FIRE DETECTION USING GRAYSCALE VIDEO PROCESSING USING BLACK AND WHITE VIDEO CAMERAS

Research · April 2015

DOI: 10.13140/RG.2.1.1420.7526

1 author:

Gaurav Kishor Deshmukh
Guidewire Software
2 PUBLICATIONS **0** CITATIONS

FIRE DETECTION USING GRAYSCALE VIDEO PROCESSING USING BLACK AND WHITE VIDEO CAMERAS

by

GAURAV KISHOR DESHMUKH

A research project submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science

Leonard Brown III, Ph.D., Committee Chair

College of Business and Technology

The University of Texas at Tyler
December 2014

The University of Texas at Tyler
Tyler, Texas

This is to certify that the Master's project of

GAURAV KISHOR DESHMUKH

has been approved for the research project requirements on

_____

For the Master of Science degree

Approvals:

_____
Committee Chair: Leonard Brown III, Ph.D.

_____
Member: Arun D. Kulkarni, Ph.D.

_____
Member: Naryanan Subramanian, Ph.D.

_____
Chair, Department of Computer Science

Acknowledgements

First of all, I want to express my deepest gratitude to my advisor, Dr. Leonard Brown III, for his continuous and caring help, advice and encouragement during the two years of study at The University of Texas at Tyler. I thank him for being very patient with my doubts and my progress, for countless lessons on research and technical writing methods.

I wish to thank my research project committee members Dr. Arun Kulkarni and Dr. Narayanan Subramanian for their willingness to allot their time and effort to review my research report and expert advice given for improvement of my research.

I thank the faculty and staff of the computer science department at The University of Texas at Tyler for their constant support throughout the course of my study. I would also like to extend my gratitude towards my family and friends for their immense support during the course of research project as well as my academic career.

I would also like to thank The Office of Graduate Studies, University of Texas at Tyler for their financial support through Scholarships and Assistantships offered to me during my academic career.

Table of Contents

List of Figures

List of Tables

Table of Equations

Abstract

FIRE DETECTION USING GRAYSCALE VIDEO PROCESSING USING BLACK AND WHITE VIDEO CAMERA

GAURAV KISHOR DESHMUKH

Committee Chair: Leonard Brown III, Ph.D.
The University of Texas at Tyler
December 2014

Fire is one of the destructive forces of nature. Whenever a fire occurs, there can be great damage to the lives and economy of a community. Early detection of fires is crucial to minimize the amount of damage caused when fire spreads. Consequently, techniques have been developed to prevent these calamities. Researchers have come up with various algorithms for fire detection using video processing. Most of the proposed algorithms use fire-colored pixels as the distinguishing parameter, which means that they focus only on a certain kind of fire. Other kinds of fire, such as chemical fires, may go undetected. Thus, these earlier proposed algorithms fail completely in such fire situations. The goal of this research is to ignore fire color and focus more on other peculiar characteristics of fire that could prove more efficient in the detection of fire. Therefore, the fundamental problem of this work is to devise a holistic approach to detect fire using video sensors. This research project uses grayscale video processing devices, which is more convenient to develop an algorithm to determine the self-luminous feature of fire. Variance in the brightness, which can be observed as flickering by the human eye can also be a vital parameter to enhance the efficiency of fire detection. As fire is shapeless, randomness of edges of fire can be another feature to improve the rate of detection. The central objective of this research is to design and implement fire-detection algorithms for a grayscale camera.

1. Introduction

Fire is one of the destructive forces of nature. Whenever fire occurs, there can be great damage to the lives and economy of a community. Early detection is crucial to minimize the amount of damage caused when fire spreads. Consequently, techniques have been developed to prevent these calamities. Today, the use of sensors is prevalent to detect fires. Commonly used sensors include thermal sensors which are affordable and efficient. Multi-sensors and smoke detectors are used commercially. However, these techniques are limited in their ability to detect fire primarily because they must be in the proximity of fire to detect it. Several sensors are needed to detect fire occurring in a large area. Thus, they are inefficient for early detection, so there is a need of new techniques to overcome this limitation.

A more efficient method used to detect fire is by the use of video sensors. Video sensors or cameras have the ability to show the real time environmental condition of any area under surveillance, and they can monitor large areas. As time progresses, different technologies in cameras are becoming easily available. There are thermal, visual, and infra-red cameras readily available in the market. Of these, visual cameras are low cost, easily available and simple to implement. Therefore, researchers are focusing more on the implementation of visual cameras in detecting fire [6]. Therefore, researchers have come up with various algorithms for fire detection using video processing. Most of the proposed algorithms use fire-colored pixels as the distinguishing parameter, which means that they can focus only on one particular kind of fire [1]. Other kinds of fire, such as chemical fires, may go undetected. Thus, these earlier proposed algorithms fail completely in such fire situations.

The general approach for visual fire detection is to detect the basic characteristics of fire. Fire has many peculiar characteristics like luminosity, color, motion, quasi-periodic nature, and fractal like structures. Some of these characteristics can easily be detected by making use of computer vision algorithms. Using a visual camera acting as a sensor, it is possible

to use these algorithms to easily detect fire from a distance and prevent fire from spreading and potentially causing greater losses. Detection can be enhanced using artificially intelligent systems. With these types of techniques, not only can fires be detected, but also the approximate range to which a fire can spread may be determined. This project attempts to devise an algorithm which could prove as a holistic approach to fire detection using video surveillance. This research work ignores fire color and focuses more on other peculiar characteristics of fire that could prove more efficient in its detection. As this research project uses grayscale video processing devices, it could be more convenient to develop an algorithm which will determine the self-luminous feature of fire. Variance in the brightness, which could be observed as flickering by the human eye can also be a vital parameter to enhance the efficiency of fire detection. Inclusion of the randomness feature of fire can also prove beneficial in improving the detection rate.

The remainder of this report is as follows. Section 2 has a brief review and explanation of approaches adopted in related prior work. Section 3 consists of an explanation of the algorithm adopted in this project. Section 4 has a discussion on the conducted experiments for verification. Finally, conclusions are drawn in the last section.

2. Literature Review

In this section, the previous work in the field of visual fire detection is reviewed. The literature in this area primarily focuses on the color feature of fire. Color based approaches are widely used and are prevalent in literature [2 - 15]. However, in most of these algorithms other kinds of fires such as chemical fires may go undetected because the flames may be of a different color. Most of these color approaches are based on the red-green-blue or RGB color model or some combinations of other color models with RGB [4, 9, 10, 12]. This is the most basic model used in computer graphics. This model is very close to how humans perceive color in the visual spectrum. In this model, any color is the intensity level of the colors red, green, and blue. The range of intensity values for each color is 0–255. The RGB color model may be easily implemented as cameras already use RGB color model for its image processing. Previous studies identify fire based upon RGB values and fail to consider the luminosity and chrominance feature of fire.

Some of the algorithms in literature are also based on other color models such as YUV, HSI, HSV, HSL and CIE Lab as well [2, 11, 13, 15]. The HSV, or HSI, model describes colors in terms of hue, saturation, and value (Intensity). Hue is a term describing a pure color, that is, a color not modified by tinting or shading. Hues are formed by combining two primary colors. When two primary colors are combined in equal intensities, the result is a new color. A saturation is produced by "lightening" a hue. This is equally good as adding white. Full saturation produces no tint, while zero saturation produces white, a shade of gray, or black. An Intensity "dims" a hue. The HSL model describes colors in terms of hue, saturation, and lightness (also called luminance). The definition of saturation in HSL is substantially different from HSV, and lightness is not the same as intensity. The transition from black to a hue to white is symmetric and is controlled solely by increasing lightness. This improves the color distinguishing ability. As noted above, luminance is not intensity, thus using this model one cannot extract the self-luminous feature of fire. In addition, this model considers luminosity as the constant for partial ranges of hue and saturation, which fails to detect different brightness patterns in fire.

'Y' in YUV stands for the luma component (the brightness) while U and V are the chrominance (color) components. Luminance is denoted by Y and luma by Y'. The prime symbol ' denote gamma compression (for electronic devices) with "luminance" meaning visual spectrum brightness, while "luma" is electronic (voltage of display) brightness. The digital version of YUV color model is YCbCr is more or less derived from it and is sometimes called Y'UV. Cb and Cr are blue–luminance and red–luminance differences. The Y'UV (YCbCr) components are derived from RGB components using the following matrix multiplication as follows:

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad \textbf{(1)}$$

Observing the matrix derivation of YCbCr model, the Y has 58% of Green (G) from RGB that means it gives more significance to the green component. This may lead to detection of green objects also as potential fire, increasing the false alarm rate.

Many of the approaches in literature are based on an approximate threshold of combination of color parameters to detect fire [5, 7, 9, 13]. This makes the approach unpredictable and ineffective in fire detection. Most of the approaches are assuming fire to be of yellowish – red in color, which may not always be true [2 -15]. Mixtures of different chemicals in the surrounding area may cause fire to be of variant colors [1]. Most of the papers use other features of fire as a filtration mechanism to reduce the false detection rate such as flickering, self-luminosity and fractal like nature of fire. Since the color is a more crucial aspect of color based algorithms they may break down when the color of the fire changes. Some of the approaches reviewed also use other features of fire for its detection. Features, such as variance in the fire pixels, are observed as flickering by the human eye. This is another characteristic that can be used to identify fire in a given set of video frames. [2] emphasizes more on variance in the fire pixels. The color histogram is used for variance detection and analysis of fire pixels. [10] uses the intensity difference between two

consecutive frames to estimate the amount of flickering which is compared to an experimental threshold value. [4] describes an integrated fire detection and suppression system. It uses the RGB color model with Gaussian density functions to determine the candidate pixels. It also uses a motion orientation feature of the fire to determine the direction in which it may spread. Therefore, motion not only helps distinguish fire from static images but additionally may determine the direction in which fire may spread becoming a vital characteristic in fire detection algorithms.

If one observes vigilantly in any fire video, the edges of fires seem to be random and have a fractal like structure. This could be another characteristic utilized to detect fire. [2] uses the color histogram to detect contours in the edges of the fire. In [10] the approach focus is more on detecting fire related features. This approach uses the mixture of RGB and HSI color spaces. Then motion of the fire flame is detected using multi-Gaussian background to foreground differencing method. Then it detects the external features like the flame's circularity, flickering and spatial wavelet energy. [12] segregates the real burning fire from fire like images using chromatics, statistical and dynamic features. In this approach, the characteristics of fire are randomness of area size, fractal nature of the boundaries, coarseness and skew of the surface. It also considers the distribution of space occupied by fire, which can be crucial in reducing false detection rate.

Some of the literature surveyed also uses a peculiar kind of classification mechanism with the help of neural network or support vector machines. [4] uses the LMS algorithm for classification of final candidate pixels. In [9], the approach uses the orientation feature of the fire. Using the rule-based threshold on the RGB color model, candidate pixels are determined. Then the set of pixels from each frame are given to Support Vector Machine (SVM), which classifies the candidate pixels and reduces the false positives. In [11], the technique of feature extraction is used and it passes the data to the trained neural network to detect fire in the given video. It focuses more on the characteristic motion of the fire pixels providing conversion of RGB frames into images pertaining more of fire like colors. To extract fire like colors, the HSV color model is used in this approach. Then, the motion

in the given video sequence is analyzed using the two proposed techniques, namely Optical Flow Estimation which detects and analyzes the similarities and the flow of pixels in the given video frames and Feature Extraction which is concerned with the detection of fire like features from the video. The output from the previous steps is given to the trained neural network, which classifies the non-fire frames from fire pixels. In [12], the technique of neural network is used to differentiate between and classify non-fire pixels and fire pixels. Fire pixels are the pixels in a given video frame which have the fire like characteristics that are defined in the algorithm. For each considered frame, a Potential Fire Mask (PFM) is generated using the Gaussian color model. It segregates the fire pixels using other characteristics of fire as explained earlier. For further classification, these fire pixels are given to a trained neural network like Bayes classifier.

3. Proposed System Overview

This section describes the components of the system. It focus on the general structure and describes the working of each component individually.

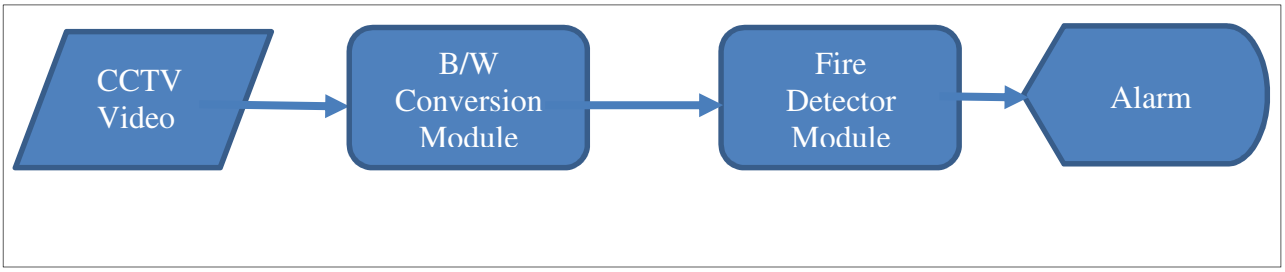

Figure 3.1: Proposed system overview

The figure above shows the systematic design which helps in the detection of fire. It consists of four major modules.

1. **CCTV Video module**

   This module is concerned with the video data processing required for the system to work. Its major role in the system is to read the video data from CCTV camera or user specified video source.

2. **B/W Conversion module**

   The camera may use different formats or configurations for the processing of raw video data. For the system to work, it needs the data to be homogeneous with the same format and configuration. This module converts the video data to RGB format, which makes further processing of video data easier.

3. **Fire Detector module**

   This is the module concerned with the main functionality of the system. It performs frame and pixel analysis and helps in the classification of fire pixels. These analyses will be explained in detail later in this section.

4. **Alarm module**

   This module raises an alarm on the detection of fire in the given video.

3.1 CCTV Video Module

The CCTV video module reads the input from a given video source and sends the frame to the next module. It uses the OpenCV library function videoCapture ( ) as a black box. This module reads multiple video frames and sends only a single frame at a given instance.
The figure below explains the transfer of each frame to next module from CCTV module.

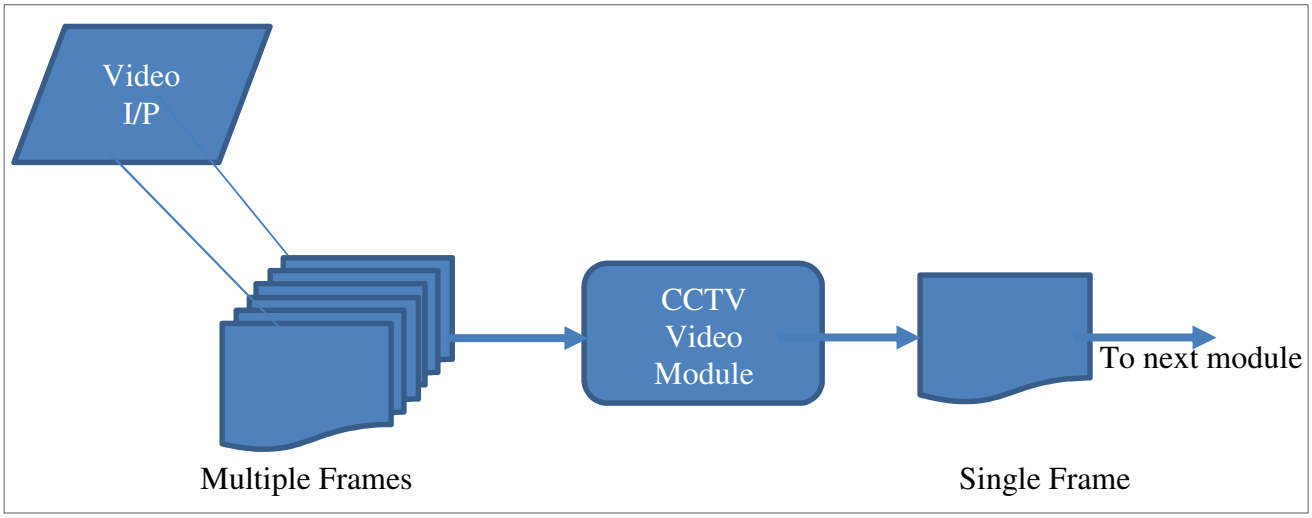

Figure 3.2: CCTV Video Module

3.2 B/W Conversion Module

The frame from the CCTV module is sent to this module. In this module, for each pixel in the given frame, every RGB value is converted to the corresponding grayscale value. This is done using cvCvtColor ( ) function from OpenCV library as a black box. This function uses Equation (2) for its conversion.

$$Y = 0.299 * R + 0.587 * G + 0.114 * B \quad \textbf{(2)}$$

As observed in the above equation, the gray scale value has 29.9 % of R value, 58.7% of G value and 11.4% of B value. The greater amount of G – component in coressponding Gray pixel makes luminosity factor of fire easily detectable.

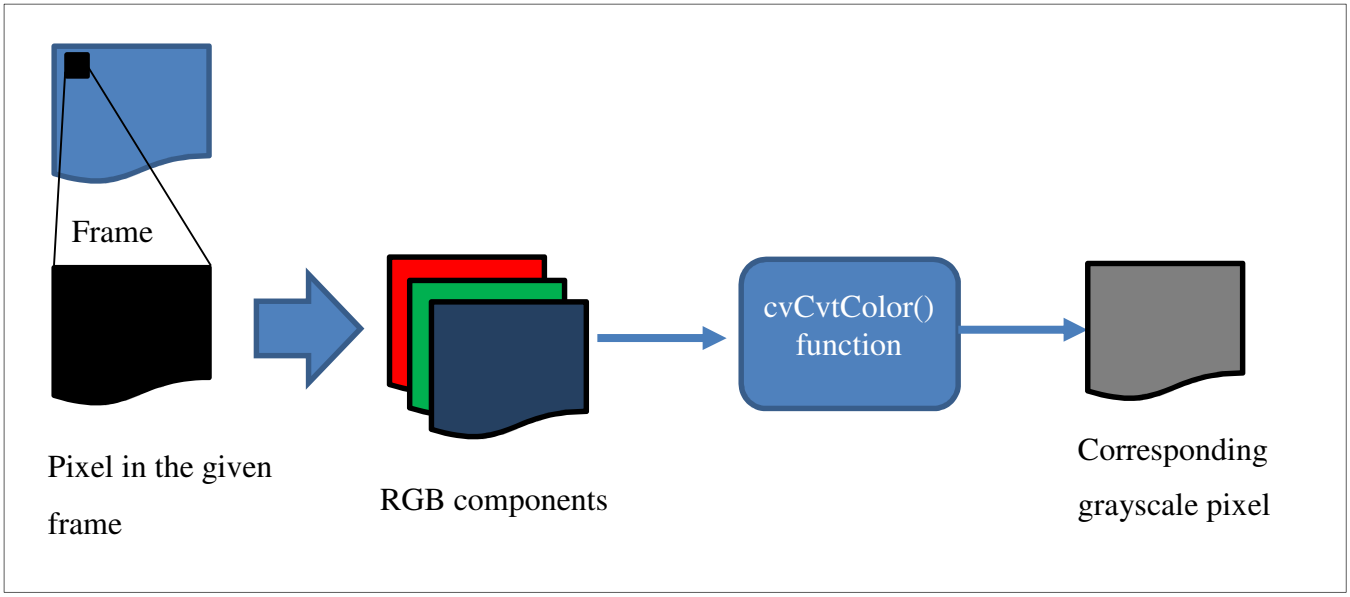

Figure 3.3: CCTV Video Module

3.3 Fire Detection Module

This module is a crucial component of the system. It is concerned with the pixel and frame analysis, which are two basic methods used in the classification of fire pixels from background pixels and non-fire pixels. Thus this module can be subdivided into these two analysis components and a classifier component.

3.3.1 Frame Analysis Component

The frame analysis component is concerned with the frame by frame analyisis of the given video frame, which is sent to the fire detection module by the previous module. This module performs the comparison between two consecutive frames.The tasks of this module are further divided into background subtraction and contour analysis.

a. Background Subtraction

The main idea is to maintain a static background image of the monitored scene, then to identify the motion object by using the current image to subtract the background. It is based on the fixed camera, and its detecting effect depends on the reliability of the static background image. Some approaches surveyed in [6] follow this method for motion detection. The approaches mentioned in [6] are based on the temporal frame differencing method, in which the difference in frames is calculated and detects moving objects in the video. However, this approach can have many false detections because even a static fire image such as a photograph moved by a person can be detected as potential fire. Hence, just using this approach for motion detection is undesirable. This approach is explained with an example illustrated in Figure 3.4.

Suppose the prior frame is the N-1th frame, the current Nth frame and upcoming N+1th frame. The background frame serves as a template for the Nth frame to compare the differing pixels. The pixels may change between consecutive frames. This is either due to motion, introduction of a new object in the scene or the possible flickering of the object in the scene [10]. The intensity of the pixel changes is calculated using Equation (3).

$$D(x, y) = \sum_{i=2}^{n} | f_i(x, y) - f_{i-1}(x, y) | \quad \textbf{(3)}$$

In the above equation *D(x,y)* is the candidate flame pixel, $f_i(x, y)$ represents the value of i^{th} frame pixel and $|(f_i(x, y) - f_{i-1}(x, y))|$ represents the absolute value difference between the previous frame and the current frame in the same coordinate [10].

The literature suggests that motion and flickers are the fundamental characteristics of fire [3,7,10]. Therefore, it is one of the most important tasks of this analysis to narrow down the number of the candidate pixels solely to pixels possesing similar property of both having motion and flickers. Such pixels could prove to be strong candidates for fire pixels.

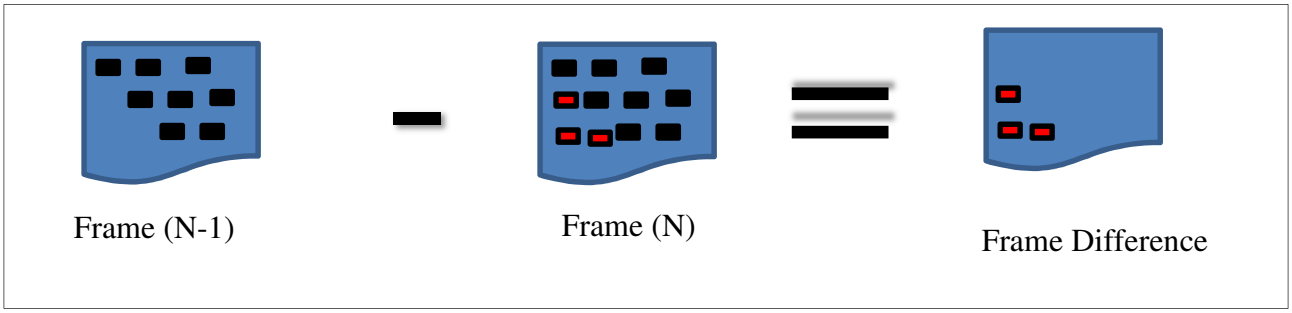

Figure 3.4: Background subtraction

This method of eliminating candidate pixels from a static background frame is called background subtraction. This is used for detection of motion in the given video data sequences in many algorithms in the literature[6].

b. Contour Analysis

Frame analysis helps find the contours of the objects which are either moving or flickering. The contours of candidate pixels help determine the space-time fluctuation of the fire. These contours may prove efficient to further limit the candidate pixels [2].
The contours are the boundaries or edges of the pixels in the motion. This helps detect the extent of the candidate pixels and narrow them down which may also prove beneficial in locating the position of true fire pixels in a given frame.

The system uses the findContours( ) function of the OpenCV library. This function returns the vector of the boundary pixels from the given frames. Contours are only drawn for the pixels in motion and are within the boundaries mentioned in the vector given by the OpenCV funtion findContours( ).

The combination of these two tasks narrows the number of false positives and improves the efficiency of the system. Now at this stage, pixels in motion are fetched from the background subtraction component and contour component. A pixel-by-pixel logical AND

operation is performed for every pixel in frames from both components to get a final frame with maximum potential fire pixels. Equation (4) illustrates the AND operation.

$$D(x, y) = \sum_{i=0}^{n} B_i(x, y) \,\&\, C_i(x, y) \quad \textbf{(4)}$$

In Equation (4), $D(x,y)$ is the new frame generated by performing the AND operation between pixels , n is the number of pixels in each frame, B_i and C_i are the frames from the background subtraction component and from the contour analysis component repectively. x and y are the pixel coordinates. To perform the AND operation as mentioned in Eqaution (4), the cvAnd( ) function from OpenCV library is used.

3.3.2 Pixel Analysis Component

The pixel analysis component is responsible to detect variance in the intensities of the pixels. To capture this vital characteristic of fire, pixel by pixel analysis is performed.
For pixel analysis, the Canny edge detection algorithm is used [16]. For this purpose, the Canny( ) funtion in the OpenCV library is used. The Canny function contains a number of adjustable parameters, which can affect the computation time and effectiveness of the algorithm. Two of the important parameters passed to this funtion are aperture size and thresholds.

The smoothing filter used in the first stage directly affects the results of the Canny algorithm. Smaller filters cause less blurring and allow detection of small, sharp lines. A larger filter causes more blurring and smears out the value of a given pixel over a larger area of the image. Larger blurring radii are more useful for detecting larger, smoother edges. In the system, a value of 3 is used for aperture size, as it is recommended by the manual book for OpenCV [17].

The use of two thresholds allows more flexibility compared to a single-threshold approach, but general problems of thresholding approaches still apply. A threshold set too high can

miss important information. On the other hand, a threshold set too low will falsely identify irrelevant information. It is difficult to give a generic threshold that works well on all images. No tried and tested approach to this problem yet exists. The optimal solution is required to address this issue of the function.The thresholds are taken as maximum value of 100 is used, so that the smallest possible edge from the given video frame can be detected.

Now at this stage, pixels in motion are fetched from the frame analysis component and a pixel-by-pixel AND operation is performed for every pixel in a given frame to get a final frame with the maximum potential fire pixels. This can be illustrated by Equation (5).

$$D(x, y) = \sum_{i=0}^{n} P_i(x, y) \,\&\, F_i(x, y) \quad \textbf{(5)}$$

In the above equation *D(x,y)* is the new frame generated by performing logical AND between pixels, *n* is the number of pixels in each frame, P_i and F_i are the frames from the pixel analysis component after canny edge function and frame fetched from the frame analysis components, repectively. *x* and *y* are the pixel coordinates. To perform this logical AND,the cvAnd( ) function from the OpenCV library is used. This frame generated by the logical AND is an output from this module.

The combination of these two tasks narrows the number of false positives and improves the efficiency of the system. As the system can narrow down candidate pixels to those which flicker and also are in motion as potential fire pixels.

3.3.3 Classifier Component

There is a third small crucial component of the fire detector module. This module helps in reduction of false positives. The system applies the method of thresholding to limit the minimum intensity requirement for the candidate pixel. The threshold value of 180 is set for the output frames from pixel and frame analysis. To perform this thresholding,

cvCmpS( ) function of the OpenCV library is used. If the frames have some pixels, whose value intensity values is less than 180, then those pixels are ignored.To achive this binary thresholding function, CvThreshold( ) is used. Then the AND operation is performed using the CvAnd( ) function of the OpenCV.

3.4 Alarm Module

The alarm module is concerned with raising an alarm at the detection of fire in the frame under consideration. This module continuously checks for fire pixels in the final frame submitted by the classifier component. Once the potential fire frame is detected, it raises an alarm to denote the presence of fire.

3.5 Working of the System

In this section, the individual modules and their outputs are presented.

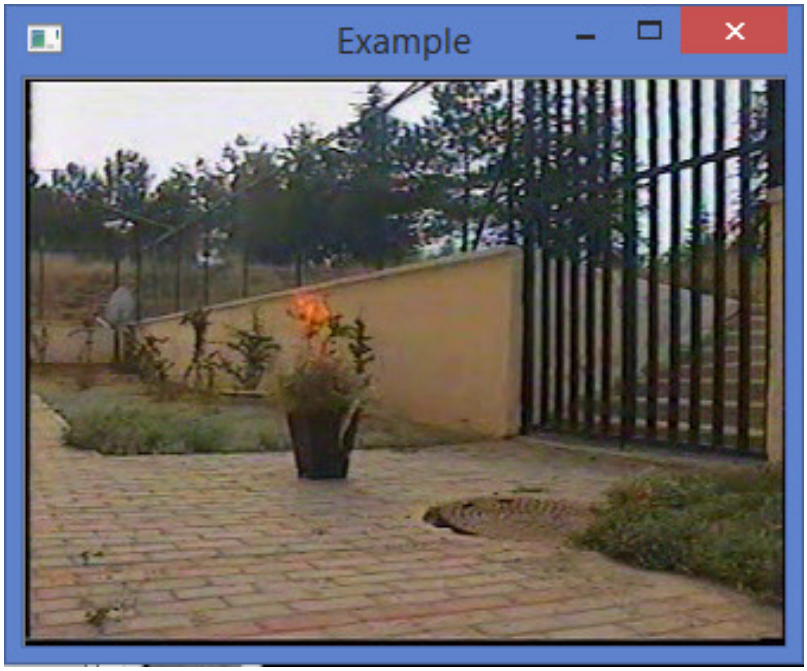

Figure 3.5: Example video frame

In the Figure 3.5, the example window shows the input video frame which is read from the given video by the CCTV video module and fed as an input to B/W conversion module. The B/W conversion module with the help of cvCvtColor( ) function, converts it to a grayscale frame as shown in Figure 3.6. The output from the B/W conversion is fed into the fire detection module. The fire dectection module has the individual sub-modules for frame and pixel analysis respectively. Figure 3.7 shows the frame after motion detection and clearly displays the diffrence in the frames.

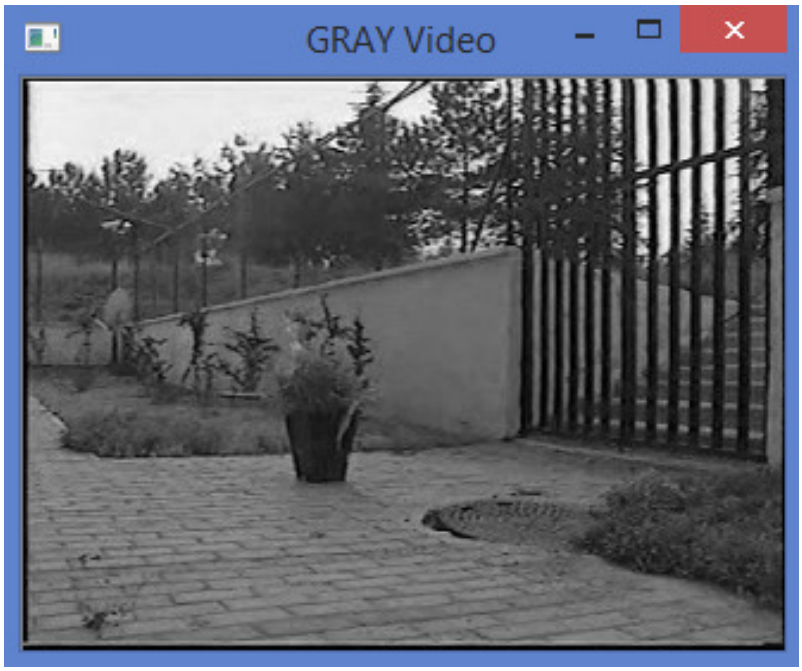

Figure 3.6: Gray video frame

Figure 3.8 shows the output from the contour analysis component. In the contour analysis, the contours moving pixels is calculated. Figure 3.9 shows the overall frame analysis output. This is the combined output from motion detection component and contour analysis component.

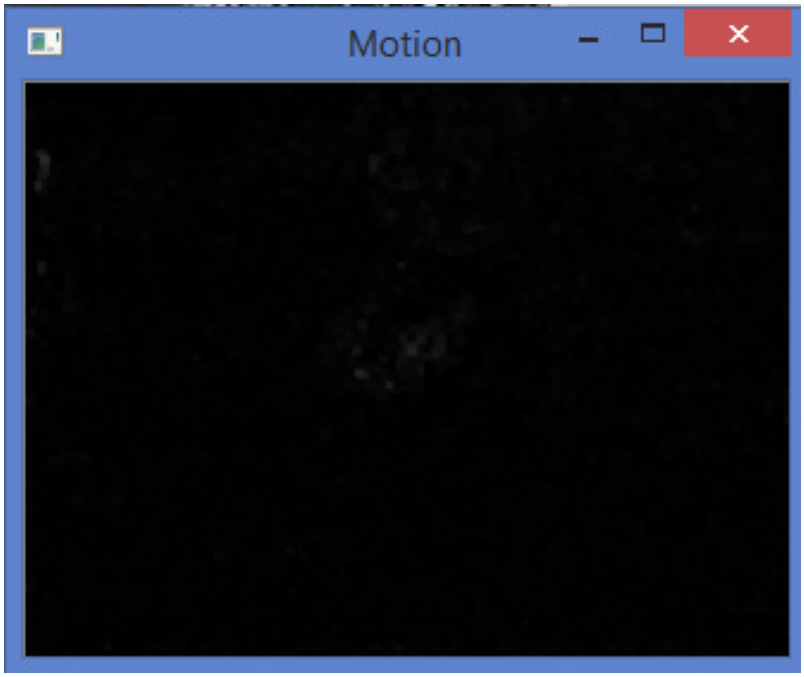

Figure 3.7: Motion detector output

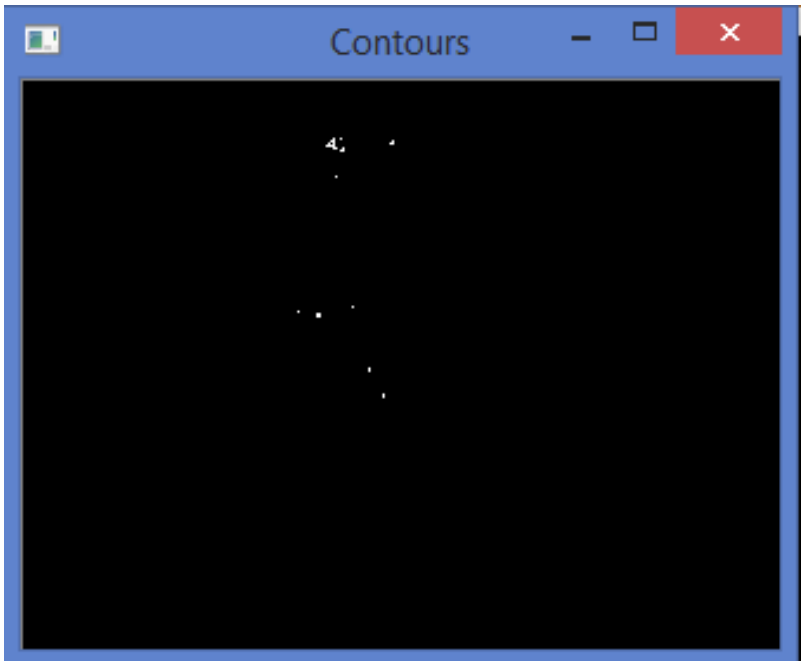

Figure 3.8: Contour analysis output

Figure 3.10 and Figure 3.11 are the outputs from the pixel analysis component. In Figure 3.10, the Canny edge detector output is shown. Canny edge detector helps in finding edges of the fractal like structures in a given video. Figure 3.11 shows the combined output from the individual modules of the pixel analysis component. The frame generated by the AND operation of the outputs shown in Figure 3.9 and Figure 3.11 are given to classifier module.

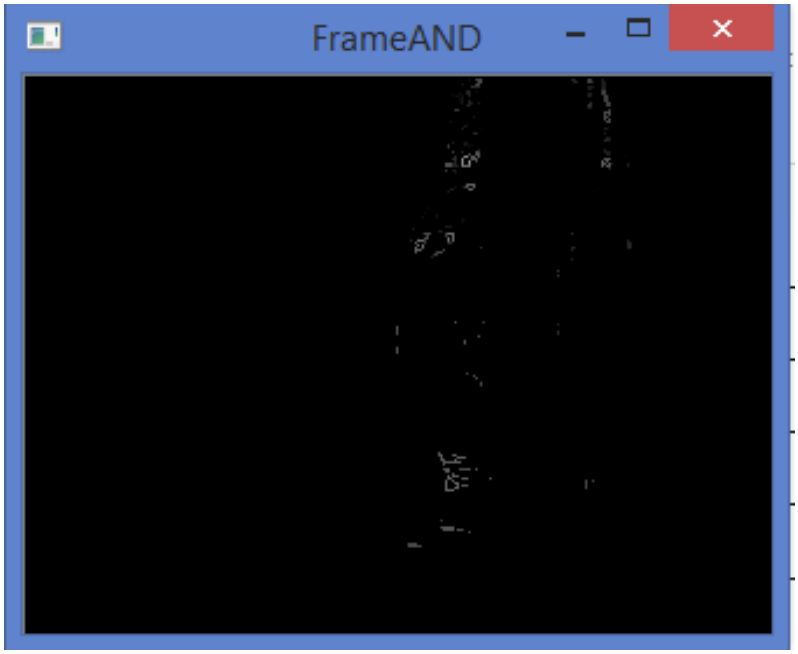

Figure 3.9: Overall Frame analysis output

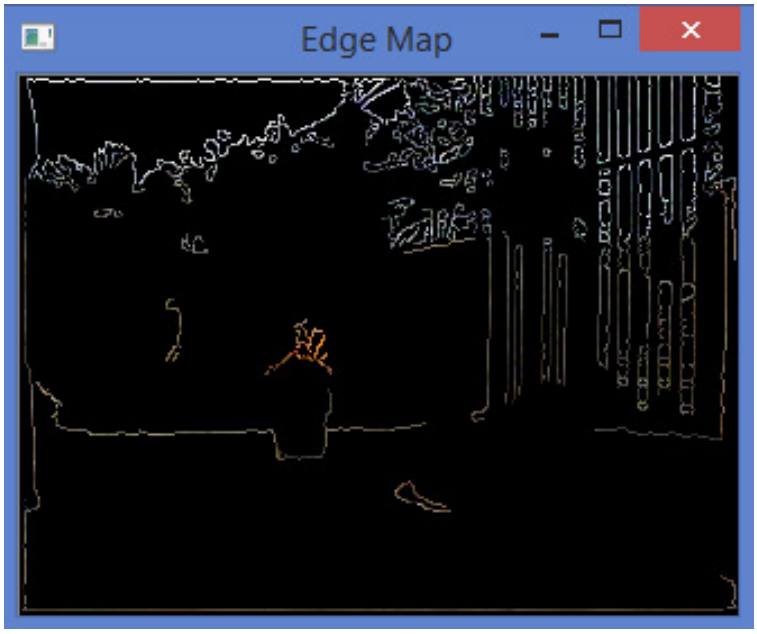

Figure 3.10: Canny edge detector output

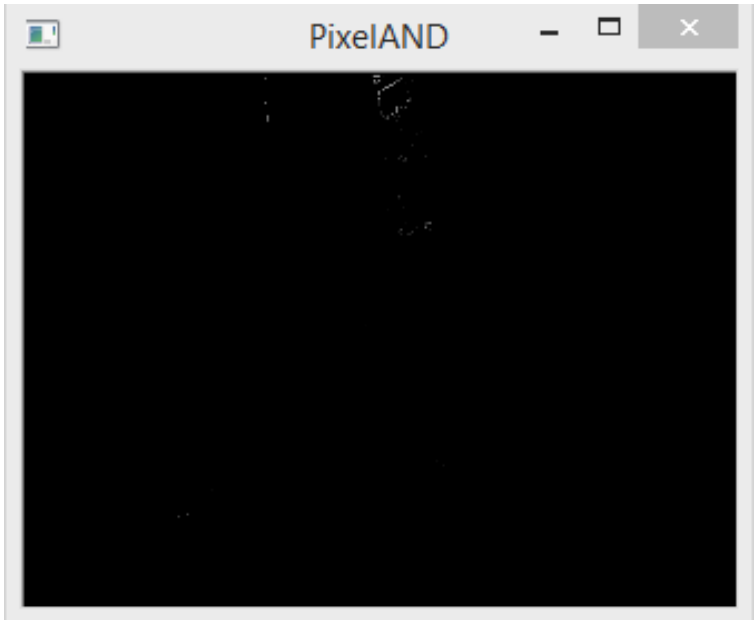

Figure 3.11: Overall pixel analysis output

4. Performance Evaluation

To demonstrate and evaluate the performance of the proposed system, A C++ based system application was built as described in Section III. This system is subjected to various test case videos obtained from the video dataset [15]. The dataset contains video sequences, which contain fire as well as non-fire scenes. For the performance evaluation, videos are categorized in fire and non-fire categories manually.

4.1 Ground Truth Evaluation

The video dataset is subjected to ground truth evaluation. During the ground truth evaluation, each frame in a given video sequence is checked for a presence of fire pixels. Open source VLC media player are used for obtaining each frame in the given video. Frames are obtained using the scene filter functionality of the VLC player. To be considered a fire video frame, the frame must contain at least a fire pixel. If a video has even one frame with a fire pixel it is considered as a fire video otherwise it is classified as a non-fire video. The categorization of the dataset is given in the table below:

Video #	01	02	03	04	05	06	07	08	09	10
Fire	X	X	X	X	X	X	X	X	X	X
Non-Fire										
Video #	11	12	13	14	15	16	17	18	19	11
Fire	X	X	X	X					X	X
Non-Fire					X	X	X	X		
Video #	30	31								
Fire										
Non-Fire	X	X								

Table 4.1: Dataset Categorization

4.2 Experiment Design

As described in Section III, the system has three basic threshold parameters on which the fire detection algorithm is based. Therefore, these are the parameters for the test cases of the experiments performed. Each video from the obtained dataset is considered as a separate test case.

4.2.1 Metrics

As in every recognition system, this system could also be measured in terms of a precision, recall and accuracy. The time taken to detect the first positive frame measured in milliseconds is a metric that provides the information about the time taken by the system to detect a positive frame. This is important for the system if implemented in real time.

4.2.2 Dynamic Parameters

The brightness threshold is a parameter which varies from values 0 to 255 for every case in the increments of 15. As proposed in Section III, this is a threshold used to detect brightness in the candidate pixels. Brightness determines the self-luminosity feature of fire, which is used as a parameter in the system. As the Canny and flicker thresholds also depend upon the brightness threshold, they are not considered as dynamic parameters in these experiments.

4.2.3 Static Parameters

The total number of frames in a given video is a static parameter. Twenty five frames make up a ‘group’. Generally, twenty five frames per second is the optimal frame rate; therefore, it is considered as a grouping factor for these experiments. Hence, for each test case a ‘group’ is considered rather than the number of frames retrieved to deduce the confusion matrix. Hence, the number of scenes is a static parameter for these experiments. The threshold value for Canny edge detector algorithm, helps detect the fractal like edges of the fire boundary making it another static parameter. As proposed in the earlier sections, fire

has a flickering property. Flickering threshold is a static parameter that controls the sensitivity to detect flickers in the given candidate pixels making it a static parameter.

The default value for both static flicker and Canny thresholds is set to the value of 15 in these experiments. The number of frames processed in every second, the resolution of each video frame and video length are the other three remaining static parameters.

4.3 Test Cases

The dataset contains a total of 31 videos. Each video is considered as a separate test case. The test results give the overall performance of the system with respect to the accuracy to detect fire in videos. Some of the test cases are presented below. Summarized results of the remaining test cases are presented in Appendix I.

4.3.1 Test Case 1

Test case 1 is performed on video Fire14.avi. This video contains some fire scenes and other moving objects like cars and a person who lit the fire on a road crossing. The results of test case are presented in Table 4.3. Figure 4.1 shows the graph of precision – recall and accuracy with respect to bright threshold. From Figure 4.1 performance of the system for this test case can be evaluated. The maximum accuracy of 65.71% was achieved at a threshold of 105, having precision at 59.2% and recall at 31.25%. On reaching the threshold value near 195, the system rejected the frames as the maximum gray value of the image was attained. The recall of 60% was recorded having precision at 42% and accuracy being at 60% at the threshold of 37. However, when threshold rises up to 63, the precision and recall are achieved at 50%. In Figure 4.1 some local maxima are observed in accuracy at threshold 15 and 45 whereas in precision at 45 and 105. Therefore conclusion can be inferred that threshold value of 45 is affecting the precision and accuracy greatly and could prove to be optimum threshold for this video test case. Also reaching at the threshold value of 195, the system rejects all the frames and takes it as true negative frame. Figure 4.2

shows the graph of time taken to detect first fire frame at each threshold for this test case. The maximum time taken to detect the first positive frame is 142480 milliseconds and minimum is 140 milliseconds. From the graph, it is observed that at the maximum value of a threshold of 195 is attained. If a maximum gray value for a given frame is reached the system starts rejecting frames which do not meet the threshold criteria. This causes an exponential rise in the time to detect the first positive frame as the maximum gray value is attained. Accuracy is stabilized at 60% when threshold of 150 is reached. The precision – recall show gradual decrease till the threshold value of 195 is attained. Figure 4.3 and 4.4 are the examples of true positive and false positive frames detected from this test case.

Static and Dynamic parameters

Static Parameter Name	**Static Value**
Test Video Name	Fire14.avi
Number of frames	5246
Number of scenes	210
Frames per second	21
Resolution	320 x 240
Canny Threshold	15
Flicker Threshold	15
Total Time (milliseconds)	365400
Dynamic Parameter Name	**Range**
Brightness Threshold	0 – 255

Table 4.2: Test Case 1 – Static and Dynamic Parameters

Bright Threshold	False Positives	True Positives	False Negatives	True Negatives	Precision	Recall	Accuracy	Time Taken to detect first positive frame in milliseconds
0	130	80	0	0	0.380952381	1	0.380952	140
15	70	49	31	60	0.411764706	0.6125	0.519048	190
30	62	46	34	68	0.425925926	0.575	0.542857	190
45	48	44	36	82	0.47826087	0.55	0.6	190
60	45	41	39	85	0.476744186	0.5125	0.6	190
75	37	36	44	93	0.493150685	0.45	0.614286	190
90	25	30	50	105	0.545454545	0.375	0.642857	190
105	17	25	55	113	0.595238095	0.3125	0.657143	290
120	12	17	63	118	0.586206897	0.2125	0.642857	430
135	11	13	67	119	0.541666667	0.1625	0.628571	620
150	10	6	74	120	0.375	0.075	0.6	1190
165	5	3	77	125	0.375	0.0375	0.609524	1240
180	4	2	78	126	0.333333333	0.025	0.609524	1330
195	2	0	78	128	0	0	0.619048	142480
210	0	0	80	130	0	0	0.619048	∞
225	0	0	80	130	0	0	0.619048	∞
240	0	0	80	130	0	0	0.619048	∞
255	0	0	80	130	0	0	0.619048	∞

Table 4.3: Test Case 1 Results

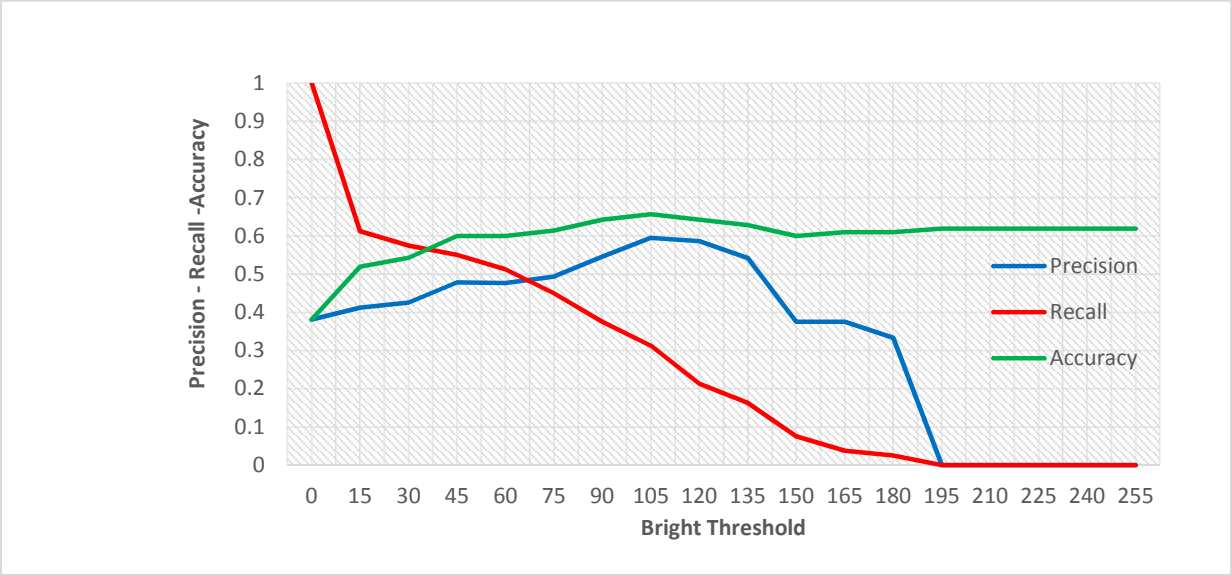

Figure 4.1: Test Case 1 – Precision – Recall – Accuracy graph

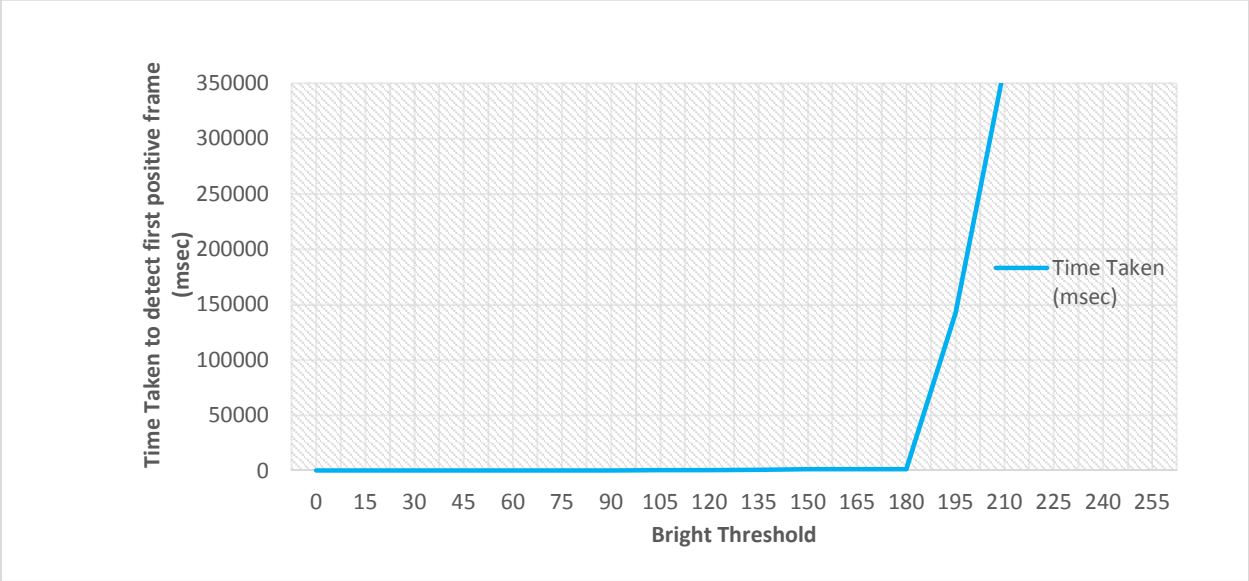

Figure 4.2: Test Case 1 – Detection time graph

Frame 4198 Frame 5205

Figure 4.3: Test Case 1 – True positive frames

Frame 3014 Frame 2051

Figure 4.4: Test Case 1 – False positive frames

4.3.2 Test Case 2

Test case 2 is performed on video Fire13.avi. This video is a capture of a room which has a trashcan on fire and a person walking by who happens to diffuse it. The results of test case are presented in Table 4.5. Figure 4.5 shows the graph of precision – recall and accuracy with respect to bright threshold. From Figure 4.5 performance of the system for this test case can be evaluated. The maximum accuracy of 85.07% was achieved at the threshold value of 15, having precision at 83.72% and recall at 92.30%. Thus, it can be concluded that threshold value 15 will prove as the optimum threshold for this video test case, which is very low as compared to 255. Also reaching at the threshold value 120, the system rejects all the frames and takes it as true negative frame. Figure 4.5 shows the graph of precision – recall and accuracy with respect to bright threshold. Figure 4.6 shows the graph of time taken to detect the positive frame from the given video at each threshold value. From the graph, it observed that at the maximum value of threshold of 120, it is taking maximum time to detect the first positive frame of 41720 milliseconds. Figure 4.7

and 4.8 are the examples of true positive and false positive frames detected from this test case. The minimum time taken to detect the first positive frame is 120 milliseconds. From the graph, it is observed that at the maximum value of a threshold of 120 is attained. If a maximum gray value for a given frame is reached the system starts rejecting frames which do not meet the threshold criteria. This causes an exponential rise in the time to detect the first positive frame as the maximum gray value is attained. Accuracy is stabilized at 40% when threshold of 120 is reached. The precision – recall show gradual decrease from 75 till the threshold value of 105 is attained. Figure 4.7 and 4.8 are the examples of true positive and false positive frames detected from this test case.

Static and Dynamic parameters

Static Parameter Name	**Static Value**
Test Video Name	Fire13.avi
Number of frames	1663
Number of scenes	67
Frames per second	25
Resolution	320 x 240
Video length (milliseconds)	63600
Canny Threshold	15
Flicker Threshold	15
Dynamic Parameter Name	**Range**
Brightness Threshold	0 – 255

Table 4.4: Test Case 2 – Static and Dynamic Parameters

Bright Threshold	False Positives	True Positives	False Negatives	True Negatives	Precision	Recall	Accuracy	Time Taken to detect first positive frame in milliseconds
0	28	39	0	0	0.582089552	1	0.58209	120
15	7	36	3	21	0.837209302	0.9230769	0.850746	320
30	7	34	5	21	0.829268293	0.8717949	0.820896	440
45	7	30	9	21	0.810810811	0.7692308	0.761194	440
60	7	29	10	21	0.805555556	0.7435897	0.746269	440
75	6	26	13	22	0.8125	0.6666667	0.716418	440
90	4	12	27	24	0.75	0.3076923	0.537313	2120
105	3	0	39	25	0	0	0.373134	41360
120	1	0	39	27	0	0	0.402985	41720
135	0	0	39	28	0	0	0.41791	∞
150	0	0	39	28	0	0	0.41791	∞
165	0	0	39	28	0	0	0.41791	∞
180	0	0	39	28	0	0	0.41791	∞
195	0	0	39	28	0	0	0.41791	∞
210	0	0	39	28	0	0	0.41791	∞
225	0	0	39	28	0	0	0.41791	∞
240	0	0	39	28	0	0	0.41791	∞
255	0	0	39	28	0	0	0.41791	∞

Table 4.5: Test Case 2 Results

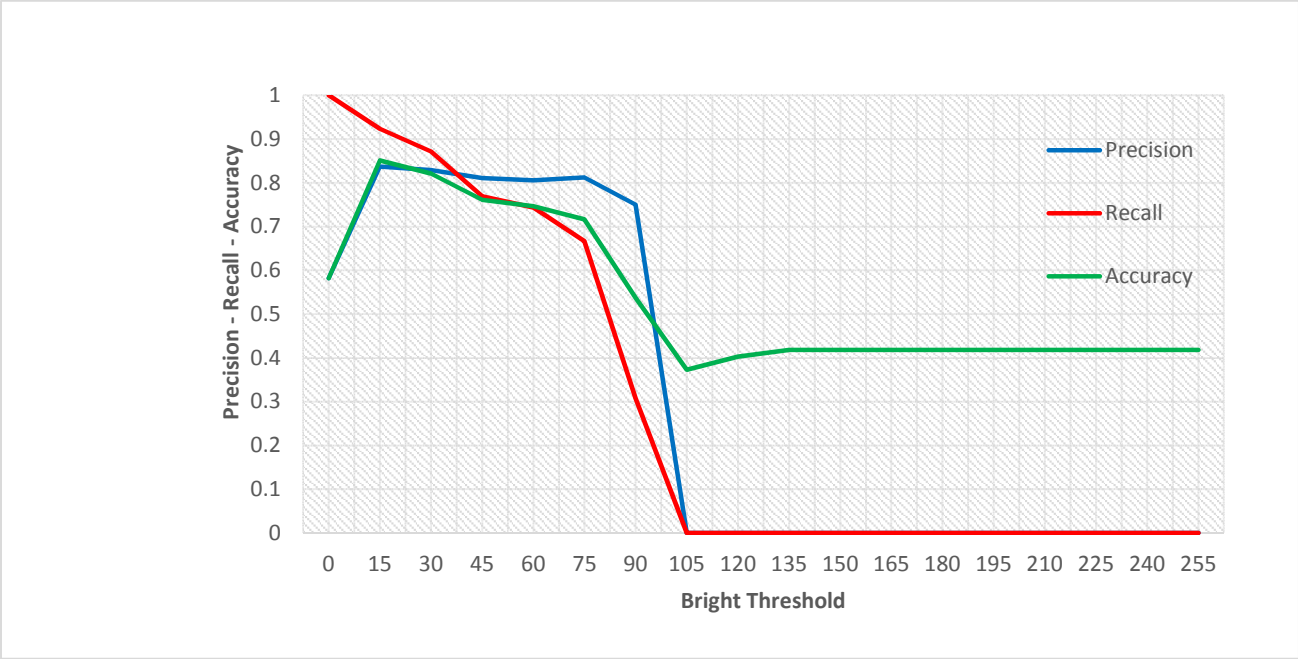

Figure 4.5: Test Case 2 – Precision – Recall – Accuracy graph

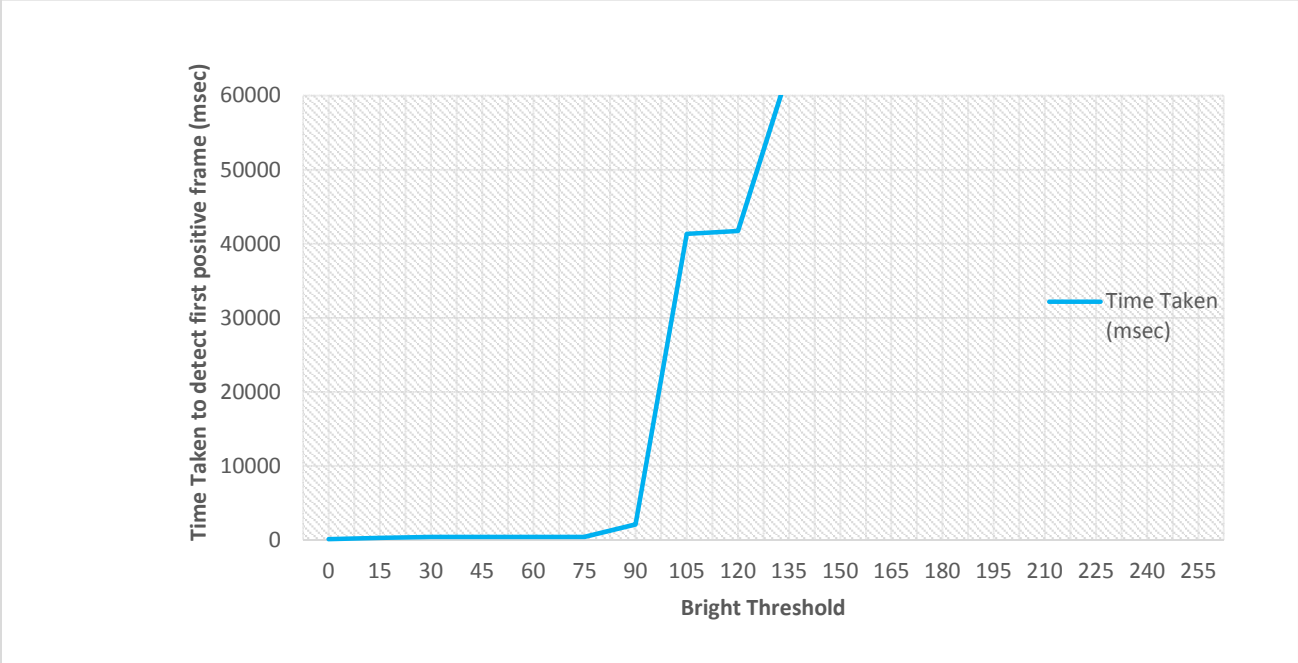

Figure 4.6: Test Case 1 – Detection time graph

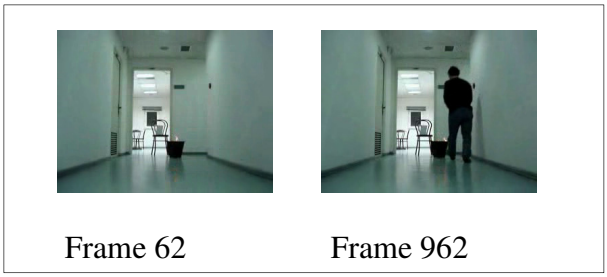

Figure 4.7: Test Case 2 – True positive frames

Figure 4.8: Test Case 2 – False positive frames

4.3.3 Test Case 3

Test case 3 is performed on video Fire31.avi. This video is a capture of a computer lab and has no fire frame present, but a person playing with a red ball and a red laptop cover to make fire like movements. The results of this test case are presented in the Table 4.7. From Figure 4.9 it is observed that the accuracy, precision and recall are increasing gradually. Thus, it can be concluded that with few false positives system was able to reject most of negative frames. Figure 4.10 shows the graph of time taken to detect the positive frame from the given video at each threshold value. From the graph, it is observed that at the maximum value of threshold of 105, it is taking maximum time to detect the first positive frame of 70210 milliseconds. Figure 4.11 shows the examples of false positive frames detected from this test case.

Static and Dynamic parameters

Static Parameter Name	**Static Value**
Test Video Name	Fire31.avi
Number of frames	1496
Number of scenes	60
Frames per second	14
Video length (milliseconds)	84000
Resolution	800 x 600
Canny Threshold	15
Flicker Threshold	15
Dynamic Parameter Name	**Range**
Brightness Threshold	0 – 255

Table 4.6: Test Case 3 – Static and Dynamic Parameters

Bright Threshold	False Positives	True Positives	False Negatives	True Negatives	Precision	Recall	Accuracy	Time Taken to detect first positive frame in milliseconds
0	60	0	0	0	0	0	0	200
15	19	0	0	41	0	0	0.683333	51070
30	19	0	0	41	0	0	0.683333	51140
45	15	0	0	45	0	0	0.75	51140
60	14	0	0	46	0	0	0.766667	51140
75	6	0	0	54	0	0	0.9	59940
90	4	0	0	56	0	0	0.933333	59940
105	1	0	0	59	0	0	0.983333	70210
120	0	0	0	60	0	0	1	∞
135	0	0	0	60	0	0	1	∞
150	0	0	0	60	0	0	1	∞
165	0	0	0	60	0	0	1	∞
180	0	0	0	60	0	0	1	∞
195	0	0	0	60	0	0	1	∞
210	0	0	0	60	0	0	1	∞
225	0	0	0	60	0	0	1	∞
240	0	0	0	60	0	0	1	∞
255	0	0	0	60	0	0	1	∞

Table 4.7: Test Case 3 Results

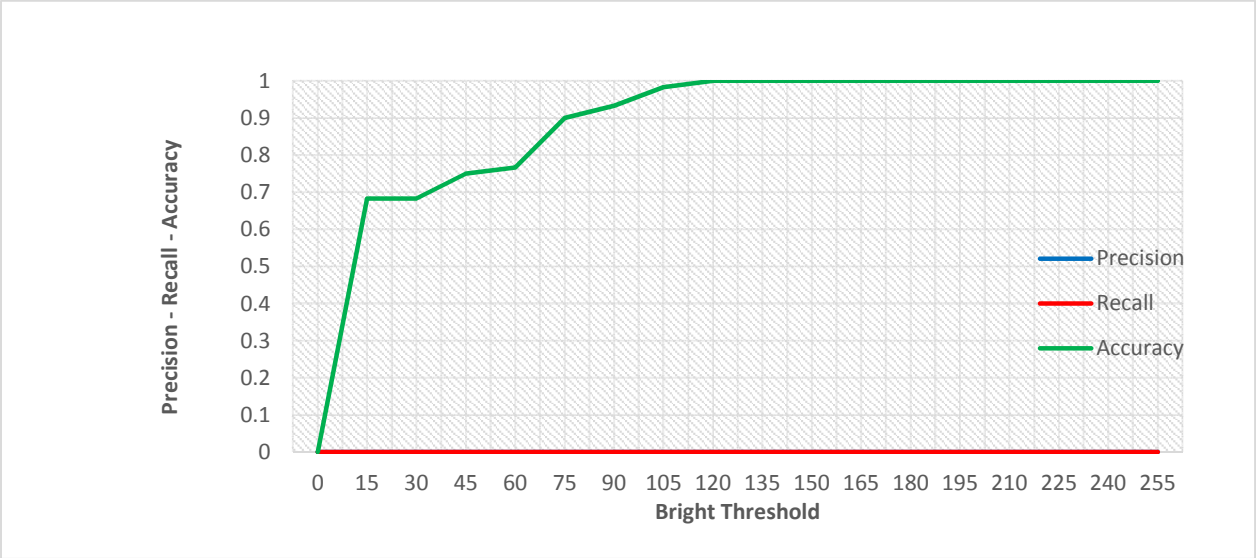

Figure 4.9: Test Case 3 – Precision – Recall – Accuracy graph

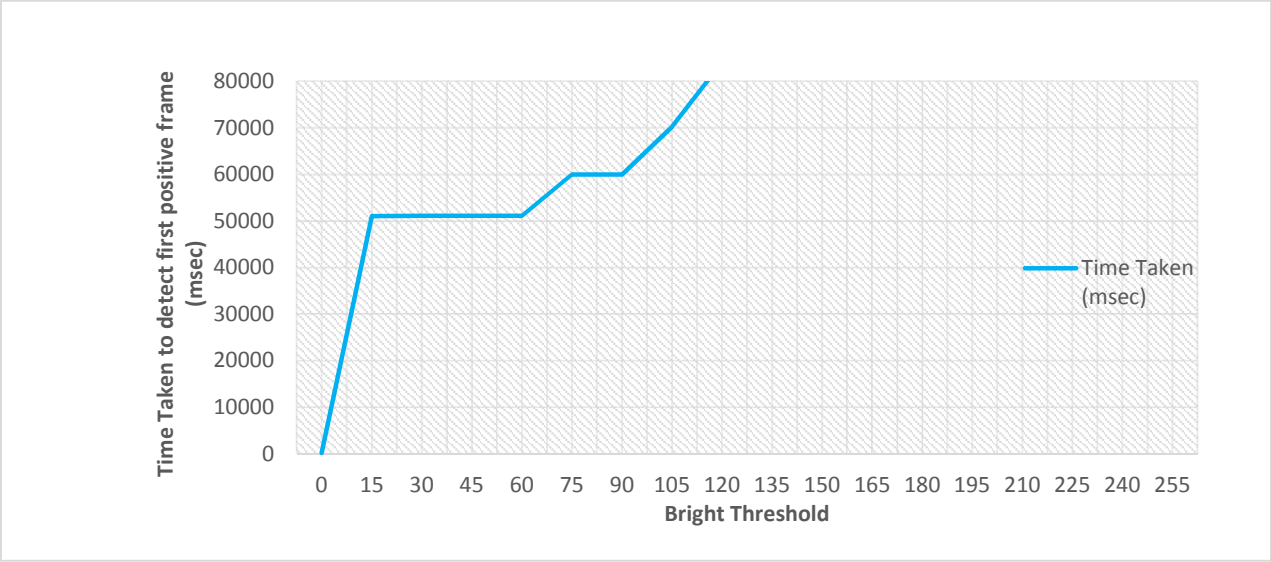

Figure 4.10: Test Case 3 – Detection time graph

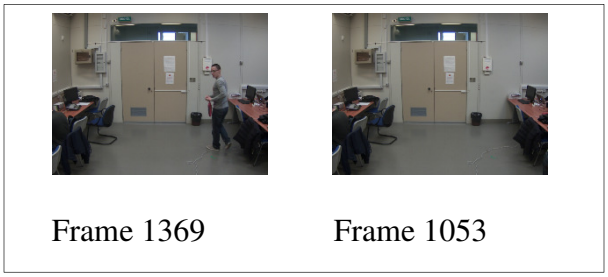

Figure 4.11: Test Case 3 – False positive frames

5. Conclusion and Future Work

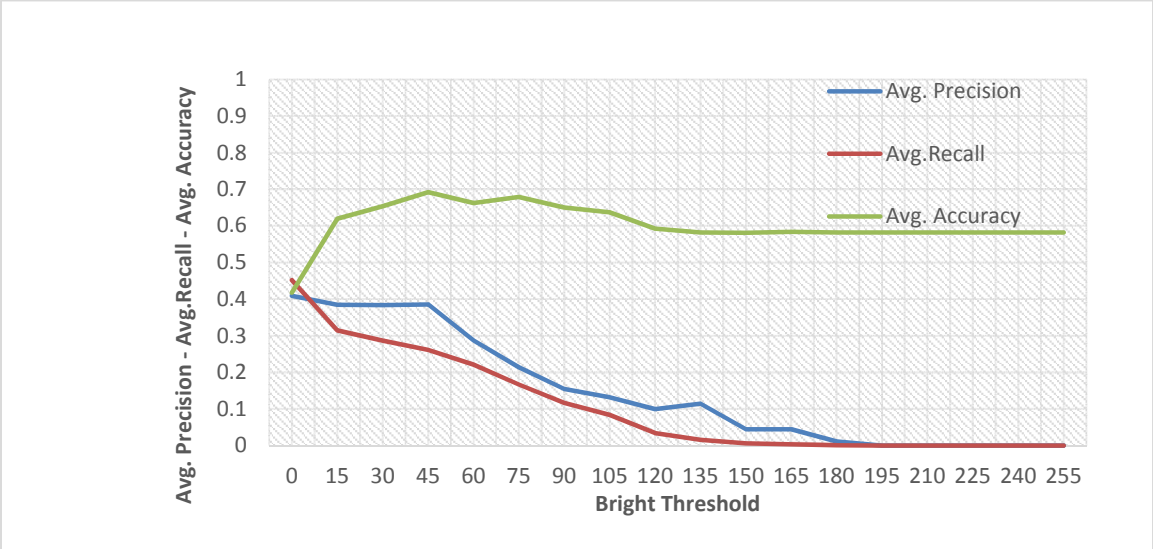

Figure 5.1: Average Precision – Recall – Accuracy graph

Figure 5.1 helps to evaluate overall performance of the system. From the Figure 5.1 it can be observed that precision and recall are having downward slope with increase in the threshold value. Whereas accuracy seems to stabilize at 60% after the threshold value of 165. This means that the bright threshold is not having much effect on accuracy of the system. However, precision and recall are greatly affected.

To implement the system in real time the precision and recall should be improved. The threshold must traverse through entire range to improve the precision and recall of the system. Histogram equalization could be the solution to achieve improvement in the performance of the system.

The future research work for this system could be in the improvement of precision and recall. The solution of histogram equalization makes algorithm more complex, as histogram of every frame needs to be equalized. Therefore, this becomes a non-feasible solution to improve precision and recall of the system. Even there could be possibility of parallel processing algorithms to achieve improvement of the system which needs further research in the field.

References

[1] Y. Do, “Flame detection in grey-scale images of a B/W camera”, in *Sensor Review*, vol. 34, no. 1, pp. 80–88, 2014.

[2] H. Yamagishi and J. Yamaguchi, “A contour fluctuation data processing method for fire flame detection using a color camera”, in *Industrial Electronics Society 26th Annual Confjerence of the IEEE*, IEEE, 2000, vol. 2, pp. 824–829.

[3] L. Hongliang, L. Qing, and W. Sun’an, “A novel fire recognition algorithm based on flame’s Multi-features Fusion,” in *International Conference on Computer Communication and Informatics (ICCCI)*, 2012, pp. 1–6.

[4] F. Yuan, “An integrated fire detection and suppression system based on widely available video surveillance”, in *Machine Vision and Applications*, 2010, vol. 21, no. 6, pp. 941–948.

[5] X. Zhou, F. Yu, Y. Wen, Z. Lu, and G. Song, “Early fire detection based on flame contours in video”, in Information Technology Journal, 2010, vol. 9, no. 5, pp. 899–908.

[6] A. Çetin, K. Dimitropoulos, B. Gouverneur, N. Grammalidis, O. Günay, Y. Habiboglu, B. Töreyin, and S. Verstockt, “Video fire detection–Review”, in *Digital Signal Processing*, 2013, vol. 23, no. 6, pp. 1827–1843.

[7] R. Bohush and N. Brouka, “Smoke and flame detection in video sequences based on static and dynamic features,” in *Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, 2013, pp. 20–25.

[8] T. Song, B. Tang, M. Zhao, and L. Deng, “An accurate 3-D fire location method based on sub-pixel edge detection and non-parametric stereo matching”, in *Measurement*, 2014.

[9] T. Li, M. Ye, F. Pang, H. Wang, and J. Ding, “An efficient fire detection method based on orientation feature” in *International Journal of Control, Automation and Systems*, 2013, vol. 11, no. 5, pp. 1038–1045.

[10] L. Mengxin, X. Weijing, Z. Ying, and Z. Rui, “A new fire detection method based on integrated features,” in *24th Chinese Control and Decision Conference (CCDC)*, 2012, pp. 965–970.

[11] M. Mueller, P. Karasev, I. Kolesov, and A. Tannenbaum, “Optical Flow Estimation for Flame Detection in Videos”, in *IEEE Transactions on Image Processing*, 2013, vol. 22, no. 7, pp 2786 – 2797.

[12] P. Borges and E. Izquierdo, “A probabilistic approach for vision-based fire detection in videos”, *in IEEE Transactions on Circuits and Systems for Video Technology*,IEEE, 2010 , vol. 20, no. 5, pp. 721–731.

[13] G. Marbach, M. Loepfe, and T. Brupbacher, “An image processing technique for fire detection in video images” in *Fire safety journal*, 2006, vol. 41, no. 4, pp. 285–289.

[14] B. Ko, K. Cheong, and J. Nam, “Fire detection based on vision sensor and support vector machines” *in Fire Safety Journal*, 2009, vol.44, no. 3, pp. 322–329.

[15] R. Lascio, A. Greco, A. Saggese and M. Vento, “Improving fire detection reliability by a combination of videoanalytics”, *International Conference on Image Analysis and Recognition (ICIAR)*, 2014.

[16] J. Canny, “A Computational Approach to Edge Detection”, in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, 1986, vol. 8, no. 6, pp. 679-698.

[17] The OpenCV Reference Manual, Revision 2.4.9.0, *OpenCV Community*, April 2014.

Appendices

Appendix I: Research Results

Performance data was collected while testing this system to track the effects of the dynamic parameter of brightness threshold. This appendix section documents a subset of data recorded where the accuracy was found maximum for each test case.

Test Case#	Video File	Max accuracy	Precision	Recall
1	Fire14.avi	0.65	0.59	0.32
2	Fire13.avi	0.85	0.82	0.92
3	Fire31.avi	0.90	0.00	0.00
4	Fire1.avi	0.74	1.00	0.74
5	Fire2.avi	1.00	1.00	1.00
6	Fire3.avi	1.00	1.00	1.00
7	Fire4.avi	1.00	1.00	1.00
8	Fire5.avi	1.00	1.00	1.00
9	Fire6.avi	0.98	1.00	1.00
10	Fire7.avi	1.00	1.00	1.00
11	Fire8.avi	1.00	1.00	1.00
12	Fire9.avi	1.00	0.72	1.00
13	Fire10.avi	1.00	1.00	1.00
14	Fire11.avi	1.00	1.00	1.00
15	Fire12.avi	1.00	1.00	1.00
16	Fire15.avi	1.00	0.00	0.00
17	Fire16.avi	0.89	0.00	0.00
18	Fire17.avi	0.98	0.00	0.00
19	Fire18.avi	0.68	0.00	0.00
20	Fire19.avi	0.96	0.00	0.00
21	Fire20.avi	0.99	0.00	0.00
22	Fire21.avi	0.25	0.00	0.00

Appendix I (Continued)

23	Fire22.avi	0.99	0.00	0.00
24	Fire23.avi	0.99	0.00	0.00
25	Fire24.avi	0.94	0.00	0.00
26	Fire25.avi	0.00	0.00	0.00
27	Fire26.avi	0.00	0.00	0.00
28	Fire27.avi	0.93	0.00	0.00
29	Fire28.avi	0.99	0.00	0.00
30	Fire29.avi	0.95	0.00	0.00
31	Fire30.avi	0.94	0.00	0.00

Appendix II: Working Code

This appendix section documents the working code of the system.

```
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv\cv.h"
#include <iostream>
#include <windows.h>
#pragma comment (lib , "winmm.lib")

using namespace cv;
using namespace std;

// Define global matrices for cannythreshold module
Mat src, src_gray;
Mat dst, detected_edges;
char* window_name = "Edge Map";
int edgeThresh = 1;
int lowThreshold=100;
int const max_lowThreshold = 100;
int ratio = 3;
int kernel_size = 3;
int i=0;
// Canny Threshold function call
void CannyThreshold(int, void*)
{
  /// Reduce noise with a kernel 3x3
  blur( src_gray, detected_edges, Size(3,3) );

  /// Canny detector
  Canny( detected_edges, detected_edges, lowThreshold, lowThreshold*ratio,
kernel_size );

  /// Using Canny's output as a mask, we display our result
  dst = Scalar::all(0);

  src.copyTo( dst, detected_edges);
  //imshow( window_name, dst );
 }

int main( int argc, char** argv )
{
       double mean;
       int flicker_threshold = 255;
       int canny_threshold   = 255;
       int bright_threshold  = 180;

       if(argc < 1)
       {
         printf( "file not found \n usage: %s filename threshold", argv[0] );
         exit(1);
         //getchar();
       }
```

Appendix II (Continued)

```
if(argc < 2)
       {
      printf( "threshold missing \n usage: %s filename threshold", argv[0] );
         exit(1);
         //getchar();
       }
       //change dynamically
       flicker_threshold = atoi(argv[2]);
       //canny_threshold = atoi(argv[2]);
       //bright_threshold = atoi(argv[2]);
       //flicker_threshold=canny_threshold;
       //bright_threshold=flicker_threshold;

       CvCapture* capture = cvCaptureFromAVI(argv[1]);

       printf("\nflicker_threshold=%d",flicker_threshold);
       printf("\ncanny_threshold=%d",canny_threshold);
       printf("\nbright_threshold=%d",bright_threshold);

       //getchar();
       //CvCapture* capture = cvCaptureFromCAM(0);
       //cvSetCaptureProperty( capture, CV_CAP_PROP_FRAME_WIDTH, 320 );
       //cvSetCaptureProperty( capture, CV_CAP_PROP_FRAME_HEIGHT, 240 );
       //int value=12;
       int nFrames = (int) cvGetCaptureProperty( capture , CV_CAP_PROP_FRAME_COUNT
);
       printf("\n#Frames=%d",nFrames);
       int fps = (int) cvGetCaptureProperty( capture , CV_CAP_PROP_FPS );
       printf("\nFPS=%d",fps);
       //printf("\nOLDFPS=%d",fps);
       //printf("\nNew FPS is set to %d",value);
       //cvSetCaptureProperty( capture, CV_CAP_PROP_FPS, value);

       IplImage
*frame,*image=0,*grayimg1=0,*frameTime1=0,*frameTime2=0,*frameForeground=0,*edgeMa
p=0,*FrameAND=0,*PixelAND=0,*AND=0;
       while(1)
       {
       IplImage*     g_image = NULL;
       IplImage*     g_gray = NULL;
       int           g_thresh = 100;
       CvMemStorage*        g_storage = NULL;
       frame = 0;//every time create/initialize an image (which name is frame) to
process
        //int  c;                //integer to exit program
        frame = cvQueryFrame( capture );   //grabs and returns a frame from a
camera input


             if( !frame )
             {
                    //if there is no frame exit the while(1)
```

Appendix II (Continued)

```
printf("\n NO INPUT !!!");
                    //getchar();
                    break;
             }

             if( !image )    //if there is no image, do the following
        {
            /* allocate all the buffers */
            image = cvCreateImage( cvGetSize(frame), 8, 3 );
            frameTime1 = cvCreateImage( cvGetSize(frame), 8, 1 );
            frameTime2 = cvCreateImage( cvGetSize(frame), 8, 1 );
            frameForeground = cvCreateImage( cvGetSize(frame), 8, 1 );
            grayimg1 = cvCreateImage( cvGetSize(frame), 8, 1 );
            FrameAND = cvCreateImage( cvGetSize(frame), 8, 1 );
                    PixelAND = cvCreateImage( cvGetSize(frame), 8, 1 );
                    edgeMap = cvCreateImage( cvGetSize(frame), 8, 1 );
                    AND = cvCreateImage( cvGetSize(frame), 8, 1 );
        }

             int frameno = (int) cvGetCaptureProperty( capture ,
CV_CAP_PROP_POS_FRAMES );
          //printf("\nFrame#=%d",frameno);
          float ntimeSec = (float) cvGetCaptureProperty( capture ,
CV_CAP_PROP_POS_MSEC )/1000;
          //printf("\n#FrameSec=%d",ntimeSec);

             cvNamedWindow("Example",CV_WINDOW_AUTOSIZE);
             cvMoveWindow("Example",0,0);
             cvShowImage( "Example", frame );
             // B/W Conversion
             cvCopy( frame, image, 0 );
        cvCvtColor( image, grayimg1, CV_RGB2GRAY );
             cvNamedWindow("GRAY Video",CV_WINDOW_AUTOSIZE);
             cvMoveWindow("GRAY Video",50,50);
             cvShowImage( "GRAY Video", grayimg1 );
             // Frame Analysis
        cvCopy( grayimg1, frameTime1, 0 );    //currently frame in grayscale
        cvAbsDiff(
                  frameTime1,
                  frameTime2,
                  frameForeground
                  );

      if( g_storage == NULL )
      {
             g_gray = cvCreateImage( cvGetSize(frame), 8, 1 );
             cvCopy(frameForeground,g_gray);
             g_storage = cvCreateMemStorage(0);
      }
      else
      {
             cvClearMemStorage( g_storage );
      }
```

Appendix II (Continued)

```
	CvSeq* contours = 0;
	cvThreshold( g_gray, g_gray, g_thresh,flicker_threshold, CV_THRESH_BINARY
);
	//cvThreshold( g_gray, g_gray, g_thresh, 255, CV_THRESH_BINARY );
	cvFindContours( g_gray, g_storage, &contours );
	cvZero( g_gray );
	if( contours )
	{
		cvDrawContours(
			g_gray,
			contours,
			cvScalarAll(255),
			cvScalarAll(255),
			100 );

	}

	cvNamedWindow("Contours",CV_WINDOW_AUTOSIZE);
	cvMoveWindow("Contours",150,150);
	cvShowImage( "Contours", g_gray );

	 cvErode(
			frameForeground,
			frameForeground,
			0,
			1);
	 cvDilate(
			frameForeground,
			frameForeground,
			0,
			1);
	 cvDilate(
			frameForeground,
			frameForeground,
			0,
			1);
	 cvErode(
			frameForeground,
			frameForeground,
			0,
			1);

	 cvNamedWindow("Motion",CV_WINDOW_AUTOSIZE);
		cvMoveWindow("Motion",100,100);
	 cvShowImage( "Motion", frameForeground );

		//	cvCopy( frametime1, frametime2, 0 ); // now get the next frame
for frame by frame analysis
	cvCopy( frameTime1, frameTime2, 0 );
		cvAnd(frameForeground,g_gray,FrameAND,0);
		cvShowImage( "FrameAND", FrameAND);
```

```
        //Get edges from the video using CannyThreshold
        // allocate buffer for frametime1 , 2 and frameForeground

              // Pixel by pixel analysis
        src = frame;

          /// Create a matrix of the same type and size as src (for dst)
              dst.create( src.size(), src.type() );

              cvtColor( src, src_gray, CV_BGR2GRAY );

              //createTrackbar( "Min Threshold:", window_name, &lowThreshold, ,
CannyThreshold );
              //Pixel Analysis
              CannyThreshold(canny_threshold, 0);
              IplImage *cannyimg = cvCloneImage(&(IplImage)dst);
              cvCvtColor(cannyimg,edgeMap,CV_BGR2GRAY);
              cvAnd(FrameAND,edgeMap,PixelAND,0);
              cvNamedWindow(window_name,CV_WINDOW_AUTOSIZE);
              cvMoveWindow(window_name,200,200);
              cvShowImage( window_name, cannyimg);
              cvShowImage("PixelAND",PixelAND);

              // Classifier Component
              // And all the final output images from individual modules

              cvCmpS(FrameAND,bright_threshold,FrameAND,CV_CMP_GE);
              cvThreshold(FrameAND,FrameAND,1,255, CV_THRESH_BINARY);
              cvShowImage("Frame255",FrameAND);
              cvCmpS(PixelAND,bright_threshold,PixelAND,CV_CMP_GE);
              cvThreshold(PixelAND,PixelAND,1,255, CV_THRESH_BINARY);
              cvShowImage("Pixel255",PixelAND);
              cvAnd(FrameAND,PixelAND,AND);
              cvThreshold(AND,AND,1,255, CV_THRESH_BINARY);


              if(AND)
              {
                     mean=0;
                     mean = cvMean(AND,0);
                     //mean = mean / 255;
                     printf("\n %d",mean);

                     if(mean > 0)
                     {
                            i++;
                            if(i>2)
                            {
                                   printf("\nFIRE!!!!");
                                   printf("\nFrame#=%d",frameno);
                                   printf("\n#FrameSec=%0.2f",ntimeSec);
                                   //cvShowImage("FIRE",AND);
                                   //PlaySound(TEXT("alarm.wav"), NULL,
SND_FILENAME);
```

Appendix II (Continued)

```
                                }
                        }


                }


        // check for ESC character
        char c = cvWaitKey(33);
        if(c == 27)
        break;
        }

        cvReleaseCapture(&capture);
        cvDestroyWindow("ExampleFromCamera");
        exit(0);
}
```