Simple Motion Detection in Video Sequences

Filip Milić
Student of Computer Science
Faculty of Electrical Engineering and Computing
Zagreb, Croatia

Tomislav Grdenić
Student of Computer Science
Faculty of Electrical Engineering and Computing
Zagreb, Croatia

Barthélémy Marsault
Student of Computer Science
Graduate school of engineering
University François Rabelais
(Polytech'Tours)
Tours, France

Florian Gigot
Graduate school of engineering
University François Rabelais
(Polytech'Tours)
Tours, France

Gaëtan Jagorel
Graduate school of engineering
University François Rabelais
(Polytech'Tours)
Tours, France

***Abstract*—We discuss and implement two algorithms for motion detection on (presumably) static cameras. The first is the well-known method of background subtraction, whereas the other is using magnitudes of dense optic flow estimation. The ultimate goal is to create a model that does not require learning before being applied to footage (in other words, no pre-trained models), and can estimate the positions of moving objects on scenes.**

Keywords—motion detection, background subtraction, optical flow

I. Introduction

The modern approaches to modern motion detection have largely become intertwined with the area of object detection. For instance, the most famous among these approaches is probably the "You Only Look Once" (YOLO) algorithm, which aims to be a real-time object detection system, not only *detecting* objects, but properly *classifying* them, as well.

The core point of this paper can, therefore, in some ways be considered a step-down. Our objective is to merely detect the presence of moving objects in video sequences, create a bounding box around them, and track them as they move through the scene. Since we are not focused on classifying the entities captured, we can more comfortably step away from pre-trained models and focus our attention on more genera-use-ready algorithms.

We will discuss and implement two such approaches. As such, the paper is divided into two main sections. The first focuses on the well-known background subtraction model. The other, on the more unstable model based on optical flow estimation. The final section of this paper covers tests on this model on the LASIESTA motion detection dataset.

The main presumption – a crucial one – is one of the static camera. What both of the models have in common is the idea that any and all movement present is always the result of moving bodies, and *never* the result of the camera itself. Here we are careful to point out that we do not equate "movement" with general, more permanent changes to the scenery, such as lighting conditions and objects simply left behind. Yet, both are an undeniable obstacle in achieving desirable results. Solving those issues was the driving force behind the implementation of each approach.

II. BACKGROUND SUBTRACTION METHOD

A. On Background Subtraction

The key principle of background subtraction is, ultimately, a simple one. It starts with the assumption that a scene can be divided into two parts – a background and a foreground. The background is always perfectly still and never-changing, whereas the foreground objects make themselves distinct from the background precisely because they are not. In other words, identifying the *foreground* objects becomes equivalent to detecting *moving* objects.

In our specific case of detecting moving objects on a static camera, the approach is rather straightforward. First, we take a still frame of what we declare to be the scene's background. Then, going through the footage, for every frame, we subtract the background image. What remains will always be the pixel values that don't match – IE, the differences – IE, the *foreground*. Fig. 1 displays this process.

Of course, the process is not entirely so simple. Pixel values between video frames can be mismatched for a variety of reasons – noise, change in focus, subtle change in shadow or lighting. These changes usually don't take a large portion

Fig 1. Simple Background Subtraction

Fig 2. The Issue of Simple Background Subtraction

of the resulting foreground image or have a significant difference in values, and can therefore be removed in two steps:

The first – after properly normalizing the image values – is to simply set pixel values that are below a given threshold to 0. This is also usually standard practice for removing soft shadows from the foreground. After all, shadows also move, but are not something we want detect as moving objects. Therefore, taking into account that the difference between the background and the area where the shadow falls are likely to be smaller than those of moving objects, we can safely remove them by raising the threshold value to about 128.

The second is to take the remaining foreground image and try to discern the shapes of objects from it. We do this by first dilating the image – letting the non-zero pixel values spread lightly around their general area. We do this to ensure that one object is not accidentally discerned as two distinct ones. For example, a person's upper and lower body can often be detected as two separate objects, because the result of the background subtraction failed to properly connect the two when processing the foreground image.

After dilating the image, we still cannot assume that all areas remaining are, by default, actual objects. As helpful as the process of dilation is, it may have had the side-effect of letting the few stray pixels remaining to connect into a larger area on the image. To combat this, we calculate the areas of the objects in the foreground and eliminate those that are too small.

Finally, what we are left with are all objects in the foreground. In-line with our previous assumption, we determine that they are, indeed, the objects that are moving. We draw a bounding box around each and display it over the footage.

This process is fast and easily applicable in real-time. However, as it is, it's not entirely accurate. The problem is the presumption of a never-changing background. As we've discussed before, we require that our model can discern that gradual changes in lighting and unmoving objects left behind or flat-out gone from the scene do *not* count as detection-worthy. The current model simply cannot do this. If we want to reduce the problem of motion detection to finding foreground objects, then we will need a method that accounts for the background reference being *changed* over time.

Fig 2. shows this requirement more clearly.

In it, a man is seen arriving at the scene with the box, leaving it behind, and walking away. If the reference background image is that of just the couch, the background subtraction will always place the box in the foreground. And since we are always assuming that foreground objects are the ones moving, it will always highlight the box – even though it's perfectly still.

B. *Adapting to Background Changes*

One might, at first, consider that updating the background image could simply be done by periodically switching out the previously selected background reference image for another one. In practice, however, this is nigh impossible to achieve. While such an approach could be used to combat different times of day (knowing in advance that shadows are always likely to fall a certain way for certain hours, and thus preparing a certain background image for every hour), it cannot account for pedestrian behavior. A person may come along and leave a unique object behind. How is the reference image supposed to be updated with it? And at what intervals?

A far more novel approach is suggested by Zivkovic Z. and van der Heijden F. in [1]. They present a non-parametric method very similar to a k-nearest neighbors model. In it, recognizing the background and foreground objects comes down not to simply subtracting two images, but classifying each individual pixel as either belonging in the background or foreground, based on a certain number of the most recent video frames. From those handful of frames, the oldest one is removed from the model, and the newest one is added.

This entire process repeats for each frame.

The process of determining whether the pixel for the given frame should be considered a part of the background or foreground comes down to estimating the probability of it being a part of the background. If the probability is above a certain threshold, it is classified as such. Otherwise, it is determined as part of the foreground.

The probability is calculated by estimating the Euclidean distance between the pixel value and the pixel values of the pixels previously classified as part of a background (that are still in the model and in the same position as the pixel being considered). The shorter the distance is for more background pixels, the higher the probability is.

The exact formula is given as:

$$p(\boldsymbol{x}\,|O_T, BG) \approx \frac{1}{TV}\sum_{m=t-T}^{t} b^{(m)} h\left(\frac{||\boldsymbol{x}^{(m)} - \boldsymbol{x}||}{D}\right) \qquad (1)$$

Where $p(\boldsymbol{x}\,|O_T, BG)$ is the probability of the given pixel being a part of the background, O_T the reference frames saved by the model, V the volume of the kernel formed around the

pixel we're trying to classify, and T is the number of frames kept in the model. $\boldsymbol{x}^{(m)}$ is a pixel from frame m in the model, in the same position as pixel $\boldsymbol{x}$, which is the one we're trying to classify. Since O_T contains both foreground and background pixels, we need only the background ones. Because of that, we use the $b^{(m)}$ function, which is set to 1 for background pixels, and 0 otherwise.

D is the diameter of the hypersphere formed by the kernel. It is estimated using the median of the absolute difference between the samples in O_T. $h(u)$ is a function that equals 1 if u is below a certain threshold, and set to 0 otherwise. In our case, the threshold equals $1/2$.

This formula essentially calculates the number of background pixels that are within the hypersphere formed around pixel $\boldsymbol{x}$.

Upon estimating $p(\boldsymbol{x}\,|O_T, BG)$, the pixel is classified as a background if the probability is above yet another threshold.

After eliminating the oldest frame in the model, we next need to determine how the pixels of the newest frame will influence the model from that point on. This is where the background adaptation happens. Even if we classified a certain pixel as part of the foreground for that specific moment, it is entirely possible that, when looking *all* of the sample pixels – be they in the background or the foreground – that it shares enough similarities between them for us to conclude that it represents a *transition* from a foreground pixel to a background one. In other words, a pixel value that has previously been in the foreground has become prevalent enough to be considered static, and therefore can eventually be considered as part of the background. So, we save it in the model as such, marking $b^{(m)} = 1$ for it.

The formula used for this second step is very similar to (1), with the key difference being that we are now considering *all* of the samples in that pixel position, instead of just background ones. We are now essentially determining the probability of a pixel being a part of the background *and* foreground. If it is high enough, it means that we can mark that pixel as a background one, since it's clear that whatever foreground object that pixel is a part of is there to stay. Of course, that's just the idea – this needs to happen for the next several frames to follow for future pixels of that value to be classified as part of a background.

The formula itself is:

$$p(\boldsymbol{x}\,|O_T, BG + FG) \approx \frac{1}{TV} \sum_{m=t-T}^{t} h\left(\frac{||\boldsymbol{x}^{(m)} - \boldsymbol{x}||}{D}\right) \quad (2)$$

Which is simplified to:

$$p(\boldsymbol{x}\,|O_T, BG + FG) \approx \frac{k}{TV} \quad (3)$$

Where k is the number of samples within the kernel formed around pixel $\boldsymbol{x}$.

C. Summary

Our workflow can be summed up as follows:

1. For every frame, using the previously-described k-NN-like model, determine whether or not a pixel is part of the background or foreground. The higher the probability of it being a background is, the closer the pixel value of the subtraction will be to 0. The lower the probability of it being, the closer it will be to a white color. After producing the results of the subtraction, update the model with the new frame pixel values accordingly.
2. Remove all pixel values on the given foreground below a certain threshold.
3. Dilate the foreground image to connect sections really close to each other to avoid presenting one object as two.
4. Remove the shapes that are too small (below a given area threshold).
5. Create a bounding box around the remaining shapes.
6. Display the bounding box on the footage.

III. OPTICAL FLOW APPROACH

A. On Optical Flow and Its Application in Motion Detection

Generally speaking, optical flow can be described as a set of vectors that estimate the "movement" of an image's pixels between one video frame and the next. Obviously, pixels do not "move", their values change on positions – however, it is by taking those changes into account that we can try and determine not only the area of movement present in the image, but the intensity of that movement.

It is from this basic principle that the idea of using optical flow came to be. While there have been approaches using optical flow in movement detection, such as [2], they seem to be more focused on the application of optical flow in image segmentation. Our approach was broader – it seemed reasonable that, if we have a measure which allows us to determine movement in an image, that we can directly use said measure in detecting movement.

As stated before, the result of an optical flow estimation is a set of vectors. The direction of these vectors specifies the direction of the pixel's movement between frames, whereas its length approximates the magnitude of that movement. The main idea of using them to detect movement was simple – after estimating the optical flow, we would keep only the vectors with magnitude of $mag > thresh$, where $thresh$ is a pre-determined threshold. We would then visually represent the remaining vector magnitudes – the areas of high magnitudes would appear as white contours. Similarly to the background segmentation method, we proceed to draw a bounding box around those contours, displaying the box on the running video frame.

Fig 3. displays this approach.

In other words, whereas the previous method relied on the extraction of foreground, this method relies strictly on detecting areas of the image with high levels of movement, with those levels estimated by optical flow vectors.

Fig 3. Optical Flow Estimation Used as Movement Detection *(Left: The places where the optical flow magnitudes are high. Right: The bounding box created around the high magnitude area.)*

Optical flow can be divided into two categories – sparse and dense. *Sparse* optical flow is based around estimating the vectors on only a few certain points in the image. Its typical usage has these points of interest change from frame to frame, with it being designed around following the directions of the calculated optical flow from the frame before. This is generally useful in determining an object's general trajectory throughout its time on camera.

Unfortunately, this means that it is now applicable for our current case. Ideally, we want to have an optical flow vector estimated for every pixel on the frame. If an object moves, all of the vectors around its position would have higher magnitudes. They would form that concentrated area we wish to create a bounding box around. With sparse optical flow, we cannot do that.

That is where the second type of optical flow comes into play – *dense* optical flow. It does precisely what we need it to, estimating the optical flow for every pixel on the image. The most commonly-used approach is the Gunner Farneback algorithm, as presented in [3]. Explained in broad strokes, it's based around polynomial estimation of each pixel's neighborhood – for both the previous and the current frame. It then tries to determine the vector $\boldsymbol{d}$ – the optical flow vector – by solving:

$$f_{prev}\,(\boldsymbol{x}+\boldsymbol{d}) = f_{curr}(\boldsymbol{x}) \qquad (4)$$

Where f_{prev} and f_{curr} are the polynomial estimations based on the previous and current frame respectively. $\boldsymbol{x}$ marks a pixel's position.

Of course, like all methods, this one is not perfect. Similarly to the background subtraction approach, it is susceptible to noise and round errors, causing spikes in optical flow magnitude where it isn't warranted. The solution is also similar to the previous method – detected contours which are blow a certain minimum area threshold are taken out of consideration before we begin drawing bounding boxes around them.

B. *Summary*

Our optical flow approach can be summed up as follows:

1. Estimate the optical flow of the current frame using the Farneback dense flow algorithm for every point on the image.
2. Determine the magnitudes of each optical flow vector.
3. Remove all of the vectors which have a magnitude below a given threshold.
4. Using the remaining areas of high magnitudes to determine object shapes.
5. Remove the shapes below a certain minimum area threshold.
6. Draw a bounding box around the remaining shapes.
7. Display the bounding boxes over the current frame.

IV. EVALUATION

The two algorithms were implemented using OpenCV in Python. Both the background subtraction method and the optical flow estimation methods were built into OpenCV, and those were used in the algorithm itself.

For testing, we used the LASIESTA [4] dataset. We found that it has the most diverse number of test-cases, with a very precise ground truth. We also ran some of the CDNET 2012 [5] dataset. However, due to the fact its ground truth is catered for some specific motion detection goals – ones which do not match ours (it is focused on change detection, not motion detection) – we used it only to study the system's general behavior.

A. *Scoring Methodology*

Our method of scoring the system's accuracy is as follows:

For every frame, we create a set of bounding boxes around the objects shown on the frame's ground truth. Those bounding boxes were then filled, to mark the area of activity. We denote this area as BB_{gt}. The same is done with the

bounding boxes estimated by a given algorithm. We denote this area as BB_{est}.

First, we calculate the absolute difference between the two:

$$BB_{diff} = abs(BB_{gt} - BB_{est}) \quad (5)$$

This leaves us with all the areas in which the estimated bounding boxes differ from the ground truth.

Next, we calculate the union of the two areas:

$$BB_{union} = BB_{gt} \cup BB_{est} \quad (6)$$

By estimating a ratio of $r = BB_{diff} / BB_{union}$, we have a ratio that tells us how much of the union is made up of ground truth deviations in an interval of $[0, 1]$. If r is 0, then there are no deviations and the estimated bound boxes match the ground truth perfectly. If r is 1, it means that there is no activity in the ground truth, but that our system has incorrectly detected movement. Everything in-between can be taken as an error rate.

We can therefore define our accuracy as:

$$\text{accuracy} = 1 - \text{r} \quad (7)$$

On frames where neither the ground truth nor the system detect any activity, we would have $BB_{union} = 0$. In those cases, we simply determine that $accuracy = 1$ for that frame.

At the end, we take the average accuracy across all of the frames of the clip as its accuracy score.

This approach, however, does have one oversight. We found that there are clips where there are long periods of no activity. Detection when there is no activity was not terribly frequent with either systems – especially not with the background subtraction one. As a result, thanks to those long stretches of no activity, by automatically setting accuracy = 1, we would boost the average accuracy, even when the system under-performed when there ultimately was activity.

To combat this, we introduced another metric – $relative\ accuracy$. It is calculated exactly as the previous one, with the only exception being that, when calculating the average, we only take into account the frames where $BB_{union} \neq 0$, thus giving a better idea on the performance when detection is expected and/or false positives.

B. Results

When using our subtraction model, we resized the frames to a width to 600. For the optical flow method, we made it to just around 400. We did these changes to get a better idea on the model's performance in case of footage with a slightly higher resolution, while still keeping it on the lower end for the sake of performance. Additionally, this lessens the impact of the estimated bounding boxes not perfectly matching the ground truth, thus making the numbers reflect the capabilities a bit more accurately.

The minimum areas for the detected objects were 2500 and 2000 square pixels for the subtraction model and optical flow model respectively. The threshold for the optical flow magnitude was set to 0.8.

In our testing, we did not include the following clips in calculating the average accuracy:

- The Camouflage (CA) indoor scenes, due to the fact they were designed to test people wearing clothes with similar color to the background *but were not moving*
- The first Modified Background (MB) clip, due to the fact it was designed to test detecting an unmoving backpack left behind
- The Simulated Motion (SM) clips for both indoor and outdoor footage, due to the fact that the movement was very intense and against the assumption of a generally unmoving camera. We did, however, use the Moving Camera (MC) outdoor clips to test how it would look for more natural, slow motion.

The results can be seen on Table 1.

On average, the background subtraction model performed significantly better. Its bounding boxes were more stable, tracked fairly well, and were more resistant to subtle changes in the footage or mistakes in the estimation. It was also generally faster, which isn't a surprise, given that the calculations the optical flow model had to do for every frame were far more taxing.

C. Observations

Based on the testing on the LASIESTA set and some clips from the CDNET dataset, we were able to spot some noticeable areas of the models' shortcomings.

The first is shadows. Even though we'd early stated that we hope to eliminate shadow influence from the background subtraction model by simply eliminating certain values, the fact is that this trick is only applicable to soft shadows. Hard shadows are inevitably noticed, due to their high contract to the background, and always detected as movement. This system that, under extremely sunny conditions, the system is likely to be a lot more imprecise.

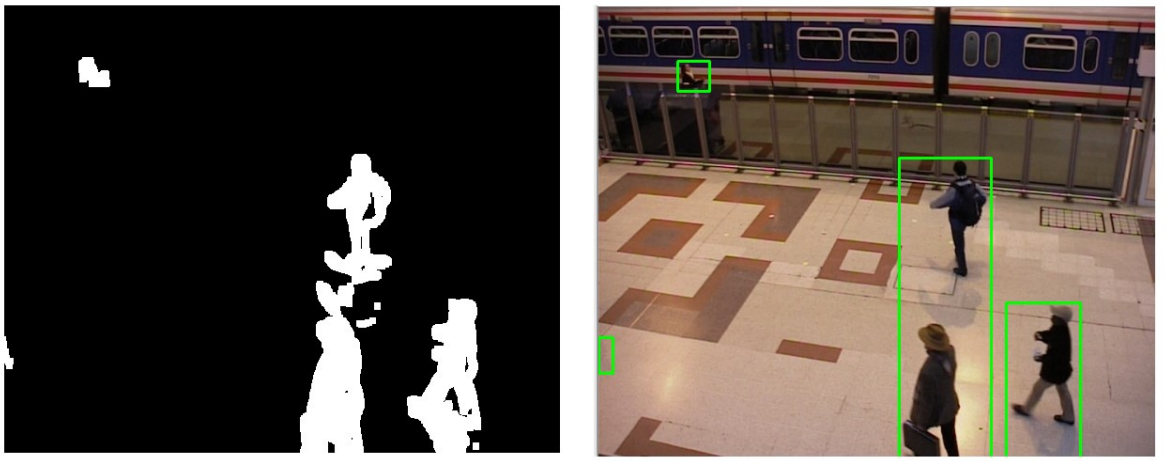
In the case of the background subtraction model, this also propagates to situations where the camera has an isometric view of the scenery with a larger crowd. Fig 4 (a) shows an

Table 1. Results

	Accuracy	Relative Accuracy
Background Subtraction	0.735	0.573
Optical Flow	0.629	0.425

example where a man's shadow causes one person's bounding box to extend to a whole other person, because their detected shadow, after dilation, connects the two people together and makes them appear as one moving object.

the same place between frames, making the system detect is as part of the background. Luckily, the rest of her body is in motion and discernable, allowing the system to still detect it. However, the issue of color is still worth noting. Fig. 4 (b)



(a)

(b)

Fig 4. Failures of Background Subtraction Detection

(a) Failure of detection due to shadows extending the bounding box (b) Failure of detection due to moving foreground object and background object have the same color

Something similar is true in the case of reflections – for instance, in the footage O_RA_01, we observe a man's reflection in the puddle he's walking on as part of the movement being detected.

Another is perspective. Due to the fact that we've put a constraint on the minimum area of detected objects and their bounding boxes, in instances where the subject is in the camera's eyeline and walking towards it from afar, it becomes more difficult for the system to detect it if, from afar, their silhouette is below the given minimum size.

While on the subject of perspective, another interesting thing was noticed in the background subtraction model. In the I_SI_02 clip, a woman is walking towards the camera. While the system does detect her (albeit with some mentioned difficulty), we notice something interesting when studying the results of the background subtraction – namely, that her shape is hollow. This is because, as she approaches the camera, a green patch of her shirt ends up being in roughly

illustrates another issue with matching colors, where the system failed to detect a part of a man's leg due to the color of his pants matching the shade of the couch in that specific area.

When it comes to weather conditions, light rain did not seem to affect either of the systems. The same could not be said for the snowy conditions. Whereas the background subtraction model was relatively resistant to false positives in footage where it snowed, the optical flow model failed completely – it was simply too sensitive.

When it came to the single moving camera test, something surprising was observed. While the background subtraction model failed completely, the optical flow model fared far better in the footage with smooth and slow camera motions.

V. CONCLUSION

We have implemented two motion detection, learning-free algorithms – one based on background subtraction, and the other on optical flow. The former has shown to be far more effective.

REFERENCES

[1] Z. Zivkovic, F. van der Heijden, "Efficient adaptive density estimation per image pixel for the task of background subtraction," Pattern Recognition Letters, Vol. 27, Issue 7, pp. 773-780, May 2006

[2] J. Huang, W. Zou, Zhu and J. Thu, "An Efficient Optical Flow Based Motion Detection Method for Non-stationary Scenes," 2019 Chinese Control and Decision Conference (CCDC), pp. 5272-5277, June 2019

[3] G. Farneback, "Two-Frame Motion Estimation Based on Polynomial Expansion," Lecture Notes in Computer Science, Vol. 2749, June 2003

[4] C. Cuevas, E. M. Yanez, N. Garcia, "Labeled dataset for integral evaluation of moving object detection algorithms," Computer Vision and Image Understanding, Vol. 152, pp. 103-117, 2016

[5] N. Goyette, P.-M. Jodin, F. Porikli, J. Konrad, P. Ishwar, "changedetection.net: A new change detection benchmark dataset," IEEE Workshop on Change Detection at CVPR-2012, June 2012

[6] G. Bradski, "The OpenCV Library," Journal of Software Tools, January 2000