Documentation to COMP0028 (Course Work 1)

1 Cut Scene Detection

Scene cuts occurs when the editor put several footages together, the joint between two different footages is where the cut is. The task of this section is to develop an algorithm that detects those cuts then put some overlay text to inform the viewer that a cut happens.

1.1 General Approach

One characteristic of a cut is the difference of intensity between the two frame are comparatively high. Therefore, all we need to do for detection is to find the frame that has high intensity difference with its next frame. In order to achieve this, I use the image histogram as a feature for valuing intensity of a image.

Below shows the chunck of code for the cut detection approach :

```
%% Scene cut detection
  for i=2:frame_num
      currentImg = v_int(:,:,i);
      [counts,binLocations] = imhist(currentImg);
      count(i,:) = counts;
      sum_history(i) = sum(abs(count(i,:) - count(i-1,:))); % ...
          Comparing the histogram differences
  end
  for i=2:frame_num
      if sum_history(i) > Threshold
         counter = counter +1;
         flag_changeScene(counter) = i;
      end
  end
```

The code first computes the all histogram differences between each pair of frames. Then, a thresholding is applied to find the high difference value.

1.2 Discussion and solution of potential problems of algorithm

The theory of the algorithm relies on high intensity difference between frames. In practical, however, there are many other factors will result in a high intensity difference as well, one particular example is the flickers:

Figure 1: Example of flickers

As the Figure 1 shows, we can clearly notice a huge intensity shifts between two frames. In practical, the above algorithm will miss classifying those flickers as cuts. To address this problem, we can do two things: deflickering or changing the threshold adaptively. Deflickering will be introduced in the next section, so we need to modify the code to get better threshold for now.

```
1    max(sum_history)*0.97
```

One good way to do this, is to pick top 3% highest difference to be the cut instead of giving a fix value. By doing this, the mis-classifying cut due to flickers is avoided. Below shows two consecutive frames and the scene cut is detected and labelled.

2 Correction of global flicker

As introduecd in previous section, intensity flicker is an unwanted temporal fluctuation in image intensity. The task of this section is to remove or reduce the effect of those flickers.

Figure 2: Scene cuts detection result

2.1 Initial General approach

To correct flicker, first we need to establish a model for the flickers as the [1] proposed:

$$Y(x,y,t) = \alpha(t)(I(x,y,t) + \gamma(x,y,t)) + \beta(t) + \eta(x,y,t) \tag{1}$$

Where Y is the observed image intensity and I is the real image intensity. γ and η are noises that is image-depented and image-independent respectively.

According to the equation(1), to restore the image, we need to successfully find the value for α and β:

$$\beta(x,y,t) = E(Y(x,y,t)) - \alpha(t)EI(x,y))$$

$$\alpha(x,y,t) = \sqrt{\frac{var(Y(x,y,t))}{var(I(x,y) + \gamma(x,y,t) + \eta(x,y,t))}}$$

The variance and the expectation is computed as follows:

$$E((I(x,y)_t)) = \kappa E((I(x,y)_{t-2})) + (1-\kappa)\frac{E(Y(x,y,t-1)) - \beta(t-1)}{\alpha(t-1)}$$

$$var((I(x,y))) = \kappa var(I(x,y)) + (1-\kappa)\frac{var(Y(x,y,t-1))}{\alpha(t-1)^2}$$

Where κ represents the importance of previous image. Introducing this term would reduce the memory load as it does not require summing up all previous sequences for computing variance and expection. The noise term γ,η is omitted because noise is not considered while completing this task.

We do this for every frame except the first two and get their corresponding flicker paramerters α and β and then perform the restoration.

```
%% Flicker correction
for t = 3:frame_num
  cur_frame = v(:,:,t);
  frame_last2 = restoration(:,:,t-2);
  frame_last1 = v(:,:,t-1);

  E_t = kappa * mean2(frame_last2) + (1-kappa) * (mean2(v(:,:,t-1)) - ...
      betas(t-1))/(alphas(t-1));
  var_t = kappa * var(frame_last2(:)) + (1-kappa) * ...
      (var(frame_last1(:)))/(alphas(t-1)^2);

  alpha_t = sqrt((var(frame_last1(:)) )/(var_t));
  alpha_t(isnan(alpha_t)) = 1;
  beta_t = mean2(cur_frame) - alpha_t * E_t;

  alphas(t) = alpha_t;
  betas(t) = beta_t;
  restoration(:,:,t) = (v(:,:,t) - beta_t)/alpha_t;
end
```

2.2 Algorithm evaluation

By running the above code, the flicker indeed reduced: For better illustration of flickers

Figure 3: Example of flickers correction

correction result, I blend the un-corrected and corrected image together. The middle part

are the corrected image, so we can see the flicker indeed get flickerd.

The problem of the algorithm is relying on the previous two frames, which is the reference frame, the algorithm would try to change the current frame's instensity to match the previous two's, thus, if the referece frame already suffers the flicker, the algorithm does not work. Therefore, the referece frame must be choosen manually in that case.

3 Correction of vertical artefacts

The footages, especially the last one has many vertical stripes, the task for this section is to remove them.

3.1 General Approach

The simpliest way to remove such artefacts is by applying the Gaussian filter, however, this cause strong blur to the original image thus not desired. Another suggested way is to use the median filter.

The pixels over a row of image can be visualized as waves: The spikes in this figure

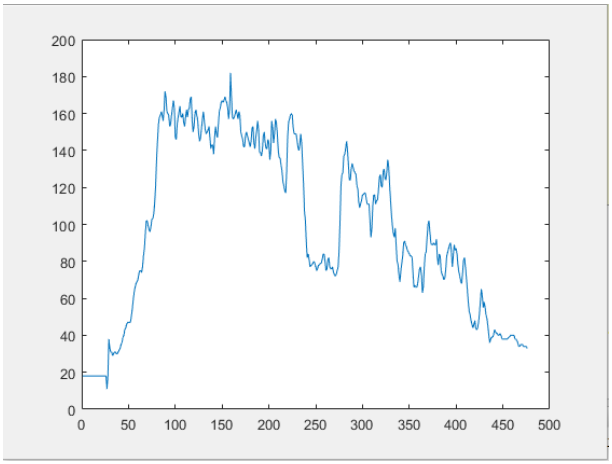

Figure 4: Single row visualisation of frame: 497 row: 120

is defined to be the stripes. Here is the wave after median filtering(orange): The sharp spikes are smoothed while the details(trace) of the image is preserved.

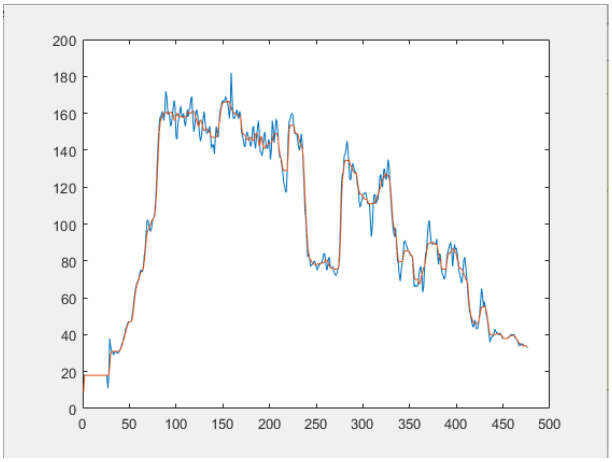

Figure 5: Median filtered

3.2 Algorithm evaluation

The median filter indeed remove the stripes, but the image also loss the shapness as some of the details still being filtered out. So we need to add the lost details back, one way to

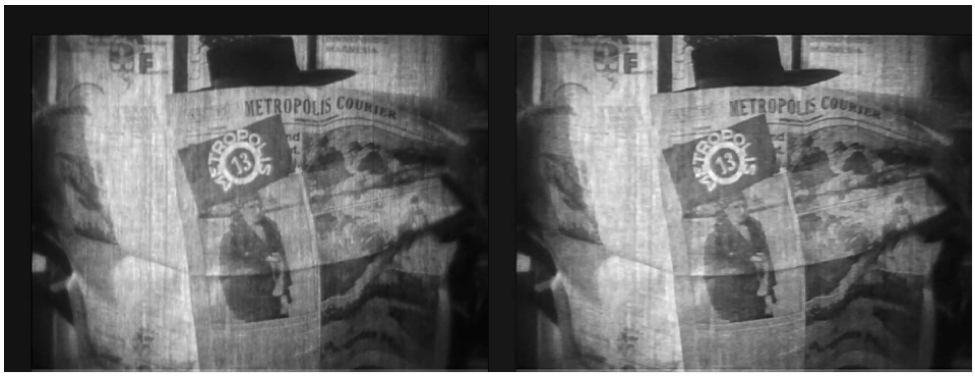

Figure 6: General approach result

achieve this is to compute the difference between before and after, that is the lost detail as well as the strips. Next, I divide this by the reciprocal of median filter size.

```
1    for t = 497:657
2    for i = 1:height
3        % The approach is to scanning through each line and applying ...
             the median
```

```
4         % filter.
5         output(i,:,t)   = medfilt1(double(output(i,:,t)),4);
6         output(i,:,t) = output(i,:,t) + ((double(v(i,:,t)) - ...
              output(i,:,t))./4);
7     end
8 end
```

As can be seen from the below graph, we restored some of image details and keeps the strips at minimum.

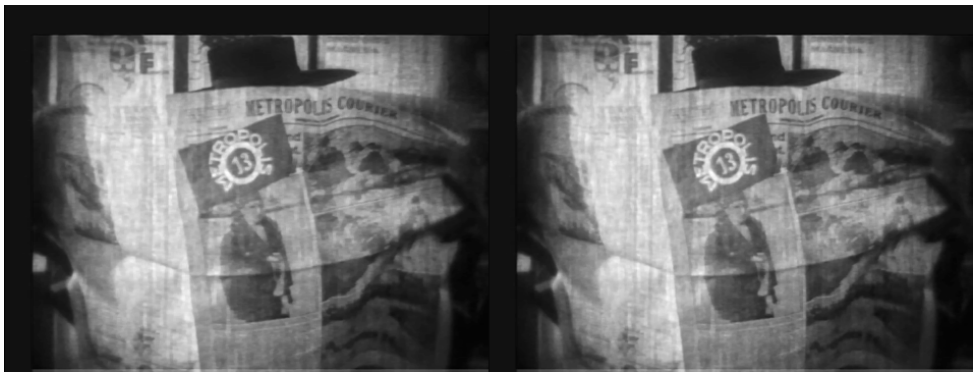

Figure 7: left:pure median filter, right:add details

4 Correction of blotches

The blotches are some disturbing white or dark spots on the film, and they are usually inevitable for long-term film storage. The task for this section is to remove those spots.

4.1 Initial General Approach

The idea of blotch removal is first detect them, them do some correction. The correction is simply clone some pixels from the previous frame.

In terms of the blotch detection, I use the S-ROD[2]:

$$d_n(z) = \begin{cases} min(p_{n,i}(z)) - I_n(z) & \text{if } min(p_{n,i}(z)) - I_n(z) > 0 \\ I_n(z) - max(p_{n,i}(z)) & \text{if } I_n(z) - max(p_{n,i}(z)) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Where $I_n(z)$ represents pixel at spatial location $z = (x, y)$ at frame n. $p_{n,i}(z)$ is a set of six reference points obtained from the current pixle's vertical neighbours from previous and next frame.

The blotch is detected if:

$$d_n(z) > T \text{ with } T \geq 0.$$

The implementation is:

```
% For each frame do
for i = 2:height-1
  for j = 1:width
      refpoints(1) = out(i-1,j,t-1);
      refpoints(2) = out(i,j,t-1);
      refpoints(3) = out(i+1,j,t-1);
      refpoints(4) = out(i-1,j,t-2);
      refpoints(5) = out(i,j,t-2);
      refpoints(6) = out(i+1,j,t-2);
      if (min(refpoints) - out(i,j,t)) > 0
          dn(i,j,t) = min(refpoints) - out(i,j,t);
      elseif (out(i,j,t) - max(refpoints)) > 0
          dn(i,j,t) = out(i,j,t) - max(refpoints);
      else
          dn(i,j,t) = 0;
      end
  end
end
img = out(:,:,t);
d = dn(:,:,t);
d(d(:,:)<Tn) = 0;
d(d(:,:)>0) = 1;
```

4.2 Discussion and solution of potential problems of algorithm

Setting threshold for detection would always run into a question, how to choose the proper value to make the algorithm more general. In my case, I choose the value which is suggested by the paper.

The algorithm won't work because I haven't discussed a serious factor yet, that is

the motion. The first footage contains several carriages comes by which creates a large dark area throughout its taxing. This would in term casue large pixel differences between frames.

To avoid this, we need to estimate the motion then make a mask accordingly.

Figure 8: Cumulate sum of frame difference

Figure 8 is the sum of first 15 frames differences, we can see the motion of carriage leaves a very bright trail. We need to utilise this discovery to make the mask. Here is the implementation:

```
for div = 1:divisions
  sumup = 0;
  for t = (div-1)*subseqsize : (div)*subseqsize-1
      sumup = sumup + abs(diffMat(:,:,t+1));
  end
  t = max(sumup(:))*0.25;
  sumup = sumup - t; % Only the densed area remained
  % Using the average filter to connect the disjoints
  filter = fspecial('average', 60);
```

```
10    sumup = imfilter(sumup, filter);
11    % Do a close operation to close the concave shape.
12    se = strel('disk',50);
13    sumup = imclose(sumup,se);
14    rgSize = 55; % region growing by 55 pixels
15    se = ones(rgSize, rgSize);
16    sumup = imdilate(sumup,se);
17    sumup(sumup(:,:) >0 ) = 1;
18    sumup(sumup(:,:) <0 ) = 0;
19    cumulateSum(:,:,(div-1)*subseqsize+1 : (div)*subseqsize) = sumup.* ...
          ones(size(v,1),size(v,2),subseqsize);
20  end
```

Now we have the mask, the masked area will tell the algorithm ignore the pixel and proceed. Let's look at the result and improvement: I have highlighted the blotches for

Figure 9: Image with blotches

you, here is the comparison before using motion mask and after motion mask: The bizzare look of carriage on the left is fixed. I also using imflate to increase the blotch region so the blotch correction looks more realistic on the right.

Figure 10: Left:without mask Right: With mask

5 Correction of camera shake

While doing a photography, handheld camera would always introduce frame unstablised or motion blur for longer exposure, not to mention shooting video without a tripod. In theory, shaking is not limited to panning along x and y axis but also inwards and outwards,or even different perspective plane, the later might need homography to fix but is too comlicated to discuss in this coursework. So I will assume the camera shake is happened within a 2D space, which is x and y translation. The task for this section is to develop an algorithm that correct the shakings.

5.1 Initial General Approach

My idea to this task is to use the spatially template matching. A template moving across the targeted frame and compute the differencese. I choose the tamplate to be the top-middle area with several considerations. Becuase of composition technique, photographer tends to put more elements in the centre than the edges and top of the image tends to have less variation. Choosing a good template may not increase the accuracy but would greately faster the program as there are less sub-routinue to execute.

The referece could be arbitrary and I choose the first frame to be the reference frame. The idea of using template matching is after the matching, we have a table formed by moving offsets and the difference values. The lowest value among them represents the area that best resembles the template. So we do the template matching for each frame with

the reference frame, we find the lowest difference value thus get the pair of offsets. These offsets tells the degree of shake, so the correction of shake is to apply the negative value of those offsets. Fortunately, matlab provides a handy function for template matching: the cross correlation:normxcorr2.

5.2 Problems of algorithm and refinement

In practise, doing template matching frame by frame is time-consuming, so I divide the frame into top and bottom parts, the program will only perform matching over bottom part if there is no reliable matching for the top part.

Since we are using the minimum value so it is not always reliable. The feasible matching result is defined to be matching confidence less than 60 percent, or, the offset is within a certain limits, the value of limits is set to be adaptive as it is obtained based on the patch size. Having such curb procedure would make sure there is no absurd translation. Here is the implementation of shaking correction including the curb procedure.

```
1   for t = 1:frame_num
2   current_frame = restoration(:,:,t);
3   tm_matrix = normxcorr2(patch,current_frame);
4   [m_max, n_max] = find(tm_matrix==max(tm_matrix(:)));
5   if tm_matrix(m_max,n_max) < 0.7 % not a reliable result
6       %continue;
7       % Do template matching with the revious frame instead of the
8       % reference frame
9       patch2 = v_d( startidx_m:patch_tail-round(template_size/6), ...
            startidx_n: width - round(width/6),t-1);
10      tm_matrix = normxcorr2(patch2,current_frame);
11
12      [m_max2, n_max2] = find(tm_matrix==max(tm_matrix(:)));
13      max2 = tm_matrix(m_max2,n_max2);
14      %if tm_matrix(m_manx2,n_max2) < 0.6
15      % Template matching for the lower part of the frame.
16      startidx_m3 = round(template_size/6) + patch_tail +1;
17      patch3 = v_d( startidx_m3:height-round(template_size/6), ...
            startidx_n: width - round(width/6),1);
18      tm_matrix = normxcorr2(patch3,current_frame);
19      [m_max3, n_max3] = find(tm_matrix==max(tm_matrix(:)));
```

```
20        max3 = tm_matrix(m_max3,n_max3);
21        if(max2 > max3)
22           if(max2 <0.7)
23               frame_shifted = ...
                     circshift(current_frame,[offset_sets(1,1,t-1) ...
                     offset_sets(1,2,t-1)]);
24               % Check the offset size if it is too large.
25               if(startidx_m+patch_head-m_offSet > ...
                     size(patch,1)*0.1)||(startidx_n-n_offSet-1> ...
                     size(patch,1)*0.1)
26                    continue;
27               end
28               offset_sets(1,1,t) = startidx_m+patch_head-m_offSet;
29               offset_sets(1,2,t) = startidx_n-n_offSet-1;
30               restoration(:,:,t) = frame_shifted;
31               continue;
32           end
33           m_max = m_max2;
34           n_max = n_max2;
35           patch = patch2;
36           m_offSet = m_max-size(patch,1);
37           n_offSet = n_max-size(patch,2);
38           frame_shifted = ...
                circshift(current_frame,[startidx_m+patch_head-m_offSet+...
39                   offset_sets(1,1,t-1)startidx_n-n_offSet-1+offset_sets(1,2,t-1)]);
40           if(startidx_m+patch_head-m_offSet > ...
                size(patch,1)*0.1)||(startidx_n-n_offSet-1> ...
                size(patch,1)*0.1)
41               continue;
42           end
43           offset_sets(1,1,t) = startidx_m+patch_head-m_offSet;
44           offset_sets(1,2,t) = startidx_n-n_offSet-1;
45           restoration(:,:,t) = frame_shifted;
46           continue;
47        else
48           if(max3 <0.7)
49               frame_shifted = ...
                     circshift(current_frame,[offset_sets(1,1,t-1) ...
                     offset_sets(1,2,t-1)]);
50               if(startidx_m+patch_head-m_offSet > ...
```

```
                    size(patch,1)*0.1)||(startidx_n-n_offSet-1> ...
                    size(patch,1)*0.1)
51                    continue;
52                end
53                offset_sets(1,1,t) = startidx_m+patch_head-m_offSet;
54                offset_sets(1,2,t) = startidx_n-n_offSet-1;
55                restoration(:,:,t) = frame_shifted;
56                continue;
57            end
58            m_max = m_max3;
59            n_max = n_max3;
60            patch = patch3;
61            startidx_m = startidx_m3;
62        end
63    end
64    m_offSet = m_max-size(patch,1);
65    n_offSet = n_max-size(patch,2);
66    frame_shifted = ...
          circshift(current_frame,[startidx_m+patch_head-m_offSet ...
          startidx_n-n_offSet-1]);
67    if(startidx_m+patch_head-m_offSet > ...
          size(patch,1)*0.1)||(startidx_n-n_offSet-1> size(patch,1)*0.1)
68       continue;
69    end
70    offset_sets(1,1,t) = startidx_m+patch_head-m_offSet;
71    offset_sets(1,2,t) = startidx_n-n_offSet-1;
72    restoration(:,:,t) = frame_shifted;
73    %imshowpair(uint8(patch),uint8(current_frame),'montage');
74
75  %     figure
76  %     imshow(uint8(v_d(:,:,t)));
77  %     imrect(gca, [n_offSet+1, m_offSet+1, size(patch,2), ...
        size(patch,1)]);
78  end
```

One cause of unreliability is the occlusion, like the carriage covering the whole frame. To tackle this case, we can choose the referece frame more flexible, like two frames before, so each frame is doing the alignment with the t-2 frames. However, this would introduce drifting, a very common object tracking problems, that is, if one frame is not correctly

aligned, then all the subsequent frames would all doing the wrong alignment and this error alignment is cumulative.

The shake correction(translation) is too trivial to be observed by just showing the image of frames, I suggest viewing the corrected video, which would make more sense of the stablisation.

References

[1] J. Biemond P.M.B. van Roosmalen R.L. Lagendijk *Restoration of Old Film Sequences.*

[2] J. Biemond P.M.B. van Roosmalen R.L. Lagendijk *Restoration and Storage of Film and Video Archive Material.*