LAB 2 : Blob extraction and classification

Charlotte Assemat, Camille Maurice

April 23, 2015

IntroductionThe goal of this assignment was to implement a system to extract some blobs from a video and then assign them to classes.

1 Implementation details

1.1 Blob extraction

The foreground segmentation
We use a built-in function of OpenCV to get the FG segmentation, based on the mixture of Gaussian with 3 Gaussians, that computes the foreground mask rapidly enough.
But the mask isn't optimal as it is for what we need it to do:

- there's a lot of small noise (it's very sensitive to the flickering of light in *mh.mpg*);
- and the areas that we need to detect as one blob are too sparsily detected as foreground to give one full-size blob (we used to get several little ones instead).

This is why we implemented an opening with some erosion to get rid of (most of) the noise, then some wider dilating to "fill out" the connected regions where some points are lacking.

We use the function `floodFill()` to get the connected components. In the mask we have, the foreground is marked as 255; but for this function, the binary input image will stop at non-zero values, so we invert the mask.
Then we can feed it to the floodFill function, that will perform the grass-fire algorithm with a seed point.

The grass-fire algorithm
The original algorithm is as such:

```
textbfFireGrass(seedPoint p, Image i)
for each neighbor n
        if n is an object pixel // = if it is grass
                if it is unmarked // = unburnt
                        you label it and mark it// = you burn it
                        you call FireGrass(n,i) // = spread the fire on neighbor n
                if it is marked // = burnt
                        terminating condition// = it has already been taken care of
        else (it's not an object) // = it's a river
                terminating condition// = the fire doesn't spread over a river
```

We chose to implement the grass fire algorithm in a sequential way to avoid any memory issues.
To follow the original algorithm, we'd have had to make a list to keep all the neighbors that we had to visit, but as we scan the whole image, we know we won't miss a pixel, so we can just go through the pixels sequentially. (If we needed to know the connected component at a certain point in the image, then we would have needed the list.)
So we scan the image and each time a pixel hasn't yet been visited (we use one mask that is updated

at each call, so we know this) we call the function on it, that updates the mask with the newly-burnt pixels, and gives us the bounding blob.
We call the function with the 8-connectivity option to compensate for the sometimes-sparse foreground mask (same reason why we did the opening).
Before adding the blob to the list, we check its size is above the minimum defined in the header.

1.2 Blob classification

To find suitable parameters for the aspect ratio classifier we will use a training set. The training sets used are : http://lear.inrialpes.fr/people/marszalek/data/ig02/ and the masks provided in the additional content of this lab.
`MatLab script`
Extraction : It reads all the mask frames in the training set folder, then resizes them to the size of the largest blob.
Compute AR : Then it goes though all the extracted blob images to compute their aspect ratio. At the end it returns the mean and the standard deviation of the aspect ratio of a given class.
Using the MatLab script we found out the following parameters.

```
For person                          For car
mean_aspect_ratio =  0.3645         mean_aspect_ratio =  1.8537
std_aspect_ratio  =  0.0621         std_aspect_ratio  =  0.6112
```

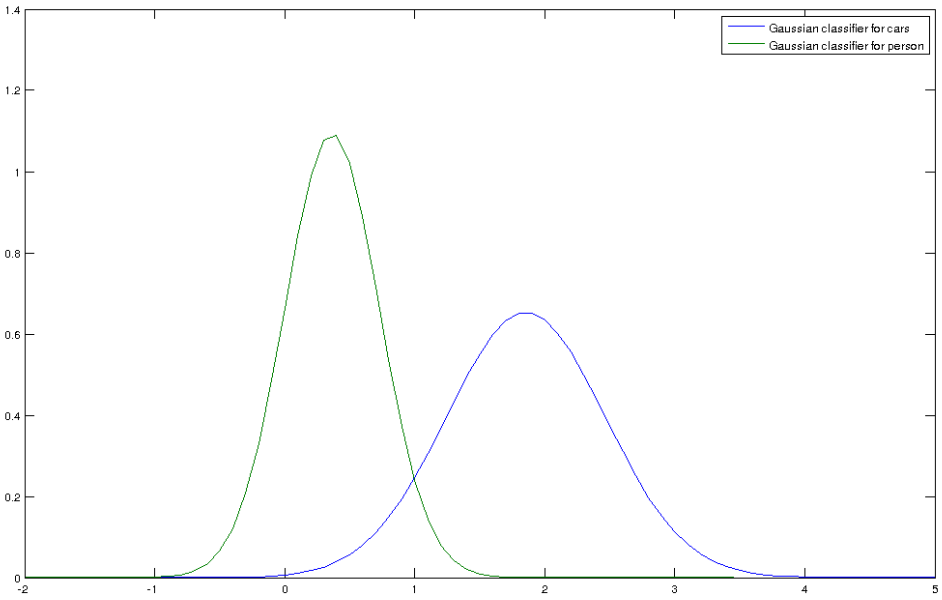


Figure 1: Gaussians for person and car classes using the aspect ratio feature. Probability wrt AR

The standard deviation for the cars is slightly too high, maybe we should remove some of the data from the training set. Some picture of the training set are not relevant to detect cars
`classifyBlobs`
Goes through all the detected blobs of the current frame. For each blob the aspect ratio is computed. Then the gaussians corresponding to each class are computed. Once we found the gaussian that gives the higher probability with respect to the aspect ratio, we have the corresponding class of the blob.
To found probabilities, we compute the image of the AR of the considered blob by the following probability density function :

$$f(\mathrm{AR}, \mu_{class}, \sigma_{class}) = \frac{1}{\sigma_{class} * \sqrt{2*\pi)}} * \exp \frac{-(\mathrm{AR} - \mu_{class}).^2}{2 * \sigma_{class}.^2}$$

We compute the image of the aspect ratio as many as the number of classes. For each pair of mean and std for each class.
For example an aspect ratio of 0.5 is more likely to be a human, and an aspect ratio of 2 is more likely to be a car.
This is not surprising usually people are at least half as much large than tall. And looking at a car from the side view, this is often twice as much large than tall.

(a) Person blob, AR = 47/97 = 0.48

(b) Car blob, AR = 218/107 = 2.03

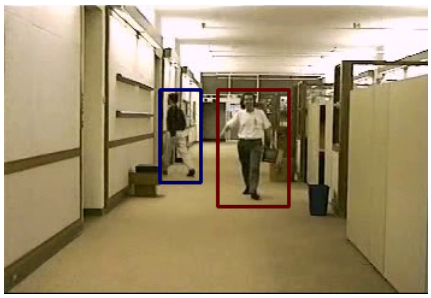
Figure 2: Aspect ratio comparison

2 Results and discussion

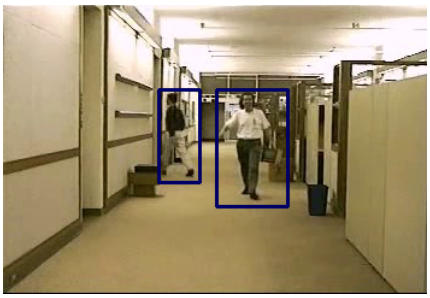
Once we had a running code, we looked at the result. We noticed many person classified as cars, and problem due to small overlapping blobs. We decided to use a larger training set for people. And then update our mean and standard deviation for the person class. We combined "..." and "..." masks set.

```
New values for person class
mean_aspect_ratio =  0.4067
std_aspect_ratio  =  0.1449
```

With this values it performs better. We can see it out of the following example from the "mh.mpg" video.



(a) Gaussian classifier uses old values, person detected as a car (red rectangle)



(b) Gaussian classifier uses new values, person detected as a person (blue rectangle)

Figure 3: A larger training set is used to compute new std and mean.

Another detected problem, the overlapping blobs problem. To avoid miss classification it would be interesting to merge overlapping blobs into a bigger one.

Another problem is the occlusion. The mask of the object is deformed, the 2 foreground objects form a bigger component. We do not have any solution to this problem.

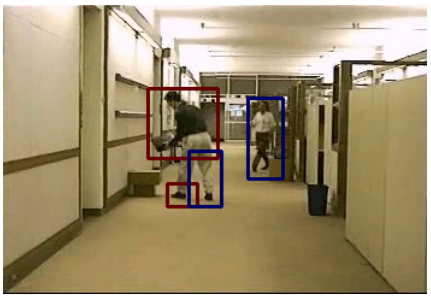


Figure 4: Overlapping blobs and miss classification

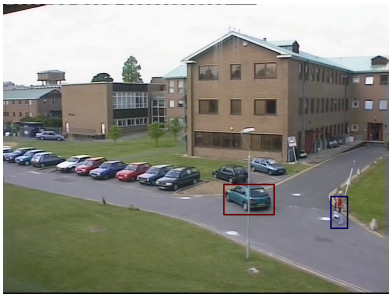


Figure 5: Cars and pedestrian good detection

3 User Manual

Once the zip-file extracted, specify the video you want to use in main.cpp

```
#define INPUT_VIDEO "yourvideo"
```

use `make` to compile. Then an executable file called main is created. Run it. Compilation issue : Depending on which computer we tried the program we had a problem with these lines of code in the main.cpp:

```
    subtractor.nmixtures = 3;
    subtractor.bShadowDetection = true;
    //subtractor.set("nmixtures", 3);
//subtractor.set("detectShadows",1);
```

If you are in these case comment the first 2 lines and uncomment the 2 last lines of the snippet of code above.

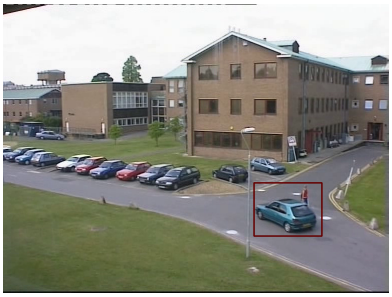


Figure 6: Problem of occlusion