Computer Vision Challenge 2020

Foreground and Background Detection of Videos

Yujia Gu, Qiyue Huang, Donghao Song, Murong Xu

July 12, 2020

1 Introduction

In these years, video-call has became a more crucial part for social use such as business, education, and communication among family and friends. It provides lots of flexibility for remote working while maintaining a relatively high quality to simulate the real physical meetings. To protect individual privacy, many video conferencing softwares allow users to change their background during the call. So in this year's computer vision challenge, out task is to perform segmentation of foreground and background from a given video sequence.

The main strategy of our work is to use Robust Principal Component Analysis and decompose the LAB-based input frame to low-rank and sparse matrices. Then, applying frame differencing to ensure the completeness of object contour. And lastly, combining morphological transformations and the method of "Block-wise ROI" to extract the foreground object precisely. One of the highlight of our work is that we use the adaptive parameters to process the images, namely analyzing each frame individually and then performing the segmentation. In this way, the detected results will be more robust against scene variance.

In the next chapter, we will describe some important techniques and algorithms used in our project. After that, the experiment procedures will be given in chapter 3, including several attempts to find the suitable input data and adaptive parameters to process the boundary features. Lastly, the generated results from six videos will be presented in chapter 4.

2 State of the Art

2.1 Robust Principal Component Analysis

Through out the years, the method of sparse and low-rank modeling has shown remarkable progress in many applications, i.e., face recognition, latent semantic indexing, and video surveillance. To realize the decomposition-based object detection, Robust Principal Component Analysis [1] has brought the performance to a new level.

Suppose an observed data matrix M is the superposition of low-rank component L and sparse component S, as shown in Equation 1:

$$M = L + S \tag{1}$$

For a given measurement matrix M, it is possible to recover L and S exactly by solving the convex program "Principal Component Pursuit", i.e., minimizing a weighted combination among all feasible decompositions in Equation 2:

$$\begin{aligned} &\min \|L\|_* + \lambda \|S\|_1, \\ &\text{subject to} \quad L + S = M \end{aligned} \tag{2}$$

where $\|\|_*$ refers to the nuclear norm (sum of eigenvalues), $\|\|_1$ is $\ell 1$ norm (sum of absolute values), λ is the normalization factor and is computed by $\lambda = 1/\sqrt{\max(\text{size}(M))}$.

For a given video, we can first reshape each frame from the sequence to a row vector to obtain matrix M of size [number_frames, number_pixels]. It is reasonable to make assumption that foreground object is likely to account for relatively small amount in the scene, and the background is usually static and thus has low variance. Hence, we can identify the moving foreground objects by computing the sparse matrix S, and the stationary background based on low-rank matrix L. In comparison with traditional PCA, the RPCA can outperform when the measurements are arbitrarily corrupted, for instance the non-Gaussian distributed noise.

In this project, we use the library of [4] which is based on the fast principal component pursuit via alternating minimization algorithm [3]. It can further reduce the computational cost while constructing the low-rank and sparse component of the same equality.

2.2 Concept of Frame Differencing

The idea of performing frame differencing is to take absolute difference between current frame and the previous one back to k frames, as described in the formulation 3:

$$D_t = |F_t - F_{t-k}|, \tag{3}$$

where D_t is a created difference image at time t, F_t is the original image at the current time and k refers to the step size of the flashback. For instance, with an extraction frame rate at 30 fps in ChokePoint dataset, using step size $k = 1$ means that the time difference between adjacent frames is 0.1s.

By doing this, we ignore a great amount of pixels which belong to the static objects by subtracting two adjacent frames, i.e., the background. Since moving objects are usually challenging to predict, we can now preserve and highlight more information of them while maintaining a relatively high processing speed.

However, simply using traditional difference frame is likely to fail in detection of pixels which have no significant intensity changes over time series, i.e., area belongs to cloth or body with relatively large surface and uniform texture. Inspired by the idea from [2], we use the "three frame difference" method fused with RPCA to realize the accurate extraction of moving foreground object. The algorithm works as follows:

1. Use the background of current frame, i.e., the low-rank matrix L produced by RPCA as a reference frame.
2. Compute foreground images of both previous and current frame $foreground_1$, $foreground_2$ by calculating absolute difference between the two original frames and the reference background respectively.

$$\text{foreground}_1 = |\text{frame}_1 - \text{reference}| \tag{4}$$

$$\text{foreground}_2 = |\text{frame}_2 - \text{reference}| \tag{5}$$

3. Convert the two foreground images to binary representation $binary_1$ and $binary_2$, using threshold value T.

$$\mathrm{binary_1(x,y)} = \begin{cases} 1 & \mathrm{foreground_1(x,y)} \geq \mathrm{T} \\ 0 & \mathrm{foreground_1(x,y)} < \mathrm{T} \end{cases} \tag{6}$$

$$\mathrm{binary_2(x,y)} = \begin{cases} 1 & \mathrm{foreground_2(x,y)} \geq \mathrm{T} \\ 0 & \mathrm{foreground_2(x,y)} < \mathrm{T} \end{cases} \tag{7}$$

4. Assume that the pixel locations from most of the moving objects will not change within two neighboring frames. So perform logical conjunction on the two binary foregrounds to obtain the final detected moving objects.

$$\mathrm{foreground_{final}(x,y)} = \begin{cases} 1 & \mathrm{binary_1(x,y)} \cap \mathrm{binary_2(x,y)} = 1 \\ 0 & \mathrm{binary_1(x,y)} \cap \mathrm{binary_2(x,y)} = 0 \end{cases} \tag{8}$$

This procedure is also shown in Figure 1.

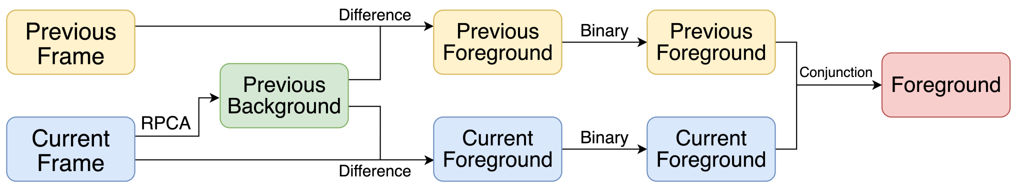


Figure 1: Procedure of performing RPCA-based three difference frame method

In this way, it can overcome the hurdle of producing to many "holes" in the object interior. Moreover, since there is an additional step of foreground conjunction, the extracted result will also be more reliable for complicated scenarios and robust against the environmental noise and decision failures.

2.3 Foreground Extraction with Higher Precision

Due to the occurrence of noisy, the result of aforementioned processing can not be directly used to obtain a complete foreground mask. Therefore, additional morphological processing is necessary. We first apply a median filtering. It is widely used in image denoising since it can remove noise without destroying edge features. Secondly, an morphological opening operation is adopted to further remove noise pixels, an result example is shown in Figure 2.

After eliminating the noise, a precise and complete foreground extraction scheme is implemented by generating separated Region of Interests (ROIs) based on several adjacent foreground sub-blocks. This method is explained in Figure 3. We first separate the resulting image obtained from morphological transformation vertically into multiple sub-blocks with the same size. In each sub-block, we find all of the white pixels and analyze their indices to get the largest rectangle area, namely the pixel positions of four vertices that locate in the top-left, top-right, bottom-left, and bottom-right. In this way, the precision of image segmentation can be easily controlled by the number of sub-blocks, i.e., more sub-blocks, the higher precision can be achieved. After that, all

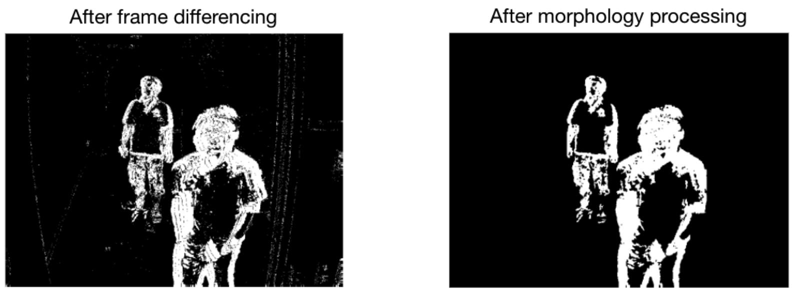


Figure 2: An illustrative example of morphological processing

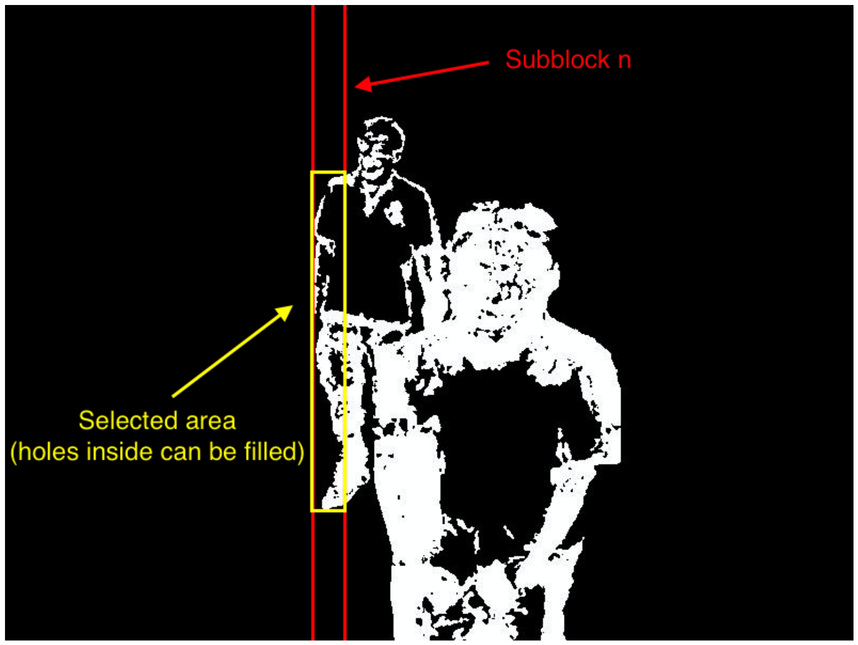


Figure 3: Explanation of sub-block ROI processing

of the pixels that inside the extracted rectangle area will be set to 1. Lastly, by combining all the sub-blocks we can get a complete mask of foreground object and without interior "holes", see Figure 4.

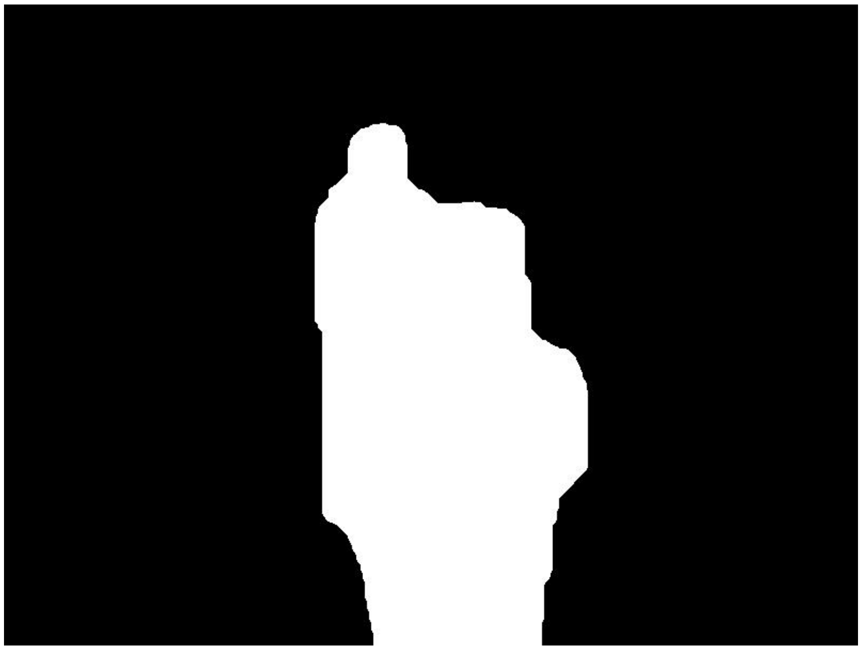


Figure 4: The result of sub-block processing (number of sub-blocks is 200)

2.4 Adaptive threshold

The result after the step of frame differencing describes the difference between the selected frame and the previous one. Hence, a pixel with larger intensity value here can indicate a larger difference of the captured scene. So it is reasonable to classify a pixel as foreground when it has relatively large intensity, since the moving objects can lead to big changes. On the other hand, a pixel that has small intensity value can be identified as background since static objects change very little. Figure 5 shows a distribution of pixel intensity values of an differencing result.

Based on the assumption that the majority of pixels from the image should belong to background (highly concentrated near the origin) and only a small part is moving (with the most largest intensity values), we can relatively easily figure out a threshold to distinguish foreground and background and then convert it to a binary representation. However, a fixed threshold can lead to unreliable segmentation results since the scene varies. Therefore, we adopt an adaptive threshold method. To prevent the misleading that caused by some noisy frames, we assume that there will not be significant changes within 100 consecutive frames. So that the adaptive threshold can only be updated for every 100 frames. This adaptive threshold updating algorithm works as follows:

1. Extract the foreground using morphological transformations and Block-wise ROI Method.

2. Analyze the distribution of intensity values from current frame.

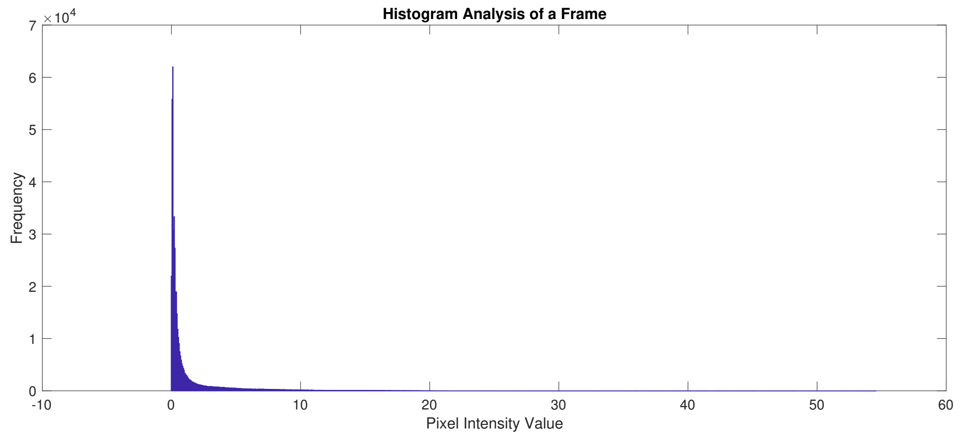


Figure 5: Pixel intensity value distribution of frame differencing result

3. Determine the adaptive threshold according to the percentage of foreground object. If there are many pixels belonging to the foreground and can account for 30% in entire image, the threshold will be set to 0.6 to fit the high-density foreground case. Otherwise, the threshold will be set to 0.85 to deal with the situation that foreground object occupies only small image area.

The whole procedure is described in Figure 6.

2.5 CIELAB Color Space

Since color images allow for more reliable image segmentation than for gray scale images, choosing a proper color space can be beneficial. There are different types of color spaces available for use, for instance, L*A*B*, HSV, RGB, YCbCr, they are all frequently chosen options. Among them, the study of [5] showed that L*A*B* is best suited for foreground segmentation, as the intrinsic color component is better presented in L*A*B* color space.

The L*A*B* color space is derived from the CIEXYZ tristimulus values, which consists of a luminosity component L*, a chromaticity layer A* related to red-green color axis, and a chromaticity layer B* related to blue-yellow axis. Since the luminance and chrominance information are already well separated, L*A*B* color space can outperform when dealing with separation of image pixels in the field of color. The conversion between RGB and L*A*B* is described in Equation 9 and 10, where it needs the XYZ space as a bridge.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.43057 & 0.34155 & 0.17833 \\ 0.22202 & 0.70666 & 0.01733 \\ 0.02018 & 0.12955 & 0.93918 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{9}$$

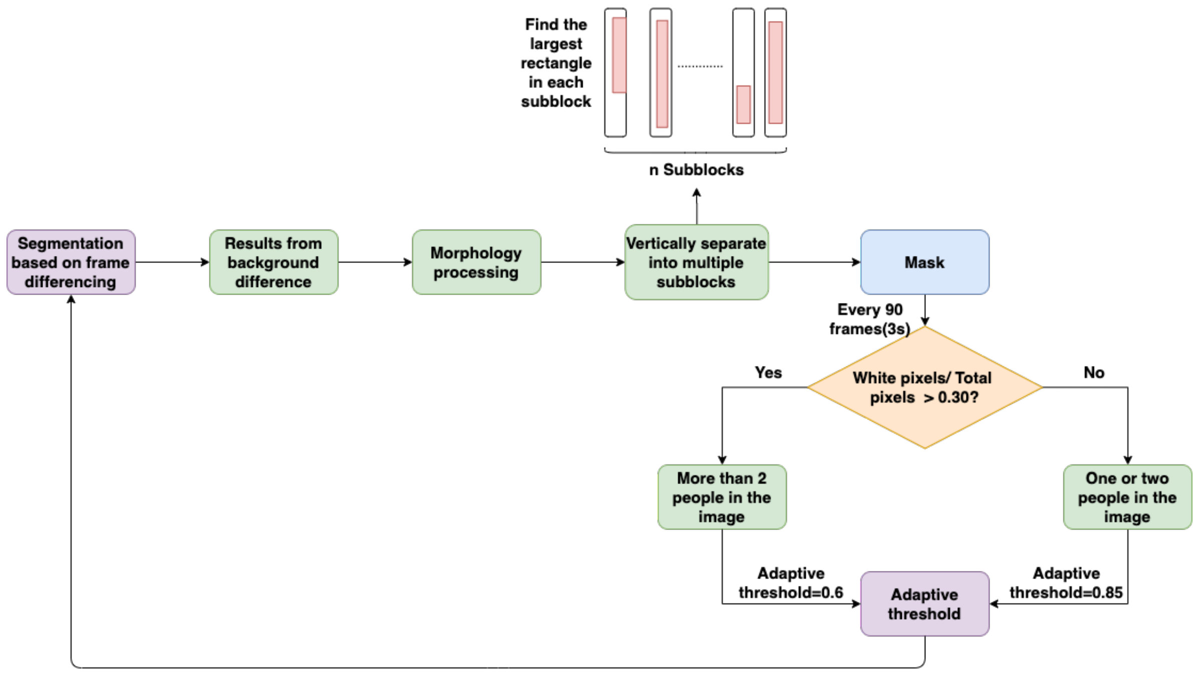

Find the largest rectangle in each subblock
n Subblocks
Segmentation based on frame differencing
Results from background difference
Morphology processing
Vertically separate into multiple subblocks
Mask
Every 90 frames(3s)
White pixels/ Total pixels > 0.30?
Yes
No
More than 2 people in the image
One or two people in the image
Adaptive threshold=0.6
Adaptive threshold
Adaptive threshold=0.85

Figure 6: Procedure of updating adaptive threshold

$$\begin{aligned}
&L = 116f(Y/Y_n) - 16\\
&a = 500(f(X/X_n) - f(Y/Y_n))\\
&b = 200(f(Y/Y_n) - f(Z/Z_n))\\
&f(t) = \begin{cases} t^{1/3}, & t > 0.008856\\ 7.787t + 0.13793, & t \leq 0.008856 \end{cases}
\end{aligned} \tag{10}$$

where $X_n = 95.047$, $Y_n = 100$ and $Z_n = 108.883$ are the CIE XYZ tristimulus values of the reference white point.

Since L* represents the pure brightness and has no relation to the color of the image, it can better respond to shadow and other lighting changes and thus detect the edge characteristics of moving foreground objects more efficiently. So after transforming the original RGB frame into L*A*B* color space, we use the L* component to perform further segmentation.

3 Function implementation

3.1 Load Images

Since the time interval between two frames is only 0.03s with a frame rate at 30 fps, there is not significant difference between neighboring frames. Hence, computing mask for each individual frame is unnecessary. However, the number of total frames can not always be divisible by $(N + 1)$. To keep every input tensor with the same size, an auto-complete method is used to avoid the case that the size of last tensor is different. Namely, when the number of frames in the last tensor is less than $(N + 1)$, a certain number of frames that before the first frame of the last tensor will be automatically loaded. The size of each input tensor is thus the same.

Additionally, *nargin* is used in ir.next() to prevent error when there is no input of variable *start* and N (both optional). We set a fixed order of inputs as follows: *src*, L, R, *start*, N. When the last two inputs are not given, *start* will be set to default value 0 and N to 1.

3.2 Segmentation of Foreground and Background

In order to accelerate the processing speed, we calculate only one mask for each image tensor which consists of $N + 1$ frames. According to the preliminary experiments, the best result can be obtained when $N = 4$. Figure 7 shows the results of each key steps that in segmentation algorithm (number of sub-blocks is 200, and $N = 4$). In morphological processing, functions such as medfilt2(), bwareaopen(), bwareafilt(), and filloutliers() are used to remove noise. After morphological transformation, we separate the resulting image into multiple sub-blocks. In each sub-block, we first identify white pixels and use find() to get the corresponding indices. According to the indices, the coordinates of each white pixel can be obtained by using ind2sub(). To get the largest rectangle area that contains the foreground, we use max() and min() to find the pixels that located in the top-left, top-right, bottom-left, and bottom-right positions of the sub-block. Then, set the pixels that inside the rectangle to 1 to fill the interior holes. Lastly, we combine all of the sub-blocks to get the final binary mask.

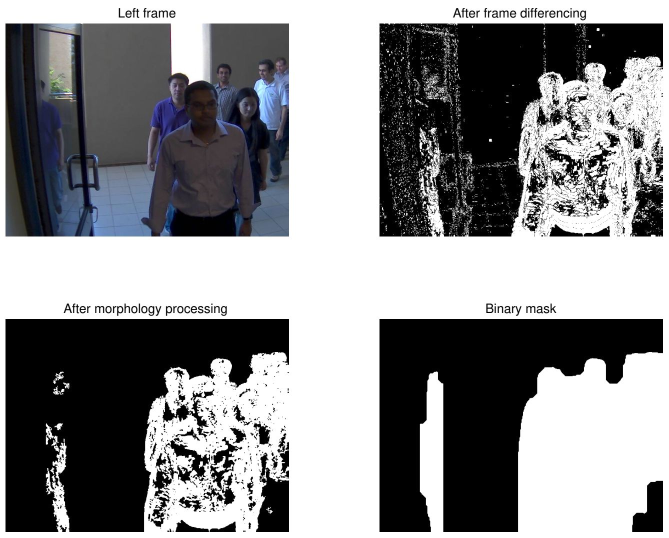


Figure 7: Results of each key step in segmentation

Figure 8 shows the low-rank objects that are computed by fast RPCA algorithm.

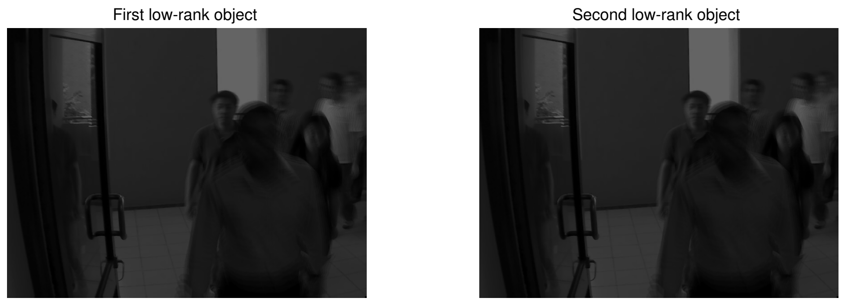


Figure 8: Low-rank objects obtained using fast RPCA

3.3 Rendering

After obtaining the binary mask from segmentation, we can render the current frame from left camera in 4 different modes, i.e., 'foreground', 'background', 'overlay', and 'substitute'. The rendered images are showed in Figure 9.

3.4 GUI

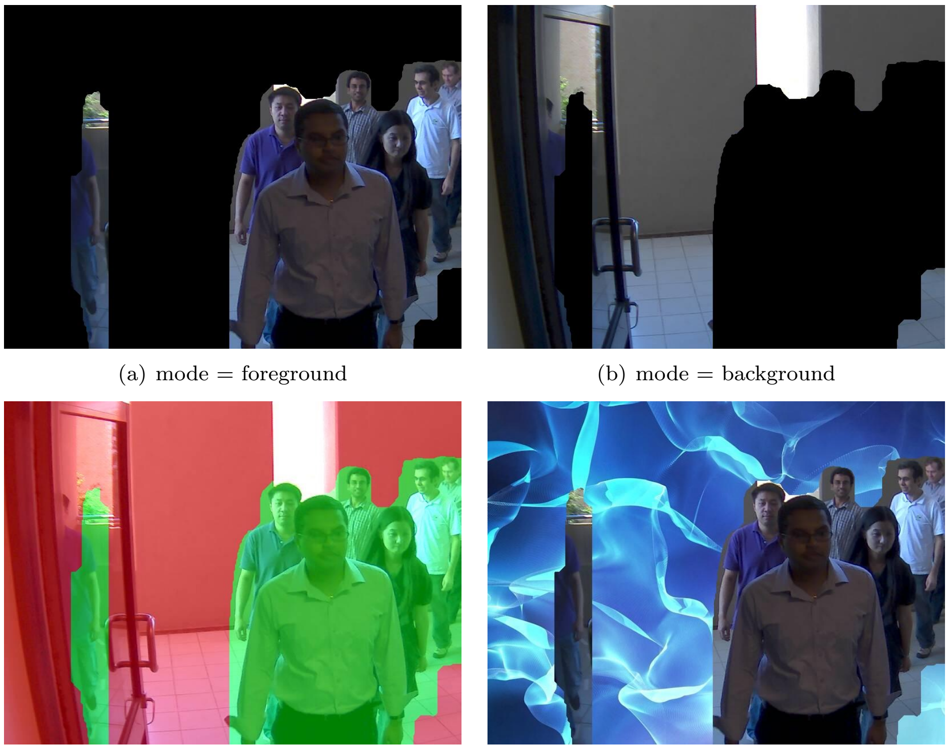
The GUI interface is shown in Figrue 10.

3.4.1 Introduction of Select Buttons

The GUI interface includes three select buttons. The function of the first select button is used to choose the path of original video. This path will be saved at the app class property "app.src", and it will be used as one of the input parameters of the function 'ImageReader'. The second select button is used to select the path to save the portion of the original video that read and the processed video. A folder named 'results' will be created under this path. These two videos will be saved in the folder 'results'. With the third select button you can choose the path of the background, if the $BgMode$ set to 'substitute'. If the $BgMode$ is not 'substitute', you do not need to choose the $BgPath$. Moreover, the background image or video will be resized to the same size as original video frame by call the function 'ResizeVideo' and then save it to a cell variable. This cell variable will be used in the function 'render'. If the background is a video, the video is split into individual frames and saved to the folder 'backgroundVideo'

3.4.2 Introduction of other fields

L for left camera, it can be chosen between 1 and 2. R for right camera, it can be chosen between 2 and 3. The value of $start$ is the starting frame from this frame the frames of left camera and right camera will be read. It can only be set between 0 and the length of the original video frames. After give a reasonable value to start, this value will be send to the app class property 'app.startFrame', and it will be used as input parameter of the function 'ImageReader'. N is the number of images

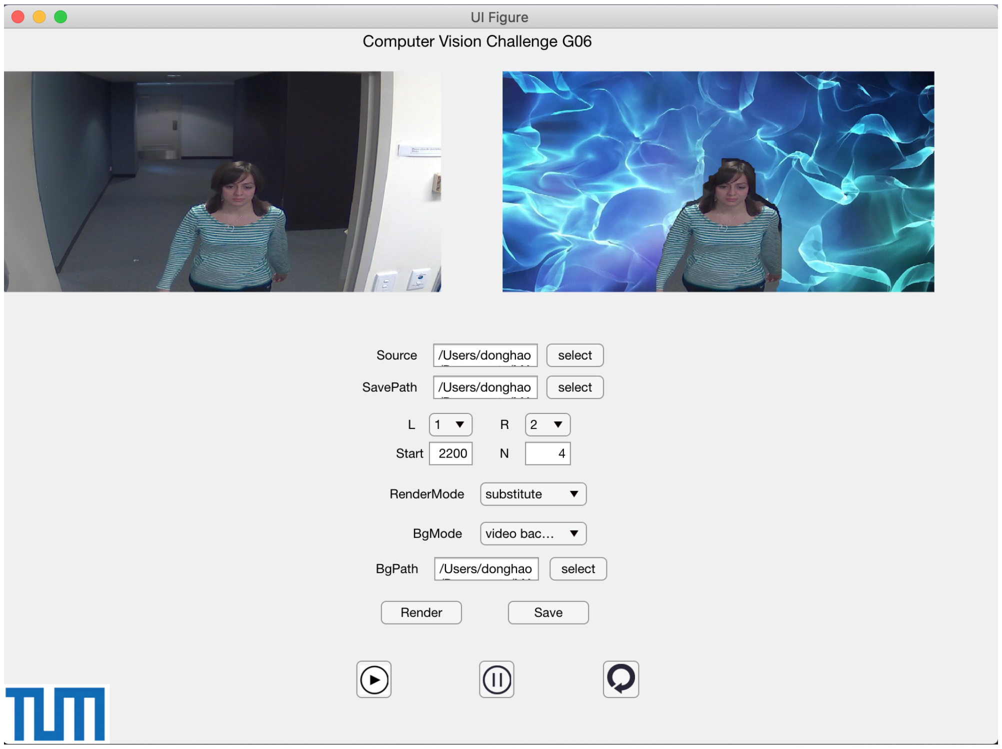

(a) mode = foreground

(b) mode = background

(c) mode = overlay

(d) mode = substitute

Figure 9: Resulting image with different rendering modes



Source /Users/donghao select
SavePath /Users/donghao select
L 1 R 2
Start 2200 N 4
RenderMode substitute
BgMode video bac...
BgPath /Users/donghao select

Figure 10: GUI interface

processed at a time. In our tests, we found that the algorithm used in this paper works best when N is set to 4. *render_mode* is the input parameter of the function 'render'. It can be chosen between 4 render modes. You can see its details at the file named 'readme.txt'. The value of $BgMode$ determines what the background image will look like. It needs to be set only when the *render_mode* is 'substitute'.

3.4.3 Introduction of other buttons

RENDER button is the main button of this GUI. After you set all necessary variables you can press this button to process the original video frames. The functions 'ImageReader', 'segmentation' and 'render' are called sequentially. After that all processed frames and its original frames will be saved separately to the folders named 'Movie' and 'oriVidFrame'. Also, the created folder 'backgroundvideo' will be deleted. Then you can press the SAVE button to convert these frames to video. These two videos will be saved under the folder 'results' by call the function 'SaveMovie'. Then the folders 'Movie' and 'oriVidFrame' will be deleted. Up until now all files for displaying are ready. Now press the play buttons to display it. The way how it is displayed is shown in the figure 10. There are two screens. The left screen is for original video and the right screen is for rendered video.

4 Results

The figure below shows the results of the rendered images in different scenes with mode = foreground, number of $sub - blocks = 200$, $N = 4$:

In scene P1E_S1, the number of people is 1 or 2, the light condition is soft light. The image in Figure 11 shows the result that, the foreground can be accurately and completely segmented.

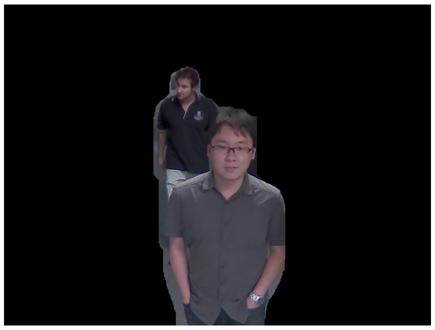


Figure 11: P1E_S1,L=1,R=2,start=230

The scene P1L_S3, P2E_S2 and P2E_S4 are similar, only one person appears. Due to the strong sunlight from the window, it is more difficult to extract the foreground, but it can also be correctly recognized as the background. The images in Figure 12 respectively show the results in scene P1L_S3, P2E_S2 and P2E_S4:

In scene P2L_S5 and P2E_S5, more than one person appear in the captured scene, which makes the foreground more complicated. In addition, the influence of illumination caused by strong sunlight (scene P2L_S5) and strong lights (scene P2E_S5) makes the foreground more difficult to

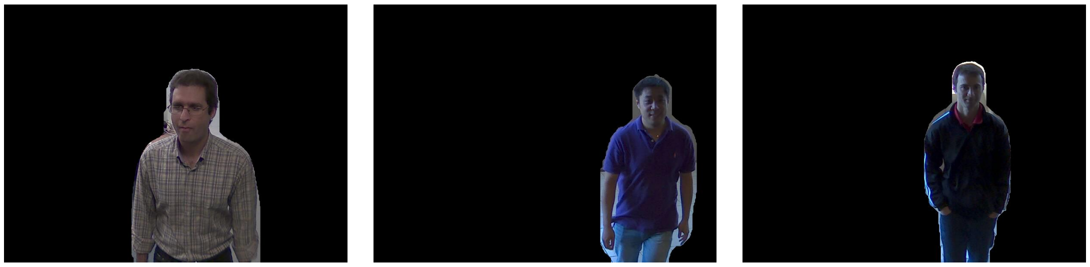

(a) P1L_S3, L=2, R=3, start=2000 (b) P2E_S2, L=1, R=2, start=2000 (c) P2E_S4, L=1, R=2, start=1670

Figure 12: Result with single person and strong light intensity

extract. Even so, we can see from the images in Figure 13, the boundary and corner points can be also clearly identified and segmented. But it can be seen that the distance between the outline of the foreground and that of people slightly increases.

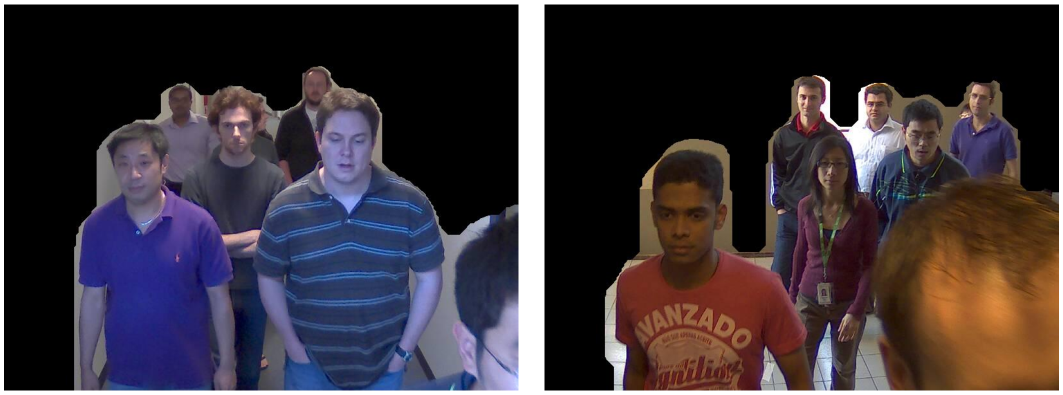

(a) P2L_S5, L=2, R=3, start=450 (b) P2E_S5, L=2, R=3, start=400

Figure 13: Result with more people and strong light intensity

References

[1] Emmanuel J. Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *CoRR*, abs/0912.3599, 2009.

[2] Xiaojing Li Jie Kang. Moving object detection based on three frame difference fused with rpca. 2018.

[3] P. Rodríguez and B. Wohlberg. Fast principal component pursuit via alternating minimization. In *2013 IEEE International Conference on Image Processing*, pages 69–73, 2013.

[4] Andrews Sobral, Thierry Bouwmans, and El-hadi Zahzah. Lrslibrary: Low-rank and sparse tools for background modeling and subtraction in videos. In *Robust Low-Rank and Sparse Matrix*

Decomposition: Applications in Image and Video Processing. CRC Press, Taylor and Francis Group., 2015.

[5] X. Wang, R. Hänsch, L. Ma, and O. Hellwich. Comparison of different color spaces for image segmentation using graph-cut. In *2014 International Conference on Computer Vision Theory and Applications (VISAPP)*, volume 1, pages 301–308, 2014.