Foreground Segmentation and Shadow Detection

Eduardo Daniel Bravo Solis, Santiago Herrero Melo

I. INTRODUCTION

The main task to perform during this laboratory session is to implement a simple foreground segmentation module based on the frame difference. Once this is completed, through selective running average, the background will be progressively implemented in order to avoid false positives and possible ghosts.
We will use four different videos for this task, but all of them will be evaluated with the same code and parameters (an individual code won't be developed for each video sequence) and the code will be implemented in C++ using the OpenCV library.
Afterwards, we will explore other different possibilities for foreground segmentation.

II. METHOD

All the methods used during this laboratory have been studied during the lessons, therefore, they can be found in the slides of the subject.

A. Generation of Foreground Segmentation Mask

The basis for solving this problem has been the frame difference, in which we select the first frame of each video as the background, and we make a simple difference operation between this frame and the following ones. Afterwards, this difference is thresholded in order to visually differenciate background and foreground.

B. Progressive update of background model through selective running average

The purpose of this exercise is, once we have our foreground mask, to update the background model with the current image, with the pixels that do not belong to this foreground mask. This method is performed in order to avoid possible false positives and ghosts. Here we introduce an α parameter which determines the adaptation speed. This process can be either blind or general. For the first one, only the background pixels are updated, while the later one does not make any discrimination for the update process.

C. Extension of Frame Difference using other Color Spaces

For this exercise, we will perform the same task as in the first problem, but instead of converting the image into grayscale, we will work in each of the channels of the color space (in this exercise both BGR and HUV spaces).

D. Advanced Background Substraction (Single Gaussian)

During this exercise we will apply a Gaussian distribution to each of the pixels of the frame, estimating the mean and standard deviation and updating these parameters depending on the incoming values and foreground mask.

III. IMPLEMENTATION

A. Functions for Generation of Foreground Segmentation Mask

For selecting the first frame as background, we have created a *counter* variable that will hold the number of the current frame. When its value is equal to 1, meaning the first frame of the video sequence, it enters a function called **setBackgroundModel** that copies such frame (in grayscale) into the *_bkg* two dimentional matrix type element.
For proceeding with the difference operation, in the **process** function we transform the input frame (which is the current one according to the counter) and transform it into grayscale. The difference between this image and the previously one saved as background will give us the foreground. As we are working in grayscale images, a simple thresholding is enough for the purpose of this exercise. The value of the threshold is previously selected by the user.

B. Functions for Progressive update of background model through selective running average

(a) Original image (b) No background upgrade (c) Background upgrade

Fig. 1: Solving of ghost problem by using the background update

In the first place, we will add to the constructor of the background segmentation a parameter called *_alpha* which will also be selected by the user. Then, in the **process** function we apply the formula in order to use the adaptation speed:

$$\_bkg1 = \_alpha * (\_frame) + (1 - \_alpha) * \_bkg;$$
$$\_bkg1.copyTo(\_bkg, (1 - \_fgmask));$$

Where *_bkg1* is a new matrix in which we store the upgraded background, and we copy it to the background

except the points that belong to the foreground.

C. Functions for Extension of Frame Difference using other Color Spaces

Fig. 2: Results for the H, S and V channels for a HSV image

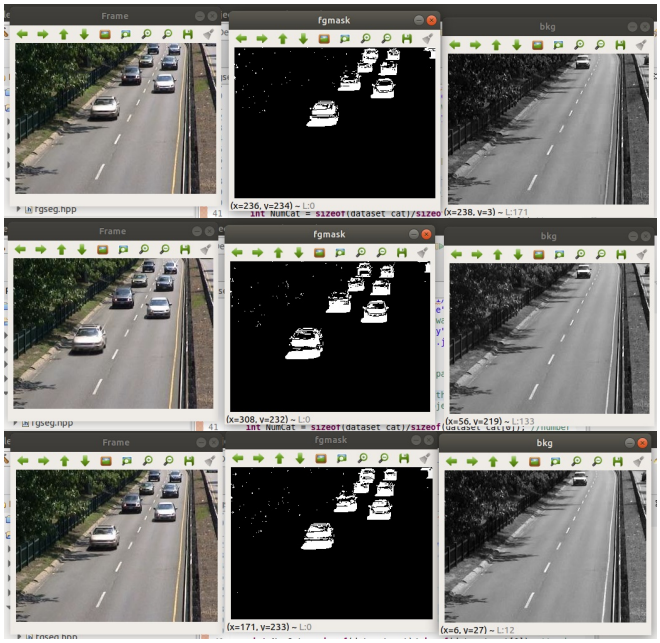


Fig. 3: Results for the B, G and R channels for a BGR image

In this case, we are basically recycling the code implemented for the two previous exercises. Instead of converting the background and frames into grayscale images, we convert them to HUV images. The channels are stored in an array, and the channel to work on will be selected as an input parameter of the function (in functions **process** and **setBackgroundModel**). The rest of the code works exactly the same. In the case of BGR images, as input images are of this type, it is enough to comment the lines in which the image is transformed to HUV space color and proceed the same way.

D. Functions for Advanced Background Substraction (Single Gaussian)

First, the window size α and initial standard deviation σ are set by the user. With the last one, the variance matrix is created. Following the same logic we used in the simple foreground substraction, we take the first frame as the background model, which is stored as the mean matrix *_mean* that holds the mean values of each pixel. This will after be used for calculating if a pixel belongs or not to the foreground, and to update the background.
For creating the mask, we have developed a function called **get_new_fgmask** which receives as an input the current frame of the video and returns the calculated mask. For doing so, we first check if the pixel belongs to the background. This will happen in the case that the difference between the background pixel and the mean is inside the range $[-N\sigma, N\sigma]$, where N is a threshold stated by the user. In the case the pixel belongs to the background, we will apply the following formula at each pixel for updating:

$$\mu_{t+1} = \alpha * Image_t + (1 - \alpha) * \mu_t$$

Then, for all the frame, we will upgrade the σ with this formula:

$$(\sigma_{t+1})^2 = \alpha * (Image_t - \mu_t)^2 + (1 - \alpha) * (\sigma_t)^2$$

Our first interpretation of the algorithm was to update both μ and σ^2 only for the background pixels. This will cause a constant decrease of σ^2 values in most pixels, turning a great part of the image into foreground.
So after some test and following the original paper, we decided to condition only the update of the μ, and update the value of the σ^2 on every frame.
Finally, in the **process** function we threshold the image over 0, so that all the foreground pixels appear white and the background ones appear black, similarly to the case of the simple foreground segmentation mask.

E. Running the code

This program is very easy to run. The only thing the user needs to do is to select an appropriate path for the video frames (otherwise the program will return an error) and select the threshold and alpha values that he/she considers appropriate. Also, it is important that the output address is also correct, in order to properly store the foreground frames for further evaluation.
In the case of using color spaces, the user also has to select which color space to use and with which channel of that color space he wants to work.

In the case of the Gaussian program, the user also has to select the value of α and the number of standard deviations he wants to work with.

IV. DATA

The dataset consists of four different videos, each of which present different challenges for the proposed tasks. The first video is the *highway* video, in which a highway is shown, where cars and vans are circulating. The main problems here are the trees next to the highway, which are moved by the wind so they are wrongly detected as cars; and that in the background image there is a moving element (a car) that becomes a ghost.
The second video is the *office* video in which we can see an office and, after a certain amount of time, a man appears to take a book and start to read it. Here the main issue is to deal with his t-shirt because it has white components, which are not detected as part of the person when thresholding, and with the ghost that this man may produce after staying during a long time in the same spot.
The third video is the *pedestrians* video in which we see people walking thrpugh a park. Here we face a problem with the shadows of people. As they move with the people of the video, they are detected as a part of the person itself.
The last video is the *PETS2006* video which shows the terminal of a station. The main problems to deal with here are occlusion of certain people, and a man who stays for a long time in a same position, so he can either create a ghost or become background, depending on the parameters we work with. Also with the reflection of the shadows of people in the pavement.
For all the videos, noise is a main problem.

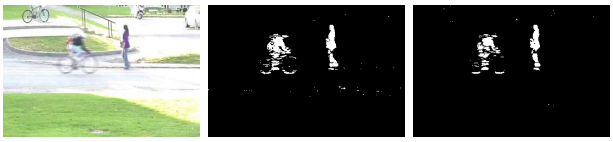
V. RESULTS AND ANALYSIS

A. Baseline category

We have tested the code with the following parameters, and then evaluated the algorithm using the ChangeDetection datasets. For the evaluation, we export each frame of the foreground mask into an appropriate folder, depending on the video we are working on, so that the evaluation with the MatLab code is faster and easier.

Params	Test1	Test2	Test3	Test4	Test5
Thres	30	35	40	45	50
Alpha	0.2	0.01	0.005	0.01	0.003
Sp	0.989948	0.99378	0.99549	0.996367	0.996883
Re	0.77612	0.827802	0.802081	0.773663	0.742259
FPR	0.010052	0.00622	0.00451	0.003632	0.003117
FNR	0.22388	0.172198	0.197919	0.226337	0.257741
PWC	1.824627	1.203956	1.134429	1.152075	1.215217
Pre	0.729	0.818058	0.864627	0.892661	0.910488
FM	0.7482218	0.818582	0.829795	0.826962	0.815545

We can see that the program developed is very good in terms of not detecting false positives, but not as good for false negatives, which is due to the problems with shadows or tree movement, depending on the video sequence. Precision gets increased with the increasement of the threshold value as the false positives rate decreases, but on expense of a worse false negative rate. From all the threshold-alpha combinations tested, we can see that, in general, low values of alpha around threshold values between 40 and 50 perform better.



(a) Original image (b) Threshold = 30 (c) Threshold = 40

Fig. 4: Difference of output images with different thresholds

In relation to the approaches from ChangeDetection datasets, we can see that we have reached a very good Percentage of Wrong Classifications, in some cases only worse than the PAWCS algorithm and outperforming the rest of algorithms in that table in what regards to that aspect.
In the case of precision, we have managed to outperform all of the algorithms presented, as well as in terms of specificity and false positive rate.
Our weakest performance is in what regards the false negative rate and recall (which is obvious as it is directly related to false negatives). Here our performance has just been average for the table.

B. Dynamic Background category

We have tried the code with different amounts of standard deviation (N in the operation N*σ) and different α values to see how the program behaves differently. Then we have tested the algorithm with the ChangeDetection datasets. As in the previous case, we export the images of the mask and use the provided MatLab code.
We have always used the same threshold and σ values (40 and 100 respectively), in the case of threshold because gave good results when used in the foreground mask calculation, and for standard deviation because we wanted to use a big enough value to avoid misclassifications.

Parms	Test1	Test2	Test3	Test4	Test5	Test6
α	0.05	0.05	0.05	0.1	0.1	0.1
N	2	3	8	2	3	8
Sp	0.923	0.958	0.971	0.946	0.968	0.974
Re	0.832	0.718	0.334	0.837	0.716	0.343
FPR	0.077	0.042	0.029	0.054	0.032	0.026
FNR	0.168	0.282	0.666	0.163	0.284	0.657
PWC	8.006	5.219	5.638	5.798	4.252	5.323
Prec	0.272	0.368	0.309	0.347	0.449	0.372
FM	0.385	0.446	0.265	0.468	0.514	0.302

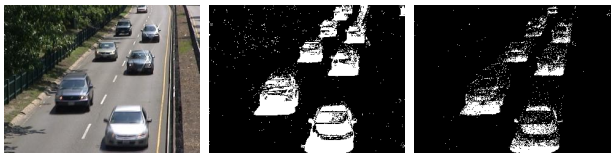
The main problem faced with this code is that in the first frames of each video sequence, a great part of the background becomes white, for afterwards becoming black eventually, which leads to a very big misclassification and a lot of false positives. Also, in the second and fourth sequences, people standing for a long time in the same

place leave ghosts that are difficult to remove. This model shows more noise than the previous one, as it only deals with gaussian noise.

(a) Original image (b) Gaussian filtered background

Fig. 5: Main problem with the background at the beginning of the filtering

We tried a very big N value because the result is visually better, but the overall performance, as can be seen in the table, is much better with a small N. The recommended values are usually 2.5 and 3, which is why we chose to test the code with the $[-2\sigma, 2\sigma]$ and $[-3\sigma, 3\sigma]$ intervals. As we see, for an interval $[-8\sigma, 8\sigma]$ the performance is much poorer.



(a) Original image (b) Use of [-3σ, 3σ] (c) Use of [-8σ, 8σ]

Fig. 6: Difference of outputs with different N values

In this case, our results are not as good as the one presented in the website. The algorithm that ours is more similar to is the *Euclidean distance* algorithm, which is very bad rated. The amount of false negatives is a definite factor to explain this poor performance, as we have several misdetections. Although the false positives rate and the specificity is overall good, it is very far from the performances performed by the best algorithms. Actually, these values become better with the usage of wider ranges of standard deviation, but on expenses of a dramatic decrease on performance regarding parameters like recall or precision. For the α values tested, there is no significative difference of performance when dealing with the same N.

VI. CONCLUSIONS

Threshold values between 40 and 50 are good enough to perform foreground mask with a good precision, specially regarding the objective of not getting false positives. Avoiding false negatives results a more complicated task, because of ghosts and noise, but can be improved with the appropriate employment of adaptation speed. This last parameter also helps to improve in the previous case.

VII. TIME LOG

A. Generation of Foreground Segmentation Mask

For the segmentation mask, the simple case using a threshold, the time spend goes as follow: To write the code and run the first test we needed around one hour. For running trails on the threshold optimum value we spent thirty minutes.

B. Progressive update of background model through selective running average

The time spent to create the code for this task was the lowest. We needed around thirty minutes to change the code. However, the trails took us around two hours, since we wanted to analyze the results of all test videos under several alpha before any conclusion.

C. Extension of Frame Difference using other Color Spaces

Coding for segmentation using color space took us around one hour for coding. Analysis of the results under different color spaces and threshold, took us one more hour.

D. Advanced Background subtraction (Single Gaussian)

For the last task of this work we invest around 12 hours, reading the documentation, running code trails and testing. First five hours were used for coding first approach, attempting to implement algorithm matrix wise. Then, due to usability, we changed to a friendly point wise method, performed trials with several parameters and correct our algorithm after consulting different references. This took us close to four hours. The rest two hours were spent on the store analysis and evaluation 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.