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

Over the past few decades along with advent of digital cameras, video processing has been used in numerous fields. One of the greatest applications of this is detecting and tracking moving objects. Although human is good in this field, the main problem is automating this procedure. For alleviating this issue, there are some methods that the most prevalent ones will be described in the following. Furthermore, I have benefited from some Image Processing techniques like contour and dilation, to improve the result.

- 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. Based on this method and edge detection techniques, Zhan [4] *et al*. (2007) proposed an algorithm with a high recognition rate and speed along with a noise restraining method for detecting moving objects from the sets of images from a static camera. Firstly, this algorithm gets the difference between the edges of two adjacent frames. Then, it divides the edge difference image into small blocks, and by comparing the number of non-zero pixels shows 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. Although his method can tolerate camera motion, corresponding points in the successive frames might not be allowed to be more than a few pixels away. Additionally, this approach is sensitive to noise, has poor anti-noise performance, requires a high volume of computation, and can be affected badly by environmental and illumination changes, which these downsides could make it unsuitable for real-time demanding occasions. [1] [2] [3].

3. Background subtraction

It could be said that Background Subtraction is the most commonly-used approach for motion detection with static cameras. This method is a fast way for distinguishing Foreground from the Background and detecting moving objects. In a video, each frame is divided into two sets of pixels. One, Foreground, referred to moving objects such as humans, vehicles, boats, 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. This modeling should be sensitive enough for detecting motions. Currently, mean and median filters are two most popular ones. For background modeling, 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. As a result of this, 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. This technique includes various methods such as approximate median, adaptive Background, Gaussian of mixture.

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. But its downside may be in videos with slow-moving objects which require a larger buffer.

As we detect moving objects correctly, we would be able to perform numerous modifications and prepare them for different applications. For instance, we can benefit from this method for counting number of people in a store or vehicles that passing specific area or use it for human-computer interaction and medical imaging. However, the Background Subtraction method is sensitive to sudden changes in illumination and camera motion. Furthermore, implementing this method has some obstacles. In cases that objects have a shadow, heir shadows are detected as a part of the subjects and makes the postprocessing procedure more complicated. 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 these methods.[5] [6]

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 ditect gradual illumination change. For solving this problem, numerous solutions have been presented, which some of them are: adaptive parametric mixture model of three Gaussian distributions by Friedman and Russell [7] and Kalman filter for tracking changes in background illumination by Koller et al. [8]. 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. Following this, some people like Toyama K et al. [9], tried to improve the computational complexity of this algorithm, but they were not able to efficiently tackle this issue. Therefore, KadewTraKuPong and R. Bowden [10] 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. The BackgroundSubtractorMOG function In OpenCV has been implemented based on the paper of KadewTraKuPong and R. Bowden, which uses a mixture of 3 to 5 Gaussian distributions to model each background pixel. In this mixture more sustained colors in the scene have higher weights. This algorithm has some arbitrary parameters like memory length, number of mixtures, threshold and, so on.

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; however, this model can be more complex, and the Gaussian mixture model (GMM) is required for better representation. Numerous research has been done based on GMM by various scholars such as N. Friedman et al.[7], C. Stauffer et al.[11] and E.Hayman et al.[12] But Zoran Zivkovic [13] developed some recent research and presented an algorithm that adopts a number of components of the mixtures and parameters for each pixel. One advantage of this algorithm in comparision to MOG is that in that one, we expedited K-fixed Gaussian distribution while In this one for each pixel, an appropriate number of Gaussian distribution is used. OpenCV has implemented the Zoran Zivkovic algorithm under the name of BackgroundSubtractorMOG2, and we can have separate shadows from the objects with setting “detectingshadows” parameter to true.[6]

C. *BackgroundSubtractorGMG*

GMG is another background-foreground separator that can track moving objects, even in variable lighting environments. In BackgroundSubtractorGMG, OpenCV has benefited from Andrew B. Godbehere et al. [14] paper. In this paper, they used statistical background image estimation, per-pixel Bayesian segmentation, and bank of Kalman filters and Gale-Shapley matching as an approximate solution to the multi-target tracking problem. Their algorithm could improve recall and f_2-score of the algorithm used in OpenCV 2.1. The newly-presented algorithm has some upsides; for instance, it can become ready to produce output in a few seconds, and no training and calibration are required. Also, it is capable of being used in real-time (15 frames/s) processing. This algorithm has two sections, in the first part recognizes

possible foreground objects using Bayesian, and the background consisting of nonparametric distributions on RGB color space for each pixel. Furthermore, for dealing with variable illumination, it uses heavier weights for new observation. The second part, filters the detected moving objects with a heuristic confidence model that uses error covariance. Also, it sues Kalman filter along with the Gale-Shapley algorithm. After these main parts, feedback is introduced to update the background image and separating it from the foreground. Since this algorithm was implemented in an interactive system for testing, the false-negative cases were less tolerable; therefore, the false-positive tolerance was increased, and after a few months of observation the results showed a significant improvement in compared with algorithms in OpenCV 2.1. 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 [11]. 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 colours 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:

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

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

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

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

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

The main problem of this algorithm is when the first pixel belongs to the foreground object. In this case, we only have one Gaussian, and one-color background that takes $\log_{1-\alpha} T$ frames to generate a dominant background, and the busy environments can make this situation worse. Kadew TraKuPong and R. Bowden, in their paper, worked on this shortcoming and developed a solution, Online EM Algorithm, for it. In their paper, when the algorithm is processing the first L samples, they try to generate a reasonable estimation of the Gaussian Mixture model. And sufficient statistics update equations can do so even before completion of collecting sample procedure and even makes the convergence faster. In the following equations, we would see L-recent window version and Online EM, respectively.

$$w_K^{N+1} = w_K^N + \frac{1}{L}(p(\omega_K|\mathbf{x_{N+1}}) - w_K^N)$$

$$\hat{\mu}_K^{N+1} = \hat{\mu}_{\boldsymbol{K}}^{\boldsymbol{N}} + \frac{1}{L}\left(\frac{\hat{p}(\boldsymbol{\omega_K} \mid x_{N+1})\, x_{N+1}}{w_K^{N+1}}\right)(\hat{\mu}_K^N)$$

$$\sum_K^{N+1} = \sum_K^N + \frac{1}{L}\left(\frac{\hat{p}(\boldsymbol{\omega_K} \mid x_{N+1})(X_{N+1} - \hat{\mu}_K^N)(X_{N+1} - \hat{\mu}_K^N)^T}{w_K^{N+1}} - \sum_K^N\right)$$

$$w_K^{N+1} = w_K^N + \frac{1}{N+1}(p(\omega_K|\mathbf{x_{N+1}}) - w_K^N)$$

$$\hat{\mu}_K^{N+1} = \hat{\mu}_{\boldsymbol{K}}^{\boldsymbol{N}} + \frac{\hat{p}(\boldsymbol{\omega_K} \mid x_{N+1})}{\sum_{\boldsymbol{i=1}}^{\boldsymbol{N+1}} \hat{p}(\boldsymbol{\omega_K} \mid x_i)}(X_{N+1} - \hat{\mu}_K^N)$$

$$\sum_K^{N+1} = \sum_K^N + \frac{\hat{p}(\boldsymbol{\omega_K} \mid x_{N+1})}{\sum_{\boldsymbol{i=1}}^{\boldsymbol{N+1}} \hat{p}(\boldsymbol{\omega_K} \mid x_i)}((X_{N+1} - \hat{\mu}_K^N)(X_{N+1} - \hat{\mu}_K^N)^T - \sum_K^N)$$

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. One practical solution to this problem is chromatic color space representation. But considering a chromatic model without brightness not only can cause instability in the model specifically in objects with very high or low intensity level, but also is very computational expensive, especially in large

images. Therefore, KadewTraKuPong and R. Bowden decided to benefit from an RGB space [15], Chromatic and Brightness, for distinguishing moving objects and shadows. After a comparison between current background components and a given pixel, if chromatic and brightness difference be within the specified thresholds, that pixel is labeled as a shadow. The benefited color model has 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. PRACTICAL IMPLIMENTATION

In this section, I implement two sets of codes for detecting moving objects in the first set, I use frame differencing method alongwith some Image processing techniques to improve the result, and in the latter one, I use MOG2 method.

A. First method (Image processing techniques)

At the beginning, we read two frames of a video file after that we convert these frames to Grayscale mode 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. After that, we perform the following morphological techniques:

Fig01. Original frame 218

Fig02. Original frame 396

1) *Noise reduction*

for decreasing noises, we need to blur the difference frame to become prepare 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 of original image and original image after applying this filter with (3,3) and (5,5) kernels.

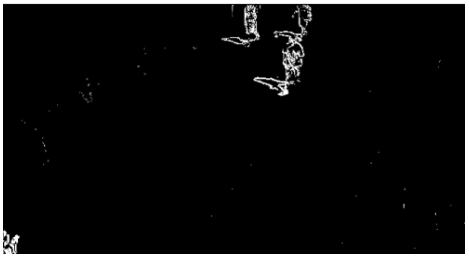

Fig03. Original image

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 a considerable amount of noises. 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 cut off number is considered zero, and any higher is set to max number. 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. For solving this issue, 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.

In Fig09-09 can be seen that dilation with three iterations performs better than two or four as it could efficiently fill the holes while reasonably maintains the objects separated.

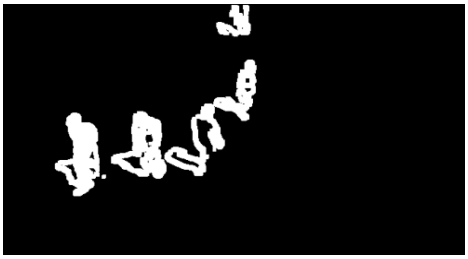

Fig09. Dilate (iteration=2)

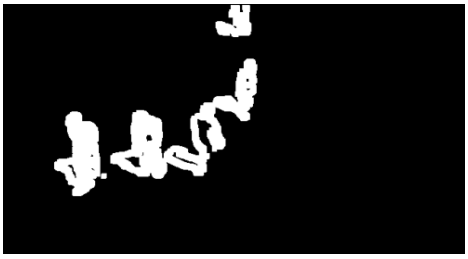

Fig10. Dilate (iteration=3)

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.

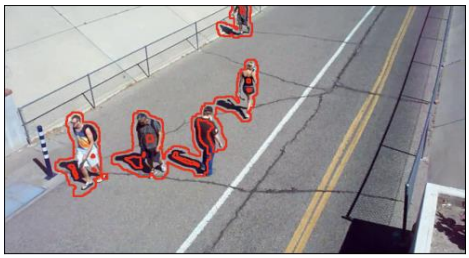

Fig12. Contour

5) *Bounding Rectangle*

In this step, we constrict the detected moving objects with Rectangles. I order 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.

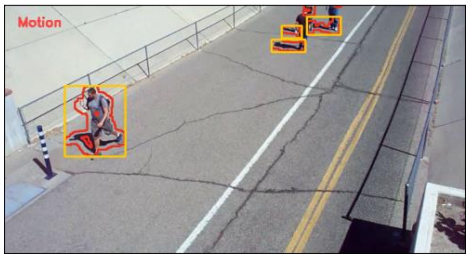

Fig13. Bounding th = 300

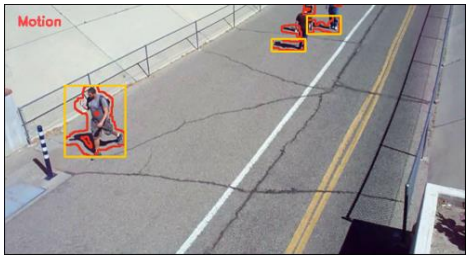

Fig14. Bounding th = 600

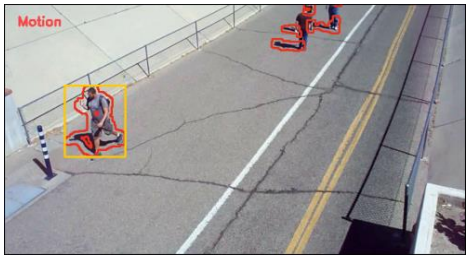

Fig15. Bounding th = 900

B. Second method (MOG2)

In the following I implement the MOG2 function for detecting moving object and in the Fig16 we can see that it detects foreground with a high precision. After applying Blurring, thresholding, and opening, I ended up with the Fig17 mask.

Fig16. MOG2

Fig17. MOG2 after Opening

V. CONCLUTION

When we compare the outputs of two methods, 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.

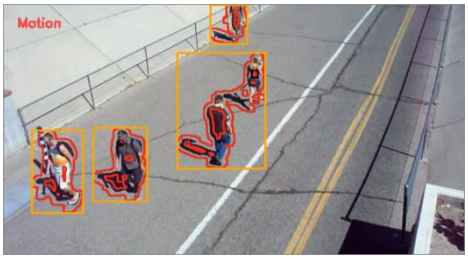

Fig17. Frame differencing approach

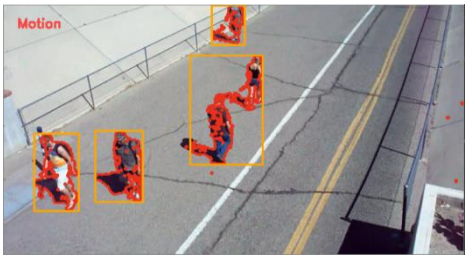

Fig18. MOG2 approach

REFERENCES

[1] Melody Suzan1, G Prathibha2 “Detection and tacking of moving objects using hybrid model (GMM & Horn-Schunck)”
[2] Sepehr Aslani, Homayoun Mahdavi-Nasab “Optical Flow Based Moving Object Detection and Tracking for Traffic Surveillance”.
[3] Rupali S.Rakibe, Bharati D.Patil “Background subtraction algorithm based human motion detection”.
[4] Zhan Chaohui “Detect the problem of background subtraction in frame differencing and give the improved method to solve the problem with high detection speed and solve complicated background problem”.
[5] Dhara Trambadiya 1, Chintan Varnagar “A Review on moving object detection and tracking methods”.
[6] opencv-python-tutroals.readthedocs.io/ en/ latest/ py_tutorials/ py_video/ py_bg_subtraction/ py_bg_subtraction.html
[7] Friedman N., Russell S. “Image segmentation in video sequences: A probabilistic approach”.
[8] Koller D, Weber J. Huang T. Malik J. Ogasawara G. Rao B. Russell S. “Towards robust automatic traffic scene analysis in realtime”.
[9]: Principles and practice of background maintenance.
Toyama K, Krumm J. Brumitt B. Meyers B. Wallflower Part vol.1, 1999. 1999.
[10] P. KaewTraKulPong and R. Bowden “An improved adaptive background mixture model for realtime tracking with shadow detection”
[11] C. Stauffer and W. Grimson “Adaptive background mixture models for real-time tracking”.
[12] E.Hayman and J. Eklundh, “Statistical Background Subtraction for a Mobile Observer”.
[13] Z.Zivkovic, “Improved adaptive Gausian mixture model for background subtraction” in 2004
[14] Andrew B. Godbehere, Akihiro Matsukawa, Ken Goldberg “Visual Tracking of Human Visitors under Variable-Lighting Conditions for a Responsive Audio Art Installation”
[15] Horprasert T., Harwood D., Davis L.S. “A statistical approach for real-time robust background subtraction and shadow detection”.