Lab 2: Object Detection and Classification

Kush Gupta and Sergio Avello

1 Introduction

This lab report contains blob extraction and classification routines, as well as a routine to detect stationary foreground pixels in video sequences. The code is written in C++ using the OpenCV library and designed on Ubuntu using Eclipse IDE. All these tasks are performed on 8 videos captured from a still camera that belong to the 2020AVSALab3 dataset. Our algorithm relies on a foreground mask obtained for each frame with the MOG2 background subtraction module provided by OpenCV. The routines developed for this lab were, first, extracting blobs using the foreground segmentation mask and then classifying them as person, car or object. Finally, we develop a slightly more complex method for extracting stationary foreground pixels based on the paper "Stationary foreground detection for video-surveillance based on foreground and motion history images". We also took advantage of a function to paint the bounding boxes containing the extracted blobs and the classified ones with identifiable colors.

2 Blob Extraction

For the provided videos, we firstly use the Opencv's MOG2 background subtraction method to segment the foreground and the background. The MOG2 method performs background subtraction using the approach of gaussian mixture model (GMM) to provide a foreground mask ($fgmask$) where the values 255, 127 and 0 correspond to foreground, shadow and background, respectively. Then in the extractBlobs function we use this foreground mask and the OpenCV's built-in function floodFill in order to apply the sequential Grass-Fire algorithm [2] for each of the foreground pixels. This algorithm goes through the whole input foreground mask from the top left corner to the bottom right corner and acts when a foreground (255) pixel is found: it firstly labels the pixel in the output image and secondly sets to zero that same pixel in the input image. Then, the neighbours of this "burnt" pixel are checked (4 or 8, depending on the parameters given for the connected components analysis), and any of those neighboring pixels being also foreground pixels will be treated the same way (they are labeled in the output image and burnt in the input image), and so on. When no more neighboring foreground pixels can be found, the label is incremented and the normal scanning (from top left to bottom right) of the new modified foreground mask resumes, until the next foreground pixel is encountered, labeled, burnt, and its neighbours are investigated. The glass-fire algorithm results in coordinates of the left corner and the width and length of all the blobs containing the objects existing in the frame. Moreover, we implemented the removeSmallBlobs function to remove all blobs whose areas were smaller than 400 pixels (this function is controlled by the MINWIDTH=20 and MINLENGTH=20 parameters).

3 Blob Classification

After obtaining the foreground segmentation mask using the MOG2 method and extracting the blobs from it, we next have to classify the obtained blobs in 3 different classes: PERSON, CAR and OBJECT. To implement the classification method we compared the aspect ratio feature of each blob with predefined models in order to determine which one resemble the most to that current aspect ratio and therefore assign the corresponding class to that blob. The aspect ratio feature can be defined as:

$$aspectratio = blob.width/blob.height \tag{1}$$

Also, we used the paintBlobImage function which paints a blob with different colors depending on its class: blue for PERSON, green for CAR and red for OBJECT.

4 Stationary foreground extraction

This routine for extracting stationary foreground is based on simplified version of the algorithm proposed in "Stationary foreground detection for video-surveillance based on foreground and motion history images" [1].

Firstly, for every frame, we initialize a Foreground History Image (FHI). Then, we check the given foreground mask pixel by pixel, if the pixel is foreground, we update the FHI following the equation 2 given in the Figure 1 (we used $w_(pos)^f = 1$), if the pixel is background, we update the FHI following the equation 3 given in the Figure 1 (we used $w_(neg)^f = 15$). After updating all the pixels of FHI, we normalize it in order to have values between 0 and 1. Finally, we compare it pixel by pixel with a threshold (in our implementation 0.8). Only if the pixel value is larger than the threshold it is identified as a stationary pixel.

Regarding the configuration of our implementation, we realised that the main parameter that strongly affected our algorithm's performance was the update rate of the foreground model. We had to choose this parameter cautiously in order to give enough time for the stationary foreground objects to create enough history information enabling the algorithm to detect them. At the end we selected the learning rate as 0.001 because we got our best results with that value.

$$FHI_t(\mathbf{x}) = FHI_{t-1}(\mathbf{x}) + w_{pos}^f \cdot FG_t(\mathbf{x}), \qquad (2)$$

$$FHI_t(\mathbf{x}) = FHI_{t-1}(\mathbf{x}) - w_{neg}^f \cdot (\sim FG_t(\mathbf{x})), \qquad (3)$$

Figure 1: Equations for updating the Foreground History Image.

5 Implementation

In the submission zip file there is only one folder with the corresponding sources files. The three tasks have been implemented in the same blobs.cpp and the original version of this code runs the three tasks consecutively as they are related to each other. However, if you want to check the different tasks independently (the results of these individual tasks are presented on the Results and Analysis section), you have to follow the instructions that are commented in the blobs.cpp file in order to comment/uncomment out certain parts of the functions. On the other hand, the Lab2.0AVSA2020.cpp file acts as the main file of the project, in which you can modify the main parameters that control it, such as the videos used, MINWIDTH, MINHEIGHT, connectivity, the learning rate of the createBackgroundSubtractorMOG2...

5.1 Running the code

In order to compile, link and run the project, it is only needed to run make in the terminal in the src folder and then run ./(built object).

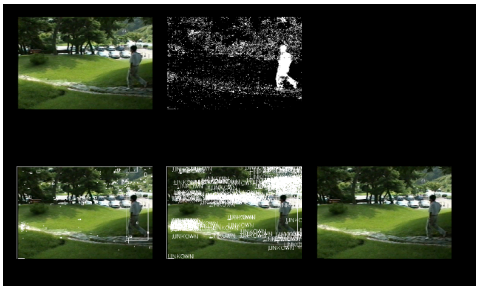
6 Results and Analysis

In this section we will discuss, analyze and compare the results obtained for each task with the different parameters used.

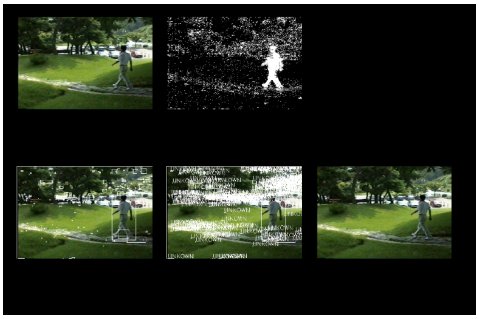
6.1 Task 1 (Blob Extraction)

The results for this task are obtained with a learning rate of 0.001 (instead of -1) because after some research we found out that for task 3 the best performance was achieved with it. In addition, this task only uses the code in the extractBlobs and removeSmallBlobs (if you want to check this task idividually, follow the instructions in capital letters about commenting/uncommenting on the the remaining functions of the file blobs.cpp). The main paramater that controls this algorithm is

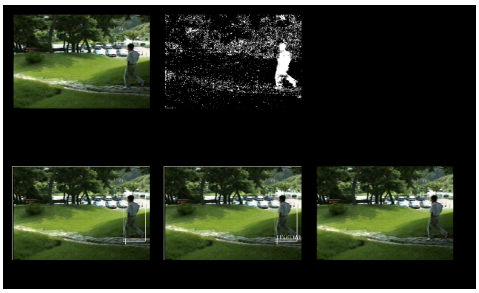
the connectivity, however, as we can see on the image pairs a,b and c,d on Figure 1 it is difficult to find remarkable differences between using 4 or 8 connectivity. It is also important to mention that the image pair a,b of Figure 1 corresponds only to the implementation of the Fire-Grass algorithm (without removing the small blobs), that is why there are many irrelevant blobs. On the other hand, the image pair c,d shows the blobs obtained after removing the small ones in the removeSmallBlobs function. 2



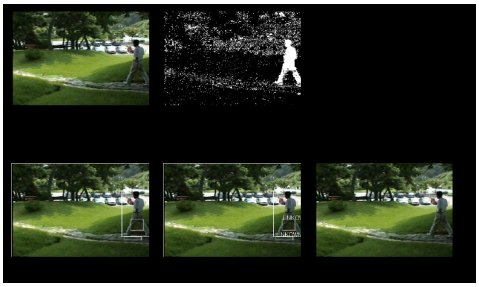
(a) Without removing small blobs, Using 4 connected component



(b) Without removing small blobs, Using 8 connected component



(c) After removing small blobs, Using 4 connected component



(d) After removing small blobs, Using 8 connected component

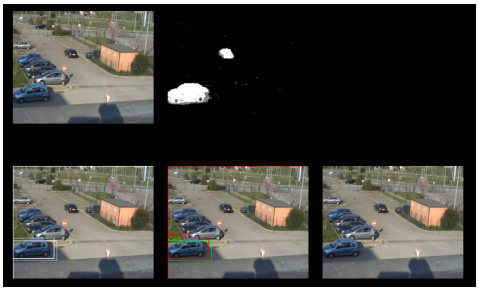
Figure 2: Results obtained using 4 and 8 components in the Glass fire algorithm.

6.2 Task 2 (Blob Classification)

The second task takes advantage of the output of the previous one and then applies a very simple blob classificator on the classifyBlobs function (if you want to check this task idividually, follow the instructions in capital letters about commenting/uncommenting on the the function extractStationaryFG). As it was previously explained, the classification is based on the aspect ratio of each blob, due to its trivial complexity, the results were not very accurate. Nevertheless, it still managed to detect objects correctly as it can be seen on Figure 3. 3

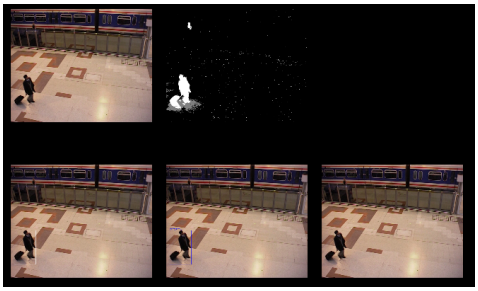
6.3 Task 3 (Stationary Foreground extraction)

Task 3 is performed on the extractStationaryFG function, using the previous results of Tasks 1 and 2 (to check this task run the code directly after downloading the code). After visualizing the results of this algorithm with different parameter combinations, we decided to fixed them to: FPS = 25, SECSSTATIONARY = 2, ICOST = 1, DCOST = 15, STATTH = 0.8 because they outperformed other configurations. Besides that, this algorithm is strongly affected by the learning rate of the MOG2 background subtraction module, because it controls how fast the background model is updated. We tried several values (as it is shown in Figure 4) and finally we achieved the best results with a learning rate of 0.001. 4



(a) Green blob means a car detection

(b) Blue blob means a person detection and red blob means an object detection



(c) Blue blob means a person detection

Figure 3: Task 2 results with different object detections

7 Conclusion

In conclusion, we observed that overall our implemented blob extraction and classification routines displayed fairly reasonable results (besides its lack of complexity) on the different videos that we tried. While working on this project we realised that an accurate foreground mask is one of the most important aspects in order to process the videos for further surveillance analysis. Using the foreground mask provided by the OpenCV MOG2 background subtraction module was one of the main reasons why we achieved such good results. Also, we found the task of stationary foreground extraction really enjoyable as we learnt another way to enhance the quality of the foreground mask.

If we have had more time, it could have been interesting to explore other different classification methods and to implement them in the code for more accurate results. Moreover, to make the model more robust we could have also added more classes to it.

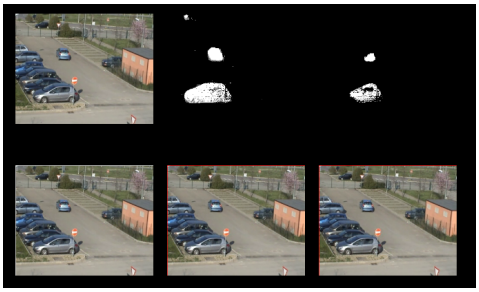
8 Time-Log

Coding: Around 8 hours for coding and debugging the errors.
Testing: At least 6 hours selecting relevant video sequences, allowing the program to process them and then analyzing the behaviour of our methods.
Report: It took us approximately 7 hours to write the report, take the screenshots of relevant frames,adding results and formatting them. Proof reading the report to eliminate the possible typo errors.

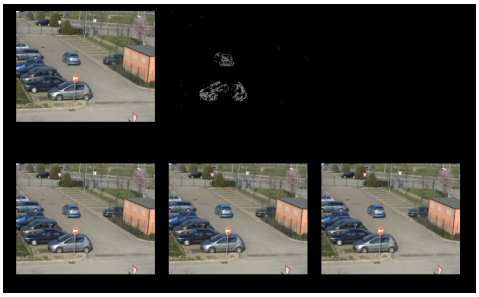
References

[1] Á. Bayona, J. C. SanMiguel and J. M. Martínez, "Stationary foreground detection using background subtraction and temporal difference in video surveillance," 2010 IEEE International Conference on Image Processing, 2010, pp. 4657-4660, doi: 10.1109/ICIP.2010.5650699.

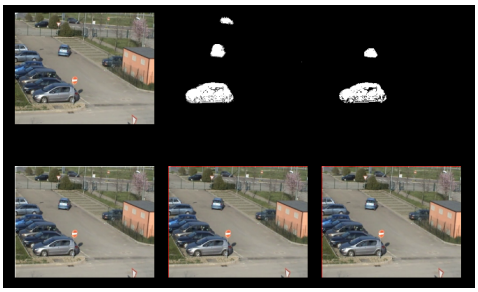
[2] Jose M. Martinez Sanchez, Unit 2: Foreground segmentation and object segmentation,Applied Video Sequence Analysis.



(a) Stationary foreground extraction using learning rate=-1. This learning rate updates almost perfectly the foreground model for each frame so it is really difficult to identify the stationary objects



(b) Stationary foreground extraction using learning rate=0.5. This foreground model mostly produces shadows and since task 3 relies only on the foreground pixels, the algorithm does not work correctly



(c) Stationary foreground extraction using alpha=0.001. This configuration performs really adequately, it is able to identify cars that are parked as stationary objects and not the car that is moving on the top.

Figure 4: Task 3 results obtained for the stationary foreground extraction using different learning rates.

[3] A. Cavallaro, O. Steiger and T. Ebrahimi, "Semantic video analysis for adaptive content delivery and automatic description," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 15, no. 10, pp. 1200-1209, Oct. 2005, doi: 10.1109/TCSVT.2005.854240.

[4] OpenCV reference manual. http://docs.opencv.org/2.4.13.2/modules/refman.html.