Object detection and classification

María Fernanda Herrera and David Savary

I. INTRODUCTION

The objective of this report is to document the implementation and results of blob extraction and classification routines, as well as stationary foreground detection routines. The blob extraction was implemented using the *Grass-Fire* algorithm and a simple size based filter. The implemented blob classification routine is based on the aspect ratio of the extracted blobs and a statistical Gaussian pre-trained classifier based on mean and variance. Lastly, the static foreground extractor is based on a foreground motion history image that is able to detect stationary blobs. All these routines were tested and evaluated on the provided datasets for the second laboratory programming task.

II. METHOD AND IMPLEMENTATION

All three routines, blob extraction, blob classification and stationary foreground were implemented in one or more functions within the "*blobs.cpp*" file.

A. BLOB EXTRACTION

The blob extraction is performed in two steps. First, the "*extractBlobs*" step, which is the one that does the actual blob extraction. Once the list of blobs is extracted from the foreground mask, all the small blobs are filtered out in the "*removeSmallBlobs*" function.

1) "extractBlobs": This function creates a list of blobs from a foreground mask according to a connectivity parameter (which has to be either 4 or 8). The function checks every pixel of the foreground mask, and calls the OpenCV "*floodFill*" function for every non marked foreground pixel. Every time the "*floodFill*" function is called, a new blob is added to the blob list, where the blob is defined as the initial pixel and the pixels connected to the initial pixel. Finally, all the pixels within the new blob are marked.

2) "removeSmallBlobs": This functions creates a new list by filtering all of the blobs with dimensions smaller than the given parameters from the original list of extracted blobs.

B. BLOB CLASSIFICATION

The "*classifyBlobs*" function goes through a blob list and tags each blob in the list with a class according to its similarity to a different predefined models. The models for each category are defined as a Gaussian model by considering the mean and standard deviation of the aspect ratio for each category: person, car or object. The classifier consists in a simple classifier that computes the weighted distance between the aspect ratio of a blob and each one of the pretrained models. The function assigns to each blob the class for which the weighted distance between the blob aspect ratio and the class model is the smallest.

C. STATIONARY FOREGROUND DETECTION

The stationary foreground detector is based on [1] and was implemented in the "*extractStationaryFG*" function. This function implements the equations needed to calculate the foreground history image described in [1], as well as the equations that update the foreground mask with the information of the foreground history image. All of the operations are implemented with OpenCV functions.

III. RUNNING THE CODE

To run the implemented code, it is necessary to compile the code in a Linux machine with OpenCV installed by using the Makefile. The *make* command will generate a "*Lab2.0AVSA2020*" executable that can be run without any argument. Alternatively, the *make run* command will compile and run the code. The path of the videos can be changed inside the "*Lab2.0AVSA2020.cpp*" file. Some parameters can also be adjusted in the "*blobs.cpp*" file, such as "*SECS_STATIONARY*", "*I_COST*", "*D_COST*" or "*STAT_TH*", that are used in the "*extractStationaryFG*" function. The parameter "*learning rate*" in the "*Lab2.0AVSA2020*" was also adjusted during testing.

IV. DATA

The algorithms were tested with several video sequences from the dataset provided for this second programming task. These datasets are suitable to test the developed routines, as they contain moving persons and cars, as well as multiple objects that remain static to test the static foreground detector.

V. RESULTS AND ANALYSIS

In order to evaluate the performance of the developed code, each part was tested with multiple video sequences. In order to tune the parameters, only two sequences were selected for each part. This allowed to find optimal parameters in multiple situations that represent the problem that each part of the developed code is trying to solve.

A. BLOB EXTRACTION

Two different video-sequences are shown below for the two stages of the blob extraction. The first video-sequence correspond to the "*VISOR/visor_Video00.avi*" video, while the second video-sequence correspond to the "*PETS2006/PETS2006_S4/PETS2006_S4_C3.avi*" video. A sample of the result for the "*extractBlobs*" stage for the first video-sequence is shown in Fig 1, where many white boxes appear due to the fact that also noise is detected as separate blobs. The same effect can be seen in Fig 2, which also

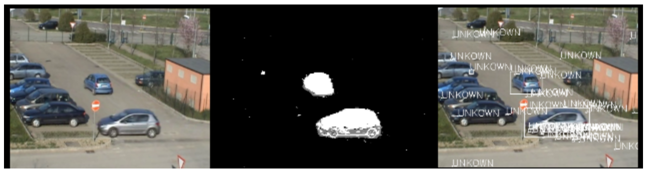

Fig. 1. Extract blob stage result for "*VISOR/visor_Video00.avi*" video-sequence. First image correspond to the current frame, second image correspond to the foreground mask and third image correspond to the detected blobs from left to right.

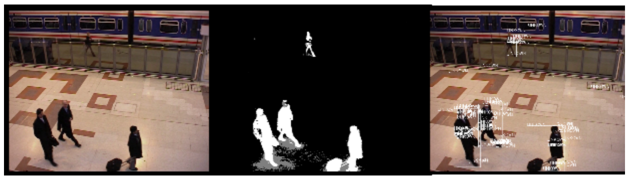

Fig. 2. Extract blob stage result for "*PETS2006/PETS2006_S4/PETS2006_S4_C3.avi*" video-sequence. First image correspond to the current frame, second image correspond to the foreground mask and third image correspond to the detected blobs from left to right.

correspond to the "*extractBlobs*" stage, but for the second video-sequence. The result for the "*removeSmallBlobs*" stage is shown in Fig 3 for the first video-sequence and in Fig 4 for the second video-sequence, where small blobs are filtered.

Minimum width and height used to filter small blobs depends on the size of the objects of interest. It is important to consider that a scene may contain a high variety of blob sizes for different type of objects, where the size is also influenced by the desired range of detection to be covered by the camera since objects that are far away from the camera look smaller. For these reasons it may be better to keep low the thresholds. On the other hand, low thresholds cause blobs coming from noise to also be recovered. Given this

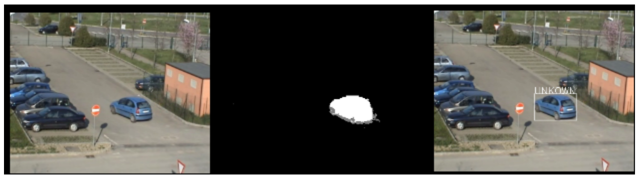

Fig. 3. Remove small blobs stage result for "*VISOR/visor_Video00.avi*" video-sequence. First image correspond to the current frame, second image correspond to the foreground mask and third image correspond to the detected blobs from left to right.

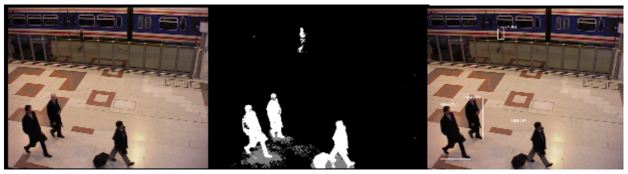

Fig. 4. Remove small blobs stage result for "*PETS2006/PETS2006_S4/PETS2006_S4_C3.avi*" video-sequence. First image correspond to the current frame, second image correspond to the foreground mask and third image correspond to the detected blobs from left to right.

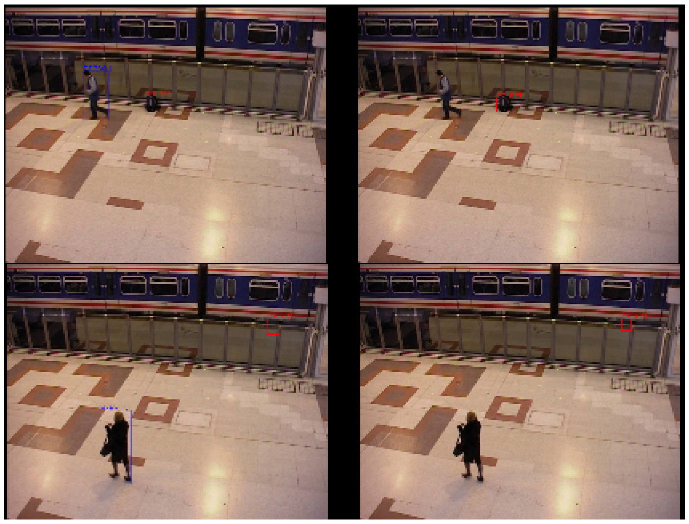

Fig. 5. Remove small blob result for blob stage result for "*PETS2006/PETS2006_S4/PETS2006_S4_C3.avi*" video-sequence. The first row recovers a small object while the second row recovers a noise blob of similar size for both foreground and static foreground from left to right.

situation, size thresholds were set to 20 for both width and height. Besides, the same thresholds were set for removing small blobs coming from static foreground to allow the same type of objects to be recovered. A comparison of the effect of threshold selection for both foreground and static foreground is shown in Fig 5 where the first row correspond to a small object that should be recovered and the second row correspond to a blob coming from noise of similar size, also first column is coming from foreground, while second is coming from static foreground.

B. BLOB CLASSIFICATION

The previously considered video-sequences are also used to show the behavior of the "*classifyBlobs*" function. A sample result for the for the first video-sequence is shown in Fig 6 and in Fig 8 for the second video-sequence. An additional sample result is shown for the first video-sequence in Fig 7. The blob classification is represented by colors, where blue blobs correspond to person category, green blobs to car category and red blobs to object category.

Fig 6 and Fig 7 allows to visualize two different results of car classification. In the first case, the car is correctly classified due to the fact that the orientation of the car allows the blob to have the proper aspect ratio, according to the model. However, for the second case, the car was not classified as a car and was classified as another object instead. This mistake was common in car detection for the simple statistical aspect ratio classifier due to the similarity of the Gaussian models defining a car and an object.

Fig 8 allows to visualize the results of person classification. The scene contains three people, where two of them are correctly classified as person and one of them is classified as an object. In general, the classifier works properly for person blobs, except for the case when a person is detected along with an object that changes the shape of its blob, such as

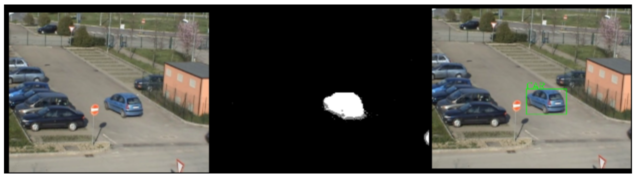

Fig. 6. Classify blobs stage result for "*VISOR/visor_Video00.avi*" video-sequence. First image correspond to the current frame, second image correspond to the foreground mask and third image correspond to the detected blobs from left to right.

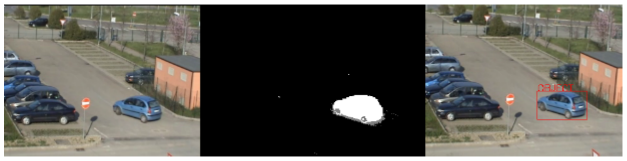

Fig. 7. Classify blobs stage result for "*VISOR/visor_Video00.avi*" video-sequence. First image correspond to the current frame, second image correspond to the foreground mask and third image correspond to the detected blobs from left to right.

luggage, in this case the person may be wrongly classified.

C. *STATIONARY FOREGROUND DETECTION*

The stationary foreground detector was tuned for the "*VISOR/visor_Video00.avi*" and "*PETS2006/PETS2006_S4/PETS2006_S4_C3.avi*" videos. The best parameter values found were 0.0003 for "*learningrate*", 25 for "*FPS*", 5 for "*SECS_STATIONARY*", 1 for "*I_COST*", 4 for "*D_COST*" and 0.4 for "*STAT_TH*". These parameters work well generally, not only on the target sequences, but in all of those in which this learning rate does not produce excessive noise. In the "*VISOR/visor_Video00.avi*" video, the cars are perfectly shown as static foreground when they come to a stop, as shown in Fig 9. In the "*PETS2006/PETS2006_S4/PETS2006_S4_C3.avi*" video, static people is detected as static background after a little while as shown in Fig 10. Although the result is good, there are some problems with big objects, slow moving objects and hotstarts. The problem with big or slow moving objects is that they are detected as static foreground as shown in Fig 11. The "*SECS_STATIONARY*" and "*STAT_TH*" values could be lowered to deal with these type of ghosts, but the problem then would be that there is not enough time to

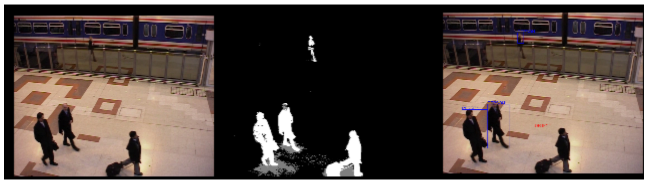

Fig. 8. Classify blobs stage result for "*PETS2006/PETS2006_S4/PETS2006_S4_C3.avi*" video-sequence. First image correspond to the current frame, second image correspond to the foreground mask and third image correspond to the detected blobs from left to right.

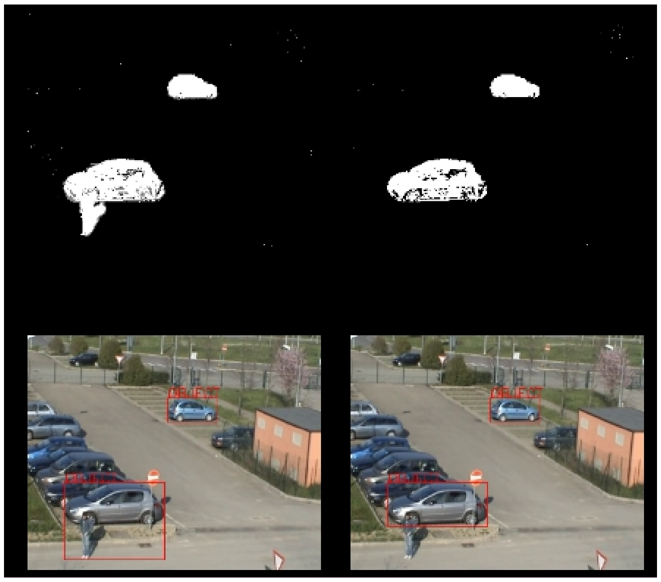

Fig. 9. The two images to the left are showing foreground data, while the two images to the right just show static foreground data. In the left column, the person is not shown as static foreground, as it is moving.

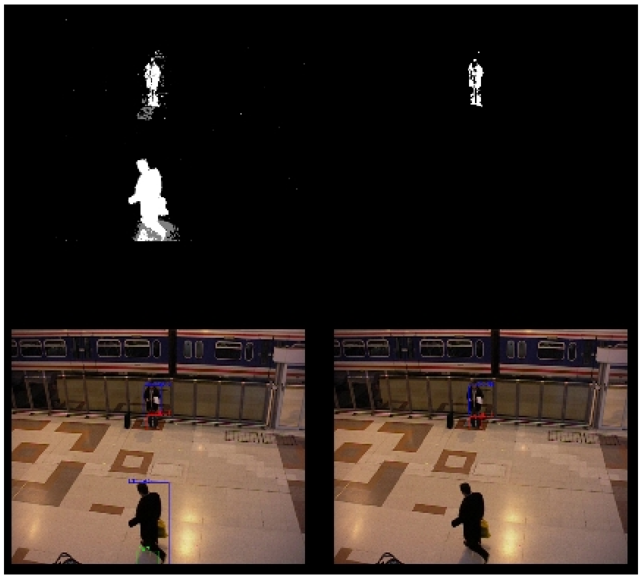

Fig. 10. Only the static people is shown and detected in the static background mask.

detect static foreground pixels before they are incorporated to the background. Regarding the hotstart problem, all of the pixels detected as foreground due to a hotstart will also be detected as static foreground until they are incorporated to the background model. As shown in Fig 12, these pixels will stay in the static foreground mask for a while even when they have already disappeared in the foreground mask. As mentioned before, the selected learning rate has also problems to adapt to highly noisy environments, resulting in noise also being introduced in the static foreground mask. This mainly happens in outdoor scenes with complex backgrounds.

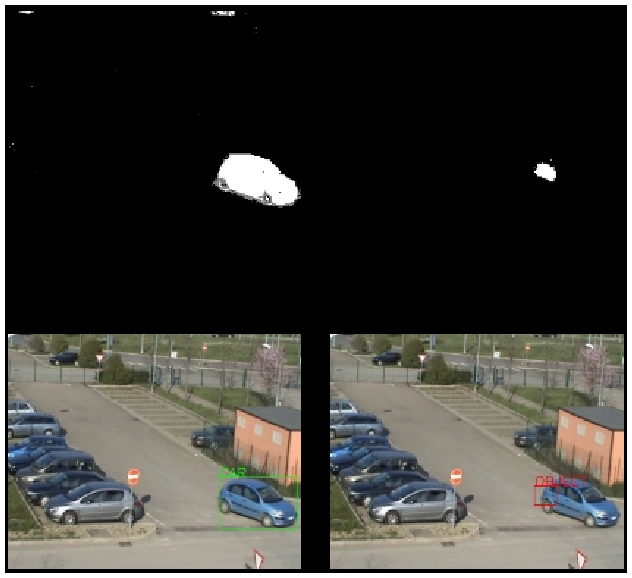

Fig. 11. Although the car is moving, it is moving slow enough to be picked as static foreground by the algorithm.

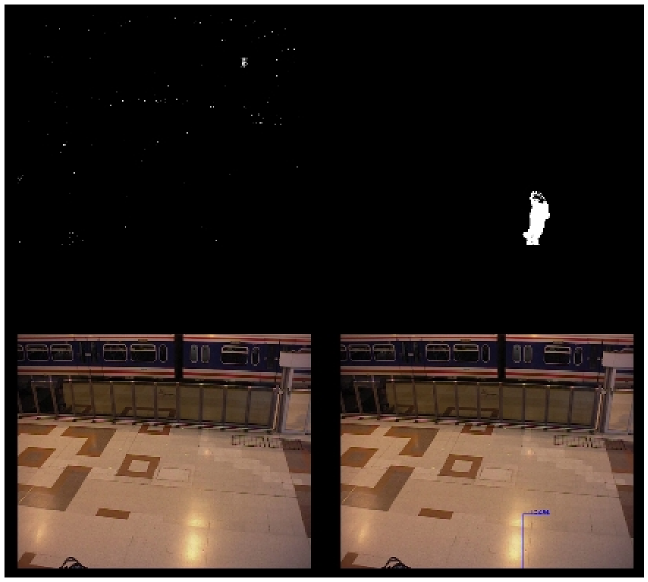

Fig. 12. Ghosts due to hotstarts stay longer in the static foreground mask that in the foreground mask.

VI. CONCLUSIONS

Blob extraction was implemented according to *Grass-Fire* algorithm. Its implementation allowed to recover different blobs for different objects according to pixel connectivity from a foreground mask and then filtering blobs with small size. The approach demonstrates to obtain a proper blob extraction when the parameters for foreground mask creation allow to recover a mask without many noise disturbances and when one object does not intersect with another object, which would create a single and bigger object. Besides, blob extraction also depends on blob size, where small blob removal is determined by size thresholds, and the selection of these thresholds come from a balance among the size of objects of interest and the size of blobs coming from foreground mask noise. Blob classification was performed according to the distance between aspect ratio of a blob and aspect ratio Gaussian model of each category. The simplicity of the classification method lead to miss classifications when the orientation of the object alters the aspect ratio of the blob, for example, when cars are not seen from a lateral view. The method also lead to inaccuracies when an object intersects with another object, which also alters the aspect ratio of a new blob, for example, when a person is connected to another person or luggage. However and given the simplicity of the classifier, the implementation allows to proper detection of person blobs and car-object blobs. Finally, static foreground detection was implemented according to a simplified version of the method described in [1]. The final implementation allows to extract static foreground objects according to a counter and different thresholds and parameters based on the foreground mask. Final thresholds (0.0003 for "*learningrate*", 25 for "*FPS*", 5 for "*SECS_STATIONARY*", 1 for "*I_COST*", 4 for "*D_COST*" and 0.4 for "*STAT_TH*") demonstrated to be suitable for most of the scenes during testing. It was also observed that a proper static foreground detection is highly influenced by the learning rate, which may be adjusted for different scenes depending on the level of noise and variations on the scene, while keeping it low enough to allow for static foreground objects to be detected.

VII. TIME LOG

A. Maria Fernanda Herrera Perez

1) *Algorithm development and evaluation - Blob extraction stage:* : 1 hour.

2) *Algorithm development and evaluation - Blob classification stage:* : 1 hour.

3) *Algorithm development - Stationary foreground detection:* : 1 hour.

4) *Algorithm evaluation - Stationary foreground detection:* : 6 hours.

5) *Result reporting:* : 4 hours.

B. David Savary Martinez

1) *Algorithm development and evaluation - Blob extraction stage:* : 1 hour.

2) *Algorithm development and evaluation - Blob classification stage:* : 1 hour.

3) *Algorithm development - Stationary foreground detection:* : 1 hour.

4) *Algorithm evaluation - Stationary foreground detection:* : 6 hours.

5) *Result reporting:* : 6 hours.

REFERENCES

[1] Ortego, D. Sanmiguel, J.C.. (2013). Stationary foreground detection for video-surveillance based on foreground and motion history images. 2013 10th IEEE International Conference on Advanced Video and Signal Based Surveillance, AVSS 2013. 75-80. 10.1109/AVSS.2013.6636619.