LAB 3.1 a - Kalman filtering for object tracking
Measurement extraction part

APPLIED VIDEO SEQUENCES ANALYSIS

Zsófia Sára Budai
budai9902@gmail.com

I. INTRODUCTION

his report describes the development of a foreground segmentation and blob extraction method using C++ and the OpenCV library on Eclipse. The method was designed to process video sequences from the AVSA LAB3 dataset, which consists of colored frames captured from still cameras. The algorithm utilizes the MOG function provided by OpenCV for foreground detection, and applies a morphological opening operation using the morphologyEx function to filter out noise from the resulting foreground mask. Blob extraction is then applied to the foreground segmentation mask to identify and extract blobs. The largest blob from each frame is chosen, and its coordinates are stored for future use.

II. METHOD

The MOG function, along with the morphologyEx function provided by OpenCV, was employed for foreground detection, and a morphological opening operation was applied. After that, blob extraction was performed on the resulting foreground segmentation mask.

A. Foreground detection

First I created the "BackgroundSubtractorMOG2" and set the parameters history to 50 and threshold variable to 36 and true for detectShadows. The code loops throw the provided video sequence and converts the current frame into grayscale. To the grayscale image the "bgsubtractor" is applied with a learning rate 0.001. Once the foreground regions are identified, the morphological opening operation is applied to these regions to remove any noise or small artifacts, resulting in a cleaner and more accurate segmentation. I applied with "MORPH_RECT" and with a 3x3 size in order to get the best results. The opened foreground mask is shown in the plotted figure in the top right corner.

B. Blob analysis

In the blob extraction process, I implemented an "extractBlob" function which utilized OpenCV's built-in "floodFill" function. This allowed to apply the sequential Grass-Fire algorithm to each foreground pixel in the input foreground mask. The algorithm traverses the entire foreground mask from the top left corner to the bottom right corner, identifying foreground pixels (with a value of 255) and performing two actions. Firstly, the algorithm labels the pixel in the output image, and secondly, it sets the same pixel to zero in the input image (burning it). The neighboring pixels of this "burnt" pixel are then checked (either 4 or 8 neighbors, depending on the parameters given for connected components analysis), and any neighboring pixels that are also foreground pixels are treated in the same manner (i.e., labeled in the output image and burnt in the input image), and so on. This process continues until no more neighboring foreground pixels can be found. At that point, the label is incremented, and the algorithm resumes scanning the modified foreground mask from top left to bottom right, until the next foreground pixel is encountered, labeled, burnt, and its neighbors are investigated. It results in coordinates of the left top corner and width and length of all the blobs.
I have developed the "getBiggestBlob" function to extract the largest blob from a frame, which represents the tracked ball. For removing small blobs I used the previously implemented "removeSmallBlobs" with a threshold for minimum hight and width. To obtain the center coordinates of the largest blob, I determined the x position by adding the provided x position of the blob to half of its width. Similarly, for the y position, I added the y position of the blob to half of its height. The detected blobs are marked with a surranding rectangle in the plotted figure on the bottom left and the biggest blob with the measured center coordinates are on the bottom right in the figure.

III. IMPLEMENTATION

A. Functions, variables

- BackgroundSubtractorMOG2 from OpenCV for foreground mask
- morphologyEx from Opencv for morphological opening with MORPH_RECT and size (3,3)
- extractBlobs created for Lab2, implementing the Grass-Fire algorithm to extract blobs
- removeSmallBlobs created for Lab2, selecting only bigger blob than a provided threshold
- getBiggestBlob maximum search for blob sizes in one frame, only stores the biggest one per frame in the output list
- ShowManyImages with putText and drawMarker to visualize my result as it is shown on 1 figure.

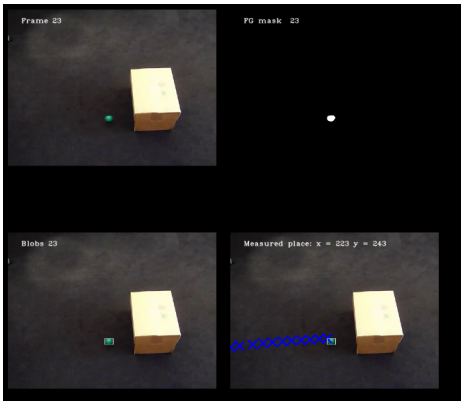

Fig. 1: Visualize result.

B. Running the code

The submitted files adhere to the format specified for the assignment, comprising the requisite source C++ code and corresponding Makefile.To compile, link, and execute the task 3.1.a, simply navigate to the source folder for the corresponding task and run the ”make” command in the terminal. This will generate the necessary built object file, which can be executed by running ”./main”.

IV. DATA

The algorithms underwent testing on video sequences in dataset_lab3 lab3.1 called singleball.mp4.There were 45 frames in the video, which was taken with a still camera. In the video, a ball moves in a left-to-right direction, but it is obscured by a box from frame 27 to 33.

V. RESULTS AND ANALYSIS

A. Learning_rate

A higher learning rate implies that the background model is updated more quickly with new frames, resulting in a faster adaptation to changes in the scene, however with higher learning rate not only the ball was detected as blob but parts of the shadow as well before the box, as it is visible on the 2 figure. For the final version I used 0.001.

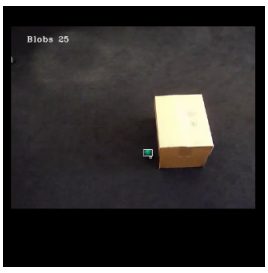

Fig. 2: More then one blob detected with learning_rate 0.01.

B. Minimum size for blob extraction

To filter out small blobs from an image, I utilized the ”removeSmallBlobs” function with a minimum width and height of 8. When I tried using a smaller value, such as in the case of the image shown in Figure 3, it resulted in more detected blobs due to the presence of a shadow after the box.

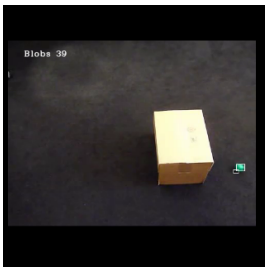

Fig. 3: Too small minimum blob size results detection in the shadow part.

C. Size for morphological filtering

To address artifacts in the background of an image, I applied morphological opening using a rectangular structuring element with a size of (3,3). I found that using a smaller size did not effectively eliminate the artifacts, while using a larger size caused deformation in the mask for the ball as it sometimes resulted in the boundaries being removed. The 4 figures illustrates size (1,1), (3,3) and (5,5).

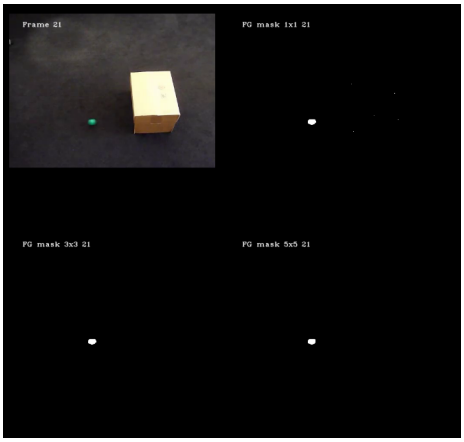

Fig. 4: Foreground masks with different sizes for morphological filtering.

VI. CONCLUSIONS

This project allowed me to acquire practical experience in utilizing built-in functions of OpenCV, as well as implementing my own custom functions from previous labs. I successfully optimized the parameters to obtain the best results and prepared the code for the next, more complex part of this lab exercise.

VII. TIME LOG

Approximately 9 hours were dedicated to this project.

- Studying: 0.5 hours
- Coding, debugging code : 4 hours
- Testing, screenshots: 0.5 hours
- Report: 4 hours