Technische Universität München

Computer Vision SS 2020

Documentation of the Challenge

Group 16:
Alaeddine Yacoub
Kheireddine Achour
Mohamed Mezghanni
Oumaima Zneidi
Salem Sfaxi

Supervisors:
Prof. Klaus Diepold
Stefan Röhrl

July 12, 2020

Contents

1 Introduction

1.1 Motivation

Video calls and online conferences belong nowadays to our daily routine at home. However, it might be considered as a privacy breach to show the chaotic study-room behind us or the dining room in the background, which would be unpleasant for the coworkers at work or fellow students at university. The Computer Vision Challenge in the 2020 summer semester offers a possible way to avoid such inconvenient situations through the distinction between foreground and background of a given scenery and discard undesirable scene elements.

1.2 Task Definition

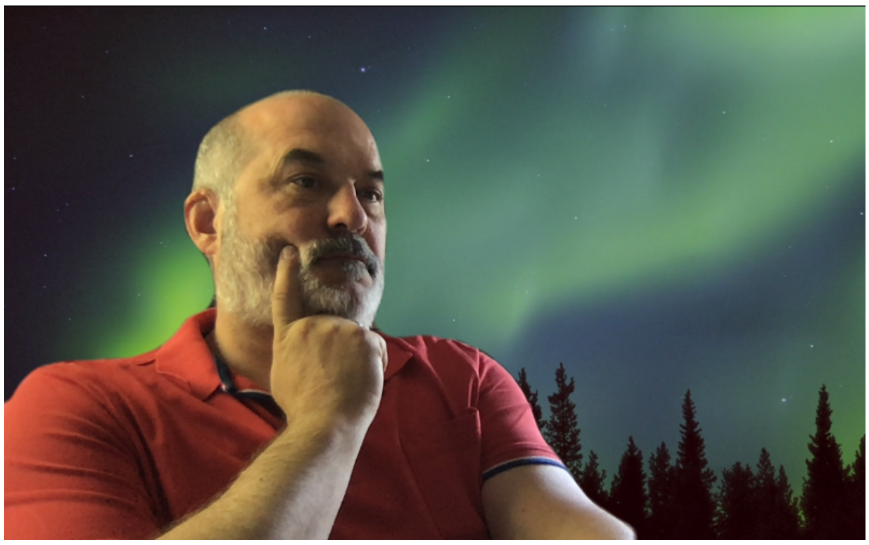


Figure 1: Distinction and foreground replacement of a given scenery.

Figure 1 shows schematically the task. The foreground view, or every non-static element in the scenery, should be located and freely dissectable from the rest of the image (background). With the help of a selectable mode parameter, the foreground and background elements will be modified accordingly and a scene will be rendered depending on the choice of the user (foreground, background, overlay, substitute)

1.3 Task Requirements

The requirements for the program are:

- The program should be created in *Matlab* with the help of the standard libraries and MathWorks Computer Vision toolboxes and run with the version *2020a* on *Windows* and *Linux* complying with a maximum running time of 30 minutes.
- The class *imagereader* should be able to read two of the three available scenery images from a specific folder as an endless loop. The method next should deliver two tensors left and right for each selected camera accordingly and a next functionality to browse the scenery.
- The required function *segmentation* receives the two previously created tensors left and right with successive image pairs as input parameters, in order to create foreground and background for the first pair of pictures. This function returns a binary segmentation mask as an output. Pixels that correspond to the Background are assigned the value 0 and pixels that belong to the foreground get the value 1.
- The program should be able to deal with different levels of scene complexity and handle the edge treatment at the transition between foreground and background.
- Inside the function *rendering* a dissection mode will be selected by the user (foreground, background, overlay, substitute) and an image will be rendered according to the mode.
- Possibility of a virtual background video.
- Operation of the program through a user-friendly graphical interface (GUI).

The algorithms and procedures used to solve the task are described in the following sections. In the section 3 our procedure and the basic program flow are presented in an overview form, before the individual steps are explained in more detail in the following sections.

1.4 Outline of the work

This work contains theoretical fundamentals in image segmentation techniques. A conceptual framework of the task will be later presented before implementing the solutions. Additionally, the results of these methods will be compared and the limitations of each model will be discussed. Finally, an overview of the results will be given and probable outlooks for further works will be suggested.

2 Theoretical Fundamentals

In this chapter, we will introduce the theoretical fundamentals behind the algorithms and techniques for foreground detection we used to solve this task. We will also present other possibilities and aspects to consider when faced with a scene dissection problem.

This fundamental theoretical knowledge will be required for further reading of this work.

2.1 Gaussians Mixture Segmentation

Our first approach was to investigate the Adaptive Gaussian Mixture Models presented by Chris Stauffer and W.E.L Grimson in their paper [**Adaptive background mixture models for real-time tracking**].

We opted for this adaptive approach since the standard non-dynamical models use background substraction over an average scenery and need manual re-initialization for each given sequence. In our case and with 6 different scenes {S1,...,S6} from 3 different camera angles {C1,C2,C3} in our dataset, this would result in a significant workload.

In case we opt for not resetting after each scenery, the background model errors will accumulate over time, due to changes in scene lighting and non-static objects moving slowly. Proceeding this way would result in a less robust backgrounds models and a significantly inaccurate segmentation.

This paper describes modeling each pixel in the image as a mixture of Gaussians distributions and using an online approximation to update the background model after each iteration. These Gaussian distributions are then evaluated based on the variance and their persistence to determine those resulting from a background segment.

The pixel value until a point t in time is represented as a time series $\{X_1, ..., X_t\}$ and modeled by a mixture of K Gaussian distributions. The probability of observing the current pixel value at the timestamp t is

$$P(X_t) = \sum_{i=1}^{K} \omega_{i,t} * \eta(X_t, \mu_{i,t}, \Sigma_{i,t}) \tag{2.1}$$

where K is the number of distributions in the mixture and $\omega_{i,t}$, $\mu_{i,t}$, $\Sigma_{i,t}$ are an estimate of weight, mean value and covariance matrix of the i^{th} Gaussian mixture at time t, and where η is a Gaussian probability density function. The variable K will be determined heuristically based on the image resolution, computer performance and background complexity. The weight $\omega_{i,t}$ represents the data explained by the particular i^{th} gaussian distribution.

The R, G, and B components of each pixel are assumed to be independent. In turn, the probability distribution η for a given pixel $X_t = (R_x, G_x, B_x)$ with mean μ and standard deviation Σ can be computed as:

$$\eta(X_t, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} \mid \Sigma \mid^{\frac{1}{2}}} e^{-\frac{1}{2}(X_t-\mu_t)^T\Sigma^{-1}(X_t-\mu_t)} \tag{2.2}$$

Each pixel will then be assigned to a gaussian distribution model if it is included in 2.5 deviations standard range.

$$\mu_k - 2.5 * \sigma_k < X_t < \mu_k + 2.5 * \sigma_k \tag{2.3}$$

We then sort these distributions by the value of ω/σ^2 and select the highest B distributions (depending on the pre-configured threshold T) as we are interested in strongest weights and minimal variance for formulating our background model.

$$B = \text{argmin}_b \left(\sum_{k=1}^{b} \omega_k > T \right) \tag{2.4}$$

Each pixel is then categorized depending on the classification of its own Gaussian distribution.

The main argument for using this particular method to solve this task is that it is proven to be robust when it comes to lighting changes, repetitive motions of scene elements and slow-moving objects (since their color has a larger variance than the background)

3 Conception and Implementation

In this chapter, we will present our approach to solving the task and discuss the reasoning behind the methods employed.

3.1 General Overview

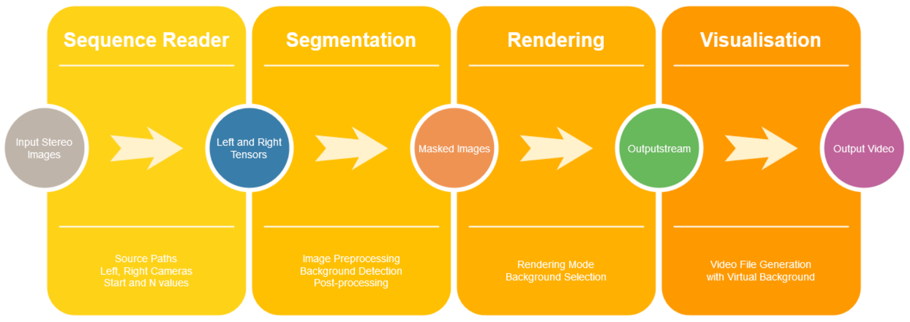
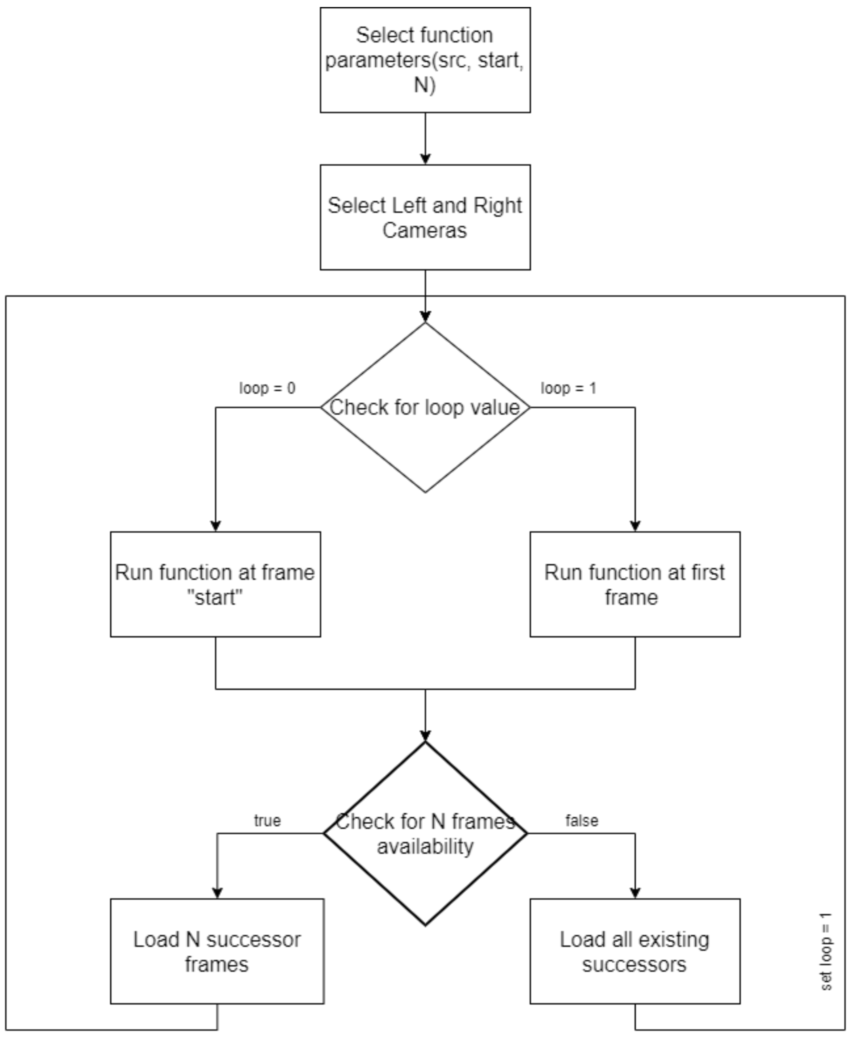


Figure 2: Overview of Concept

3.2 Image Reader

The extraction of the frames follows the logic represented in figure 3.2. The user can pick the starting frame for extraction and the total number of frames to load. If the number of loaded frames reaches the end of the available frames, a loop value will be set to one and the next run will automatically start at the first frame of the stream, independently from the start value.

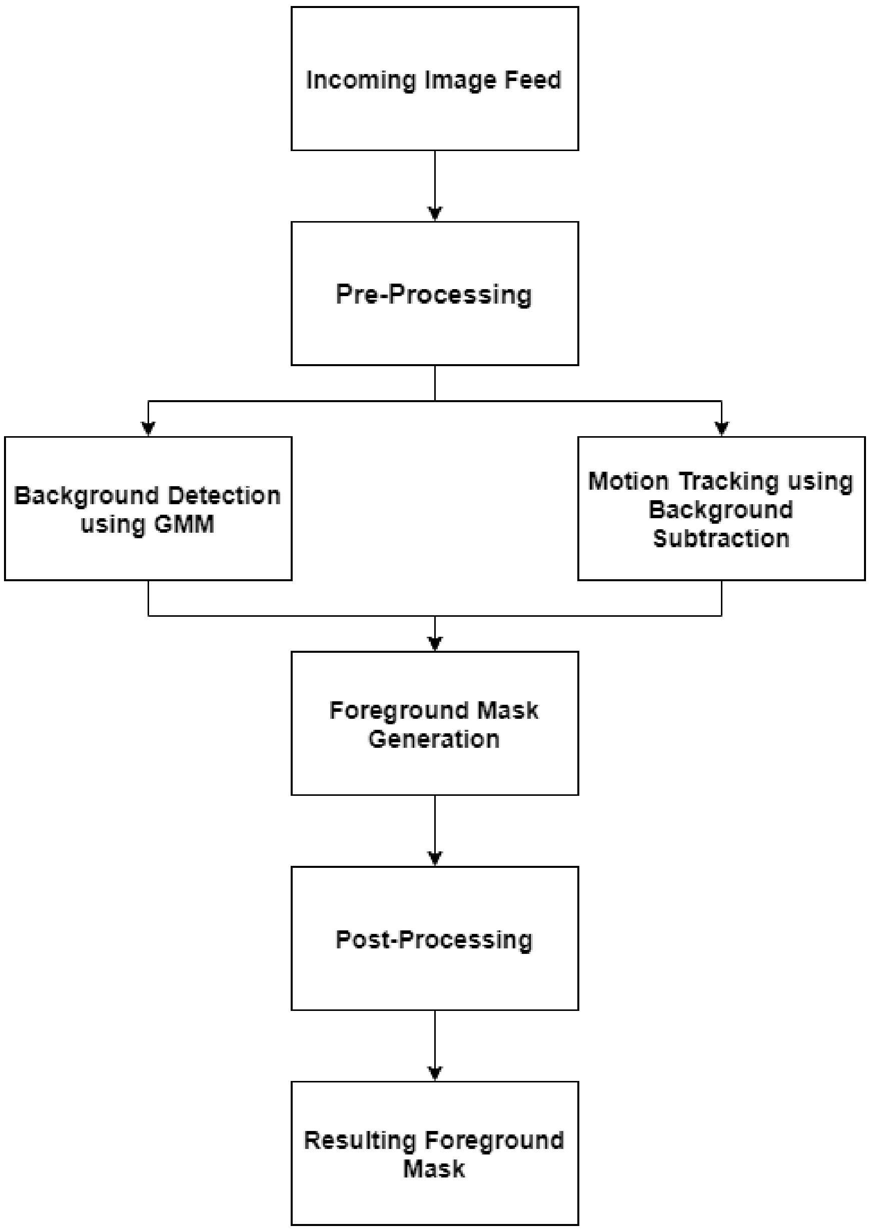

Select function parameters(src, start, N)
Select Left and Right Cameras
Check for loop value
loop = 0
loop = 1
Run function at frame "start"
Run function at first frame
Check for N frames availability
true
false
Load N successor frames
Load all existing successors
set loop = 1

First, to read the images we have in each sequence it was important to implement the ImageReader Class which has several input parameters given through the constructor and the method Next which returns N subsequent image pairs starting at the value specified in the parameter start.
Next, as we have multiple views angles from three different cameras it was important to select the two source paths left path and right path (C1 for the first camera, C2 for the second camera, C3 for the third camera) using the variable src and both of properties L and R which identify the cameras. Due to the gaps in the images numbers, it was more convenient to extract the images pairwise from the text file which contains all their names.
As a further step, The variable loop was initialized with 0 and then could be changed to 1 if the reader had reached the end of the path. Depending on its value the method next would either start reading from the beginning

of the file or the given start.
This method has three outputs: loop variable for the next call of the function and two tensors (left tensor for the left camera, right tensor for the right camera)in which the frames of both cameras will be saved.

3.3 Image Segmentation

The following Block diagram explains our proposed solution for the segmentation part of the program:


Incoming Image Feed
Pre-Processing
Background Detection using GMM
Motion Tracking using Background Subtraction
Foreground Mask Generation
Post-Processing
Resulting Foreground Mask

3.3.1 Pre-processing

Pre-processing the incoming image feed is an important step in a computer vision application. The main goal of this phase is to raise the quality of the data either through the enhancement of certain image features or the removal of undesirable ones. Denoising for example could be achieved by using specific filters such as the median filter (especially known for its efficiency against salt and pepper noise) and the low-pass filter. Gaussian blur could also be used as a Gaussian noise reduction technique as well as a means to enhance some image structures.
Another popular technique commonly used in image processing is normalization. This term could refer at the same time to statistical normalization and also luminance normalization. The former stands for the conversion of all images to "normalized" images having 0 as their mean values and 1 as the standard deviation. As for the latter, the process seeks to change the intensity values of the individual pixels in images, this could be achieved using the method of histogram stretching (also known as contrast stretching) to increase contrast or using histogram equalization to enhance it.

3.3.2 Foreground Detection using Gaussian Mixtures Models

The Computer Vision Toolbox available in Matlab provides a ready-to-use foreground detection method using Gaussian mixtures models. The vision.ForegroundDetector function comes with a set of properties that provide a certain degree of freedom during implementation such as the number of Gaussian modes used in the mixture model and the number of frames used for training background model. Another crucial property of the foreground detector is the Initial Variance which depends on the range of pixel values in the image feed. For images with data type 'double' or 'single', pixel values range between 0 and 1 meaning that the initial variance should be set to $(30/255)^2$ whereas if the images are loaded as a 'uint8' object, then the initial variance takes in a value of 30^2.
As the provided dataset is comprised of mostly images taken inside an indoor environment, we use 2 Gaussian modes in the mixture models; this is due to the fact that indoor images have a static background most of the time thus meaning that increasing the number of Gaussian modes (typically applied to model multiple background modes) may lead, in our case, to wrong or inaccurate detections. For the other properties, the algorithm initially loads the first 100 frames to use for training with an adaptative learning rate corresponding to $1/(current\ frame\ number)$. After this initial training, the learning rate is set to 0.01 for the remainder of the foreground detection process leading at the end to the generation of a binary foreground mask.

3.3.3 Motion Tracking using Background Subtraction

The proposed algorithm used in this part aims to estimate moving edge pixels by comparison of two successive masks. These masks are obtained by extracting the background from two images. To do this, we first define a threshold ratio for pixels values. Next, we subtract the previous image from the current one and compare the resulting image to the previous image scaled by the defined threshold value. This would result in a binary mask where pixels that are below the previous image threshold are considered as background and those with greater values are assumed belonging to the foreground. To reduce noise in the mask, we multiply it with the mask generated in the previous run.

3.3.4 Foreground Mask Generation

The previous methods presented above output two binary masks that could be used for foreground detection, however, both of the generated masks present wrong detections. On one hand, the foreground detection sometimes identifies parts of the foreground as background and vice-versa. For example, somber clothes are sometimes identified as background when the scene has bleak lighting whereas, in a bright scene, some parts of the background could be identified as foreground if the person(s) is(are) wearing bright clothes. On the other hand, the mask generated in the third step should now indicate the shape of moving objects in the image. As the objective is to consider moving objects as foreground, we draw the boundary of these shapes then fill their insides with white pixels. This method proves however to be challenging when dealing with an image containing 2 or more persons that are distanced from each other. To circumvent these issues, we combine the two masks obtained through the previous techniques to generate the resulting foreground mask which would be passed on to post-processing.

3.3.5 Post-Processing

The resulting mask from the previous phase may still include some unwanted portions that distort the detection. This may arise due to some changes in the background condition and takes the shape of noise or small holes in the mask. To deal with these issues, post-processing techniques are implemented to remove unwanted portions from the foreground mask and "fill" other undetected areas that originally belong to the foreground of the scene. To remove small noisy blobs, noise filtering algorithms are of extreme importance; this step is extremely important and needs to proceed with other post-processing operations as this noisy detection may badly interfere with future operations. The second half of post-processing implies the changing of shapes of detected blobs to bring them closer to the ground truth. Dilation and erosion are considered to be the most basic morphological operations.

After defining the size and shape of the structuring element, dilation expands objects by adding pixels to the boundaries of detected blobs in the image according to the defined structuring element whereas erosion does exactly the opposite by removing pixels.

3.4 Image Renderer

After segmenting the foreground in images, the function rendering can process an image of the left camera frame using its segmented mask according to a chosen mode given as a string. The argument mask is a matrix of a binary image which have the same size as the frame image. The function rendering can be called using one of these modes:

- **foreground:** When calling the function render using the foreground mode the background is set to black and the foreground in the image will be displayed. Since the foreground pixels are ones and the background pixels are zeros in the matrix "mask", the output is computed by simply performing an element-wise multiplication of the mask and the frame image. By doing so, all background pixels are set to zero and only foreground ones keep their RGB values.
- **background:** When calling the function render using the background mode the foreground is set to black and only the background is displayed. To keep the values of the background pixels and set the foreground pixels to zero, the mask is first inverted so that its element-wise multiplication by the frame image keeps only the RGB background values.
- **overlay:**When calling the function render using the overlay mode foreground and background is displayed with two different colours transparently. For the overlay mode, a copy of the mask is inverted. Each of the masks and the inverted mask matrices is multiplied by two different RGB vectors and summed, then the two 3 dimensional matrices are summed so that the foreground and background are shown with two different colours.
- **substitute:**When calling the function render using the substitute mode the background is replaced with a given virtual background. As the background image should have the same size as the frame image, the virtual background image is resized. To keep the pixel values of the foreground, the frame is element-wise multiplied by the mask matrix, whereas the virtual background is multiplied by the inverted mask. After summing the two RGB images, the existing background is replaced with the virtual background.

4 Graphical User Interface

In this chapter, we will present our graphical user interface. We will also discuss the steps required to run the program and visualize these with the help of screenshots.

This graphical user interface has been created using the Matlab built-in app development framework App Designer. This framework provides user friendly standard components such as buttons, checkboxes, trees, and drop-down lists, that made the usage of our program considerably intuitive.

4.1 Structural Design

While developing the graphical user interface (GUI) to meet the required functions, we tried to keep the interface as simple and intuitive as possible, in order not to overwhelm the user.

Therefore we split the given specifications into 5 steps and 1 optional. The first being dedicated to selecting paths of the source, background and output directories. If these paths are not valid, an error message will be displayed to the user.

The second step is allocated to selecting the configuration parameters variables Left, Right, Start and N. These variables are also subject to a control routine, which verifies their validity.

Before running the program, the user should select a rendering mode in the third step. The user has a toggle button group containing all 4 possible rendering modes. The use of this toggle box will prevent the user from selecting multiple modes at once.

4.2 Interaction Design

After running the program, the stereo input image stream will be displayed on the left side of the console whereas the rendered output stream will be displayed on the right side. To let the user control these streams and be able to play it, pause it or repeat it on loop, a playback control has been developed.

As an additional feature, a video with a virtual background will be played in an extra window with the help of the Matlab built-in video player.

5 Results

5.1 Segmentation

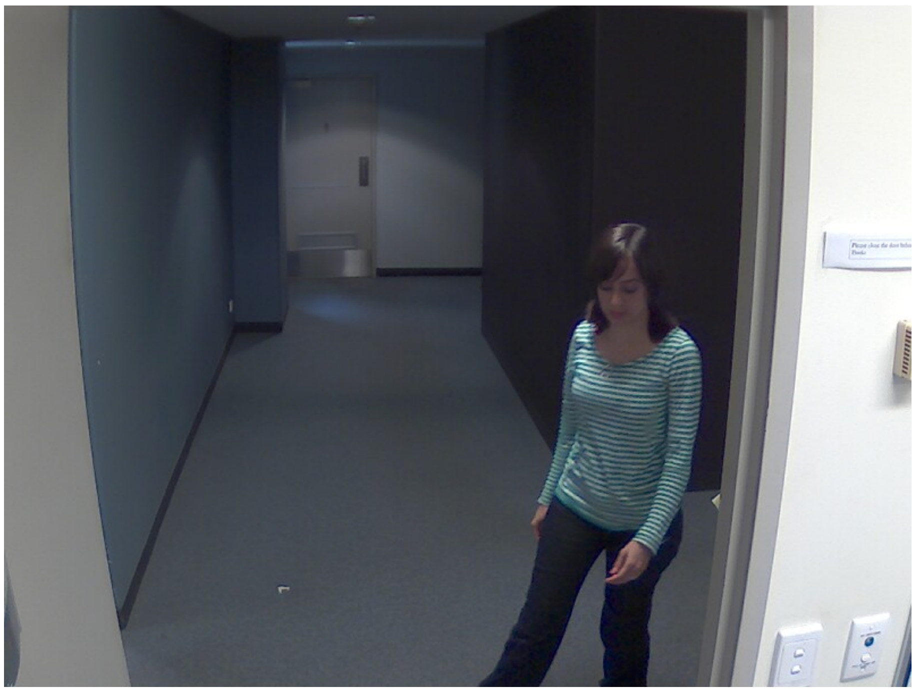


Figure 3: original image from P1ES1

The segmentation using Gaussian Mixture Models has proven to be successful. As we can see in the following illustrations, moving individuals or objects are recognised as being a part of the foreground. The model is also robust to changes in lighting, however, it has flaws that can be improved on.

First, as seen in the figure 11, shadows and reflections of moving objects are considered as being part of the foreground.

Then, due to the model only considering the changes in the variance of a pixel, moving objects with the same colour as the background will not be recognised. The torso of an individual wearing a strapped shirt matching the gray floor will be cut by the mask. This can be seen in the figure 5

Furthermore, as the figure 9 shows, the model is robust when working with a large crowd. Everyone in the foreground will be efficiently distinguished. Finally, slow moving objects will be ignored as well if the change in the variance does not exceed the threshold.

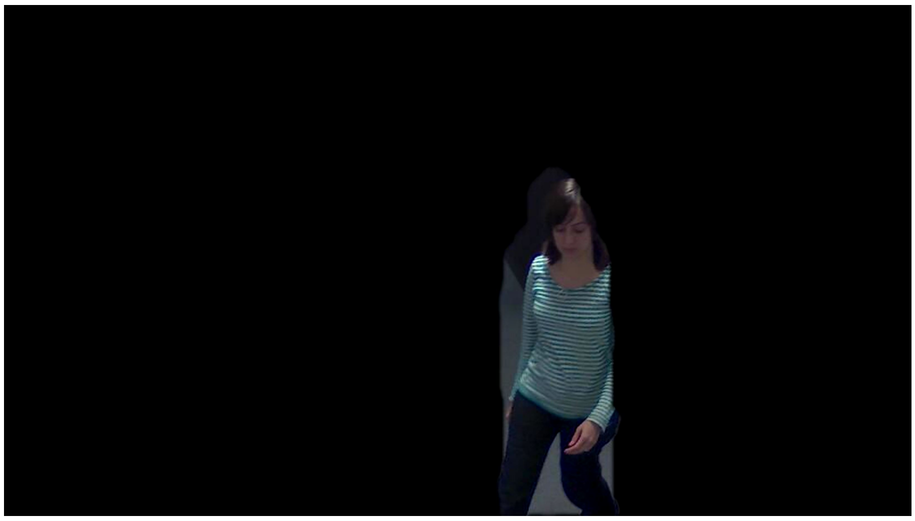


Figure 4: rendered image from P1ES1

5.2 Video

6 Conclusion & Outlook

In conclusion, we have reached our goal and created a functioning foreground detection and segmentation algorithm. Even though our project is successfull, it can still be refined. First of all, Slow moving objects should also be recognised as being part of the foreground. Then, the robustness of the algorithm against shadows and reflections should also be improved. Finally, a new concept should be developed in order to differentiate between background and foreground of the similar colour.

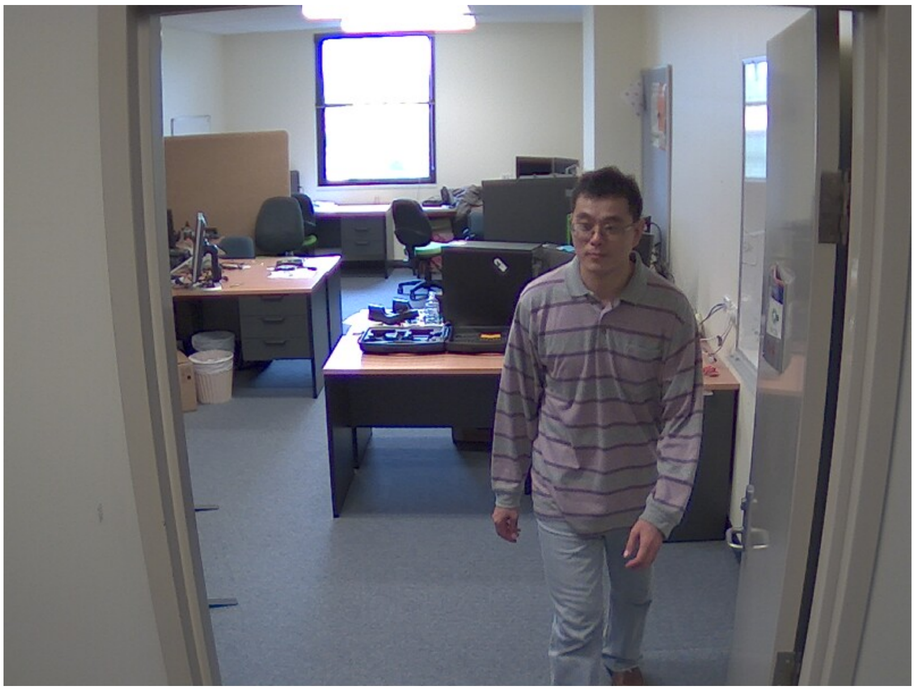


Figure 5: original image from P1LS3

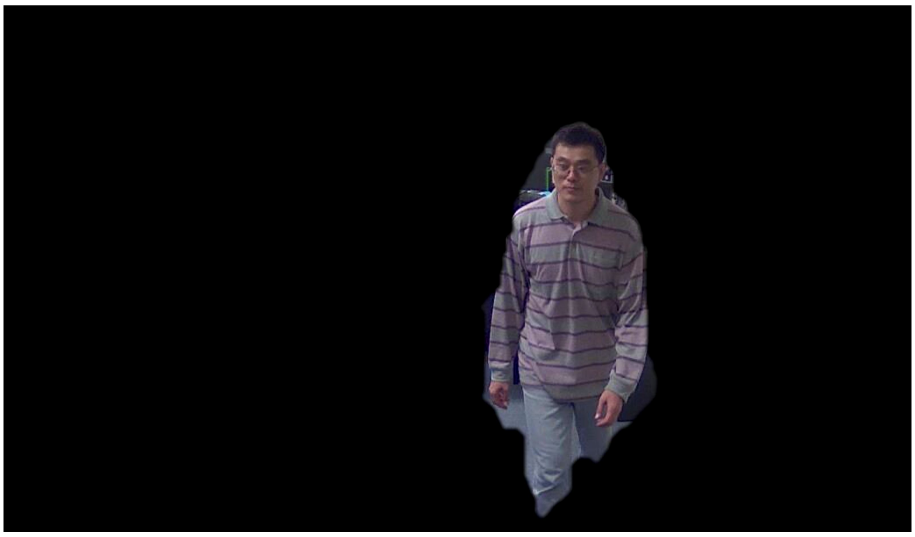


Figure 6: rendered image from P1LS3

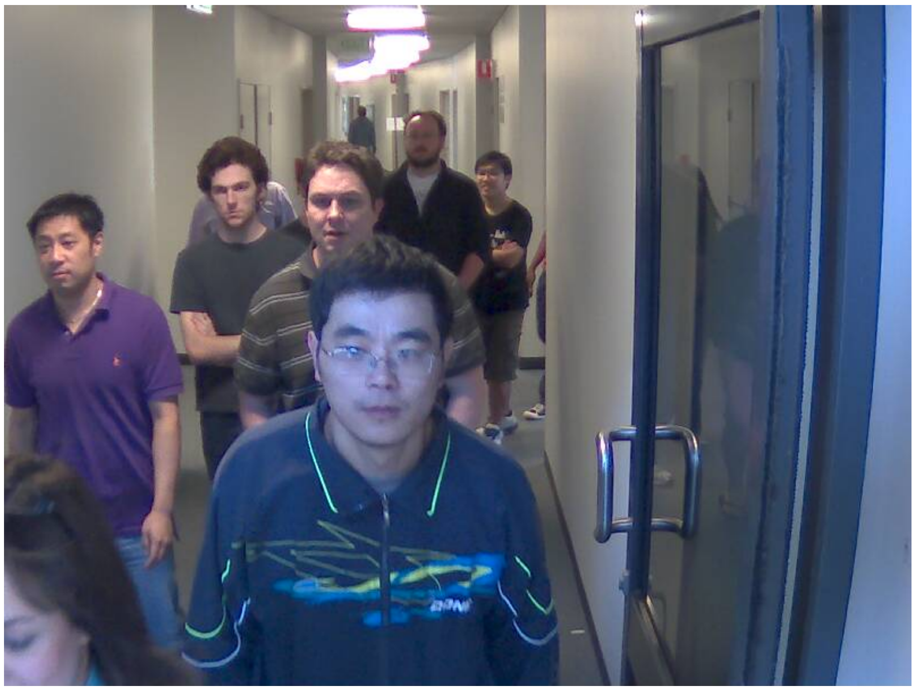


Figure 7: original image from P2LS5

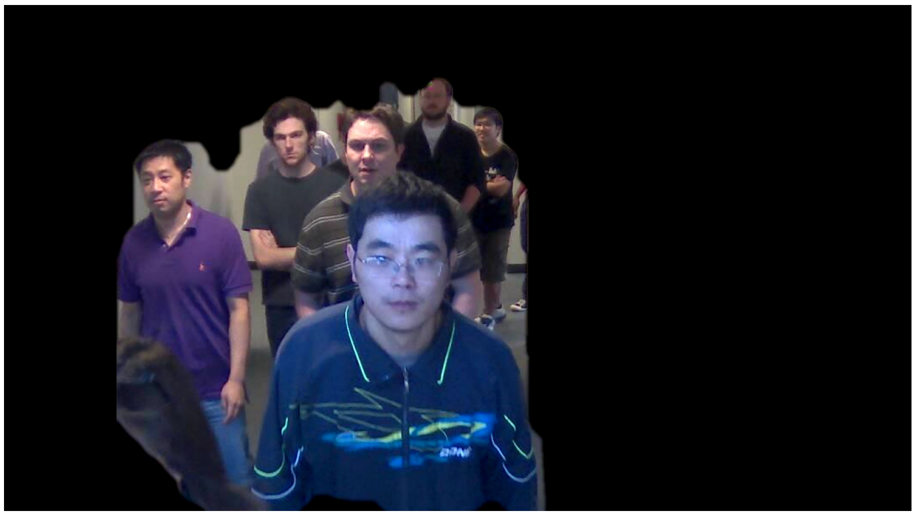


Figure 8: rendered image from P2LS5

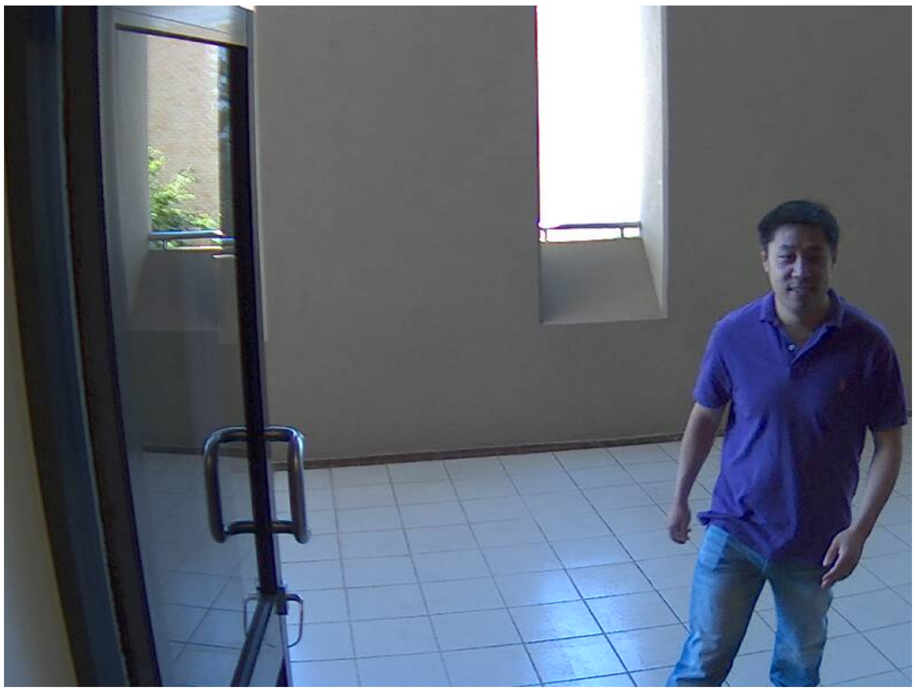


Figure 9: original image from P2LS5

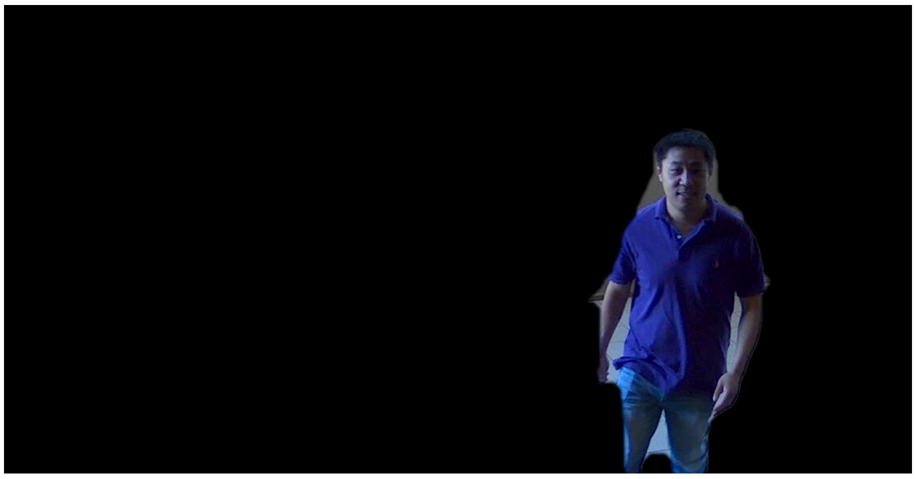


Figure 10: rendered image from P2ES2

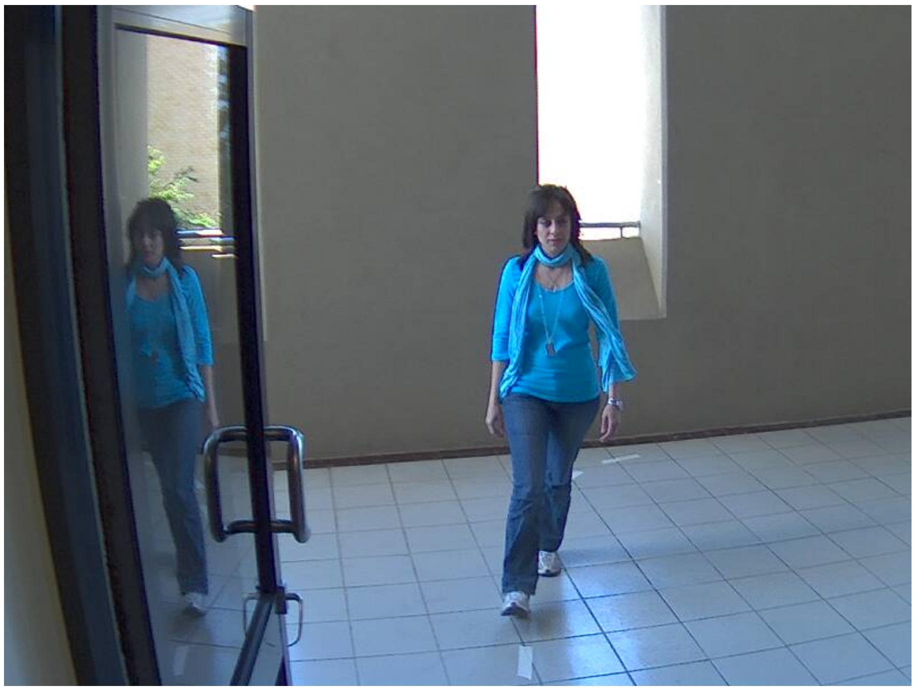


Figure 11: original image from P2ES4

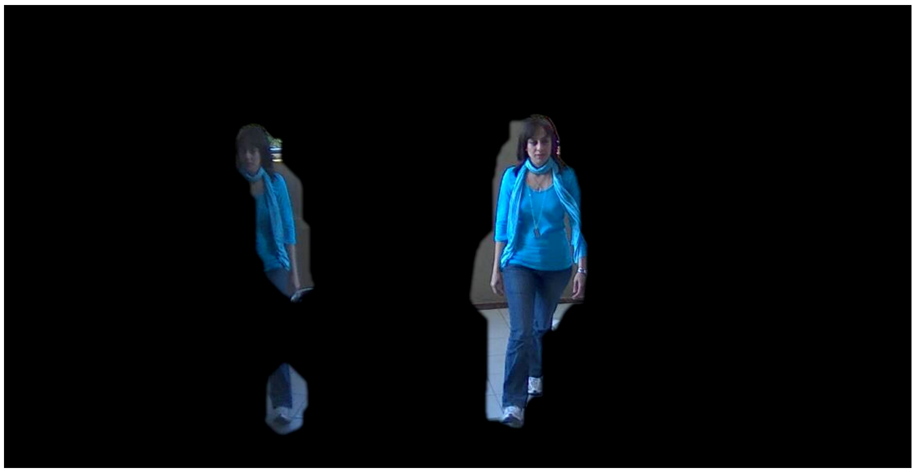


Figure 12: rendered image from P2ES4

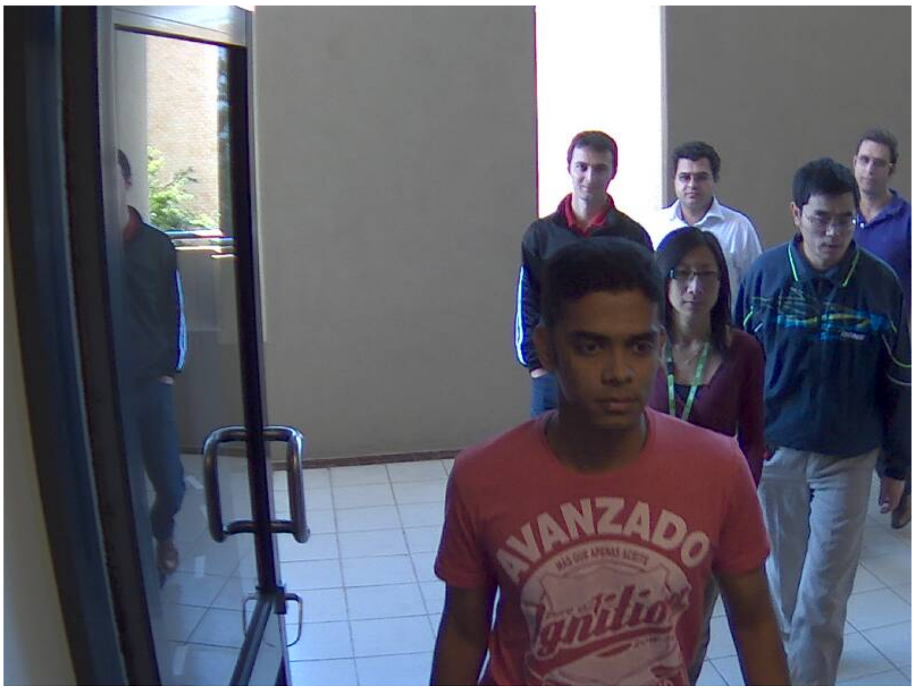


Figure 13: original image from P2ES5

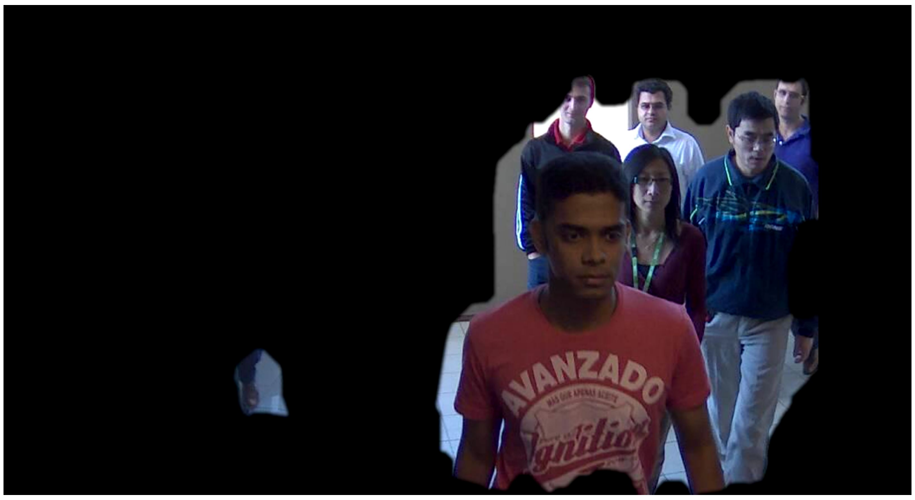


Figure 14: rendered image from P2ES5