AVSA Lab1 Task1

Zsófia Sára BUDAI

I. INTRODUCTION

This report discusses foreground segmentation methods developed in C++ using the OpenCV library on Eclipse. The methods were implemented to work with video sequences captured from still cameras, both on black and white and colored frames. The report covers several different foreground segmentation techniques, including frame difference and progressive update of the background as well as suppression of ghosts. Initially, the task involved becoming acquainted with the Eclipse programming environment on Ubuntu, which included adding libraries, creating a workspace.

II. METHOD

Three consecutive methods for foreground and background segmentation were implemented to process the provided video sequences. Below, a brief overview of each method will be provided.

A. Generation of foreground segmentation mask

The "frame difference" approach was used to generate a foreground segmentation mask for each video frame. This involves calculating the difference between two consecutive frames to detect moving objects in the scene. The resulting difference image is thresholded and with this creates a binary mask that selects the objects that are assigned to the foreground. The first parameter was tau, which was used for thresholding the difference image. The second parameter was a boolean value named "rgb", which determined whether the code should run on colored or grayscale images.

B. Progressive update of background model through selective running average

Selective running average is an advanced method used for foreground segmentation in computer vision, where the background model is progressively updated over time. The method involves selectively computing the running average of the background pixels to update the background model over time. This is done based on a weight factor (alpha) that assigns a higher weight to pixels that are similar to the current background and a lower weight to pixels that are dissimilar. By using selective updating, the method can adapt to changes in the background more efficiently and reduce ghost effects which are appearing often with the first implementation.

C. Suppression of stationary objects that appear or are removed

This method involves identifying and removing objects that remain stationary in a video sequence. It can be achieved by analyzing the difference between consecutive frames in the video and identifying the pixels or regions that have not changed significantly. Objects that remain stationary over multiple frames are suppressed, effectively removing them from the sequence.

III. IMPLEMENTATION

A. Functions, variables

For computing the absolute difference between the background and frame pixels I used the absdiff OpenCV function. To process every pixel in a frame, a nested double for loop was utilized. In the case of working with color frames, the values of each pixel were represented as a cv::Vec3b data type, which required an additional for loop to extract the red, green, and blue values (r, g, and b) of each pixel. During the testing of the elective running average, the background mask was examined, and if the pixel was not belonging to the foreground mask, then the pixel was added to the background. The newly introduced variable were _alpha that determines adaptation speed, selective_bkg_update boolean variable that is responsible for the implementation of the selective running average. The _threshold_ghosts2 and the counter, that incorporate a pixel to the background if the counter reaches a certain threshold.

B. Running the code

The submitted files adhere to the format specified for the assignment, comprising the requisite source C++ code and corresponding Makefiles for each method. I created a new project for each method, therefore there are three source code and three Makefiles. To compile, link, and execute each task, simply navigate to the source folder for the corresponding task and run the "make" command in the terminal. This will generate the necessary built object file, which can be executed by running "./main".

IV. DATA

The algorithms underwent testing on five video sequences that varied in the number of frames they contained:

- hall.avi
- empty_office.avi
- stationary_objects.avi
- eps_hotstart.avi
- eps_shadows.avi

The videos are featuring complex situations, including objects with similar brightness and color levels as the background, displaced objects from the initial background and adding new objects to the background, changes in brightness during a stationary video, slow and fast movements, and extreme occlusion scenarios. Analyzing these videos in detail helped identify the strengths and weaknesses of simple segmentation algorithms.

V. RESULTS AND ANALYSIS

The basic generation of foreground segmentation mask algorithms performance highly depended on the selected threshold value as the 1 figures illustrate it. In most cases, a threshold value of 25 was used. This value was chosen as a compromise between sensitivity and noise.

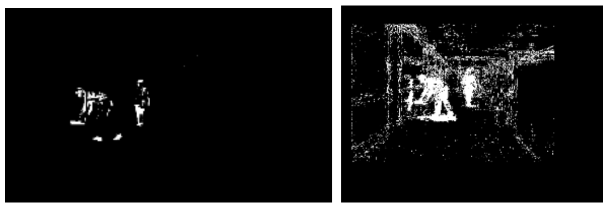
Fig. 1: On the left hight threshold (100) therefore little noise but low sensitivity. On the right low threshold (10) therefore big noise and high sensitivity.

The process applied to the RGB color channels reduces camouflage in the foreground segmentation. By analyzing the color information in each channel, the segmentation algorithm can distinguish between foreground and background pixels more accurately. As a result, the camouflage effect, where foreground objects blend in with the background due to similar colors, is reduced, shown in 2 figures.

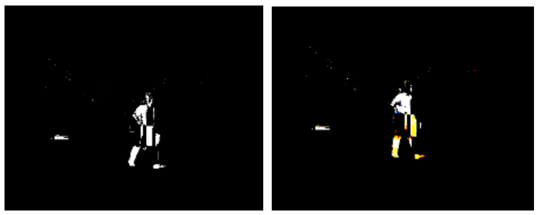
Fig. 2: Camouflage on gray (left) and on colored (right) image

Although the basic method mentioned earlier has its advantages, it also has limitations. Easily noticeable limitations are the so-called ghost effects, where moving objects leave behind residual traces or shadows in the background, or small, progressive variations of background are not supported. These ghost effects can be mistaken for foreground objects by the segmentation algorithm, leading to inaccurate results as 3 figures illustrates.

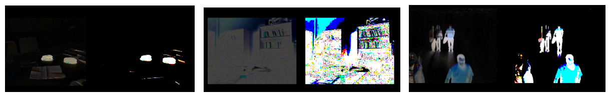
Fig. 3: Ghost effects

Therefore, more advanced methods are needed to overcome these limitations and improve the accuracy of foreground segmentation. Implementing the progressive update of background model through selective running average helped to get rid of the ghost and made the selections more accurate. The background is calculated from the given equation: $BKG_{t+1}[x,y] = \alpha Image_t[x,y] + (1-\alpha)BKG_t[x,y]$. Where the adaptation speed is determined by the value of α. The 4 figures illustrates how by increasing the value of alpha results in a faster adaptation speed, causing moving objects to disappear more quickly.

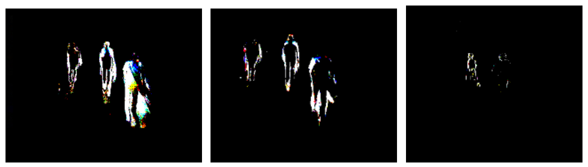
Fig. 4: Changing alpha parameter 0.05, 0.15, 0.8 from left to right

To suppress ghosts and stationary objects in the video sequence, a third method was implemented. In the "init_bkg" function, a counter variable was initialized with the size of the image so that every pixel has a corresponding field in the counter. The counter was increased by one every time the corresponding pixel was detected as foreground, and reset every time it was detected as background in current frame. If the value of a pixel's corresponding counter field reaches the threshold, that pixel is classified as a background pixel. The result shown in the 5 figures. While the third method may have produced better results than the previous methods, ghosts may still be detected by the human eye.

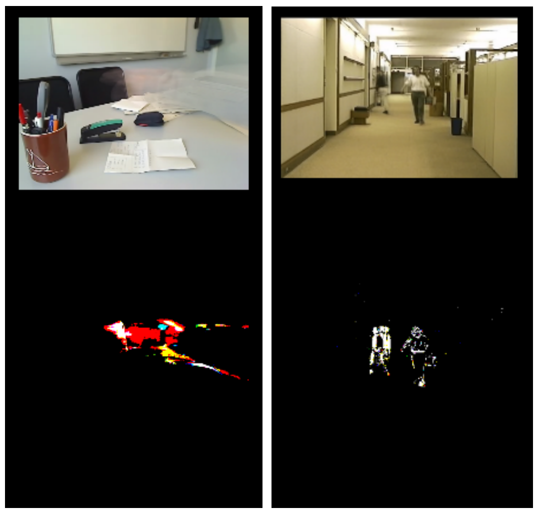
Fig. 5: Visualizing stationary objects that appear or are removed

VI. CONCLUSIONS

Through this project, I gained hands-on experience in implementing and practicing various foreground segmentation techniques, deepening my understanding of them allowed me to discover their limitations, advantages, and disadvantages. While the methods used in this project were relatively simple to implement and had a low computational cost, it became apparent that their lack of generalizability and robustness was due to the specific parameter configuration utilized. Additionally, I was able to gain a better understanding of using the C++ language in the Eclipse environment and became more familiar with the OpenCV library.

VII. TIME LOG

Approximately 31 hours were dedicated to this project. The time spent on each part was divided as follows:

- Studying: 3 hours
- Virtual Machine is not running on my computer due to storage shortage therefore I needed to boot into Linux from a USB drive: 15 hours
- Coding, debugging code : 4 hours
- Testing, screenshots: 3 hours
- Report: 6 hours