FIRE AND SMOKE DETECTION BASED ON LOW-COST CAMERA

NG CHING HAU

A project progress report submitted in partial fulfilment of the requirements for the award of the degree of Bachelor (Hons.) of Mechatronics Engineering

Faculty of Engineering and Science

Universiti Tunku Abdul Rahman

May 2010

DECLARATION

Signature	:	______________________
Name	:	______________________
ID No.	:	______________________
Date	:	______________________

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"FIRE AND SMOKE DETECTION BASED ON LOW-COST CAMERA"** was prepared by **NG CHING HAU** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons.) Mechatronics Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature	:	__________________________
Supervisor	:	Dr. Tay Yong Haur
Date	:	__________________________

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Dr. Tay Yong Haur for his invaluable advice, guidance and patience throughout the development of the project.

In addition, I would also like to express my gratitude to my loving parents and friends who had helped and given me encouragement and also motivated me to do more than was necessary.

I would also like to thank Ho Wing Teng and Lee Chee Wei of Recogine Technologies, as they guided me through the methods and process of OpenCV, which was used in this project. The project would not have worked so well if not for their help in showing me the ropes to OpenCV.

FIRE AND SMOKE DETECTION BASED ON LOW-COST CAMERA

ABSTRACT

Fire is a disaster that can strike anywhere and can be very destructive. A method to detect smoke and fires would allow the authorities to detect and put out the fires before it becomes out of control. One of the cost effective methods would be to use cameras which are already on the roads to detect the fires early in order to inform the relevant parties. This project suggests a method to use surveillance cameras in order to monitor occurrences of fire anywhere within camera range. Since cameras are already installed in most places, this method would be cost effective as there would be no need to purchase and install the hardware required. The response can also be quicker due to information being gathered early. An algorithm that can process videos in order to detect smoke is developed in this project. Using properties of smoke, such as its colour and seemingly random motion, the video is processed, and elements which match the properties are determined to be smoke. The results of this method show that visible smoke can be reliably detected in relatively bright or well lighted areas, such as indoors and on highway roads. Although the performance of the algorithm degrades on darker areas, as smoke is not so visible and difficult to trace, this can be overcome by coupling it with another fire detection algorithm.

TABLE OF CONTENTS

DECLARATION **ii**
APPROVAL FOR SUBMISSION **iii**
ACKNOWLEDGEMENTS **v**
ABSTRACT **vi**
TABLE OF CONTENTS **vii**
LIST OF TABLES **viii**
LIST OF FIGURES **ix**
LIST OF APPENDICES **x**

INTRODUCTION **11**
1.1 Rationale....................11
1.2 Objective....................14
1.3 Scope....................15
LITERATURE REVIEW **16**
2.1 Industry Application....................16
2.2 Technical Analysis....................18
METHODOLOGY **22**
3.1 Research Methods....................22
3.2 Method of Solution....................24
3.3 Method Used....................26
RESULTS AND ANALYSIS **31**
4.1 Experiment....................31
4.2 Results....................32
4.3 Analysis....................34
4.4 Discussion....................35
CONCLUSION **37**
5.1 Evaluation of Project against its Objectives....................37
5.2 Strengths and Weaknesses....................38
5.3 Recommendations....................39
5.4Conclusion....................40

LIST OF TABLES

TABLE	TITLE	PAGE
Table 4.2.1	Accuracy of system for Video	32

LIST OF FIGURES

FIGURE	TITLE	PAGE
Figure 3.3.1	System Flowchart	26
Figure 3.3.2	Subtracted image	27
Figure 3.3.3	Grayscaled Image	28
Figure 3.3.4	Thresholded Image	28
Figure 5.3.4	Contour drawn	29

LIST OF APPENDICES

APPENDIX	TITLE	PAGE
A	Code	
B	Experimental Results	

CHAPTER 1

INTRODUCTION

1.1 Rationale

1.1.1 Hazards of Fire

Fire is one of the few disasters in which damage can be prevented or reduced, when compared to other natural disasters such as earthquakes and hurricanes. However, in order to control the fire, it is important to detect it early so that it can be put out before it grows too large to be put out easily, since a small fire can be put out with a lot less effort than a big one.

Fire is one of the worst hazards. Being too close to a fire can cause burns, while the smoke that is emitted by the fires may cause asphyxiation. Fire destroys property like other natural disasters. Certain things which are set on fire can release poisonous gases, while a large amount of combustible material may cause a huge explosion if set on fire.

Some common fire hazards include overheating electrical appliances, candles, smoking, cooking appliances and heating appliances. These places should be monitored for fire more often as they have a tendency to cause fire, and are common locations where fires occur.

Among such places is the outdoors, where people may have a barbeque party or smoke. As these are activities that can lead to fires, they should be done responsibly and be put out afterwards.

An example of a large fire that occurred due to carelessness is the Yellowstone fires of 1988. Starting as small individual fires, the effects of wind and drought brought them together into one large fire that burned on for several months. The fire destroyed two tourist attractions, and 3213 km^2 of the park, which is about 36% of the area of the park, was destroyed, despite the efforts of thousands of firefighters. The situation was so bad that military personnel were brought in to control the spread of the fire. The fire cost 120 million dollars just to fight it, not including the damage caused by the fire.

1.1.2 Fire Detection and Control

For the purpose of reducing the damage of fire, as mentioned above, the fire should be put out while small. However, fire naturally grows larger as long as there is fuel. It is therefore important to put out the fire before it grows larger. However, small fires are more difficult to see, and therefore many fires are only spotted too late. This can be remedied by the use of fire and smoke detectors, which can detect fire using which is emitted from the fire as well as the fire itself.

These fire detection systems can be placed where fire hazards are, in order to respond quickly in case of any untoward accident.

Currently there are a number of different smoke detectors already in the market. The main ones are: optical smoke detectors, where a light travels from one point to another and is dispersed by smoke; ionization smoke detectors, where smoke particles prevent the current from flowing inside a circuit; and air sampling smoke detectors, where the air is sampled in a given time period in order to detect trace amounts of smoke by a large system.

It can be noticed that these detectors are quite expensive to set up and is usually only for closed spaces. Another problem with them is that the response time is slow, as it requires the smoke and heat to dissipate.

As the examples of fire hazards include the outdoors, where smoke can go undetected because the particle smoke detectors are unable to detect smoke in such a large area due to scattering of particles, it is not useful for these situations.

The response time is also important, as can be seen by the example of the Yellowstone fires. The incident would not have been so severe if the fires were suppressed early before they converged.

Therefore, this project aims to come up with a detection system that can be used anywhere, especially open spaces, as well as can enable a quick response to a fire.

1.2 Objective

The objective of this project is to create a system which would be able to detect fire and smoke using images from a feed, such as a video feed. The system should be made in order to be able to detect fires while they are still small and have not grown too large.

The system should have a reasonable response time. There is no point to having detected that a fire has occurred only after the fire has grown too large, as response time is needed in order to react to the fire.

The system must not detect smoke and fire where there is none. This means that if there is no fire, another object or objects that look like fire or smoke should not be considered as fire or smoke, since this would mean that the system is not working correctly and giving false responses.

1.3 Scope

The fire and smoke detection covers a very large field, and it would be impossible to cover all aspects in one project. As the title stipulates, the focus in this project is detection using a low cost camera.

A low cost camera is easily available in the market. This would mean that the program does not only work with expensive technology such as infrared cameras or other such cameras. The cameras that are required to at least work with this program is the CCTV camera, such as those in shopping complexes or malls.

One factor that needs to be taken into consideration is that, unlike other smoke detectors, this system is not a point type detector. It should be able to detect fire or smoke in large open spaces, so the whole scenario must be considered, and not just a single point on the image from the video feed.

With this, the scope has been narrowed down to a manageable level for this project. The intention and aim for this project is now very clear.

CHAPTER 2

LITERATURE REVIEW

2.1 Industry Application

The system should have advantages over older systems for smoke detection. In buildings, for example, the smoke detectors can detect smoke inside because the particles of smoke is more concentrated, as there is less chance of wind separating the particles. This is why they are effective indoors. Outdoors, however, there are not many forms of fire detection which is cost effective. One of the best methods to find a fire is using Infrared cameras, as they can pick up large concentration of infrared light coming from fires. However, Infrared cameras are expensive and generally not suitable for the outdoors. This is one of the main reasons why the author would prefer to use normal cameras, and put more effort into detecting fire and smoke visually, using a program to alert us to fires. Cameras would have great range, and large tracts of land can be covered by using such a system, which would ideally be cheap.

When compared to current fire and smoke detection method, the system is good if it can tell when smoke and fire exists in the picture/video feed. Errors should be minimized. Common smoke detectors are already known to save millions of lives by warning people of fires as they occur inside buildings, and are also known to reduce damage caused by fires, as the fire stations are immediately informed.

As such, a fire and smoke detection based on low cost camera can be used in any places that either smoke or fire can be seen. This is especially useful in open areas, where smoke particles are difficult to detect using old methods. At the same time, it can work in places such as tunnels and indoors. The applications for these projects are many, including monitoring forest fires, keeping a close watch on places where fire would be disastrous such as on the roads and tunnels, where a fire would slow down the traffic drastically and is dangerous to any nearby people. Of course, it can also be used like the conventional indoor fire detecting systems, which are used to detect fires in buildings, which triggers alarms in cases of emergency so that no one will be injured or caught unawares.

This means that the system which this project builds will be usable in almost any environment, and should be cheaper or competitively priced compared to the systems that are currently in the market.

2.2 Technical Analysis

A fire can be detected using various methods. All fires emit an infrared light accompanying varying intensities of the colour red, depending on the temperature. Fires also tend to emit smoke, and a defining trait of fire is the emission of heat. In this project, since the author is dealing with low cost cameras, the author looks at the few optical options that a fire will normally emit.

Infrared light can be detected by certain cameras; however it is not visible to the human eye. This makes it impractical to use in a system meant for cheap cameras which are easily obtainable and therefore is not likely to be able to capture the infrared spectrum. Heat cannot be detected visually, which leaves only the smoke and the flame as easily visible components of a fire.

Aside for clean fires, most fires will emit smoke, which are solids and liquids which are remainders of the fuel which is not cleanly burnt. Smoke is usually visible under normal conditions, and can be an early indicator of fire.

The main problem with smoke is that it is not easy to define, as the particles undergo movement which varies greatly depending on the environment, described as Brownian motion. It is therefore important to have a model for smoke before a program can consider smoke as being detected.

The system would preferably have a smoke detection and fire detection program, which uses an algorithm to detect smoke and fire through the algorithm, which would a picture of the area in which the author wishes to detect a fire if it occurs. The program would sound the alarm once smoke of fire has been determined. Ideally, this includes fires that are just beginning and light traces of smoke such as that from a matchstick, but realistically, a small fire, perhaps the size of a fire produced by a stove, is detected.

Various methods of fire or smoke detection have already been researched and are still undergoing research. The algorithms used widely differ from each other, as well as the method used and calculations.

Fire detection and flame detection rely on the detection of the naked fire in order to work. Several of the methods proposed is to detect the flame and comparing the colors which are similar to fire, like red, orange and yellow. Another method is to detect colours near the infrared range. Both methods of detecting flame required a way to differentiate between actual flames and background colours such as that of a shirt or car, which may be of similar colours.

Several smoke detection methods based on cameras rely on analyzing the random movement of smoke. Various analysis of the smoke's nature is used, including fourier analysis, wavelet analysis, Gray level co-occurrence matrices. The results are then usually separated by a program, commonly a neural network which has been trained, in order to differentiate between fire, smoke and neither, by discrimination processes. As smoke is an early indication of fire, there is more focus on smoke detection than fire detection.

Using the known properties of smoke, various other methods of detecting smoke from a video feed can be made. Evaluating the result or using a neural network to understand how smoke acts like would be a good method. However, because smoke is highly irregular it is difficult to train such a network on a standalone basis.

A few other researchers have already gone into this field, and used various methods to come up with a fire or smoke detection system. They are split into Fire detection and Smoke detection respectively. Each are split into two main methods of solving.

For fire and flame detection, two main methods can be seen from the results from the research. The first would be the use of colour similar to fire in movement. The second would be to analyze the flicker caused by a flame. These are the most common methods. The first method was employed by Y. Le Maoult[7] and Phillips et al[6], who used a colour scheme near the infrared. This scheme used the colour scheme of fire, and analyzes it through movement. If a object with a colour similar to the colour scheme was detected in the video feed, it would be assumed to be fire. The second method, analysis of the flicker of the flame, was employed by Toreyin et al[5]. Toreyin et al used the wavelet analysis, which is detecting wave that oscillates, based on both time and space. This allows the system to be able to detect something like fire, of which the edges flicker in and out. Phillips et al had a system which utilizes colour information, which was then processed and had the motion of the flame filtered out.

Smoke detection is less widespread, as there are many problems that need to be overcome. Smoke signals coming from video tend to have problems. Among the problems which arises is that it is variable in density and is only visible depending on the lighting, background and other objects. Another problem is that not many image features like the edges and movements of smoke are easily characterized, making it difficult to model and therefore allow for a system to detect.

A few methods have been proposed by various other researchers. Fujiwara and Terada[1] proposed to use fractal encoding concepts to extract smoke regions from an image, using the self similarity of smoke shapes. Similarly, Toreyin et al.[3] extracted features such as motion, flickering, edge blurring regions from a video, using spatial and temporal wavelet transformation and background subtraction in order to determine smoke. Ziyou Xiong et al[8] proposed a similarly based method, where flickering pixels are extracted, then the contour is initialized.

Another method is to take advantage of the irregularities in motion due to smoke being not rigid. Kopilovic et al.[2] used this method by computing optical flow field using adjacent images and used the distribution of motion direction to differentiate smoke motion from other motions. Vicente and Guillemant[4] did something similar, extracting local motions from cluster analysis of points in order to track local groups of pixels, and then use features of the velocity distribution histogram to discriminate between smoke and other movements such as trees and clouds.

All these works which have been presented as reference will be used and improved, as quite a lot of them have drawbacks. For instance, the method proposed by Ziyou Xiong[8] on smoke detection, although it is capable of detecting smoke in regions with smoke, it sometimes fails, despite the presence of smoke.

One drawback to a flame detection or smoke detection system is that, sometimes, either flame or smoke is absent, such as in a case where a small fire has started behind the tree, causing the flame to be invisible. Similarly, some kinds of fire do not emit a lot of smoke when they start, and only emit smoke once the fire has grown large enough. This being the case, it would be more beneficial to have a system where both fire and smoke can be detected, and ideally using the same system. This would be one of the greatest advantage of the system this project aims to create.

The systems which were used by others were all tested similarly, that is, using the data set provided by FIRESENSE, which will be what this project will also use as testing for the initial part. However, that will not be all, as testing will be done in the real world as well, just so that it can confirm the results.

From all these research, certain challenges which the author will face have been discovered. The first challenge is to match smoke problems, as the smoke pattern varies with air flow. Another problem would be that images for smoke are difficult to threshold, as the density can vary. Both fire and smoke detection suffers from this problem. Thus, the aim of this project is to solve these problems.

CHAPTER 3

METHODOLOGY

3.1 Research Methods

There will be three main research methods in this project. The first method is one that identifies the problem given and gives the best method for data collection. This will be followed shortly by research to help solve the problem given. After the solution is found, some more in depth research based on the results of the previous research will be done in order to fine tune it so that it’s potential is maximized.

Identifying the problem, which is to create a low cost fire and smoke detector, requires several steps. First, it requires that it is known how fire and smoke is detected. Because the title specifically states the use of low cost cameras, such as CCTV, more focus should be put into those fields. These keywords were used in order to narrow down the data required. A majority of the data collection was done online, although some books from the library were used as reference. Journals and articles are the main source of data for this project.

After the data has been collected, research through all the data will be done. This primarily involves delving deeper into the subject, and understanding the terms which will be used in this project. All the data will be used to compile a solution for the problem. The data from this initial stage can be reviewed in the literature review section.

After the solution is created, it will undergo research via testing. A dataset will be used, or a set of videos which have been used to test all the other methods proposed by other researchers, to test the data. The results should be accurate, that is, there must not be any false positives. The false alarm is an error where, while there is no fire or smoke, the system will give a result indicating smoke.

Similarly, another error would be the true negative, where there is a fire, but the system gives a result indicating that there is no fire or smoke. These are results that are not desirable, as it is preferable that the resulting signal matches the input, detecting fire when there is a fire and detecting nothing when there is no fire.

Another point of testing would be the real world testing, where a fire would be started in various places. In order for the system to have significantly less error, the tests should be done repeatedly.

For the results to be accurate, various places should be tested. A few of the testing places would be out on the road, in a dark or darkened road, inside a building. In order to test the smoke detection as well, the fire could also behind some object so that the flame cannot be seen.

The result should be done again during various stages of the day, as visual based systems may respond differently depending on the condition. In order to test if the system can handle difficult situations, it is also important to test the system under various weather conditions. The system should be tested when it is raining, cloudy and under bright sunshine.

As the placing of the camera may result in differences, there should be various angles of positioning in order to get the best result. The camera should be placed differently so that the effectiveness of the system can be seen. For instance, the camera could be placed with the fire as focus and with the fire out of focus. Also, from both top view and bottom view, as well as from the side.

Once a suitable method is found, more research will be needed based on the results such that there will be less error, allowing the maximum effectiveness of the solution. If the solution is successful, the project will have come to an end, as the objectives will have been achieved.

3.2 Method of Solution

After observing smoke on low cost cameras, it can be observed that most smoke has a constant colour, which is gray. However, this feature, on its own, is not very reliable as many things around it can be of that same colour, such as roads, cars and clouds. Another observable fact is that smoke usually moves about. Therefore, where a still image may be insufficient, a moving video might give a better result. The movement of smoke at the base of the fire is usually quite fast when compared to other moving objects such as clouds. Also, the area of smoke usually grows larger as time goes on, and maintains its size or splits when it becomes too large. Using these observations, an attempt to develop a solution was made.

In order to reduce the amount of area needed to analyze, the first step is to determine the motion in the video. This was done by subtracting the image with a previous image, which results in the differences in the two images, which are the parts that have changed. The positions of where these parts are are kept in the program for future references.

The areas are then checked to see if they are of a gray colour. Because not all the points inside the smoke is necessarily gray due to holes, a certain percentage is taken, such as 75% and above.

The growth of the smoke is then checked. This is done by comparing the area and position of the moving area which is gray and the previous area which meets the criteria. If the positions coincides and the area is larger than the previous area, the system is determines that the area has grown.

If the area is smaller, but there is more than one of the areas which is grey inside, it is considered to have split. Otherwise, the area is considered to have shrunk.

Using the above values, the author can assume that an area has smoke if the image has an area that is gray, and is either growing or splitting. In order to increase the accuracy of the system, several readings of smoke should be taken before smoke is assumed to exist in the video.

3.3 Method Used

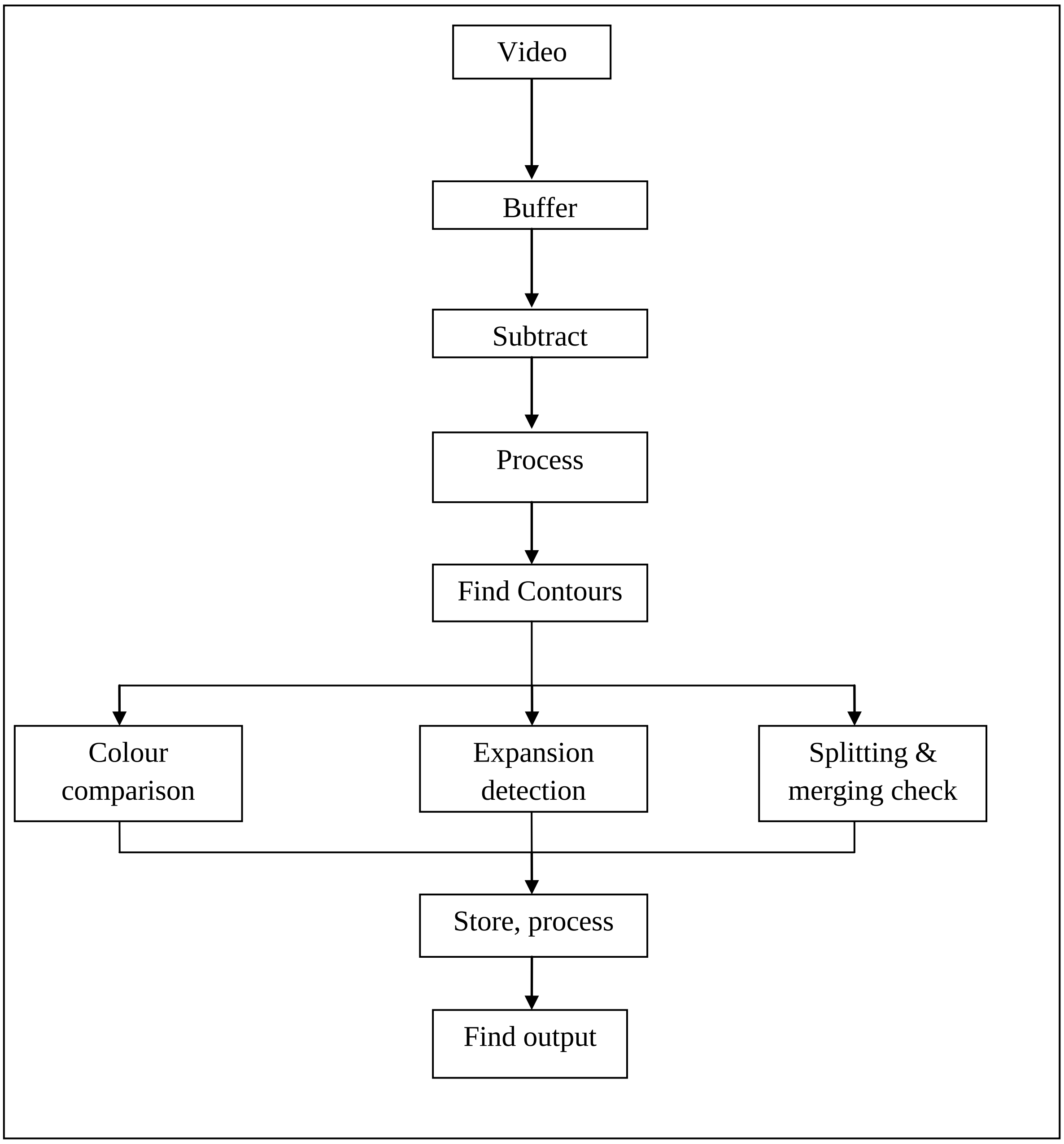

Figure 3.3.1 System Flowchart

In order to detect fire using a video feed, a program needs to be written to process the video feed. Figure 3.2.1 shows the process flow or this program. As can be seen, the program starts by storing up a buffer for the images, which are used for comparing to each other in order to check for movement.

The first step is to subtract the current image from a previous one in order to detect movement, as the difference between the images will show movement. Image processing is then done in order to reduce noise in the image and make the movement clearer. The contours are then taken from the subtracted image. The result of the subtraction can be seen in the figure 3.3.1.

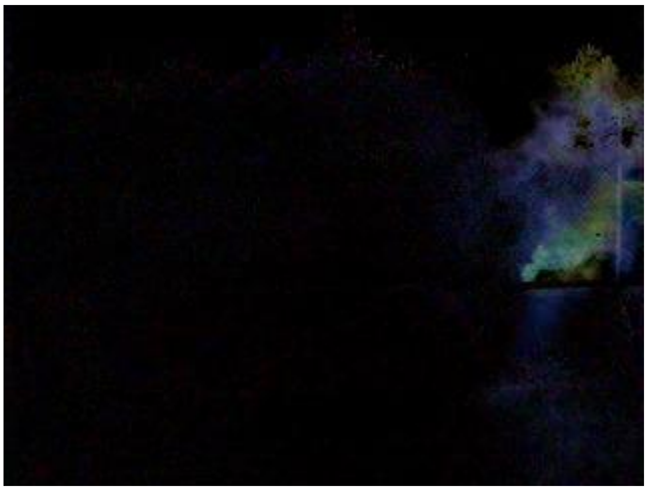

Figure 3.3.2 Subtracted image

The image is then turned into grayscale in order to threshold it. The threshold image is the image where there are only two values, either black or white. The threshold level should be set so that the smoky contour can still be seen after thresholding.

Figure 3.3.3 Grayscaled Image

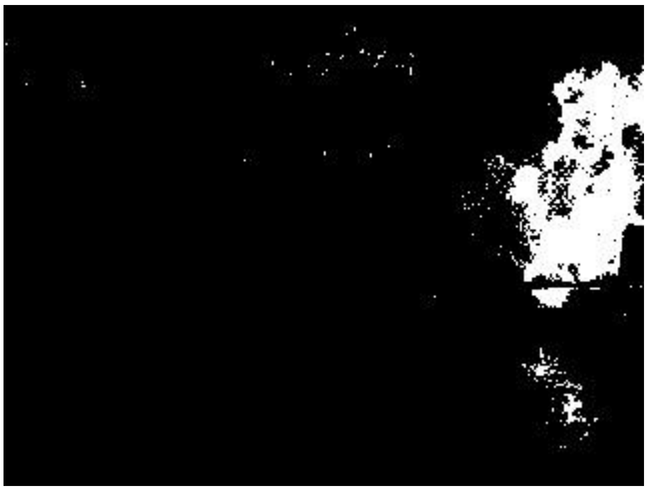

Figure 3.3.4 Thresholded Image

The processing of the contour includes finding out the size, colour and movement of the contours between each image. With this information, the growth of each contour, as well of information on whether the contours split apart from each other or merge together can be known. The colour of the insides contour is checked for good measure, since the colour of smoke is consistently gray for non-chemical smoke.

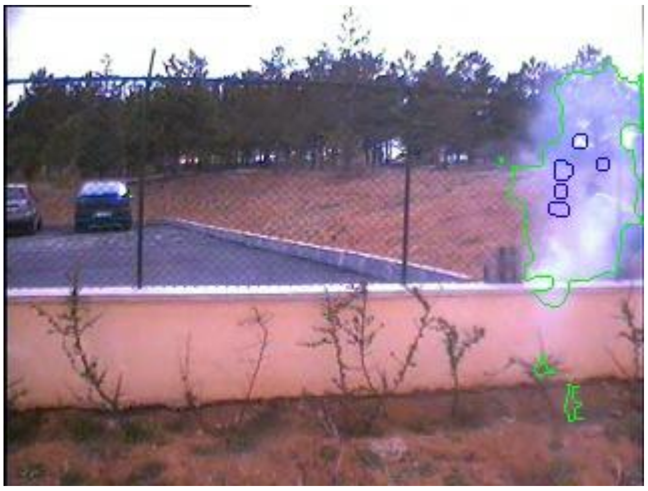

Figure 5.3.4 Contour drawn

The size of the each contour is compared to the previous image's contour size in the general area. If the contours overlap, it is assumed to be the continuation of the previous contour. Otherwise, the contour is labeled as a new contour. Comparing the number of pixels inside each contour gives the information of whether the contour has increased in size or decreased in size. This is useful, because normal moving objects has very little increase in size, while smoke increases in size in a short amount of time.

For each contour that is detected, the colour of the pixels are extracted and processed. Here, the Red, Green and Blue (RGB) values are taken and compared to that of smoke. The simplest method is to compare the RGB values, as it can be noticed that the RGB values are equal for most shades of gray, i.e. R=G=B.

The movement of the smoke contour is also taken into account, using the similar method for measuring the change in size of the contours. Unique contour movement such as merging together and splitting apart is noted, as smoke contours have a habit of splitting and merging in short periods, while a solid moving object's contour is less likely to split and merge in a short period of time.

The areas which fit a certain criteria, such as being smoke coloured and is growing, is kept track of using an array, which has a decaying factor.

$$A_{ij}(t) = \rho A_{ij}(t-1) + C_{ij}$$

$$C_{ij}(t) = \begin{cases} 1, & Contour\ at\ i,j \\ 0, & Otherwise \end{cases}$$

Where:

A is the value tracked

ρ is the decaying factor

t is the time

This step is to detect the source of the smoke, if it exists on the video feed. Otherwise, it will show the side of the video from which the smoke is emitting. This method uses the rationale that the source of smoke would have the most contours over a certain period of time, as the direction of smoke would change depending on the wind and naturally moves upwards. However, the source of the smoke would remain in the same area, so the area where the concentration of detected contours would be targeted as the origin of the smoke.

CHAPTER 4

RESULTS AND ANALYSIS

4.1 Experiment

The results of this method are explained in the following segment. In order to tell how effective the program is, an algorithm is added to the program to tell whether flame is detected individually in each frame. Using different videos, the result of all the videos are compared to the videos themselves. There are four main results that can be the outcome, which are the true positive, the true negative, the false positive and the false negative.

The results are only correct for the true positive and false negative. The other two results are considered to be incorrect. The ratio of correct readings is taken to measure the effectiveness of the program.

The program is altered to list the values in a text file, which will be compared to the video itself in order to determine how accurate the program is. A few different videos will be used in this test, half of which will include smoke, the other half will not. One will be a video of a road, where cars of various colors will pass through, in order to test the system.

The results are tabled as can be seen in Table 4.1.1, and the ratios for accuracy are taken. The delay before the system detects a fire when there is one is also taken into account.

4.2 Results

Table 4.2.1 Accuracy of system for Video

	Video 1 (smoke)	**Video 2 (canfire)**	**Video 3 (traffic)**	**Video 4 (field)**	**Video 5 (nite)** Extreme lighting
True Positive	93	99	0	0	0
False Negative	7	1	0	0	0
True Negative	0	0	89	100	40
False Positive	0	0	11	0	8
Accuracy	93%	99%	89%	100%	83%

Where

$$Accuracy = \frac{Correct\ Readings}{Total\ Readings} \tag{4.1}$$

True positive: smoke correctly detected as smoke

False positive: no smoke incorrectly identified as smoke

True negative: no smoke correctly identified as no smoke

False negative: Smoke incorrectly identified as no smoke.

The average accuracy is 92.8%

The average delay is 20 frames, which would equate to 4 seconds.

Specificity relates to the ability of the test to identify negative results.

$$\text{specificity} = \frac{\text{number of True Negatives}}{\text{number of True Negatives} + \text{number of False Positives}}$$

Sensitivity relates to the test's ability to identify positive results.

$$\text{sensitivity} = \frac{\text{number of True Positives}}{\text{number of True Positives} + \text{number of False Negatives}}$$

For this system, the specificity is 92.3%, and the sensitivity for this system is 96%.

4.3 Analysis

As can be seen from the experimental results above, the system seems to work well. The chance of false positives is much higher than that of true negatives.

The results also show that the case with the worst result is the video “nite”, which is a video of a bright light going past on a dark road. Another dark road was also shown and gave similar results. This implies that the system fares very poorly in dark conditions.

The experiment shows promising result aside from that, as it can be seen that the accuracy rate is quite high for the other videos. Video “Canfire” and “field” have very high accuracy, while “smoke” and “traffic” has reasonably high accuracy.

4.4 Discussion

C++ and OpenCV were used in this project. The use of both software is because of its many advantages to using them. Among the advantages of using C++ is that it is portable, as it is easy to install and is one of the platforms on which OpenCV works with. C++ is also fast as it does not incur the run-time expense and garbage collection associated with most "pure" object oriented languages. (Capabilities of C++, 2011)

The debugger comes with C++ makes it easy for a beginner to find mistakes in the program instantly, which is very handy when it comes to troubleshooting. Finally, C++ is relatively inexpensive and does not require a graphical environment, while allowing developers to write code at the appropriate level.

Open Source Computer Vision Library, also known as OpenCV, is an open source freeware which is aimed at computer vision. Among its applications include object identification and motion tracking. It is used in this project because of its versatility as well as the fact that it has a C++ interface. OpenCV runs on most of the major Operating Systems (OS), which makes it useful when using another computer to program or test.

There were quite a lot of videos that were actually used to analyze the effectiveness of the program. The experiment above only used a few for the result, as it would be easier to understand. Various other videos, such as smoke under bright light and calm day smoke were also used during the development of the project.

In the industry, the system is still quite practical. It is useful because it gives an observer or operator a warning if smoke is detected, allowing the operator to have more time on his hand to do other things. As an operator is usually human prone to mind drifting during tedious jobs such as watching cameras, the system can first alert the operator if it suspects something out of the ordinary, and therefore the operator can act on said warning.

The steps needed to implement such a system are also quite simple. The system needs to be connected to a video feed, such as a camera. This can be done simply by directly using the camera to the program. The output can of the system can be obtained and then linked to another program, such as an alarm or an interface that can allow the smoke to be seen on a camera.

CHAPTER 5

CONCLUSION

5.1 Evaluation of Project against its Objectives

The objectives of this project were to create a system which would be able to detect smoke using images from a feed, such as a video feed. The system was made to detect fires while they are still small and have not grown too large. These objectives were achieved, as the system

The system also has reasonable response time which is another objective. The five second delay is much faster than an average point type smoke detector.

The third objective, where the system should not detect smoke and fire when there is none, did not succeed, as sometimes there are false positives. False positives means that, while there is no fire, smoke is detected by the system. This means that the system can be improved by achieving this objective.

5.2 Strengths and Weaknesses

The system uses a low cost camera to capture images in order to determine whether smoke exists in the video caught by the camera. This method therefore has all the weaknesses and strength associated with a machine vision system.

One of the weaknesses in this system is that it cannot detect what it cannot see. If the smoke being emitted in the range of the video is being blocked by an object, or is dispersed quickly by the wind or other external factors, the system will not be able to detect it.

The method used also has its weaknesses, as can be seen from the results. It faces problems when trying to detect smoke in extreme lighting conditions, such as an area that is too bright or too dark, as the system cannot detect motion due to the background being too strong.

Another weakness is that, due to the method, the system recognizes any grey coloured shaped object that grows larger as smoke. This implies that a mostly grey object moving towards the camera satisfies the criteria, and has a chance of being considered smoke.

The strength of the system lies in the speed of the system. As the system only requires a certain number of frames before giving out the output, the system is not instantaneous; however, it is still faster than smoke particle detectors, especially if the fire is a distance away.

Another boon of the system is the range. As the smoke detection algorithm works on the entire screen of the video, the range of smoke detection depends on the range of which the camera can capture. It is especially useful in large open areas, such as a field, where smoke particle detectors would have difficulty, because of the smoke dissipating before reaching the sensors.

5.3 Recommendations

In order to cover up the system's weaknesses, several steps can be taken to further improve upon it. Some of these steps taken can complement the system.

For instance, a flame detection algorithm, which is a system that is able to detect flames, can be used to cover the smoke detection system's weakness in the dark. In the darkness, a fire would produce flame as well as smoke, of which flame can be clearly seen in the dark while smoke cannot.

An object detection algorithm could also be used to improve this project. As smoke does not have a clearly defined shape, it should not trigger object detection programs. This allows us to certify what the movement in the video is, as if the movement comes from an object, it is not smoke.

Since the percentage error of true negatives is low compared to false positives, the system can be implemented immediately as long as there is an operator around to confirm the results from the system. Even if the system gives out a false positive, the observing operator would be able to confirm whether it is a true fire or not. This allows an operator to be assisted by the system.

An Artificial Intelligence neural network can also be used to process the videos in order to reduce the false positive errors, as the neural network can use the data obtained more effectively than the program by itself.

For closed spaces, such as office rooms, it is better for the system to work in conjunction with normal smoke particle detectors, since furniture and other objects may impede the camera's vision range. Despite being slower, it would be better if the smoke detector detected the smoke at a later time than not at all.

5.4 Conclusion

Based on the results produced, the system is shown to be effective at detecting smoke. Although the system has its limitations and drawbacks, it can be further improved in order to reduce the drawbacks, making it a cost effective and fast method of detecting smoke and fire.

As fire should be put out early in order to prevent spreading, the speed of the fire detection system is important, as this allows the fire to be put out while small, restricting the damage that the fire may do.

The main smoke detectors currently in the market are expensive to set up and are usually limited to certain areas only, reducing the cost. The response time is also slower, as it requires smoke and heat to dissipate. This makes the vision based smoke detector a very viable replacement for these systems, as it eliminates the weaknesses of current smoke detectors.

Many improvements can be implemented to this system in order to improve it so that the system can perform better. The technology can even be merged with other technologies, and work in conjunction with other systems, such as security systems, in order to save on costs.

References

[1]N. Fujiwara and K. Terada, “Extraction of a smoke region using fractal coding”, IEEE International Symposium on Communications and Information Technology, 2004, ISCIT 2004, Volume 2, 26-29 Oct. 2004, Page(s):659 - 662.

[2]I. Kopilovic, B. Vagvolgyi, and T. Sziranyi, “Application of panoramic annular lens
for motion analysis tasks: surveillance and smoke detection”, Proceedings of 15th International Conference on Pattern Recognition, 2000, Volume 4, 3-7 Sept. 2000 Page(s):714 - 717.

[3] B. U. Toreyin, Y. Dedeoglu, and A. E. Cetin, “Wavelet based real-time smoke detection in video,” in EUSIPCO ’05, 2005.

[4]J. Vicente, and P. Guillemant, “An image processing technique for automatically detecting forest fire”, International Journal of Thermal Sciences
Volume 41, Issue 12, December 2002, Pages 1113-1120.

[5]B.U. Toreyin, Y. Dedeoglu, U. Gudukbay, A.E. Cetin, “Computer vision based method for real-time fire and flame detection”, Pattern Recognition Letters. 27 (2006) 49–58.

[6]W. Phillips, M. Shah, N.V. Lobo, “Flame recognition in video”, in: Fifth IEEE Workshop on Applications of Computer Vision, 2000, pp. 224–229.

[7]Y. Le Maoult_, T. Sentenac, J. J. Orteu and J. P. Arcens, “Fire Detection: A New Approach Based on a Low Cost CCD Camera in the Near Infrared”, Trans IChemE, Part B, Process Safety and Environmental Protection, 2007, 85(B3): 193–206

[8]Ziyou Xiong, Rodrigo Caballero, Hongcheng Wang, Alan M. Finn, Muhidin A. Lelic, and Pei-Yuan Peng, “Video-based Smoke Detection: Possibilities, Techniques, and Challenges”

[9]ByoungChul Ko, Kwang-Ho Cheong, Jae-Yeal Nam, “Early fire detection algorithm based on irregular patterns of flames and hierarchical Bayesian Networks”, Fire Safety Journal l45(2010)262–270

Appendix

```
// FireDetect.cpp : Defines the entry point for the console
application.
//

#include "stdafx.h"

#include <cv.h>
#include <cxcore.h>
#include <highgui.h>
#include <iostream>
#include <vector>//for vector
#include <math.h>// for sqrt
#include <algorithm>//vector calculation
using namespace std;


      IplImage* frame[30];
      IplImage* frameG[30];
      IplImage* frame2[30];
      IplImage* frame2G[30];
      IplImage* frame3[30];
      IplImage* frame3G[30];
      IplImage* frame4G[30];
      IplImage* frame4GG[30];




      FILE * fp=fopen("experiment.txt","w");
      double growth[2]={0,0};
      int speedcounter=0;
      int splitcheck=0,split2 = 0,smokechecker=0;
      double area3[30],spdgr;
void countourarea(double area1);
void colourtesting(IplImage*frameb);
      CvSeq* contour2;
      CvSeq* contour;
      CvMemStorage* storage;
      //rect
      vector<CvRect> Recty;
      vector<CvRect> Rectx;

void contourprocess(IplImage*frameb,IplImage*framec,CvSeq*
contour,double area1);
void ghosteliminator(IplImage*frameb,IplImage*framec);
void centertrack(IplImage* frameb);
void splitmergegrowshrink();
void output(IplImage*frameb);

struct rectinfo{
      int coordinatex;//center of the rectangle
      int coordinatey;
      int height;
      int width;
      int smokeval;
      };
vector<rectinfo> xyhw,xyhwpast;
vector<vector<int>> dynamicArray;
vector<vector<int>> summon;
int var=0;//for output of images
int _tmain(int argc, _TCHAR* argv[])//MainProgram
{

      int b=0,a=0,d=0;

      //specify the input
      CvCapture* capture;
      int input=0,bfv = 0,counter = 0;
      double mean=10;
      double area1 = 0;

      cout<<"Please specify input\nvideo(1) or webcam(2)?";
      cin>> input;

      if (input == 1)
      {
      /*video1*/
      char filename[255];
      memset(filename,0,255);
      cout<<"Please key in the adress to the video\n";
      cin>>filename;
      capture = cvCreateFileCapture( filename );
      /*capture = cvCreateFileCapture("smoke.avi");*/
      }
      if (input == 2)
      {     capture =cvCreateCameraCapture(0);}
      //declare stuff

      IplImage* start;
      IplImage* edges[30];
      IplImage* edge[30];
      IplImage* remainder[30];
      IplImage* edgesC[30];
      IplImage* edgeC[30];
      IplImage* remainderC[30];

      start = cvQueryFrame(capture);
//initialize images?
      for (int counter =0;counter<30;counter ++)
      {

            frame [counter] = NULL;
            frameG [counter] = NULL;
            frame2 [counter] = NULL;
            frame2G [counter] = NULL;
            frame3 [counter] = NULL;
            frame3G [counter] = NULL;
            frame4G [counter] = NULL;
            frame4GG [counter] = NULL;
            edges[counter] = NULL;
            edge[counter] = NULL;
            remainder[counter] = NULL;
            edgesC[counter] = NULL;
            edgeC[counter] = NULL;
            remainderC[counter] = NULL;
      }
      //grab image to begin working
      cvNamedWindow( "FireDetect", CV_WINDOW_AUTOSIZE );
      CvMemStorage* storage=NULL;
      CvMemStorage* storage2=NULL;
```

```
//imageCapture and process to said images
      while(1)
      {
            if(storage==NULL)storage = cvCreateMemStorage(0);
            if(storage2==NULL)storage2 = cvCreateMemStorage(0);
            a=0;
            /*IplImage* tImg = cvQueryFrame( capture );
            frame[a] = tImg;*/
            frame[a] = cvQueryFrame( capture );
            IplImage* _img= cvQueryFrame( capture );
            if(_img==NULL)break;

            frame[a]= cvCreateImage(cvGetSize(_img),_img-
>depth,_img->nChannels);
            frameG[a]=cvCreateImage(cvGetSize(_img),8,1);
            frame2[a]=cvCreateImage(cvGetSize(_img),_img-
>depth,_img->nChannels);
            frame2G[a]=cvCreateImage(cvGetSize(_img),8,1);
            frame3[a]=cvCreateImage(cvGetSize(_img),_img-
>depth,_img->nChannels);
            frame3G[a]=cvCreateImage(cvGetSize(_img),8,1);
            frame4G[a]=cvCreateImage(cvGetSize(_img),8,1);
            frame4GG[a]=cvCreateImage(cvGetSize(_img),8,1);
            edges[a] = cvCreateImage(cvGetSize(_img),8,1);
            remainder[a] = cvCreateImage(cvGetSize(_img),8,1);
            edge[a] = cvCreateImage(cvGetSize(_img),8,1);
            edgesC[a] = cvCreateImage(cvGetSize(_img),_img-
>depth,_img->nChannels);
            remainderC[a] = cvCreateImage(cvGetSize(_img),_img-
>depth,_img->nChannels);
            edgeC[a] = cvCreateImage(cvGetSize(_img),_img-
>depth,_img->nChannels);

            cvCopy(_img,frame[a]);

            if( !frame ) break;
            cvShowImage( "FireDetect", frame[a] );
            char c = cvWaitKey(33);
            if( c == 27 ) break;

            cvCvtColor(frame[a], frameG[a], CV_BGR2GRAY);
            //findmean
            CvScalar avg = cvAvg(frameG[a]);
            mean =(avg.val[0]/6);
            //release
            if(frame[29])
                  cvReleaseImage(&frame[29]);
            if(frameG[29])
                  cvReleaseImage(&frameG[29]);
            if(frame2[29])
                  cvReleaseImage(&frame2[29]);
            if(frame2G[29])
                  cvReleaseImage(&frame2G[29]);
            if(frame3[29])
                  cvReleaseImage(&frame3[29]);
            if(frame3G[29])
                  cvReleaseImage(&frame3G[29]);
            if(frame4G[29])
                  cvReleaseImage(&frame4G[29]);
            if(frame4GG[29])
                  cvReleaseImage(&frame4GG[29]);
            if (edges[29])
            {
      cvReleaseImage(&edges[29]);cvReleaseImage(&edge[29]);}
            if (remainder[29])
                  cvReleaseImage(&remainder[29]);
            if (edgesC[29])
            {
      cvReleaseImage(&edgesC[29]);cvReleaseImage(&edgeC[29]);}
            if (remainderC[29])
                  cvReleaseImage(&remainderC[29]);
            //move stuff by step
            for ( int counter2=29;counter2>(0);counter2--)
            {
                  frame[counter2]= frame [(counter2-1)];
                  frameG[counter2] = frameG [(counter2-1)];
                  frame2[counter2]= frame2 [(counter2-1)];
                  frame2G[counter2] = frame2G [(counter2-1)];
                  frame3[counter2]= frame3 [(counter2-1)];
                  frame3G[counter2]= frame3G [(counter2-1)];
                  frame4G[counter2] = frame4G [(counter2-1)];
                  frame4GG[counter2] = frame4GG [(counter2-1)];
                  edges[counter2] = edges[(counter2 - 1)];
                  remainder[counter2] = remainder[(counter2 - 1)];
                  edge[counter2] = edge[(counter2 - 1)];
                  edgesC[counter2] = edgesC[(counter2 - 1)];
                  remainderC[counter2] = remainderC[(counter2 - 1)];
                  edgeC[counter2] = edgeC[(counter2 - 1)];
            }
            //BufferValue
            //if (bfv<=50)
                  bfv++;

            //let b and a be a constant with difference
            a=0;
            b=15;
            d=24;

            //compare to previous frame if any,note b is the oldest
frame and a is newest
            if ( bfv >=26)
            {
                  cvSub(frame[b],frame[a],frame2[a]);
                  cvSub(frame[5],frame[a],frame3[a]);
                  //need to compare mean
                  cvCvtColor(frame2[a],frame2G[a],CV_BGR2GRAY);
                  cvCvtColor(frame3[a],frame4G[a],CV_BGR2GRAY);

      cvThreshold(frame2G[a],frame3G[a],mean,255,CV_THRESH_BINARY);

      cvThreshold(frame4G[a],frame4GG[a],mean,255,CV_THRESH_BINARY);
                  cvSmooth(frame3G[a],frame3G[a],CV_MEDIAN);
                  cvSmooth(frame4GG[a],frame4GG[a],CV_MEDIAN);
                  cvDilate(frame3G[a],frame3G[a],NULL,2);
                  cvErode(frame3G[a],frame3G[a],NULL,2);
                  cvDilate(frame4GG[a],frame4GG[a],NULL,2);
                  cvErode(frame4GG[a],frame4GG[a],NULL,2);
      cvFindContours(frame3G[a],storage,&contour,sizeof(CvContour),C
V_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE );

      cvFindContours(frame4GG[d],storage2,&contour2,sizeof(CvContour
),CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE);


//DO NOT REMOVE ANYTHING ABOVE THIS POINT

                  //start of tracking corner (edges) in image?
                  cvDilate(frameG[a],edges[a],NULL,1);
                  cvErode(edges[a],edges[a],NULL,1);
                  cvCanny(edges[a],edges[a],128,255);
                  /*cvDilate(edges[a],edges[a],NULL,1);
                  cvErode(edges[a],edges[a],NULL,1);*/
      //          cvNamedWindow("debuggin2");
      //          cvShowImage("debuggin2",edges[0]);

      cvThreshold(edges[a],edges[a],128,8,CV_THRESH_BINARY);
                  //colouring
                  cvConvertScale(frame[a],edgesC[a],0.00333203125);

                  if (edges[29]!=NULL)
                  {
                  for (int counter=29;counter>0;counter--)
                        cvAdd(edges[counter],edges[(counter-
1)],remainder[(counter)]);

                  for (int counter=29;counter>0;counter--)
                        cvAdd(remainder[counter],remainder[(counter-
1)],remainder[counter-1]);


      cvThreshold(remainder[a],remainder[a],250,255,CV_THRESH_BINARY
);

      //subtract contour curren from contour background
                  cvSub(remainder[a],frameG[a],edge[a]);
      //endcontourbackground

      //trying colour background image

                  for (int counter=29;counter>0;counter--)
                        cvAdd(edgesC[counter],edgesC[(counter-
1)],remainderC[(counter)]);

                  for (int counter=29;counter>0;counter--)

      cvAdd(remainderC[counter],remainderC[(counter-
1)],remainderC[counter-1]);

      //trying colour background image /end
                  }


      cvThreshold(edge[1],edge[2],1,255,CV_THRESH_BINARY);

                  double SE=0,RE=0;
                  CvScalar Sedge = cvSum(remainder[a]);
                  CvScalar Redge = cvSum(edge[a]);

                  SE =(Sedge.val[0]);
                  RE =(Redge.val[0]);


      //          cvNamedWindow("debuggin");
      //          cvShowImage("debuggin",remainder[a]);

//DO NOT MODIFY BELOW THIS POINT
//Contour Processes HERE!!!
IplImage*frameb = frame[15];
IplImage*framec = frame[16];
contourprocess(frameb,framec,contour,area1);



                        //clearing stuff,
                        if (contour!=NULL)
                              cvClearSeq(contour);
                        if (contour2!=NULL)
                              cvClearSeq(contour2);
                        cvClearMemStorage(storage);
                        cvClearMemStorage(storage2);

                  }
            }//while loop end
            printf("End of File\n");
            cvReleaseMemStorage(&storage);
            //cvReleaseMemStorage(&storage2);
            cvReleaseCapture( &capture );
            cvDestroyWindow( "FireDetect" );
            cvDestroyAllWindows();
            fclose(fp);
            return 0;
      }
```

```
//void stuff here...


struct garupa{
	vector<CvRect>grouped;
	CvRect overallRect,overallEnd;
};
vector<CvRect> allcontours;
vector<int>decopy;
vector<garupa> grouper;
rectinfo xyhw_;
int Contcounter=0;
int area1=0;
CvScalar s,r;
int framecounter = 0;
struct gray{//colour checking gray
	double grey;
	int positiongreyx,positiongreyy,xspot,yspot,xlength,ylength;
	CvScalar Outline;
};
vector<gray> greyness;
vector<gray> greynessp;
gray gravy;
garupa objGrp;
int errorchecker = 0;
vector<gray> flamechecker;
vector<gray> flameprevious;
int experimentcounter = 0;

double onecontarea;
vector<double> contarea,contareap;
struct individualcontourinformation{
	vector<int>
contournumber,colourgrey,colourred,splitmerge,splitmerge2,growshrink
,mimic,sizemin,smokeval;
}unit;








void contourprocess(IplImage* frameb,IplImage* framec,CvSeq
*contour,double area1)//ContourProcess
```

```
{
	int b= 15,a=0;
	double Side, Adjacent, Hypertenuse;
			//finding contour
				if (contour!=NULL)

	cvDrawContours(frameb,contour,cvScalar(0,255,0,0),cvScalar(255
,0,0,0),255);

				char imagestring[255];
		memset(imagestring,0,255);
		sprintf_s(imagestring,"C:\\Documents and
Settings\\Ng\\My Documents\\Visual Studio
2008\\Projects\\FireDetect\\FireDetect\\savedstuff\\Image_%d.jpg",va
r++);
		cvSaveImage(imagestring,frameb);

	/*if (contour2!=NULL)
	cvDrawContours(frame[b],contour2,cvScalar(0,0,255,0),cvScalar(
255,0,0,0),255);*/

	cvNamedWindow("contours");
	cvShowImage("contours",frameb);

//	cvNamedWindow("debugging2");
//	cvShowImage("debugging2",frame2[a]);
//	cvNamedWindow("debugging3");
//	cvShowImage("debugging3",frame2G[a]);
//	cvNamedWindow("debugging4");
//	cvShowImage("debugging4",frame3G[a]);

	for( CvSeq* e = contour; e!= 0; e = e->h_next )
	{
		onecontarea = fabs(cvContourArea(e));
		area1 = area1 + onecontarea;
		CvRect z=cvBoundingRect(e,0);
	//
	cvDrawRect(frameb,cvPoint(z.x,z.y),cvPoint(z.x+z.width,z.y+z.h
eight),cvScalar(255,0,0,0));
		Recty.push_back(z);
		Contcounter++;
		contarea.push_back (onecontarea);

// Contour Area, a void function
// countourarea( area1);
		allcontours.push_back(z);
	}

	int i = 1;
	for( int j=0 ;j<Contcounter; j++)
	{
		unit.contournumber.push_back(i);
		i++;
	}

//Grouping
	for( int i=0; i<Contcounter;i++)
	{
		xyhw_.coordinatex =Recty[i].x + (Recty[i].width/2);
		xyhw_.coordinatey =Recty[i].y + (Recty[i].height/2);
		xyhw_.height = Recty[i].height;
```

```
			xyhw_.width = Recty[i].width;
			xyhw.push_back(xyhw_);
		}
//	contour colour
colourtesting(frameb);
splitmergegrowshrink();
ghosteliminator(frameb,framec);
output(frameb);
centertrack(frameb);


//for (int i=0;i < (int)greynessp.size();i++)
//	{
//		cvDrawRect(frameb,cvPoint(greynessp[i].xspot,
greynessp[i].yspot),
//
	cvPoint((greynessp[i].xspot+greynessp[i].xlength),(greynessp[i
].yspot +greynessp[i].ylength)),
//			greynessp[i].Outline);
//	}

//important resizing
	decopy.resize(xyhw.size());
	for(int i =0;i<(int)decopy.size();i++)
	{
		decopy[i]=-1;
	}
	double range,reverserange;

	for (int i=0;i<Contcounter;i++)
	{
		if(decopy[i]<0)
		{
			objGrp.grouped.push_back(allcontours[i]);
		}
		range =
(xyhw[i].height)*(xyhw[i].height)+(xyhw[i].width)*(xyhw[i].width);
		for (int j=i+1;j<Contcounter;j++)
		{
			reverserange =
(xyhw[j].height)*(xyhw[j].height)+(xyhw[j].width)*(xyhw[j].width);
			Side = abs(xyhw[i].coordinatex-
xyhw[j].coordinatex);
			Adjacent = abs(xyhw[i].coordinatey-
xyhw[j].coordinatey);
			Hypertenuse = sqrt(((Side*Side)/4) +
((Adjacent*Adjacent)/4));		//<--modify later
			if(Hypertenuse<=(sqrt(range)))
			{
				if(decopy[i]<0)
				{
					if (decopy[i]<decopy[j])
					{
						decopy[i]=decopy[j];
					}else
					{

	objGrp.grouped.push_back(allcontours[j]);

						decopy[i] = i;
						decopy[j] = i;
```

```
                                    }
                              }else if(decopy[j]<0)
                              {
                                    if (decopy[j]<decopy[i])
                                    {
                                          decopy[j] = decopy[i];
                                    }else
                                    {

      objGrp.grouped.push_back(allcontours[j]);

                                          decopy[i] = i;
                                          decopy[j] = i;
                                    }
                              }
                        }else if(Hypertenuse<=sqrt(reverserange))
                        {
                              objGrp.grouped.push_back(allcontours[j]);
                              if (decopy[j]<decopy[i])
                              {
                                    if(decopy[j]>=0)
                                          decopy[i] = decopy[j];
                                    else
                                          decopy[j] = decopy[i];
                              }else if (decopy[j]>decopy[i])
                              {
                                    if(decopy[i]>=0)
                                          decopy[j] = decopy[i];
                                    else
                                          decopy[i] = decopy[j];
                              }
                        }
                  }
                  objGrp.overallRect.x = INT_MAX;
                  objGrp.overallRect.y = INT_MAX;
                  objGrp.overallEnd.x = 0;
                  objGrp.overallEnd.y = 0;

                  if(objGrp.grouped.size()>0)grouper.push_back(objGrp);
            }

            for (int i=0;i<(int)grouper.size();i++)
            {
                  for (int j=0;j<(int)grouper[i].grouped.size();j++)
                  {
                        if(grouper[i].grouped[j].x <
grouper[i].overallRect.x)
                        {
                              grouper[i].overallRect.x =
grouper[i].grouped[j].x;
                        }
                        if(grouper[i].grouped[j].y <
grouper[i].overallRect.y)
                        {
                              grouper[i].overallRect.y =
grouper[i].grouped[j].y;
                        }
                        if(grouper[i].grouped[j].x +
grouper[i].grouped[j].width > grouper[i].overallEnd.x)
                        {
                                grouper[i].overallEnd.x  =
grouper[i].grouped[j].x+grouper[i].grouped[j].width;
                        }
                        if(grouper[i].grouped[j].y +
grouper[i].grouped[j].height > grouper[i].overallEnd.y)
                        {
                                grouper[i].overallEnd.y  =
grouper[i].grouped[j].y+ grouper[i].grouped[j].height;
                        }
                }
        }
        //grouping done!(don't remove the above)

        //scan all value, take last similar value.
        vector<int> lumped;
        vector<int>::iterator leong;
        vector<garupa> lumpy;

        for (int i=0;i<(int)grouper.size();i++)
        {
                lumped.push_back ( decopy[i]);
        }
        sort(lumped.begin(),lumped.end());
        leong = unique( lumped.begin(), lumped.end() );
        lumped.erase( leong, lumped.end() );

        for(int i=0; i<(int)lumped.size();i++)
        {

                //while (decopy[j]!=lumped[i])
                for (int j=((int)grouper.size()-1);j>0;j--)
                {
                        //find value of lumped from behind
                        if (decopy[j]==lumped[i])
                        {
                                lumpy.push_back (grouper[j]);
                                break;
                        }
                }
        }



        for (int i=0;i<(int)lumpy.size();i++)
        {
                if (lumped[i]!=-1)//remove loners
                {
                        for (int j=0;j<(int)lumpy[i].grouped.size();j++)
                        {
                                cvDrawRect(frameb,
                                        cvPoint(lumpy[i].overallRect.x,
                                        lumpy[i].overallRect.y),
                                        cvPoint(lumpy[i].overallEnd.x,
                                        lumpy[i].overallEnd.y),
                                        //gravy.Outline);
                                        cvScalar(0,255,255));
                        }
                }
        }
//splitmerge
        double totalarea=0,pastotarea=0;
```

```
		for (int i=0;i<(int)lumpy.size();i++)
		{

			totalarea = contarea[i]+totalarea ;

			for(int j=0;j<(int)xyhwpast.size();j++)
			{
			//previous square same location area of contour in
contact. find xyhwpast inside
			if((xyhwpast[j].coordinatex <= lumpy[i].overallRect.x)&&
				(xyhwpast[j].coordinatex >=
lumpy[i].overallEnd.x)&&
				(xyhwpast[j].coordinatey <=
lumpy[i].overallRect.y)&&
				(xyhwpast[j].coordinatey >= lumpy[i].overallEnd.y
))
			{	//find the area.
				pastotarea= pastotarea+contareap[j];
			}
			}
			//comparepast to present,
			/*if (pastotarea<totalarea)
			{
				cout<<"totalareal growth";

			}else {
				cout<<"totalareal shrinkage";
			}*/
		}

		for (int i=0;i<(int)grouper.size();i++)
		{
			grouper[i].grouped.clear();
		}
		grouper.clear();

		//for (i=0; i <groupedcontours.size ;i++)
		cvNamedWindow("greymove");
		cvShowImage("greymove",frameb);

		decopy.clear();

		xyhwpast.clear();
		xyhwpast = xyhw;
		xyhw.clear();
		allcontours.clear();
		objGrp.grouped.clear();
		lumped.clear();

		lumpy.clear();

		Contcounter = 0;
		if (contour!=NULL)
		{
			cvClearSeq(contour);
		}
//GroupingEnd

		split2=0;

		Rectx.clear();
```

```
        Rectx = Recty;
        Recty.clear();

        contareap.clear();
        contareap = contarea;
        contarea.clear();
        unit.colourgrey.clear();
        unit.contournumber.clear();
        unit.growshrink.clear();
        unit.splitmerge.clear();

        framecounter++;
}

void colourtesting(IplImage*frameb)// Colour
{
        for( int i=0;i<Contcounter;i++)
        {

                s = cvGet2D(frameb, xyhw[i].coordinatey
,xyhw[i].coordinatex);
                if (s.val[2]=s.val[1] = s.val[0])
                {
                        gravy.grey = s.val[2];
                        gravy.positiongreyx = xyhw[i].coordinatex;
                        gravy.positiongreyy = xyhw[i].coordinatey;
                        gravy.xspot = (xyhw[i].coordinatex-
(xyhw[i].width/2));//point left top corner of rectangle
                        gravy.yspot = (xyhw[i].coordinatey-
(xyhw[i].height/2));
                        gravy.xlength = xyhw[i].width;
                        gravy.ylength = xyhw[i].height;
                        gravy.Outline = s;
                        greyness.push_back(gravy);
                        unit.colourgrey.push_back (1);
                }else{ unit.colourgrey.push_back (0);}

                //flame test
                CvScalar areat;
                double areaT=(xyhw[i].height*xyhw[i].width),areatt=0;
                double areaTT = 0;

                for(int j=0;j<gravy.xlength ;j++)
                {
                        for (int k=0;k<gravy.ylength;k++)
                        {
                                areat =cvGet2D(frameb, gravy.yspot +k
,gravy.xspot +j);
                                if
((areat.val[2]>areat.val[1])&&(areat.val[2]>areat.val[0])&&(areat.va
l[1]>areat.val[0])&&(areat.val[2]>70))
                                {
                                        areatt++;
                                }
                        }
                }
                areaTT = (areatt/areaT);
                if (areaTT>0.6)
                {
                        unit.colourred.push_back(1);
                }else{unit.colourred.push_back(0);}
```

```
            if (areaT>6)
            {
                  for(int j=0;j<gravy.xlength ;j++)
                  {
                        for (int k=0;k<gravy.ylength;k++)
                        {
                              areat =cvGet2D(frameb, gravy.yspot +k
,gravy.xspot +j);
                              if
((areat.val[2]>areat.val[1])&&(areat.val[2]>areat.val[0])&&(areat.va
l[1]>areat.val[0])&&(areat.val[2]>70))
                              {
                                    areatt++;
                              }
                        }
                  }
                  areaTT = (areatt/areaT);
                  /*if (areaTT>0.6)
                  {
                        //if previous spot also moving fire
object,suspect fire

      cvDrawRect(frameb,cvPoint(gravy.xspot,gravy.yspot),cvPoint((gr
avy.xspot + gravy.xlength),(gravy.yspot
+gravy.ylength)),cvScalar(0,0,255),5);
                        flamechecker.push_back(gravy);
                        for (int i = 0;
i<(int)flameprevious.size();i++)
                        {
                              for(int j=0;j<flameprevious[i].xlength
;j++)
                              {
                                    for (int
k=0;k<flameprevious[i].ylength ;k++)
                                    {
                                          if ((gravy.positiongreyx
== (flameprevious[i].xspot +j))&&(gravy.positiongreyy ==
(flameprevious[i].yspot + k)))
                                          {
                                                //cout<<"f";

      //cvWaitKey(0);//pause to see where it is, to be removed in
actual
                                          }
                                    }
                              }
                        }
                  }*/
            }
            flameprevious.clear();
            flameprevious = flamechecker;
            flamechecker.clear();
            //flame test end
            if (greynessp.size()>0)//make sure it exist
            {
                  for (int j=0; j < (int)greynessp.size();j++)
                  {
                        r =
cvGet2D(frameb,greynessp[j].positiongreyy,greynessp[j].positiongreyx
);
```

```
                        if ((r.val[0]=r.val[1] =
r.val[2])&&(r.val[2]<250))
                        {

      if((r.val[2]>(greynessp[j].grey))||(r.val[2]<(greynessp[j].gre
y)))
                              {
                                    //cout<<"g";

                                    //check to see if currently have
any contours on it?add range
                              }
                        }else
                        {
                              if
((r.val[0]==r.val[1])||(r.val[0]==r.val[2])||(r.val[1]==r.val[2]))
                              {
                                    //cout<<"?"<<endl;
                              }
                        }
                  }
            }
      }
      greynessp.clear();
      greynessp=greyness;
      greyness.clear();
}

void splitmergegrowshrink()//input is current frame, previous
frame,unit?,
{
      int splitchecker = 0,mergechecker = 0;
      int A=0,B=0,C=0,D=0,A1=0,B1=0,C1=0,D1=0;
      for (int i=0;i<Contcounter;i++)
      {
            mergechecker = 0;
            splitchecker = 0;
            A = xyhw[i].coordinatex-(xyhw[i].width/2);
            B = xyhw[i].coordinatex+(xyhw[i].width/2);
            C = xyhw[i].coordinatey-(xyhw[i].height/2);
            D = xyhw[i].coordinatey+(xyhw[i].height/2);
            //compare to all previous squares?
            for(int j=0;j<(int)xyhwpast.size();j++)
            {
                  //problem
            /*    if (xyhwpast[j].smokeval==1)
                  {
                        smokechecker++;
                  }*/


                  A1 = xyhwpast[j].coordinatex-
(xyhwpast[j].width/2);
                  B1 =
xyhwpast[j].coordinatex+(xyhwpast[j].width/2);
                  C1 = xyhwpast[j].coordinatey-
(xyhwpast[j].height/2);
                  D1 =
xyhwpast[j].coordinatey+(xyhwpast[j].height/2);
```

```
      if(((A1>=A)||(B1>A)&&((A1<=B)||(B1<=B)))&&(((C1>=C)||(D1>=C))&
&((C1<=D)||(D1<=D))))//inside point(dont care which part? double
check for certaincy later
                  {
                        mergechecker++;
                        if (xyhwpast[j].smokeval==1)
                        {
                              smokechecker++;
                        }
                  }

      if(((A>=A1)||(B>A1)&&((A<=B1)||(B<=B1)))&&(((C>=C1)||(D>=C1))&
&((C<=D1)||(D<=D1))))//inside point(dont care which part? double
check for certaincy later
                  {
                        splitchecker++;
                        if (xyhwpast[j].smokeval==1)
                        {
                              smokechecker++;
                        }
                  }
            }
            //check for merge
            //yes ->count contours, Greater than 1?
            if (mergechecker>1)
            {
                  unit.splitmerge.push_back(1);
            }else
            {
                  unit.splitmerge.push_back(0);
            }
            //check for split
            //compare previous,any contour outside(part if contour)?
            //yes -> note outside contour corners,compare current
frame, no frame contour>1?
            if (splitchecker>1)
            {
                  unit.splitmerge2.push_back(1);
            }else
            {
                  unit.splitmerge2.push_back(0);
            }
            if (smokechecker>0)
            {
                  unit.smokeval.push_back(1);
            }else{unit.smokeval.push_back(0);}

            char m=0;
            if (mergechecker ==1&& splitchecker ==0)//not greater
than one? growth occured(check growth factor
                  m = 'g';
            if (splitchecker == 1&& mergechecker==0)//shrink
occured, check shrink factor?
                  m = 's';
            if(splitchecker ==0&&mergechecker==0)
                  m = 'n';
            smokechecker=0;
            switch (m)
            {
```

```
            case ('g')://not greater than one? growth occured(check
growth factor
                  {
                        unit.growshrink.push_back(1);
                        break;
                  }
            case ('s')://shrink occured, check shrink factor?
                  {
                        unit.growshrink.push_back (2);
                        break;
                  }
            case ('n'):
                  {
                        unit.growshrink.push_back(0);//size same or
does not exist
                        break;
                  }
            default:
                  unit.growshrink.push_back(4);


            }
      }
}
void output(IplImage*frameb)
{
      if (framecounter==0)
      {
            summon.resize(frameb->width,vector<int>(frameb-
>height));
            for(int i = 0;i < (int)summon.size();++i)
            {
                  for(int j = 0;j < (int)summon[i].size();++j)
                  {
                        summon[i][j] = 0;//set 0
                  }
            }
      }
      int smokiness=0;
      for (int i = 0; i <(int)unit.contournumber.size();i++)
      {
            if (unit.sizemin[i]==0&&unit.mimic[i]==0)
            {
                  if
((unit.colourgrey[i]==1)||(unit.colourred[i]==1))
                  {
                        if (unit.growshrink[i]!=0)
                              smokiness++;
                        if (unit.splitmerge[i]==1)
                              smokiness++;
                        if (unit.splitmerge2[i]==1)
                              smokiness++;
                        if (smokiness>1)
                        {
                              xyhw[i].smokeval =1;
                              unit.smokeval[i] = 1;
                        }
                  }
            }
            smokiness=0;
            int j=0;
            if((xyhw[i].smokeval==1))
            {
                  cvDrawRect(frameb,cvPoint((xyhw[i].coordinatex-
(xyhw[i].width/2)),(xyhw[i].coordinatey)-(xyhw[i].height/2)),

      cvPoint((xyhw[i].coordinatex+(xyhw[i].width/2)),(xyhw[i].coord
inatey)+(xyhw[i].height/2)),
                  cvScalar(255,0,0,0));

                  experimentcounter++;
                  if (experimentcounter>2)
                  {
                        cout<<"\a";
                  }

                  for(int a = (xyhw[i].coordinatex-
(xyhw[i].width/2));a < (xyhw[i].coordinatex+(xyhw[i].width/2));++a)
                  {
                        for(int b = (xyhw[i].coordinatey)-
(xyhw[i].height/2);b < (xyhw[i].coordinatey)+(xyhw[i].height/2);++b)
                        {
                              summon[a][b]++;
                        }
                  }

      /*cvDrawContours(frameb,c,cvScalar(255,0,0,0),cvScalar(0,0,0,0
),1,2,8 );*/

      }/*else{cvDrawContours(frameb,c,cvScalar(255,255,0,0),cvScalar
(0,0,0,0),1,1,8 );}*/



      }
fprintf(fp,"%d\t",experimentcounter);
errorchecker++;
      smokechecker=0;
      unit.colourgrey.clear();
      unit.colourred.clear();
      unit.contournumber.clear();
      unit.growshrink.clear();
      unit.splitmerge.clear();
      unit.splitmerge2.clear();
      unit.mimic.clear();
      unit.sizemin.clear();
      unit.smokeval.clear();
}
void ghosteliminator(IplImage*frameb,IplImage*framec)
{
      CvScalar areat,areap;
      for (int i=0;i<(int)xyhw.size();i++)
      {
            double areaT=(xyhw[i].height*xyhw[i].width),areatt=0;
            if (areaT>6)//size filter, if too small return 1

      {unit.sizemin.push_back(0);}else{unit.sizemin.push_back(1);}
            double areaTT = 0;
            for(int j=0 ;j<gravy.xlength ;j++)
            {
                  for (int k=0 ;k<gravy.ylength;k++)
                  {
                        areat=cvGet2D(frameb, gravy.yspot+k
,gravy.xspot+j);
                        areap=cvGet2D(framec, gravy.yspot+k
,gravy.xspot+j);
                        if
(((areat.val[0]==areap.val[0])&&(areat.val[1]==areap.val[1])&&(areat
.val[2]==areap.val[2])))
                        {
                              areatt++;
                        }
                  }
            }
            areaTT = (areatt/areaT);
            if (areaTT>0.7)
            {
                  unit.mimic.push_back(1);
            }else unit.mimic.push_back(0);
      }

}



void centertrack(IplImage* frameb)
{
      //track centre of fire... via density?(smoke central point)


      if (framecounter==0)
      {
            dynamicArray.resize(frameb->width,vector<int>(frameb-
>height));
            for(int i = 0;i < (int)dynamicArray.size();++i)
            {
                  for(int j = 0;j < (int)dynamicArray[i].size();++j)
                  {
                        dynamicArray[i][j] = 0;//set 0
                  }
            }
      }

      // for each contour pixel add one.
      //for (int k=0;k<(int)xyhw.size();k++)
      //{
      //    for(int i=gravy.xspot;i<(gravy.xspot +gravy.xlength)
;i++)
      //    {
      //          for (int j=gravy.yspot
;j<(gravy.yspot+gravy.ylength);j++)
      //          {
      //                dynamicArray[i][j]=dynamicArray[i][j]+1000;
      //          }
      //    }
      //}
      int smokecount=0;
      for(int i=0;i<(int)dynamicArray.size();i++)
            {
                  for (int j=0;j<(int)dynamicArray[i].size();j++)
                  {
                        if (summon[i][j]>0)
                        {
                              smokecount++;
                        }
                  }
            }
      fprintf(fp,"%lf\n",experimentcounter);
      experimentcounter=0;
      for(int i=0;i<(int)dynamicArray.size();i++)
      {
            for (int j=0;j<(int)dynamicArray[i].size();j++)
            {
                  if (summon[i][j]>0)
                  {
                        dynamicArray[i][j]=dynamicArray[i][j]+1000;
                        summon[i][j]=0;
                  }
            }
      }


      //see densest area of contour, point is suspected to be smoke
emitting point.
      for(int i=0;i<(int)dynamicArray.size();i++)
            {
                  for(int j=0;j<(int)dynamicArray[i].size();j++)
                  {

                        if (dynamicArray[i][j]>0)
                        {
                        dynamicArray[i][j]=7*dynamicArray[i][j]/10;
                        }
                  }

            }


      //output side
      for(int i=0;i<(int)dynamicArray.size();i++)
      {
            for(int j=0;j<(int)dynamicArray[i].size();j++)
            {
                  if (dynamicArray[i][j]>3000)//test 10 first
                  {
                        CvPoint pt ={i,j};
                        ((uchar*)(frameb->imageData + frameb-
>widthStep*pt.y))[pt.x*3] = 255;
                        ((uchar*)(frameb->imageData + frameb-
>widthStep*pt.y))[pt.x*3+1] = 0;
                        ((uchar*)(frameb->imageData + frameb-
>widthStep*pt.y))[pt.x*3+2] = 255;
                  }
            }
      }

}
/*          char imagestring[255];
            memset(imagestring,0,255);
            sprintf_s(imagestring,"C:\\Documents and
Settings\\Ng\\My Documents\\Visual Studio
2008\\Projects\\FireDetect\\FireDetect\\savedstuff\\Image_%d.jpg",va
r++);
            cvSaveImage(imagestring,img);
```

```
*/
```