École Polytechnique Montréal
Génie Informatique

INF6804 : Vision par ordinateur
Présenté à Guillaume-Alexandre Bilodeau

Laboratoire 1

Travail remis par :
XING, Jin Ling- #1915481
MICHÉA, Luc - #1921803

26 Janvier 2019

1 Introduction

In this lab, we are going to use two different techniques of foreground/background video segmentation. We want to see how efficient one technique is compared to another on different cases. So first, we will explain the two different methods, how they work, what they do. Then, we will set down some requirements in which the two methods should work and assume which one will work better in those conditions. Finally, we will test our methods in those condition, evaluate the results and analyse them.

2 Identification and explanation

2.1 background subtraction

First of all we want to use a method of background subtraction. The background subtraction is the first technique we could have thought of when we want to do a foreground/background segmentation. At first, the idea was to extract what is moving in a picture at the time *t+1* from subtracting the picture at the time *t*. From this, we developped a lot of different techniques, and we are going to use the Gaussian Method for this laboratory.

The Gaussian Method is based on the following principles :

- First, we need to have some pictures that represents the background of the video as a sample in which we can base ourselves for the rest. Then, we create some Gaussian distributions that represents every pixels of the picture.

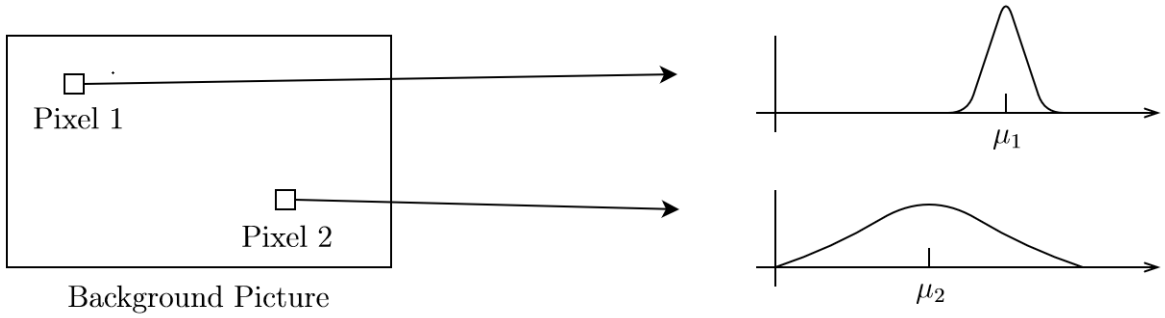

Fig. 1 – Illustration of the first step

- Then, from the new pictures, we can extract the background from a simple comparison, the aim of that comparison is to know if the new pixel is inside the Gaussian or not, with a certain threshold. If it's not inside the Gaussian, then it's not background.

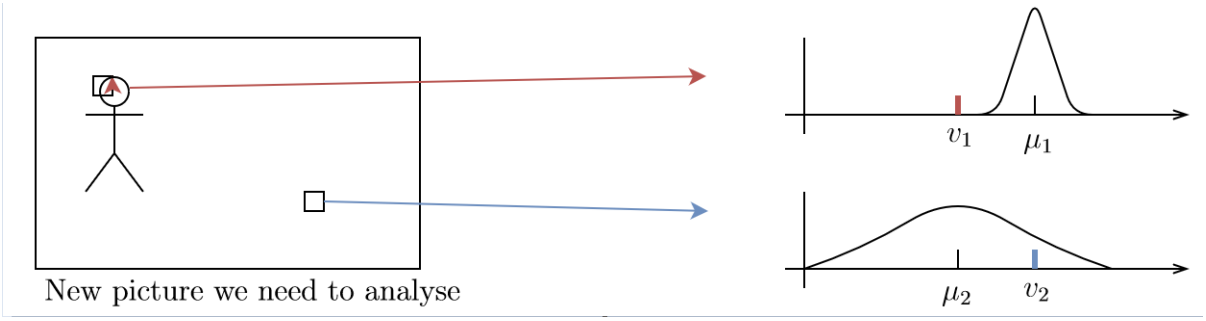

Fig. 2 – Illustration of the second step

Here, for example, we can see that the value of the Pixel1 (v_1) is out of our Gaussian of Pixel1, then it will be considered as Foreground. On the other hand, the value of the Pixel2 (v_2) is inside the Gaussian of Pixel2, so we can deduce it is Background.

2.2 object detection by classification

With time, some more complex and more precise techniques were created. The purpose of the object detection by classification techniques are different from the Gaussian Method. The true aim of the Gaussian Method is to do a foreground/background segmentation. The aim of the object detection by classification is different, it is to tell if objects in pictures are something or not. For example, if we are looking for humans, the algorithm will tell us if an object in a picture is an human or not. Those algorithms are slow if we don't use Edge boxes, which are a way to detect objects in a picture. During this Lab, we are going to use the HOG (Histogram of Oriented Gradients) algorithm. It is based on the gradient of a picture. Basically, the Gradient of a picture is a Vector going from White Pixels to Black Pixels.

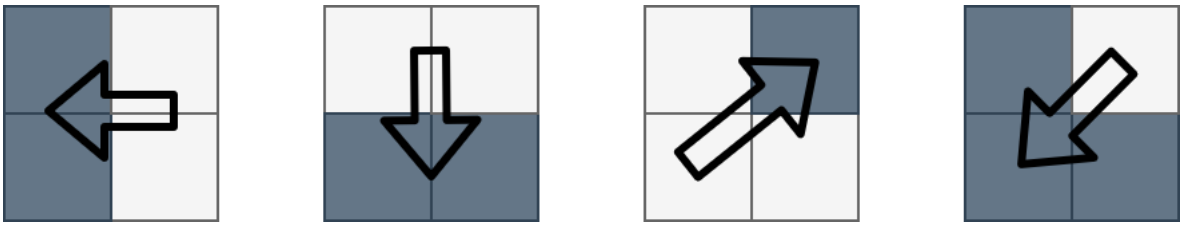

Fig. 3 – Illustration of a gradient of a picture

Then, from those Gradients we extract the different angles and we create an histogram containing the angles describing the object we want to detect. Finally, we develop a neural network that works to understand if a new histogram describes the object we want to detect or not. (Most of the time, the approximation on the angles are of around 8 degrees.)

2.3 Specific cases

So, during this lab we want to create certain situations that will challenge the two algorithms we will try to guess which one is best in which case and then we will do some experiments to either prove our hypothesis or make a discussion why it didn't work.

- Complex Background :
  In this case, the camera is not moving but the background is complex of some sort. The aim of this background is to be complex enough to trick the HOG algorithm to think there's people (what we want to detect) where it's only the background. We can conclude that in this case, the simple Gaussian will be better at detecting the people in the picture (cause they will be different than his background). The idea of this experiment went from the fact that if we want to detect cars passing on a road to determine the flow of cars and we have a car parked on this same street ; then the HOG algorithm will tell you there's a car passing on this road every frames because of the parked car. Whereas the simple Gaussian algorithm will have the car in his background and will only tell the true number of car passing. This example is the same idea but exploited differently.

- Camera moving :
  In this case we want to see if the algorithms can adapt themselves to a situation where the camera is moving during the period of the detection. In this case we can guess that the best algorithm would be the HOG detection. For the simple reason that the simple Gaussian can't adapt itself to movement. Even if we coded some parts that made the simple Gaussian adapt change the mean and variance of it overtime to adapt itself from the change of luminosity, the adaptation would be too slow for the camera moving because the pixels will be completely different from one another. The HOG method on the other part can really adapt itself to background change because it's not based on the pixels themselves but more on the shape of things. So, the HOG may not be able to detect a distorted shape from the camera movement, but will be able to detect the shape when the camera is more stable again.

- Low contrast :
  In this case the camera is not moving but the background and the foreground have a low contrast. We really think that this will challenge both of our methods in detection. The results of this experiment is hard to predict. We although expect that the HOG method would be more precise, because if the background pixels are not really different from the foreground pixels, then the simple Gaussian is not going to detect it has foreground. On the other hand, if the HOG method has sufficient elements it could detect the shape.

3 Experiments

3.1 Data-set used

The data-sets we used are from the website "http ://changedetection.net", we used only the 2012 files. The data-sets have several videos files. In each video file, we will use images from *"input"* file to do the detection, and use the *"ground truth"* to help us evaluate the detection results.

- Complex Background :
  The perfect example to show this has been seen in class. It is the *"PETS2006"* testing file. As we have seen in class, in this example, the HOG algorithm sometimes detect human where's there is none.

- Camera Moving :
  In the *"CameraJitter"* file we can find a *"Sidewalk"* file that seems to be perfect to measure the detection of pedestrian with each methods when the camera is moving.

- Low Contrast :
  In the datasets, we can see some Thermal videos, one of those seems really interesting and is the *"Park"* video. It is really interesting because the colors of the background and the colors of a pedestrian are really close from one another.

3.2 Evaluation criteria

The Evaluation Criteria is something we use to say if our method works well enough to do the Foreground/Background segmentation. As a first impression we can compare visually the ground truth images and the results images we got. This is mainly used to set roughly the parameters of a method.
Then we can use a much more precise metric, that is the ***Intersection and Union ratio*** which is a really common metric used in detection. We will explain it using a couple of graphs. First, what is the ***Intersection*** and what is the ***Union*** :

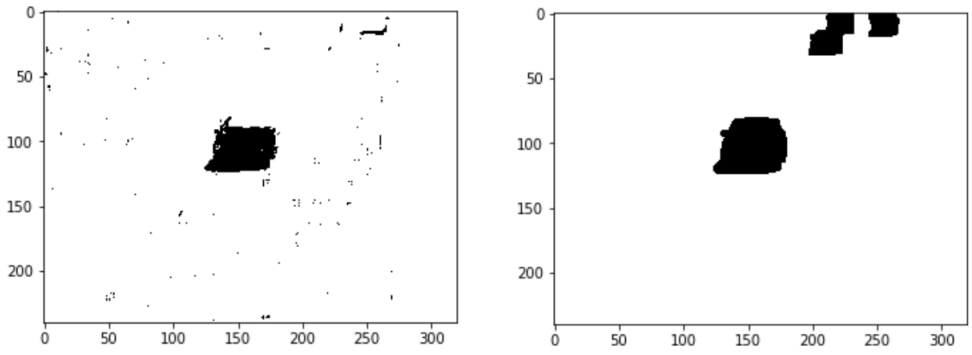

Fig. 4 – On the left : what we detect, on the right : the groundtruth

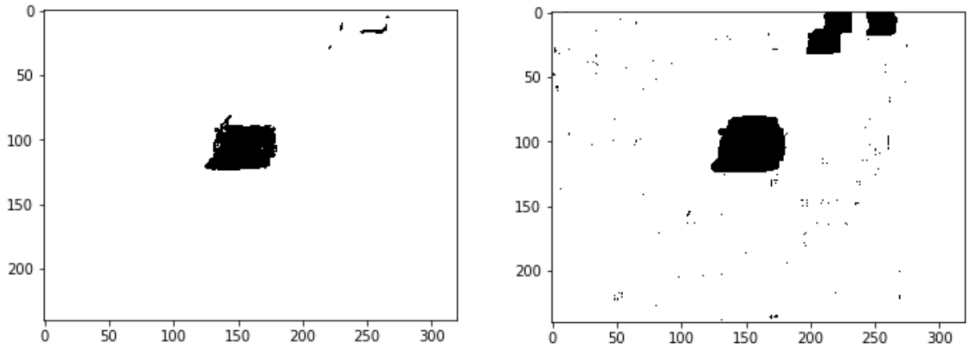

Fig. 5 – On the left : the intersection, on the right : the union

Then, if we have a picture and it's ground-truth, we can say that the intersection represents the number of pixels we detected that are in fact pixels of the object and the union represents the numbers of pixels that were not detected and the number of pixels that were detected when they shouldn't have. In a way, we can represent some cases and values in the following graphs :

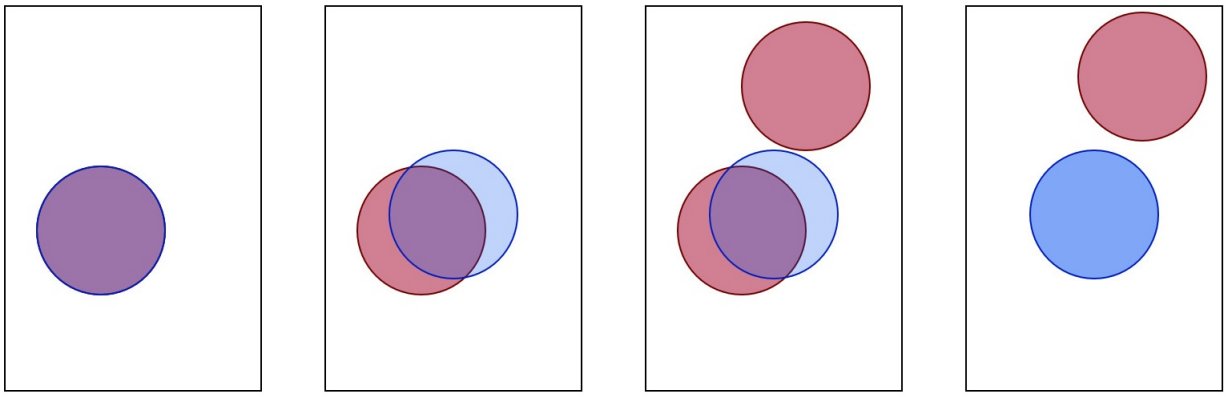

Fig. 6 – Illustration of some values of our criteria

On the previous picture, the red part is what we detect, the blue part the ground-truth.
All of this can be implemented really easily in Python with the following code :

```
####
##  evaluation criteria
####
imlist_crit_inter=[]
imlist_crit_union=[]
crit_result=[]

for p in range(0,20):
    inter_im_test = np.zeros((height, width), np.int)
    union_im_test = np.zeros((height, width), np.int)
    im1=imlist_out[p]
    im2=imlist_gt[p]
    sum_inter=0.
    sum_union=0.
    for j in range(0, height):
        for k in range(0, width):
```

```
            inter_im_test[j,k] = im1[j,k]&im2[j,k]
            union_im_test[j,k] = im1[j,k]|im2[j,k]
            sum_inter=sum_inter + inter_im_test[j,k]
            sum_union=sum_union + union_im_test[j,k]
    crit_result.append(sum_inter/sum_union)
    imlist_crit_inter.append(inter_im_test)
    imlist_crit_union.append(union_im_test)
```

4 implementation of the methods

First of all, for both the methods we have an import part in the beginning, which allows us to have both the images of the file we want to proceed detection on and the different libraries we need to use. Also we set the information of the images : the height and width of images, the number of images. These information will be used in the rest of the implementation.

4.1 Method 1 : Single Gaussian Method

4.1.1 Implementation

For this method we need to calculate the mean and variance for pictures that represents the background. We set two matrices of zeros "sum_im" and "power_im" and filled the sum and power values of the same pixel locations of each image to the "sum_im" matrix and "power_im" matrix.

```
M = 100  #number of images we used to initialize mean and variance
sum_im = np.zeros((height, width ,RGB), np.float)
power_im = np.zeros((height, width ,RGB), np.float)
mean = np.zeros((height, width ,RGB), np.float)
variance = np.zeros((height, width, RGB), np.float)
for i in xrange(0, M):
    im = imlist[i]
    for j in range(0, height):
        for k in range(0, width):
            for p in range(0, RGB):
                sum_im[j,k,p] = sum_im[j,k,p] + im[j,k,p]
                power_im[j,k,p] = power_im[j,k,p] + np.power(im[j,k,p], 2)
mean = sum_im/M
variance = power_im/M - np.power(mean, 2)
```

Then, we set the value of sensitivity and calculated the foreground by Gaussian distribution. What we got of foreground will be a matrix filled bool type of values. When we wanted to change the TRUE/FALSE to 1/0 values of matrix, we created a current all zeros matrix and did add operation to the foreground matrix and current matrix then we got the binary foreground. Because the calculation is huge for all the video frames, we chose 20 images randomly to calculate. Considering the RGB channel, we set if the RGB channel has more one value greater than "1", it should be the white pixel. Here we had a problem, when we used the intersection/union criterial, we will compare the background except the foreground. But what we want is to compare the foreground with ground truth. Therefore, we set if the RGB channel has more one value greater than "1", it will be the black pixel.

```

im = imlist[200]
n = 10 #Sensitivity
foreground = np.abs(im - mean) > n * np.sqrt(variance)
curr_matrix= np.zeros((height, width, RGB), np.int)
bin_fg = foreground + curr_matrix
bin_fg2 = np.zeros((height, width), np.int)
im_gt2 = np.zeros((height, width), np.int)
bin_gt =  np.zeros((height, width), np.int)
xl = []
imlist_out = []
imlist_gt = []

for i in range(0, 20):
    x = random.randint(0, N-1)
    xl.append(x)
```

```
31
32      im = imlist[x]
33      im_gt = imlist_ground_truth[x]
34
35      foreground = np.abs(im - mean) > n * np.sqrt(variance)
36      curr_matrix = np.zeros((height, width, RGB), np.int)
37      bin_fg = foreground + curr_matrix
38
39
40      bin_fg2 = np.zeros((height, width), np.int)
41      im_gt2 = np.zeros((height, width), np.int)
42
43      for j in range(0, height):
44          for k in range(0, width):
45              bin_fg2[j,k] = bin_fg[j,k,0] + bin_fg[j,k,1] + bin_fg[j,k,2]
46              if bin_fg2[j,k] > 0:
47                  bin_fg2[j,k]=1
48              else:
49                  bin_fg2[j,k]=0
50
51      for j in range(0, height):
52          for k in range(0, width):
53              im_gt2[j,k] = im_gt[j,k,0] + im_gt[j,k,1] + im_gt[j,k,2]
54              if im_gt2[j,k] > 0:
55                  bin_gt[j,k]=1
56              else:
57                  bin_gt[j,k]=0
58
59      imlist_out.append(bin_fg2)
60      imlist_gt.append(bin_gt)
```

The last part of implementation is evaluation criteria. The detailed description is at part 3.2.

4.1.2 Parameters

M = 100 the number of images we used to initialize mean and variance
There has some objects always moving in the videos, it's better to choose more than one image to set the background. Maybe try to find somewhere in the video where it's not moving much too.

n = 10 Sensitivity
The sensitivity is a really important value as it changes a lot on what we detect in the end. If we set the sensitivity too low we may end up detecting some background that was subject to noise. But, if we set the sensitivity too high, we will maybe end up not detecting some things we want to detect. For this reason, we used different sensitivity numbers to do the test in the chapter 5.2 for camera moving and chapter 5.3 for low contrast. When we set the "sensitivity = 10", the result is bad, when we set the "sensitivity = 5", the result become better, when we set the "sensitivity = 2", we got the best result, when we set the "sensitivity = 1", single gaussian method will detect the background and the result is not what we want.

4.2 Method 2 : HOG

4.2.1 Implementation

After the common importing part, we initialized the HOG detector AND got the default people detector from "hog.setSVMDetector".

Then, for the random 20 images, we got the rects and weights from "hog.detectMultiScale". By means of comparing two methods in three cases and using the intersection/union criteria, we need to set the detected rectangles as foreground. Although we thought all the rectangles are the object we detected is not really accurate, we need to compare two methods in the same criteria standard(intersection/union criteria).

```
61 #Multi-scale detection and display.
62 for i in range(0,20):
63     x = random.randint(0, N-1)
```

```
    xl.append(x)
    im = imlist[x]
    im1 = im[:, :, 0]
    (rects, weights) = hog.detectMultiScale(im1, winStride = (4,4), padding = (8,8), scale = 1.05)

    rects_list.append(rects)
    im_gt = imlist_ground_truth[x]

    bin_fg = np.zeros((height, width), np.int)
    im_gt2 = np.zeros((height, width), np.int)
    bin_gt =  np.zeros((height, width), np.int)

##Viewing
    for x, y, w, h in rects:
        cv2.rectangle(im, (x, y), (x + w, y + h), (0, 255, 0), 2)

    for l in range (0,len(rects)):
        pad_w,pad_h = int(0.15 * rects[l,1]),int(0.05 * rects[l,3])
        bin_fg[rects[l,1]:rects[l,1]+rects[l,3],rects[l,0]:rects[l,0]+rects[l,2]]=1

    for j in range(0, height):
       for k in range(0, width):
           im_gt2[j,k] = im_gt[j,k,0] + im_gt[j,k,1] + im_gt[j,k,2]
           if im_gt2[j,k] > 0:
              bin_gt[j,k]=1
           else:
              bin_gt[j,k]=0

    imlist_out.append(bin_fg)
    imlist_gt.append(bin_gt)
```

The last part of implementation is evaluation criteria. The detailed description is at part 3.2.

4.2.2 Parameters

This method doesn't have any parameters.

5 Evaluation/Analysis of the results

First, sometimes in our results, the criteria gives us zero. Most of the time in our results it means that we had nothing to detect, that the picture was only composed of background :

Fig. 7 – One of the pictures that give us a criteria of zero

Another issue we have from our code is that because we fill the rectangle in the HOG method, our results in our criteria are never really precise, then we can say that our results with the HOG method are perfects when the criteria is around 0.4 to 0.5 (from experiments we did).

5.1 Complex Background

For the complex background, what we expect is the Gaussian method to be really precise and the HOG method to find people in the background. For the Gaussian method we obtain this result :

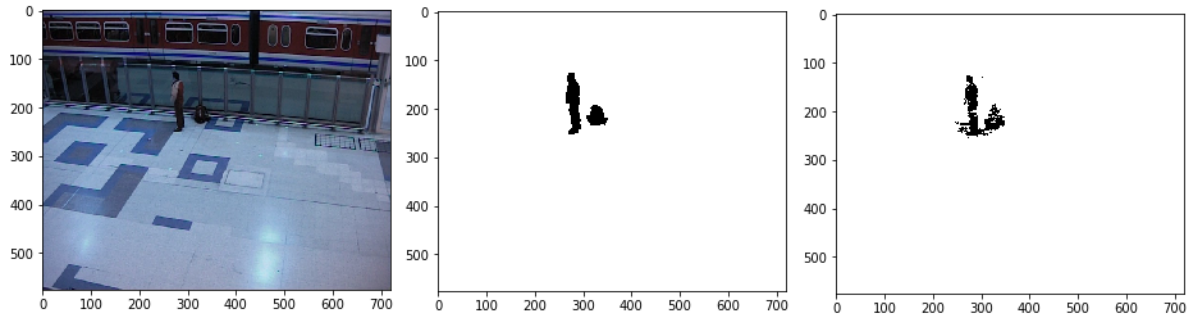

Fig. 8 – Results of the Gaussian method (1 : The picture, 2 : The ground-truth, 3 : The detection done)

Here, the result we have is really precise, the only thing we have in addition is the shadow of the person. We had other results :

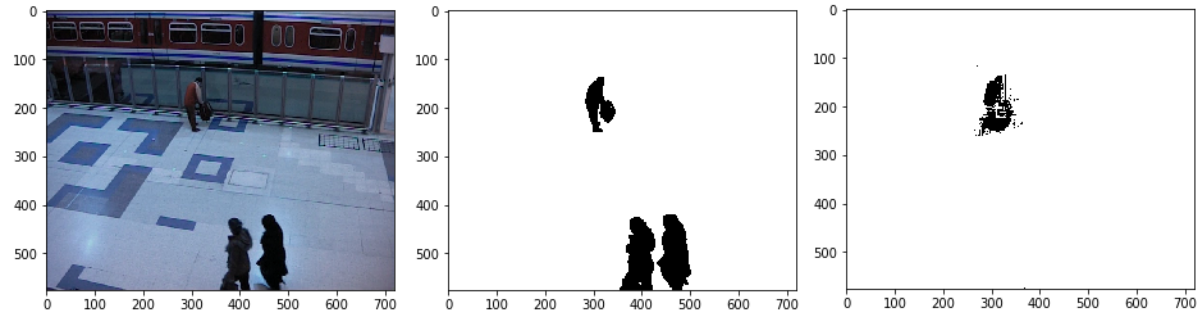

Fig. 9 – Results of the Gaussian method (1 : The picture, 2 : The ground-truth, 3 : The detection done)

Here, the result is less accurate, some peoples are not detected while some noise is detected. On the other hand, the HOG results are the following :

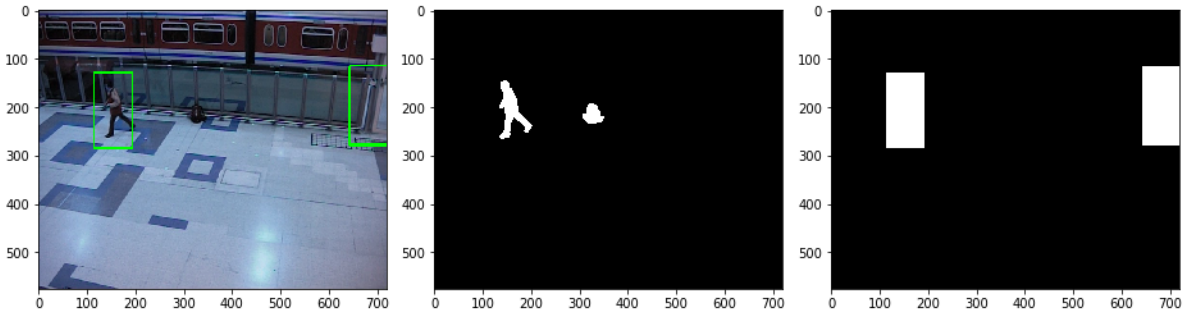

Fig. 10 – Results of the HOG method (1 : The picture, 2 : The ground-truth, 3 : The detection done)

Here, and in all the other pictures of the *PETS2006* data-set we can see that something in the background is detected as a person. Worst, sometimes some people are not detected while this part of the background is detected. Then we can say that the simple Gaussian is better than the HOG in this case. Our criteria can prove that even more :

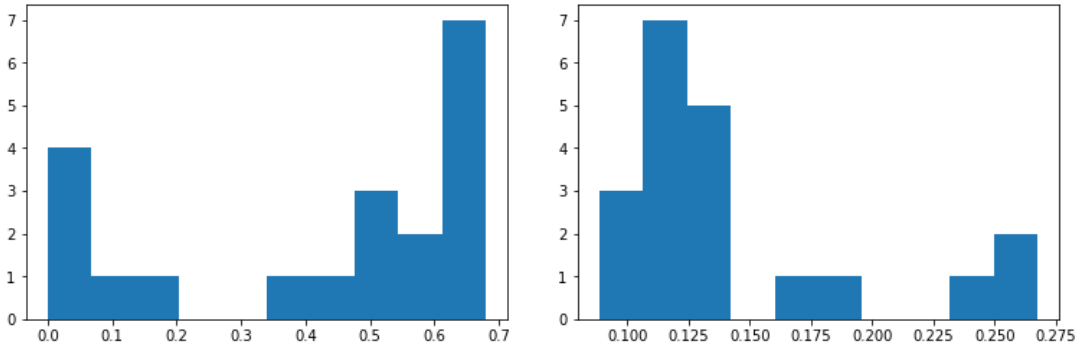

Fig. 11 – Histograms of our results on the 20 pictures. (Left : Gaussian, Right : HOG)

The results of the histogram shows that (if we don't take care of the zeros) the results of the Gaussian are way better than the results of the HOG.

5.2 Camera moving

On the *sidewalk* data-set the camera is sometimes moving sideways. It implies that the simple Gaussian method can't obtain good background results. If we use the method with a pretty high sensitivity, then we obtain only some noise that are detected. We can see that on the following figure :

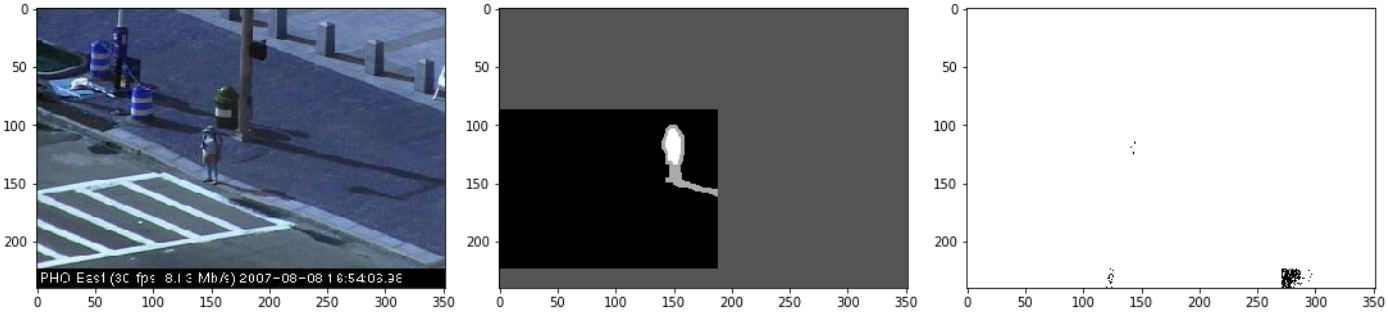

Fig. 12 – Results of the Gaussian method (1 : The picture, 2 : The ground-truth, 3 : The detection done)

As expected, we only detect some noise in the picture but we can't detect the pedestrian, for the only reason that our background is made of Gaussian approximations of pixels values and the pixels values are changing a lot because the camera is moving when we are trying to get the background. In a way, even if the camera wasn't moving when we tried to have the background, if the camera was moving for the rest of the detection, then when the camera moved, we would have detected the whole picture. Maybe we can evaluate the movement of the pictures using optical flow and change the simple Gaussian code to adapt itself from some movement.
For the HOG method we have some pretty different results. First we can say that it worked cause we detect the pedestrian in the picture. As we expected, because the HOG is not based on the values of pixels but the difference. We can obtain this result :

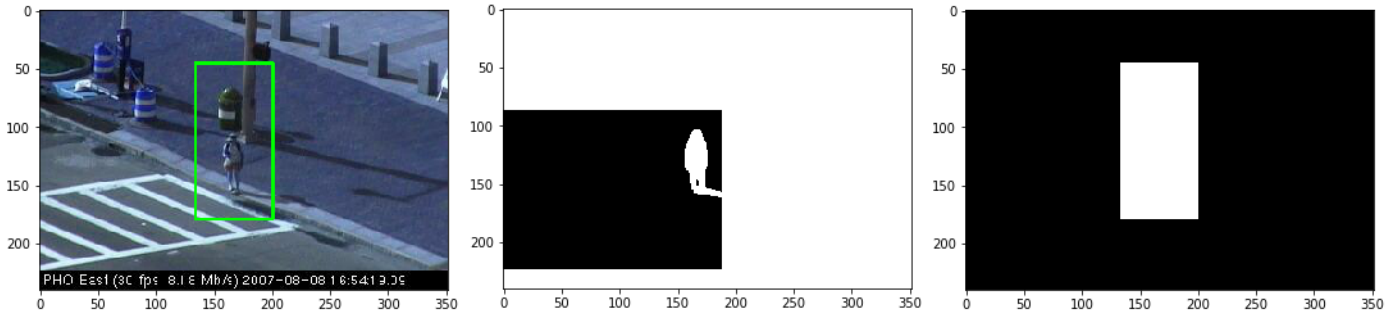

Fig. 13 – Results of the HOG method (1 : The picture, 2 : The ground-truth, 3 : The detection done)

Now, to compare both of the results, first we can say that in the simple Gaussian method we did not detect the pedestrian and in the HOG method we detected a pedestrian. Then, if we use the Histograms of our criteria results on the 20 different pictures we obtain this result :

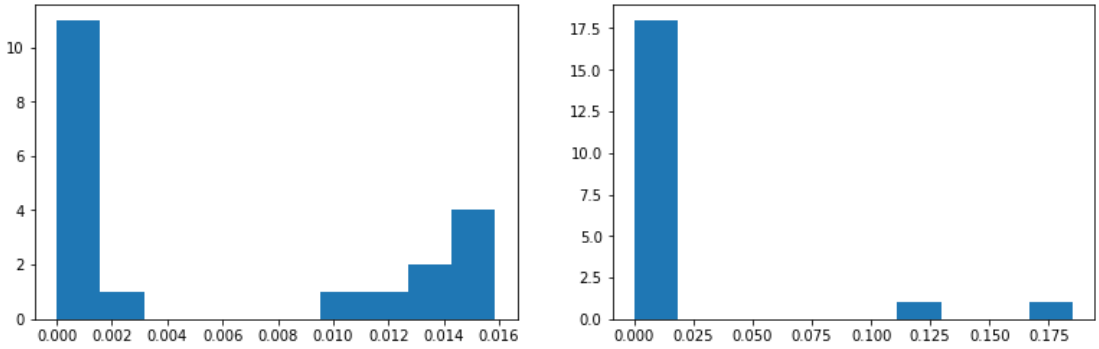

Fig. 14 – Histograms of our results on the 20 pictures. (Left : Gaussian, Right : HOG)

If we analyse our two histograms, we see that first, the results of the Gaussian method are really low compared to the results of the HOG method. Then we can say that both of them have a lot of zeros, this comes from the fact that there's a lot of pictures without any ground-truth in this data-set. Then, we can say that although the results of the HOG are better, we have seen a lot of pictures where it didn't detect the people.
We decided to change the sensitivity of the Gaussian Method, and we were really surprised to see that the results of the Gaussian became really good. This could be understood by the fact that even if the camera is moving sideways, the initial position is always the same. We can obtain the different results on the sidewalk :

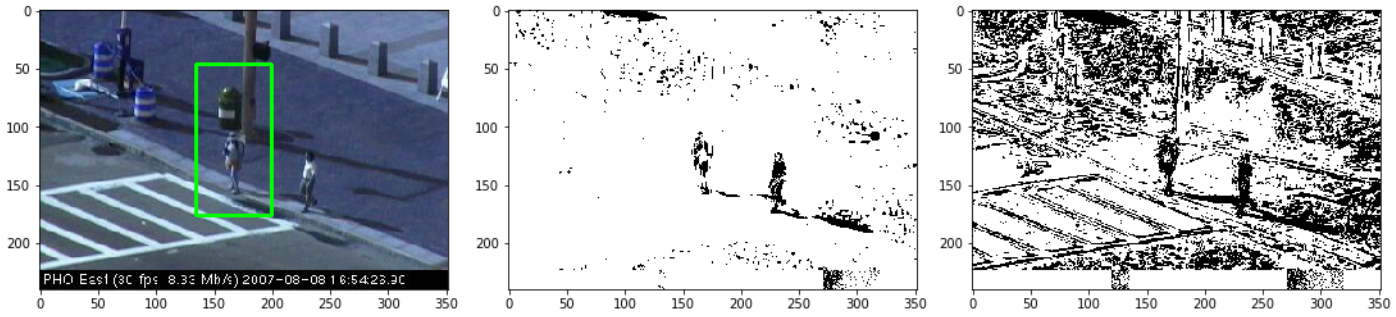

Fig. 15 – Results of the Gaussian method when changing the sensitivity

We can see in this picture that the HOG method didn't see the second pedestrian, though if the sensitivity of the Gaussian is set at a correct value, then both of the pedestrian are detected.

To conclude, we can say that maybe in this case the Gaussian method is better. But, we can say that usually in a case where the camera is moving from left to right to scan a place (security cameras for example) the simpel Gaussian would not have worked cause the background changes too much.

5.3 Low contrast

The data-set *"park"* is composed of thermal images and most of them are low contrast images. When we used the simple Gaussian method, we set the "sensitivity = 5", we can get the clear walking people from results, but the results we got have double images, it means the border between the background and foreground is not clear. We also tested the sensitivity's effect in the *"park"* data-set, we set the "sensitivity = 10" and we found the results are worse compared to the "sensitivity = 5". When the sensitivity is too high, some part of our foreground are not detected. The results we got for simple Gaussian method are the following :

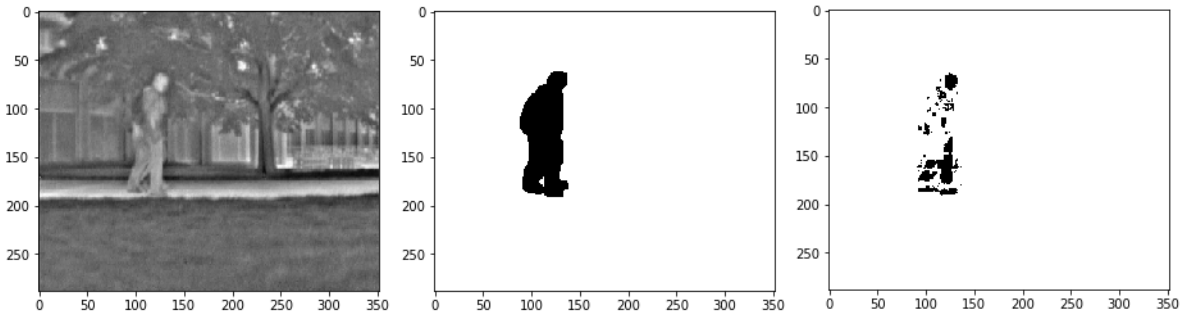

Fig. 16 – Results of the simple Gaussian method ("sensitivity = 10")(1 : The picture, 2 : The ground-truth, 3 : The detection done)

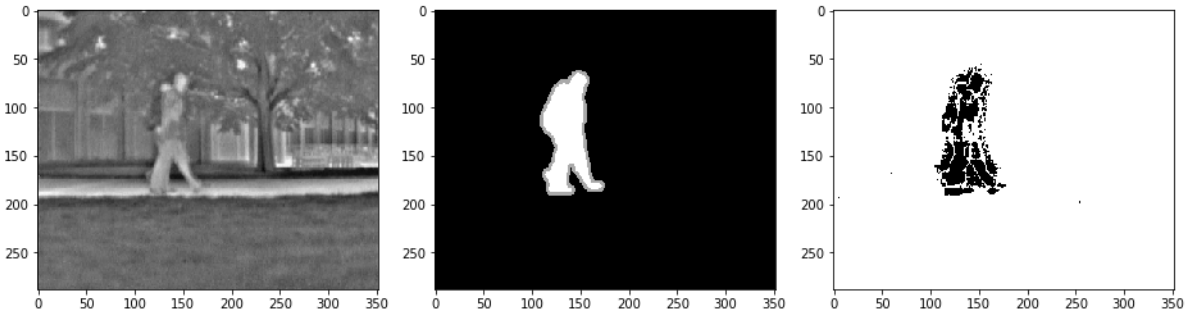

Fig. 17 – Results of the simple Gaussian method ("sensitivity = 5")(1 : The picture, 2 : The ground-truth, 3 : The detection done)

As we thought, the Simple Gaussian will struggle with low contrast pictures. Because the *"park"* data-set are thermal images, it's only gray and black colors, the color of people are really similar to the color of the background (low contrast). Then the Gaussian method can't differentiate noise and foreground.

The HOG method for low contrast images have better performance compared to the Gaussian method. The HOG results are below :

Fig. 18 – Results of the HOG method (1 : The picture, 2 : The ground-truth, 3 : The detection done)

The HOG method works well in this case, because the color differences of foreground and background don't affect the results much. In fact, it would affect the HOG method only if the background and foreground were of the exact same color (no gradient).

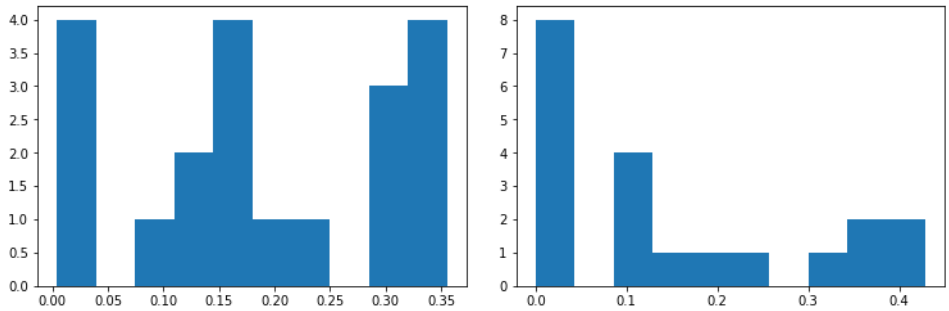

Fig. 19 – Histograms of our results on the 20 pictures("sensitivity = 10"). (Left : Gaussian, Right : HOG

When we analyse the two methods, we can see the HOG works well in the low contrast first. The simple Gaussian works not really good compared to the other two cases. We changed the sensitivity of the simple Gaussian to improve the performance. When the "sensitivity = 10" , as the histograms above, the average results are around 0.1 to 0.35, after we changed the "sensitivity = 5", the average results of histograms are around 0.2 to 0.4. But it represents nothing compared to the really good results of HOG (with our criteria, HOG is perfect around 0.5).

6 Conclusion

To conclude, we can say that in most cases we proved or almost proved what we wanted to show. We had a lack of conclusion mainly on the sidewalk data-set because the camera was not moving enough to show our hypothesis. We were through the entire Lab surprised by the results shown by the simple Gaussian method compared with the results expected from the HOG. In the end, the mistakes made by the HOG are a great counter part to this method. We really think that the complex background and the low contrast experiments supported our hypothesis. We also think that our HOG results could have been improved a lot. First we could have changed the way we detected with HOG, in our code we just filled in the rectangle, whereas trying to find the person detected inside (using superpixels for example). Second, we could have used a different criteria with the HOG method, as our criteria didn't seem to be accurate enough for this method.

7 Code Library and Parts used

we used opencv 3.4.1 to do most of our things in the code. The code is wrote by ourselves for the most part. We only used one help online on a forum :

https ://stackoverflow.com/questions/30230592/loading-all-images-using-imread-from-a-given-folder

This forum helped us understand how to import pictures from a file. The rest of the code was written by ourselves.