# LAB 2 - OBJECT DETECTION AND CLASSIFICATION

APPLIED VIDEO SEQUENCES ANALYSIS

Zsófia Sára Budai and Juan Manuel Peña Trapero
{budai9902, juanmaptcg}@gmail.com

## I. INTRODUCTION

This laboratory is composed of 4 tasks:

- Task 0: Study the sample code
- Task 1: Blob Extraction
- Task 2: Blob Classification
- Task 3: Stationary Blob extraction and classification

## II. METHOD

This section describes briefly the methods we have implemented.

### *A. Void object detection and classification using template (Task 0)*

In this task we have performed an in-depth analysis of the code provided. This step is very important since all the implementations made for this lab will be based on this code.

The general operation of the program is similar to the one used for lab 1 of this course: within the main function the following steps are performed:

- Initialisation of arrays to store images and the binary masks obtained.
- New: vectors are initialised to store all the objects of type **Blob (Binary Large Object)** as they are detected.
- Initialisation of strings storing the input path of video sequences and output results.
- Creation of the object in charge of performing the segmentation by means of createBackgroundSubtractor-MOG2.
- Loop traversing each of the frames of the sequence to be analysed, nested inside another loop that traverses all the selected videos.
- Initialisation of the ***learningrate*** value for the operation of the segmentation algorithm. In the case of this laboratory, the value selected is -1, which means that the optimum value is automatically selected when updating the background model on which the segmentation is based.
- Steps necessary for the extraction of the *blobs*: extraction of the foreground binary mask $\rightarrow$ application of the blob extraction algorithm $\rightarrow$ post-processing of the blobs $\rightarrow$ classification of the blobs obtained.
- Finally, the function *ShowManyImages* is called to display the results obtained as a whole as the frames are processed. At this point it is noteworthy that among the functions provided is the function *paintBlobImage* which is crucial for debugging and code review as it is able to show the blobs that the algorithm is extracting and show the results of the live classification. In the results section these functions with their different options will be used for the analysis of the results.

### *B. Blob extraction (Task 1)*

The goal of this task is to implement the Grass-Fire algorithm [1]. It basically works by assigning a ”fire” to each pixel in the image, which spreads to neighboring pixels in a sequential manner, similar to how fire spreads in dry grass. The algorithm terminates when all pixels in the connected region have been visited by the fire. The resulting output is a set of connected regions or components in the image, which are going to be used in the following tasks.

In the given documentation in Moodle the algorithm is explainig with two different approaches: sequential and recursive.

- In the **sequential** approach, the algorithm visits each pixel in the image one by one and spreads the fire to its neighbors. This process continues until all pixels in a connected region have been visited. This approach is simple to implement and is memory-efficient, but it can be slow for large images since it processes pixels one by one.
- In the **recursive** approach instead of visiting the neighboring pixels one by one, it calls the same function recursively for each neighboring pixel. This process continues until all pixels in a connected region have been visited. The recursive approach is faster than the sequential approach for large images since it processes pixels in a parallel manner. However, it requires more memory to store the function calls on the stack, which can limit its practical use for very large images

### *C. Blob classification (Task 2)*

For this task an Aspect ratio feature based classifier is introduced: The aspect ratio of a blob is the ratio of its width to its height. This feature can be used to distinguish between different types of objects, as some objects (e.g. cars are usually wider) tend to have a certain aspect ratio while others (e.g. person is narrower) have a different aspect ratio. To implement the blob classification routine, we will need to extract the aspect ratio feature for each blob in the input list and use the mean and variance parameters for each class to calculate the likelihood of each blob belonging to each class. The class with the highest likelihood will be the predicted class for each blob. For this calculation we are going to used two types of distances as shown in the listing below.

It is important to understand and compare the obtained results that the empirical distributions that are used for this lab. In the Figure 1 we can see how the distribution of the different objects that are classified is overlapping with each other. This means that there will be certain level of uncertainty for all the obtained classifications as the distribution are easily confused

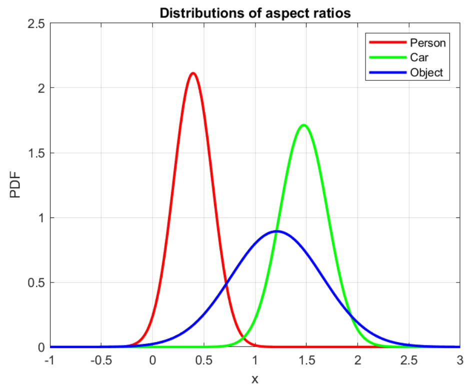

Fig. 1. Distribution of the 3 classes detected by our blob classifier.

## D. Stationary Foreground Extraction (Task 3)

The algorithm described in section 3.1 of the paper[2] is a method for detecting stationary foreground objects in a video sequence. The algorithm is based on the observation that the pixels belonging to stationary foreground objects do not change over time, and therefore have a high persistence in the foreground mask.

First the algorithm computes the foreground mask for the current frame by subtracting the background model from the current frame. The background model is updated using the learning Rate parameter, which controls the rate at which the model adapts to changes in the scene.

Compute the foreground and motion history images using the foreground mask. The foreground history image records the time since each pixel was last classified as foreground, while the motion history image records the motion of each pixel over time.

Compute the persistence of each pixel in the foreground mask by computing the ratio of the number of times the pixel has been classified as foreground to the total number of frames processed so far. Pixels with a high persistence are likely to belong to stationary foreground objects.

```

float ED(float val1, float val2)
{
  return sqrt(pow(val1-val2,2));
}

float WED(float val1, float val2, float std)
{
  return sqrt(pow(val1-val2,2)/pow(std,2));
}
```

## III. IMPLEMENTATION

## A. Blob Extraction (Task 1)

The implementation for this task is divided into 3 routines:

*1) extractBlobs:* The function *extractBlobs* takes in three arguments:

- *fgmask*: A binary input image where the foreground pixels are set to 1 and the background pixels are set to 0. This image is used to extract the connected components or blobs.
- *fbloblist*: A reference to a vector of *cvBlob* objects. This vector will be populated with the detected blobs. Each *cvBlob* object represents a connected component and contains information about its bounding box, centroid, and area.
- *fconnectivity* An integer value that specifies the connectivity of the pixels. A value of 4 means that only the four neighbors (top, bottom, left, and right) are considered, while a value of 8 means that the four neighbors and their diagonals are considered.

The function extracts the connected components or blobs from the input binary image using the Grass Fire Algorithm. For each detected blob, a new *cvBlob* object is created and its properties such as bounding box, centroid, and area are computed. The *cvBlob* object is then added to the *bloblist* vector using the push_back method.

Internally, the function loops through each pixel of the input binary image aux, and if a foreground pixel (i.e., a pixel with value *255*) is found, it initiates the flood fill algorithm to extract the connected component or blob to which the pixel belongs.

The function *cv::floodFill* is used to fill the connected component starting from the seed point (i,j) with a specified color (64 is used), and to get the values of the bounding rectangle of the component through a structure that is updated by the function with the bounding rectangle of the filled region (*rect* reference).

After the connected component is filled, a new *cvBlob* object is created using the *initBlob* function with the blobId(initiallized to 1), rect.x, rect.y, rect.width, and rect.height values, which represent the unique ID and the bounding box coordinates of the blob. The blob object is then added to the *bloblist* vector using the push_back C++ method. Finally, the *blobId* is incremented by 1 to prepare for the next blob.

*2)* ***removeSmallBlobs****:* This function takes in a vector of Blobs called *bloblist_in* in, which represents a list of detected blobs. It also takes in two integer values *min_width* and *min_height*, which represent the minimum width and height that a blob must have in order to be considered ”valid” and not removed.

The function is expected to remove any ”small” blobs from *bloblist_in*, where a small blob is defined as one that has a width or height less than the specified minimums. The remaining blobs that meet the minimum size requirements are then added to a new vector called *bloblist_out*.

The function takes in *bloblist_out* as a reference parameter, which means that any changes made to *bloblist_out* inside the function will persist outside of the function as well. This allows the function to return the updated list of blobs that meet the minimum size requirements as an output parameter.

## B. Blob Classification (Task 2)

The ***classifyBlobs*** function is implemented for this task: it takes a vector of blobs as input and classifies each blob based on its aspect ratio.

The function uses the previously defined mean and standard deviation of aspect ratios of each object class (person, car, and object). It then iterates over each blob in the input vector, calculates the aspect ratio of the blob, and computes the Euclidean distance from each object class's aspect ratio model. In the initial version of the code we used the Weighted Euclidean Distance but it wasn't classifying the blobs properly.

The object class with the smallest distance is assigned to the blob's label. Finally, the modified blob is stored back into the input vector. The function returns 0 to indicate success.

Note that the implementation needed to be changed due to the lack of precision of the chosen datatype.

## C. Stationary Foreground Subtraction (Task 3)

For this task the function ***extractStationaryFG*** is implemented: it takes three input arguments, namely *fgmask*, *fgmask_history*, and *sfgmask*, which are all of the type *Mat*.

The matrix fgmask represents the foreground mask at a certain time instant, while fgmask_history represents the history of the foreground masks, and sfgmask is the output stationary foreground mask.

The function first calculates the numframes4static variable as the number of frames that need to be stationary (i.e., not changing) for a certain pixel to be considered as stationary. This is calculated based on two pre-defined constants: *FPS* (Frames Per Second ) and *SECS_STATIONARY*.

The function then updates the fgmask_history matrix based on the current fgmask. For each pixel in the fgmask, the function checks whether it is foreground or background. If it is foreground, the corresponding pixel value in fgmask_history is incremented by *I_COST*, while if it is background, it is decremented by *D_COST*.

After updating fgmask_history, the sfg mask is computed by thresholding fgmask_history using the constant STAT_TH and converting the resulting matrix to a binary image. Pixels in the sfg mask with a value greater than or equal to STAT_TH are considered stationary and set to 255 in sfgmask. Pixels with a value less than STAT_TH are considered non-stationary and set to 0 in sfgmask.

Specifically the three key parameters that control the behaviour of the algorithm are:

- *I_COST* is the increment cost for stationarity detection. It determines how quickly a pixel is considered stationary. When a pixel is classified as foreground in the current frame, its corresponding value in the history matrix fgmask_history is incremented by *I_COST*. A higher value of *I_COST* leads to faster accumulation of history, which makes it easier for a pixel to be classified as stationary. Conversely, a lower value of *I_COST* leads to slower accumulation of history, which makes it harder for a pixel to be classified as stationary.
- D_COST is the decrement cost for stationarity detection. It determines how quickly a pixel is considered non-stationary. When a pixel is classified as background in the current frame, its corresponding value in the history matrix *fgmask_history* is decremented by *D_COST*. A higher value of *D_COST* leads to faster decay of history, which makes it easier for a pixel to become non-stationary. Conversely, a lower value of *D_COST* leads to slower decay of history, which makes it harder for a pixel to become non-stationary.
- STAT_TH is the threshold value for classifying a pixel as stationary in the sfg mask. It determines the minimum value of *fgmask_history* that a pixel needs to have to be classified as stationary. A higher value of *STAT_TH* means that a pixel needs to have a higher accumulation of history to be classified as stationary, resulting in a more selective *sfg mask*. Conversely, a lower value of *STAT_TH* means that a pixel needs to have a lower accumulation of history to be classified as stationary, resulting in a more permissive *sfg mask*.

TABLE I
SUMMARY OF THE MAIN CHARACTERISTIC OF THE DATASET WITH THE SEQUENCES MORE USED FOR TESTING THE IMPLEMENTED ALGORITHM.

| Sequence | **Background complexity** | **Average height objects[pixels]** | **Prblems for segmentation** | **Problems for classification** |
|---|---|---|---|---|
| **AVSS 2007** | Medium | 50 to 100 | Occlusions, shadows, small scale. | People carrying luggage |
| **ETRI** | Low | 60 to 120 | Color of clothing | – |
| **PETS 2006** | High | 40 to 80 | Occlusions, shadows, dynamic background | People with objects |
| **VISOR** | Low | 30 to 90 | Occlusions | Different car perspectives |

## IV. DATA

As shown in the Table I the seleceted video sequences show natural images obtained from several point of view, distances and both outdoor and indoor. It is important to note that, for this group, only VISOR includes cars in the sequences so this is going to be mainly used to perform the main comparison in the object classification since it contains all the classes.

## V. RESULTS AND ANALYSIS

### A. Blob Extraction (Task 1)

The results of this part are crucial because, if they are not adequate, the following classification algorithms will not be able to work properly.

To demonstrate the correct functioning of the blob extraction, Figure 2 shows the results of the blob extraction for a simple case with clear segmentation.

Figure 3 shows comparatively the result of disabling the small blob removal function and it can be seen how

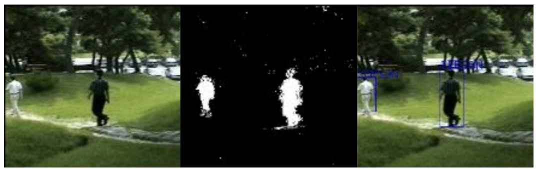
Fig. 2. Frame sample of ETRI sequence (left) at he foreground mask (center) and the labeled blobs (right).

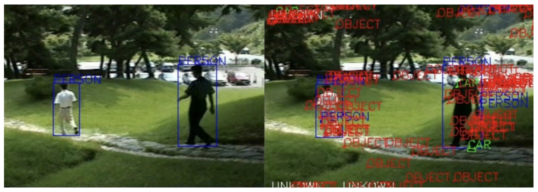
Fig. 3. Frame sample of original blobs in ETRI frame without filtering (left) and filtering blobs of less than 20 by 20 pixels(right).

efficiently and effectively our algorithm manages to remove all those small structures that do not have an indicated size.

In view of the results, the implemented functions work as expected.

### B. Blob Classification (Task 2)

The results of the blob classifications are shown in the following figures. In the Figure 4 the algorithm classifies correctly the approaching car in the sequence VISOR. As we can observed in the Figure 5 sometimes the classifier fails due to the shadowing to incorrect segmentation of the object. This approach can face some challenges when dealing with images of moving cars and pedestrians (the number of people can be high as shown in the Figure 6). When a car or pedestrian is moving, their shape can change rapidly, making it difficult to accurately measure their aspect ratio. For example, a car that is viewed from the side may appear long and narrow, while the same car viewed from the front may appear wider and shorter. Similarly, a pedestrian's aspect ratio can change as they move their limbs.

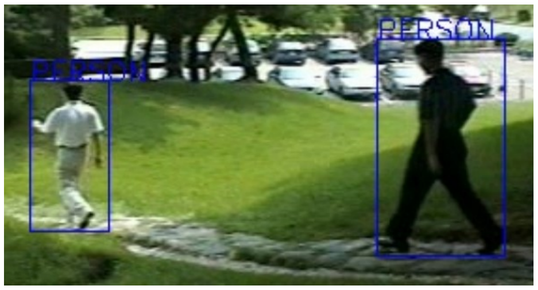

Fig. 4. Frame sample of ETRI where both pedestrians are correctly classified.

Despite our efforts the classifier is still really simple and it only relies on segmentation. THis means that there are obvious discontinuities thorough all the videos, leading to inconsistent classifications. Moreover, when dealing with images of moving cars and pedestrians, the foreground segmentation can also be challenging as the algorithm has

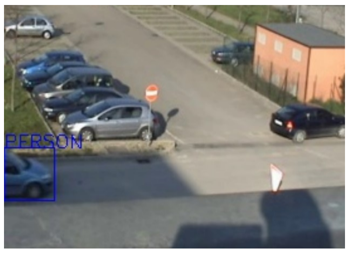

Fig. 5. Frame sample of VISOR where car is incorrectly classified.

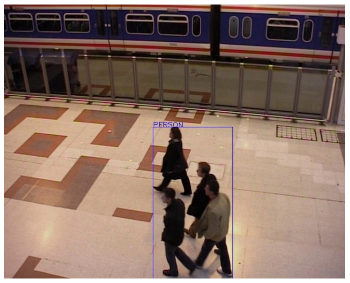
Fig. 6. Frame sample of sequence PETS2006 where several people is crossing moving across scene.

to distinguish the moving object from the background accurately. If the algorithm fails to do so, it can lead to incorrect measurements of the object's aspect ratio, leading to inaccurate classification.

Therefore, in order to accurately classify moving cars and pedestrians, the algorithm needs to take into account the object's motion and shape variation over time. More advanced techniques, such as motion detection, optical flow, and deep learning-based object detection and tracking, can be used to improve the accuracy of classification in such scenarios.

### C. Stationary Foreground Subtraction (Task 3)

The routine implemented takes in the current foreground mask, the history of foreground masks, and an output variable to store the stationary foreground pixels. The parameter learningRate is used to control the update rate of the background model to avoid including stationary objects in the background model and to give time to the foreground objects to create history. The Figures 9, **??** and **??** show a direct comparison on the effect of the learning rate for the same sequence.

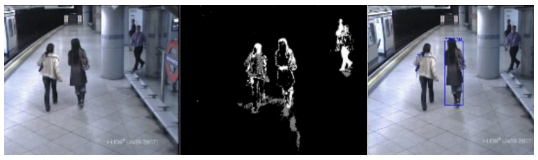
Fig. 7. Frame sample of sequence AVSS2007(left) and obtained segmentation mask for a small learningRate=0.01

The implemented foreground substracion method is tested for other video sequences and the results are satisfactory. For example in the Figures 12 11 **??** the algorithm behaves as

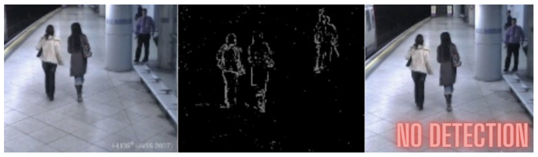

Fig. 8. Frame sample of sequence AVSS2007(left) and obtained segmentation mask for a medium learningRate=0.05.

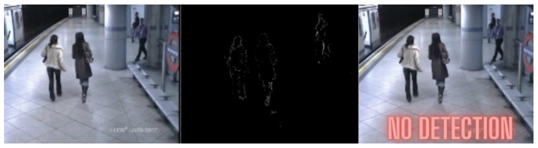

Fig. 9. Frame sample of sequence AVSS2007(left) and obtained segmentation mask for a large learningRate=0.5

expected for the different kinds of situations once an optimal parameter of learning rate is obtained empirically.

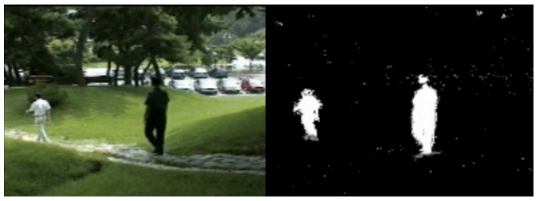

Fig. 10. Frame sample of sequence ETRIVSS2007(left) and obtained segmentation mask for its optimal learningRate= -0.5

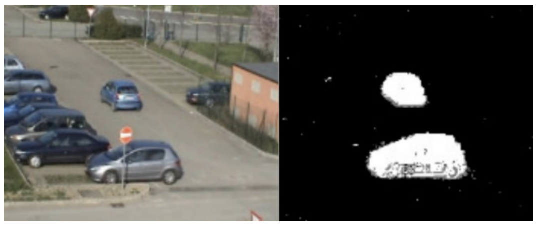

Fig. 11. Frame sample of sequence ETRIVSS2007(left) and obtained segmentation mask for its optimal learningRate= -0.01

Comparison of the routine's performance with and without equations (9) and (12) to evaluate the contribution of these equations to the accuracy of the stationary foreground extraction.

Calculation of performance metrics such as precision, recall, and F1 score to quantitatively evaluate the accuracy of the stationary foreground extraction.

Overall, the successful implementation of the routine for stationary foreground extraction is an essential step towards the accurate detection and tracking of moving objects in video surveillance applications. The proposed ideas for figures and tests can help to evaluate the performance of the routine and identify areas for improvement.

## VI. CONCLUSIONS

In this lab report, we have presented the results of our work on four tasks related to blob extraction and classification.

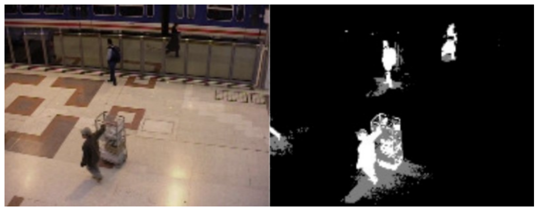

Fig. 12. Frame sample of sequence PETS2006(left) and obtained segmentation mask for its optimal learningRate= -0.4

In Task 0, we studied the sample code provided to gain an understanding of the basic concepts and algorithms used in blob analysis. In Task 1, we implemented a routine in C/C++ language to extract blobs from a video sequence using the OpenCV library. In Task 2, we extended the routine to classify the extracted blobs based on their size and shape characteristics. Finally, in Task 3, we implemented a routine to extract and classify stationary blobs in the scene based on their persistence and shape characteristics.

Overall, our results demonstrate the effectiveness of blob analysis techniques for detecting and classifying objects in video sequences. We found that by combining different features such as size, shape, and persistence, we were able to accurately distinguish between different types of blobs, including moving and stationary objects. Our work highlights the importance of understanding the underlying algorithms and parameters involved in blob analysis, as well as the need to carefully tune these parameters for optimal performance. We believe that the techniques presented in this report have broad applications in fields such as surveillance, robotics, and computer vision, and can be further improved and extended in future research.

## VII. TIME LOG

The time invested for this Lab is shown in the following points:

- Task 0: 2 hours of studying/understanding the sample code and setting up projects.
- Task 1: 4 hours of coding and debugging the Blob Extraction algorithm.
- Task 2: 2 hours coding the Blob Classification and 3 hours debugging it.
- Task 3: 1 hour reading and understanding requirements, 2 hours coding the Stationary Foregorund algorithm and 2 hours tuning parameters and debugging.
- Report: 6 hours of report writing and saving results for figures.

The total time load for this second lab has been, approximately, of 22 hours.

## REFERENCES

[1] Blum, H. (1967). A transformation for extracting new descriptions of shape. Models for the perception of speech and visual form, 362-380.

[2] Ortego, D., & SanMiguel, J. C. (2013, August). Stationary foreground detection for video-surveillance based on foreground and motion history images. In 2013 10th IEEE International Conference on Advanced Video and Signal Based Surveillance (pp. 75-80). IEEE.