



# Computer Vision Challenge

## Report of G37

| Matrikel-Nr. | Name          |
|--------------|---------------|
| 03729693     | HONGYU SONG   |
| 03728965     | YUCHONG ZHENG |
| 03728467     | HUIWEN ZHENG  |
| 03724681     | JIAMIN WU     |
| 03697297     | RYAN LEUNG    |

---

# Computer Vision Challenge Report of G37

## Table of Contents

|                          |    |
|--------------------------|----|
| Introduction             | 2  |
| Algorithm                | 2  |
| Image Reader             | 2  |
| Segmentation             | 2  |
| Rendering                | 6  |
| Configuration            | 7  |
| Challenge                | 8  |
| Graphical user interface | 12 |
| Results                  | 15 |
| The rendered frames      | 15 |
| The accuracy analysis    | 17 |
| The speed analysis       | 17 |
| Conclusion               | 18 |
| References               | 18 |

# 1. Introduction

In this report, we use the left and right camera images to separate the foreground and the background, and replace a virtual background, which can be a picture or a video. This project can make our home office meetings during the pandemic more formal and professional.

In Section 2, we explain the principle and detailed steps of each function, and the obtained results. In the 3 section, we show the rendered image examples from scenes P1E\_S1, P1L\_S3, P2L\_S5, P2E\_S2, P2E\_S4 und P2E\_S5 and also analyze the speed at which we processed the image. In the fourth section, we summarize the results of our project and give possible optimization methods.

## 2. Algorithm

### 2.1. ImageReader

ImageReader is a class that requires the arguments *src*, *L*, *R* at instantiation and has optional arguments *start* and *N*.

*src* - relative or absolute path to scene folder (P\*X\_S\*; \*=integer; X='L','E'); either Windows or Unix style

*L,R* - L=1,2; R=2,3 - indicates which camera should be taken as the left/right camera

*start* - default value 0; it indicates which image in the folder to start from

*N* - default value 1; indicates how many succeeding images to display

At instantiation, the arguments are parsed and checked to see if they satisfy constraints. After that using some string manipulation the list of \*.jpg files are obtained.

The class has a method *next()*, which returns *left*, *right*, *loop*.

*left,right* - tensors of size  $H \times W \times (N+1) \times 3$ , where H=height, W=width, and *N* is as parsed at instantiation. The tensors contain the 3-channel images of the left and right cameras as given by *L,R*.

*loop* - 0 indicates there are still images remaining in the folder, 1 indicates that no images remain

The first execution of the method *next()* returns the *N*+1 images beginning from *start*; subsequent executions will return *N*+1 images beginning from the image after the last image of the previous execution. If there are no remaining images, or fewer than *N* images remaining, only the remaining images will be returned regardless of *N*. The next execution will then start from the beginning of the folder (*start*=0).

## 2.2. Segmentation

### 2.2.1. Brief description of the working principle

This module is mainly used to segment the foreground and background of the image. It consists of three parts, and the flow chart is shown in Figure 2.1

1. Image Gaussian mixture distribution clustering background training
2. Image background subtraction
3. Digital image processing

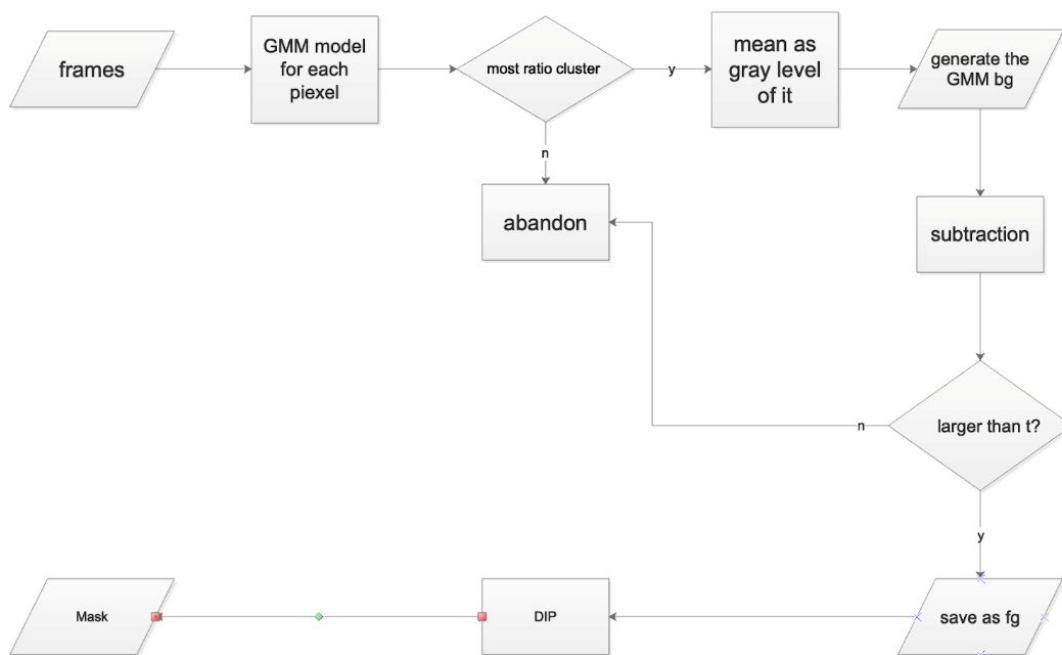


Figure 2.1: flow chart of segmentation

### 2.2.2. Detailed description of each step

After getting the flow chart of the program for segmentation, we want to introduce the program into a detailed description of each step.

First, we want to introduce the image Gaussian mixture distribution [1] clustering background training which is shown in Formula 2.1. Here  $X$  is the dataset with  $n$  pictures,  $u$  is the mean value of these pictures,  $d$  is the dimension of the data,  $\Sigma$  is the diagonal matrix of variance and  $u$  is the expectation matrix.

$$f(X) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(X - u)^T \Sigma^{-1} (X - u)\right], X = (x_1, x_2 \dots x_n)$$

*Formula 2.1 : GMM*

The frames of the entire data set are read and treat all gray values at a certain point in the same position in the 600x800 picture array as a binary clustering Gaussian mixed distribution. Since a point appears as the background most of the time, the cluster with a high mixing ratio is the single Gaussian distribution when the point is used as the background. We take the average value of this cluster as its Gaussian background value. In this way, we have taken the entire data set into consideration and provided a more accurate basis for the subsequent calculation of background subtraction. Here we take P2L\_S5\_C1 as an example, we use the first 200 pictures to train the GMM background model, the result is shown in Figure 2.2. And the result using the GMM algorithm for segmentation is shown in Figure 2.3.

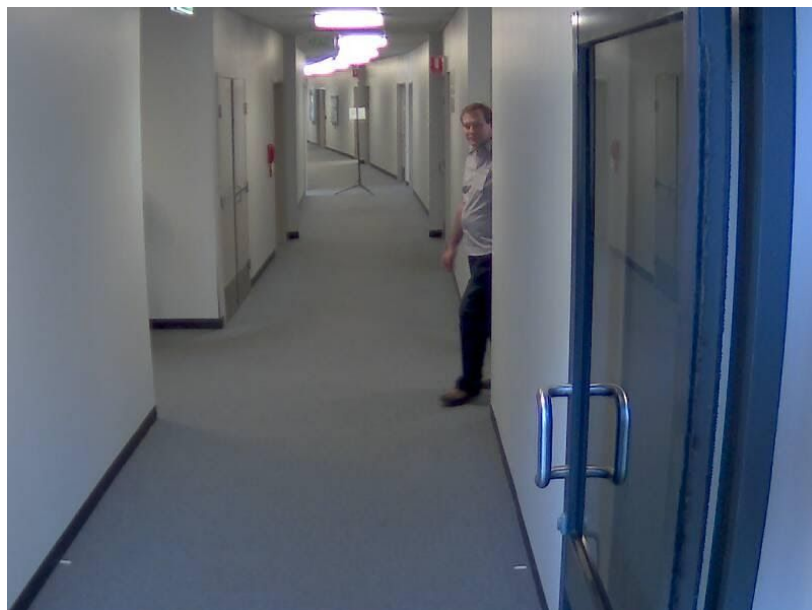


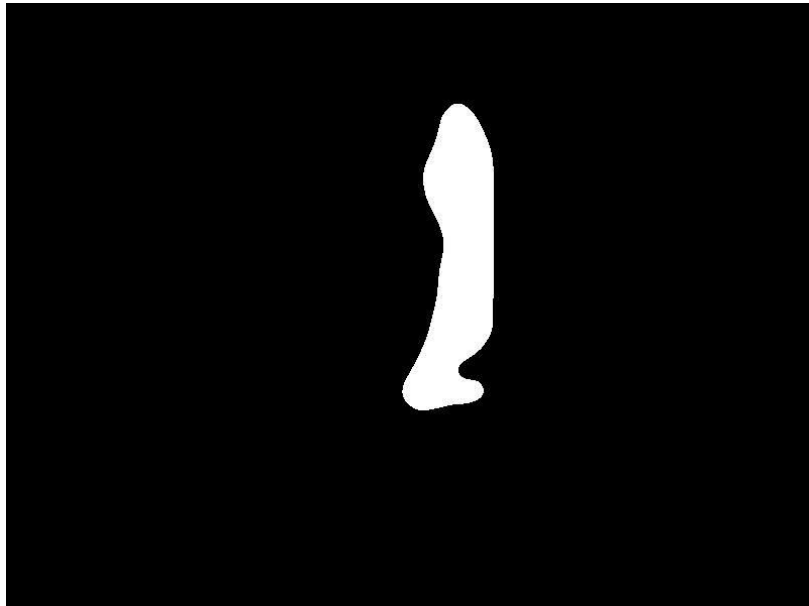
*Figure 2.2: GMM Background*



*Figure 2.3: The Version Using GMM*

In the implementation we find out that the time cost of GMM is too large. Take the folder P2L\_S5\_C1 as an example : In order to get the result of the GMM, it takes about twelve minutes to get the result, which is too slow, because it takes about 2/3 time of the whole program so we decided to not use GMM and to find optimization methods in segmentation. For this reason, we tried the second method. Here we use a combination of median filter and Gaussian filter to extract the foreground of the picture, and finally get a better result. As for the background picture, we choose the first picture of each scene, which does not have any foreground people. The result of the mask is shown in Figure 2.4. As can be seen, with the combination of Gaussian filtering and median filtering still gets a good result and the boundary is also clear.





*Figure 2.4: The Version without GMM*

After getting the trained background, we use image background subtraction to get the background. In order to get the background, we set a threshold of 0.12. If the absolute value of the difference between the pixels of the frame and the background is greater than this threshold, then the pixel is classified as foreground, and we set this point as 1, while for the values smaller than the threshold we set it to 0. This method yields the binary mask. After further optimization of the binarized image, we can use these masks in render to get the result. The program is shown in Figure 2.5.

```
2 %% foreground segmentation
3   % initial setting
4 -   frame = rgb2gray(im2double(frame));
5 -   threshold = 0.12;
6   % initial mask: if the absolute difference between frame and background
7   % is greater than the threshold, then the pixel is 1
8   % (foreground), otherwise 0 (background)
9 -   foreground_gray = abs(frame - background_one) > threshold;
```

*Figure 2.5: Binarized Picture*

To achieve this result in the segmentation part we use Gaussian filter and median filter because Gaussian filter can make the system function smooth and avoid ringing, and that makes the boundary of the picture easy for the computer to recognize. For median filter, the purpose is that it is especially useful for speckle noise and salt and pepper noise. The code is shown in Figure 2.6.

```

19 -         mask = double(mask);
20 -         % Use gaussian filter to blur the edges
21 -         mask = imgaussfilt(mask,2)*10;
22 -         % Removing salt and pepper noise by twice median filtering
23 -         mask = medfilt2(mask,[5 5]);
24 -         % thicken edges to clump together small connected components
25 -         mask = bwmorph(mask,'thicken',1);
26 -         % fill holes
27 -         mask = imfill(mask,'holes');
28 -         mask = medfilt2(mask,[40 40]);
29 -         mask = imfill(mask,'holes');
30 -         mask = medfilt2(mask,[25 25]);
31 -         mask = imfill(mask,'holes');

```

Figure 2.6: Optimizing

## 2.3. Rendering

### 2.3.1. Brief description of the working principle

This part is the function to achieve the four mode choices, 'foreground', 'background', 'layout', 'substitute'.so that we can according to the mask from segmentation to set the background and foreground in four modes.

### 2.3.2. Detailed description of each step

1. Transform the *frame* and *bg* from *uint8* to *double*.

2. Using switch to choose the mode :

- **foreground**: Elementwise multiply mask and frame, so that the background pixels will be set to black (0).
- **background**: Elementwise multiply the inverse of the mask and frame, so that the foreground pixels will be set to black (0).
- **overlay**: The red and green channels of the image are extracted. Using *find()* on the mask to obtain the indices of only the foreground pixels, the red channel of the foreground pixels is multiplied by 0.8 to give a red tint. Using *find()* on the inverse mask to obtain the indices of only the background pixels, the green channel of the foreground pixels is multiplied by 0.5 to give a green tint. The modified red and green channels are then combined with the original blue channel to obtain the result where the foreground is tinted red and the background is tinted green.
- **substitute**: Firstly, elementwise multiply mask and frame as in *foreground* mode, so that the background pixels are 0. Then, elementwise multiply the inverse mask and the virtual background (resized to the same 800x600 as the dataset) so that all the



foreground pixels of the virtual background are 0. Then, add these two together to obtain the foreground on the virtual background.

3. Finally, the result is converted to *uint8* and returned.

### 2.3.3. Obtain the effect picture

Using a short test function, which sets the background is a small square (mask=0), the following shows the effect in the four modes.

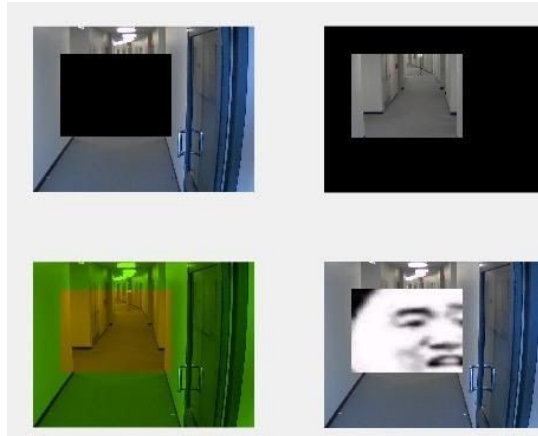


Figure 2.7: 4 modes of rendering

## 2.4. Configuration

*config.m* is an important file that decouples the calculation from the special settings of the computer. It contains all the necessary variables and is called before the main program *challenge.m*. Through this configuration file, we can modify all the variables that need to be set to quickly run other people's code. According to the actual operation, it contains the following three major variables:

1. General settings: Group number, members' names, members' email addresses.
2. Reading settings: Scene folder path, left and right camera numbers, starting point, the number of succeeding frames.
3. Output settings: virtual background image path, rendering mode, whether to store, video output path.

## 2.5. Challenge

### 2.5.1. Procedure steps

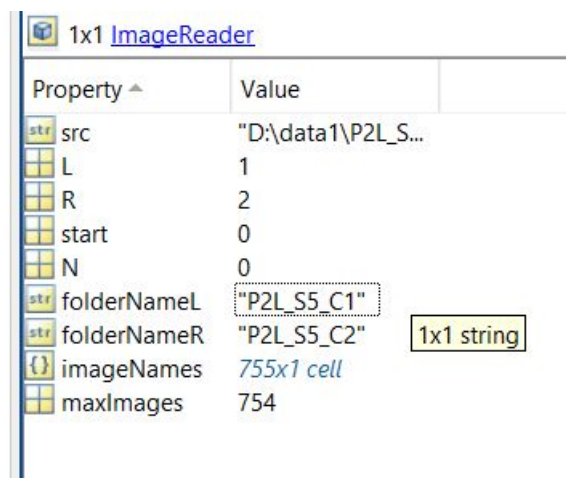
In order to get the final video processing result, the challenge procedure steps mainly include the following four steps:

1. Read each picture file from the scene and camera folders.
2. Get the mask corresponding to the foreground and background of the picture from the segmentation.
3. Render a new image from the input picture and the mask according to the selected mode.
4. Output the result file as a video, if *store*=true.

## 2.5.2. Detailed description of each step

### 2.5.2.1. Pre-Processing

In order to get the final video processing result, the challenge procedure steps mainly include the following four steps. First, we analyze how to read files from the folder. In chapter 2.1 of the reference material we can have the class *ImageReader* and its method *next()*. They have many purposes: First, obtain the basic information in each subfolder of the folder, including the name, path and number of picture files in the subfolder and the second is use the images in the folder to get foreground. After instantiating an *ImageReader* object, we obtain the basic information in the folder (Shown in Figure 2.8).



| Property ^  | Value               |
|-------------|---------------------|
| src         | "D:\data1\P2L_S..." |
| L           | 1                   |
| R           | 2                   |
| start       | 0                   |
| N           | 0                   |
| folderNameL | "P2L_S5_C1"         |
| folderNameR | "P2L_S5_C2"         |
| imageNames  | 755x1 cell          |
| maxImages   | 754                 |

Figure 2.8: Detailed Information of ImageReader

Since we use the first picture of every scene to turn it into the background, for example, we use the first picture in P2L\_S5\_C1 as the background. The codes are shown in Figure 2.9.

```

7      % Use First Picture as Background
8  -   ir2 = ImageReader(src, L, R, 0, 0);
9  -   [leftImages,~,~] = next(ir2);
10  -   background_one = rgb2gray(im2double(leftImages));

```

Figure 2.9: Using the first picture as background

### 2.5.2.2. Mask Processing and Image Writing

After the training, it is convenient for subsequent background extraction methods to carry out the foreground extraction processing. We perform the second part, using a while loop to read all the image information in the folder until loop=1 to exit out of the loop. In addition, we will also use the *render* function in the program to obtain the required pictures. For example, in our program we can use “substitute” to get the picture where the background is replaced with the selected virtual background and then write it to the output video file if desired. The detailed code part will be shown in Figure 2.10, the processing result is shown in Figure 2.11. The details of foreground extraction techniques have been discussed in chapter 2.2.

```

48 - [left, right, loop] = next(ir);
49 - % obtain segmentation mask
50 - mask = segmentation(left, background_one);
51 - % render the frame with the mask, background and mode
52 - renderedFrame = render(left,mask,bg,mode);
53 - % save the output to video file if this is desired
54 - if store
55 -     writeVideo(aviobj,renderedFrame);
56 - end

```

Figure 2.10: Get Mask of Each Picture and then Render and write to video



Figure 2.11: The Final Result

In this section we also consider using a video as background, so we use *VideoReader* to get the video and use them in loop while turning the video into pictures. When the frame of the video reaches the last frame, the program sets it back to frame zero to make sure background pictures are also looped. The code is shown in Figure 2.12.

---

```

24     % if the background is a video
25 -   if vidbg
26       % VideoReader object
27       v = VideoReader(fullfile(bg));
28       % read the frames of the video
29 -     vidFrames = read(v);
30       % set the current background frame to index 0
31 -     bgFrame = 0;
32 -   end
33
34     % loop is 0 initially
35 -     loop = 0;
36 -   while loop == 0
37       % if the background is a video
38 -     if vidbg
39         % increment frame counter by 1
40         bgFrame = bgFrame + 1;
41         % extract the next frame from the background video
42 -       bg = vidFrames(:, :, :, bgFrame);
43         % if we've reached the end of the background video, loop back to the beginning
44 -       if bgFrame == v.NumFrames
45         bgFrame = 0;
46 -       end
47 -     end

```

*Figure 2.12: Use Video as Background*

### 2.5.2.3. Movie

After obtaining all the processed frames, we need to write these graphic files as a movie file. So here we need to use the *VideoWriter* object in Matlab. Before processing the movie, we need to determine some basic movie information. In this program, we determine the information of the three movies: the playback speed of the movie, the path of the movie to read the picture, and the frame at which the movie starts and ends. The codes are shown in Figure 2.13.

```

12     % if we are saving video output
13 -   if store
14       % fps of the video
15 -     fps = 30;
16       % VideoWriter object at path of output video file (from config.m)
17 -     aviobj = VideoWriter(fullfile(dest));
18       % set framerate
19 -     aviobj.FrameRate = fps;
20       % open the VideoWriter object so we can write into it
21 -     open(aviobj);
22 -   end

```

*Figure 2.13: The Code of Movie Pre-Processing*

After all images have been processed and written to the video file, it can be closed. The method is shown in Figure 2.14.

---

```

58      % close the output video file
59 -    if store
60 -        close(aviobj);
61 -    end

```

Figure 2.14: Closing the VideoWriter object

## 2.6. Graphical user interface

### 2.6.1. Brief description

For users, the graphical user interface is more easily accepted and used than the command interface. In this project, the function of the GUI is like the combination of the *config.m* file and the main program *challenge.m*. Finally, it can be started by the command *start\_gui*. We choose App Designer to finish this task, because App Designer provides more cosmetic options, is easier to use than GUIDE and also because GUIDE will be deprecated in the near future.

### 2.6.2. Detailed description of each step

We create some public properties to deliver the values in the different components, so we can just through **app.x** get or change those values. We create the following properties, **src**, **mode**, **L**, **R**, **N**, **state**, **start**. These properties will also appear in the subfunctions as necessary parameter inputs.

We employ many App Building Components for the required tasks:

- As for **Scene Folder Path**, **Virtual Background Path** and **Save Path**, we choose the buttons on which we add the corresponding callback function. If we click the button, the corresponding callback function will be executed. If we change the value of the Scene Folder Path button, then the *app.src* will be accordingly changed. In the same situation, the *app.bg* and *app.savepath* corresponding to the virtual background path and the save path will also change accordingly when we click on these buttons to select the paths.
- As for **L**, **R** and **Mode**, we use the Dropdown components to select the number of the camera. Hence the *app.L* and *app.R* will be changed.
- To input the value of the **starting point**, we add the Edit Field component. If we change the value of the starting point, then the *app.start* will receive the value.
- We use three buttons to implement the *play control* function and add the icon for each button. If we click the **start** button, the main function will be executed and the video of the input (left and right camera) and the output video will display. Then we click the **stop** button, the playing of the video will be stopped. But if we click the **loop** button, the playing of the video will be endless.
- For the display of the *input* and *output* video, we use three UIAxes. We use the *imshow* and *drawnow* function to achieve the display.

We create the **startupFcn** function. In this function, we initialized all the parameters and assigned a default value to all the parameters, which can prevent each variable from being omitted in the configuration.

For the play control function, we set a variable state: *start*, *stop*, *loop*, which can be changed with three buttons. We also write callback functions in the three buttons to complete the output of the entire project. In the callback function of the start button, we combine the background generating and the content in *challenge.m* together. We use a loop and *app.state* to achieve the play control function. For start, *loop*=0, *app.state*=1. At the callback function of stop, the *app.state* will be directly assigned as 0. The playing will be stopped as it interrupts the start callback that is running, and the loop within the start callback detects that the *app.state* has changed to 0.

### 2.6.3. Obtain the picture

The picture below is the interface of our design

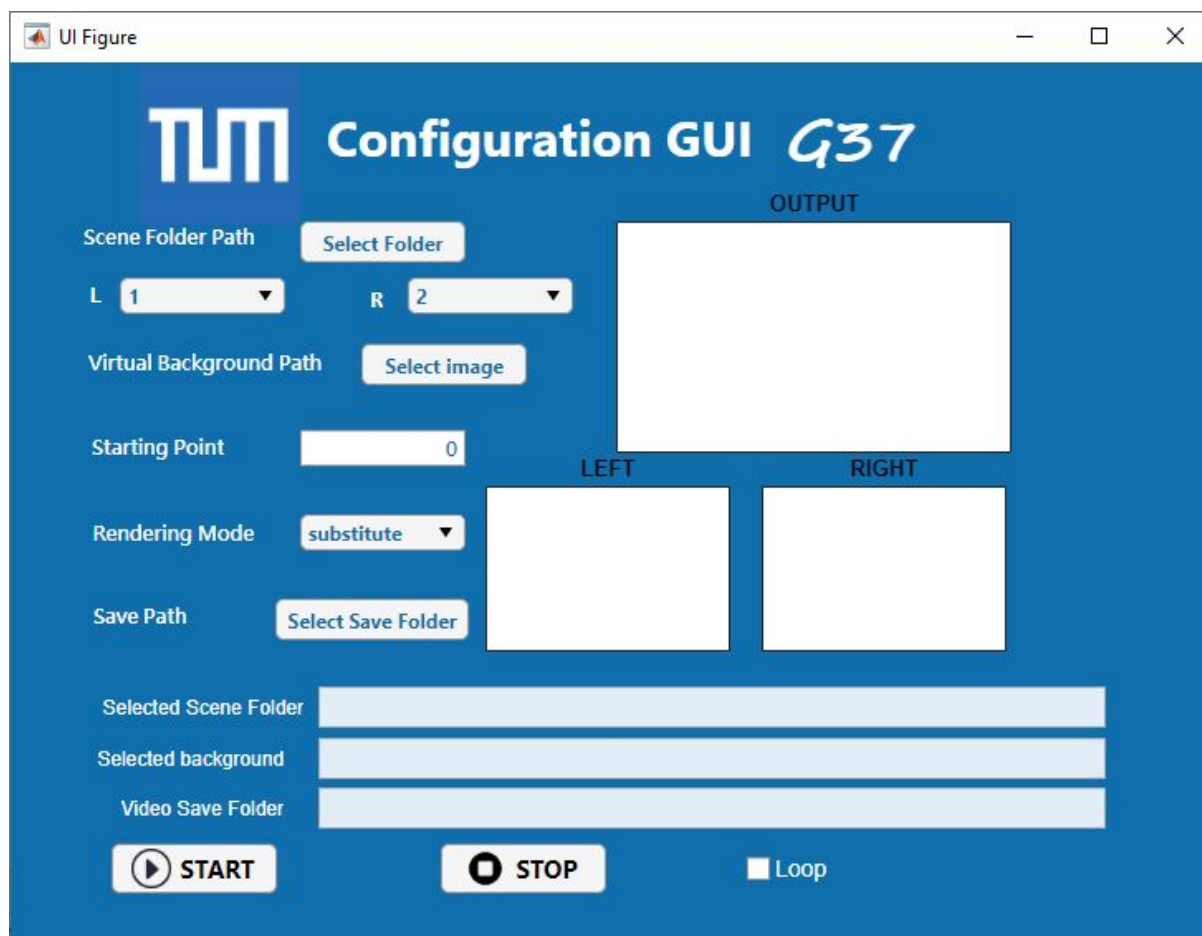


Figure 2.15: GUI

- **Scene Folder Path** : Select the scene folder path
- **L ,R**: Select the left (1 or 2) and right (2 or 3) camera



- 
- **Virtual Background Path:** Select the virtual background path
  - **Starting Point:** Select the starting point (nonnegative integer)
  - **Rendering Mode:** Select the render mode (foreground/background/overlay/substitute)
  - **Save path:** Select the rendered movie store path
  - **Play control:** start, stop, loop (endless play)
  - **Left,Right:** Display of the inputs
  - **Output:** Display of the rendered output

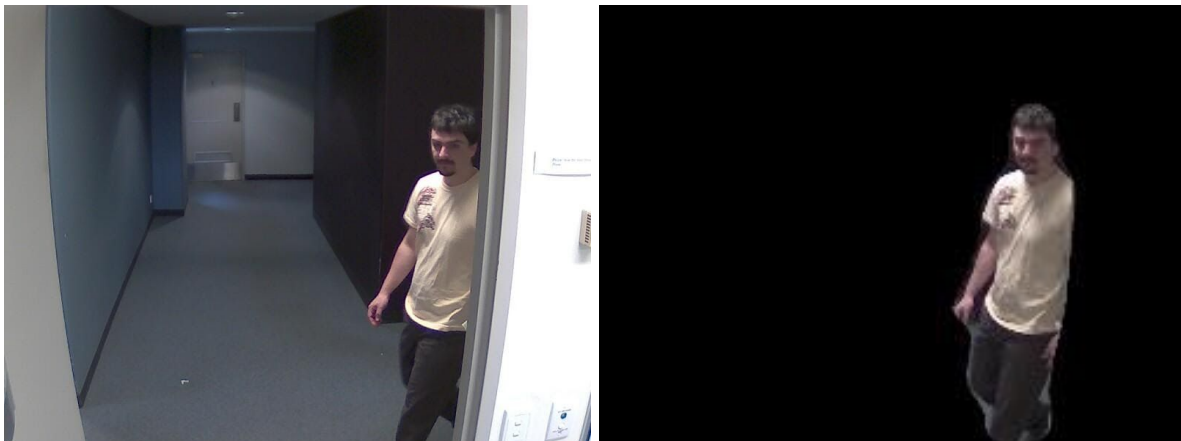
As can be seen, by completing this configuration, we can get the output that we want.

## 3. Results

### 3.1. The rendered frames

According to the method we use, we list result examples of the processed images by mode “foreground”. The left side of the following image is the original frames from the dataset, and the right side is the rendered image.

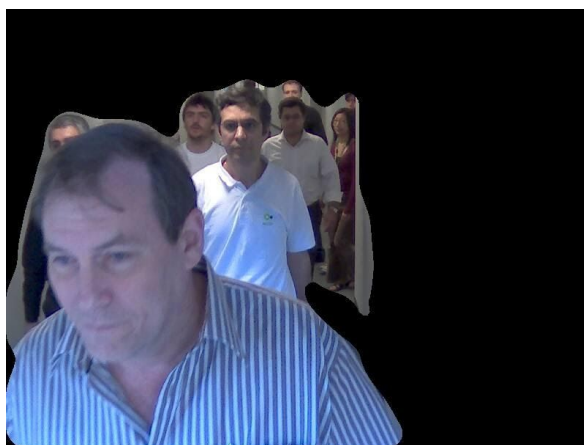
P1E\_S1



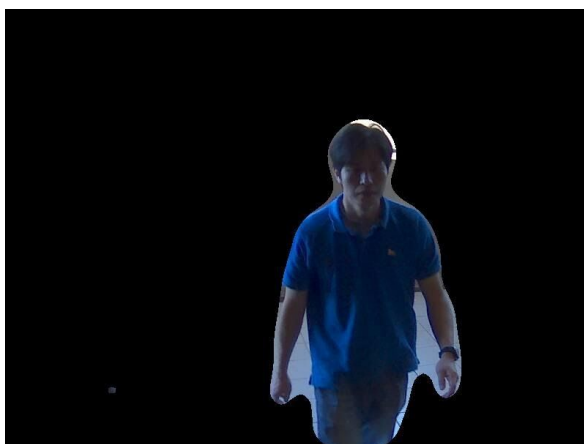
P1L\_S3



P2L\_S5

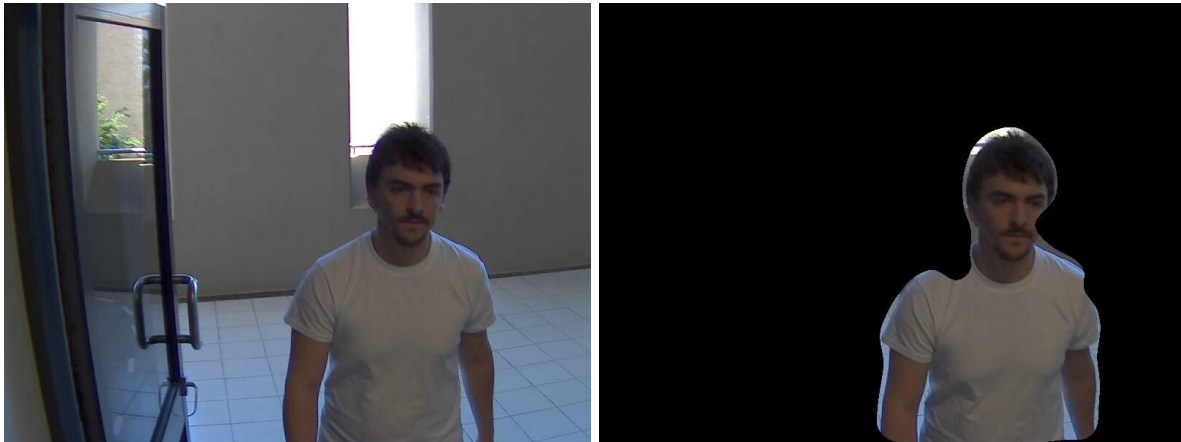


P2E\_S2



P2E\_S4





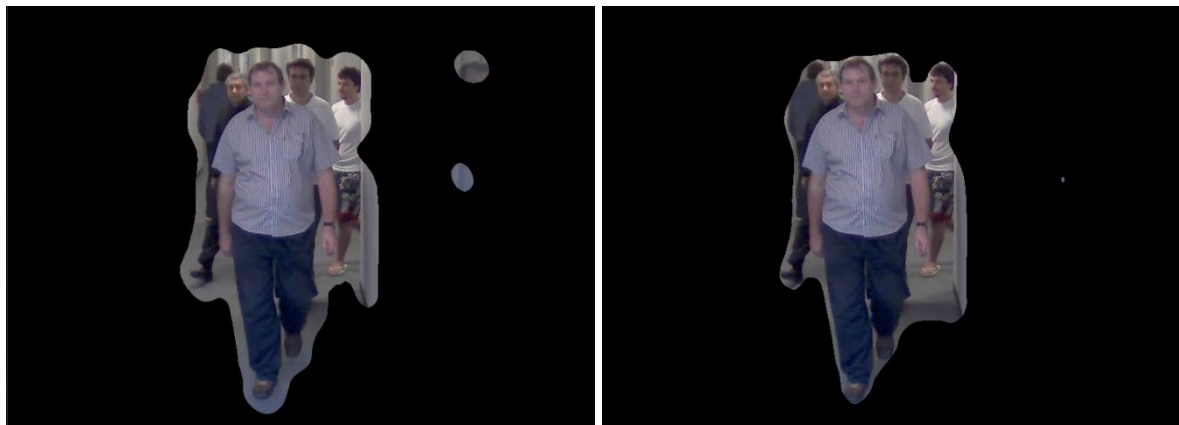
P2E\_S5



*Figure 3.1: **Left:** the original frames **Right:** rendered frames*

As you can see, in each scene, the image processing by the “foreground” mode achieves a certain accuracy, basically reaching the goal of this project. It can realize the recognition of people and convert the background to black. But there still are some errors at the edges of the characters: sometimes the limbs of the characters are not recognized, and sometimes the border is too wide. This is where optimization needs to be continued in the future.

## 3.2. The accuracy analysis



*Figure 3.2: Rendered Frame by GMM (Left) and*

*Using combination of median filter and Gaussian filter (Right)*

In the accuracy analysis, we can clearly see that there is no significant change of using the rendered frame by GMM and the combination of median filter and Gaussian filter. Based on the consideration of speed, we choose the method of median filter and Gaussian filter.

## 3.3. The speed analysis

Since we have already tried the GMM algorithm in Section 2.2, this method requires more time to train the GMM image in the segmentation phase, just in training the GMM image it has already taken about 12 minutes. However, the second method using Gaussian filtering and media filtering for segmentation is faster. This method only needs 0.5 seconds to process a frame, and it takes about 13 minutes to process a 20 seconds video.

In comparison, it is indeed faster to employ the second method to achieve the task.

But obviously, in actual applications, this speed is still far from meeting the practical needs, especially for the video conferences, which requires at least 10 frames per second. Therefore the processing speed is also a point that needs to be optimized.

## 4. Conclusion

In this project, segmentation plays an important role. We try to use GMM and a combination of Gaussian filter and media filter to conduct experiments. As we discussed and analyzed before, the two methods have little difference in accuracy, but in terms of speed, the second method is obviously dominant. In the foreground recognition, the person reflected by the glass door will be mistakenly recognized as the foreground, which can be further optimized in the future.

---

## 5. References

[1] Desert Fox MSFollower “GMM Gaussian mixture model for background modeling (Matlab)”[online] Available at: <[https://blog.csdn.net/LiuPeiP\\_VIPL/article/details/73382354](https://blog.csdn.net/LiuPeiP_VIPL/article/details/73382354)> [Accessed 12 July 2020].