Background Initialization and Maintenance

Eduardo Daniel Bravo Solis, Santiago Herrero Melo

I. INTRODUCTION

The main goal of this assignment is to obtain and image that represents the background for each of the proposed videos. This background image will be obtained from the original video sequences, and tested all together in the same code with the same parameters. This code will be developed in C++ using the OpenCV library.
Once the background images are obtained, and using some provided images as reference, we will evaluate the performance of our code.
Finally, this new code will be integrated in the code developed for foreground substraction in the previous assignment.

II. METHOD

The methods performed in this assignment are based in the "hot restart" implementation studied during the course in the theoretical lessons.

A. Background initialization and Maintenance

The objective of this exercise is to obtain a representation of the background (still part) of the video sequence. For doing so, all the video sequence will be run, selecting as background pixels those that, after a defined number of frames, don't change their value. After this number of frames, they will remain as background pixels in the background image, independently of varying their value in the following frames.

B. Integration with Foreground Segmentation

The goal of this exercise is apply the background initialization model in a foreground segmentation. During the performance of the previous exercise, the first frame of the sequence was considered as the background over which we should perform the segmentation. With this new approach, we search for a cleaner segmentation.

III. IMPLEMENTATION

A. For Background Initialization and Maintenance

The background model starts with a dark (zeros) image that along the process is filled with pixels when they reach a threshold. With this method is easier to visualize the pixel behaviour in the model. For this, every frames is process by *processBackgroundModel* function, that using a counter matrix to keep track of the number of times a pixel is classified as background.
At the end of the sequence, the pixels that remained as zero are set to their value on the last frame. This with the intention of using pixel values that were not in the first frame and that help creating a better representation of the background.
To deal with different sizes of videos, we decided to use all the frames in the sequence to create our model. Also, in case the threshold counter is lower than the number of frames, the threshold is reduced to total number of frames.
This section only creates the background model and stores it. Background models for all videos will be saved a the path that should be set correctly on the main.cpp file.
The function *initializeFrames* makes sure that, when working in each video sequence, the size of the intermediate and output images is going to be the same as the one of the input image. This function is called only for the first frame of each video sequence and sets all the intermediate matrices to have the same size, but filled with zeros, and sets the first frame as a reference background.
The *processBackgroundModel* function receives as an input the current frame, and transforms it into grayscale. Then it creates a mask between the successive frames and the reference background (first frame) for the first *BKG_ESTIMATION* frames, updating the background via selective running average, whose formula is shown in the equation (1).

$$\_bkg1 = \_alpha * (\_frame) + (1 - \_alpha) * \_bkg; \quad (1)$$

In an intermediate matrix called *_bkg_counter* each pixel receives a value depending on the number of frames it has kept its value invariant. If this value changes, it will have a zero value in the *_bkg_counter* matrix. Every original pixel with a value in the *_bkg_counter* matrix over *BKG_THRESHOLD* will be added to the final background model.
When the number of frames settled by *BKG_ESTIMATION* has been achieved, it is considered that the final background model has been generated. It is stored both in grayscale and in color, and can be returned by the methods *getBackgroundModel* and *getColorBackgroundModel* respectively.

B. For Integration with Foreground Segmentation

To integrate the background model with foreground segmentation, we perform two loops for each video The first one gets the background model and the second uses this model as a base for the foreground segmentation algorithm. We found this method optimal for having a complete evaluation of all the frames in the sequence, and be able to evaluate the segmentation even from the first frame.
The method *processFGSegmentation* was added with the intention to isolate the process of foreground segmentation from the background modeling. This method is called during

the second loop of the video, and uses the background generated previously.
The user only needs to specify correctly the path to read and save the outputs. A folder with the name of the video must be on the results path.

C. Running the code

This program is very easy to run. In the first place, the user must provide the correct path from which the frames of the video sequence will be taken, and then the path to which the generated images will be sent.
The other parameters to be selected are the threshold and α value for the segmentation, the amount of frames that the pixel has to remain still to be considered part of the background (*BKG_THRESHOLD*) and the number of frames that will be used to generate the background image (*BKG_ESTIMATION*).

IV. DATA

The dataset is extense, and is divided depending on the problems that may be faced using them.

A. backgroundMotion

The main problem with this video is an advertisement panel that changes its content during the sequence. Although its position in the video doesn't change, its pixels values change with its content, so it is detected as a false positive (we are looking for cars that actually vary their position in the sequence).

B. basic

In this sequence we face similar problems as we did in the previous assignment. There are two different video sequences, and in both we deal with small variations because of the enviroment (wind moving leaves of trees, water moving aquatic plants). Although, in the second video, we may also deal with the problem that it is a quite short sequence.

C. clutter

There are two video sequences in this folder. In the first one, the people are constantly moving so it is difficult to find a good threshold for the background update. Also, the threshold itself is chaotic, as it is a board with things written, not just a road or a landscape as in other sequences.
The second sequence shows several elements that stay for a long time in the same position but will change its location or disappear after several frames, so they can be selected wrongly as background.

D. illuminationChanges

In this video we see a dark room that, during the video sequence, is suddenly illuminated by switching on the lights of the room. For creating the background, we can deal with the dark part of the sequence, the bright part of the sequence, or make an average image.

E. intermittentMotion

In these sequences, we see people staying for a long period of time in the same position, so we deal with the problem of them being perceived as background when they are not.

F. jitter

The main problem in these videos is that the camera recording is not completely still, either because of bad positioning or external influence (such as wind). Therefore, in this case, pixels will vary their values shortly and will be difficult that they are stored as background.

G. veryLong

The video sequences in this family are formed by many thousand of frames, which makes its computation very costly.

H. veryShort

In this family of video sequences, the main problem is its short length. The sequence is formed by just a few frames, so numerical values for the parameter *BKG_THRESHOLD* that work well in this video may be too less for the other families. What's more, the sequences are not consecutive, meaning there are jumps of time between one frame and the following one, which may add chaos and a lot of noise to the algorithm.

V. RESULTS AND ANALYSIS

A. Scene Background Modelling

In order to analyze the performance of our algorithm, we will check the performance of every family trying different values in the variables that the user can modify.
The code used for this purpose is based on the one provided in [4], from which we use the functions *UTILITY* to dive into every folder that has data that we are going to use and generate the .txt file where our results are stored, and the *Evaluate* for calculating every parameter. In the script *evaluate_datasetSBMNet* we only need to put the proper paths from which we will take the background groundtruths and the resulting backgrounds estimated by our algorithm.
For all our tests we will use the following values:
$threshold = 40$
$\alpha = 0.05$
$BKG\_THRESHOLD = 50$
In our first test we have used the parameter $BKG\_ESTIMATION = 200$, which means that we have used the 200 first frames of each video sequence to generate the background image. In the table I we can see the results obtained.

Using the same parameters as in the previous test, but now using all the images of the sequences for generating the background image, we obtain the results shown in table II, which are widely improved for many video sequences and in the general picture.

In table II we can see that our performance has been improved respect to the values in table I, especially with the *veryShort* family of videos, where this change

Videos	AGE	pEPs	pCEPs	MSIM	PSNR	CQM
bkMtn	3.1542	0.0135	0.0037	0.9598	27.3963	28.1424
basic	7.7289	0.0903	0.0500	0.8663	58.5931	59.7913
clutter	63.4007	0.4563	0.3865	0.2274	9.4355	11.1095
ilChg	10.8962	0.1095	0.0779	0.6040	18.3546	19.9716
inMtn	29.8695	0.2063	0.1858	0.6921	13.8536	15.2408
jitter	59.2397	0.4182	0.3155	0.1542	8.6734	10.3072
vLong	13.5405	0.0978	0.0157	0.7343	17.3231	18.6519
vShort	107.1019	0.689	0.4971	0.2552	5.3733	6.6821
Ovall	26.8328	0.1988	0.1479	0.6054	21.9471	23.3164

TABLE I: First test

Videos	AGE	pEPs	pCEPs	MSIM	PSNR	CQM
bkMtn	3.6558	0.0266	0.0048	0.969	31.1718	31.9331
basic	10.206	0.1784	0.0623	0.8972	24.4411	25.904
clutter	15.6701	0.2028	0.0833	0.7283	19.3722	20.656
ilChg	32.381	0.8087	0.6597	0.7515	17.2021	18.2287
inMtn	14.7879	0.165	0.0805	0.7994	19.1092	20.1223
jitter	11.5017	0.1575	0.0394	0.7839	21.7278	22.8307
vLong	8.4319	0.0917	0.0017	0.9614	25.6902	26.2921
vShort	9.34	0.113	0.0143	0.9186	23.6704	24.3873
Ovall	13.2468	0.2180	0.1182	0.8512	22.7981	23.7943

TABLE II: Second test

of approach has meant a significant improvement. The resulting backgrounds are shown in the image 1

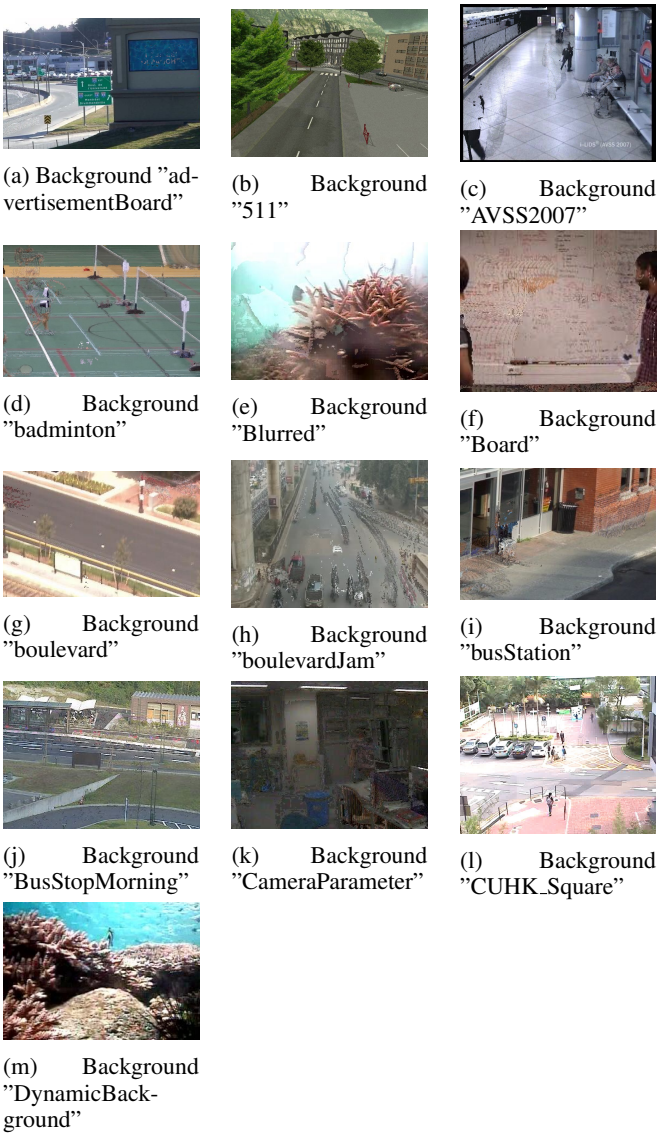
For the evaluation, we will take as reference both the best algorithm (**MSCL**) and the worst one (**RFSA**) in the global classification. Our performance is way poorer than **MSCL** algorithm, but very similar (although worse) than **RFSA** in the overall picture. We perform better than both algorithms in the category *backgroundMotion* (category in which we perform better than any other algorithm). For categories like *jitter* and *clutter* we outperform the **RFSA** algorithm in several aspects like percentages of error, average of graylevel error and similrity index. Unfortunately, for every category (except for *backgroundMotion*) we get worse results than any other algortihm in terms of Peak Signal to Noise Ratio and Color Image Quality Measure, which is our Achilles' heel. It is important to mention the huge improvement performed in *veryShort* category, which has definitely lead to an overall improvement. In terms of percentage errors and similarity we are between the best graded algorithms, having in all the aspects a similar performance than **RFSA** algorithm.

B. Foreground Segmentation. Baseline category

In table III we show the parameters of the evaluation of the baseline category in the CDNet, but using a generated background, and using a threshold of 40 and an α parameter of 0.05.

We take as reference the best algorithm in average of the website, which is the **PAWCS** algorithm. Our algorithm is clearly worse in recall and false negative rate, but we outstand it in terms of specificity and false positive rate. We have a very similar but bit worse result in precision, PWC and F-Measure respect to the **PAWCS** algorithm. Compared to all of the algorithms, our ranking places us in the eighth place.

In the image 2 we see all the obtained backgrounds for the video sequences in the baseline category, using the code for background initialization explained in the section III-A of the report.



(a) Background "advertisementBoard" (b) Background "511" (c) Background "AVSS2007"

(d) Background "badminton" (e) Background "Blurred" (f) Background "Board"

(g) Background "boulevard" (h) Background "boulevardJam" (i) Background "busStation"

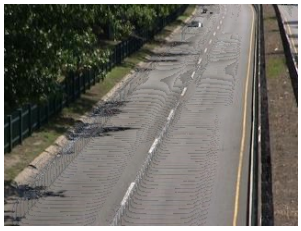
(j) Background "BusStopMorning" (k) Background "CameraParameter" (l) Background "CUHK_Square"

(m) Background "DynamicBackground"

Fig. 1: Backgrounds of all the video sequences tested

Parameters	Test
Specificity	0.9952878339
Recall	0.7296432765
False Positive Rate	0.0047121661
False Negative Rate	0.2703567235
Percentage of Wrong Classifications	1.4996947035
Precision	0.8512292820
F-Measure	0.7839224243
Average Ranking	10.86

TABLE III: Test with foreground mask



(a) Background ”highway”

(b) Background ”office”

(c) Background ”pedestrians”

(d) Background ”PETS2006”

Fig. 2: Backgrounds of the baseline category video sequences

VI. CONCLUSIONS

To create a robust algorithm for background modeling, more challenging video sequence are required, to simulate situations that are close to real-world applications. Our methods proved to be robust in scenarios with illumination changes, moving objects in first frames and low amount of data. This performance was obtained only after we decided to use all the frames in a sequence as reference for creating the background model.
The integration of this model to the foreground segmentation allowed us to improve the results of our previous works providing less noisy and ghost-free results.
These methods can be used to surveillance applications with the advantage of generating the background and segmentation on the fly.

VII. TIME LOG

A. Background Initialization and Maintenance

The first part of the work took around 90 minutes to create a first version of the code and configure for the new set of videos. After analyzing the problems arise from the new sequences, a second code version was created, were 60 minutes were needed, including debugging and testing.

B. Evaluation of the Algorithm for Background Initialization and Maintenance

The understanding and preparing of the MatLab code to evaluate the performance of the background initialization took around 90 minutes. The obtention of the performance parameters, their understanding and comparison with the other algorithms presented took around an extra hour.

C. Integration with Foreground Segmentation

Coding a two loop algorithm with different behaviour required close to 2.5 hours. Documentation process, including comments and version control required 30 minutes.

D. Evaluation of Foreground Segmentation with the Integration of the Algorithm for Background Initialization and Maintenance

As for this section the code used is the same as in the first assignment, there was no need for modifying the code or understanding it. Therefore, 20 minutes were enough for the processing and analysis of the results.

REFERENCES

[1] G. O. Young, Synthetic structure of industrial plastics (Book style with paper title and editor), in Plastics, 2nd ed. vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 1564.
[2] W.-K. Chen, Linear Networks and Systems (Book style). Belmont, CA: Wadsworth, 1993, pp. 123135.
[3] C. Wren; A. Azarbayejani; T. Darrell; A. Pentland (July 1997). ”Pfinder: real-time tracking of the human body”, IEEE Transactions on Pattern Analysis and Machine Intelligence. 19 (7): 780785.
[4] URL: http://pione.dinf.usherbrooke.ca/code/