Object Detection and Classification

Iuliia Alekseenko and Ricardo Luque

I. INTRODUCTION

The implementation of a foreground segmentation algorithm to detect moving objects on a stationary background can become the initial step in the pipeline of object detection and classification algorithms. This laboratory is focused on developing a *blob* extraction method using Grass-Fire and statistical-classifier. In computer vision a *blob* refers to a **binary large object**, which basically indicates a region of interest in an image which differs in properties from the surrounding regions of the image in context. In this project, a *blob* represents an object of interest belonging to the foreground.

The mentioned processes are performed both on a dynamic foreground mask, as the result of *pMOG2*'s background model subtraction, and on a stationary one which was created from the foreground mask and motion history images. The report describes the recursive *Grass-Fire* algorithm as well as sequential *Grass-Fire* algorithm by using **OpenCV**'s *FloodFill*. The simple statistical-classifier based on aspect ratio feature is able to classify objects in different categories such as person, car, object or group.

In the final part of this report, evaluation is performed qualitatively solely, given a set of video sequences. Moreover, the summary of the performance of the implemented algorithms is presented in the conclusion.

II. METHOD

There are two main methods to be implemented during this laboratory session: *blob* extraction and classification. The fist one is coded by following the implementation of *GrassFire* algorithm through **OpenCV**'s built in *floodfill* function and as well as self-coded one. The latter, classification, is based on aspect ratio information. Moreover, the processes are carried out on stationary and dynamic foreground masks. The implemented methods are described below.

A. *Blob Extraction*

The main purpose of this step is to find *blobs* (objects) in the foreground mask. Since a *blob* is represented as a group of connected pixels, the connectivity principal can be applied to check whether or not neighborhood pixels are connected. There are two commonly used types of connectivity - 4 and 8. In general, the 8-connectivity may work better than the 4-connectivity. However, the blob extraction with 4-connectivity is faster compared to 8-case. The algorithms, which in particular are based on the idea of connectivity are called connected component analysis algorithms, for example, the recursive Grass-Fire algorithm and the sequential Grass-Fire algorithm.

1) The Recursive Grass-Fire Algorithm: The recursive algorithm scans the foreground mask (binary image) from top-left to bottom-right. When the white pixel is found, the algorithm tries to find another white pixel among the neighboring ones according to the connectivity. This process is repeated until all pixels are visited. Simultaneously, the pixel is labelled as object on the output image [1], and the pixel is '*burnt*' in the input image in order not to check this pixel again [1].

2) The Sequential Grass-Fire Algorithm: Roughly, the algorithm performs a scanning on the input foreground mask in the same manner as the recursive mode does until an object (white) pixel is found. When this occurs, two things follow: first, the pixel is '*burnt*' in the image by setting it to zero; second, the output foreground mask is given a value of 1[1] to denote it is as object.

The algorithm works the same way as recursive up to the moment when neighborhood pixels are checked: if any of the neighboring pixels is an object, it is labeled as 1 in the output image and burnt in the input one as well as placed in a list. Finally, the first pixel out of the list is checked and its neighbors. If they are object pixels, they are labeled in the output image,burnt, and added to the list. This process continues until all pixels in the list are checked[1].

B. *Stationary FG Extraction*

At this step of the laboratory session, pixels which are classified as foreground for a long period of time are detected and create the stationary foreground mask. When these objects are not stationary, they will be removed from the stationary foreground mask. The work of the algorithm is based on the idea of foreground and motion history presented in D. Ortego and J. C. SanMiguel's writing [2].

C. *Blob Classification on Foreground and Foreground Stationary Masks*

Having extracted the blob, the next step of the algorithm is classification. The approach used in this laboratory session is based on the aspect ratio feature and a simple Gaussian

[1]The value of 1 or 255 refers to labelling a pixel as an object depending if it is a binary or integer *unit8* mask respectively

statistical classifier. The mean and the variance for each class were already provided in the code as the result of training the classifier. For the statistical-classifier distance, between the feature vector and the model is checked, where the aspect ratio (Equation 1) is the feature to be checked and defined as:

$$Aspect\_ratio = \frac{blob\_width}{blob\_height} \tag{1}$$

To measure the distance, weighted euclidean distance was used due to its faster computation and potentially proper results by using an additional parameter σ_{mi} (showed on Equation 2).

$$WED(\vec{f}, \vec{f_m}) = \sum_{i=n}^{n} \frac{\sqrt{(f_i - u_{mi})^2}}{\sigma_{mi}} \tag{2}$$

III. IMPLEMENTATION

A. *Blob Extraction*

1) *Grass-Fire Algorithm through OpenCV's Floodfill built in Function -The Sequential Grass-Fire Algorithm:*

As mentioned before, the sequential *Grass-Fire* algorithm was implemented (the *extractBlobs* function), in which *FloodFill* fills a connected region of pixels starting from a seed-point (given by the list of '*burnt*' values in the input image or the object pixels in the output one) with a specified color, given a specified connectivity. The connectivity is determined by the color/brightness closeness of the neighboring pixels. In here, the neighboring pixels were filled with a value of 1 or 255.

Finally, the *blob* is created taking as a starting point the objects detected by the algorithm. When the blob is created it is stored in a list of blobs, and a counter is increased to keep track of the existing *blobs*.

2) *Grass-Fire Algorithm Implementation through a Self-coded Function - The Recursive Grass-Fire Algorithm:*

The main logic of the self-coded Grass-Fire algorithms is implemented in the *check_nghb_pixel* function which provides two modes - 4-connectivity (default) and 8-connectivity (the connectivity can be chosen in the main function of the program). This function returns a *blob*, the counter is incremented to keep track of every new blob found, and the blob is placed in the list of blobs - *bloblist*.

<PIXEL> pixel_list - structure for storing the (x,y) coordinates of the points of interest for the algorithm (white pixels of the foreground mask). The foreground mask is scanned on a pixel level in two nested loops, and when a white pixel is found, its coordinates are saved on the list. Afterwards, the *check_nghb_pixel* function which implements the *Grass-Fire* function per se comes into play.

In here, this pixel is set to value 0 ('*burnt*')- converted into a background pixel before checking the neighbors of the pixel given to the algorithm. Thus, the pixels that have already been classified as foreground will not be classified again in the algorithm.

Depending on the *connectivity* parameter, the algorithm either checks 4 or 8 neighborhood pixels. If they satisfy the condition - they are white pixels, then their coordinates (x,y) are added to a pixel list. Then, the neighbors of these "new" pixels in the list are also checked following the same logic.

In order to calculate the weight and height of every blob, the next idea is implemented in the $maxmin\_coordinates$ function: the coordinates of every pixels added to the list are checked whether they are higher or lesser than the coordinated of the pixels stored as maximum and minimum for x and y. The maximum and minimum will be updated if this condition is met. Hence, the x and y of every blob are smallest coordinates along each axis, and the width and height are calculated as the difference between the maximum and minimum values for x and y.

3) *Removing small blobs:*

After the width and height of the blobs have been computed we can get discard those blobs that do not meet a specific size threshold in order to avoid noise in the detections. Here, in the *removeSmallBlobs* function, every blob is checked to be bigger than the minimum *blob* dimensions which the user can tune (currently set to MIN_WIDTH = 20 and MIN_HEIGHT = 20). The function is called after both the recursive Grass-Fire algorithm and the sequential Grass-Fire algorithm.

B. *Stationary FG Extraction*

The stationary foreground extraction is implemented in the $extractStationaryFG$ function. As mentioned on [2] it is *'based on the spatio-temporal variation of foreground and motion data'* using motion data, since it allows to filter out the moving regions while extracting the stationary behavior of our regions of interest. The final step of the extraction is thresholding motion activity to help both creating a blob and progressively destroying it.

This process is performed using motion history images and the foreground mask ($fgmask$) with the following parameters for tuning:

- $FPS = 30$ refers to the frames per second of the video to be analyzed. It was set to 30 since this is a standardized video speed for TV/broadcast.
- $SECS\_STATIONARY = 10$ - time threshold to consider an object as static (can be set by the user). When an object is in the same location for the time longer than this value, this object will be detected as stationary. Therefore, the higher the value the longer it would take for an object to appear the blob on the stationary foreground mask $sfgmask$
- $I\_COST = 5$ - parameter referring to the change speed of pixels to be detected as stationary on the foreground mask.
- $D\_COST = 5$ - parameter referring to the disappear time of pixels from the stationary foreground mask.

- $STAT\_TH = 0.5$ - refers to a threshold value for obtaining detections in scenarios with motion.

In rough terms, the process follows the steps:

1) Obtain the foreground mask ($sfgmask$) through background subtraction.
2) Measure the foreground temporal variation to obtain a **Foreground History Image**, named $fgmask\_history$ (showed on 3).

$$FHI_t(x) = FHI_{t-1} + \omega_{pos}^{f} * FG_t(x)$$
$$FHI_t(x) = FHI_{t-1} - \omega_{neg}^{f} * (\sim FG_t(x)) \tag{3}$$

   Where FHI_t, ω_{pos}^{f}, ω_{neg}^{f} and $\sim FG_t$ refer to the $fgmask\_history$, $fgmask$, I_COST, D_COST and $bgmask$ respectively.
3) Normalize $fgmask\_history$ to the range [0, 1] considering the video framerate FPS and the stationary detection time $SECS\_STATIONARY$, obtaining $fg\_history\_norm$ (showed on 4).

$$\overline{FHI}_t(x) = min\{1, \frac{FHI_t}{FPS * secs\_stationary}\} \tag{4}$$

4) Applying a binary thresholding to the normalized history image values of $\overline{FHI}_t(x) \geq \eta$.

Generally speaking, the appearing and disappearing ratio of the blobs is given by I_COST and D_COST factors, respectively. Mathematically speaking FPS and $SECS\_STATIONARY$ are not relevant on their own, yet their product is the important value since it is used for thresholding the $fgmask\_history$. Moreover, the *blobs* should not appear on the stationary foreground mask ($sfmask$) until the objects stop and they should disappear afterwards. If they remain on the mask for longer time, it is due to the tuning of both I_COST and D_COST values.

C. *Blob Classification on Foreground and Foreground Stationary Masks*

The same algorithm is used to classify blobs on both, foreground and foreground stationary masks, which is implemented in *classifyBlobs*. In total, the function is able to classify the object into 5 groups: UNKNOWN=0, PERSON=1, GROUP=2, CAR=3, OBJECT=4. The classifier is based on the aspect ratio (AR) feature and simple statistical-classifier. Since this approach requires the training step, the three aspect ratio models were already provided for such classes as PERSON, CAR, and OBJECT.

First, the aspect ratio is calculated according to Equation 1, and the weighted euclidean distance is measured for each of three classes according to 2. Later the set of if-else is performed for the classification based on the obtained distances:

- IF person<car && person<=object THEN it is PERSON
- ELSE IF car<person && car<=object THEN it is CAR
- ELSE IF object<person && object<car THEN it is OBJECT
- ELSE it is GROUP

D. *Running the code*

Before running the code, it it required to set the valid path to the video sequences being processed. The user should provide it in the main function of the program. By default, the folder is already chosen, but it is required to place the files in the folder on the user computer. $learningrate = .0005$ - value between 0 and 1 that (or -1 value which uses some automatically chosen learning rate) indicates how fast the background model is learnt. This parameter can be tuned in the main function of the program.

The thresholds and parameters of algorithms are set, but the user can tune them in case if it is needed. Every possible parameter to tune is described in the corresponding parts of the implementation.

IV. DATA

The dataset consists of four video groups - *AVSS2007*, *ETRI*, *PETS2006*, and *VISOR*.

The first video group is *AVSS2007*, which demonstrates the platform of an underground station. Some frames have people waiting for a train or walking, as well as trains arriving and departing, and people taking and leaving the train. In this sequences, it is important to set proper values in order not to detect big objects as those of our interest in the foreground mask. Also, such problems as occlusions (between various objects and people) and reflections make classification more challenging.

The second video group is *ETRI*. The scene is a nature environment and there are some frames containing person or people walking. The background is quite difficult due to leaves movements which may lead to not perfect foreground segmentation.

The third video group shows the entry hall of an underground station (*PETS2006*). The main challenge is that the scale of object is different - people are walking closes to the camera and people are in some distance from the camera who may be not detected and classified properly due to scale issues.

The last video group (*VISOR*) in the dataset is a parking lot with some frames showing cars entering and leaving. There is also a road with traffic in the upper part of the video. False detection may occur due to the position of the camera which leads to some objects are less then the minimum size threshold.

V. RESULTS AND ANALYSIS

A. *Blob Extraction*

Since blob extraction is based on the background subtraction approach, there is a strong segmentation dependency - the better the foreground is detected, the better blob extraction is performed. Thus, it is not the best approach for "complex" scenarios since overlapping objects and partial object detections are not supported.

In fact, some problems were faced when a fairly high value of $learningrate$ was chosen. The effects were that the full object was not observed, hence, we can notice a poor detected object on the *blobs* shown in Fig. 1

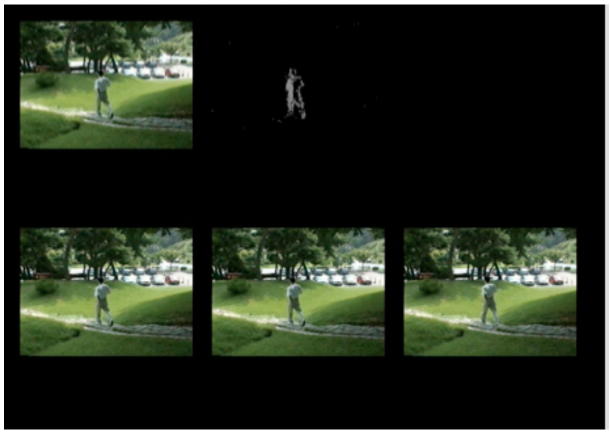

Fig. 1. Results of a poor *blob* extraction.

However, we can observe descent results in most scenarios as shown in Fig. 2.

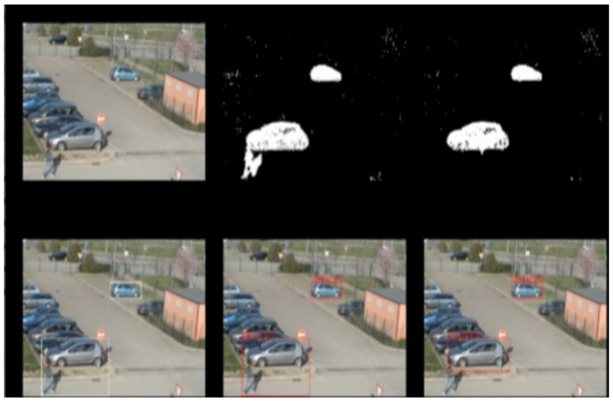

Fig. 2. Results of Blob extraction, showing a good and a bad blob extraction since two objects are extracted as one.

B. Blob Classification on Foreground and Foreground Stationary Masks

The blob classification is strongly dependent on the blob extraction quality, because of the latter aspect ratio computation. For example, Fig. 3 shows the correct classification of a person, whereas Fig. 4 shows a classification labeling confusion, since it shows a blob classified as an *object (4)*, while it is actually a *person (1)*

In general, when classifying the objects on foreground and foreground stationary masks, there are similar problems have been seen on both of them. Since very simple feature (AR) is used as well as the basic classifier, many objects, for example, cars or persons are not classified properly until the size of their blobs do not reach minimum width and height thresholds. This means that in case of partial occlusions the objects may be misclassified.

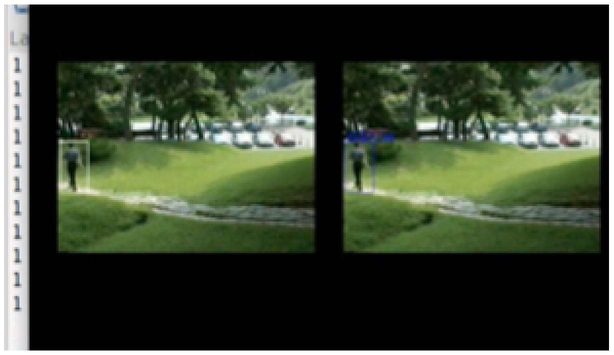

Fig. 3. Classification mistake between person and object, categories 1 and 4.

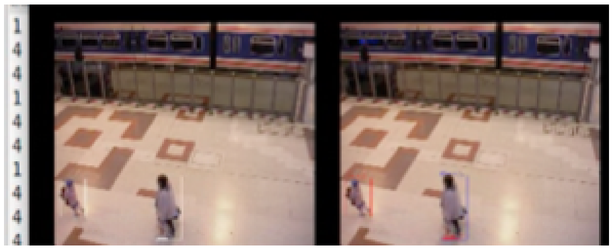

Fig. 4. Classification mistake between person and object, categories 1 and 4.

Also, the aspect ratio method leads to the problem when any object fulfilling the ratio will be classified either as an object, car or person. As well as simple classifier is quite straightforward method which is affected by errors in computing AR. The typical misclassification problem is shown in Fig. 3.

C. Blob creation and destruction on both Foreground and stationary Foreground Mask

As mentioned earlier, the correct behaviour is expected to have an object disappearing on the foreground mask, while it should start appearing on the stationary foreground mask after a given time. It is important to mention that this is strongly dependent of the learning ratio since it defines how objects incorporate into the background model.

As a matter of fact, problems were faced when tuning the parameters, which led to some failures in the results at first. For example, although the blob appeared on the stationary mask as soon as the car stopped the blobs take too long to disappear from the stationary foreground mask. Therefore, distorted blobs are seen. We can observe this with a $learningratio = 0.0001$, in Fig 5,

Additionally, although the *blob* is disappearing on the foreground mask, on the stationary foreground mask, extra blobs are observed even after the object in context has moved. This can be observed in Fig. 6

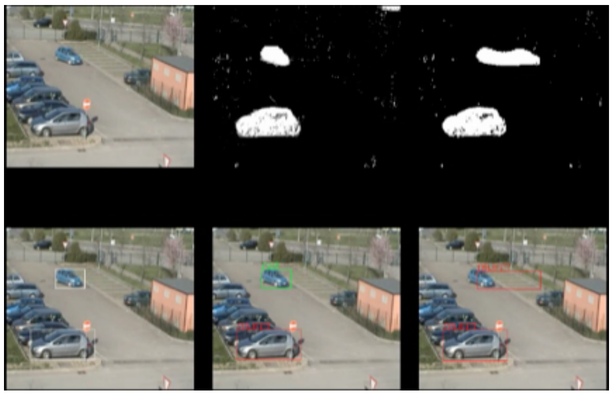

Fig. 5. Distorted *blobs* on the stationary foreground mask as a result of a $learningratio = 0.0001$.

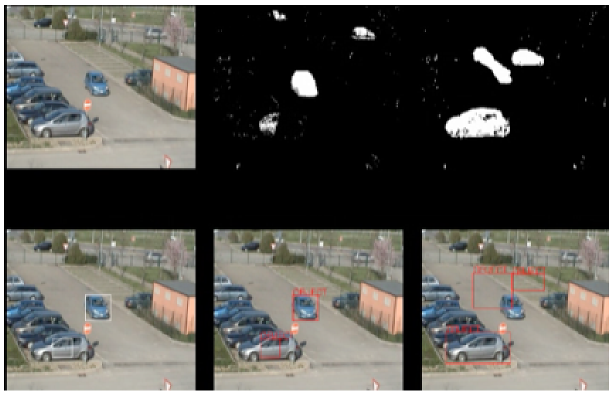

Fig. 6. Disappearing object on the foreground mask and non disappearing *blob* on the stationary foreground mask as a result of a $learningratio = 0.0001$.

However, when the right tuning was found, the behavior improved drastically. In the following Fig. 7 and Fig. 8 a correct behavior is observed.

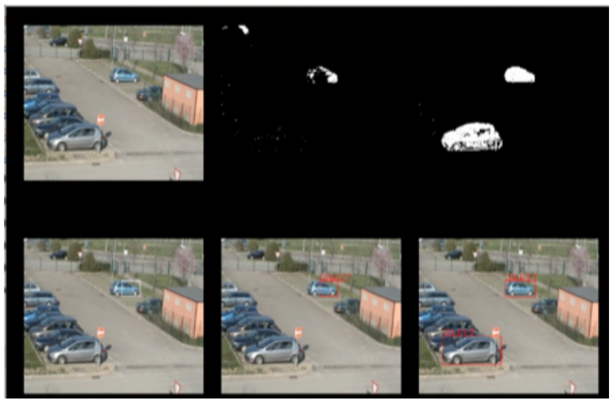

Fig. 7. Disappearing object on the foreground mask and non disappearing blob on the stationary foreground mask as a result of a $learningratio = 0.0001$.

VI. CONCLUSIONS

We can observe throughout the development of the laboratory session that the behavior of the algorithm improved

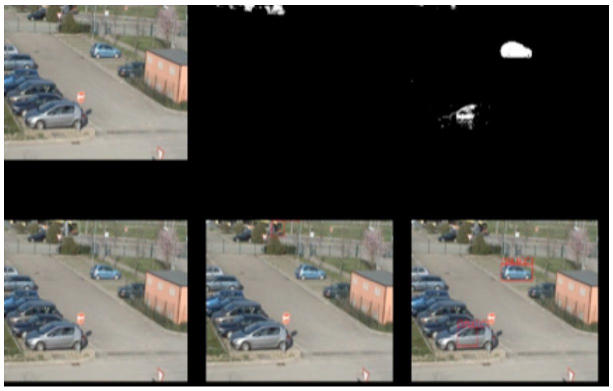

Fig. 8. Disappearing object on the foreground mask and disappearing blob on the stationary foreground mask after some time as a result of a $learningratio = 0.0005$.

drastically when small changes on the tuning was performed. In fact, the objectives were met and both the foreground and stationary foreground masks resulted in fairly proper blob extraction and behavior. The classification, however, could have been improved since it is only based on an $Aspectratio$ approach. The learning ratio is one of the most essential and critical factors in this algorithm since it indicates how fast a background model is learnt, and therefore determines the performance as the whole.

VII. TIME LOG

A. *Blob Extraction using floodfill Function and Classification*

The time spent on this part of the laboratory session is 3 hours in total. The blob extraction using *floodfill* function took around 2 hours to implement (including the function *removeSmallBlobs* - remove small blobs, and the classification step was coded during 1 hour.

B. *Stationary FG Extraction and Classification*

The time spent on this part is around 10 hours - around 4 hours required to read the article [1] and learn the concepts out of it, 4 hours to implement the algorithm, as well as 2 hours were spent to find bugs and tune the parameters.

C. *Grass-Fire Algorithm Implementation through a Self-coded Function*

The implementation of Grass-Fire algorithm from scratch took around 10 hours - 2 hours to understand the ideas behind the algorithm, 5 hours to implement it, and 3 hours to find bugs and correct the syntax.

D. *Report*

Around 8 hours: writing the draft, proof-reading and editing.

References

[1] Moeslund, Thomas. (2012). Introduction to video and image processing. Building real systems and applications. 10.1007/978-1-4471-2503-7.

[2] D. Ortego and J. C. SanMiguel, ”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, Krakow, 2013, pp. 75-80.

[3] Bradski, G (2008-01-15) ’Dr Dobb’s Journal of Software Tools’ The OpenCV Library (2000)