Detecting fire in video stream using statistical analysis

Karel Koplík[1], *Peter* Janků[2,*], *Olga* Voznyuk[2], *Tomáš* Dulík[2], *Petr* Snopek[3]

[1]Brno University of Technology, Faculty of Information Technology, Czech Republic
[2]Tomas Bata University in Zlin, Faculty of applied informatics, Nad Stranemi 4511, 760 05 Zlin, Czech Republic
[3] UNIS, a.s., Jundrovská 33, 624 00 Brno, Czech Republic, Czech Republic

Abstract. The real time fire detection in video stream is one of the most interesting problems in computer vision. In fact, in most cases it would be nice to have fire detection algorithm implemented in usual industrial cameras and/or to have possibility to replace standard industrial cameras with one implementing the fire detection algorithm. In this paper, we present new algorithm for detecting fire in video. The algorithm is based on tracking suspicious regions in time with statistical analysis of their trajectory. False alarms are minimized by combining multiple detection criteria: pixel brightness, trajectories of suspicious regions for evaluating characteristic fire flickering and persistence of alarm state in sequence of frames. The resulting implementation is fast and therefore can run on wide range of affordable hardware.

1 Introduction

In this paper, we present new algorithm for detecting fire in video based on tracking suspicious regions in time with statistical analysis of their trajectory. Our goal was to develop a fast algorithm which can run on an affordable hardware with minimum rate of false alarms.

The purpose of detecting fire in video is to improve current systems based on traditional fire and smoke detection sensors, which are limited in several ways: e.g., they cannot be used in industrial plants, where burning is an inherent part of the manufacturing process and/or where the occasional occurrence of smoke cannot be avoided. The current sensors have also relatively short range: they need to be close to the potential fire, so there must be many of them in order to cover all dangerous places.

Cameras can monitor much larger areas and their operation can be more reliable then chemical or optical smoke sensors, which need to be regularly checked, cleaned or exchanged. Therefore, it is desirable to develop a camera-based fire detection system which can become a welcomed alternative to traditional fire and smoke detection sensors.

Of course, there are already other systems for fire detection in video streams. What makes our system different is that it is very fast and doesn't require too much computation power.

2 Detailed description

The entire algorithm is very simple. It consists of the following steps:

1. Detect suspicious regions
2. Find bounding rectangles
3. Track rectangles in time
4. Analyse trajectories
5. Check for persistence

In the following sections we will look at each step. To demonstrate, let's use a video of a burning tree as an example.

Fig. 1. Burning fire.

2.1 Detecting suspicious regions

The regions of interest in this case would be those pixels which have fire-like colour and also pixels which change

* Corresponding author: janku@fai.utb.cz

visibly in time (i.e. contain movement). Suspicious pixels are then a union of these two sets.

2.1.1 Detecting pixels with fire-like colour

Finding pixels based on colour is very simple and it can be done using different colour models like RGB, YUV, YCbCr HSI or HSV. [1][6][7] First two focus on colour spectrum and latter two on colour intensity. Fire is usually the brightest part in the video so using brightness as an additional criterion is very useful.

In our algorithm we work with RGB colour model and the procedure looks like this: We take red (R), green (G), and blue (B) channels and calculate colour saturation (S).

Then the following rules are applied:

$$R > G > B$$
$$S > S_T$$

where:

S_T is saturation threshold.

Fig. 2. Fire colour map-

2.1.2 Detecting movement

To detect movement we actually need three frames. Because we need the previous, current and the next frame, the algorithm is always one frame behind a real-time video stream from a camera.

Taking only greyscale frames, we basically subtract the background to detect motion in foreground. First we calculate an absolute difference between previous (fi-1) and next (fi+1) frame. This will subtract the background. Then we calculate an absolute difference between current (fi) and the next frame. This will update that information. Then we apply binary AND operator on the two calculated differences to obtain information about the movement in the foreground.

$$| f_{i-1} - f_{i+1} | \wedge | f_i - f_{i+1} | \quad (1)$$

It is also necessary to filter out lone pixels (i.e. noise) in the result movement map. The amount of noise depends on camera and lighting conditions.

Fig. 3. Movement map.

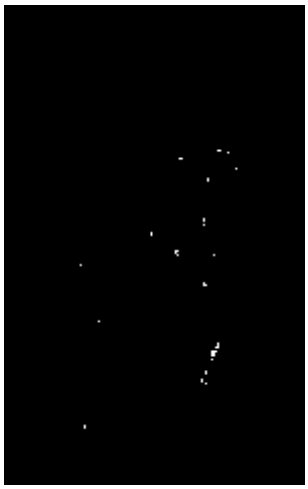

Fig. 4. Suspicious areas.

2.2 Founding bounding rectangles

The binary map of suspicious pixels will most likely consist of lots of small disconnected areas concentrating in separated larger regions. In one of our first attempt, we tried to find bounding rectangles for each of them and then to connect them into larger bounding rectangle if they were close enough to each other. This worked but also turned out to be very costly in terms of processing time. The solution was to dilate the small areas using a morphology operator.

Dilatation (as a morphological operation) consists of convoluting an image with a small kernel shaped like a simple shape (circle, square, etc.).

This results into much fewer bounding rectangles to be found and aggregated in the next step and significantly increases the algorithm speed.

2.3 Tracking suspicious regions in time

Similarly to [8] we test suspicious regions fire-like characteristic in time. Our method consists of tracking. Before we start we need to aggregate found areas in space and then link them in time.

2.3.1 Aggregating regions in space

After we are done marking up areas of interest in each frame we need to look into previous frames to find out if we can link them to previously marked areas.

For that we use a tree data structure consisting of two layers of bounding rectangles.

In first frame, we add rectangles to the root of the tree. If a rectangle to be added overlaps (with added tolerance) any rectangles stored in the tree, they are all replaced with a rectangle which is based on the largest one and is large enough to contain all of them. The parent rectangle is not reduced in size in this step and they are added as its children.

If they already have children, the children are moved to the new parent and the original parent is discarded.

Once all the rectangles have been added to the tree, the bottom layer is removed. Before that we check if each top-layer rectangle has any children.

If it has, we find the smallest bounding rectangle for all children and resize the parent to the average between the original and the smallest size.

2.3.2 Linking regions in time

If the parent ends up with no children, we reduce its RTL (right to live) parameter. If it has at least one child, we will set RTL to predefined maximum value.

The RTL parameter basically tells us how long we will keep a rectangle which has not been updated (e.g. for 3 frames).

In all of the following frames, we repeat the same steps as with the first frame, but now we have preexisting first-layer.

2.3.3 Tracking regions in time

The tracking is based on keeping the history of middle points. These are not the centre points of bounding rectangles. Instead, we take the original map of suspicious regions (without applied dilatation) and average the coordinates of suspicious pixels. The new average point is then added to stack.

The reason we use the stack is that we remove points if their count surpasses defined limit.

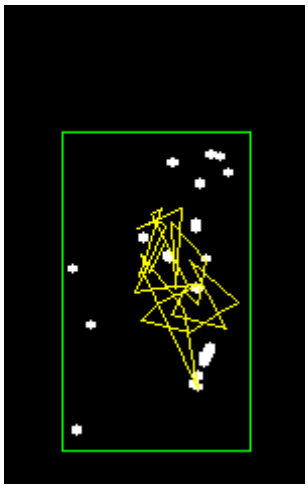

Fig. 4. Suspicious region's trajectory.

2.4 Analysing trajectories

This and the following step are very important. Without them, we would be detecting a large spectrum of objects and features which are similar to fire in colour and in "not being static". The fire, however, has a very specific way of moving. First of all, it stays in one place and is characteristic by constant flickering. This is very useful because it's hard to find anything else what is (yellow and bright and) constantly changing shape while staying in one location. The trajectory of the middle points should basically fit the normal distribution.

To test this with our suspicious regions, we use horizontal and vertical coordinates of the middle points. First we calculate the mean value μ and the standard deviation σ for each axis. If the distribution is normal, according to the gaussian curve, 68.2% of values should belong to the interval:

$$(\mu-\sigma,\mu+\sigma) \quad (2)$$

Since we work with object that doesn't only change shape but also size over time, it is not wise to rely on too many values to increase precision. Instead, we work with fewer values (according to our tests, the optimal number seems to be around 100) and expect some reasonable error. That is why we had settled to expecting 60% of data to fall into the above specified interval.

2.5 Checking for persistence

To further eliminate false alarms, we apply the last criterion, which is persistence. If the previous step gives us positive result, we don't trigger the alarm yet but save it. This we do for each frame.

Depending on the video source quality and captured scene properties, we usually set the persistence limit to at least 15 seconds (which would be 375 frames for 25 fps video source) or even more. With such setting, we found our algorithm gives satisfactory low false alarm rates.

Longer persistence limit could result in longer delay or missing short positive detections.

To evaluate persistence, we don't require the positive test on normal distribution in each frame. It can be just some higher percentage. Empirically, we had set it to the top quarter.

Fig. 6. Output screen.

3 Algorithm parameters

There are several parameters which need to be set after a camera providing video stream for our algorithm is installed into a new environment.

3.1 Brightness threshold

Defines the minimum brightness needed for classifying pixels as fire-coloured. Normally the best value would be around 225 (out of 255) but in some test videos it had to be adjusted because the fire was too dark due to the type of video source.

3.2 Movement threshold

During the movement detection processing the camera noise is reduced by filtering out some lone pixels. This parameter sets the threshold. Usually in badly illuminated areas the threshold needs to be set higher. This is important because noise has normal distribution so it could cause a false alarm.

3.3 Small regions gap size

Maximum distance between two bounding rectangles of small regions that can be aggregated. This parameter should have a small value (e.g. 10 px) and depends on video resolution.

3.4 Regions minimal size

If there are too many very small regions found because of camera noise, we eliminated them by setting a minimal bounding rectangle size. If the size is set too high, small flames will not be detected. If it's too low, the noise can trigger false alarm.

3.5 Region "right to live"

As was explained in the previous chapter, this parameter sets how long we keep a region appearing to be empty in a few frames.

This mostly happens when there is no movement detected. Since the fire has multiple independently moving flames it can be expected that with high probability, it will not stop moving for more than few (e.g. 3) frames.

3.6 Number of middle points

The parameter defines how many middle points we want to remember for each region before we test them for normal distribution. It should be enough for this criterion to be calculated relatively accurately but also not too many so we adapt to changes in the scene quickly. We use empiric value of 100.

3.7 Persistency interval

This parameter determines the number of frames, for which a positive detection has to persist before triggering the alarm. This parameter needs to be changed according to number of frames per second in each video stream.

The higher this number is the harder it is to fool the algorithm but also the longer delay we get. Fire keeps its characteristic movement all the time, but most other objects and phenomena move chaotically and in short intervals. In our tests, 15 seconds turned out to be sufficient minimal length.

4 Performance

To test our algorithm, we had acquired several videos containing flames, moving bright objects and phenomena similar to fire.

4.1 Algorithm speed

Our algorithm is very simple so when optimized, it runs in real time on average hardware. It does not rely on frequency of fire flickering unlike some algorithms so the frame rate doesn't have to be that high either.

How fast the algorithm detects flame really depends on the camera setting and scene characteristics, but in ideal conditions it depends solely on the defined length of the minimal persistence interval.

Not ideal conditions would be e.g. the flame is too small or too far from the camera, if the flame is not fully visible in camera's field of view or the flame movement is being distorted by another moving object (in one test video it was a burning piece of wire rolling out of flames).

4.2 Strengths and weaknesses

Similarly to other fire-detecting algorithms, the most concerning weakness is false alarm ratio. Avoiding false alarms is implemented by combining multiple criteria

[2][3][4][5]. If all criteria are fooled, the false alarm occurs.

During our testing we had encountered some problematic scenes. To test the constant movement with normally distributed middle points of suspicious regions we had used some videos of dancing people. Interestingly, some types of dance don't have the normal distribution of movement and some do.

In the end, none of these videos triggered the alarm but some were really close. When dancing, the movement often briefly stops and that makes the persistence criterion come out negative.

What was most problematic was white background in some videos. Anything what moves in front of such background can pass the movement-and-brightness criterion because even when the object is not bright the algorithm sees only bright areas changing shape as the object covers and uncovers the background.

Other concern we have is with natural phenomena similar to fire. When testing the algorithm on videos of sunset being reflected on moderately moving ocean surface, the false alarm was triggered every time. This means that our algorithm cannot be used at locations where something like this occurs. However, it can be useful in locations such as a factory hall or a tunnel where these phenomena don't appear.

Other than that, our algorithm proved to work without problems and was almost impossible to fool intentionally when processing real time camera input.

5 Conclusion

We had developed and successfully tested a new algorithm. In this stage in can be used with any camera system and will work mostly without error. However, there are certain situations in which the algorithm triggers false alarm. This will be the subject of our future research.

Acknowledgement

This paper is supported by Technology Agency of the Czech Republic (TA ČR) within the Visual Computing Competence Center - V3C project No. TE01020415, by Ministry of Education, Youth and Sports of the Czech Republic within the National Sustainability Programme project No. LO1303 (MSMT-7778/2014) and also by the European Regional Development Fund under the project CEBIA-Tech No. CZ.1.05/2.1.00/03.0089.

References

1. T. Chen, P. Wu, and Y. Chiou, "An Early Fire-Detection Method Based on Image Processing," *Proc. IEEE Int. Image Process.,* (2004), pp. 1707-1710.
2. B.U. Toreyin, Y. Dedeoglu, and A.E. Cetin, "Flame Detection in Video Using Hidden Markov Models," *Proc. IEEE Int. Conf. Image Process.,* (2005), pp. 1230-1233, 2005.
3. W. Krüll et al., "Design and Test Methods for a Video-Based Cargo Fire Verification System for Commercial Aircraft," *Fire Safety* J., **41**, no. 4, (2006), pp. 290-300.
4. T. Celik, "Fast and Efficient Method for Fire Detection Using Image Processing," ETRI Journal, **32**, no. 6, (Dec. 2010), pp. 881-890.
5. L., Zhigang, G.Hadjisophocleous, G. Ding and Ch.S. Lim. Study of a Video Image Fire Detection System for Protection of Large Industrial Applications and Atria [online]. (cit. 2013-12-01).
6. Poobalan, Kumarguru and Liew, Siau-Chuin. Fire Detection Algorithm using Image Processing Techniques. In: E-Proceeding of the 3rd International Conference on Artificial Intelligence and Computer Science (AICS2015), 12-13 (October 2015) , BayView Hotel, Penang, Malaysia. pp. 160-168.. ISBN 978-967-0792-06-4.
7. Jiang, B., Lu, Y., Li, X. et al. Multimed Tools Appl (2015) **74**: 689. doi:10.1007/s11042-014-2106-z.
8. A. E. Gunawaardena, R. M. M. Ruwanthika, A. G. B. P. Jayasekara, "Computer Vision Based Fire Alarming System", in Proceedings of the 2nd International Moratuwa Engineering Research Conference (MERCon), pp. 325-330, Moratuwa, Sri Lanka, (April 2016).