neighborhood could be seen as Figure 3. Initialize the first and last frame index of reference neighborhood frames firstly. Next, adjust index of neighborhood frames according to given frame scope to produce the correct reference frame index.

```
%% function to find neighborhood index range
% Input:
% current_f: current frame
% start_f: start frame of one shot
% end_f: end frame of one shot
% neighbor_size: neighborhood size
function [start_index, end_index]=neighbor_index(current_f, start_f, end_f, neighbor_size)

    % get initial start and end index of neighborhood
    neighbor_r = round(neighbor_size/2);
    start_index = current_f - neighbor_r;
    end_index = current_f + neighbor_r;

    % if initial start index is out of range, move backward the neighborhood
    if start_index < start_f
        start_index = start_f;
        end_index = start_index + neighbor_size;

    % if initial end index is out of range, move forward the neighborhood
    elseif end_index > end_f
        end_index = end_f;
        start_index = end_index - neighbor_size;
    end
end
```

Figure 3: Function to get frame index within neighborhood

The function for global flicker correction is shown as Figure 4. The algorithm obtains the index of reference neighborhood frames by applying function named *neighbor_index(),* and then calculate the average grayscale frame of reference frames. Function named imhistmatch(), which has the ability to adjust histogram of a 2D image based on reference histogram, is applied to adjust intensity of current frame based on calculated average frame.

```
%% function for global flicker correction
% Input
% input: input image sequence
% start_f: start frame within one shot
% end_f: end frame within one shot
% neighbor_size: previous or later frames number within neighborhood
function [output] = flicker_remove(input, start_f, end_f, neighbor_size)

    % initialize output image sequence
    output = input;

    for i = start_f : end_f
        % get neighborhood index range
        [start_index, end_index] = neighbor_index(i, start_f, end_f, neighbor_size);

        % get the pixel sum of neighborhood frames
        neighbor_sum = sum(input(:,:,start_index:end_index),3);

        % get the average of neighborhood frames
        neighbor_avg = neighbor_sum / (neighbor_size + 1);

        % adjuest current frame histogram according to average histogram
        output(:,:,i) = imhistmatch(input(:,:,i), neighbor_avg);
    end
end
```

Figure 4: Function for global flicker correction

The effect could be seen in some of frames and one of examples is shown below. The example illustrates the frames with index from 46th to 49th. It could be found that the intensity of two center frames is brighter slightly than other two frames on the sides, while output frames shows that the luminance of these frames has been adjusted to become more similar to neighborhood.

(a) Original frames

(b) Output frames after flicker correction

Figure 5: 46th-49th frames

Nevertheless, this algorithm works ineffectively under some situations due to neighborhood size. For instance, if the number of frames with flicker is larger than the size of reference neighborhood, the output frame won't be satisfied due to too

bright or dark average histogram. With the increase size of neighborhood frames, more object motions will be taken account into calculation and it is possible to make the average histogram become too unstable to be reference. Therefore, the selection of neighborhood should be considered based on specific image sequence.

2.3. Correction of blotches

Blotches is one of common types of artefacts, which appear as bright or dark spot within frame. The reason this happens is dirt and the limited amount of gelatin covering the film. One of characteristics of blotches is that they merely appear at the same spatial location within in consecutive frames. Additionally, the intensity values of blotches are quite different from that of original content that they appear. According to these characteristics, the procedure of blotches correction could be summarized as two major steps. Firstly, detecting blotches and generating blotch mask. Next, correcting covered pixels through interpolation based on previous frames' histogram. The last shot has less blotches, thus this algorithm was applied on first two shots only.

```
%% function to create motion mask
function [output_mask, output_diff] = motion_mask(mov, input_mask, input_diff, start_f, end_f, neighbor_size, threshold, avg_k, extend)
    % initialize difference array
    output_diff = input_diff;
    % initialize mask
    output_mask1 = input_mask;

    % compute difference between two consecutive images
    for f1 = start_f : end_f - 1
        output_diff(:,:,f1) = abs(mov(:,:,f1) - mov(:,:,f1+1));
    end
    % compute mask
    for f2 = start_f : end_f
        [start_index, end_index] = neighbor_index(f2, start_f, end_f, neighbor_size);
        output_mask1(:,:,f2) = sum(output_diff(:,:,start_index:end_index-1),3);
    end

    % smooth mask edges and eliminate any unnecessary parts
      avg_fil1 = fspecial('average', avg_k);
      output_mask2 = imfilter(output_mask1, avg_fil1);
      output_mask2(output_mask2<threshold)=0;
      output_mask2(output_mask2>threshold)=1;
      se1 = strel('disk', extend, 4);
      output_mask2 = imdilate(output_mask2, se1);

    if start_f == 1
         avg_fil2 = fspecial('average', 40);
         mask_tem = imfilter(output_mask1, avg_fil2);
         mask_tem(mask_tem<0.92)=0;
         mask_tem(mask_tem>=0.92)=1;
         se2 = strel('disk',35,4);
         mask_tem = imdilate(mask_tem, se2);
         output_mask2(:,:,24:45) = mask_tem(:,:,24:45);
    end
    output_mask = output_mask2;
end
```

Figure 6: function for generating mask for moving objects

Because of characteristics of blotches that has been introduced above, blotch mask could be obtained from the difference between consecutive frames. However, the difference image not only shows the position of blotches, but also records the outline of objects with movements. Therefore, a mask for moving objects should be generated to eliminate wrong blotch detection caused by objects with movement.

The function for generating mask for moving objects could be seen as Figure 6. In this function, absolute difference between two consecutive frames is computed firstly. Based on predefined neighborhood size, mask is generated by summing difference images within neighborhood. Average filter is applied on mask to smooth mask edge and removing small unnecessary region with high difference values. Based on given threshold, mask is convert into binary version. Note that the threshold value varies according to specified content of scene. If movements of objects occupy the major part of the scene, the threshold should be larger; otherwise, the threshold should be smaller. Finally, function *imdilate()* is called to extend mask region to make sure each part of moving object is covered. An example of mask of moving objects of 34th frame could be seen as Figure 7.

Figure 7: 34th frame (left) and its mask for moving objects (right)

The function for blotches correction is called *blotch_correction()*. It computes a mask to extract outline of detected blotches based on input difference images and masks sequence for moving objects. The blotch mask of 34th frame is shown as Figure 8. The corrected frame is calculated by put opposite masks on previous adjusted frames and current frame that to be corrected. The detailed code for the function is shown as Figure 9.

Figure 8: 34th frame (left) and its blotch mask (right)

```
%% function for blotch correction
% Input
% mov: input image sequence
% input_mask: input motion mask
% input_diff: input difference array
% start_f: start frame of one shot
% end_f: end frame of one shot
function [output] = blotch_correction(mov, input_mask, input_diff, start_f, end_f, threshold)
    % initialize output image sequence
    output = mov;
    blotch = input_diff;

    % set a threshold to evaluate input difference array, it larger than
    % threshold, set it as blotch
    blotch(blotch>threshold)=1;

    % remove blotches within motion mask
    blotch = blotch .* (1 - input_mask);
    se = strel('disk', 3, 4);
    blotch = imdilate(blotch, se);

    for f = start_f+2 : end_f
            pre_frame = imhistmatch((output(:,:,f-2)+output(:,:,f-1))/2,output(:,:,f-1));
            output(:,:,f)=  mov(:,:,f) .* (1-blotch(:,:,f-1)) + pre_frame .* blotch(:,:,f-1);
    end

end
```

Figure 9: function for blotches correction

The correction effect could be seen in some of frames and Figure 10 illustrates one of the examples which from 33th to 35th frame sequence. It could be found that there is an obvious dark spot appear on the right part of the original image, which shows gray in 33th and 35th frames and black color in 34th frame. After being applied blotches function, almost the whole spot has been removed without losing major content of scene.

(a) Frames before blotches correction

(b) Frames after blotches correction

Figure 10: 33th-35th frames

However, this algorithm has some limitations because of its working method, which has quite high dependence on major content of frames. For example, a moving object with uniform color and large volume is hard to create mask as it has low difference between consecutive frames so is easy to be treated as 'background'. What is more, some blotches, which cover on moving objects, are not corrected because they are seen as part of moving objects in mask generation. Therefore, this algorithm still need to be improved further.

2.4. Correction of vertical artefacts

This task aims to correction vertical artefacts that appear in last image sequence in given footage. The main method that algorithm used to remove vertical artefacts is making use of median filter to smooth pixels and then apply Laplacian filter to enhance edges of smoothed frames. The detailed code of function for removing vertical artefacts is shown in Figure 11.

```
%% function for vertical artefacts correction
% Input:
% mov: input image sequence
% edge: edge information
% start_f: start frame of one shot
% end_f: end frame of one shot
function [output] = vertical_artefacts(mov, edge, start_f, end_f)
    [row, ~, ~] = size(mov);
    output = mov;

    % median filter
    for f = start_f:end_f
        for r=1:row
            for k=1:5
                output(r,:,f)=medfilt1(output(r,:,f),6-k);
            end
        end
    end

    % extract outline edge
    la_fil = fspecial('laplacian',0);
    edge(:,:,start_f:end_f) = imfilter(output(:,:,start_f:end_f), la_fil, 'replicate');

    % shapen edges of smoothed frames
    output= output-edge;
    output(output<0)=0;
    output(output>1)=1;
end
```

Figure 11: Function for removing vertical artefacts

This function applies 1D median filter for five times on pixels that has been smoothed already. Pixels are smoothed in row unit, because vertical artefacts appear in vertical direction and the smooth filter won't works if it is applied on pixels in vertical direction. The kernel size is decreased as the bigger size the blurrier image. In order to keep the clear edges of objects without blurring the major content, Laplacian filter is used to enhance the objects outline. The effect of the filter could

be seen as Figure 12, which illustrates the change of grayscale values of pixels in one row. It could be found that curves in original frame has been smoothed significantly and many high frequency elements are removed, while the major structure of curves has been remained.

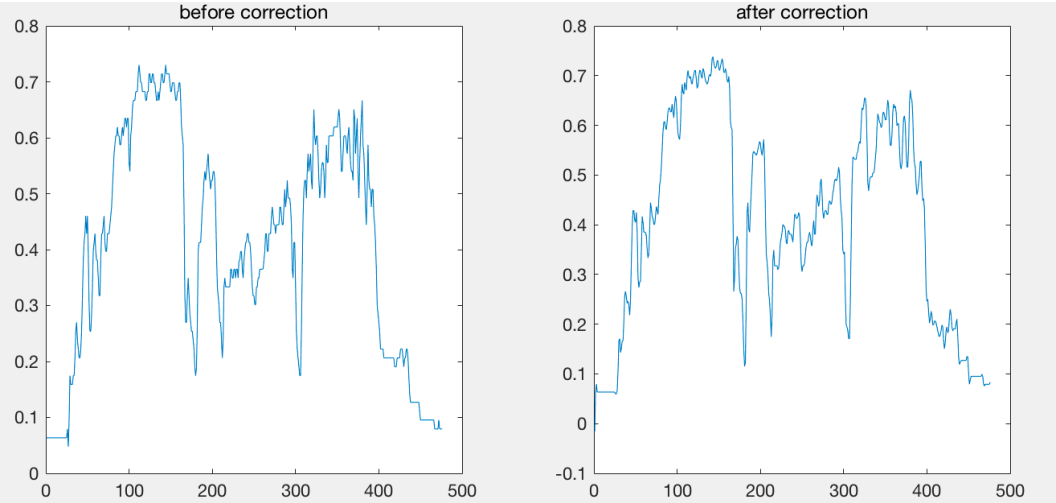


Figure 12: Original histogram (left) and filtered histogram (right) for pixels in one row

One example after vertical artifacts correction could be seen as Figure 13, which illustrates the effect of the implemented algorithm. It could be learned that most of vertical artefacts have been removed without blurring major content of the frame.

Figure 13: Original 652th frame (left) and filtered frame (right)

2.5. Correction of camera shake

The purpose of this task is to remove unwanted frame motion by aligning the frames together within one shot. The main idea of my algorithm could be summarized as align frames based on extracted feature points of average frame by using template

matching method.

First of all, in order to obtain stable features of the whole single shot, the average frame of shot is calculated and compared with the first frame of the shot. The common feature point pairs are extracted and they are used as reference coordinate for frames within shot.

```
output = mov;

% get first and average frame of single shot
avg_f = sum(mov(:,:,start_f:end_f),3)/(end_f-start_f+1);
first_f = mov(:,:,start_f);

% detect feature points of first and average frame
featureP_avg = detectSURFFeatures(avg_f);
featureP_first = detectSURFFeatures(first_f);

% obtain feature descriptors
[feature_avg, featureP_avg] = extractFeatures(avg_f, featureP_avg);
[feature_first, ~] = extractFeatures(first_f, featureP_first);

% match features of two frames
indexPairs = matchFeatures(feature_avg, feature_first);  % indexPairs is a Px2 matrix [feature_avg, feature_first]
matchP_avg = featureP_avg(indexPairs(:,1),:);    % match points in average frame
```

Figure 14: Code for features match between average frame and the first frame of the shot

The feature matching results of the second shot could be seen as Figure 15. It shows the common feature pairs of average frame and first frame, which are to be used as reference standard in the further implementation.

Figure 15: Matching feature pairs of average frame (green) and first frame (red)

This algorithm uses patch matching within search window to find the certain offset for current frames. To obtain the position of four sides of both patch and search window, the function *window_coord()* is programmed based on the center pixel location and given window (or patch) size. The function is shown in Figure 16.

```
%% function to get the window coordinate
% Input:
% center_r: row of center pixel
% center_c: column of center pixel
% r: radius of square window
% row_num: total number of rows in frame
% col_num: total number of columns in frame
function [top_r, bottom_r, left_c, right_c] = window_coord(center_r, center_c, r, row_num, col_num)
    % initialize coordinate
    top_r = center_r - r;
    bottom_r = center_r + r;
    left_c = center_c - r;
    right_c = center_c + r;

    % check boundaries
    if top_r < 1
        top_r = 1;
        bottom_r = top_r + 2 * r;
    elseif bottom_r > row_num
        bottom_r = row_num;
        top_r = bottom_r - 2 * r;
    end

    if left_c < 1
        left_c = 1;
        right_c = left_c + 2 * r;
    elseif right_c > col_num
        right_c = col_num;
        left_c = right_c - 2 * r;
    end
end
```

Figure 16: Function window_coord()

Next, iterate each frame within the shot and compare each feature position of current frame with that of average frame. Get reference patch from average frame and set corresponded search window in the current frame. Search the similar patch within search window and compute correlation coefficients between similar patch and reference patch. The maximum correlation coefficient among all search windows is extracted for further offset calculation. Appeared gaps, which caused due to frame shift, is filled with black color simply. The detailed code is shown as Figure 17.

```
% number of patch
patch_num = matchP_avg.Count;
% position of match points
matchP_pos = round(matchP_avg.Location);

for f = start_f : end_f
    current_f = mov(:,:,f);
    cor_max = 0;

    for p = 1:patch_num
        % get patch
        [top_r_pat, bottom_r_pat, left_c_pat, right_c_pat]=window_coord(matchP_pos(p,2), matchP_pos(p,1), patch_r, row_num, col_num);
        patch = avg_f(top_r_pat:bottom_r_pat,left_c_pat:right_c_pat);
        % get search window
        [top_r_win, bottom_r_win, left_c_win, right_c_win]=window_coord(matchP_pos(p,2), matchP_pos(p,1), window_r, row_num, col_num);
        search_window = current_f(top_r_win:bottom_r_win, left_c_win:right_c_win);
        % correlation calculation of patch within search window
        c = normxcorr2(patch, search_window);
        c = c(2:size(c,1)-1, 2:size(c,2)-1);

        % get maximum correlation coefficient
        if max(c(:))>cor_max
            cor_max = max(c(:));
            [max_match_r,max_match_c]=find(c==cor_max);
        end
    end

    % shift amount calculation
    shift_r = window_r + patch_r - max_match_r(1) + 1;
    shift_c = window_r + patch_r - max_match_c(1) + 1;

    % eliminate unreliable shift amount
    if abs(shift_r)<8 && abs(shift_c)<8
        output(:,:,f)=imtranslate(mov(:,:,f), [shift_c, shift_r]);
    end
end
```

Figure 17: Code for patch matching

The specific effect of this algorithm could be illustrated by checking the average frame of one single shot. Figure 18 shows the change of average frame of given second shot after removing camera shake. It could be observed that the average frame after shift shows clearer outline and edges than the average frame before the shift, which means that frames after shifts realize better alignment.

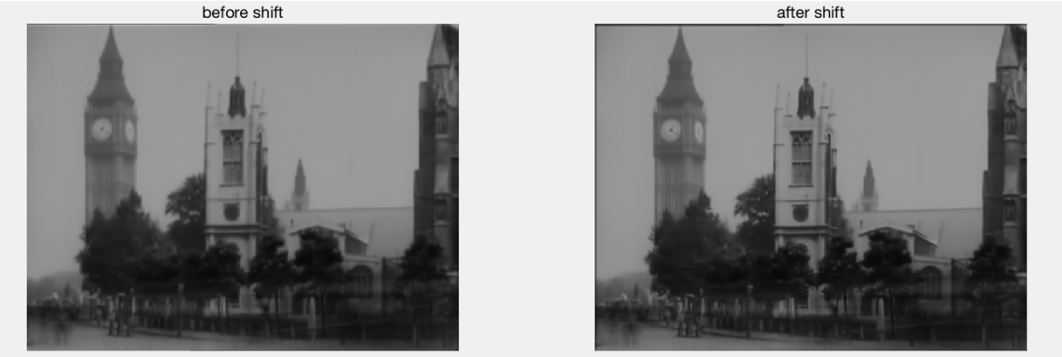


Figure 18: Comparison between average frame before (left) and after shift (right)

However, it is hard for template matching algorithm to make sense when the scene within shot is not stationary. The method works depends on the features consistency of scene. If camera moves along any certain track, such as tracking a running athlete, this approach will not work successfully as it cannot find common feature pairs.

3. Conclusion

In this coursework, I have tried to implement algorithms in five tasks to restore give old films, including scene cuts detection, correction of flicker and blotch and removing vertical artefacts and camera shake. Due to the use of amount of recursive approaches, the computational cost is quite large. This limitation could be developed future in the future improvement by using more smart algorithms.