UNIVERSITY OF CALIFORNIA, SAN DIEGO

Real-Time Fire Detection in Low Quality Video

A Thesis submitted in partial satisfaction of the requirements for the degree
Master of Science

in

Computer Science

by

Nicholas James True

Committee in charge:

Professor Serge Belongie, Chair
Professor David Kriegman
Professor Truong Nguyen

2010

The Thesis of Nicholas James True is approved and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____
Chair

University of California, San Diego

2010

DEDICATION

To my Mom and Dad for all of their love and support.

EPIGRAPH

"Time is the fire in which we burn." - Delmore Schwartz

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank Chris for the new box, without which I'd still be waiting for my research results! I would also like to thank Prof. Serge Belongie, Prof. David Kriegman, and Prof. Truong Nguyen for their advice and help on my thesis.

ABSTRACT OF THE THESIS

Real-Time Fire Detection in Low Quality Video

by

Nicholas James True

Master of Science in Computer Science

University of California, San Diego, 2010

Professor Serge Belongie, Chair

For over fifty years, simple smoke and heat sensors have been the primary means of automated fire detection. We are now at the point where computer processing power is cheap enough and machine vision technology is sophisticated enough for a new generation of automated fire detection systems: video-based fire detection (VBFD). While current smoke and fire detection technology has proven to be reliable and effective, VBFD technology promises to go where existing systems can't and to detect fires faster than its venerable predecessors ever could.

This thesis explores a few methods for achieving real-time video-based fire de-

tection in low quality data. Assuming a stationary source camera, we describe an algorithm that uses a support vector machine to classify short, targeted video sequences as fire/non-fire. The algorithm achieves a classification rate of 96.0% on a holdout set of real world data. Furthermore, the system is robust with respect to the distance from the fire source, works day or night, and only requires the processing power of a common desktop computer.

Chapter 1

Introduction

Every year, thousands of structure fires destroy property and hurt people in the United States (37). On the front line in the defense against fire disasters are automated fire detection systems such as smoke alarms and heat sensors. These systems work well at providing warning to unsuspecting victims, thus saving lives and reducing property damage.

However, these fire detection systems are far from perfect. Take the common smoke alarm for example. It depends on there being sufficient quantities of smoke to build up before it will go off. If the room is big or if irregular heating and cooling of different parts of the room cause the smoke to stratify [1], it can take a very long time for the smoke alarm to go off.

Heat sensors face problems similar to those of fire alarms. In large rooms it can take a long time for heat to build up enough to set off an alarm. As common sense suggests, the more time it takes to set off the alarm, the more time the fire has to grow and do damage.

A video-based fire detection (VBFD) system has none of these problems. VBFD systems can detect fire at small or large distances and they can sound the alarm in as little time as it takes to process the input video. There is no unnecessary waiting for smoke

[1] Stratification of smoke and hot gases is an uneven and often layered distribution of these fire byproducts in an enclosed space (1, 33).

or heat to build up. Furthermore, cameras are *area sensors* meaning that each sensor is designed to cover a large area efficiently. Point sensors such as smoke alarms and heat sensors need to be sprinkled all throughout an area in order to effectively cover the target region. While it is true that smoke and heat will eventually travel to a set of far flung point sensors, the extra time it takes to reach the necessary conditions for the alarms to go off means more property damage and increased risk of injury or death to any occupants of the burning structure. In other words, you need more point sensors to cover the same amount of space that area sensors can cover. Lastly, cameras can work effectively both indoors and outdoors. In outdoor settings, smoke and heat sensors are virtually useless.

So why don't people have VBFD systems monitoring their homes and businesses instead smoke alarms? The answer is a combination of economic and technological factors. Smoke alarms are very cheap and high quality cameras and processors aren't. Furthermore, the computer vision technology behind the fire detection has only been reliable with toy examples or under limited and specific circumstances. However, as processing power gets cheaper and computer-vision based fire detection algorithms get more sophisticated VBFD systems will become more prominent in the real world.

1.1 Problem

The goal of this thesis is to find a reliable and robust algorithm for detecting fire in low quality video data. In other words, say we have a regular closed circuit television (CCTV) surveillance system setup in a building, it would be nice to send the video from the CCTV cameras to a computer and have the computer monitor the video feed, turning a "blind" surveillance system into an automated fire detection network. Thus it is necessary for the fire detection algorithm to handle video with a low resolution and frame rate.

For this project, we expect that the camera generating the input video is stationary. We consider fire detection using non-stationary cameras as a completely separate

problem. Also, we restrict the problem to detecting fire using a camera which can only visualize the (human) visible light spectrum, i.e. a color video camera.

The point of this project is to visually detect fire like a human would. The reasons for focusing on detecting fire in color cameras are two-fold. The algorithm put forth by this thesis can easily be applied to existing networks of color cameras without the need for investment in additional hardware such as infrared cameras. Furthermore, the flame recognition algorithms described in this thesis can be applied to infrared cameras and yet this VBFD algorithm is not dependent on the use of such specialized hardware.

1.2 Why Is Fire Detection A Non-Trivial Problem?

At first blush, it probably seems like detecting fire with a camera should boil down to looking for orange and yellow blobs that move about. However, our research definitively shows that there are *many* visual phenomenon in real life situations which would produce false positives from such a simple heuristic.

Simply put, the difficulty in creating a machine-vision-based fire detection algorithm is that most of the common techniques for tackling vision problems simply don't apply to fire detection. For example, a significant percentage of computer vision research relies on feature point extraction, detection, and/or matching. Unfortunately, fire has a number of characteristics which effectively thwart such an approach.

First off, fire is completely deformable. This makes tracking and matching very difficult at best. Even worse than this is the fact that feature points are usually determined using a corner detector which depends on consistent changes in intensities. Fire is a semi-translucent object which means that it has no defined edges, no consistent intensity changes on it's 'surface', and no corners of any kind on it's surface.

Another common approach to solving computer vision problems is to perform some version of shape or template matching on the input data. This works fairly well for objects like doors or faces but would fail miserably on fire which has no defined shape of any kind.

While it is currently quite hard to scientifically explain how humans and animals recognize fire, intuition suggests that it is not through any of the methods mentioned so far. Sure, we humans use color to weed out non-fire colored objects, but we also use the behavior of fire and an excellent understanding of the surrounding scene to make our final decision of fire/non-fire. Computer vision research has not yet reached a point where a computer can classify everything in a scene except a possible fire object. However, in this thesis we can and do leverage fire behavior to help use create a fire detection and recognition algorithm.

1.3 Previous Work

Probably the most common first step in current VBFD algorithms is to detect fire colored objects. Common methods for achieving this are to classify pixels as being fire/non-fire using a segmented color space model (9, 18, 26, 39). Others use statistical color models such as Gaussian models for each color channel in various color spaces (35). Still others training a neural network to classify pixels based on color (23). For some of these researchers this was the one and only step performed. Invariably, the data sets used by these researchers were very specialized and usually focused on fire detection in natural and very *green* environments.

An interesting approach taken by (34, 35) is to search for pixels with the right amount of flicker from background color to fire color and back again. In (35), a discrete wavelet transform is used to distinguish the flicker transitions and if the flicker count is above a threshold, the pixel in question is labeled as being fire. In (34), a hidden Markov model is used to detect flame flicker for each pixel.

Another tactic used to detect fire is to classify candidate fire blobs based on their shape. The authors of (27) used the Fourier Descriptors to model the spatial frequency information of the target blob's shape. They then used an auto-regressive model along with a support vector machine to classify blobs based on their shape. The results reported were quite good. However, the fire blobs needed to be large to overcome edge

sampling and quantization issues.

In this thesis, we re-explore a number of the methods mentioned so far, incorporating the more successful and robust techniques into our research. We also introduce techniques that have been successful in other areas of machine vision research but have yet to find their way into the fire detection sub-community.

1.4 Thesis Structure

This thesis is organized into six chapters. Chapter 1 is the introduction. Chapter 2 provides a historical background of fire and smoke detection systems and it gives a rational for developing a robust video-based fire detection system. Chapter 3 specifies the system setup used for this research include the hardware and situational setups used to gather and classify the input video data. Chapter 4 describes the methods used to detect regions of interest in videos. Chapter 5 describes the methods used to classify regions of interest in videos. Chapter 5 also enumerates the results achieved. Chapter 6 summarizes this thesis. Chapter 6 also analyzes the results with an eye for real-world usefulness as well as opportunities for future work.

Chapter 2

The History of Fire and Smoke Detection Systems

2.1 Introduction

Fire detectors are dime a dozen. Over the years, a myriad of different technologies and methodologies have been applied to automated fire detection. Each technology offers its own set of strengths and weaknesses. This chapter focuses on describing many of the most common and successful fire detection devices and the technologies behind them. We then show why there's a need for a vision-based fire detection system.

2.2 Heat Sensors

Developed in the 1860s, heat sensors were among the first automated fire detection devices (5). Like their name implies, these devices use the heat of the surrounding environment to detect if a fire is present. Heat sensors fall into two categories.

The first type of heat sensor is the *fixed temperature* heat detector. Fixed temperature devices activate an alarm after the sensor detects that the temperature has risen above a predefined threshold which is usually 135°F (58°C) or higher (5). These devices work best when placed on a ceiling or in an area where heat from a fire can build

up sufficiently to set of the alarm.

The second type of heat sensor is the *rate of rise* heat sensor. Like their name implies, these devices activate when the surrounding temperature rises at a rate exceeding the inbuilt threshold which is usually around 12 to 15 degrees Fahrenheit (7-8°C). These devices do not work reliably in areas which experience rapid temperature changes such as kitchens, laundries, or near heaters and vents (5).

Overall, heat sensors are very reliable and don't see many false positives if used in the right locations with the proper conditions. However, they are slow to react to fires and only work well in settings with the right environmental makeup.

2.3 Smoke Detectors

Smoke detectors are a common sight in commercial and domestic buildings. There are currently a number of different smoke detector variants, each with its strengths and weaknesses.

The first type of smoke detector is the *photoelectric smoke detector*. Photoelectric smoke detectors have a light sensor and a light source in a chamber secluded from external light. The light source is blocked from view of the light sensor. However, when smoke enters the chamber, light bounces off of the smoke into the light sensor, setting off the alarm. Photoelectric smoke detectors respond quickly to smoldering fires, but take more time to detect flame-heavy fires (5).

The second type of smoke detector is the *ionizing smoke detector*. This type of smoke detector uses a small amount of radioactive material to ionize air in a chamber. The ionized air allows a small current to pass from electrodes on opposite sides of the chamber. When smoke enters the chamber, the ionization effect is reduced and the current drops, setting off the alarm (2).

A benefit of ionizing smoke alarms is that they do not experience as many false alarms as photoelectric smoke alarms. Furthermore, ionizing smoke detectors are good at detecting flame-heavy fires. However, ionizing smoke detectors are slower than pho-

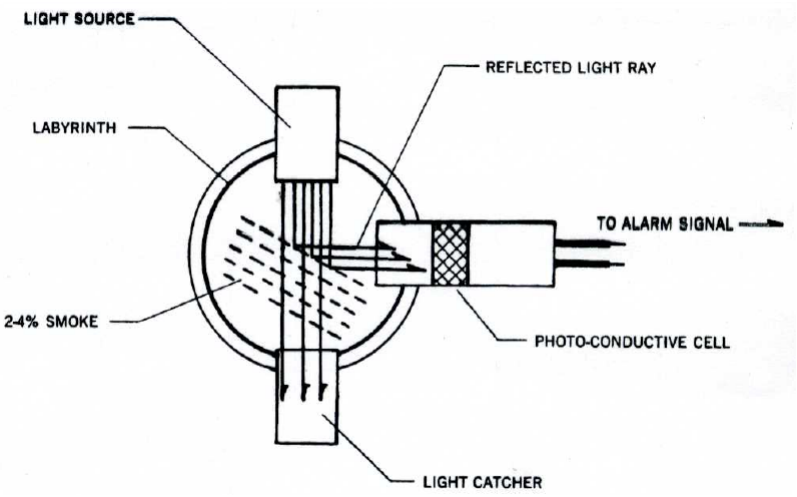

Figure 2.1: A diagram of a photoelectric smoke detector (3).

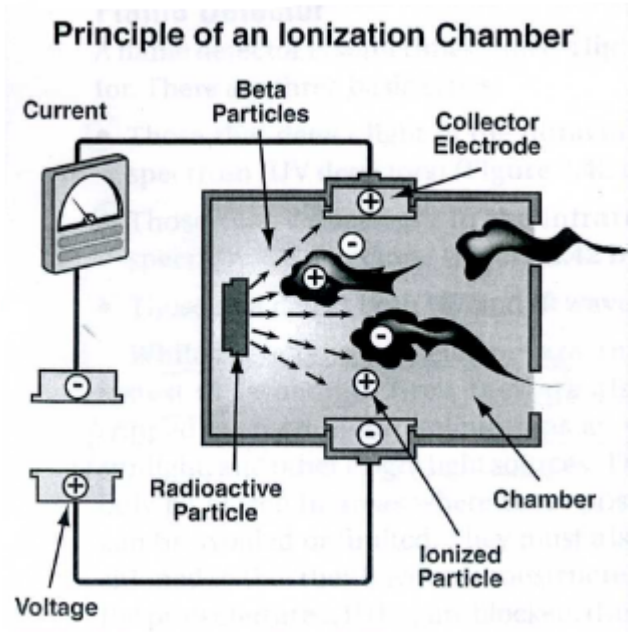

Figure 2.2: A diagram of an ionizing smoke detector (2).

toelectric smoke detectors to detect smoldering fires (5).

2.4 Point vs Line sensors

Up until this point, all of the smoke and heat devices described in this thesis have been point sensors. Point sensors simply check for the target condition at one specific location. Point sensors rely on environmental factors to bring the requisite trigger (like smoke or heat) from a fire to the sensor. Still, it is necessary to disperse multiple point sensors over region to cover an are effectively.

Another type of sensor is the *line sensor*. These sensors check for a specific condition over a long sensor line. For a heat sensor, this consists of a long tube or pipe which is sensitive to heat over its entire length. For smoke sensors, a laser beam is shot across a room or hallway to a receiver. Should the beam get blocked—hopefully by smoke and not dust or a balloon—the alarm goes off.

On the positive side, line sensors can cover more area than a single point sensor. Of course, these sensors suffer from most of the same problems that their point-sensor brethren do.

2.5 Flame-Based Fire Detection

Many of the problems and short comings suffered by the point and line sensor from the previous section can be overcome by a visually-based fire detection device. This fact has not gone unnoticed and people have tried to implement visually-based fire detectors. In fact, simple if not somewhat crude devices have existed on the market for some time now.

The first flame-detection device is the *ultraviolet-based flame detector*. These devices are set off by the presence of ultraviolet light which is commonly emitted by a fire. On the positive side, these devices can cover an *area*, not just a point, and they don't suffer from any of the false alarm issues of the previously-mentioned devices. However,

ultraviolet-based flame detectors are susceptible to detecting non-fire UV sources such as the sun. Sunlight can be hard to avoid and for this reason, ultraviolet-based fire detectors only work well in specific situations. Lastly, UV detectors don't work well when the flames are far away from the device, making UV detectors useful only in specific situations (3).

The second type of flame-detection device is the *infrared-based flame detector*. As the name suggests, infrared-based flame detectors use a camera that is sensitive to infrared radiation to detect flames. Like ultraviolet-based flame detectors, infrared-based flame detectors are among the fastest fire detection devices (2) out there but they do suffer from a rather high false alarm rate. Infrared detectors often use flicker detection to reduce the false alarm rate (2). Still, things like infrared lamps, cigarette lighters, car headlights, neon signs, and solar reflection off of water can all trigger false positives in an IR-based fire detection device (3).

2.6 The Motivation for a Robust Video-based Fire Detection System

If we analyze the various fire and smoke detection devices, two facts stand out: each technology has it's strengths and weaknesses, and one always needs specialized hardware to ensure a vigorous watch for fires. Clearly there is room for improvement.

Video-based fire detection is an obvious answer to many of the problems listed in the previous sections. VBFD devices are area sensors which means that one device can efficiently and effectively cover a large amount of space. Furthermore, most of the conditions which cause false alarms in smoke sensors, heat sensors, and flame sensors don't trigger VBFD systems. This VBFD algorithm can be "added" to existing security camera systems simply with the addition of video processing hardware. Given the ever increasing capabilities of computers and given that the cost for computers to run the VBFD system will undoubtedly be lower than the cost of installing many new fire

detection devices.

To date, fire detection devices such as smoke alarms and heat sensors have done a good job at alerting us to fires that would otherwise have hurt and killed more people and damaged more property. However, the advances in computer vision and computer hardware have come to a point where the next wave of even more effective and versatile fire detection devices will probably be computer-vision based.

Chapter 3

Setup and Data

3.1 Hardware Setup

The intent of this project was to develop fire detection algorithms specifically designed for low quality and low resolution videos based only in the visible light spectrum. While this project focused only on color videos, most of this research would still work well on gray-scale videos so long as candidate fire sequences could still be accurately detected and extracted by a modified region of interest (ROI) detection stage. The reason for focusing on low quality video is because CCTV cameras used in security surveillance systems use relatively cheap low quality cameras. Given that the most likely initial usage of VBFD systems is to augment existing security surveillance systems, it makes sense to focus on this particular hardware setup. Furthermore, the fire detection algorithm that we developed for low quality cameras should also work just as well on video from expensive, high quality cameras.

To simulate low quality CCTV video data, we used a PowerShot A540 from Cannon. All the videos created for both the training and testing sets were shot at 32 frames per second with a resolution of 320x240. All videos were shot with a completely stationary camera.

3.2 Training and Testing Sets

The training set is composed of night videos, bright day videos, dim day videos shot at varying distances and altitudes compared to the fire source. There are 28 videos in all with a variety of backgrounds with a mixture of 17 videos with fire in them and 11 videos with no fire.

The fire detection algorithm is based on detecting candidate ROIs and then classifying them as being fire or non-fire. We decided upon using 50 frame sequences to create the fire candidate ROI sequences. There are a total of 1874 training candidate ROIs at 50 frames a piece. This can be broken down to 604 positive sequences and 1270 negative sequences. The following are a few screen shots from videos with and without fire in them.

The testing set has a total of 36 videos and is composed of night videos, bright day videos, dim day videos which were shot with a variety of backgrounds and distances to the fire. The test set has 766 positive sequences and 1442 negative sequences at 50 frames apiece. Note that the set of training videos and the set of test videos share no members. Thus, the training set of 50-frame video sequences share no members with the set of test sequences.

Figure 3.1: Screen shots of training videos with fire in them.

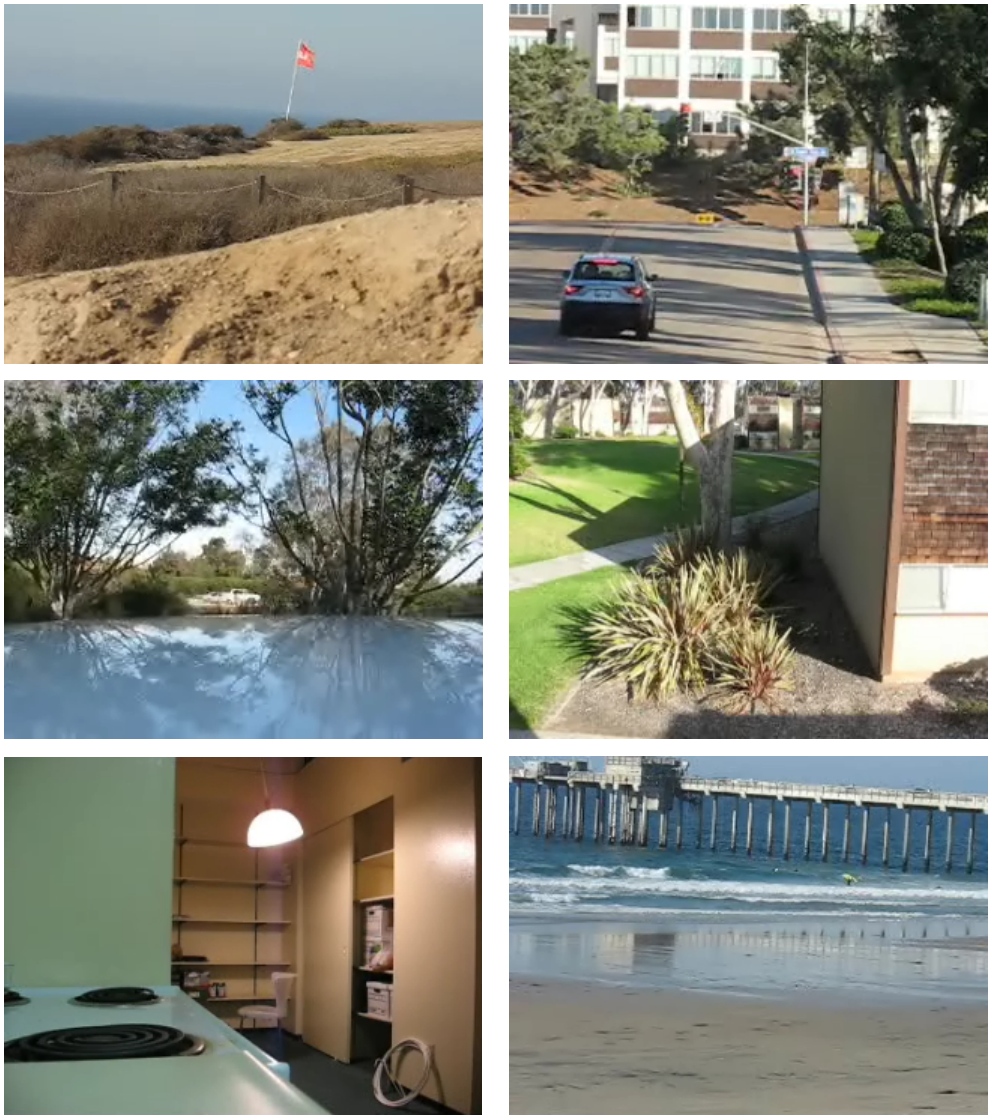

Figure 3.2: Screen shots of training videos with no fire in them.

Chapter 4

Region of Interest Detection

4.1 Introduction

If we step back for a second to examine what a computer really is, we quickly see that computers are just glorified calculators with only one innate skill: a prodigious ability to add. To us humans, an image has instant meaning because we have an incredible ability to learn, remember, and understand the light patterns hitting the backs of our retinas. We are designed to understand this information. On the other hand, a computer "understands" an image to be an array of numbers. That's it. To give a computer the ability to *see* objects in an image, an object recognition algorithm is required to locate the target items in the image.

The first step of any object recognition algorithm is to identify and extricate regions from the input image. The resulting candidate regions then need to get passed to the another algorithm which separates the target objects (true positives) from everything else (false positives). Obviously, finding these candidate regions is very important as nothing can get classified if no candidate objects are getting passed to the object classifier in the first place. We call this stage of an object recognition algorithm, the "Region of Interest (ROI) Detection" stage because it finds the candidate ROIs which are then passed to the object classifier.

Like most object recognition algorithms, the fire detection and recognition algo-

rithm described in this thesis has an ROI detection stage. To achieve good classification results, it is of the utmost importance that the ROI detection stage has the following two properties.

First it is critical that the ROI detection stage find every instance of fire that it comes across. If it misses any fire sequence ROIs, the fire candidate object classification stage won't be able to classify ROIs that it never receives and this will translate into a larger false negative rate. Ideally we want the ROI detection stage to find and pass along every fire ROI to the classification stage.

The other important property of the fire ROI detection stage is that it cannot return hybrid ROIs: sequences with candidate blobs composed of fire and non-fire objects. The fire classification algorithm separates fire sequences from all non-fire sequences.

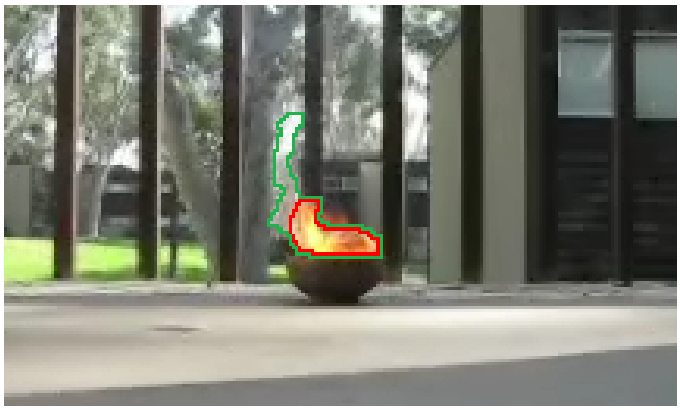

Figure 4.1: The green outline shows a hybrid fire-and-non-fire segmented output from the feature detection stage. The red outline indicates a "clean" segmented output from the feature detection stage.

Non-fire sequences may have behavior that is completely different from fire, or behavior that is fairly similar to fire. Therefore, it is critical that all fire sequences *only* be composed of flames outlines as including non-fire blobs or blob bits will confuse the classifier. To put this in perspective, human classification of the silhouettes from hybrid sequences was quite poor as well. Thus it is of of critical importance that the ROI detection stage return no hybrid sequences. The following section describes how this is

achieved.

4.2 Detecting Fire Only Using Color

So why don't we just detect fire using some sort of color detector? As we all know, fire is (usually) an orangish-yellow color which means that it stands out and therefore should be easily detectable. Unfortunately, life is never that easy and this is no exception. Take the left-hand picture from below as an example. It may appear that the fire is a distinctly different color than any of the other parts of the image. As the right hand picture tries to show, fire color is not unique or distinct as both fire and non-fire objects often share the same color in the same image. Furthermore, the color

Figure 4.2: Fire color is not unique to fire! The left-hand picture is the original and the right-hand picture shows a fire pixel and a non-fire pixel have the same color.

of fire changes drastically depending on the amount of ambient light and the intensity of light sources in the scene. Plus, there is no way to reasonably guarantee the colors in the background. Even if there's no fire color now, an orange squirrel could bound into the scene and generate quite a few false positives in a color-only fire detection system.

What if we were to add a motion component to color classification? Could we

detect fire using color and the presence of motion alone to detect fire? Could we get a good classification rate simply by classifying all fire-colored, moving regions in a video as being fire?

To answer this, we created a program that would locate all 50-frame sequences with a moving, fire-colored object in it. These sequences were then hand classified as being fire or non-fire. This was done using a perceptron-based color classifiers trained in the RGB, HSV, and $L^*a^*b^*$ color spaces. The side effect of this method is that the sets of sequences from the three color spaces are not the same, even though the videos from which the sequences were extracted are the same. This makes the comparison of the effects of color spaces on a color and motion classifier inexact but it makes the creation of fire-candidate sequence[1] sets for the purposes of training and testing *much* more manageable.

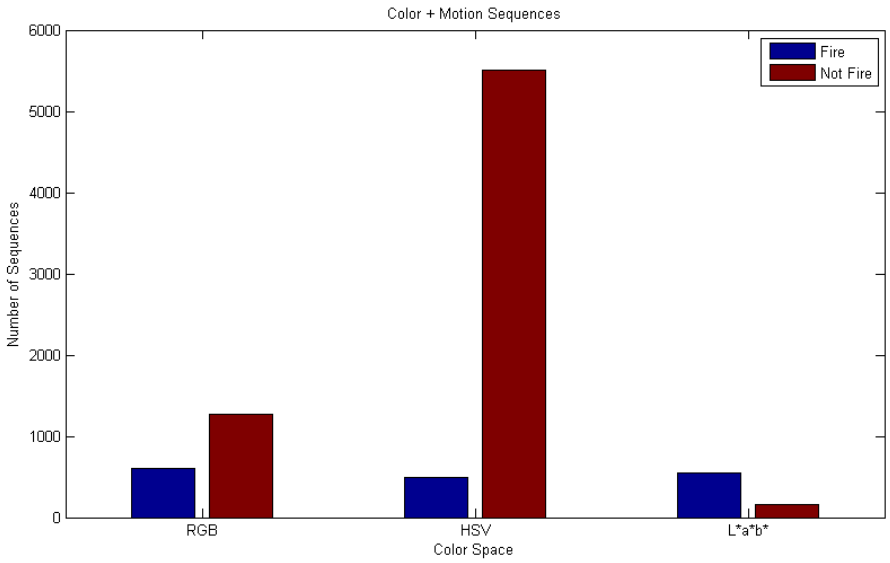

Figure 4.3: The number of candidate sequences using a neural network trained on RGB, HSV, L*a*b* input data. Clearly, classifying fire-colored moving objects as being fire will return poor results in any of these color spaces.

[1] A *fire-candidate sequence is a 50-frame sequence which has a fire-colored moving object in it.*

After examining figure 4.2 we can clearly see that there are too many fire-colored moving objects to use the existence of fire color, motion, or both to detect fire. Color and motion detection is only a good first step in the fire detection process. Additional discriminative stages are required to complete the classification.

4.3 Motion Detection

4.3.1 Common Motion Detection Algorithms

The goal of motion a detection algorithm is to notice and locate objects that have moved in the input scene. Most of the time, performance is of critical importance. This means that we can't spend the processor cycles trying to recognize and track objects in the input video. We need to detect motion without understanding what types of objects are in the scene. As you might imagine, motion detection is quite challenging when we can't distinguish objects in the target scene. The follow section describes different motion detection algorithms and discusses some of the strengths and weaknesses of these algorithms.

The simplest and probably the fast method for motion detection is frame differencing. This idea assumes that the lighting of different objects remains the same as time changes. Given this lighting assumption, the non-zero difference of two successive frames indicates where motion has occurred. Of course, shadows, reflectance, and changes in light source locations can cause the lighting assumption to be wrong and severely distort the resulting motion map.

A slightly more sophisticated version of frame differencing is median filtering. This is a commonly used technique in motion detection and general background subtraction. For this algorithm, the background image is approximated by creating the time-averaged median image which is then subtracted from successive images in the video sequence to determine object motion (29, 21, 14, 15). This method has the advantage that it helps to remove or reduce minute lighting changes that can create false

positives in the motion detection results.

Another method described in (19, 13) for estimating the background image is to recursively compute a weighted combination of the background image and the current frame.

$$\beta_{t+1} = \alpha\beta_t + (1 - \alpha)\gamma_t$$

Simply put, a background pixel β at time $t+1$ is a combination of the current background pixel and the pixel γ from the current frame. Note that α controls the combination of background pixel and the current pixel. This method has the advantage of being simple and fast.

A background estimation method (which can easily be turned into a motion detection algorithm) that has become quite popular is to use mixtures of Gaussian (MoG) to model the color probability of each pixel in an image. First proposed for background modeling in (20), the MoG method models a pixel distribution as

$$f(I_t = u) = \sum_{t=1}^{K} w_{i,t} \cdot v(u; \mu_{i,t}, \sigma_{i,t})$$

where K is the number of Gaussian$v(u; \mu_{i,t}, \sigma_{i,t})$ is Gaussian i, and $\mu_{i,t}$ and $\sigma_{i,t}$ are the intensity mean and standard deviations respectively (11).

There are even more background estimation and motion detection algorithms out there. However, these methods often make certain assumptions about the scene in question or the target object(s) that the algorithm is trying to detect movement from. Invariably these methods don't translate well to detecting the flicker of flames.

4.3.2 The Motion Detection Algorithm

There are two key concerns when deciding upon which motion detection algorithm to use to find moving objects that might be fire.

The first concern is that the algorithm should be as fast as possible since the input into the fire detection system is a real-time video stream; any delays in any stage of the fire detection algorithm ultimately slows down the response time of the overall alarm.

The second concern is that we need to make sure that regions of interest that actually contain flames should fully contain those flames. If the motion detection algorithm is insensitive to the occasional flame flicker at the ends of fire, these edges may get cut off. This in turn would distort the behavior of the candidate region which could distort the results of the classification stage.

The algorithm that fit this fire detection problem best in terms of combined accuracy and efficiency was a modified version of frame differencing. The algorithm is as follows

Algorithm 1 Fire Candidate Motion Detection

Require: F {a sequence of K binarized movie frames}

Require: S {a 2D array of pixel motion counts with the same dimensions as F}

$S \leftarrow zero(S)$ {set all of the counts to zero}

for i=1:K **do**

S $\leftarrow |F_i - F_{i-1}|$

end for

motion_mask $\leftarrow threshold(S)$

return motion_mask

4.4 Color Classification

4.4.1 Color Classification of Fire

Color is one of the primary features humans use to distinguish fire from non-fire objects and other visual phenomena. While color alone is insufficient to completely detect and classify objects as being fire/non-fire, it is a good feature for filtering out obviously non-fire objects. This section discusses different color classification algorithm that have been used in various fire detection algorithms.

In the simplest class of color classification algorithms are rule-based classifiers. These methods usually involve human-determined rules and color channel thresholds

Figure 4.4: An example of fire color classification. The top row are the original images. The bright red color in the bottom row indicates where pixels have been classified as being fire colored.

for classifying pixels. A good example of this method is the work done in (6) where the authors used the following rule set for classifying color using the $YCbCr$ color model

$$f(x) = \begin{cases} 1 & \text{if } Y(x,y) > Y_{mean}, Cb(x,y) < Cb_{mean}, Cr(x,y) > Cr_{mean} \\ 0 & \text{otherwise} \end{cases}$$

With this method, the authors reported achieving a detection rate on fire data of 0.990 with a false alarm rate of 0.315 (6). These results are pretty good for a rule-of-thumb algorithm and they can be achieved with a bare minimum of computational effort. The extreme efficiency of this method makes it a perfect algorithm for low-powered devices and as a first pass filtering step in an algorithm with a slower but more accurate (lower false positive rate) second stage.

In (39), a similar approach to the last method was used except instead of using the $YCbCr$ color space, the HSV color space was used. In (10) a rule-based color thresholding algorithm was used in conjunction with the RGB color space. And in (18), the same style thresholding algorithm was used based on the RGB and HSV color spaces.

A completely different approach to fire color classification is to build a fire color probability map and use it to determine whether a pixel is fire or not. This is the method used in (24). More precisely, a sequence of n color values is accumulated for a specific pixel in the target image and if the average probability of that sequence is greater than an experimentally determined constant, the pixel is classified as being fire colored. This operation is performed for each pixel in an image until the whole image has been labeled. Just like the last class of fire color classification algorithms, this one has the advantage of being computationally light weight.

The one downside of this algorithm is that it doesn't classify each pixel in an single frame as being fire or non fire. This algorithm classifies a pixel as being fire or non fire on average over a *sequence* of frames. Because of this feature, the algorithm does not work well with the methodologies put forth in this thesis because these methodologies require that each *frame's* pixels be labeled fire or non fire.

A more sophisticated approach to fire color classification is to use a neural network as was done in (23). This method is a little slower to compute than the previous methods but it has the advantage of getting excellent results. Because neural-network-based color classifiers achieve good results, we decided to use one in our fire detection algorithm. The following is a cursory introduction to neural networks.

4.4.2 Neural Networks

A neural network, or perceptron[2] as it is sometimes known, is a supervised learning algorithm that is very effective at approximating mathematical functions which have the form

$$f : \mathbb{R}^n \to \mathbb{R}^m$$

$$\mathbf{x} \mapsto \mathbf{y} = f(\mathbf{x})$$

where $\mathbf{x} = (x_1, x_2, ..., x_n)^T$ and $\mathbf{y} = (y_1, y_2, ..., y_n)^T$ (22). Put more specifically, it has been proven that "given any $\epsilon > 0$ and any Riemann-integrable function $f : [0,1]^n \subset \mathbb{R}^n \to \mathbb{R}^m$, there exists a perceptron $P(\mathbf{x}, M, \mathbf{w})$ such that"

$$\int_{[0,1]^n} |f(\mathbf{x}) - P(\mathbf{x}, M, \mathbf{w})|^2 d\mathbf{x} < \epsilon$$

where **x** is an input vector, M is the number of hidden nodes, and **w** is the weight vector (22). In other words, for most functions used to model the real world, we can train a neural network to approximate the function to an arbitrary degree of accuracy where accuracy is defined as the mean squared error (MSE).

So far, we have seen that it is possible to approximate Riemann-integrable functions using a perceptron. However, to use the perceptron as a classifier, it needs to work on real world data and real world data is rife with noise. In fact, we find that more often than not there is no known mathematical function that precisely models data from any number of real world phenomena. How then can we use a perceptron as a classifier?

[2]Note, some authors label single layer neural networks as *perceptrons* and multilayer neural networks as *multilayer perceptrons*. No such distinguishing is made in this thesis.

The answer is to treat real world data as being the sum of a mathematical function and a noise factor (22). Therefore, the function that neural networks try to learn is a regression model of the output values given particular inputs (38, 22).

Up to this point we've briefly discussed the capabilities of neural networks and hinted at why they make excellent classifiers. Logically speaking, the next topics that should get covered are the perceptron learning and classification algorithms. Before we delve into the nitty gritty of each of these algorithms, we should mention that there are many, *many* perceptron variants. The variant used in this project is a multi-layer perceptron, specifically a two layer perceptron. Given n inputs, M hidden nodes, and m outputs and output nodes, the following is the formula for computing the output of

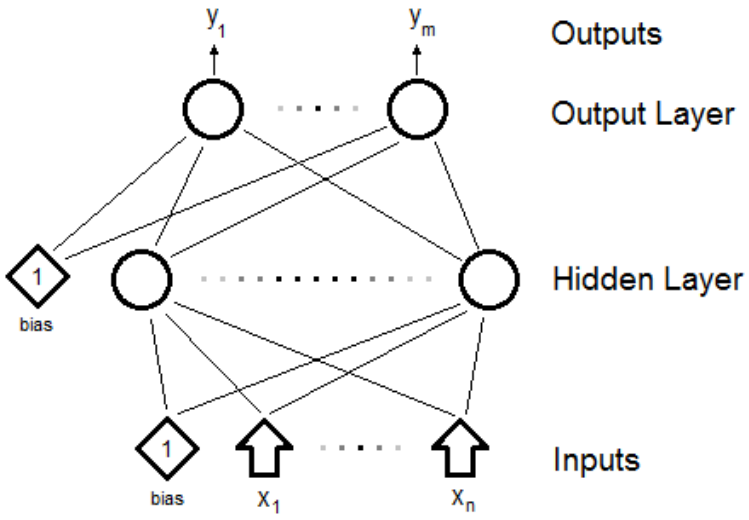

Figure 4.5: A two-layer perceptron. The input is $\mathbf{x} = (x_1, x_2, ..., x_n)^T$ and the output is $\mathbf{y} = (y_1, y_2, ..., y_m)^T$. There are M hidden nodes, m output nodes, and there are two bias values set to 1.0 at the input and hidden layers (the triangles). These bias inputs are *crucial* for the perceptron to work properly.

hidden node z_j:

$$z_j = \tanh\left(\sum_{k=0}^{n} u_{jk} x_k\right)$$

(22) where $j = 1, ..., M$ and u_{jk} is the weight for hidden node j on input k. The values computed by the hidden nodes are then used to compute the output of the output nodes. The output nodes perform their calculations as follows

$$y_i = \tanh\left(\sum_{j=0}^{M} v_{ij} z_j\right)$$

(22) where v_{ij} is the weight for the output node i with input from hidden node j. It should be noted that input x_0 is not actually an input but a bias value 1.0 that is internal to the perceptron algorithm. The same is true for the hidden node z_0.

Before you can use a perceptron as a classifier, you need to train the classifier. In this training stage, the hidden node and output node weight values are adjusted to reduce the MSE of the perceptron output vs the expected output. But before you can start the training process, you need to decide upon two parameter values: the number of hidden nodes in the perceptron and the learning rate α.

The number of hidden nodes determines the discriminative power of the perceptron. Too few hidden nodes and the perceptron won't be able to classify data from certain functions. Too many hidden nodes makes the training process slow. The reason for all this is because a "perceptron has an effective number of degrees of approximating freedom equal roughly to the product of the number o hidden and output layer weights" (22).

The second parameter that controls the perceptron learning is the learning rate α. The perceptron learning functions use hill climbing methods to adjust the perceptron node weight values (22). The α parameter controls the learning rate—how far each weight is adjusted per training epoch—in the learning process.

When we get right down to it, perceptron learning is just an integrative process of adjusting the weight values. The weight adjustment functions are as follows

$$v_{pq}^{\text{new}} = v_{pq}^{\text{old}} + 2\alpha[y_{kp} - P_p(x_k, M, w^{\text{old}})]z_q(x_k, u^{\text{old}})$$

is the equation for updating the output node weights, (where $p = 1, 2, ..., m$; $q = 0, 1, 2, ..., M$)

$$u_{qr}^{\text{new}} = u_{qr}^{\text{old}} + 2\alpha \left(\sum_{j=1}^{m} [y_{jk} - P_j(x_k, M, w^{\text{old}})] v_{jq}^{\text{old}} \right) [1 - z_q^2(x_k, u^{\text{old}})] x_{kr}$$

is the equation for updating the hidden node weights, (where $q = 1, 2, ..., M$; $r = 0, 1, 2, ..., n$) (22).

Neural networks are powerful learning algorithms which can be utilized to solve problems in a myriad of arenas. For problems dealing with the approximation of high input, high output functions, ($f : \mathbb{R}^n \to \mathbb{R}^m$ where $n >> 1$ and $m >> 1$), neural networks are considered by some researchers to be the only viable option (22). On the other hand, neural networks have the disadvantage of being particularly difficult to train.

The first issue that often complicates perceptron training is the fact that perceptrons are subject to over fitting the training data. It was mentioned earlier that a perceptrons computational power stems from how many hidden nodes it has. Too few hidden nodes and the perceptron won't have the discriminative power to accurately classifying the target data. However, too many hidden nodes combined with too many training epochs and the perceptron will effectively memorize the training data (31). This commonly described as *over fitting* the data. After over fitting the training data, the perceptron would get terrible results on the test sets because the testing data wouldn't look exactly like the training data.

The second weakness of perceptrons is the fact that they are susceptible to getting stuck in local minima and never actually reach the global minima. Given that perceptron learning utilizes gradient descent to update its node weights and given that gradient descent optimization suffers from problems with local minima, it is no surprise that perceptrons do too. That said, perceptrons are extremely useful so long as sufficient work is put into running the training stage.

4.4.3 Color Classifier Training and Results

The training of the color-classifying neural network can be broken down into two stages: the data gathering and preparation stage, and the parameter adjustment and training stage. It should be noted that both stages were repeated to get perceptron weight values tuned to bright daytime lighting, dark daytime lighting, and nighttime lighting. This breakdown is critical for the overall accuracy of the fire detection algorithm given that we ended up using the RGB color space. The RGB color model gave use better results at segmenting the training videos than the HSV or $L^*a^*b^*$ color models. Undoubtedly, there is a lot of room to create a one size fits all fire color classification algorithm which achieves a very high true positive rate and a low false positive rate. The creation of multiple perceptrons based on lighting conditions is a weakness of this fire detection algorithm but it allows for this concept demonstration.

To create the perceptron training values, images were captured from the fire training videos. Candidate fire and non-fire pixels were human-labeled, extracted, and reduced to a set of unique RGB color values. A training set and a testing set were created from a equal-sized sets of positive and negative color values. When one set was smaller than the other, the smaller set would have its size increased with copies of values already in the set.

Using a training rate of $\alpha = .01$ and 1000 epochs, the following perceptrons were trained. The perceptron weight values created by the training process where used for fire color classification in the rest of this thesis.

The training and testing sets were created by hand marking fire and non fire regions in a set of images. The individual pixel values from each of the marked regions were then converted into the proper values for the target color space. Finally, each "bag" of pixel values was filtered so that the result was a true set where each element in the set is a unique pixel value. This is how the *Positive (unique)* and *Negative (unique)* sets were created. It should be noted that set of pixel values for one color space may be a different size from another color space because of how different colors are represented

Table 4.1: This table shows the perceptron training results for night videos.

Night Lighting				
		RGB	HSV	$L^*a^*b^*$
Hidden Nodes: 1	Training error	7.7×10^{-4}	1.4×10^{-3}	2.0×10^{-3}
	Training set size	125,035	116,335	46,894
	Testing set size	31,258	29,083	11,723
	Positives (unique)	918	908	170
	Negatives (unique)	78,262	72,778	29,037
Hidden Nodes: 10	Training error	5.6×10^{-4}	1.2×10^{-3}	1.8×10^{-3}
	Training set size	125,035	116,335	46,894
	Testing set size	31,258	29,083	11,723
	Positives (unique)	918	908	170
	Negatives (unique)	78,262	72,778	29,037

Table 4.2: This table shows the perceptron training results for bright day videos.

Bright Daylight Lighting				
		RGB	HSV	$L^*a^*b^*$
Hidden Nodes: 1	Training error	4.6×10^{-5}	1.9×10^{-5}	2.6×10^{-4}
	Training set size	32,500	29,781	14,797
	Testing set size	8,125	7,445	3,669
	Positives (unique)	1,704	1,404	1,002
	Negatives (unique)	20,177	18,974	9,478
Hidden Nodes: 10	Training error	5.1×10^{-5}	1.2×10^{-3}	2.1×10^{-4}
	Training set size	32,500	29,781	14,797
	Testing set size	8,125	7,445	3,669
	Positives (unique)	1,704	1,404	1,002
	Negatives (unique)	20,177	18,974	9,478

Table 4.3: This table shows the perceptron training results for dim day videos.

Dim Daylight Lighting				
		RGB	HSV	$L^*a^*b^*$
Hidden Nodes: 1	Training error	4.7×10^{-2}	3.3×10^{-2}	3.6×10^{-2}
	Training set size	15,998	17,960	8,613
	Testing set size	3,999	4,489	2,153
	Positives (unique)	4630	2993	1906
	Negatives (unique)	10,737	10,477	5,048
Hidden Nodes: 10	Training error	2.0×10^{-2}	2.3×10^{-2}	2.5×10^{-2}
	Training set size	15,998	17,960	8,613
	Testing set size	3,999	4,489	2,153
	Positives (unique)	4630	2993	1906
	Negatives (unique)	10,737	10,477	5,048

in each color space.

From the *Positive (unique)* set and the *Negative (unique)* set, a new set with equal numbers of positive (fire) and negative (non-fire) pixel values was created by duplicating elements from the smaller of the *Positive (unique)* and *Negative (unique)* sets until both positive and negative were of equal size and were then added together.

The actual training and test sets were created by dividing the newly created set using a ratio of 80% training and 20% testing. These sets were then used by the perceptron training algorithm to determine the color classifier weight values for each of the given color spaces. Lastly, the size of these sets is described in the preceding tables by the *Training set size* and *Testing set size* fields.

In the end, we decided upon using one hidden node since more hidden nodes did not increase the perceptron's discriminative power much if any and because fewer hidden nodes sped up the classification process.

Chapter 5

Region of Interest Classification

5.1 Introduction

Pretend for a moment that we have a magic black box which takes videos as input and outputs short videos which fall into two classes: non-fire object silhouetted sequences and fire silhouetted sequences. The output of the black box is always pure in the sense that fire outlines do not incorporate any non-fire outlines and non-fire outlines do not contain any flame outlines in them. If we were given such a black box, all we would need to do to detect fire is to create an algorithm which could differentiate between the two output types based solely on the behavior of the objects outlined in the video sequences.

Of course, no magic black box exists. However, we do have a pretty good approximation of the black box: the fire motion and color ROI generating algorithm. That algorithm is designed to take a video as input and output short sequences outlining what it thinks are flickering flames. So long as a fire isn't burning in front of a fire-colored backdrop, the algorithm will return nearly pure output with almost no fire + non-fire hybrids. This next section discusses ways of classifying regions of interest based only on the outlined object's behavior.

5.2 Dynamic Texture Recognition

5.2.1 Introduction

A classic way for approaching object recognition and tracking in video data is to detect features like edges and interest points in consecutive frames, then exploit some sort of spatial or temporal relationship between those features to make the final class determination. These methods are well suited for recognizing objects like cars and buildings. However, phenomena like smoke, ocean waves, and rustling leaves are difficult to detect and classify using the aforementioned classification methods. Intuition suggests that to detect this class of phenomena one should focus on the dynamics of the input video sequence and the target objects in that sequences. In other words, maybe we should focus on detecting and differentiating between the way that leaves rustle and smoke billows. This is exactly what object recognition using dynamic texture analysis tries to do.

The idea behind dynamic textures is that videos of certain moving objects can be thought of as a sequence of images which are the output of a stochastic dynamic model (32). The trick, it would seem, is to map videos to models and then use some sort of distance metric to compare models for the purpose of classifying the associated videos. Unfortunately, there are two challenges which need to be overcome.

> First, the map from a sequence to a model is not necessarily one-to-one: very different scenes can be [the] output of the same model. Second, even the simplest linear models learned from data represent equivalence classes of statistics: the same scene can result in very different models depending on the initial conditions (32).

It turns out that there isn't just one model, but an equivalence class of models from which the video sequence could have been generated from. Therefore, we must choose a representative, or *canonical model realization*, to "describe" the given video sequence and allow for comparisons of different videos (16, 32, 17). The following section goes over the formal definition of the dynamic texture models, the algorithms used to classify those models, and the distance metric used by the dynamic texture classifier.

5.2.2 Formal Definition of Dynamic Textures

Say we're given a video made up of a sequence of noisy images $\{Y(t)\}_t = 1...\tau$. A noisy image y_t is defined to be $y(t) = I(t) + w(t)$ where $I(t) \in \mathbb{R}^m$ is an ideal, non-noisy image and $w(t) \in \mathbb{R}^m$ (16). This means that the input video can be "represented as a time-evolving state process $x_t \in \mathbb{R}^n$, and the appearance of the frame $y_t \in \mathbb{R}^m$ is a linear function of the current state vector with some observation noise" (8). The formal definition is as follows

$$\begin{cases} x_t = Ax_{t-1} + v_t \\ y_t = Cx_t + w_t \end{cases}$$

where we assume that there exists a positive integer n and a process $\{x(t)\}, x(t) \in \mathbb{R}^n$ with initial condition $x_0 \in \mathbb{R}^n$ and symmetric positive-definite matrices $Q \in \mathbb{R}^{n\times n}$ and $R \in \mathbb{R}^{m\times m}$ (16). Furthermore, $A \in \mathbb{R}^{n\times n}$ is the state-transition matrix, $C \in \mathbb{R}^{m\times n}$ is the observations matrix such that $v_t \sim_{iid} \mathcal{N}(0, Q)$, $w_t \sim_{iid} \mathcal{N}(0, R)$, and

$$\begin{bmatrix} Q & S \\ S^T & R \end{bmatrix} \geq 0$$

is a positive semi-definite matrix where $S = \mathrm{E}[w_t v_t^T]$ (8, 32).

5.2.3 Finding the Model Parameters

Given a sequence of images $y_{t=1,...,\tau}$ we want to compute the maximum likelihood solution

$$\hat{A}, \hat{C}, \hat{Q}, \hat{R} = \underset{A,C,Q,R}{\operatorname{argmax}} P(y_1, ..., y_\tau | A, C, Q)$$

(16, 32). Unfortunately, current computer simply don't have the power to compute this equation in a reasonable amount of time. Fortunately there is a suboptimal closed form solution which approximates the optimal equation in a reasonable amount of computational time (16, 17, 32).

Let $Y^\tau = [y_1, ..., y_\tau] \in \mathbb{R}^{m \times \tau}$. Now compute the singular value decomposition (SVD) of Y^τ to get $Y^\tau = U\Sigma V^T$. From this we can get the following

$$\hat{C} = U,\ \hat{X} = \Sigma V^T$$

To find $\hat{A}$ w need to solve the following problem

$$\hat{A}(\tau) = \underset{A}{\operatorname{argmin}} ||X_2^\tau - AX_1^\tau||_{Frobenius}$$

where $X_2^\tau \doteq [x(2), ..., x(\tau)] \in \mathbb{R}^{n \times \tau}$. This can can be done in closed form as follows

$$\hat{A} = \Sigma V^T D_1 V (V^T D_2 V)^{-1} \Sigma^{-1}$$

where

$$D_1 = \begin{bmatrix} 0 & 0 \\ I_{\tau-1} & 0 \end{bmatrix},\ D_2 = \begin{bmatrix} I_{\tau-1} & 0 \\ 0 & 0 \end{bmatrix},$$

(16). Thus, using the previous equations, one can compute the parameters of the dynamic texture model $M = \{\hat{A}, \hat{C}\}$.

5.2.4 Dynamic Texture Classification

So far, we've described how to represent video data as a dynamic texture $M = \{\hat{A}, \hat{C}\}$. Now we need some way to classify dynamic textures. We can use a nearest neighbor classifier to classify dynamic textures but to do so we need to define the distance metric used by the nearest neighbor classifier to compare the dynamic textures.

The next step is to recognize different dynamic texture models via a nearest neighbor classifier which is utilizing Martin Distance (28) as the means to compare dynamic texture models.

The intuition behind Martin Distance is to compare the subspace angles between extended observability matrices of two dynamic textures (12). Put more formally, let $M_a = \{A_a, C_a\}$ be the first dynamic texture model and let $M_b = \{A_b, C_b\}$ be the second

dynamic texture model. The square of the Martin Distance between the two models is defined to be

$$d^2(M_a, M_b) = -\log \prod_{i=a}^{n} \cos^b \theta_i$$

where θ_i are the angles between the extended observability matrices of M_a and M_b (12).

The real question is how does one compute θ_i? First let's define the extended observability matrices of M_a and M_b to be

$$\mathcal{O}_i = \begin{bmatrix} C_i \\ A_i^T C_i \\ \vdots \\ (A_i^T)^n C_i \\ \vdots \end{bmatrix}$$

where $i = \{a, b\}$ and $\mathcal{O}_i \in \mathbb{R}^{\infty \times n}$ (12). It turns out that $\cos \theta_i$ for i=1, ..., n are just the n largest eigenvalues found by solving the generalized eigenvalue problem

$$\begin{bmatrix} 0 & \mathcal{O}_{ab} \\ \mathcal{O}_{ab}^T & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \lambda \begin{bmatrix} \mathcal{O}_{aa} & 0 \\ 0 & \mathcal{O}_{bb}^T \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

object to $x^T \mathcal{O}_{aa} x = 1$ and $y^T \mathcal{O}_{bb} y = 1$ where

$$\mathcal{O}_{pq} = \mathcal{O}_p^T \mathcal{O}_q = \sum_{j=0}^{\infty} (C_p A_p^j)^T C_q A_q^j$$

for some dynamic texture models p and q (12). At this stage it is handy to note that infinite sums of the form

$$X = \sum_{j=0}^{\infty} A^j Q (A^j)^T$$

can also be written as

$$X_j = A X_{j-1} A + Q^T$$

where in steady state the equation becomes

$$X = A X A + Q^T$$

But this is just the *Lyapunov equation* and can be solved in Matlab using the function `dlyap.m` (7).

Thus, all one has to do to compute the Martin Distance between M_a and M_b is to compute the extended observability matrices, solve the generalized eigenvalue problem, take the largest n eigenvalues, and compute the Martin Distance. With this distance metric, a nearest neighbor algorithm can be used to classify dynamic texture models which by proxy classifies videos as fire/non-fire.

5.2.5 Results

Using 50 frame sequences, we were able to achieve an overall classification rate of 51.5% using dynamic texture classification. Clearly, these results are not what we'd hoped for. While there are further improvements, such as kernel-PCA-based dynamic texture classification (8), that we could have tried there was one impassible roadblock that stop this line of research: classifier run time.

Table 5.1: Dynamic texture test results from a nearest neighbor classifier using Martin Distance.

True Positive	False Positive	True Negative	False Negative	Classification Rate
718	1011	356	2	51.5%

With a training set of 564 positive dynamic textures and 1211 negative dynamic textures the nearest neighbor classifier took roughly 1,100 seconds to classify *one* dynamic texture in Matlab using a single core from a 3 GHz quad-core AMD Phenom™ 945 processor and 4GB of RAM! This is "slightly" slower than necessary for a real-time VBFD system. Clearly there has to be a better way to classify fire videos in real time.

5.3 SVM-based Fire Detection

5.3.1 Introduction

Fire lacks many of the features (interest points, rigid shape, etc.) that are commonly leveraged in popular object recognition algorithms. The application of dynamic texture recognition to the detection and recognition of fire in videos is a logical research course to pursue. Unfortunately, the results achieved by the dynamic texture classifier leaves much to be desired. Furthermore, the dynamic texture classifier is far too slow in its current configuration—i.e. using a nearest neighbor classifier—to be applied to a real-time fire detection system. A different approach is required to solve the fire detection problem in a reasonable and robust way. This is where support vector machines come to the rescue.

5.3.2 Support Vector Machines

A support vector machine (SVM) is a another type of supervised learning algorithm that can be trained for data classification or regression. Just like neural networks, SVMs can model complex functions with many inputs and are well adapted for binary classification problems, like fire detection. Unlike, neural networks, SVMs are easy to train making them quite popular in machine learning and vision research.

The intuition behind support vector machines is as follows. Say we have have a set of data living in a two dimensional world and belonging to two different classes. As the example below shows, a line separating the two classes would classify the data. Obviously, there are an infinite number of lines that separate the data but which one would be best? Support vector machines work under the assumption that the best separating line is the separating line with the largest distance or *margin* from the closest positive and negative example points. So how does one find the maximum-margin hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$ depicted in the right-hand image? First let's define a few things. Let $\mathbf{w}$ be a normal line to the separating hyperplane which means that $|b|/||\mathbf{w}||$ is the perpendicular

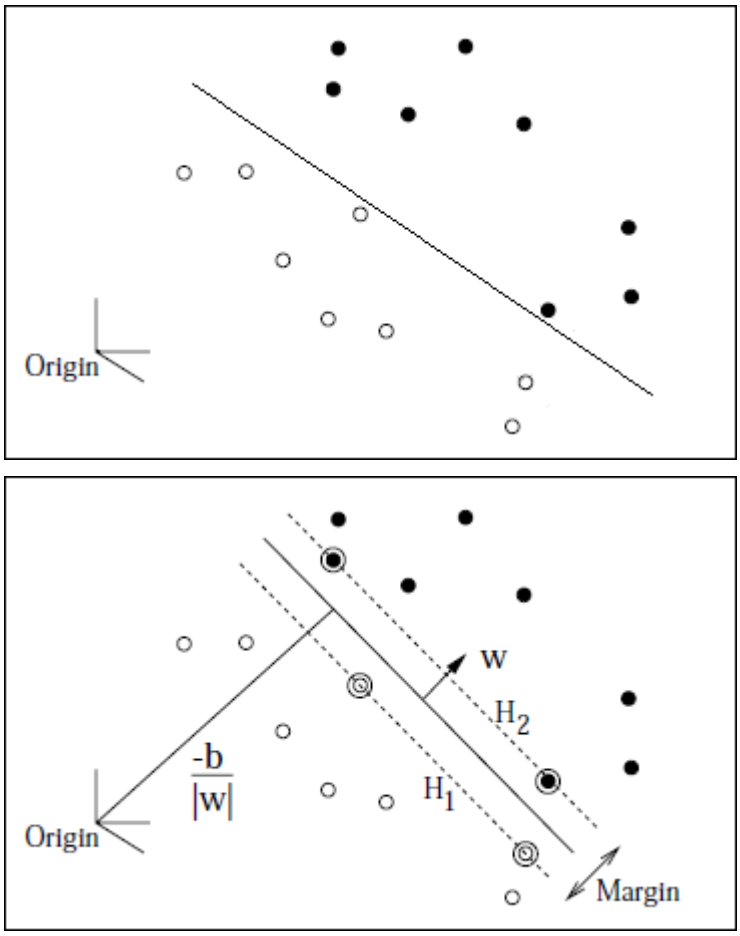

Figure 5.1: The top picture shows two clusters being separated by a randomly chosen hyperplane. The bottom picture shows the hyperplane with the largest margin (4).

distance from the origin to the separating hyperplane (4). Training a SVM basically boils down to finding a **w** and b such that the following inequalities hold

$$x_i \cdot \mathbf{w} + b \geq +1 \qquad \text{for } y_i = +1$$
$$x_i \cdot \mathbf{w} + b \geq -1 \qquad \text{for } y_i = -1$$

which can be combined into

$$y_i(x_i \cdot \mathbf{w} + b) - 1 \geq 0 \qquad \forall i \tag{5.1}$$

(4, 36). The margin is defined by the two hyperplanes H_1 and H_2 which are perpendicular to and equidistant from the separating hyperplane. They are defined as

$$x_i \cdot \mathbf{w} + b = +1 \qquad \text{for } H_1$$
$$x_i \cdot \mathbf{w} + b = -1 \qquad \text{for } H_2$$

(36). If d_+ and d_- are the distances from the separating hyperplane to H_1 and H_2 respectively and $d_+ = d_- = 1/||\mathbf{w}||$ then the margin is defined to be $2/||\mathbf{w}||$ (4). To find the maximum margin hyperplane, we need to minimize $||\mathbf{w}||^2$ subject to the constraints from 5.1 (4). This can be done using quadratic programming techniques. By introducing Lagrange multipliers α_i where $i = 1, ..., l$ for each constraint from 5.1, we get the following Lagrangian

$$L_P \equiv \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{l} \alpha_i y_i (x_i \cdot \mathbf{w} + b) + \sum_{i=1}^{l} \alpha i \tag{5.2}$$

(4). It is required that the derivatives of L_P with respect to each α_i disappear which gives us the following conditions

$$\mathbf{w} = \sum_i \alpha_i y_i x_i \tag{5.3}$$

$$0 = \sum_i \alpha_i y_i \tag{5.4}$$

(4). Substituting constraints 5.3 and 5.4 into 5.2 gives us

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

which we then try to maximize subject to the constraints $\alpha_i \geq 0$ and $\sum_i \alpha_i y_i = 0$ (4, 31). Thankfully, L_D has a single global maximum which can be located efficiently.

The two-class case we just used to introduce support vector machines was a two dimensional, linearly-separable problem which makes it fairly trivial to classify. Obviously, many problems won't have this simple form or this low dimensionality. So where do SVMs go from here? It turns out that in many cases where the data is inseparable in the input space, if we were to map the data into a higher dimensional space the data becomes separable by a hyperplane. Put more precisely, given a set of N data points, we can map these data points to an $N - 1$ dimensional space (in most cases) (31).

So does one map data points into a higher dimension such that we can then separate the points by class? It turns out that the mapping is accomplished by what is known as the "the kernel trick". The kernel trick is simply to replace the $\mathbf{x}_i \cdot \mathbf{x}_j$ term with some kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$. This is an important step because according to Mercer's theorem, any kernel where $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ is a positive definite matrix corresponds to some feature space (30). This is how we map the data into a higher dimension. Furthermore, if you look back at 5.5 you'll notice that "data enter[s] the expression only in the form of dot products of pairs of points" (31), i.e. the $\mathbf{x}_i \cdot \mathbf{x}_j$ term. This is important because $F(\mathbf{x}_i) \cdot F(\mathbf{x}_j)$ can be computed without first computing $F(\cdot)$ for $\mathbf{x}_i$ and $\mathbf{x}_i$ (31).

It is important to note that "a linear separator in a space of d dimensions is defined by an equation with d parameters, so we are in serious danger of over fitting the data if $d \approx N$" (31) where N is the number of data points. Consider equation 5.5 for a second. It turns out that the weights α_i associated with each data point are all zero except for the points closest to the separating hyperplane. Each of these points are said to "hold up" the hyperplane and are therefore called the support vectors. Because the number of support vectors is almost always far smaller than the number of data points,

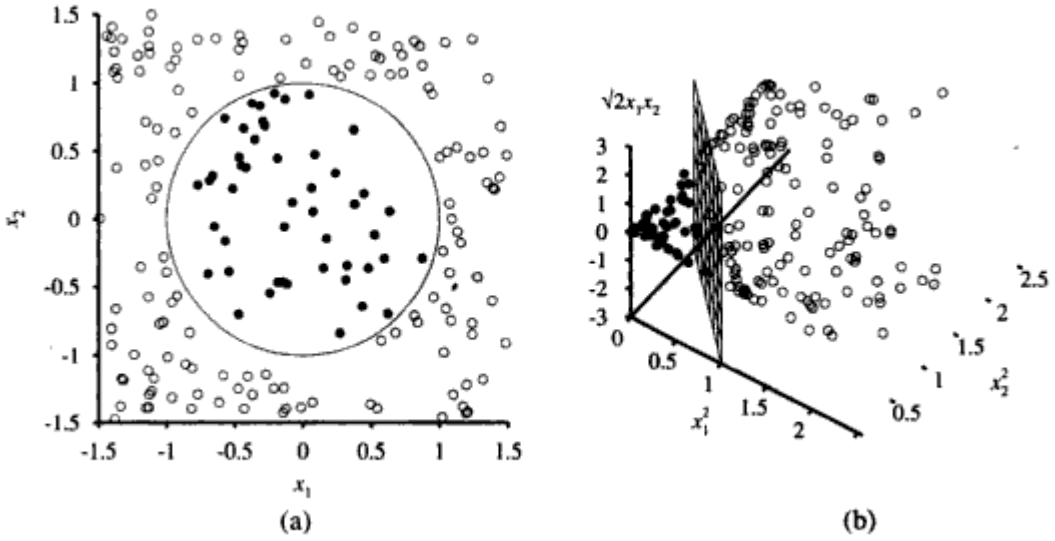

Figure 5.2: The left-hand image shows the two-class input data. No linear classifier can separate the two classes of data. The right hand image shows the data after it has been projected into 3D space using the kernel trick. The data is now separable by a plane (31).

we avoid "death by over fitting" (31).

We have seen how to compute $\mathbf{w} = \sum_{i=1}^{L} \alpha_i y_i x_i$ for a linearly separable set of binary-class data. We know how to get the support vectors S by finding the $\alpha_i > 0$. Therefore we can compute b as follows

$$b = \frac{1}{N_s} \sum_{s \in S} (y_s - \sum_{m \in S} \alpha_m y_m x_m \cdot x_s)$$

Finally, we can now classify new data x' (such as the test set data) by computing

$$y' = sign(\mathbf{w} \cdot x' + b)$$

(36). Support vector machines can also be used to classify data that is not linearly separable. However, we will not go over the theory behind partially inseparable data here.

In this thesis, all support vector machine results were gathered using the SVM Light package (25).

5.3.3 Features

A critical component in this fire detection algorithm is a support vector machine which we use to classify candidate fire sequences. Perhaps even more important pieces of the puzzle are the features extracted from the candidate fire sequences. Without high quality descriptive features—features which (at least some of the time) manifest themselves differently when the target sequence is fire as opposed to non-fire—the support vector machine won't be able to distinguish between the two classes. In this section we describe the different features used and why they work so well at differentiating fire sequence behavior from non-fire sequence behavior.

Fire Motion Feature

The first feature that we use to distinguish between fire and non-fire sequences is the edge motion of flame outlines. At first glance, fire behavior seems completely

unpredictable. Flames bob and weave in unpredictable patterns and the fire as a whole grows and shrinks seemingly with a mind of its own. However, closer inspection reveals that there is a pattern hidden in fire's chaotic behavior: there is much more motion near the top and sides of a fire then near the base. The trick is to encode this concept into a feature vector suitable for a SVM classifier. This is were the fire motion feature comes in.

Algorithm 2 Motion Feature Algorithm

Require: S $\{$a fire candidate sequence with N frames$\}$

for i=2:N **do**

$in = S_i$ AND S_{i-1}

$out = S_i$ OR S_{i-1}

Compute the $edge\_distance$ between in and out.

Bin the $edge\_distance$ for each point on the contour of in.

end for

$motion\_feature =$ mean of each bin.

return $motion\_feature$

1. *Contour*: the outer edge (one pixel thick) of a white blob.

2. *Edge distance*: the closest distance from an out contour point to an in contour point.

3. *Motion Region*: a slice of a circular region centered on the centroid of the rectangle that encompasses the cumulative *OR*ing of the blobs frames in the candidate sequence S.

4. *Bin*: given an *edge distance* value derived from an in contour point, the *Bin* operation adds the $edge\_value$ to the motion feature bin associated with the motion region from which the in contour point resides.

Say we have a fire candidate sequence of N frames F. The sequence is the binarized outline of something that is suspected to be fire. Motion along the edge of the candidate fire blob can be determined via differencing consecutive frames. The motion is therefore defined to be the non-zero parts of the following image difference

$$\begin{aligned} Motion \equiv &= |F_i - F_{i-1}| \\ &= (F_i \text{ AND } F_{i-1}) \text{ XOR } (F_i \text{ OR } F_{i-1}) \\ &= in \text{ XOR } out \end{aligned}$$

Let the concept of *quantity of motion* be defined to be the distance between a point on the edge of a flame silhouette at time t and that same edge point's new edge location at time $t+1$. The edge motion algorithm tries to approximate this concept in a computationally quick manner.

Now imagine a circle which is centered on the fire candidate blob and which circumscribes the blob. The circle is then divided like a pizza into a fixed set of slices which we will call *motion regions*. Thus, the motion feature that this fire detection algorithm tracks is the average quantity of motion observed in each motion region for the target fire candidate blob.

Edge Angle Feature

Say we are given an outline of some object, one way of describing that object is to create a histogram of normal angles from the object outline. Even though we are dealing with sequences of frames, this methodology still applies. The real question is: why should this distinguish fire sequences at all? If we consider that flames are usually long and pointy, we can already see that there are probably few vertical normal angles in a genuine fire sequence. Furthermore there are probably other patterns that a support vector machine is far better equipped than a human to ascertain. The algorithm for computing the edge normal angle feature is as follows

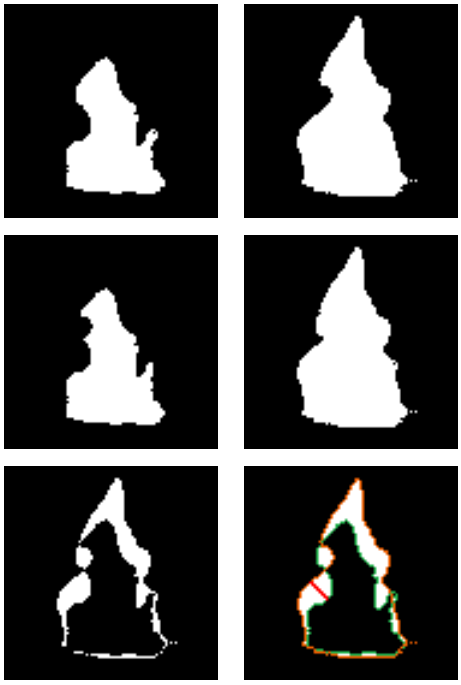

Figure 5.3: Top left: frame 1, top right: frame 2, middle left: inner, middle right: outer, bottom left: raw motion image, bottom right: annotated motion image. For the annotated motion image, the green line is the inner contour, the orange line is the outer contour, and the red line between the green and orange lines demonstrates edge distance for one inner contour point.

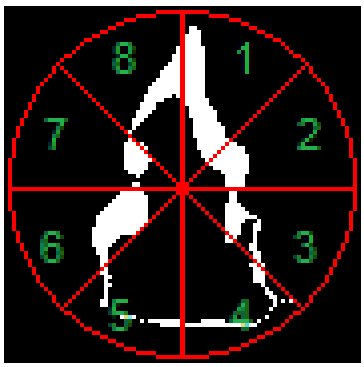

Figure 5.4: A conceptual demonstration of how edge motion is localized to motion regions. In this image, there are eight motion regions corresponding to eight motion features values.

Algorithm 3 Edge Angle Algorithm

Require: S {a fire candidate sequence with N frames}

create a feature histogram H and initialize it

for i=1:N **do**

Compute the contour C of blob in S_i

for each point p on contour C **do**

Compute the normal angle θ to the contour edge centered at p.

Bin θ in the feature histogram

end for

end for

Normalize the feature histogram H

return H

Flicker Count Feature

Another good fire feature to track is flame flicker. As we all have seen, flames usually move about a lot in a seemingly random but generally repetitive fashion. This pseudo periodic behavior can be tracked at the pixel granularity level with ease. The idea is to maintain a flicker counter—where flicker is defined to be when a pixel changes color from black to white—for every pixel in the input sequence. Pixels with flicker counts above some threshold are then classified as being a fire pixel and pixels with flicker counts below the threshold are classified as being non fire.

While this pixel-level fire classification algorithm works pretty well, we would like something a little more sophisticated since a fire candidate sequence will undoubtedly have many fire and non-fire pixels in it, making the sequence hard to classify solely using pixel level information. To classify the actual input sequence, the SVM uses the ratio of fire pixels detected using the fire-flicker-based pixel classification algorithm to the total number of pixels in the target blobs from the input sequence.

In both (35) and (34), pixel-level fire flicker was a prominent feature used to detect fire. In both cases, wavelets were used to detect the actual flame flicker based

on variations on the pixel intensity over a sequence of frames. We didn't use a bank of wavelets to detect the flicker because our algorithm has already gone through the work of determining whether a pixel was fire colored or not. All that it needs to do to detect flame flicker is to track when a pixel goes from being fire colored to non-fire colored or vice versa.

5.3.4 Putting Everything Together

In the previous sections we discussed the different features used to distinguish fire candidate sequence. For both the flame motion and edge angle features, a sampling rate must be chosen. These sampling rates determine how often to compute the edge motion or edge normal angle while traversing the edge contour. For the flicker count feature, the flicker count threshold must be chosen. Lastly, we must decide upon the SVM kernel that we're going to use to perform the classification with.

The edge motion feature requires us to decide upon the number of motion regions to use. The more motion regions we have, the more "focused" the analysis of edge motion. However, more regions require more processing. We can see that the granularity decision is not affected by any of the other parameters used in this fire detection algorithm. Therefore, we can fix all other parameters and just vary the number of motion regions to see which setting produces the best results

Finding the best number of edge normal angle bins is virtually the same as finding the best number of edge motion regions. Here too we fix all other parameters and just vary the number of normal angle histogram bins.

The edge normal angle sampling rate effectively controls the amount of processing required to compute the given feature. We expect that lower sampling rates might reduce the accuracy of the feature when compared to higher sampling rates. While it is probable that the edge normal angle sampling distance and the number of edge normal angle bins are not independent from each other with respect to their affect on classification results, we treat them as if they are independent. This independence assumption

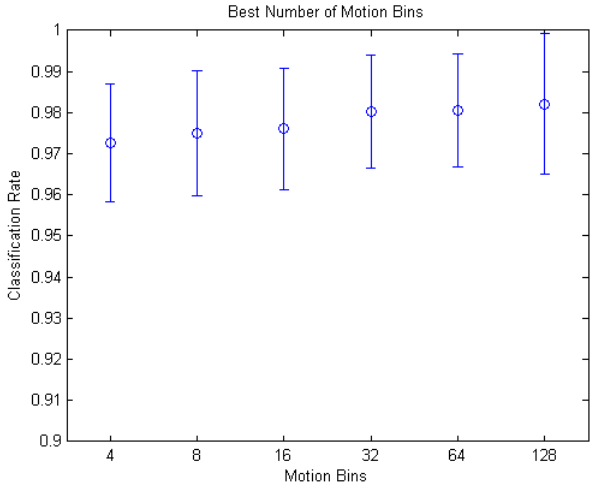

Figure 5.5: A graph of classification rates vs the number of edge motion bins. All other feature parameters are held steady.

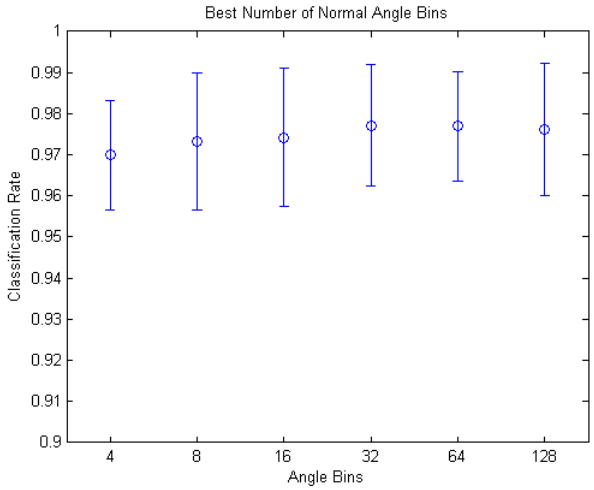

Figure 5.6: A graph of fire classification using an SVM and edge normal-line angles. All other feature parameters are fixed.

reflects our belief that neither parameter affects the results of the other much, nor is it absolutely crucial to get the absolute best classification results via the best parameter settings. Good is good enough for this application. Therefore, we can fix all other parameter values and only vary the sampling rate to determine the best sampling rate.

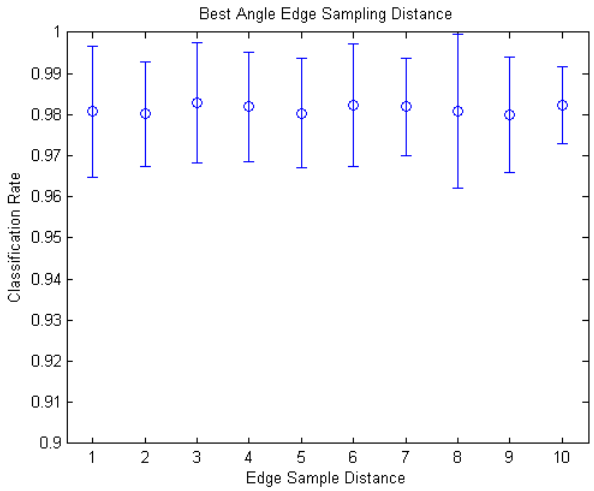

Figure 5.7: A graph of fire classification using a support vector machine and edge normal-line angles. All other parameters are fixed.

Just like the case with the sampling rate parameters, the flame flicker feature is completely independent of any of the other features with respect to their parameter values. Therefore, we can simply use an ROC curve to determine an acceptable flicker count threshold value. The flicker threshold that we decided upon is 8 flickers per 50 frames which was chosen because it gives us a true positive rate of 16.2% and a false positive rate of 0.57%.

Because these rates apply to pixel level classification, it seemed like a good idea to keep the false positive rate as low as possible and let other features (motion, normal angle, etc.) pick up the slack.

Table 5.2: Pixel flicker table showing the relationship between the flicker threshold value and the subsequent true/false positive classification rate of pixel-level classification.

Flicker Threshold	True Positive Rate	False Positive Rate	Flicker Flicker Threshold	True Positive Rate	False Positive Rate
1	3.14×10^{-1}	8.45×10^{-2}	⋮	⋮	⋮
2	3.11×10^{-1}	7.12×10^{-2}	22	9.99×10^{-4}	8.25×10^{-5}
3	2.74×10^{-1}	3.67×10^{-2}	23	4.70×10^{-4}	7.07×10^{-5}
4	2.67×10^{-1}	2.75×10^{-2}	24	2.21×10^{-4}	6.55×10^{-5}
5	2.25×10^{-1}	1.59×10^{-2}	25	8.19×10^{-5}	5.93×10^{-5}
6	2.15×10^{-1}	1.23×10^{-2}	26	5.16×10^{-5}	5.49×10^{-5}
7	1.80×10^{-1}	7.69×10^{-3}	27	1.78×10^{-5}	4.82×10^{-5}
8	1.62×10^{-1}	5.73×10^{-3}	28	3.56×10^{-6}	4.31×10^{-5}
9	1.29×10^{-1}	3.58×10^{-3}	29	1.78×10^{-6}	3.68×10^{-5}
0	1.14×10^{-1}	2.61×10^{-3}	30	0.00	3.42×10^{-5}
11	8.78×10^{-2}	1.64×10^{-3}	31	0.00	2.87×10^{-5}
12	7.31×10^{-2}	1.16×10^{-3}	32	0.00	2.50×10^{-5}
13	5.38×10^{-2}	7.64×10^{-4}	33	0.00	1.88×10^{-5}
14	4.21×10^{-2}	5.49×10^{-4}	34	0.00	1.51×10^{-5}
15	2.93×10^{-2}	3.66×10^{-4}	35	0.00	1.07×10^{-5}
16	2.15×10^{-2}	2.59×10^{-4}	36	0.00	8.84×10^{-6}
17	1.39×10^{-2}	1.98×10^{-4}	37	0.00	4.79×10^{-6}
18	9.26×10^{-3}	1.55×10^{-4}	38	0.00	4.79×10^{-6}
19	5.55×10^{-3}	1.26×10^{-4}	39	0.00	1.47×10^{-6}
20	3.39×10^{-3}	1.06×10^{-4}	40	0.00	3.68×10^{-7}
21	1.86×10^{-3}	9.09×10^{-5}	41	0.00	0.00
⋮	⋮	⋮	⋮	⋮	⋮

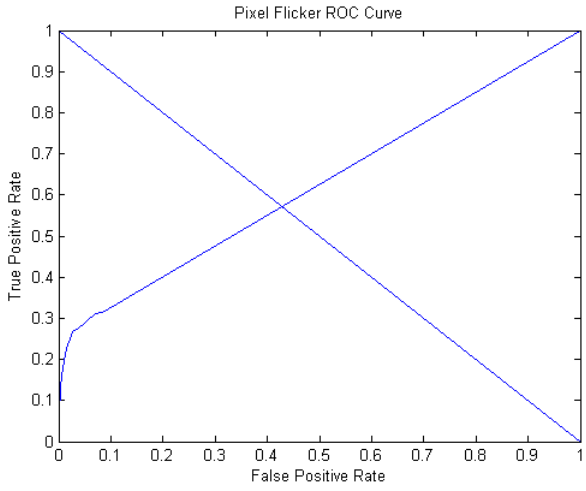

Figure 5.8: Pixel flicker ROC curve used to determine the flicker count threshold value for the sequence flicker feature. Note that this ROC curve is an alternate representation of the data from table 5.2.

Finally, the best support vector machine kernel type is dependent on the structure of the input data in projected space to the classifier. The different feature parameters don't change the data structure meaning that we can fix all other parameter values and only vary the SVM kernel parameter to determine which one is best.

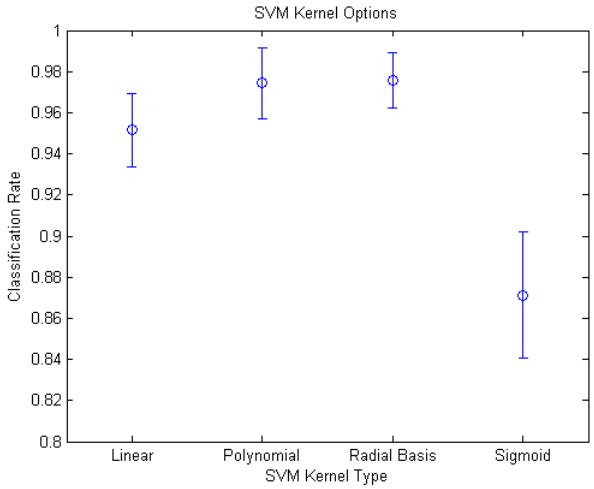

Figure 5.9: A graph of classification rates vs SVM kernel. All other feature parameters are held steady.

5.3.5 Results

Using 20-fold cross validation on the training set, we can get a pretty good idea how each individual feature classifies fire videos on its own. As the table below shows, each feature does well at classifying fire videos. However, the individual feature results max out in the early ninety percent classification range.The end of this section shows that combining all of these features increases the classification rate into the high ninety percent range.

The following are the final results achieved using the best parameter values for the edge motion feature, the edge normal angles feature, and the fire flicker feature. We compute results for the training set and for the test set which we left untouched

Table 5.3: 20-fold cross validation results on the training set.

Solo Feature Results			
	Edge Motion	**Edge Normal Angle**	**Sequence Flicker**
classification rate	0.9130 ± 0.029	0.950 ± 0.021	0.907 ± 0.0316

(untrained upon) until now. The training set results were computed using 20-fold cross validation. The testing set results were computed using a single run with the SVM trained on the complete training set.

Table 5.4: Total classification results for the training set and for the test set.

	Training Set	**Testing Set**
True Positives	585	712
True Positives	12	42
True Negative	1258	1400
False Negative	19	54
Classification Rate	0.983 ± 0.011	0.96

5.3.6 Run Time

The wall clock run time of an algorithm is somewhere between difficult and impossible to determine in a general, hardware-agnostic sense. The following estimation of algorithmic wall clock run time is described using the development hardware for this research project. More specifically, we computed the average run times of ten sample videos where each is tested ten times. The following graph show the 50-frame, square-pixel-area normalized, sequence run times of each of the ten videos as well as the global average sequence computation run time.

These results were obtained using a 3 GHz quad-core AMD Phenom™ 945 processor and 4GB of RAM. It should be noted that the test scripts and the fire detection algorithms are single-process, single-threaded applications.

In the end, the system was able to achieve a global average run time of one 50-frame, area-normalized, pixel per 2.0865×10^{-5} second. In other words, the algorithm and hardware were able to process roughly 120, 20x20, 50-frames sequences per second.

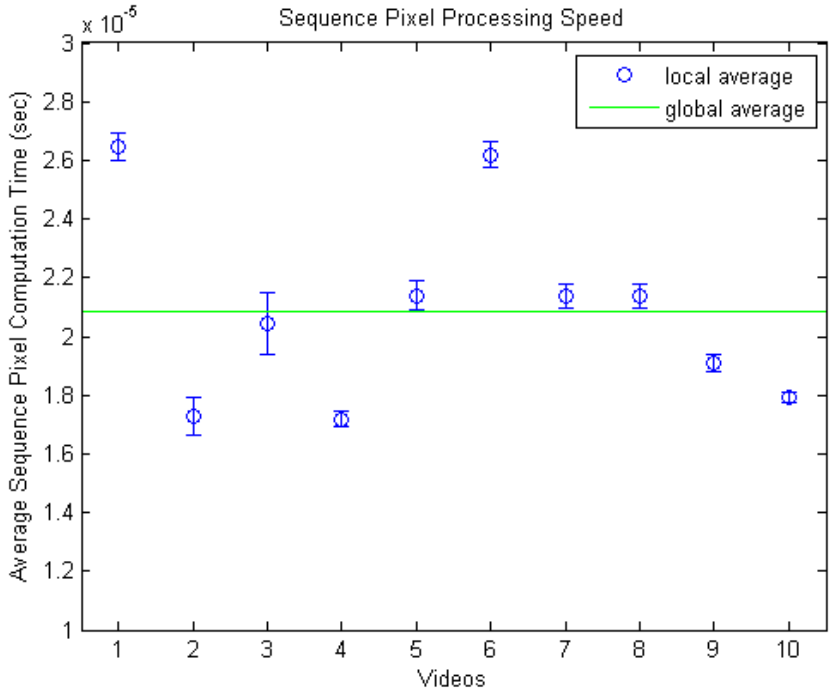

Figure 5.10: A graph of 50-frame sequence pixel run times.

Given that the training and testing videos used in this research project never saw more than 50 sequences per second, and usually saw far less, the algorithm easily qualifies as being a real-time fire detection algorithm.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis we analyze different vision-based fire detection algorithms and we describe a new, robust method for detecting fire with low quality cameras. As we showed the previous section, the VBFD algorithm introduced by this thesis achieves good results and is very effective. But how does it stack up to other VBFD algorithms? Unfortunately, given that there currently is no common fire video data set being used by VBFD researchers, the following is a qualitative analysis and comparison of some of the current VBFD methods.

Many of the current VBFD algorithms rely solely on color-based classification to distinguish fire regions from non-fire regions. Some of these algorithms use human-generated rules based in various color spaces such as (6, 9, 39, 18, 10) and some trained classifiers, like a neural network, to detect fire color (23). The one big problem with this class of VBFD algorithm is that color simply is not enough to detect fire in most real world settings without accepting a significant false alarm rate. For any realistic application of a VBFD system using color-based fire detection, the alarm would cry wolf too often to let it be of much use.

Adding flame flicker to the mix of fire features to analyze significantly improves the effectiveness of a VBFD system by reducing the false alarm rate. After all, far fewer

visual phenomena exhibit a high rate of luminance flicker than are fire colored. Vision-based fire detection algorithms such as (35, 34) which have flame flicker detection report good results. Our experience with this particular feature has also been positive. However, classification on our data sets shows use that luminance flicker does show up in the real world in ways that mimic fire flicker. Additional features are required to improve the classification results from good to great.

Another interesting type of VBFD algorithm uses shape to distinguish fire from other objects. For example, the authors of (27) used Fast Fourier Transforms to math-

Figure 6.1: Fire shape outlines (27).

ematically featurize candidate blob edge shapes. They then used a support vector machine to categorize fire shapes and non-fire shapes. While they report excellent results it is worth noting two weaknesses of this method.

First, quantization errors crop up when sampling the boundary of a small fire

candidate blob. Thus, the authors of (27) decided to ignore regions which were too small. This means that this type of algorithm suffers when fire is far away from the camera. Furthermore, it means that the algorithm has a hard time detecting fires when they are just starting which is when you want to catch them and put them out, i.e. before they grow big and cause a lot of damage.

Second, the algorithm has trouble differentiating between the uneven edges of flames from a house fire and the ovular shapes of the sun and a lamp (27). The problem is that the feature detection algorithm put forth in this thesis outputs sequences with outlines which are much closer to the shape of fire than to such obviously non-fire shapes as the sun. In other words, the shape classification algorithm is probably not robust enough to deal with the output of the feature detection stage of this algorithm. We

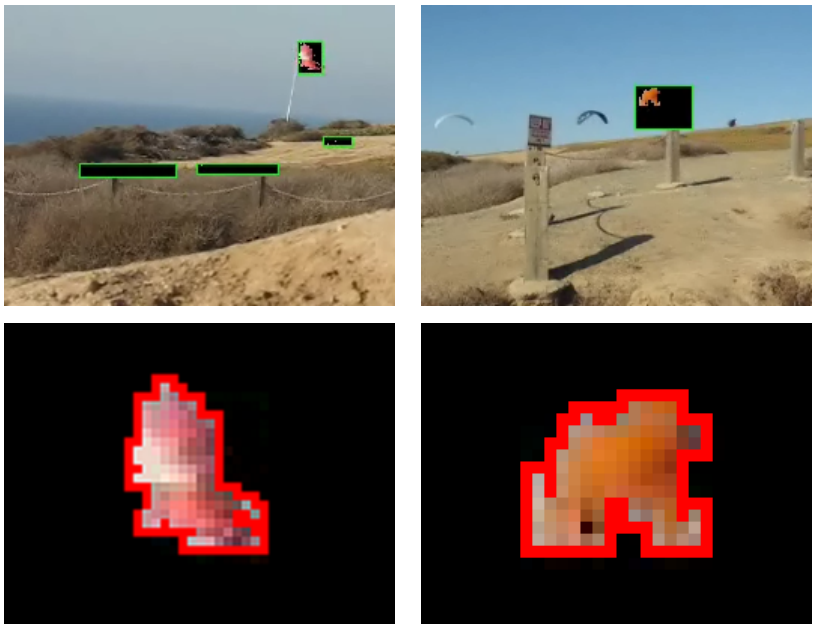

Figure 6.2: Top row: non-fire shape outlines derived from the feature detection stage. Bottom row: blown up versions of the shape outlines from the top row images.

conducted some basic experiments along the lines of (27) to see how well a support vector machine could classify fire candidate sequences based solely on the shape of the candidate's outline. To test this classifier, we randomly generated square images with

either a circle or a square in each of these images. The shape was randomly sized—from half the size of the image to the size of the whole image—and was randomly placed in the image so that the image contained the whole shape. The following are the training results using 40 Fourier descriptors to represent a shape just like (27). While these

Table 6.1: Each training and testing file was composed of 1000 circles and 1000 squares.

Image width/height (pixels)	50	75	100	125	150
Polynomial Kernel	71.05%	72.10%	71.35%	73.55%	73.55%
Radial Basis Function Kernel	70.11%	78.98%	78.43%	74.85%	73.60%

results are passable, they aren't nearly as good as we expected. This could be because of the number of descriptors used, the size of the training sets, or the size of the shapes in the training and testing images.

When we ran the Fourier descriptor shape classifier on each of the static frames of the sequences from the training and testing sets for this thesis, we were only able to achieve a classification rate or 50.55% using a radial basis function for the SVM kernel. This classification rate is probably the result of very small shapes being classified (averaging less than 100 pixels to a side) and the result of the positive and negative data having similarly complicated and irregular shapes.

Because this shape classification algorithm has trouble with small and distant objects, and because the algorithm requires much "cleaner" data than the candidate sequences outputted by the feature detection stage, it seems likely that the algorithm put for in this thesis is applicable to more real world fire detection situations than a shape-based fire detection algorithm.

To create the fire detection and recognition algorithm presented in this thesis, we tried to take the best ideas from the VBFD research community and add some ideas which were successful in non-VBFD research to create a fast and robust VBFD algorithm. The results achieved by this algorithm are very exciting and show that this algorithm has a lot of promise for real world applications. However, there are still a number of significant weaknesses which still need to be addressed.

The biggest weakness for this system is the ROI detection algorithm. The ROI classification stage relies on the ROI detection stage to deliver clean candidate sequences which contain no fire/non-fire hybrid sequences. Currently, the ROI detection stage relies on color and motion detection to find these candidate sequences and both color and motion algorithms have significant Achilles heels. Color detection is always hostage to variations in lighting conditions. Fast motion detection algorithms lack the specificity necessary to differentiate motion from lighting changes. The undeniable conclusion is that more work needs to be done to make this stage more reliable and robust.

Another drawback for this fire detection algorithm is the fact that the cameras that capture the input data need to be completely stationary. The motion detection algorithms rely on the camera to be stationary since they don't have any understanding of the scene that they capture and they don't track any objects in the scene. Furthermore, the camera can't be rotated about the focal axis because both the edge motion feature and the edge normal angle feature are anisotropic.

6.2 Future Work

Beyond improving the weaknesses mentioned in the previous section, there are a number of promising avenues of research for improving this VBFD algorithm. These research trajectories promise to improve the classification rate of the system, and add new capabilities that would significantly improve a VBFD system.

One interesting avenue for research would be to try to improve the overall classification rate by examining the spatial and temporal clusters of candidate fire sequences. Clustering analysis has the potential to dramatically reduce the false alarm rate because isolated fire sequences could be detected and ignored since fire rarely pops up for a few seconds and then completely disappears.

Another idea is to incorporate smoke detection capabilities into the algorithm. This would strengthen the certainty of some classification decisions where the system thinks it sees both fire and smoke. Furthermore, a fire may start outside the line of site

of the camera. Adding smoke detection capabilities would seriously improve a VBFD system's overall effectiveness at detecting fire. After all, where there's smoke, there's fire.

In the conclusion, this thesis, demonstrates a new, robust VBFD algorithm that is perfectly suited for low quality video data. The algorithm achieves good results and shows a lot of promise toward real world application.

References

[1] Beard, A., and Carvel, R., 2005: *The Handbook of Tunnel Fire Safety*. Thomas Telford, Ltd.

[2] Boothroyd, T., 2005: *Fire Detection and Suppression Systems*. International Fire Service Training Association.

[3] Bryan, J., 1993: *Fire Suppression and Detection Systems*. Macmillan.

[4] Burges, C. J. C., 1998: A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, **2**, 121–167.

[5] Burke, R., 2007: *Fire Protection: Systems and Response*. CRC Press.

[6] Çelik, T., and Demirel, H., 2009: Fire detection in video sequences using a generic color model. *Fire Safety Journal*, **44**, 147–158.

[7] Chan, A., 2005: Computing the martin distance.

[8] Chan, A. B., and Vasconcelos, N., 2007: Classifying video with kernel dynamic textures. *Computer Vision and Pattern Recognition*, 1–6.

[9] Chen, T.-H., Kao, C.-L., and Chang, S.-M., 2006: An intelligent real-time fire-detection method based on video processing. In *Joint Conference on Information Sciences*.

[10] Chen, T.-H., Wu, P.-H., , and Chiou, Y.-C., 2004: An early fire-detection method based on image processing. In *Image Processing*, volume 3, 1707–1710. ICIP.

[11] Cheung, S.-C. S., and Kamath, C., 2004: Robust techniques for background subtraction in urban traffic video. In *Visual Communications and Image Processing*, volume 5308. SPIE.

[12] Cock, K. D., Cock, K. D., Moor, B. D., and Moor, B. D., 2000: Subspace angles between linear stochastic models. In *Conference on Decision and Control*, 1561–6. IEEE.

[13] Collins, R., Lipton, A., Kanade, T., Fujiyoshi, H., Duggins, D., Tsin, Y., Tolliver, D., Enomoto, N., and Hasegawa, O., 2000: A system for video surveillance and monitoring. Technical report, Robotics Institute.

[14] Cucchiara, R., Piccardi, M., and Prati, A., 2003: Detecting moving objects, ghosts, and shadows in video streams. In *Pattern Analysis and Machine Intelligence*, volume 25, 1337–1342. IEEE.

[15] Cutler, R., and Davis, L., 1998: View-based detection and analysis of periodic motion. In *International Conference on Pattern Recognition*, volume 1, 495–500.

[16] Doretto, G., Chiuso, A., Wu, Y. N., and Soatto, S., 2003: Dynamic textures. *International Journal on Computer Vision*, **51**, 91–109.

[17] Doretto, G., Cremers, D., Favaro, P., and Soatto, S., 2003: Dynamic texture segmentation. In *International Conference on Computer Vision*, volume 2, 1236–1242. IEEE.

[18] Ebert, J., and Shipley, J., 2007: Computer vision based method for fire detection in color videos.

[19] Eveland, C. K., Konolige, K. G., and Bolles, R. C., 1998: Background modeling for segmentation of video-rate stereo sequences. In *Computer Vision and Pattern Recognition*, 266–271. IEEE.

[20] Friedman, N., and Russell, S., 1997: Image segmentation in video sequences: A probabilistic approach. In *Conference on Uncertainty in Artificial Intelligence*, 175–181.

[21] Gloyer, B., Aghajan, H. K., Siu, K.-Y., and Kailath, T., 1995: Video-based freeway monitoring system using recursive vehicle tracking. In *Symposium on Electonic Imaging: Image and Video Processing*, volume 2421, 173–180. SPIE.

[22] Hecht-Nielsen, R., 2008: Perceptrons.

[23] Horng, W.-B., and wen Peng, J., 2006: Image-based fire detection using neural networks. In *Joint Conference on Information Sciences*.

[24] III, W. P., Shah, M., and da Vitoria Lobo, N., 2000: Flame recognition in video. In *Workshop on Applications of Computer Vision*, 224–229. IEEE.

[25] Joachims, T., 1999: Making large-scale svm learning practical. *Advances in Kernel Methods - Support Vector Learning*, 169–184.

[26] Lai, C.-L., and Yang, J.-C., 2008: Advanced real time fire detection in video surveillance system. In *International Symposium on Circuits and Systems*.

[27] Liu, C.-B., and Ahuja, N., 2004: Vision based fire detection. In *International Conference on Pattern Recognition*, volume 4, 134–137. IEEE.

[28] Martin, R. J., 2000: A metric for arma processes. *Transactions on Signal Processing*, **48**(4), 1164–1170.

[29] McFarlane, N., and Schofield, C., 1995: Segmentation and tracking of piglets in images. *Machine Vision and Applications*, **3**, 187–193.

[30] Mercer, J., 1909: Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society London*.

[31] Russell, S. J., and Norvig, P., 2003: *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall.

[32] Saisan, P., Doretto, G., Wu, Y. N., and Soatto, S., 2001: Dynamic texture recognition. In *Computer Vision and Pattern Recognition*, volume 2, 58–63. IEEE.

[33] Terpak, M. A., 2001: *Fireground Size-Up*. Fire Engineering Books.

[34] Töreyin, B. U., Dedeoğlu, Y., and Çetin, A. E., 2005: Flame detection in video using hidden markov models. In *International Conference on Image Processing*, 1230–1233. IEEE.

[35] Töreyin, B. U., Dedeoğlu, Y., Güdükbay, U., and Çetin, A. E., 2006: Computer vision based method for real-time fire and flame detection. *Pattern Recognition Letters*, **27**, 49–58.

[36] Tristan, F., 2009: Support vector machines explained. Technical report, University College London.

[37] United States Federal Emergency Management Agency, 2009: National fire statistics. http://www.usfa.dhs.gov/statistics/national/index.shtm.

[38] Vapnik, V., 1999: *The Nature of Statistical Learning, Second Edition*.

[39] Zhang, D., Rao, Y., Zhao, J., Zhao, J., Hu, A., and Cai, B., 2007: Feature based segmentation and clustering on forest fire video. In *International Conference on Robotics and Biomimetics*, volume 4. IEEE.