Detecting Moving Object in Recorded Videos or Live Webcam Using Open-CV and MOG2

Mohammad Sorkhian Khoozani
ID 201994162
Computer Vision
ENGI-9804

Abstract-This article uses some Image Processing techniques and MOG2 function to detect moving object by Open-CV library.
Index Terms – Motion Detection, Background Subtraction, MOG2, Image Processing techniques

I. INTRODUCTION

One of the greatest challenges in the digital world is detecting and tracking moving objects in live videos. Although human is good in this field, automating this procedure is challenging. For this purpose, there are some methods such as frame differencing and Background Subtraction; however, we still need to accompany these methods with some Image Processing techniques to improve the result. In this project I have used these advanced approaches along with some techniques like contour, threshold, noise reduction, and dilation.

- Frame differencing
- Optical flow
- Background subtraction.

1. *Frame differencing*

In this approach, we calculate the difference between two consecutive frames: the current frame and the previous one considered as a background. |*current frame* (I_C) – *background frame* (I_B)| > *threshold*(T). This method is fast and straightforward to implement Since it requires less computation. But it is sensitive to noise and holes which are generated in detected moving objects. [1].

2. *Optical flow:*

Optical flow is the pattern of apparent motion of objects, surfaces, and edges, which has two different methods (Lucas-Kanade and Horn-Schunck). James J Gibson introduced the concept of optical flow in 1940. This method gives all motion information by describing the direction and velocity of each point in consecutive images in a two-dimensional vector. [1] [1] [3].

3. *Background subtraction*

This method is a fast way for distinguishing Foreground from the Background. In a video, each frame is divided into two sets of pixels. One, Foreground, referred to moving objects such as humans and vehicles, while the other one is the complementary set of pixels denoted as Background like a road without vehicles or a room without visitors.

For background subtraction, background modeling is required. There are two main approaches (Recursive and Non-Recursive Algorithms).
1. Recursive Algorithm does not require any buffer for background estimation. In this technique, a single background model is updated recursively based on each input frame. Therefore, input frames from the distant past could affect the current background model. Although this method requires less storage than non-recursive one, any error in Background model can last longer.
2. Non-Recursive Algorithm uses a buffer to store limited previous video frames and estimates the background image based on the temporal variation of each pixel within the buffer. This technique is highly adaptive since it does not depend on the history beyond those buffered frames. In these approaches, in cases that objects have shadows, heir shadows are detected as a part of the subjects. There are a lot of algorithms for alleviating this obstacle, which some of them have been implemented by OpenCV library (BackgroundSubtractorMOG, BackgroundSubtractorMOG2, BackgroundSubtractorGMG). In the next section we will consider the first two methods.[2] [3]

II. BACKGROUND SUBTRACTION

A. *BackgroundSubtractorMOG:*

In this method, we need a reference frame, and there are a lot of approaches for creating this. One method is a time-averaged background, but it has some disadvantages like inability to detect gradual illumination change. Another methods are adaptive parametric mixture model of three Gaussian distributions by Friedman and Russell [4] and

Kalman filter for tracking changes in background illumination by Koller et al. [5]. Although these were able to solve the illumination change problem, they could not handle the problem of detecting adding or removing an object. For this, a successful solution, multi-color background model per pixel, was presented by Grimson et al. This model was based on an adaptive nonparametric Gaussian model, which is robust to small camera changes and repetitive moving objects like trees by benefiting from *spatial coherence*. Despite all these advantages, it had a slow learning rate and was unable to separate moving objects from their shadows. Therefore, KadewTraKuPong and R. Bowden [6] in their paper updated the equations of Grimson et al. paper and made this algorithm more efficient, accurate, and compatible with busy environments. Furthermore, they presented a computational color space method that enabled their algorithm to distinguishes moving objects from shadows.

B. *BackgroundSubtractorMOG2*

For extracting the foreground region from based on a scene, we can assume that an image without moving objects has a regular behavior that can be described by a statistical model. This model benefits from probability density function and variances that can be different for each pixel. In this procedure, pixels density of a new frame is compared with the density function, and this pixel is considered as the background if its intensity be in the range. In this procedure, we can use a single Gaussian model. OpenCV has implemented the Zoran Zivkovic [8] algorithm under the name of BackgroundSubtractorMOG2, and we can have separate shadows from the objects with setting "detectingshadows" parameter to true [3]. in the following section we go deep into MOG algorithm.

III. BACKGROUND SUBTRACTION MOG

Indeed, this algorithm is based on the work of Grimson and Stauffer [7]. They modeled each background pixel with a mixture of gaussian distribution as different gaussians were used to represent a separate color. These distributions had application in representing the background. Colors that sustained longer were consider as probable background colors that contributed in determining background regions. Grimson and Stauffer presented a value, fitness, that holds colors space. They showed moving objects have a broader range of this value than static ones since these objects reflect light from different parts of their surface in the movement. In this procedure, if the value of a new pixel be in the existing Gaussian components, that component gets updated; otherwise, a new component will be created.

A. *Adaptive Gaussian Mixture Model*

The following formula shows the probability that a pixel has a value of X_N at time N. w_k is the weight parameter of the k_{th} Gaussian component.

$$P(X_N) = \sum_{j=1}^{K} w_j \eta(X_N; \theta)$$

$\eta(x; \theta\ k\ )$ is the Normal distribution, μ_k is mean and $\sum k = \sum_k^2 I$ is covariance of the k_{th} component.

$$\eta(X_N; \theta) = \eta(X_N;\ \mu_k, \sum k\ ) = \frac{1}{(2\pi)^{\frac{D}{2}} |\sum k|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu_k)^T \sum_k^{-1} (x-\mu_k)}$$

All distributions are sorted based on the fitness value $\frac{w_k}{\sigma_k}$ and the first $arg_b min\left(\sum_{j=1}^{b} w_j > T\right)$ are used as the background model. Where T is threshold. The background update equation is as follow:

$$\hat{w}_K^{N+1} = (1-\boldsymbol{\alpha})\hat{w}_N^K + \boldsymbol{\alpha}\hat{p}(\boldsymbol{\omega_K} \mid x_{N+1})$$

$$\hat{\mu}_K^{N+1} = (1-\alpha)\hat{\mu}_K^N + \rho x_{N+1}$$

$$\sum_K^{N+1} = (1-\alpha)\sum_K^N + \rho(X_{N+1} - \hat{\mu}_K^{N+1})(X_{N+1} - \hat{\mu}_K^{N+1})^T$$

$$P = \alpha\eta\left(\mathbf{x}_{N+1};\ \hat{\mu}_{\boldsymbol{K}}^{\boldsymbol{N}},\ \sum_{\boldsymbol{N}}^{\boldsymbol{K}}\right)$$

$$\hat{p}(\omega_K \mid x_{N+1}) =$$
1 if ω_K *is the first match;* 0 *otherwise*

B. *Shadow Detection and Color Model*

As I said before, the inability to detect moving shadows from the object is another downside of Grimson et al.'s algorithm. Therefore, KadewTraKuPong and R. Bowden decided to benefit from an RGB space [9], Chromatic and Brightness, for distinguishing moving objects and shadows with the following formula.

$$a = argmin(I - zE)^2$$

$$c = \|I - aE\|$$

Where 'E' is mean of pixel background, $\|E\|$ expected chromaticity line, 'd' chromatic distortion, 'τ' brightness threshold, 'I' given observed pixel value, 'a' brightness distortion, 'c' color distortion.

With the spherical Gaussian distortion assumption, 'σ_k' the standard deviation of k_{th} component can be set to 'd'. If 'a' be equal 2.5 the given pixel is considered as a moving shadow and $\tau < c < 1$.

IV. TECHNICAL WORKFLOW

In this section, I implement two sets of codes for detecting moving objects, frame differencing and MOG2, along with some Image processing techniques to improve the result.

A. Frame Differencing

At the beginning, we read two frames of a video file after that we convert these frames to Grayscale mode Fig01 since, in this mode, finding contour in the next few sections is less computationally expensive. In the second step for detecting the moving regions, we use *cv.absdiff*. This function takes two input arrays in this case images and gives a differential array Fig02. After that, we perform the following morphological techniques:

Fig01. Converted to Gray

Fig02. absdiff

1) *Noise reduction*

for decreasing noises, we need to blur the difference frame to become prepared for further processes on the image. For this purpose, we used *cv.GaussianBlur* blurring filter. This filter takes some input variables such as: src, ksize, sigmaX and sigmaY, which are in order the input image, size of the kernel in positive and odd numbers, standard deviation in X and Y direction. In Figs 03-05, I have shown threshold result before and after applying this filter with (3,3) and (5,5) kernels.

Fig03. Original image

Fig04. Blurred image with cv.GaussianBlur and (3,3) kernel

Fig05. Blurred image with cv.GaussianBlur and (5,5) kernel

In the Fig03, a lot of noises can be seen while the (3,3) blurring filter Fig04 has effectively removed them. But in the (5,5) filter Fig05, although it has removed some extra noises, some details of target objects have also been removed. Therefore, we choose (3,3) kernel over the other ones.

2) *Threshold*

After this, we use thresholding technique. In this technique we specify an intensity value that any pixel value lower than this is considered zero, and any higher is set to max value. In this technique, we pass our frame into the function in grayscale, set a threshold value, define the max number for pixels that exceed the threshold, and the thresholding type.

Fig06. Threshold = 10

Fig07. Threshold = 20

Fig08. Threshold = 30

I set the threshold to 10, 20, and 30. As we can see in the Fig06, with th=10, we have a significant number of false-positive pixels, and in Fig08, th=30, we have lost valuable pixels of the moving object. But in the Fig07, th=20, we have been able to preserve moving objects pixels and filter false-positive pixels effectively.

3) *Dilation*

As we can see in the thresholding result, there are a lot of holes. We can fill these holes with some approaches like Closing and Dilation. Dilation function has a few arguments like src, dst, element, anchor, iteration, borderType, and borderValue. These elements refer to the source image, output image, the shape of the kernel, location of kernel center, number of times the dilation is repeated, and bordering values in cases that kernel is working on the border pixels of an image, respectively.

Fig10 is the result of three iterations which performs better than two or four. As it could efficiently fill the holes while reasonably keeps the objects separated.

Fig09. Dilate (iteration=2)

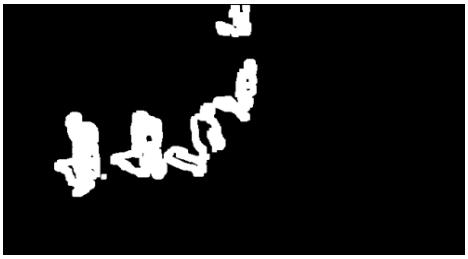

Fig10. Dilate (iteration=3)

Fig11. Dilate (iteration=4)

4) *Contour*

In this technique, we can connect continuous points with the same color or intensity in a boundary with a curve. This approach is used for detecting different objects in an image. This function takes several arguments. One of these is the mode. In this argument we can choose RETR_LIST, RETR_EXTERNAL, RETR_CCOMP, and RETR_TREE. The first mode is the simplest one, which just creates contour but doesn't represent any information about parent and child relationships between the objects. RETR_EXTERNAL only retrieves parent object and do not consider children. RETR_CCOMP returns all contours which the holes in objects and external contours are labeled in order as a level-2 and level-1 hierarchy. RETR_TREE is the most complete one retrieving contours with a full hierarchy. The last mode is very prevalent, and we use this too. The second argument is method. For this, we can select CHAIN_APPROX_NONE or CHAIN_APPROX_SIMPLE. In the first one, all the points in contours are preserved while in representing some boundaries like a line, two points suffice. So, for saving up memory, we can use the latter one, CHAIN_APPROX_SIMPLE. The result of the contour is as below.

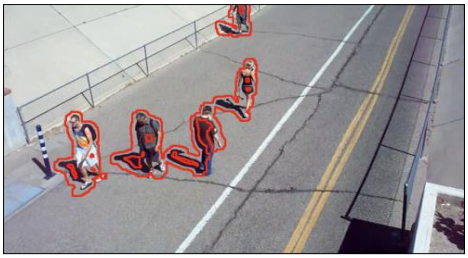

Fig12. Contour

5) *Bounding Rectangle*

In this step, we constrict the detected moving objects with Rectangles. To do this; we have two options "Straight Bounding Rectangle" and "Rotated Rectangle". The former cv2.boundingRect() gives us coordinate of top left corner and width and height of a rectangle. The latter (cv2.minAreaRect()) considers the rotation of objects for retrieving a constricting rectangle with minimum area. This function returns the top left corner, width, and height and angle of a rectangle.

In this stage, we benefited from the cv2.boundingRect() method.

6) *Threshold*

Now we use thresholding for eliminating very small rectangles that are more likely to be noise or part of an object. And we finally end up with threshold=600.

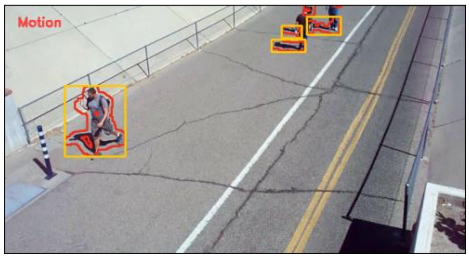

Fig13. Bounding th = 300

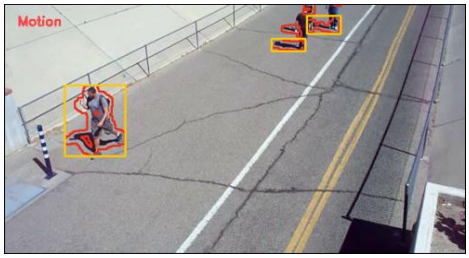

Fig14. Bounding th = 600

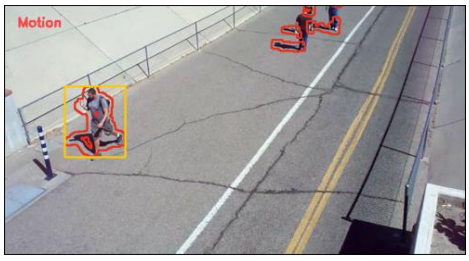

Fig15. Bounding th = 900

B. MOG2

In the following I implement the MOG2 function for detecting moving object. Although it detects foreground with a high precision, we need some Image Processing techniques like Blurring, thresholding, and opening to improve the result. After these, I ended up with the Fig17 mask. Following this I benefited from Contour, BoundingRect and threshold to present moving object in the original image.

Fig16. Row foreground mask from MOG2

Fig17. MOG2 after Opening

V. CONCLUTION

When we compare the outputs of Frame Differencing and MOG2, we can see in the Fig17 and Fig18 the MOG2 contour has detected the objects edges very precisely. And also, less holes in the objects. I observed that in these implementations, after applying Image Processing techniques, both have been able to detect moving objects effectively.

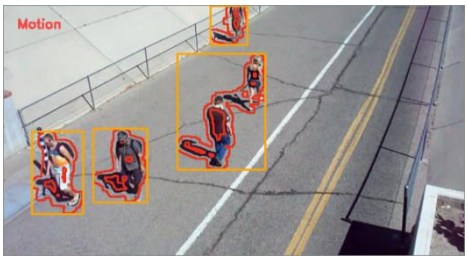

Fig17. Frame differencing approach

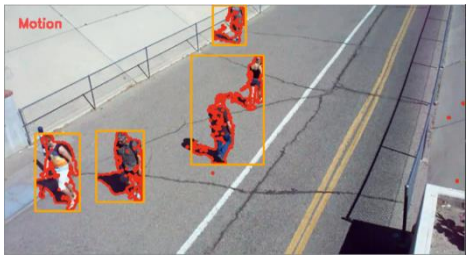

Fig18. MOG2 approach

REFERENCES

[1] Sepehr Aslani, Homayoun Mahdavi-Nasab “Optical Flow Based Moving Object Detection and Tracking for Traffic Surveillance”.
[2] Dhara Trambadiya 1, Chintan Varnagar “A Review on moving object detection and tracking methods”.
[3] opencv-python-tutroals.readthedocs.io/ en/ latest/ py_tutorials/ py_video/ py_bg_subtraction/ py_bg_subtraction.html
[4] Friedman N., Russell S. “Image segmentation in video sequences: A probabilistic approach”.
[5] Koller D, Weber J. Huang T. Malik J. Ogasawara G. Rao B. Russell S. “Towards robust automatic traffic scene analysis in realtime”.
[6] P. KaewTraKulPong and R. Bowden “An improved adaptive background mixture model for realtime tracking with shadow detection”
[7] C. Stauffer and W. Grimson “Adaptive background mixture models for real-time tracking”.
[8] Z.Zivkovic, “Improved adaptive Gausian mixture model for background subtraction” in 2004
[9] Horprasert T., Harwood D., Davis L.S. “A statistical approach for real-time robust background subtraction and shadow detection”.