EARLY FOREST FIRE HEAT PLUME DETECTION USING NEURAL
NETWORK CLASSIFICATION OF SPECTRAL DIFFERENCES BETWEEN
LONG-WAVE AND MID-WAVE INFRARED REGIONS

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Raul-Alexander Aldama

June 2013

© 2013

Raul-Alexander Aldama

ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE: Early Forest Fire Heat Plume Detection using Neural Network
Classification of Spectral Differences between Long-Wave and
Mid-Wave Infrared Regions

AUTHOR: Raul-Alexander Aldama

DATE SUBMITTED: June 2013

COMMITTEE CHAIR: Dr. John Saghri, Associate Professor

COMMITTEE MEMBER: Dr. John Jacobs, Raytheon Professor of Practice

COMMITTEE MEMBER: Dr. Jane Zhang, Associate Professor

ABSTRACT

EARLY FOREST FIRE HEAT PLUME DETECTION USING NEURAL NETWORK CLASSIFICATION OF SPECTRAL DIFFERENCES BETWEEN LONG-WAVE AND MID-WAVE INFRARED REGIONS

Raul-Alexander Aldama

It is difficult to capture the early signs of a forest fire at night using current visible-spectrum sensor technology. Infrared (IR) light sensors, on the other hand, can detect heat plumes expelled at the initial stages of a forest fire around the clock. Long-wave IR (LWIR) is commonly referred to as the “thermal infrared” region where thermal emissions are captured without the need of, or reflections from, external radiation sources. Mid-wave IR (MWIR) bands lie between the “thermal infrared” and “reflected infrared” (i.e. short-wave IR) regions. Both LWIR and MWIR spectral regions are able to detect thermal radiation; however, they differ significantly in regards to their detection sensitivity of forest-fire heat plumes.

Fires fueled by organic material (i.e. wood, leaves, etc.) primarily emit hot carbon dioxide (CO_2) gas at combustion. Consequently, because CO_2 is also present in the atmosphere, re-emission restricts the spectral transmittance and hence spectral radiance over a wide range of frequencies in the MWIR region. Moreover, as the distance between the detector and fire’s heat plume becomes greater, the additional CO_2 introduced into the detection path leads to further attenuation of photon emittance. Since these absorption frequencies also lie within the response bandwidth of the MWIR sensor material, captured heat plume radiation manifests itself as a group of “flooded” or saturated pixels that exhibit very little dynamic behavior. Meanwhile, since the LWIR spectral region is not significantly affected by atmospheric gas absorption, its sensor is able to capture the forest fire’s heat plume thermal signature at far range without such complications.

By exploiting the underlying spectral differences between LWIR and MWIR regions, this study aims to achieve early forest fire heat plume detection via direct identification of its dynamic

characteristics whilst concurrent attenuation of detected non-fire-related radiation. A land-based, co-located, cooled-LWIR/cooled-MWIR (CLWIR/CMWIR) detector camera is used to capture and normalize synchronized video from which sequential spatial-domain difference frames are generated. Processed frames allow for effective extraction of the heat plume's "flickering" features, which are characteristic to the early stages of a forest fire.

A multilayer perceptron (MLP) neural network classifier is trained with feature points generated from known target samples (i.e. supervised learning). Resulting detection performance is assessed via detection time, error metrics, computation time, and parameter variation. Results indicate that heat plumes expelled at the early stages of a forest fire can be identified with high sensitivity, low false alarm, and at a farther range than commercial detectors.

Keywords: Image Processing, Computer Vision, Long-Wave, Mid-Wave, Infrared, Spectral Absorption, Forest Fire Detection, Feature Extraction, Multilayer Perceptron, Neural Network Classifier

ACKNOWLEDGEMENTS

I would like to thank Dr. John Saghri and Dr. John Jacobs for their support and encouragement throughout the development of this thesis. Without their expertise and assistance, this would not have been possible.

In addition, a special thanks to Raytheon for sponsoring this project. Many people have been involved in this project one way or another; to those people, I extend my deepest gratitude.

Finally, I would like to thank my family and friends for always supporting me and encouraging me to achieve great things. Because of you, I feel blessed. With much love and respect to everyone in my life, thank you.

TABLE OF CONTENTS

LIST OF TABLES	x
LIST OF FIGURES	1
1. INTRODUCTION	6
1.1 Motivation	6
1.2 Background	6
1.3 Context	7
1.3.1 Ryan Radjabi	7
1.3.2 Timothy M. Davenport	7
1.3.3 Daniel G. Kohler	8
1.3.4 Georges F. Moussa	8
1.3.5 Ansel Boynton	8
1.4 Problem Statement	8
1.5 Theory	9
1.6 Common Operations	14
1.6.1 Arithmetic Mean, Expected value, or Average (1-Dimensional)	14
1.6.2 Minimum (1-Dimensional)	15
1.6.3 Maximum (1-Dimensional)	15
1.6.4 Median (1-Dimensional)	15
1.6.5 Median Filtering (2-Dimensional)	15
1.6.6 Unbiased Sample Variance (1-Dimensional)	16
1.6.7 Corrected Sample Standard Deviation (1-Dimensional)	16
1.6.8 Cross Correlation (1-Dimensional)	16
1.7 Proposed Early Forest Fire Detection Solution	17
1.8 Thesis Outline	18
1.9 Implementation Notes	18
2. DATA ACQUISITION	21
2.1 Controlled Fire Experiment	21
2.2 Raw Frame Extraction	24
2.3 Co-located Sensors	25
2.3.1 Sensor Settings	25
2.3.2 Sensor Noise	31
2.4 Pre-Processing	

3. FRAME PROCESSING	55
3.1 Spatial-Domain Sequential Frame Processing Methods	55
3.1.1 Multispectral Difference	55
3.1.2 Thresholding	59
3.1.3 Unprocessed	63
3.2 Principal Component Analysis Processing Methods	64
3.2.1 Principal Component Analysis Computation Procedure.....	65
3.2.2 Multi-Temporal Principal Component Analysis.....	68
3.2.3 Multi-Spectral Principal Component Analysis	74
3.2.4 Multi-Spectral-Temporal Principal Component Analysis	75
4. FEATURE EXTRACTION	77
4.1 Spatial-Domain Unsharp Mask Filter	77
4.2 Temporal Domain Feature Masking	83
4.2.1 Input Masking	83
4.2.2 Coefficient of Variation Masking	87
4.2.3 Cross Correlation Masking	89
4.3 Feature Point Generation	92
4.3.1 Feature Selection.....	94
4.3.2 Feature Points	95
4.4 Feature Extraction Parameters	98
5. CLASSIFICATION	99
5.1 Multilayer Perceptron Neural Network.....	99
5.2 Training.....	101
5.3 Training Performance	102
6. RESULTS	107
6.1 Frame Processing Performance.....	107
6.1.1 Earliest Successful Heat Plume Detection	107
6.1.2 Classification Error	108
6.1.3 Computation Time	108
6.2 Dissimilar Video Performance.....	110
6.3 Parameter Variation Performance	119
6.3.1 Video Registration	119
6.3.2 Gaussian Kernel Sigma Value	123
7. CONCLUSION.....	129
7.1 Concluding Remarks.....	129
7.2 Future Work	129

BIBLIOGRAPHY	130
GLOSSARY	133
APPENDIX A: MATLAB SCRIPTS, FUNCTIONS, and TOOLS	134

LIST OF TABLES

Table 1: Infrared Spectrum Regions of Interest.....	9
Table 2: Controlled Fire Camera Specifications.....	21
Table 3: Raw Video Sequence Summary	22
Table 4: Summary of ‘oats1’ CLWIR/CMWIR Pixel Intensities and Saturation Thresholds for Total 8000 Frames.....	27
Table 5: Summary of Determined ‘oats1’ CLWIR/CMWIR Pixel Intensities over Valid 2124 Frames ...	35
Table 6: 'oats1' Frame 1201 Statistics at Different Stages of Normalization.....	40
Table 7: 'oats1' difCLCM Feature Extraction Parameters	98
Table 8: 'oats1' Earliest Successful Heat Plume Detection	107
Table 9: 'oats1' Number of 1-Second Interval Classification Errors over 70-seconds.....	108
Table 10: Computer System Specifications	109
Table 11: 'oats1' Computation Time to Pre-Process and Extract Features/Classify difCLCM Frames 1201 to 1230	109
Table 12: 'oats1' Computation Time to Process Frames 1201 to 1230	109

LIST OF FIGURES

Figure 1: Atmospheric Transmittance: MWIR Sensor Response (Dashed-Green), LWIR Sensor Response (Solid-Blue) [16]	10
Figure 2: Typical Spectral Radiance of an Experimental Forest Fire from ~7m Away: MWIR Sensor Response (Dashed-Green), LWIR Sensor Response (Solid-Blue) [17].....	11
Figure 3: Comparison of Short Range Atmospheric Absorption between LWIR (Left) and MWIR (Right) Spectral Regions [19].....	12
Figure 4: Spectral Radiant Emittance: MWIR Sensor Response (Dashed-Green), LWIR Sensor Response (Solid-Blue) [16]	13
Figure 5: Flow Diagram of Proposed Early Forest Fire Detection Solution.....	17
Figure 6: ‘Gray’ Color Scale.....	20
Figure 7: ‘Hot’ Color Scale.....	20
Figure 8: Contrast Enhancement Display Range	20
Figure 9: Images at Experiment Site.....	23
Figure 10: Satellite Images of the Experiment Site [11].....	23
Figure 11: Raw CLWIR and CMWIR Frame Extraction Process [10].....	24
Figure 12: Observed Maximum Fire Core Height of the Controlled Fire Experiment.....	25
Figure 13: ‘oats1’ CLWIR Spatial Domain Sensor Response at Frame 1 (Top) and Frame 2600 (Bottom)	26
Figure 14: ‘oats1’ CLWIR (Left) and CMWIR (Right) Pixel Intensity around Fire Region over Total 8000 Frames (266.67 seconds)	27
Figure 15: ‘oats1’ Frames Considered Valid (Red) and Invalid (Blue)	28
Figure 16: Relationship between ‘oats1’ Valid Frame Numbers and Total Frame Numbers	29
Figure 17: ‘oats1’ CLWIR (Left) and CMWIR (Right) Pixel Intensity around 65x86 Fire Region over Valid 2124 Frames (70.8 seconds).....	30
Figure 18: Arbitrary Set of ‘oats1’ Frames Chosen to Illustrate Thesis	31
Figure 19: ‘oats1’ CLWIR (Left) and CMWIR (Right) Min and Max Pixel Intensity Values over Total 8000 Frames (266.67 seconds)	32
Figure 20: ‘oats1’ CLWIR (Left) and CMWIR (Right) Min and Max Pixel Intensity Values over Valid 2124 Frames (70.8 seconds)	32
Figure 21: Salt (Blue)-and-Pepper (Green)-Noise in ‘oats1’ Valid CLWIR Frame 2090	33
Figure 22: Salt (Blue)-and-Pepper (Green) Noise in ‘oats1’ Valid CMWIR Frame 2090.....	34
Figure 23: ‘oats1’ CLWIR Frame 1201 Histogram Dissection before Normalization	35
Figure 24: ‘oats1’ CMWIR Frame 1201 Histogram Dissection before Normalization	36
Figure 25: ‘oats1’ CLWIR Non-Fire Area for Frame 1201	37
Figure 26: ‘oats1’ Non-Fire Average (Left), Minimum (Center), and Maximum (Right) CM/CL ratio over Total 8000 Frames (266.67 seconds)	37

Figure 27: 'oats1' Non-Fire Average (Left), Minimum (Center), and Maximum (Right) CM/CL ratio over Valid 2124 Frames (70.8 seconds).....	38
Figure 28: 'oats1' CLWIR Frame 1201 Normalization Process: Before Normalization (Left), After CM/CL Ratio Correction (Center), and After Dynamic Range Expansion (Right).....	39
Figure 29: 'oats1' CMWIR Frame 1201 Normalization Process: Before Normalization (Left), After CM/CL Ratio Correction (Center), and After Dynamic Range Expansion (Right).....	40
Figure 30: 'oats1' CLWIR (top) and CMWIR (bottom) Frame 1201 Histograms and Intensity Maps	41
Figure 31: 'oats1' CLWIR Frame 1201 Histogram Dissection after Normalization	42
Figure 32: 'oats1' CMWIR Frame 1201 Histogram Dissection after Normalization.....	42
Figure 33: 'oats1' Frame 1201 Histograms before (top) and after (bottom) Normalization.....	43
Figure 34: 'oats1' Frame 1201 Multispectral Scatter Plot	43
Figure 35: 'oats1' CLWIR Normalized Frame 1201 (Contrast Enhanced)	45
Figure 36: 'oats1' CMWIR Normalized Frame 1201 (Contrast Enhanced)	45
Figure 37: 'oats1' CMWIR Composite Reference Frame for Frames 1201 to 1230	46
Figure 38: 'oats1' CMWIR Salient Features found in Reference Frame and Frame 1201 using the Harris Corner Detection Algorithm	47
Figure 39: 'oats1' CMWIR Path of Correspondence Points between Reference Frame and Frame 1201 ..	47
Figure 40: 'oats1' CMWIR RANSAC Determined Correspondance Inliers for Frame 1201	48
Figure 41: 'oats1' Variance of CMWIR Unregistered Frames 1201 to 1230	49
Figure 42: 'oats1' Variance of CMWIR Registered Frames 1201 to 1230.....	49
Figure 43: 'oats1' CL Registration Histograms for Frame 1201	50
Figure 44: 'oats1' CL Tones due to Registration	51
Figure 45: 'oats1' Pre-Processed CLWIR Frame Number 1201	52
Figure 46: 'oats1' Pre-Processed CMWIR Frame Number 1201.....	52
Figure 47: 'oats1' CLWIR (Left) and CMWIR (Right) Min and Max Pixel Intensity Values over Pre-Processed 2100 Frames (70 seconds).....	53
Figure 48: 'oats1' CLWIR (Left) and CMWIR (Right) Pixel Intensity around 65x86 Fire Region over Pre-Processed 2100 Frames (70 seconds)	53
Figure 49: 'oats1' Non-Fire Average (Left), Minimum (Center), and Maximum (Right) CM/CL ratio over Pre-Processed 2100 Frames (70 seconds)	54
Figure 50: 'oats1' difCLCM Process for Frame 1201	56
Figure 51: 'oats1' CL Histogram vs. CM Histogram vs. difCLCM Histogram for Frame 1201	57
Figure 52: 'oats1' difCLCM Frame 1201	58
Figure 53: 'oats1' Sensitivity to Lights in Frame 1800: Taillights (Blue-Dashed), Stationary Light (Blue-Solid).....	59
Figure 54: 'oats1' CL Pixel Intensities in 9x14 Area around the Heat Plume for Frame 1201	60
Figure 55: 'oats1' CL Histogram vs. threshCL Histogram for Frame 1201	60
Figure 56: 'oats1' threshCL Frame 1201	61
Figure 57: 'oats1' adapthreshCL Threshold Value over Time.....	62

Figure 58: 'oats1' CL Histogram vs. adapthreshCL Histogram for Frame 1201	62
Figure 59: 'oats1' adapthreshCL Frame 1201.....	63
Figure 60: 'oats1' CL Histogram for Frame 1201	63
Figure 61: 'oats1' CM histogram for Frame 1201	64
Figure 62: Multi-Temporal Principal Component Analysis Processed Frames.....	68
Figure 63: 'oats1' pcadifCLCM PC1-to-PC10 for Frame 1201.....	69
Figure 64: 'oats1' pcatreshCL PC1-to-PC10 for Frame 1201	70
Figure 65: 'oats1' pcaadapthreshCL PC1-to-PC10 for Frame 1201	71
Figure 66: 'oats1' pcaCL PC1-to-PC10 for Frame 1201	72
Figure 67: 'oats1' pcaCM PC1-to-PC10 for Frame 1201	73
Figure 68: Multi-Spectral Principal Component Analysis Processed Frames	74
Figure 69: 'oats1' pcaCLCM PC1-to-PC2 for Frame 1201	75
Figure 70: 'oats1' pcaTCLCM PC1-to-PC10 for Frame 1201.....	76
Figure 71: 'oats1' difCLCM Surface Plot of Frame 1201	77
Figure 72: 'oats1' difCLCM Gaussian Blur Kernel (sigma = 0.25)	78
Figure 73: 'oats1' difCLCM Gaussian Blurred Frame 1201	79
Figure 74: 'oats1' difCLCM Gaussian Blurred Surface Plot of Frame 1201.....	80
Figure 75: 'oats1' difCLCM 2-D FFT before and after Unsharp Mask Filtering for Frame 1201	81
Figure 76: 'oats1' difCLCM Unsharp Mask Filtered Frame 1201	81
Figure 77: 'oats1' difCLCM Unsharp Mask Filtered Surface Plot of Frame 1201.....	82
Figure 78: 'oats1' difCLCM Non-Zero Pixels before and after Unsharp Mask Filtering for Frame 1201 ..	83
Figure 79: 'oats1' difCLCM Non-Zero Pixels of Unsharp Filtered Frames 1201 to 1230.....	84
Figure 80: 'oats1' difCLCM Above-Horizon Noise Masking for Frame 1201	85
Figure 81: 'oats1' CM Top 97% Pixel Intensity vs. difCLCM Non-Zero Pixels of Unsharp Filtered Frames 1201 to 1230: White Temporal Samples Kept, Green Temporal Samples Discarded from Further Analysis	86
Figure 82: 'oats1' difCLCM Non-Zero Pixels Before and After Input Mask for Frames 1201 to 1230: White Temporal Samples Kept, Green Temporal Samples Discarded from Further Analysis.....	87
Figure 83: 'oats1' difCLCM Input Mask vs. Coefficient of Variation Mask for Frames 1201 to 1230: White Temporal Samples Kept, Green Temporal Samples Discarded from Further Analysis.....	88
Figure 84: 'oats1' difCLCM Coefficient of Variation Mask 11-by-11 Neighbors for Frames 1201 to 1230: White Temporal Samples Kept, Green Temporal Samples Discarded from Further Analysis.....	89
Figure 85: 'oats1' difCLCM Coefficient of Variation Mask 11-by-11 Neighbors vs. Cross Correlation Mask for Frames 1201 to 1230: White Temporal Samples Kept, Green Temporal Samples Discarded from Further Analysis.....	90
Figure 86: 'oats1' difCLCM Cross Correlation Mask 11-by-11 Neighbors for Frames 1201 to 1230: White Temporal Samples Kept, Green Temporal Samples Discarded from Further Analysis	91
Figure 87: 'oats1' difCLCM Input Mask vs. Feature Mask for Frames 1201 to 1230: White Temporal Samples Kept, Green Temporal Samples Discarded from Further Analysis.....	92

Figure 88: 'oats1' difCLCM Feature Mask vs. Target Region for Frames 1201 to 1230: White Temporal Samples within Target Region, Green Temporal Samples outside Target Region.....	93
Figure 89: 'oats1' CM Frame vs. Target Region for Frame 1201	94
Figure 90: 'oats1' difCLCM Temporal Samples Feature Points for Frames 2071-2100 and Frames 1 to 2100, (a)(d)(g)(h)(j)(k) Red = 'Fire' and Blue = 'Noise', (b)(c)(e)(f) Multicolor Individual Signals, (i) White = Positive Target and Green = Negative Target.....	97
Figure 91: Multilayer Perceptron Model [29].....	99
Figure 92: MLP Neural Network Structure	101
Figure 93: Summary of MLP Training Parameters.....	101
Figure 94: 'oats1' Feature Space of Training Samples	102
Figure 95: MLP Training Summary.....	103
Figure 96: MLP Confusion Matrix	103
Figure 97: MLP Response	104
Figure 98: MLP Weights and Bias Plot	104
Figure 99: MLP Epoch Training Error	105
Figure 100: MLP Training State	105
Figure 101: MLP Error Histogram.....	106
Figure 102: MLP Linear Regression.....	106
Figure 103: 'charcoal1' CLWIR (Left) and CMWIR (Right) Min and Max Pixel Intensity Values over Pre-Processed 3990 Frames (133 seconds).....	111
Figure 104: 'charcoal1' CLWIR (Left) and CMWIR (Right) Pixel Intensity around 65x86 Fire Region over Pre-Processed 3990 Frames (133 seconds).....	111
Figure 105: 'charcoal1' Non-Fire Average (Left), Minimum (Center), and Maximum (Right) CM/CL ratio over Pre-Processed 3990 Frames (133 seconds).....	112
Figure 106: 'oats1' difCLCM Classification Space and Classified Pixels, (a)(b) Red = Fire and Blue = Noise, (c)(d)(e)(f) White = Positive Classification and Red/Yellow = Negative Classification	114
Figure 107: 'charcoal1' difCLCM Classification Space and Classified Pixels, (a)(b) Red = Fire and Blue = Noise, (c)(d)(e)(f) White = Positive Classification and Red/Yellow = Negative Classification	115
Figure 108: 'charcoal1' difCLCM Pixel Classification Status over Time	117
Figure 109: 'charcoal1' CL Frame 1261	118
Figure 110: 'charcoal1' difCLCM Frame 1261	118
Figure 111: 'oats1' Registered difCLCM Pixel Classification Status over Time	120
Figure 112: 'oats1' Unregistered difCLCM Pixel Classification Status over Time	121
Figure 113: 'oats1' Unregistered difCLCM Classification Space and Classified Pixels, (a)(b) Red = Fire and Blue = Noise, (c)(d)(e)(f) White = Positive Classification and Red/Yellow = Negative Classification	122
Figure 114: 'charcoal1' difCLCM Pixel Classification Status over Time ($\sigma = 0.255$).....	123
Figure 115: 'charcoal1' difCLCM Pixel Classification Status over Time ($\sigma = 0.245$).....	124
Figure 116: 'charcoal1' difCLCM Surface Plot of Frame 1261	125

Figure 117: 'charcoal1' difCLCM Unsharp Mask Filtered Surface Plot of Frame 1261 ($\sigma = 0.25$) ..	125
Figure 118: 'charcoal1' difCLCM Unsharp Mask Filtered Surface Plot of Frame 1261 ($\sigma = 0.255$)	126
Figure 119: 'charcoal1' difCLCM Unsharp Mask Filtered Surface Plot of Frame 1261 ($\sigma = 0.245$)	126
Figure 120: 'charcoal1' difCLCM Classification Space and Classified Pixels ($\sigma = 0.255$), (a)(b) Red = Fire and Blue = Noise, (c)(d)(e)(f) White = Positive Classification and Red/Yellow = Negative Classification.....	127
Figure 121: 'charcoal1' difCLCM Classification Space and Classified Pixels ($\sigma = 0.245$), (a)(b) Red = Fire and Blue = Noise, (c)(d)(e)(f) White = Positive Classification and Red/Yellow = Negative Classification.....	128

1. INTRODUCTION

1.1 Motivation

Forest fires are an immense burden to the environment, infrastructure, economy, and most importantly, human life around the world [1] [2]. In 2012, the United States spent an excess of \$1.2 billion dollars trying to suppress fires that claimed over 9.3 million acres of land, 2,216 residences, 1,961 outbuildings, and 67 commercial structures [2]. In order to decrease these numbers, early detection is necessary, as forest fires are usually easy to extinguish and cause less destruction at its early stages [3]. This study aims to provide a fast, effective, practical, and computationally efficient approach to detect heat plumes expelled at the initial stages of a forest fire.

1.2 Background

Historically, the most rudimentary method of early forest fire detection has been to employ personnel to look out of a high-vantage-point lookout tower and report activity to the appropriate authorities. Despite being costly and susceptible to human fatigue error, this and other forms of ground/aerial/public surveillance methods are still performed today [4]. With developments in computer vision and advances in sensor technology, it is only logical to desire a more modern and automated solution which features low false alarms and high sensitivity at far ranges.

There are commercial products on the market today that have been successful at detecting hydrocarbon fires (i.e. fires that expel heat, CO₂, and other combustion products in the presence of oxygen and an ignition source) at shorter ranges [5] [6]. Triple IR (IR3) flame detectors analyze three specific wavelength bands in the MWIR region (typically 3 μ -5 μ) to detect hydrocarbon flames [7]. A typical IR3 detector can detect a 1ft² (0.1m²) gasoline pan fire at a range, field of view (FOV), and response time of 50ft-200ft (15m-60m), 90° horizontal/vertical, and 3-10 seconds respectively [8]. This detector's principle of operation is based on infrared absorption spectroscopy of CO₂, which explains its ability to identify unique spectral signatures that hot CO₂ has in the 4.2 μ -4.7 μ wavelengths where peak

emission and absorption occurs [5] [6]. Resulting samples are compared to those obtained by additional sensors sensitive to wavelengths above and below this peak [8]. In the event of a fire, samples registered at the peak wavelengths will be much more intense than samples measured at wavelengths around the peak region [8]. These additional (non-peak) channels also help reduce false alarm by identifying non-flickering high temperature sources and background radiation pertaining to non-fire stimuli [8]. These sensors, however, suffer from range limitations as “CO₂ in the atmosphere attenuates the radiation in the (peak) spectral band” where absorption and emission of radiation occur [7]. In other words, as the distance between the detector and the fire increases, the intensity of that radiation reaching the detector decreases; therefore, increasing CO₂ absorption reduces the probability of detection [8]. Furthermore, high detection performance relies heavily on direct line-of-sight of the radiating active flames [5].

1.3 Context

This thesis is a novel approach in supplement to previous work of related topics conducted under the instruction of Dr. John Saghri. Previous Master’s theses and senior projects have addressed various types of early forest fire detection solutions.

1.3.1 Ryan Radjabi

Ryan Radjabi applied multi-temporal principal component analysis (PCA) to non-overlapping sequences of MWIR video in order to produce a corresponding sequence of temporally-uncorrelated principal component images [9].

1.3.2 Timothy M. Davenport

Timothy M. Davenport expanded upon Ryan’s work by incorporating multi-temporal and multi-spectral PCA to red, green, blue (RGB) and MWIR/LWIR video sequences. Texture analysis of the resulting principal component images then helped classify forest fire smoke and heat plumes [10].

1.3.3 Daniel G. Kohler

Daniel G. Kohler attempted to extract features using statistics and principal component analysis in order to detect the behavior of a growing forest-fire heat plume. Different classification systems were analyzed and performance was assessed using common error metrics [11].

1.3.4 Georges F. Moussa

Georges F. Moussa utilized a hybrid visual/IR spectrum processing approach. Spatial domain filters in the MWIR spectrum were used to detect heat plumes, while motion analysis of multi-temporal principal component images in the visible spectrum helped detect smoke [12].

1.3.5 Ansel Boynton

Ansel Boynton uses a band-pass filter in the LWIR spectrum to identify the flicker frequency of a forest fire's heat plume. This thesis is currently under development at the time of this report; it is set for submission to California Polytechnic State University, San Luis Obispo on June 2013.

1.4 Problem Statement

Fire is defined as the “rapid oxidation of a material in the exothermic chemical process of combustion,” where heat, light, and various other reaction products are released [13]. A heat plume is defined as that heat which is released from a combustion source; it is visible in both the LWIR and MWIR spectrums [7] [14]. This thermal convection is not to be confused with smoke, which is observable in the visible spectrum, especially during the daytime, and is defined as “an aerosol of small particles of carbon and water vapor resulting from the incomplete combustion of carbonaceous materials [13].” The purpose of this study is to design an around the clock computer vision system capable of identifying heat plumes and ultimately raising an alarm when a fire is perceived. The fire core (i.e. active flames) is trivial to detect, so the goal of this study is to detect heat plumes since fire cores may be obstructed from direct line-of-sight in practice and are therefore are not a good early indicator of forest fires.

1.5 Theory

Infrared refers to a broad portion of the electromagnetic spectrum invisible to the human eye. Infrared spreads over a wide range of wavelengths, but only a limited frequency range is considered useful because of mass atmospheric absorption. LWIR is commonly referred to as the “thermal infrared” region where thermal emissions are captured without the need of, or reflections from, external radiation sources [15]. Short-wave IR (SWIR), on the other hand, is referred to as the “reflected infrared” region where reflections from external radiation dominate capture. The MWIR region lies in between that of the “thermal infrared” and “reflected infrared” regions. Sensors tuned to LWIR and MWIR frequencies are often used in applications involving thermal radiation detection [15]. This study is only concerned with the range of wavelengths (or wavenumbers) pertaining to the response bandwidth of the utilized co-located camera’s MWIR and LWIR sensors. Table 1 summarizes the subdivision of the MWIR and LWIR regions.

Table 1: Infrared Spectrum Regions of Interest

	Wavelength (μm)	Wavenumber (cm^{-1})
MWIR	3.8 – 5	2632 - 2000
LWIR	8 – 12	1250 - 833

A sample’s transmittance spectrum is a representation of the fraction of incident radiation passing through the sample over range of frequencies. Figure 1 shows the atmospheric transmittance spectrum with labeling of its major absorbing molecule bands. Highlighted are regions of interest, where dashed-green defines the MWIR region and solid-blue outlines the LWIR region.

By analyzing Figure 1, it is apparent that CO_2 causes the major horizontal atmospheric absorption band in MWIR, while LWIR has no major horizontal absorption bands.

Forest fires primarily emit hot CO_2 at combustion because of the chemical composition of the organic fuel (wood, leaves, etc.) [1] [17]. Figure 2 shows the spectral radiance of an experimental forest fire measured from approximately 7 meters away from the source of combustion; MWIR and LWIR sensor response regions are again highlighted in dashed-green and solid-blue respectively.

Figure 2: Typical Spectral Radiance of an Experimental Forest Fire from ~7m Away: MWIR Sensor Response (Dashed-Green), LWIR Sensor Response (Solid-Blue) [17]

By analyzing Figure 1 and Figure 2, it is obvious that CO_2 is the main forest fire absorption and emission band in the MWIR region [7]. This band appears as two peaks in Figure 2 because there is CO_2 present in both the atmosphere and in bulk amounts emitted by the forest fire [17] [18]. Therefore, CO_2 ultimately contributes to re-emission and an attenuation of the forest fire's photon emittance; thus resulting in a loss of signal dynamics and intensity measured at the MWIR sensor.

Imagery evidence of this effect at short range is displayed in Figure 3.

Figure 3: Comparison of Short Range Atmospheric Absorption between LWIR (Left) and MWIR (Right) Spectral Regions [19]

The person in Figure 3 is holding a cigarette lighter. Here, the excessive hot CO_2 gas produced by the cigarette lighter causes the flame to appear broader in MWIR imagery due to the re-emission of the infrared signal. Instead of capturing the thermal signature directly, as in LWIR imagery, MWIR imagery captures the scattering of infrared light as the thermal energy is absorbed by CO_2 gas. Thus, the difference in contrast between fire-related and other radiation is greater in the LWIR imagery. This effect is further amplified as the distance between the fire and detector increases because additional photon energy is being absorbed along the detection path where atmospheric CO_2 is being introduced [8] [17]. Even at a longer range, however, atmospheric gases do not heavily influence LWIR emissions so heat-plume photon emittance remains largely unaffected [9] [17] [18] [19].

“Two major factors determine how bright an object appears to a thermal imager: the object's temperature and its emissivity” [15]. Emissivity (i.e. the ratio of emitted energy to blackbody-emitted energy at the same temperature) describes a particular materials ability to emit radiation, while the temperature determines the relative energy radiated by that material. By examining the spectral radiant emittance of blackbody radiation, another important insight is gained. Figure 4 plots the spectral radiant emittance and its integration over wavelength (i.e. the integrated spectral radiant photon emittance) for ideal blackbody radiation sources. Regions of interest are again highlighted under the same convention stated previously.

Figure 4: Spectral Radiant Emittance: MWIR Sensor Response (Dashed-Green), LWIR Sensor Response (Solid-Blue) [16]

As deduced from the highlighted areas of Figure 4, given the emissivity, objects near ambient temperature (i.e. landscape, foliage, structures, the horizon, etc.) emit more energy at longer wavelengths. Therefore, a typical forest setting would exhibit more peak emittance in the LWIR band. When imaging “objects of normal earthly temperatures, a MWIR system must be more sensitive than a LWIR system to achieve identical imaging performance” [15]. However, this compensation is often difficult to achieve by nature of the co-located LWIR/MWIR camera settings and electronic limitations. Therefore, LWIR and MWIR imagery must be normalized to radiation not pertaining to fire-related stimuli in order to guarantee valid comparisons between spectral regions. This ensures that, regardless of emissivity, the average signal-intensity representative of near-ambient-temperature blackbody radiation (i.e. the imaging performance of the scene in question) is equal. Linear normalization allows MWIR imagery to retain its increased sensitivity to warmer-than-ambient blackbody radiation not affected by excessive CO₂ absorption (i.e. people, animals, vehicles, aircraft, etc.). While, LWIR imagery would be more sensitive to cooler-than-ambient blackbody radiation, but could still be able to detect hot objects easily (i.e. a forest fire heat plumes) as the total radiation emitted from hotter objects is much greater in this spectral region.

By acknowledging the discussed spectral region differences, it is apparent that digital image processing and computer vision techniques can be used to exploit LWIR’s superior ability to detect forest fire heat plumes and MWIR’s increased sensitivity to other non-fire related sources of thermal radiation.

1.6 Common Operations

1.6.1 Arithmetic Mean, Expected value, or Average (1-Dimensional)

$$\bar{x} = E(x) = \frac{1}{n} \sum_{i=1}^n x_i \quad \text{Equation 1}$$

where, x is a 1-dimensional vector

Example $x = [10 \quad 9 \quad 4 \quad 7 \quad 2]$

$$\bar{x} = 6.4$$

1.6.2 Minimum (1-Dimensional)

$$\min = \min(\mathbf{x})$$

Equation 2

where, $\mathbf{x}$ is a 1-dimensional vector

Example $\mathbf{x} = [10 \ 9 \ 4 \ 7 \ 2]$

$$\min_{\mathbf{x}} = 2$$

1.6.3 Maximum (1-Dimensional)

$$\max = \max(\mathbf{x})$$

Equation 3

where, $\mathbf{x}$ is a 1-dimensional vector

Example $\mathbf{x} = [10 \ 9 \ 4 \ 7 \ 2]$

$$\max_{\mathbf{x}} = 10$$

1.6.4 Median (1-Dimensional)

$$\tilde{x} = \text{median}(\mathbf{x})$$

Equation 4

where, $\mathbf{x}$ is a 1-dimensional vector

Example $\mathbf{x} = [10 \ 9 \ 4 \ 7 \ 2]$

$$\tilde{x} = 7$$

1.6.5 Median Filtering (2-Dimensional)

$$\tilde{\mathbf{A}} = \text{median}_{[m \ n]}(\mathbf{A})$$

Equation 5

where, $\mathbf{A}$ is a 2-dimensional matrix, m, n are odd

$\tilde{\mathbf{A}}_{i,j}$ is the median centered around an $[m \ n]$ neighborhood of symmetrically padded

$$\mathbf{A}_{\text{sym-padded}}_{(i+\frac{m-1}{2}, j+\frac{n-1}{2})}$$

Example $\mathbf{A} = \begin{bmatrix} 1 & 5 & 7 \\ 8 & 10 & 9 \\ 6 & 7 & 9 \end{bmatrix}$ $m = 3, n = 3$

$$\mathbf{A}_{\text{sym-padded}} = \begin{bmatrix} 1 & 1 & 5 & 7 & 7 \\ 1 & 1 & 5 & 7 & 7 \\ 8 & 8 & 10 & 9 & 9 \\ 6 & 6 & 7 & 9 & 9 \\ 6 & 6 & 7 & 9 & 9 \end{bmatrix}, \text{ where } \mathbf{A} \text{ is highlighted}$$

$$\tilde{\mathbf{A}} = \begin{bmatrix} 5 & 7 & 7 \\ 6 & 7 & 9 \\ 7 & 8 & 9 \end{bmatrix}$$

1.6.6 Unbiased Sample Variance (1-Dimensional)

$$\sigma_x^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 \quad \text{Equation 6}$$

where, $\mathbf{x}$ is a 1-dimensional vector

Example $\mathbf{x} = [10 \quad 9 \quad 4 \quad 7 \quad 2]$

$$\sigma_x^2 = 11.3$$

1.6.7 Corrected Sample Standard Deviation (1-Dimensional)

$$\sigma_x = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2} \quad \text{Equation 7}$$

where, $\mathbf{x}$ is a 1-dimensional vector

Example $\mathbf{x} = [10 \quad 9 \quad 4 \quad 7 \quad 2]$

$$\sigma_x = 3.3615$$

1.6.8 Cross Correlation (1-Dimensional)

$$R_{xy}(m) = \begin{cases} \sum_{n=0}^{N-m-1} \mathbf{x}_{(n+m)} \mathbf{y}_n & m \geq 0 \\ \sum_{n=0}^{N+m-1} \mathbf{y}_{(n-m)} \mathbf{x}_n & m < 0 \end{cases} \quad \text{Equation 8}$$

where, $\mathbf{x}$ and $\mathbf{y}$ are 1-dimensional vectors of N-length, $|m| = 0, 1, \dots, N-1$

$$\mathbf{c}_{xy}(\tau) = R_{xy}(\tau - N + 1) \quad \text{Equation 9}$$

where, $\mathbf{c}_{xy}(\tau)$ are cross-correlation coefficients

normalized to 1.0 at zero lag autocorrelations, $\tau = 0, 1, \dots, 2(N-1)$

Example $\mathbf{x} = [10 \quad 9 \quad 4 \quad 7 \quad 2] \quad \mathbf{y} = [2 \quad 7 \quad 4 \quad 9 \quad 10]$

$$\mathbf{R}_{xy} = [100 \quad 180 \quad 161 \quad 212 \quad \overline{182} \quad 92 \quad 65 \quad 28 \quad 4]$$

$$\mathbf{c}_{xy} = [0.4 \quad 0.72 \quad 0.644 \quad 0.848 \quad 0.728 \quad 0.368 \quad 0.26 \quad 0.112 \quad 0.016]$$

1.7 Proposed Early Forest Fire Detection Solution

A co-located CLWIR/CMWIR camera overlooking a scene captured synchronized multispectral infrared video of equal resolution, FOV, and bit depth. After a set of n -frames are accumulated, they are pre-processed to achieve normalization and motion-jitter-eliminating video registration (i.e. video frame image stabilization). Then, pre-processed n -frames corresponding to CMWIR video are arithmetically subtracted from sequential n -frames of CLWIR video in the spatial domain. For effective feature extraction, an unsharp mask (i.e. high frequency emphasizing) filter is applied to the spatial domain of each difference frame. Whereupon a feature mask, applied to the temporal domain of subsequent n -frames, selects specific samples (i.e. pixels over time) that exhibit the dynamic signature of a forest fire's heat plume. Feature points pertaining to the remaining n -frame length temporal-domain samples are then generated. Finally, a multilayer perceptron neural network classifier trained to react to the characteristic “flickering” features of a forest fire classifies resulting feature points. Upon positive forest fire detection, pixels are tagged and an alarm is raised. Figure 5 illustrates the flow diagram of the proposed early forest fire detection solution.

Figure 5: Flow Diagram of Proposed Early Forest Fire Detection Solution

1.8 Thesis Outline

This thesis is structured as follows: Chapter 2 illustrates the processes of data acquisition and pre-processing of video frames. Chapter 3 describes the process of sequential multispectral spatial-domain differencing, as well as other frame processing methods and their relation to previous works. Chapter 4 presents the process of feature extraction and describes temporal-sample selection parameters. Chapter 5 addresses concepts behind the multilayer perceptron neural network classifier, its training method, and training performance. Chapter 6 assesses system performance as contingent upon the frame processing method employed, variation in processing parameters, and classification of dissimilar video. Finally, Chapter 7 delivers concluding remarks and topics concerning future work are presented.

1.9 Implementation Notes

- All thesis work was performed in MATLAB R2012b (64-bit). Appendix A features the source code of all MATLAB scripts, functions, and graphical user interface tools created for this thesis.
- The word “fire” is sometimes used to express a forest fire’s heat plume; this is not to be confused with the fire core (i.e. active flames).
- The word “noise” is often used to reference radiation which is not related to fire-related stimuli.
- Reference to LWIR/MWIR video captured by the CLWIR/CMWIR camera is often referred to as simply CLWIR/CMWIR video.
- Dynamic range is defined as the minimum and maximum intensity levels measured. Tonal range is the span of these measured intensity levels.
- Although data is stored in 16-bit data types, image and video representations of the data are limited to 8-bits. MATLAB uses a linear transformation to maintain the general shape of the histogram, but detail is inevitably lost as 16-bit data type frames are quantized down to 8-bit representations when being displayed. Keep in mind that the actual data used for processing uses 16-bit data types and is independent of the representations being displayed, thus false coloring

and contrast enhancements can be performed on the visual representations without compromising the signal quality being evaluated by the proposed detection system.

- Unless stated otherwise, false coloring and contrast enhancements have been performed to infrared (grayscale) images for easier viewing of low-intensity-level detail.
- The word ‘enhanced’ is often used to refer to the status of a frame subjected to contrast enhancement.
- Two primary color scales are used when displaying 2-dimensional frames. Figure 6 and Figure 7 show how the color shades relate to pixel intensity values. Frames subject to contrast enhancement follow a different pixel-intensity to color-shade relationship. If deemed necessary, the minimum and maximum values of the scale are stated by the display range, similar to Figure 8, shown at the bottom of the presented contrast enhanced frame.

Figure 6: 'Gray' Color Scale

Figure 7: 'Hot' Color Scale

Figure 8: Contrast Enhancement Display Range

2. DATA ACQUISITION

2.1 Controlled Fire Experiment

On June 18, 2011, Raytheon Vision Systems (sponsors of this thesis) conducted a controlled fire experiment in Goleta, California for the purpose of early-forest-fire-detection research. All data used in this thesis was captured during the proceedings of this experiment. Weather conditions were windy and clear.

The experiment site consisted of a video tower (approximately 20 meters high) overlooking a scene in which a 5'x4' barbeque pit (placed 800 meters away) was ignited from 10:11.58 a.m. to 12:12.33 p.m. The video tower had four cameras of various sensor types mounted to it, all of which captured video of the controlled fire experiment. At every update interval, the video was fed to a control room via Ethernet where it was stored to hard drives as raw data. Table 2 summarizes the camera specifications.

Table 2: Controlled Fire Camera Specifications

Spectrum	Visible	Uncooled SWIR	Uncooled LWIR	Cooled LWIR, MWIR
Sensor Resolution	1280 x 1024	640 x 512	640 x 480	640 x 480
Frames Per Second (fps)	10	30	30	30
Pixel Size	8 μm	25 μm	25 μm	20 μm
Focal Length	200 mm	200 mm	75 mm	100 mm
Aperture	F4	-	F1	F2.3, F2.2,
Size of Pixel at Target	3 cm	10 cm	27 cm	-
Wavelength Band	-	-	-	8-12 μ , 3-8.5 μ
Bit Depth	8	8	14	14
Technology	-	-	Bolometer	HgCdTe
Special Features	-	-	-	Co-located

Initially, lighter fluid was used to ignite charcoal and start a low burning fire. Afterward, various fuel types were added to the fire as the recording sessions progressed. Fuels burned included: wild oats,

pine needles, pinecones, palm leaves, thistle, wet and dry leaves, wild rosemary, and apple wood. The fire was then extinguished with water at the conclusion of the experiment. Fuel was added incrementally at intermittent intervals to simulate a growing fire. Active flames rarely made direct line-of-sight with the sensors due to the camera angle and the obstruction caused by the barbeque pit walls and detection path. The temperature of the barbeque pit walls did rise throughout the experiment however.

The experiment generated over 300 GB of raw video. For purposes of this thesis, only video obtained by the co-located, CLWIR/CMWIR camera is used since it covers the spectral regions of interest and records synchronous video using the same lens. Furthermore, this camera suffers from fewer issues relating to noise, sensitivity, range, and cost at increased focal lengths when compared to its uncooled countertypes [20]. **All future mentions will refer to video captured from the co-located CLWIR/CMWIR camera.** Furthermore, the bulk development of this thesis used video frames of burning wild oats, specifically ‘oats1’, since it is a common test bed for previous work [9] [10] [11] [12]. System performance assessments in the results chapter are conducted using ‘charcoal1’ and ‘oats1’ video. Table 3 displays a summary of the CLWIR/CMWIR captured video.

Table 3: Raw Video Sequence Summary

Filename (date_timestamp_fuel)	Sequence	Frames
'2011-06-18_101158_LW-MW-14bit_setup.raw'	'setup1'	3753
'2011-06-18_102125_LW-MW-14bit_charcoal1.raw'	'charcoal1'	4000
'2011-06-18_102454_LW-MW-14bit_oats1.raw'	'oats1'	8000
'2011-06-18_102925_LW-MW-14bit_oats2.raw'	'oats2'	395
'2011-06-18_102949_LW-MW-14bit_oats2.raw'	'oats2b'	8000
'2011-06-18_103820_LW-MW-14bit_oats3.raw'	'oats3'	8000
'2011-06-18_104857_LW-MW-14bit_pine_needles.raw'	'pine_needles'	14934
'2011-06-18_105906_LW-MW-14bit_pine_cones.raw'	'pine_cones'	8000
'2011-06-18_110557_LW-MW-14bit_palm.raw'	'palm'	4405
'2011-06-18_110912_LW-MW-14bit_thistle.raw'	'thistle'	6670
'2011-06-18_111644_LW-MW-14bit_wet_leaves.raw'	'wet_leaves'	6063
'2011-06-18_112306_LW-MW-14bit_wet_dry1.raw'	'wet_dry1'	6117
'2011-06-18_112757_LW-MW-14bit_wet_dry2.raw'	'wet_dry2'	6364
'2011-06-18_113916_LW-MW-14bit_rosemary.raw'	'rosemary'	5609
'2011-06-18_114605_LW-MW-14bit_applewood1.raw'	'applewood1'	6100
'2011-06-18_115209_LW-MW-14bit_applewood2.raw'	'applewood2'	6100
'2011-06-18_121233_LW-MW-14bit_extinguish.raw'	'extinguish'	6100

Figure 9 and Figure 10 shows images of the experimental site from ground level and satellite imagery respectively.

Figure 9: Images at Experiment Site

Figure 10: Satellite Images of the Experiment Site [11]

2.2 Raw Frame Extraction

The CLWIR/CMWIR video is stored as little-endian raw files of unsigned 16-bit integers. The sensor captures frames at a bit depth of 14-bits, so 2-bits of zero padding are added to the least significant bit (LSB) side of each word to satisfy data type requirements. This causes all the data to be logically shifted 2-bits to the left since the camera's 14-bit data is also structured in little-endian order. The dual band bit stream stores CLWIR/CMWIR frames as two images concatenated in the x-dimension. Individual frame sequences are updated in an alternating fashion at a rate of 60 Hz (30 frames per second each). A script formulated by Timothy M. Davenport [10] is used to correct bit-words and extract individual frames from the bit stream. Resulting raw frames are then converted to single-channel (i.e. grayscale) uncompressed 16-bit Tagged Image File Format (TIFF) images that retain image quality and are easier to handle in the MATLAB environment. Figure 11 illustrates the raw CLWIR/CMWIR frame extraction process (false color has not been added, display ranges omitted). **The system described in 1.7 Proposed Early Forest Fire Detection Solution, is set to use an n-frame length set of 30 frames since it corresponds conveniently with the frame rate of the video.**

Figure 11: Raw CLWIR and CMWIR Frame Extraction Process [10]

2.3 Co-located Sensors

2.3.1 Sensor Settings

According to the Raytheon engineers who developed the co-located CLWIR/CMWIR camera, the settings for each sensor are set up differently due to electronic limitations. The MWIR's dynamic range is usually set to lie between approximately 1000 counts above and below the minimum and maximum quantize level (i.e. $0+1000$ to $2^{14}-1-1000$); while, LWIR's dynamic range is set to span 16,380 counts centered upon the desired intensity level usually indicative of near-ambient temperatures. The resulting measured intensity levels also rely on the exposure time of each sensor [20]. Therefore, if not fully calibrated to a typical scene, hotter objects (such as the heat plumes expelled when the fire core is at maximum height) may saturate the sensor when temperatures reach a maximum. During the course of the controlled-fire experiment, the maximum height of the fire core rarely exceeded 1 meter, as seen in Figure 9(d) and Figure 12.

Figure 12: Observed Maximum Fire Core Height of the Controlled Fire Experiment

“An average surface fire on the forest floor might have flames reaching 1 meter in height and can reach temperatures of 800°C ($1,073\text{K}$) or more” [21]. At this point fire detection becomes trivial, so focus should be placed on detection of radiation where sensor-setting dependencies are not an issue. The

flashpoint of forest fuels (i.e. the temperature at which these fuels burst into flames) has been measured to be around 300°C (573K) [21] [22]. Spatial and temporal domain analysis of the measured pixel intensities corresponding to heat plumes exhibiting thermal radiation around the flashpoint of forest fuels reveals no sensor saturation.

Figure 13 illustrates how CLWIR frames of ‘oats1’ appear in the spatial domain at different time instances.

Figure 13: 'oats1' CLWIR Spatial Domain Sensor Response at Frame 1 (Top) and Frame 2600 (Bottom)

Figure 13(a) displays a low burning fire; here there are no signs of saturation because the sensor is operating within its settings limitations and the heat plume temperature is around that of the flashpoint of forest fuels. The fire in Figure 13(b) is very intense and shows a large group of pixels at the core of the heat plume exhibiting possible signs of saturation.

By analyzing the intensity values of these pixels over time, it is possible to make definitive conclusions on how the sensors respond to extreme thermal radiation. Figure 14 shows a 65x86 section of

‘oats1’ pixels around the fire region in CLWIR and CMWIR whose intensities have been plotted over the total length of 8000 frames. (Display ranges for enhanced frames have been omitted.)

Figure 14: ‘oats1’ CLWIR (Left) and CMWIR (Right) Pixel Intensity around Fire Region over Total 8000 Frames (266.67 seconds)

As seen in Figure 14, there is an apparent maximum intensity level coinciding with the maximum intensity levels established by the each sensor’s settings; this is a clear sign of saturation. Early forest fire detection is the primary focus of this thesis; thus, frames exhibiting evidence of saturation are not considered for analysis. In order to avoid processing these frames, an intensity threshold is placed at a level considered safe from the exhibited saturation. Table 4 summaries the measured pixel intensity values and chosen threshold values for the total 8000 frames.

Table 4: Summary of ‘oats1’ CLWIR/CMWIR Pixel Intensities and Saturation Thresholds for Total 8000 Frames

	Overall Intensity			Intensity Around Fire Region			Saturation
	Min	Max	Tonal Range	Min	Max	Tonal Range	Threshold
CLWIR	3,832	14,653	10,822	4,139	14,537	10,399	13,800
CMWIR	4,360	12,409	8,050	5,723	12,409	6,687	11,500

Any frames with pixel intensities exceeding the saturation threshold level are discarded. Figure 15 shows which frames were considered for analysis (i.e. valid frames) and which were not (i.e. invalid frames).

Figure 16 illustrates the relationship between the valid frame numbering and total frame numbering.

All future reference to CLWIR and CMWIR video will relate to frames considered valid (unless specifically labeled).

Figure 15: 'oats1' Frames Considered Valid (Red) and Invalid (Blue)

Figure 16: Relationship between 'oats1' Valid Frame Numbers and Total Frame Numbers

Now that saturated frames have been discarded, Figure 17 shows a section of 'oats1' pixels around the fire region in CLWIR and CMWIR whose intensities have been plotted over the length of 2124 valid frames. (Display ranges for enhanced frames have been omitted.)

Figure 17: ‘oats1’ CLWIR (Left) and CMWIR (Right) Pixel Intensity around 65x86 Fire Region over Valid 2124 Frames (70.8 seconds)

It is clear from Figure 14 and Figure 17 that there is much more activity in LWIR as compared to MWIR in the areas surrounding the fire, regardless of the complications relating to sensor settings. By focusing on the valid set of frames, the ideas stated in 1.5 Theory can be demonstrated over the course of this thesis. An arbitrary set of frames is chosen to illustrate the various processes of this thesis. The frame range highlighted in Figure 18 represents a continuous set of frames from the original 8000 frames of ‘oats1’. Analysis of continuous frames is important for evaluation of temporally sensitive procedures. Formation of these procedures, however, included discontinuous frames for the sake of added immunity to issues relating to dropped frames and/or decaying frame-rates which were troublesome to previous works [10] [11].

Figure 18: Arbitrary Set of ‘oats1’ Frames Chosen to Illustrate Thesis

2.3.2 Sensor Noise

Upon examining Figure 14 and Figure 17, it is apparent that both camera sensors may be exhibiting noise, in particular, salt-and-pepper noise. Figure 19 and Figure 20 show how the overall ‘oats1’ minimum and maximum pixel intensity values vary over the length of 8000 total frames and 2124 valid frames respectively.

Figure 19: 'oats1' CLWIR (Left) and CMWIR (Right) Min and Max Pixel Intensity Values over Total 8000 Frames (266.67 seconds)

Figure 20: 'oats1' CLWIR (Left) and CMWIR (Right) Min and Max Pixel Intensity Values over Valid 2124 Frames (70.8 seconds)

According to Figure 20, there is a sharp drop in the minimum intensity value measured by CMWIR's sensor at frame 2090. Figure 21 and Figure 22 reveal evidence of salt-and-pepper noise in both CLWIR and CMWIR frames at different locations. (Display ranges for enhanced frames are omitted.)

Figure 21: Salt (Blue)-and-Pepper (Green)-Noise in 'oats1' Valid CLWIR Frame 2090

Figure 22: Salt (Blue)-and-Pepper (Green) Noise in 'oats1' Valid CMWIR Frame 2090

In order to properly pre-process these frames, it is necessary to determine the true noise floor independent of pepper noise as it offsets the dynamic range of the measured CLWIR/CMWIR data. To avoid complications associated with pepper noise (seen in Figure 20, Figure 21, and Figure 22), the overall minimum intensity values have to be averaged over the course of 2124 frames. This step essentially eliminates the major instances of pepper noise and is necessary to produce good normalization results. Salt noise is difficult to eliminate since heat plume intensities dominate the majority of the frames where the salt noise is present; this is seen in Figure 20 and Figure 21. Processes such as averaging the maximum value or applying spatial filters at this stage of analysis would degrade heat-plume signal quality since artificial pixel intensity attenuation would be introduced. Table 5 summarizes the determined pixel intensity values over the length of 2124 valid frames.

Table 5: Summary of Determined 'oats1' CLWIR/CMWIR Pixel Intensities over Valid 2124 Frames

	Overall Intensity			Intensity Around Fire Region		
	Min (Averaged)	Max	Tonal Range	Min	Max	Tonal Range
CLWIR	3,886	13,788	9,903	4,140	13,788	9,649
CMWIR	5,510	8,157	2,648	5,723	8,157	2,435

2.4 Pre-Processing

2.4.1 Normalization

As described in 1.5 Theory, CLWIR and CMWIR frames must be normalized in order to make valid comparisons. Figure 23 and Figure 24 display the dissection of CLWIR and CMWIR histograms before normalization. (The display range is equal to the range of plotted histograms.)

Figure 23: 'oats1' CLWIR Frame 1201 Histogram Dissection before Normalization

Figure 24: 'oats1' CMWIR Frame 1201 Histogram Dissection before Normalization

These figures indicate that the respective ‘sub-regions’ of CLWIR and CMWIR histograms are relatively quite similar to each other. Obviously, this relationship must be upheld to form valid comparisons. Normalization to a specific sub-region at a given time, however, may lead to poor normalization results as the temperature of objects within the sub-region’s intensity window are likely to change in a non-uniform manner over time. This could result in a biased “near-ambient-temperature” intensity reading. Therefore, a better approach is to average the measured signal intensity representative of all near-ambient-temperature blackbody radiation within a scene.

Two operations are performed to normalize CLWIR and CMWIR video. First, the pixel-intensity ratio of all near-ambient-temperature blackbody radiation between CMWIR and CLWIR sequential frames (i.e. the CM/CL ratio) is corrected to be 1:1. Then, the resulting histograms are expanded to span the full dynamic range of the 16-bit data type.

2.4.1.1 CM/CL Ratio Correction

Figure 17 designates an area around the fire region, in both spectral regions, where thermal activity is significantly higher than that of near-ambient radiation. In order to calculate the CM/CL ratio, this region of pixels must be discarded from analysis. The resulting region of analysis is referred to as the “non-fire area” and is shown in Figure 25.

Figure 25: 'oats1' CLWIR Non-Fire Area for Frame 1201

Therefore, the CM/CL ratio is established by determining the ratio of CMWIR:CLWIR pixel intensities in the non-fire area for all frames in a video.

Figure 26 shows a plot displaying the 'oats1' non-fire average, minimum, and maximum CM/CL ratio values for the total 8000 frames. Figure 27 displays that for the 2124 valid frames. The solid red line indicates the calculated average value over time.

Figure 26: 'oats1' Non-Fire Average (Left), Minimum (Center), and Maximum (Right) CM/CL ratio over Total 8000 Frames (266.67 seconds)

Figure 27: ‘oats1’ Non-Fire Average (Left), Minimum (Center), and Maximum (Right) CM/CL ratio over Valid 2124 Frames (70.8 seconds)

As seen in Figure 26 and Figure 27, the average CM/CL ratio does not vary much over time. This indicates that there is a linear relationship between the measurement of CMWIR and CLWIR near-ambient-temperature blackbody radiation. Therefore, CM/CL ratio correction is achieved by multiplying each CMWIR frame with its corresponding inverse CM/CL ratio.

The near constant variation of min and max CM/CL ratios is likely attributable to amplifier (Gaussian) noise and/or photon shot noise. These types of noise are very common in imaging systems and are known to affect individual sensor pixels [23]. Meanwhile, the sparse peaks of min and max CM/CL ratios (such as the one present at frame 2090) can be explained by the presence of salt-and-pepper noise (seen in Figure 20 and Figure 21).

2.4.1.2 Dynamic Range Expansion

By analyzing Table 5, it is obvious that tonal range of both CLWIR and CMWIR video is very limited and does not span the full dynamic range of the 16-bit data type. In order to maintain the corrected CM/CL ratio, image quality, and facilitate the viewing of video frames (i.e. not requiring contrast enhancements to individual frames, as done so far), a linear transformation, Equation 10, is used. This equation performs linear mapping of values ranging from [min, max] to [a, b].

$$V_{\text{scaled}} = a + \frac{V - \min}{\max - \min} \times (b - a) \quad \text{Equation 10}$$

where, V is a 3-dimensional matrix

The maximum overall dynamic range, in this case determined from Table 5, is used to expand the dynamic range of both videos to the full dynamic range of the 16-bit data type. This case for example, would have: $\min = 3886$, $\max = 13788$, $a = 0$, and $b = 2^{16} - 1$.

Figure 28 and Figure 29 illustrate each step of the normalization process for a particular frame. Contrast enhancements are purposely omitted in order to display how the near-ambient-temperature radiation intensities have been normalized.

Figure 28: 'oats1' CLWIR Frame 1201 Normalization Process: Before Normalization (Left), After CM/CL Ratio Correction (Center), and After Dynamic Range Expansion (Right)

Figure 29: 'oats1' CMWIR Frame 1201 Normalization Process: Before Normalization (Left), After CM/CL Ratio Correction (Center), and After Dynamic Range Expansion (Right)

Table 6 indicates how various statistic values vary over the process of normalization for a certain frame.

Table 6: 'oats1' Frame 1201 Statistics at Different Stages of Normalization

	Before Normalization		After CM/CL Correction		After Dynamic Range Expansion	
	CLWIR	CMWIR	CLWIR	CMWIR	CLWIR	CMWIR
Average CM/CL Ratio	1.4103		1.0000		0.9793	
Min	3,885	5,541	3,885	3,923	0	285
Max	7,424	7,224	7,424	5,122	23,416	8,180
Tonal Range	3,540	1,684	3,540	1,200	23,417	7,896
Number of Tones	1,198	1,159	1,198	1,102	1,198	1,102

The slight discrepancy between calculated average CM/CL ratios before and after dynamic range expansion, seen in Table 6, can be attributed to the fact that the minimum value of CLWIR is now 0. Divide by zero occurrences must be omitted from the calculation; hence, the resulting average CM/CL ratio value is altered slightly.

2.4.1.3 Normalization Results

Figure 30, Figure 31, Figure 32, Figure 33, and Figure 34 illustrate how the linear normalization process affects the intensity values of CLWIR and CMWIR frames. Notice how the relationships between Figure 23 and Figure 24 are maintained in Figure 31 and Figure 32. Figure 33 indicates that the spectral blackbody sensitivities described in 1.5 Theory are maintained, while the histograms now share a common normalization point. Figure 34 proves that re-emission is causing an attenuation of signal intensity measured at the CMWIR sensor as the pixel-intensity relationship to CLWIR differs at high-intensities.

Figure 30: 'oats1' CLWIR (top) and CMWR (bottom) Frame 1201 Histograms and Intensity Maps

oats1 CLWIR Frame 1201 Histogram Dissection After Normalization

Figure 31: 'oats1 CLWIR Frame 1201 Histogram Dissection after Normalization

Figure 32: 'oats1 CMWIR Frame 1201 Histogram Dissection after Normalization

Figure 33: 'oats1' Frame 1201 Histograms before (top) and after (bottom) Normalization

Figure 34: 'oats1' Frame 1201 Multispectral Scatter Plot

Contrast enhancements are no longer required and will not be performed to display future frames unless specifically stated otherwise.

2.4.2 Video Registration

Wind conditions caused the camera tower to wobble throughout the conducted experiment. This attributed to a motion jitter (i.e. camera shake) noticeable in the viewing of recorded video. Motion jitter corrupts the measured temporal signal of a fixed point in space because its intensity values over time are measured by a group of pixels instead of a single pixel. This presents difficulties when trying to determine the temporal signature of the forest-fire heat plume. Therefore, motion-jitter-eliminating video registration (i.e. video frame stabilization) is deemed necessary at this stage of analysis.

2.4.2.1 Feature Point Matching

The point feature matching method of video registration works without any a priori knowledge of anchor points (i.e. pre-established reference points) within frames. It instead attempts to stabilize video by using Random Sampling and Consensus (RANSAC) to determine the affine transformations between two frames [24]. RANSAC works by correlating a set of inlying observation points to determine an estimated mathematical model in the presence of data outliers. Affine transformations are linear transformations followed by a translation. Motion-jitter-eliminating affine transformations are determined by registering a 30-frame length set of unregistered frames to a reference frame extracted from each set. Once calculated, the registration transformations are applied mutually to CLWIR and CMWIR frames in order to preserve their co-located nature. Proper selection of the spectral video to be registered is important. As mentioned in 1.5 Theory, LWIR is more sensitive to cooler-than-ambient temperatures so the contrast ratio at the horizon would not be very high. MWIR on the other hand is less sensitive to the cooler-than-ambient horizon so the contrast ratio would be higher than that of LWIR. Since this proposed early forest fire detection solution is also intended for night use, the registration transformations must be determined using frames of the highest contrast ratio at the horizon (i.e. CMWIR frames). Figure 35 and Figure 36 show

how normalized CLWIR and CMWIR frames compare in regards to the signal intensity measured at the horizon (additional contrast enhancement is performed to offer the reader a clear distinction).

Figure 35: 'oats1' CLWIR Normalized Frame 1201 (Contrast Enhanced)

Figure 36: 'oats1' CMWIR Normalized Frame 1201 (Contrast Enhanced)

To facilitate the process of determining the affine transformations required for registration, an auxiliary contrast enhancing linear operation (Equation 10) is performed on every set of CMWIR frames

to be registered. This expands each set's dynamic range to the local minimum and maximum intensity levels of the frames; thus, the presence of corners in comparison to the normalized video is amplified. These un-normalized contrast-enhanced frames are not considered in future processing however. They are merely used to facilitate the extraction of salient features and are discarded once the registration transformations have been determined.

The implementation of this algorithm does not assume that the reference frame is the first frame in the video sequence. Making this assumption, could lead to poor video registration results if the conditions of the first frame are not optimal. Instead, a composite reference frame is generated by determining the temporal-domain median intensity value of each pixel in a set of frames to be registered, using Equation 4. Each subsequent unregistered frame in the set is then compared to the composite reference frame to avoid accumulation error. Figure 37 displays an example reference frame composited from a particular set of unregistered CMWIR frames that have been un-normalized for reasons mentioned above. False color is not added.

Figure 37: 'oats1' CMWIR Composite Reference Frame for Frames 1201 to 1230

Salient feature extraction is carried out using the Harris Corner Detection Algorithm. This algorithm attempts to find corners by examining the behavior of edges in 15 x 15 patches [25]. Figure 38

illustrates the process of finding corners in both the reference and unregistered frames. False color is not added to future images in this section so that overlaid points can be easily identified.

Figure 38: 'oats1' CMWIR Salient Features found in Reference Frame and Frame 1201 using the Harris Corner Detection Algorithm

The Sum Squared Differences (SSD) of 15 x 15 neighborhoods centered upon each corner are then used to determine a set a correspondences points between frames. Figure 39 illustrates the path of correspondence points between both frames. Long yellow lines indicate obvious correspondence outliers.

Figure 39: 'oats1' CMWIR Path of Correspondence Points between Reference Frame and Frame 1201

Once a set of correspondence points has been established, RANSAC is used to determine the correspondence inliers that generate the affine transformation required to register each frame to the reference frame. Only a single RANSAC trial is used in the presence of such sparse correspondence outliers. Figure 40 exhibits the RANSAC determined correspondence inliers between a reference frame and an unregistered frame.

Figure 40: 'oats1' CMWIR RANSAC Determined Correspondance Inliers for Frame 1201

Once the registering affine transformations for each unregistered frame in the set have been calculated, they are mutually applied to corresponding normalized CLWIR and CMWIR frames. This process is then repeated for all sets of 30-frames constituting a video.

2.4.2.2 Video Registration Results

By analyzing the variance of each pixel in a set of frames in regards to time, using Equation 6, it is possible to evaluate the performance of the video registration process. Figure 41 and Figure 42 describe how edges vary over a span of time as result of motion jitter. They have been contrast enhanced to local frame minimum and maximum values.

Figure 41: 'oats1' Variance of CMWIR Unregistered Frames 1201 to 1230

Figure 42: 'oats1' Variance of CMWIR Registered Frames 1201 to 1230

Judging from Figure 42 it is obvious that the video registration process has done a good job at removing motion jitter in this set of frames since the variance of edge locations has been significantly reduced over time. Unfortunately, there is evidence of registration noise near bottom edge of Figure 42, which suggests

a variation of frame border locations over time. This is an expected byproduct of applying affine transformations, which must be accounted for when analyzing registered video.

Another expected byproduct of applying affine transformations is that the MATLAB estimated geometric transformation model uses linear interpolation to designate intensities between known pixel values. Because the normalization process expands the dynamic range of the data to 16-bits, there are unused tones which interpolation is able fill. This results in an increase of the number of tones within a frame. Figure 43 and Figure 44 display the outcome of the aforementioned pixel intensity interpolation.

Figure 43: 'oats1' CL Registration Histograms for Frame 1201

Figure 44: 'oats1' CL Tones due to Registration

2.5 Pre-Processing Results

Figure 45, Figure 46, Figure 47, Figure 48, and Figure 49 illustrate the results of pre-processing. As deduced from these figures, the CLWIR and CMWIR frames can now be compared directly as the signal intensities for near-ambient-temperature radiation now span over a common dynamic range. Furthermore, the noise floor is now better defined and overall sensor sensitivity has been compensated for.

oats1 Pre-Processed CLWIR Frame Number 1201

Figure 45: 'oats1' Pre-Processed CLWIR Frame Number 1201

oats1 Pre-Processed CMWIR Frame 1201

Figure 46: 'oats1' Pre-Processed CMWIR Frame Number 1201

Figure 47: 'oats1' CLWIR (Left) and CMWIR (Right) Min and Max Pixel Intensity Values over Pre-Processed 2100 Frames (70 seconds)

oats1 65x86 Pre-Processed CLWIR Pixels Around Fire Region (Contrast Enhanced)

oats1 65x86 Pre-Processed CMWIR Pixels Around Fire Region (Contrast Enhanced)

Figure 48: 'oats1' CLWIR (Left) and CMWIR (Right) Pixel Intensity around 65x86 Fire Region over Pre-Processed 2100 Frames (70 seconds)

Figure 49: ‘oats1’ Non-Fire Average (Left), Minimum (Center), and Maximum (Right) CM/CL ratio over Pre-Processed 2100 Frames (70 seconds)

All pre-processed CLWIR and CMWIR frames will be referred to as CL and CM frames respectively from this point forward.

3. FRAME PROCESSING

3.1 Spatial-Domain Sequential Frame Processing Methods

This thesis proposes a novel early-forest-fire-detection approach based on the exploitation of differences between LWIR and MWIR spectral regions. Other spatial-domain processing techniques thought to be effective at isolating regions of interest are presented, but future chapters will mainly reference frames processed by the multispectral difference method since it is the subject of the thesis presented. Analysis of unprocessed frames was also conducted as a way to establish baseline system performance. The results section of this thesis will address the performance of processing techniques alternative to the multispectral difference method.

3.1.1 Multispectral Difference

3.1.1.1 *difCLCM*

The multispectral difference process, referred to as *difCLCM*, involves the simple arithmetic difference between sequential CL and CM frames. By subtracting CM frames from a set of CL frames, infrared spectral region differences can be revealed and thus exploited. This proves to be a computationally efficient and surprisingly effective technique. It is important to note that this method requires CLWIR and CMWIR frames to be pre-processed in order to preserve heat-plume data and provide concurrent noise attenuation. Figure 50 illustrates the process for a given frame within a set to be processed.

Figure 50: 'oats1' difCLCM Process for Frame 1201

Examination into the resulting difCLCM histograms reveals that although non-fire radiation (i.e. the detection of radiation not related to forest-fire heat plumes) is greatly attenuated, there still exists evidence of low-intensity-level non-fire radiation signals that must be accounted for in order to improve the efficiency of the detection system. Figure 51 shows the difCLCM histogram for a given frame, while Figure 52 displays a contrast-enhanced view of that frame.

Figure 51: 'oats1' CL Histogram vs. CM Histogram vs. difCLCM Histogram for Frame 1201

Figure 52: 'oats1' difCLCM Frame 1201

The LWIR spectral region is more sensitive than MWIR when detecting light-sourced radiation such as headlights, building lights, radio tower lights, etc. Therefore, stationary and moving light sources show up strongly in difCLCM frames. This is an important observation, which must be accounted for when attempting to detect forest-fire heat plumes. Figure 53 displays difCLCM's sensitivity to light-emitting radiation sources. In this example, the taillights of a moving semi-truck are identified by dashed blue circles while a stationary light on top of a building is highlighted by solid blue circles.

Figure 53: 'oats1' Sensitivity to Lights in Frame 1800: Taillights (Blue-Dashed), Stationary Light (Blue-Solid)

3.1.2 Thresholding

Thresholding operations are very common when attempting to isolate pixels exhibiting specific intensity values. Two forms of thresholding methods are investigated: static and adaptive. They are applied to CL frames since these frames are largely unaffected by atmospheric gas absorption and therefore better exhibit the dynamic characteristics of a forest fire heat plumes.

3.1.2.1 *threshCL*

The static thresholding method, known as *threshCL*, requires user-defined threshold limits to be placed on the frames pending processing. Pixels with intensity levels above and below this range are

given a zero-value. A threshold limit of [4500, 65535] is chosen since the 'oats1' heat plume over time seems to exhibit pixel intensities within this range, see Figure 48 and Figure 54. The resulting threshCL histogram for a particular frame is shown in Figure 55. While Figure 56 displays how the CL frame appears after static threshold processing.

Figure 54: 'oats1' CL Pixel Intensities in 9x14 Area around the Heat Plume for Frame 1201

Figure 55: 'oats1' CL Histogram vs. threshCL Histogram for Frame 1201

Figure 56: 'oats1' threshCL Frame 1201

3.1.2.2 *adapthreshCL*

The static thresholding limits do not account for variations in pixel intensity over time. Photon shot noise or progressive temperature fluctuations, for example, could result in the detection of more non-fire radiation over time. This may lead to even poorer heat-plume-isolation results. An adaptive threshold method is proposed which attempts to isolate the ‘fourth sub-region’ of CL’s histogram (see Figure 31). An algorithm analyzes a set of CL-frame histograms and determines the location of the minimum threshold value. The threshold range is then set to be [threshold, 65535]. Figure 57 presents the calculated threshold value for all sets of 30-frame CL images comprising a video. The resulting adapthreshCL histogram for a particular frame is shown in Figure 58. While Figure 59 displays how the CL frame appears after adaptive threshold processing.

Figure 57: 'oats1' adapthreshCL Threshold Value over Time

Figure 58: 'oats1' CL Histogram vs. adapthreshCL Histogram for Frame 1201

Figure 59: 'oats1' adapthreshCL Frame 1201

3.1.3 Unprocessed

3.1.3.1 CL

Figure 60 shows the histogram of a particular CL frame (see Figure 45).

Figure 60: 'oats1' CL Histogram for Frame 1201

3.1.3.2 CM

Figure 61 shows the histogram of a particular CM frame (see Figure 46).

Figure 61: 'oats1' CM histogram for Frame 1201

3.2 Principal Component Analysis Processing Methods

Principal Component Analysis (PCA) uses orthogonal transformations to reduce high-dimensional data to lower dimensional, linearly uncorrelated, representations known as principal components (PC). PCA is an eigenvector-based multivariate analysis method that reveals the internal structures of data in regards to its variability. Traditionally, it is recognized as a popular feature extraction method. The number of principal components which can be generated must be less than or equal to the number of original variables. These generations are structured in such a way that the first principal component exhibits the largest variance possible while succeeding components are constrained to be orthogonal to each preceding component. Once the principal components of frames have been calculated, they transformed from the principal component space back to the original frame space for further analysis. It is important to note that principal components contain no natural order so interpretation of the principal components is user-defined [26]. PCA was investigated deeply in previous forest fire detection

works [9] [10] [11] [12]; however, it is used in this thesis as a method to evaluate past frame processing techniques under the newly proposed system.

3.2.1 Principal Component Analysis Computation Procedure

Principal component analysis maps z-score normalized data from the input space, $\mathbf{X}$, to the principal component space, $\mathbf{Y}$, using the principal component transform $\mathbf{W}$. The PCA transformation that provides the maximum number of principal components is defined as:

$$\mathbf{Y}^T = \mathbf{W}\mathbf{X}^T$$

Equation 11

where, $\mathbf{X}$ and $\mathbf{Y}$ are of dimensions $[M, N]$ and $\mathbf{W}$ is of dimensions $[N, N]$

M denotes the number of observations (i.e. number of pixels in the vectored frames being analyzed), while N denotes the number of variables (i.e. number of frames analyzed) for each PCA computation. $\mathbf{W}$ is the matrix of eigenvectors of the covariance matrix $\mathbf{X}^T\mathbf{X}$, ordered by ascending variance.

To perform the PCA calculation,

1) Construct the $\mathbf{V}$ matrix

Let $\mathbf{F}$ denote a frame

$$\mathbf{F} = \begin{bmatrix} F_{11} & \cdots & F_{1c} \\ \vdots & \ddots & \vdots \\ F_{r1} & \cdots & F_{rc} \end{bmatrix}$$

Equation 12

where, $\mathbf{F}$ is of dimensions $[r, c]$

The vectored frame $\vec{\mathbf{F}}$ becomes,

$$\vec{\mathbf{F}} = \text{reshape}(\mathbf{F}, r * c, 1)$$

$$\vec{\mathbf{F}} = [F_{11}, F_{21}, \dots, F_{r1}, F_{12}, F_{22}, \dots, F_{r2}, F_{1c}, F_{2c}, \dots, F_{rc}]^T$$

Equation 13

where, $\vec{\mathbf{F}}$ is of dimensions $[M, 1]$

$\mathbf{V}$ therefore becomes,

$$\mathbf{V} = [\vec{\mathbf{F}}_1 \quad \vec{\mathbf{F}}_2 \quad \dots \quad \vec{\mathbf{F}}_N]$$

Equation 14

where, $\mathbf{V}$ is of dimensions $[M, N]$

2) Generate the $\mathbf{X}$ matrix

$$\mathbf{X} = \frac{\mathbf{V} - \bar{\mathbf{V}}}{\sigma_V} \quad \text{Equation 15}$$

3) Calculate the unbiased covariance matrix of $\mathbf{X}$

$$\boldsymbol{\Sigma} = \frac{1}{N-1} E[(\mathbf{X} - E(\mathbf{X}))^T (\mathbf{X} - E(\mathbf{X}))] \quad \text{Equation 16}$$

where, $\boldsymbol{\Sigma}$ is of dimensions $[N, N]$

4) Find the matrix eigenvectors of the covariance matrix of $\mathbf{X}$ in ascending order

$$\boldsymbol{\Sigma} \mathbf{v} = \lambda \mathbf{v} \quad \text{Equation 17}$$

where, λ is the eigenvalue of $\boldsymbol{\Sigma}$ corresponding to $\mathbf{v}$; $\mathbf{v}$ is of dimensions $[N, 1]$

$$\mathbf{W} = [\mathbf{v}_{\lambda_1} \quad \mathbf{v}_{\lambda_2} \quad \dots \quad \mathbf{v}_{\lambda_N}] \quad \text{Equation 18}$$

5) Calculate the principal components score matrix $\mathbf{Y}$

$$\mathbf{Y} = \mathbf{X} \mathbf{W}^T \quad \text{Equation 19}$$

6) Project $\mathbf{Y}$ to the original frame space

Since principal components have no natural order, only their relative magnitude variation is of concern. $\mathbf{Y}$ can be projected back to the original frame space without intensity mapping and/or additional image processing by use of Equation 20 and Equation 21.

Note: Previous works corrected negative scores (i.e. negative PCs coefficients) using methods deemed appropriate by the respective author [10] [11] [12].

$$\mathbf{Y}_V = \sigma_V \cdot \text{abs}(\mathbf{Y}) \quad \text{Equation 20}$$

$$\mathbf{Y}_V = [\overrightarrow{PC_N} \quad \overrightarrow{PC_{N-1}} \quad \dots \quad \overrightarrow{PC_1}] \quad \text{Equation 21}$$

7) Reshape PC vectors in $\mathbf{Y}_V$ to the original frame dimensions

$$PC_{n-Frame} = \text{reshape}(\overrightarrow{PC_n}, r, c)$$

Equation 22

Example

$$F_1 = \begin{bmatrix} 58712 & 36522 & 11730 \\ 6494 & 50625 & 22213 \\ 2894 & 20443 & 13772 \end{bmatrix}$$

$$F_2 = \begin{bmatrix} 22497 & 47879 & 26101 \\ 61343 & 42367 & 49140 \\ 8177 & 54601 & 54736 \end{bmatrix}$$

$$V = \begin{bmatrix} 58712 & 22497 \\ 6494 & 61343 \\ 2894 & 8133 \\ 36522 & 47879 \\ 50625 & 42367 \\ 20443 & 54601 \\ 11730 & 26101 \\ 22213 & 49140 \\ 13772 & 54736 \end{bmatrix}$$

$$\sigma_V = [19645 \quad 17833]$$

$$X = \begin{bmatrix} 1.7251 & -1.0241 \\ -0.9330 & 1.1542 \\ -1.1162 & -1.8271 \\ 0.5955 & 0.3992 \\ 1.3134 & 0.0901 \\ -0.229 & 0.7761 \\ -0.6665 & -0.8220 \\ -0.1328 & 0.4699 \\ -0.5625 & 0.7837 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 4.2882 & -4.2882 \\ -4.2882 & 4.2882 \end{bmatrix}$$

$$W = \begin{bmatrix} -0.7071 & -0.7071 \\ -0.7071 & 0.7071 \end{bmatrix}$$

$$Y = \begin{bmatrix} -0.4956 & -1.9440 \\ -0.1564 & 1.4759 \\ 2.0813 & -0.5027 \\ -0.7034 & -0.1388 \\ -0.9924 & -0.8650 \\ -0.3912 & 0.7065 \\ 1.0525 & -0.1100 \\ -0.2383 & 0.4262 \\ -0.1564 & 0.9519 \end{bmatrix}$$

$$Y_V = \begin{bmatrix} 9737 & 34667 \\ 3073 & 26319 \\ 40887 & 8964 \\ 13818 & 2476 \\ 19497 & 15426 \\ 7685 & 12598 \\ 20677 & 1962 \\ 4682 & 7601 \\ 3073 & 16976 \end{bmatrix}$$

$$PC_{1-Frame} = \begin{bmatrix} 34667 & 2476 & 1962 \\ 26319 & 15426 & 7601 \\ 8964 & 12598 & 16976 \end{bmatrix}$$

$$PC_{2-Frame} = \begin{bmatrix} 9373 & 13818 & 20677 \\ 3073 & 19497 & 4682 \\ 40887 & 7685 & 3073 \end{bmatrix}$$

3.2.2 Multi-Temporal Principal Component Analysis

Multi-temporal principal component analysis is the frame processing method predominately used in previous works [9] [11] [12]. This thesis aims to show that PCA frame processing methods are relatively costly, in terms of computation time, and may not be advantageous as the multispectral difference method allows for direct and efficient identification of heat plume dynamic characteristics.

The process of multi-temporal PCA involves selecting a t -number of frames within an n -frame set and calculating t -number of principal components representing pixel intensity observations over time. At every processing interval, the PCA processing buffer starts at the beginning of the n -frame set, where it performs t -PC calculations. Upon generating t -PC images, it shifts over one frame in time and repeats the calculation process until $t * (n - t + 1)$ total PC images have been generated. After a new set of n -frames has been accumulated, the process is repeated. Figure 62 graphically illustrates the process.

Figure 62: Multi-Temporal Principal Component Analysis Processed Frames

A selected $t = 10$ number of PCs are generated at each processing interval since it is a convenient divisor of this 30-frame set application. Analysis of the resulting principal component images reveals that PC2 images possessed the best suppression of non-fire-related radiation while simultaneously preserving dynamic heat plume signals; thus, only PC2 images are considered for further evaluation. The following subsections display all the PC images and histograms generated for each spatial-domain processing method as applied to a specific frame. Each PC image has been individually contrast enhanced to local minimum and maximum intensity values for ease of viewing.

3.2.2.1 *pcadifCLCM*

Figure 63: 'oats1' *pcadifCLCM* PC1-to-PC10 for Frame 1201

3.2.2.2 *pca*threshCL

Figure 64: 'oats1' *pca*threshCL PC1-to-PC10 for Frame 1201

3.2.2.3 *pcaadaptthreshCL*

Figure 65: 'oats1' *pcaadaptthreshCL* PC1-to-PC10 for Frame 1201

3.2.2.4 *pcaCL*

Figure 66: 'oats1' *pcaCL* PC1-to-PC10 for Frame 1201

3.2.2.5 *pcaCM*

Figure 67: 'oats1' *pcaCM* PC1-to-PC10 for Frame 1201

3.2.3 Multi-Spectral Principal Component Analysis

Multi-spectral principal component analysis involves selecting sequential s -number of spectral frames from s -sets of n -frame length videos. At each computation interval, s -PC images are generated for every n -frames analyzed. The process is complete when $s * n$ PC images have been generated. Upon accumulating a new n -frame set, the process is repeated. Figure 68 illustrates the process.

Figure 68: Multi-Spectral Principal Component Analysis Processed Frames

For the purposes of this study $s = 2$ since only CL and CM, spectral videos are considered. Analysis of the resulting principal component images reveals that again, PC2 images possessed the best noise suppression and preservation of dynamic heat plume signals. The following subsection displays both PC images and histograms for a particular frame, but only PC2 images are considered for further evaluation. Individual PC images have been contrast enhanced to full scale as previously stated.

3.2.3.1 *pcaCLCM*

Figure 69: 'oats1' *pcaCLCM* PC1-to-PC2 for Frame 1201

3.2.4 Multi-Spectral-Temporal Principal Component Analysis

Multi-spectral-temporal principal component analysis is a hybrid PCA approach. It requires multi-spectral video to first be processed by multi-spectral PCA, and then resulting PC images of interest are processed again using multi-temporal PCA [10]. The following subsection displays all 10 PC generated images and histograms for a particular frame, but again only PC2 frames are of interest. They have been contrast enhanced exactly like the previously shown PC images.

3.2.4.1 *pcaTCLCM*

Figure 70: 'oats1' *pcaTCLCM* PC1-to-PC10 for Frame 1201

4. FEATURE EXTRACTION

4.1 Spatial-Domain Unsharp Mask Filter

The frame processing method (i.e. difCLCM) described in 3.1.1 Multispectral Difference vastly attenuates non-fire-related radiation, but as witnessed in Figure 52, it does not fully eliminate it. In order to reduce classification time, it is necessary to filter these frames and further isolate areas of interest.

Figure 71 demonstrates the surface plot of Figure 52.

Figure 71: 'oats1' difCLCM Surface Plot of Frame 1201

Figure 71 indicates the presence of large amounts of spatial-domain low-frequency noise pertaining to radiation stemming from the sky and near-by metal (thermally active) rooftops. The heat plume signal, on the other hand, exhibits fluctuating peaks and is surrounded by strong edges caused from the spectral difference process. Edges and sharp fluctuations contain high-frequency components; therefore, a high-pass filter can be used to eliminate large portions of the remaining non-fire-related radiation.

Unsharp mask filtering, or unsharp masking as it is also commonly known, emphasizes high-frequency components in 2-dimensional images. This filtering process is typically conducted in the spatial domain, as opposed to the frequency domain, by use of kernel convolution. Application of kernel-based filters reduces computation time and is more efficient than performing Fourier transforms; therefore, unsharp mask filtering is the method of choice for isolating spatial-domain high-frequency components in this system.

First, a Gaussian kernel is convolved with the original image to create a blurred (i.e. unsharp) version of the image. This essentially low-pass filters the image since the Fourier transform of a spatial-domain 2-dimensional Gaussian signal results in a 2-dimensional frequency-domain Gaussian signal. Therefore, spatial kernel convolution essentially locates a low-pass cut-off frequency in the frequency domain of the image being filtered. The sigma value of the Gaussian kernel determines the bandwidth of the low-pass filter; thus, it acts as sensitivity setting for the system. Figure 72 displays the Gaussian kernel and sigma value used to blur all difCLCM frames.

Figure 72: 'oats1' difCLCM Gaussian Blur Kernel (sigma = 0.25)

The equation for a 2-dimensional Gaussian function is defined as:

$$f(x, y) = A \left(- \left(\frac{(x-x_o)^2}{2\sigma_x^2} + \frac{(y-y_o)^2}{2\sigma_y^2} \right) \right) \quad \text{Equation 23}$$

where, A is amplitude

x_o, y_o designates the center location

σ_x, σ_y defines the standard deviation in the x, y direction

$\sigma_x = \sigma_y$ in Gaussian kernels used for image processing

Once the low-frequency components have been isolated, the blurred image can be used to reveal the high-frequency components of the original image. Figure 73 and Figure 74 display the blurred versions of Figure 52 and Figure 71 respectively. Analyzing these figures, it is apparent that the Gaussian blurred versions differ very subtly from their original counterparts. This ensures that the majority of the non-fire radiation can be removed from the original difCLCM frame.

Figure 73: 'oats1' difCLCM Gaussian Blurred Frame 1201

Figure 74: 'oats1' difCLCM Gaussian Blurred Surface Plot of Frame 1201

The simple process of subtracting the Gaussian blurred image from the original image emphasizes high-frequency components of interest. The strength of the unsharp mask filter, or the percentage of blurred image subtracted from the original image, is user-definable. For this application, the filter strength is set to maximum as heat plumes mainly exhibit high-frequency components in the spatial domain. Figure 76 and Figure 77 display the results of the subtraction (i.e. masking) operation. Figure 75 compares the spatial-domain Fast Fourier Transforms (FFT) of Figure 52 and Figure 73.

Figure 75: 'oats1' difCLCM 2-D FFT before and after Unsharp Mask Filtering for Frame 1201

Figure 76: 'oats1' difCLCM Unsharp Mask Filtered Frame 1201

Figure 77: 'oats1' difCLCM Unsharp Mask Filtered Surface Plot of Frame 1201

By analyzing Figure 71, Figure 77, it is apparent that the overall height of the peaks (i.e. pixel intensity of high-frequency components) is drastically reduced, but practically all the low-frequency non-fire radiation has been removed. Figure 75 supports these observations. Furthermore, the peaks representing the fire's heat plume and possible heat plume candidates are still present and maintain relative relationships to each other. This means that the regions of interest have been successfully isolated and will therefore aid in reducing classification time. Figure 78 displays the amount of pixels, irrespective of intensity, that have been discarded from further classification due to unsharp mask filtering.

Figure 78: 'oats1' difCLCM Non-Zero Pixels before and after Unsharp Mask Filtering for Frame 1201

4.2 Temporal Domain Feature Masking

Temporal-domain feature masking helps isolate samples that exhibit forest-fire heat plume dynamic characteristics. As discussed in 2.3.2 Sensor Noise and 2.4.2.2 Video Registration Results, the pre-processed frames contain evidence of salt noise and affine transformation border-edge artifacts. Frame processing and unsharp mask filtering alone cannot eliminate these types of noise. Furthermore, as discussed in 3.1.1.1 difCLCM, moving and stationary lights show up strongly in processed frames so they must be identified in a computer vision sense in order to avoid false alarm. Given these issues, it is necessary to perform temporal-domain masking procedures in order to ensure a good sampling pool for feature extraction. Three types of masks are applied in series to a given set of processed frames with the goal of selecting temporal samples (i.e. $1 \times 1 \times 30$ temporal-domain pixel vectors) that exhibit the typical spatial relationships and “flickering” characteristics of a forest-fire heat plume.

4.2.1 Input Masking

Neither the frame edges, nor the low-intensity area above the horizon could possibly produce forest-fire-sourced thermal radiation. Input masking eliminates samples that originate from the sky and frame borders. Figure 79 displays the non-zero pixels in a set of unsharp mask filtered frames. Both

above-horizon noise (along the top of the frame) and border-edge noise (along the bottom edge of the frame) is present.

Figure 79: 'oats1' difCLCM Non-Zero Pixels of Unsharp Filtered Frames 1201 to 1230

The first step in applying the input mask is to eliminate the above-horizon noise. As mentioned in 2.4.2.1 Feature Point Matching, CM frames have the largest contrast ratio at the horizon. Therefore, these frames will serve as a reference to identify the sky (i.e. low-intensity area above the horizon). Figure 32 shows that the sky is represented by the ‘first sub-region’ in the CM frame histogram (Figure 61). So, any salt noise, described in 2.3.2 Sensor Noise, is first eliminated in CM frames by applying a 5-by-5 median filter (Equation 5); then, a simple algorithm is implemented which locates the largest peaks in the CM frameset histogram and calculates a threshold at their average value. This essentially identifies the ‘first sub-region’ of the CM frames where it is impossible to have fire-related activity. Ultimately, all the

above-horizon temporal samples in Figure 79, which lie within the bottom percentage of the CM frameset histogram, are masked out from further analysis. Figure 80 illustrates how the process affects individual frames within a set. While Figure 81 shows an overlay of how this first step in the input masking process affects the temporal samples; green samples are discarded and white samples are kept for further analysis.

Figure 80: 'oats1' difCLCM Above-Horizon Noise Masking for Frame 1201

oats1 CM Top 97% vs. difCLCM Non-Zero Pixels of Unsharp Filtered Frames 1201 to 1230

Figure 81: 'oats1' CM Top 97% Pixel Intensity vs. difCLCM Non-Zero Pixels of Unsharp Filtered Frames 1201 to 1230: White Temporal Samples Kept, Green Temporal Samples Discarded from Further Analysis

The second step in the process of applying the so-called input mask is to eliminate the noise resulting from the video registration process. This edge-noise is easily accounted for by analyzing the affine transformed frames and identifying the location of the defined border edges. The maximum width over time of these zero-intensity borders is calculated and masked off from a given frameset. Figure 82 shows a composite image how input masking affects the temporal samples to be analyzed. **This and all proceeding composite images in the chapter will show white pixels corresponding to temporal samples being kept for further analysis and green pixels designates those being discarded.**

oats1 difCLCM Non-zero Pixels Before Input Mask vs. Non-zero Pixels After Input Mask for Frames 1201 to 1230

Figure 82: 'oats1' difCLCM Non-Zero Pixels Before and After Input Mask for Frames 1201 to 1230: White Temporal Samples Kept, Green Temporal Samples Discarded from Further Analysis

4.2.2 Coefficient of Variation Masking

The coefficient of variation is a statistical normalizing function defined as:

$$c_v = \frac{\partial x}{\bar{x}} \quad \text{Equation 24}$$

where, $\mathbf{x}$ is a 1-dimensional vector

The forest fire heat plume temporal domain signals (i.e. pixel intensity values over time) typically have large average and standard deviation values. By promoting these types of signals without placing a limit on overall intensity, the number of false-candidate samples is reduced. This process not only masks out non-flickering light sources, but it also promotes flickering radiation of relatively small intensity that may be indicative of a heat plume expelled from a very small fire. Spatial relationships between samples are

also taken into account as heat plume radiation is likely grouped together or in close proximity to each other. Hence, samples that do not have any neighbors in a user-defined neighborhood are discarded from consideration. Figure 83 and Figure 84 illustrate the process of masking temporal samples that do not possess the coefficient of variation temporal-spatial characteristics of a forest fire.

oats1 difCLCM Input Mask vs Coefficient of Variation Mask for Frames 1201 to 1230

Figure 83: 'oats1' difCLCM Input Mask vs. Coefficient of Variation Mask for Frames 1201 to 1230: White Temporal Samples Kept, Green Temporal Samples Discarded from Further Analysis

oats1 difCLCM Coefficient of Variation Mask 11-by-11 Neighbors for Frames 1201 to 1230

Figure 84: 'oats1' difCLCM Coefficient of Variation Mask 11-by-11 Neighbors for Frames 1201 to 1230: White Temporal Samples Kept, Green Temporal Samples Discarded from Further Analysis

4.2.3 Cross Correlation Masking

Moving lights have strongly correlated temporal signals; therefore, they can be identified by performing cross correlation (Equation 8) and comparing subsequent coefficient values (Equation 9). Parameters and thresholds are defined so that temporal samples of certain maximum cross correlation coefficient values within a designated lag window are kept and others are eliminated. Again, of the remaining samples, only samples of close proximity to each other are kept for future consideration. Figure 85 and Figure 86 display the results of cross-correlation masking. Finally, Figure 86 summarizes the feature masking process.

oats1 difCLCM Coefficient of Variation Mask 11-by-11 Neighbors vs. Cross Correlation Mask for Frames 1201 to 1230

Figure 85: 'oats1' difCLCM Coefficient of Variation Mask 11-by-11 Neighbors vs. Cross Correlation Mask for Frames 1201 to 1230: White Temporal Samples Kept, Green Temporal Samples Discarded from Further Analysis

oats1 difCLCM Cross Correlation Mask 11-by-11 Neighbors for Frames 1201 to 1230

Figure 86: 'oats1' difCLCM Cross Correlation Mask 11-by-11 Neighbors for Frames 1201 to 1230: White Temporal Samples Kept, Green Temporal Samples Discarded from Further Analysis

oats1 difCLCM Input Mask vs. Feature Mask for Frames 1201 to 1230

Figure 87: 'oats1' difCLCM Input Mask vs. Feature Mask for Frames 1201 to 1230: White Temporal Samples Kept, Green Temporal Samples Discarded from Further Analysis

4.3 Feature Point Generation

Once the temporal samples of interested have been selected, they are analyzed and calculations are produced based on the features defined. The results of the calculations are called feature points. These measures describe analytical characteristics of the temporal signals in question. Effective feature extraction is a very important part of this thesis as is essential for good classification performance, as “the performance of machine learning methods is heavily dependent on the choice of data representation (or features) on which they are applied” [27].

By defining a target region (i.e. a region where known fire related activity exists), labels are associated with the computed feature points of temporal signals. Subsequent labels are necessary for

neural network training and classification-performance assessment. Figure 88 displays the defined target region, all samples within this region are labeled as ‘fire’ (i.e. temporal domain samples corresponding to forest fire heat plumes), while all areas outside of the target region is labeled ‘noise’ (i.e. temporal domain samples stemming from non-fire-related radiation). For spatial orientation sake, Figure 89 offers the reader a contrast enhanced overlay of the target region and a reference frame.

oats1 difCLCM Feature Mask vs. Target Region for Frames 1201 to 1230

Figure 88: 'oats1' difCLCM Feature Mask vs. Target Region for Frames 1201 to 1230: White Temporal Samples within Target Region, Green Temporal Samples outside Target Region

oats1 CM Frame vs. Target Region for Frame 1201

Figure 89: 'oats1' CM Frame vs. Target Region for Frame 1201

4.3.1 Feature Selection

By analyzing the behavior of temporal samples labeled ‘fire’ and ‘noise’, three features are chosen to help distinguish non-fire-related radiation from “flickering” heat plume signals. These features promote samples that are in close proximity to each other as this is a common property of heat plumes. Therefore, as a fire grows and its spatial imprint becomes more noticeable, the likelihood of it being detected increases.

4.3.1.1 Neighborhood Standard Deviation

The neighborhood standard deviation value for each considered sample is calculated by summing the standard deviation values (using Equation 7) of all the temporal samples within an n-by-n neighborhood of the sample's location.

4.3.1.2 Neighborhood Mean

The neighborhood mean value for each considered sample is calculated by summing the mean values (using Equation 1) of all the temporal samples within an n-by-n neighborhood of the sample's location.

4.3.1.3 Neighborhood Average Auto Correlation

Auto correlation is a special case of cross correlation (Equation 8 and Equation 9) where $x = y$. The neighborhood average auto-correlation value for each considered sample is calculated by using Equation 1 to averaging the sum of the auto correlation coefficients (Equation 8 and Equation 9) of all the temporal samples within an n-by-n neighborhood of the sample's location.

4.3.2 Feature Points

Figure 90(a) displays the temporal samples for a set of 30-frame length frames before feature masking, where red signals are 'fire' (i.e. heat plume radiation samples) and blue signals are 'noise' (i.e. non-fire-related radiation samples). Figure 90(b) and Figure 90(c) show isolated 'fire' and 'noise' signals, respectively, before feature masking for a given set of frames, where colors represent individual samples. Figure 90(d) displays the temporal samples after feature masking for a set of frames, where red signals 'fire' and blue signals are 'noise'. Figure 90(e) and Figure 90(f) show isolated 'fire' and 'noise' signals after feature masking for a given set of frames, where colors represent individual samples. Figure 90(g)(h)(j) illustrate cross sections of the 3-dimensional feature space and the locations of the calculated feature points, where red points identify a 'fire' and blue points identify 'noise'. Figure 90(k) shows an accumulated view of feature space and the computed feature points for all 30-frame length sets of the video; red points identify 'fire' and blue points identify 'noise'. Lastly, Figure 90(i) presents an overlay of

the feature-masked samples and target samples, where white pixels identify samples within the target region (positive targets) and green pixels identify samples outside the region (negative targets).

Figure 90(k) indicates that the selection of features is effective at isolating ‘fire’ and ‘noise’ samples without loss of generalization. Therefore, these feature points can be used as inputs into a non-linear separating classifier such as a multilayer perceptron neural network. The support vector machine (SVM) classifier is a popular and simpler alternative to the multilayer perceptron classifier, which operates on the same basic principle of creating a non-linear divider between two classes of inputs. However, due to multilayer perceptron’s ability to “learn” relationships not strictly enforced (i.e. extrapolate relationships between ambiguous classes) [27], multilayer perceptron is the machine learning classifier of choice.

4.4 Feature Extraction Parameters

The various feature extraction parameters used in the development of this chapter are presented in Table 7.

Table 7: 'oats1' difCLCM Feature Extraction Parameters

Parameter	Value
Gaussian Kernel Filter Size	25 x 25
Gaussian Kernel Sigma (difCLCM)	0.25
Unsharp Mask Filter Strength	100%
Maximum Coefficient of Variation Limit	3.5
Maximum Cross Correlation Coefficient Limit	0.8
Cross Correlation Lag Window	5
Feature Masking ‘Closest-Neighbor’ Neighborhood	11 x 11
Feature Point Neighborhood	25 x 25

5. CLASSIFICATION

5.1 Multilayer Perceptron Neural Network

A multilayer perceptron (MLP) neural network is a type of feed-forward artificial neural network that consists of multiple layers of interconnected processing units (i.e. neurons). Usually three or more layers (an input, output, and one or more hidden layers) are used to map sets of input data onto a set of desired outputs [28]. Figure 91 shows an MLP model with N neurons in the input layer, S neurons in the hidden layer, and L neurons in the output layer.

Figure 91: Multilayer Perceptron Model [29]

The output of the network can be expressed as:

$$y_k^o = f_k^o \left(b_k^o + \sum_{i=1}^S w_{ik}^o y_i^h \right)$$

$$y_k^o = f_k^o \left(b_k^o + \sum_{i=1}^S w_{ik}^o f_i^h \left(b_i^h + \sum_{j=1}^N w_{ji}^h x_j \right) \right)$$

where, $k = 1, \dots, L$

Equation 25

h denotes the elements in the hidden layer.

o denotes the elements in the output layer.

w_{ji}^h is the weight connecting neuron j of the input layer to neuron i of the hidden layer.

w_{ik}^o is the weight connecting neuron i of the hidden layer to neuron k of the output layer.

f_i^h is the activation function of neuron i of the hidden layer.

f_k^o is the activation function of neuron k of the output layer.

b_i^h is the bias of neuron i of the hidden layer.

b_k^o is the bias of neuron k of the output layer.

y_i^h is the output of neuron i of the hidden layer.

x_j is the input of neuron j of the input layer.

Hence, the excitation levels are:

$$n_i^h = b_i^h + \sum_{j=1}^N w_{ji}^h x_j$$
Equation 26

$$n_k^o = b_k^o + \sum_{i=1}^S w_{ik}^o y_i^h$$
Equation 27

For this application, the hyperbolic tangent activation function is used; it is defined as:

$$f(n) = \tanh(n)$$
Equation 28

where, $f(n)$ ranges from -1 to 1 for neuron i

Given the well compartmentalization of the feature points, an MLP neural network structure of 3-inputs, 10-neurons in a single hidden layer, and 1-output is chosen. This structure is shown graphically in Figure 92.

Figure 92: MLP Neural Network Structure

5.2 Training

Supervised batch learning was used to train the MLP neural network; this process uses a set of inputs and corresponding desired outputs to adjust the weights connecting neurons and bias values at each layer. The Levenberg-Marquardt algorithm is used to perform the back-propagation weight and bias adjustment. This algorithm interpolates between the Gauss–Newton algorithm (GNA) and, the often standard, method of gradient descent [30]. This algorithm was chosen based on its fast convergence speed and MATLAB efficiency [31]. Figure 93 summarizes the defined training parameters.

Figure 93: Summary of MLP Training Parameters

The feature points generated from the ‘oats1’ difCLCM, threshCL, adapthreshCL, and CL processed frames were used for training. These contributed 4,712, 133, 4,810, and 2,170 feature points respectively. Inclusion of the feature points generated from the alternative spatial-domain processing techniques ensures that no bias is placed on the proposed multispectral difference method. The feature points generated from CM and PCA-variant processed frames, however, were not included. CM frames

generated very few feature points due to the attenuated dynamic behavior of its heat plume signals. Meanwhile, because of dimensionality reduction, PCA methods seem to alter the temporal signature in a manner that caused either erratic or poorly compartmentalized feature points. Figure 94 displays the feature points considered for training.

Figure 94: 'oats1' Feature Space of Training Samples

Of the 11,825 feature points generated, 50% were selected at random for use in training, while 25% were used in validation, and the remaining 25% for testing.

5.3 Training Performance

Figure 95, Figure 96, Figure 97, Figure 98, Figure 99, Figure 100, Figure 101, and Figure 102 show a variety of training performance assessments for the proposed MLP classifier. Figure 95 reveals that it only took 8 seconds to train the network, thus the selection of the back-propagation algorithm benefits the system as any desired additional training would not result in prolonged system downtime. Figure 96 shows the confusion matrix of the neural network when classifying the considered feature

points. Figure 97 states the neural network's inability to produce a true binary output. Therefore, simple binary thresholding (Equation 29) must be performed to output '1's and '0's.

$$y = \begin{cases} 1, & x \geq 0.5 \\ 0, & x < 0.5 \end{cases} \quad \text{Equation 29}$$

Figure 98 illustrates the interconnecting weights and bias values. Figure 99 shows how the error metric (mean squared error) diminishes over training epochs. Figure 100 describes the internal state of the back-propagation algorithm over training epochs. Figure 101 displays the error histogram. While finally, Figure 102 plots the linear regression.

Figure 95: MLP Training Summary

Figure 96: MLP Confusion Matrix

Figure 97: MLP Response

Figure 98: MLP Weights and Bias Plot

Figure 99: MLP Epoch Training Error

Figure 100: MLP Training State

Figure 101: MLP Error Histogram

Figure 102: MLP Linear Regression

6. RESULTS

6.1 Frame Processing Performance

Frame processing performance is assessed by examining the results of the MLP neural network classifier when presented with the task of classifying feature points extracted from the ‘oats1’ video sequence whose frames are subject to various processing techniques. The ‘oats1’ video is used to make this assessment because the classifier is familiar with feature points extracted from this video sequence; therefore, the evaluation is an indicator of the system’s optimal performance. As mentioned in 1.8 Thesis Outline, the earliest successful detection time, detection error, and computation time will be used to determine which frame processing method performs best.

6.1.1 Earliest Successful Heat Plume Detection

The ‘oats1’ video sequence continuously displays a visible heat plume throughout the entirety of the 70-second (i.e. 2100 frames) video. Since the system described in 1.7 Proposed Early Forest Fire Detection Solution operates on 30-frame sets, the ideal earliest successful detection time should be 1-second (i.e. after processing 30 frames). Table 8 summarizes the earliest successful detection time for each of the frame-processing methods considered. The Gaussian-kernel sigma value presented is the sensitivity setting found to yield the best results for each of the frame processing techniques being evaluated.

Table 8: 'oats1' Earliest Successful Heat Plume Detection

Process	Gaussian Kernel Sigma	Successful Detection Time (seconds)
difCLCM	0.25	1
threshCL	0.218	19
adapthreshCL	0.218	6
CL	0.24	1
CM	0.24	-
pcadifCLCM	0.216	1
pcathreshCL	0.218	25
pcaadapthreshCL	0.218	20
pcaCL	0.24	1
pcaCM	0.24	-
pcaCLCM	0.24	1
pcaTCLCM	0.25	1

6.1.2 Classification Error

The second assessment on frame processing performance is based on classification error. For every 1-second (i.e. 30 frames) of video analyzed, the system is expected to identify the presence of a heat plume within the target region. False-negatives and/or undetected fire-related radiation within the target region will constitute an error. Similarly, because this study is also concerned with maintaining low false-alarm, any detected false-positives (i.e. falsely identified radiation outside the target region) will also contribute to the error count. Table 9 displays the results of the classification error assessment. It is important to note that since detection is based on a per-pixel over time basis, partial detection of the heat plume within the target region will also produce a false-negative error.

Table 9: 'oats1' Number of 1-Second Interval Classification Errors over 70-seconds

Process	Detection Failure	Detection Failure Error	False-Negatives	False-Negative Error	False-Positives	False-Positive Error	Error Sum
difCLCM	0	0.0000	0	0.0000	0	0.0000	0.0000
threshCL	37	0.5286	15	0.2143	0	0.0000	0.7429
adapthreshCL	25	0.3571	20	0.2857	0	0.0000	0.6428
CL	0	0.0000	2	0.0286	5	0.0714	0.1000
CM	64	0.9143	6	0.0857	0	0.0000	1.0000
pcadifCLCM	1	0.0143	3	0.0429	1	0.0143	0.0715
pcathreshCL	39	0.5571	20	0.2857	0	0.0000	0.8428
pcaadapthreshCL	4	0.0571	43	0.6142	1	0.0143	0.6856
pcaCL	9	0.1286	1	0.0143	1	0.0143	0.1572
pcaCM	63	0.9000	7	0.1000	0	0.0000	1.0000
pcaCLCM	2	0.0286	2	0.0286	0	0.0000	0.0572
pcaTCLCM	5	0.0714	3	0.0429	1	0.0143	0.1286

6.1.3 Computation Time

The third frame processing performance criteria will compare the computation time necessary to process a given set of 30-frames (i.e. 1-second of video). All plotting and write to disk operations were suppressed with background processes halted and memory cleared before the commencement of each trial. Processing occurred on the same MATLAB platform and machine. The computer system specifications are summarized in Table 10. CL and CM frames are not subject to additional frame processing; therefore, they are omitted from this assessment. To provide context, computation time of the

pre-processing stage and feature extraction/classification stages for the same set of 30 frames is listed in Table 11. **The frame processing assessment assumes that frame sets have been pre-processed with motion-jitter-eliminating video registration.** Results of the computation-time performance assessment are shown in Table 12.

Table 10: Computer System Specifications

CPU	Intel Core i7 920 @ 2.67 GHz
Memory	6.00 GB
Hard Drive	Western Digital Passport 1TB USB 3.0
MATLAB	R2012b, 64-bit
Windows	Windows 7 Ultimate SP1, 64-bit

Table 11: 'oats1' Computation Time to Pre-Process and Extract Features/Classify difCLCM Frames 1201 to 1230

		Computation Time (Seconds)
Pre-Processing	Registered Video	32.535651
	Unregistered Video	0.471618
Feature Extraction and Classification		2.316689

Table 12: 'oats1' Computation Time to Process Frames 1201 to 1230

Process	Computation Time (seconds)
difCLCM	0.208992
threshCL	0.342265
adapthreshCL	1.768455
CL	-
CM	-
pcadifCLCM	7.493135
pcathreshCL	7.553252
pcaadapthreshCL	7.771184
pcaCL	7.488735
pcaCM	7.48570
pcaCLCM	5.286425
pcaTCLCM	12.438535

6.2 Dissimilar Video Performance

As seen in 6.1 Frame Processing Performance, the difCLCM method outperforms the frame processing methods considered and results excellent classification of the ‘oats1’ video. This is partly associated with the fact the MLP neural network classifier was trained with an estimated 50% of feature points extracted from this video sequence. However, it is also associated with the fact that the heat plume signal-intensity levels are relatively high and can be easily identified by the human eye. Upon examining the other video sequences captured from the controlled-fire experiment, it was evident that the heat plume signals in all the videos, except for ‘charcoal1’, exhibited the same relative intensity as they did in ‘oats1’. The ‘charcoal1’ video sequence was dissimilar to the others as the heat plume was extremely difficult to identify with the human eye. As mentioned in 2.1 Controlled Fire Experiment, ‘charcoal1’ was the initial fire that was used to ignite the organic fuel. Early detection is key; therefore, it was essential to process this video through the proposed system since it more accurately portrays the earliest stages of a fire where thermal activity is low and the capture of heat plume radiation is difficult to identify. Figure 103, Figure 104, and Figure 105 summarize the analysis of ‘charcoal1’ pre-processed frames.

Figure 103: 'charcoal1' CLWIR (Left) and CMWIR (Right) Min and Max Pixel Intensity Values over Pre-Processed 3990 Frames (133 seconds)

charcoal1 65x86 Pre-Processed CLWIR Pixels Around Fire Region (Contrast Enhanced)

charcoal1 65x86 Pre-Processed CMWIR Pixels Around Fire Region (Contrast Enhanced)

Figure 104: 'charcoal1' CLWIR (Left) and CMWIR (Right) Pixel Intensity around 65x86 Fire Region over Pre-Processed 3990 Frames (133 seconds)

Figure 105: ‘charcoal1’ Non-Fire Average (Left), Minimum (Center), and Maximum (Right) CM/CL ratio over Pre-Processed 3990 Frames (133 seconds)

By comparing Figure 103 and Figure 104 to Figure 47 and Figure 48 respectively, it can be shown that there are significantly less high-intensity temporal-domain signals in ‘charcoal1’ than in ‘oats1’. Furthermore, upon examining the ‘charcoal1’ temporal-domain signals that exhibit the highest intensity, it is apparent that the dominant sources of radiation in this video are not exclusive to heat-plume thermal activity, but rather predominantly stem from non-fire-related radiation (i.e. the detection of aircraft and taillight lighting in this case). Therefore, trivial detection methods, such as the triggering to high pixel-intensity levels would be ineffective before the approximate 1500 frame (i.e. 50-second) mark and concurrently erroneous thereafter as the radiation of non-fire-related stimuli dominates the video sequence.

Figure 106 and Figure 107 display the classified feature points in the classification space and the per-pixel over time classification status. Figure 106(a) and Figure 107(a) show the classification space over the length of the entire video. Figure 106(b) and Figure 107(b) display the classification space over a specific 30-frame length set. In these subfigures, red crosses indicate the presence of known heat plume radiation (i.e. ‘known fire’) while blue points indicate known non-fire-related radiation (i.e. ‘noise’). Red circles indicate the detection of heat plume radiation (i.e. ‘classified fire’) while blue squares indicate the classification of ‘noise’ (i.e. ‘classified noise’). Figure 106(c)(d)(e)(f) and Figure 107(c)(d)(e)(f) illustrate the per-pixel classification over 30-frame status in the spatial domains, both at full-frame and at the zoomed in 65x86 area around the fire region for a specific frame. Bright white pixels in these subfigures indicate that the MLP neural network classifier has identified temporal samples at particular locations that

pertain to heat plumes (i.e. positive classification). While red-orange pixels indicate that such temporal samples have been evaluated, but the heat plume classification output result was negative (i.e. negative classification). The red scale at the top right corner of these subfigures represents the length first truck (from left to right) directly behind the barbeque pit fire (20 pixels). The current frame number is also displayed at the top left of these subfigures.

Figure 106: 'oats1' difCLCM Classification Space and Classified Pixels, (a)(b) Red = Fire and Blue = Noise, (c)(d)(e)(f) White = Positive Classification and Red/Yellow = Negative Classification

Figure 107: 'charcoal1' difCLCM Classification Space and Classified Pixels, (a)(b) Red = Fire and Blue = Noise, (c)(d)(e)(f) White = Positive Classification and Red/Yellow = Negative Classification

The classification space reveals that feature points (and hence classified points) in the ‘oats1’ video are much better separated than feature points in the ‘charcoal1’ video. The fire in ‘charcoal1’ is so low that its heat plume intensity fluctuations are similar to that of other noise in the frame; therefore, the classifier cannot correctly classify the temporal samples at these difficult to distinguish stages. There is successful positive classification of heat plume radiation over time however, which is encouraging given the degree of difficulty for human observers to spot thermal activity in the area. Furthermore, as Figure 107(f) indicates, the heat plume rises high above the barbeque pit even though the active flames at this stage are very well hidden behind the cool barbeque pit walls. This is encouraging as actual forest fires may be obscured by the environment at its earliest stages.

Figure 108 shows the ‘charcoal1’ heat plume classification over time.

Figure 108: 'charcoal1' difCLCM Pixel Classification Status over Time

As gathered from analysis of Figure 108, there is evidence that portions of the heat plume, even at its earliest stages, are identified by the system. However, these evaluated feature points are merely not closely enough related to the heat plume feature points used in training to be correctly classified. The first positive heat plume detection occurs at the interval between 42 and 43 seconds, which is well before the trivial trigger levels seen in Figure 104.

Figure 109 and Figure 110 display the CL and difCLCM frame of the first positively classified heat plume in 'charcoal1'.

charcoal1 CL Frame 1261

Figure 109: 'charcoal1' CL Frame 1261

Figure 110: 'charcoal1' difCLCM Frame 1261

When compared to the response time of the human observer, the proposed system's heat-plume detection-time is surprisingly fast as it is extremely difficult to see any noticeable thermal activity in the target area even when false color and contrast enhancement is added to the frames.

6.3 Parameter Variation Performance

It is important to analyze how the system can be adjusted to produce improved detection results. All the parameters listed in Table 7 as well as the length of n-frame sets discussed in 1.7 Proposed Early Forest Fire Detection Solution, are available for fine-tuning. The evaluation of system performance upon the adjustment of two key system parameters is presented.

6.3.1 Video Registration

The motion-jitter-eliminating video registration (i.e. video stabilization) process is a very time-consuming process, as witnessed in Table 11. This system parameter therefore is likely inhibit real-time execution of the heat-plume detection program in its current state. Therefore, it is of interest to see how the MLP neural network classifier would handle unregistered video sequences.

By comparing Figure 111 and Figure 112, it is apparent that the 'oats1' heat plume temporal samples are still detected in all 1-second intervals of the video, however, there are added false-positives in the unregistered video classification stemming from fixed lighting radiation (seen in Figure 53).

Figure 111: 'oats1' Registered difCLCM Pixel Classification Status over Time

Figure 112: 'oats1' Unregistered difCLCM Pixel Classification Status over Time

If it were possible to eliminate these false positives, the pre-processing stage would be tremendously quicker (as Table 11 shows). This system would therefore be likely to perform in real-time applications with relatively modest processing power. As mentioned in 2.4.2.1 Feature Point Matching and 3.1.1.1 difCLCM, stationary lights are an issue in LWIR imagery. The vigorous motion jitter causes the radiation signal to be repeatedly detected by a group pixels; this can be interpreted as “artificial flickering” as the signal exhibits intense fluctuations in response to the motion jitter. Figure 113, however, suggests that the feature-space of unregistered frames still exhibits a high degree of compartmentalization. This is encouraging as it may be possible to take additional measures in future work to ensure good system performance regardless of motion jitter.

Figure 113: 'oats1' Unregistered difCLCM Classification Space and Classified Pixels, (a)(b) Red = Fire and Blue = Noise, (c)(d)(e)(f) White = Positive Classification and Red/Yellow = Negative Classification

6.3.2 Gaussian Kernel Sigma Value

As discussed in 4.1 Spatial-Domain Unsharp Mask Filter, the sigma value of the Gaussian kernel can be used to define system sensitivity. As more lower-frequency components are considered, the chance of detecting “low-flickering” heat plumes increases. This also however, results in an increased detection of false-positives as more noise is being classified which may resemble the feature points of interest. Conversely, the sensitivity could be lowered to exhibit more immunity to noise. Figure 114 and Figure 115 show the how adjusting the Gaussian kernel sigma value affects system sensitivity.

Figure 114: 'charcoal1' difCLCM Pixel Classification Status over Time (sigma = 0.255)

Figure 115: 'charcoal1' difCLCM Pixel Classification Status over Time (sigma = 0.245)

Evidence of how varying the Gaussian-kernel sigma value influences individual frames can be deduced by comparing Figure 117, Figure 118, and Figure 119. It is important to recall that the 'oats1' sigma value used when training the MLP neural network classifier with difCLCM feature points was 0.25. Figure 120 and Figure 121 show how the Gaussian sigma value alters the feature point locations in the classification space when compared to Figure 107 where the parameter was set to 0.25.

Figure 116: 'charcoal1' difCLCM Surface Plot of Frame 1261

Figure 117: 'charcoal1' difCLCM Unsharp Mask Filtered Surface Plot of Frame 1261 (sigma = 0.25)

Figure 118: 'charcoal1' difCLCM Unsharp Mask Filtered Surface Plot of Frame 1261 (sigma = 0.255)

Figure 119: 'charcoal1' difCLCM Unsharp Mask Filtered Surface Plot of Frame 1261 (sigma = 0.245)

Figure 120: 'charcoal1' difCLCM Classification Space and Classified Pixels (sigma = 0.255), (a)(b) Red = Fire and Blue = Noise, (c)(d)(e)(f) White = Positive Classification and Red/Yellow = Negative Classification

Figure 121: 'charcoal1' difCLCM Classification Space and Classified Pixels (sigma - 0.245), (a)(b) Red = Fire and Blue = Noise, (c)(d)(e)(f) White = Positive Classification and Red/Yellow = Negative Classification

7. CONCLUSION

7.1 Concluding Remarks

The system proposed in this thesis is very promising. It is based on a simple, yet elegant idea that outperforms expectations with its sheer ability to respond quickly and efficiently to sensor detection of forest-fire heat plume radiation. By exploiting the differences between long-wave infrared and mid-wave infrared spectral regions, it may be possible to operate this system in real-time given some platform and system optimizations. Furthermore, evidence of system performance suggests that this method of detection may one day be suitable for field deployment, given its effectiveness at long range, high sensitivity, and low false-alarm rate. Early detection of forest fires is essential to avoid mass destruction. The ideas presented in this thesis may help advance the development of future systems and produce a solution beneficial to a worldwide cause.

7.2 Future Work

Future work consists of optimizing the system to perform in real-time. Given the limited set of training data used, this system performed remarkably well, but by further training the neural network classifier with diverse training sets and adding inputs for various system parameters, it may be possible to eliminate issues related to motion jitter and improve overall classification results. In addition, the porting of MATLAB code to a compiled language is likely to reap benefits in execution time and in its feasibility to be integrated into an embedded system.

Further experimentation is needed in order to assess the full effective operational-range of the system. Nighttime and varying climate capture of controlled-fire data is also required to further evaluate and optimize system performance.

BIBLIOGRAPHY

- [1] S. P. Urbanski, W. M. Hao and S. Baker, "Chemical Composition of Wildland Fire Emissions," in *Developments in Environmental Science*, vol. 8, A. Bytnerowicz, M. Arbaugh, A. Riebau and C. Andersen, Eds., Elsevier B.V., 2009, pp. 79-107.
- [2] National Interagency Fire Center, "Fire Information, Statistics," 2012. [Online]. Available: http://www.nifc.gov/fireInfo/fireInfo_statistics.html. [Accessed 2013].
- [3] J. San-Miguel-Ayanz, N. Ravail, V. Kelha and A. Ollero, "Active Fire Detection for Fire Emergency Management: Potential and Limitations for the Operational Use of Remote Sensing," *Natural Hazards*, vol. 35, pp. 361-376, 2005.
- [4] D. Schroeder, "Evaluation of Three Wildfire Smoke Detection Systems," *Advantage*, vol. 5, no. 24, 2004.
- [5] Electronstandart Pribor, "Fire Detection Introduction," [Online]. Available: <http://electronstandart-pribor.com/shared/docs/pres.pdf>. [Accessed 2013].
- [6] Sierra Monitor Corporation, "Model 3600I Triple IR (IR3) Flame Detector," 2010.
- [7] Scott/Bacharach Gas Detection Products, "Flame Vision Model FV-30 Flame Detector," Exton.
- [8] Spectrex Inc., "SharpEye Triple IR (IR3) Flame Detector," Cedar Grove, 2004.
- [9] R. Radjabi, J. A. Saghri and J. T. Jacobs, "Early Forest Fire Detection using Principal Component Analysis of Infrared Video," in *SPIE 8135, Applications of Digital Image Processing XXXIV*, San Diego, 2011.
- [10] T. M. Davenport, "Early Forest Fire Detection using Texture Analysis of Principal Components from Multispectral Video," California Polytechnic State University, San Luis Obispo, 2012.

- [11] D. G. Kohler, "Study of Statistical and Computational Intelligence Methods of Detecting Temporal Signature of Forest Fire Heat Plume from Single-band Ground-Based Infrared Video," California Polytechnic State University, San Luis Obispo, 2012.
- [12] G. F. Moussa, "Early Forest Fire Detection using Texture, Blob Threshold, and Motion Analysis of Principal Components," California Polytechnic State University, San Luis Obispo, 2012.
- [13] National Wildfire Coordinating Group, "Glossary of Wildland Fire Terminology," 2012.
- [14] S. Č. Bajić, "Fusion and Visualisation of the Color and Long Wave Infrared Images," Polytechnic of Zagreb, Department of Informatics and Computing, Zagreb.
- [15] C. Douglas, "IR Spectral Bands and Performance," FLIR, Arlington.
- [16] Raytheon, "Raytheon Vision Systems - The Infrared Wall Chart," [Online]. Available: <http://www.randfoo.com/2010/01/modtran/>. [Accessed 2013].
- [17] S. Briz, A. d. Castro, J. Aranda, J. Melendez and F. Lopez, "Reduction of false alarm rate in automatic forest fire infrared surveillance systems," *Remote Sensing of Environment*, vol. 86, pp. 19-29, 2003.
- [18] S. D. Gunapala, J. K. Liu, J. M. Mumolo, D. Z. Ting, C. J. Hill and J. Nguyen, "Demonstration of Megapixel Dual-band QWIP Focal Plane Array," Jet Propulsion Laboratory, Pasadena.
- [19] S. D. Gunapala, S. V. Bandara, C. J. Hill, J. K. Liu, J. M. Mumolo and S. A. Keo, "Pixel co-registered simultaneous dualband infrared sensing," in *IEEE SENSORS 2008 Conference*, 2008.
- [20] D. A. Richards and P. Boulanger, "Cooled versus uncooled cameras for long range surveillance," FLIR.
- [21] B. Gabbert, "At what temperature does a forest fire burn?," *Wildfire Today*, 26 February 2011. [Online]. Available: <http://wildfiretoday.com/2011/02/26/at-what-temperature-does-a-forest-fire-burn/>. [Accessed 2013].

- [22] G. C. Kaminski, "Ignition Time vs. Temperature for Selected Forest Fuels," San Dimas Equipment Development Center , November 1974. [Online]. Available: <http://www.fs.fed.us/t-d/programs/fire/spark/html/74511201/74511201.html>. [Accessed 2013].
- [23] R. Westervelt, "Imaging Infrared Detectors II," The MITRE Corporation, Mclean, Virginia, 2000.
- [24] Mathworks, "Computer Vision System Toolbox - Video Stabilization Using Point Feature Matching," [Online]. Available: http://www.mathworks.com/products/computer-vision/examples.html?file=/products/demos/shipping/vision/videostabilize_pm.html.
- [25] C. Harris and M. Stephens, "A Combined Corner and Edge Detector," The Plessey Company plc., 1988.
- [26] H. Abdi and L. J. Williams, WIREs Computational Statistics, vol. 2, John Wiley & Son s, In c., 2010, p. 433–459.
- [27] Y. Bengio, A. Courville and P. Vincent, "Representation Learning: A Review and New Perspectives," Department of computer science and operations research, U. Montreal.
- [28] S. Haykin, Neural Networks and Learning Machines, Pearson Education Inc., 2009.
- [29] E. Agirre-Basurko, G. Ibarra-Berastegi and I. Madariagac, "Regression and multilayer perceptron-based models to forecast hourly O3 and NO2 levels in the Bilbao area," *Science Direct*, vol. 21, no. 4, pp. 430-446, 2006.
- [30] D. W. Marquart, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," *Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431-441, 1963.
- [31] MathWorks, "Function Approximation and Nonlinear Regression-Levenberg-Marquardt backpropagation," [Online]. Available: <http://www.mathworks.com/help/nnet/ref/trainlm.html>.

GLOSSARY

Term/Acronym	Description
CL	Pre-processed CLWIR Video Frames
CLWIR	Cooled Long-Wave Infrared
CM	Pre-processed CMWIR Video Frames
CM/CL Ratio	Pixel-Intensity Ratio of Near-Ambient-Temperature Blackbody Radiation Between Cooled Long-Wave Infrared and Cooled Mid-Wave Infrared Sequential Frames
CMWIR	Cooled Mid-Wave Infrared
FFT	Fast Fourier Transform
FOV	Field of View
IR	Infrared
IR3	Triple IR
LSB	Least Significant Bit
LWIR	Long-Wave Infrared
MLP	Multilayer Perceptron
MWIR	Mid-Wave Infrared
Non-Fire Radiation	The detected thermal radiation of non-fire-related stimuli
PCA	Principal Component Analysis
RANSAC	Random Sampling and Consensus
RGB	Red, Green, Blue
SWIR	Short-Wave Infrared
TIFF	Tagged Image File Format
UCLWIR	Uncooled Long-Wave Infrared

APPENDIX A: MATLAB SCRIPTS, FUNCTIONS, AND TOOLS

```
%% Matlab Script -----%
% Script : STEP_0a_CLMMWIRvid2frame.m, formally: CLMMWIRvid2frame.m
% Author : Tim Davenport
% Editor: Raul-Alexander Aldama
% Date : 10/1/11
%
% Description : Extracts individual frames of the dual band Cooled Long
% Wave InfraRed (CLWIR) & Cooled Mid Wave InfraRed (CMWIR)
% .raw video streams from the June 18 fire test at Raytheon
% Vision Systems (RVS) in Goleta, CA.
%
% Notes : 1. Video files were originally provided in the folder:
% Fire_Data_6-18-11_MW_LW
% It has since been renamed to:
% Fire_Data_6-18-11_UCLW_CMW_CLW
% 2. The test video files are separated into multiple "burns"
% that used various types of fuel. Both the source files
% and the resulting frames are labeled as such, and the
% frames for each burn are stored in separate folders.
% 3. Pixel data is 14-bit w/ LSBs zero-padded to 16
% so frame is rightshifted by dividing by 4.
% Eg: ##### ##### ##### ##00 >> ## ##### ##### #####
% 4. Dual band video was captured with MidWave of left and
% LongWave on right (MW has less 'blooming')
% Resolution is (2*640=1280)x480
% Frame Rate is 60 fps for both (30fps per band)
% Frame size is 1280x480x2 = 1228800 bytes
%
% Revision(s) : 01/27/12 - cleaned code: updated description, comments, and
% superfluous syntax to be more elegant/succinct.
%               01/25/13 - adapted code to windows file structure
%-----%
close all; clear; clc; tic; tic = tic;
%-----%
%% User Input: Source & Destination directory locations
% Source location: raw video files
vidfldr = 'K:\Thesis\Data\ORIGINAL_DATA\';
% Target directory: raw frames will be stored here
frmfldr = 'K:\Thesis\Data\ORIGINAL_DATA\';
%% Initialize & Allocate
MWcols = 640;
LWcols = 640;
nrows = 480;
frmnum = 0;
fps = 30;
% filename format: date_timestamp_fuel.raw
vidfiles = { '2011-06-18_101158_LW-MW-14bit_setup.raw'
'2011-06-18_102125_LW-MW-14bit_charcoal1.raw'
'2011-06-18_102454_LW-MW-14bit_oats1.raw'
'2011-06-18_102925_LW-MW-14bit_oats2.raw'
'2011-06-18_102949_LW-MW-14bit_oats2.raw'
'2011-06-18_103820_LW-MW-14bit_oats3.raw'
'2011-06-18_104857_LW-MW-14bit_pine_needles.raw'
'2011-06-18_105906_LW-MW-14bit_pine_cones.raw' }
```

```

'2011-06-18_110557_LW-MW-14bit_palm.raw'
'2011-06-18_110912_LW-MW-14bit_thistle.raw'
'2011-06-18_111644_LW-MW-14bit_wet_leaves.raw'
'2011-06-18_112306_LW-MW-14bit_wet_dry1.raw'
'2011-06-18_112757_LW-MW-14bit_wet_dry2.raw'
'2011-06-18_113916_LW-MW-14bit_rosemary.raw'
'2011-06-18_114605_LW-MW-14bit_applewood1.raw'
'2011-06-18_115209_LW-MW-14bit_applewood2.raw'
'2011-06-18_121233_LW-MW-14bit_extinguish.raw'
};
vidlabels = { 'setup1'
'charcoal1'
'oats1'
'oats2'
'oats2b'
'oats3'
'pine_needles'
'pine_cones'
'palm'
'thistle'
'wet_leaves'
'wet_dry1'
'wet_dry2'
'rosemary'
'applewood1'
'applewood2'
'extinguish'
};

read_attempt = 0;
%% Video Processing
for i=1:1:numel(vidfiles) %step through each stream raw file
vidtic = tic;
disp(['Opening Video : ' vidfiles{i}]);
% Update file names & locations
vidname = vidfiles{i};
vidlabel = vidlabels{i};
vidpath = [vidfldr vidname];
LWprefix = ['CLWIR_' vidlabel '_'];
MWprefix = ['CMWIR_' vidlabel '_'];
LWsavefldr = [frmfldr 'CLWIR/' vidlabel '/'];
MWsavefldr = [frmfldr 'CMWIR/' vidlabel '/'];
mkdir(LWsavefldr); % make folder for each lwir video
mkdir(MWsavefldr); % make folder for each mwir video
fid = fopen(vidpath,'r'); % open file for read access
display('Extracting frames...')
% Extract successive frames from each raw video stream until eof
while ~feof(fid)
% Read in raw video frame of 1280x480 = 614400 16-bit pixels
dualfrm = uint16(fread(fid,[(MWcols+LWcols), nrows],'uint16'));
if ~isempty(dualfrm)
frmnum = frmnum + 1;
% Skip every other frame (ie. move file position 1228800 bytes)
% this is because at 60 fps (each band at 30) the new frame is
% captured on left side while the right still has the previous
% frame, then when the new right side frame is capture the
% 2 are in sync. Thus there is an 'inbetween' sample (@60fps)

```

```

% where the 2 frames are missaligned time-wise. For Example:
%
% t |lw |mw
%-----
% t0 | 1 | 1 <=
% t(1/60) | 2 | 1 (skip)
% t(2/60) | 2 | 2 <=
% t(3/60) | 3 | 2 (skip)
% t(4/60) | 3 | 3 <=
fseek(fid,2*(MWcols+LWcols)*nrows,'cof');
% separate bands (lw on left, mw on right) &
% divide each pixel by 4 to right shift twice
LWframe = dualfrm(1:640,:)./4;
MWframe = dualfrm(641:1280,:)./4;
% save individual frame with the following file name
LWsavepath = [LWsavefldr vidlabel '_' int2str(frmnum) '.raw'];
MWsavepath = [MWsavefldr vidlabel '_' int2str(frmnum) '.raw'];
LWfsav = fopen(LWsavepath,'w'); % open for write access
MWfsav = fopen(MWsavepath,'w'); % open for write access
fwrite(LWfsav,LWframe(:,:), 'uint16'); % write frame 16-bit data
fwrite(MWfsav,MWframe(:,:), 'uint16'); % write frame 16-bit data
fclose(LWfsav); % close file
fclose(MWfsav); % close file
end
end
disp(['Frames Saved : ' num2str(frmnum)]);
disp(['Video length : ' num2str(frmnum/fps) 's']);
disp(['Extract time : ' num2str(toc(vidtic)) 's']);
fclose(fid);
frmnum = 0;
end
toc(ticetime)
%% End of Script -----

```



```

%% Matlab Script -----%
% Script : STEP_0b_raw2Tif_script.m
% Author : Raul-Alexander Aldama
% Date : 1/21/13
%
% Description : Converts extracted RAW frames to unsigned 16-bit TIFF
%
% Notes :
%
% Revision(s) :
%-----%
clear all; close all;

path = 'K:\Thesis\Data\ORIGINAL_DATA\';

nRow = 480;
nColumn = 640;

for i=1:16
    switch i
        case 1
            filename = 'applewood1';
            startFrame = 1;
            endFrame = 6100;
            frameLength = 1;
        case 2
            filename = 'applewood2';
            startFrame = 1;
            endFrame = 6100;
            frameLength = 1;
        case 3
            filename = 'charcoal1';
            startFrame = 1;
            endFrame = 4000;
            frameLength = 1;
        case 4
            filename = 'extinguish';
            startFrame = 1;
            endFrame = 6100;
            frameLength = 1;
        case 5
            filename = 'oats1';
            startFrame = 1;
            endFrame = 8000;
            frameLength = 1;
        case 6
            filename = 'oats2';
            startFrame = 1;
            endFrame = 395;
            frameLength = 1;
        case 7
            filename = 'oats2b';
            startFrame = 1;
            endFrame = 8000;
            frameLength = 1;
    end
end

```

```

case 8
    filename = 'oats3';
    startFrame = 1;
    endFrame = 8000;
    frameLength = 1;
case 9
    filename = 'palm';
    startFrame = 1;
    endFrame = 4405;
    frameLength = 1;
case 10
    filename = 'pine_cones';
    startFrame = 1;
    endFrame = 8000;
    frameLength = 1;
case 11
    filename = 'pine_needles';
    startFrame = 1;
    endFrame = 14934;
    frameLength = 1;
case 12
    filename = 'rosemary';
    startFrame = 1;
    endFrame = 5609;
    frameLength = 1;
case 13
    filename = 'thistle';
    startFrame = 1;
    endFrame = 6670;
    frameLength = 1;
case 14
    filename = 'wet_dry1';
    startFrame = 1;
    endFrame = 6117;
    frameLength = 1;
case 15
    filename = 'wet_dry2';
    startFrame = 1;
    endFrame = 6364;
    frameLength = 1;
case 16
    filename = 'wet_leaves';
    startFrame = 1;
    endFrame = 6063;
    frameLength = 1;
end

path_CMWIR = strcat(path, 'CMWIR\', filename, '\\');
filename_CMWIR = strcat(path_CMWIR, filename, '_');

path_CLWIR = strcat(path, 'CLWIR\', filename, '\\');
filename_CLWIR = strcat(path_CLWIR, filename, '_');

iterations = (endFrame-startFrame+1)/frameLength;
for n=1:iterations
    currentStart = startFrame+(n-1)*frameLength;

```

```

currentEnd = n*frameLength + startFrame-1;

listCMWIR = generateFrameList(filename_CMWIR,currentStart,...
    currentEnd, '.raw');
listCLWIR = generateFrameList(filename_CLWIR,currentStart,...
    currentEnd, '.raw');

CMWIR = raw2Video(listCMWIR,nRow,nColumn, 'uint16');
CLWIR = raw2Video(listCLWIR,nRow,nColumn, 'uint16');

listCMWIR = generateFrameList(filename_CMWIR,currentStart,...
    currentEnd, '.tif');
listCLWIR = generateFrameList(filename_CLWIR,currentStart,...
    currentEnd, '.tif');

writeTifs(CMWIR,listCMWIR);
writeTifs(CLWIR,listCLWIR);
end
end
%% End of Script -----%
```

```

%% Matlab Script -----
% Script : STEP_1a_analyze_originalVideo_script.m
% Author : Raul-Alexander Aldama
% Date : 2/11/13
%
% Description : Analyzes the originally recorded frames for signs of
% saturation
%
% Notes :   Video Sequences   Frames
%           'setup1'           3753
%           'charcoal1'        4000
%           'oats1'            8000
%           'oats2'            395
%           'oats2b'           8000
%           'oats3'            8000
%           'pine_needles'     14934
%           'pine_cones'       8000
%           'palm'             4405
%           'thistle'          6670
%           'wet_leaves'       6063
%           'wet_dry1'         6117
%           'wet_dry2'         6364
%           'rosemary'         5609
%           'applewood1'       6100
%           'applewood2'       6100
%           'extinguish'       6100
%
% Revision(s) :
%-----
clear all; close all;

path = 'K:\Thesis\Data\';

filename = 'oats1';
startFrame = 1;
endFrame = 8000;
frameLength = 1;

colorMap = 'hot';
viewFrame = 1;

plot_enable = 1;
write_enable = 0;

targetUpperLeftCorner = [305 311]; % [X Y]
targetBottomRightCorner = [390 375]; % [X Y]
targetRows = targetUpperLeftCorner(2):targetBottomRightCorner(2);
targetCols = targetUpperLeftCorner(1):targetBottomRightCorner(1);

procFrames = frameLength*floor((endFrame-startFrame+1)/frameLength);
endFrame = startFrame + procFrames - 1;
disp(['Processing ', num2str(procFrames), ' ', filename, ' Frames...']);
iterations = (endFrame-startFrame+1)/frameLength;

dynRangeOverTimeCLWIR = zeros(2,procFrames,'uint16');

```

```

dynRangeOverTimeCMWIR = zeros(2,procFrames,'uint16');

CLWIRFireOverTime = zeros(length(targetRows),length(targetCols),...
    procFrames,'uint16');
CMWIRFireOverTime = zeros(length(targetRows),length(targetCols),...
    procFrames,'uint16');

meanRatioNoFireCMCL = zeros(1, procFrames);
minRatioNoFireCMCL = zeros(1, procFrames);
maxRatioNoFireCMCL = zeros(1, procFrames);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INPUT DATA PATHS
path_CLWIR = strcat(path,'ORIGINAL_DATA\CLWIR\',filename,'\');
filename_CLWIR = strcat(path_CLWIR,filename,'_');

path_CMWIR = strcat(path,'ORIGINAL_DATA\CMWIR\',filename,'\');
filename_CMWIR = strcat(path_CMWIR,filename,'_');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for n=1:iterations
    currentStart = startFrame+(n-1)*frameLength;
    currentEnd = n*frameLength + startFrame-1;

    list_CLWIR = generateFrameList(filename_CLWIR,currentStart,...
        currentEnd, '.tif');
    list_CMWIR = generateFrameList(filename_CMWIR,currentStart,...
        currentEnd, '.tif');

    CLWIR = tif2Video(list_CLWIR);
    CMWIR = tif2Video(list_CMWIR);

    dynRangeOverTimeCLWIR(:,n) = minmax(CLWIR(:)');
    dynRangeOverTimeCMWIR(:,n) = minmax(CMWIR(:)');

    CLWIRFireOverTime(:, :, currentStart:currentEnd) = CLWIR(targetRows,...
        targetCols,:);
    CMWIRFireOverTime(:, :, currentStart:currentEnd) = CMWIR(targetRows,...
        targetCols,:);

    noFireCLWIR = CLWIR;
    noFireCLWIR(targetRows,targetCols) = zeros(length(targetRows),...
        length(targetCols),'uint16');
    dim = size(noFireCLWIR);
    noFireCLWIR = reshape(noFireCLWIR, [1 dim(1)*dim(2)]);
    noFireIndex = noFireCLWIR > 0;

    noFireCMWIR = CMWIR;
    noFireCMWIR(targetRows,targetCols) = zeros(length(targetRows),...
        length(targetCols),'uint16');
    dim = size(noFireCMWIR);
    noFireCMWIR = reshape(noFireCMWIR, [1 dim(1)*dim(2)]);
    noFireIndex = noFireIndex & noFireCMWIR > 0;

```

```

noFireCLWIR = noFireCLWIR(noFireIndex);
noFireCMWIR = noFireCMWIR(noFireIndex);

meanRatioNoFireCMCL(n) = mean(double(noFireCMWIR)./...
    double(noFireCLWIR));
minRatioNoFireCMCL(n) = min(double(noFireCMWIR)./double(noFireCLWIR));
maxRatioNoFireCMCL(n) = max(double(noFireCMWIR)./double(noFireCLWIR));
end
if plot_enable
    f1 = figure(1);
    set(f1, 'units', 'normalized', 'outerposition', [0 0 1 1])
    subplot(1,2,1)
    hold on
    h1 = line(1:procFrames, dynRangeOverTimeCLWIR(1,:), 'Color', 'blue', ...
        'LineWidth', 4);
    h2 = line(1:procFrames, dynRangeOverTimeCLWIR(2,:), 'Color', 'red');
    hold off
    title([filename, ' CLWIR Min and Max Values Over Time'])
    ylabel('Intensity')
    xlabel('Frame')
    YLim = ylim;
    legend([h1,h2], {'Min', 'Max'})

    subplot(1,2,2)
    hold on
    h1 = line(1:procFrames, dynRangeOverTimeCMWIR(1,:), 'Color', 'blue', ...
        'LineWidth', 4);
    h2 = line(1:procFrames, dynRangeOverTimeCMWIR(2,:), 'Color', 'red');
    hold off
    title([filename, ' CMWIR Min and Max Values Over Time'])
    ylabel('Intensity')
    xlabel('Frame')
    set(gca, 'YLim', YLim);
    legend([h1,h2], {'Min', 'Max'})
end
minDynRangeOverTimeCLWIR = min(dynRangeOverTimeCLWIR(1,:));
maxDynRangeOverTimeCLWIR = max(dynRangeOverTimeCLWIR(2,:));

minDynRangeOverTimeCMWIR = min(dynRangeOverTimeCMWIR(1,:));
maxDynRangeOverTimeCMWIR = max(dynRangeOverTimeCMWIR(2,:));

minMaxDynRangeOverTimeCLWIR = [minDynRangeOverTimeCLWIR, ...
    maxDynRangeOverTimeCLWIR];
minMaxDynRangeOverTimeCMWIR = [minDynRangeOverTimeCMWIR, ...
    maxDynRangeOverTimeCMWIR];
rangeOverTimeCLWIR = maxDynRangeOverTimeCLWIR-minDynRangeOverTimeCLWIR+1;
rangeOverTimeCMWIR = maxDynRangeOverTimeCMWIR-minDynRangeOverTimeCMWIR+1;

disp(['CL Min, Max Values: ', num2str(minMaxDynRangeOverTimeCLWIR)])
disp(['CL Range: ', num2str(rangeOverTimeCLWIR)])
disp(['CM Min, Max Values: ', num2str(minMaxDynRangeOverTimeCMWIR)])
disp(['CM Range: ', num2str(rangeOverTimeCMWIR)])

meanMeanRatioNoFireCMCL = mean(meanRatioNoFireCMCL);
meanMinRatioNoFireCMCL = mean(minRatioNoFireCMCL);
meanMaxRatioNoFireCMCL = mean(maxRatioNoFireCMCL);

```

```

CMCLRatio = meanRatioNoFireCMCL;

if write_enable
    save(strcat(path_CLWIR, 'CMCLRatio'), 'CMCLRatio', '-v7.3');
end

if plot_enable
    imageCL = CLWIRFireOverTime(:, :, viewFrame);
    imageCLRange = max(imageCL(:)) - min(imageCL(:));
    imageCL = setHist(imageCL, minmax(imageCL(:)'), [0 65535]);

    imageCM = CMWIRFireOverTime(:, :, viewFrame);
    imageCMRange = max(imageCM(:)) - min(imageCM(:));
    imageCM = setHist(imageCM, minmax(imageCM(:)'), [0 imageCMRange]);
    imageCM = setHist(imageCM, [0 imageCLRange], [0 65535]);

    f2 = figure(2);
    set(f2, 'units', 'normalized', 'outerposition', [0 0 1 1])
    subplot(2,2,1)
    imshow(imageCL)
    title([filename, ' ', num2str(size(targetRows,2)), 'x', ...
        num2str(size(targetCols,2)), ...
        ' CLWIR Pixels Around Fire Region'])
    colormap(colorMap);

    subplot(2,2,2)
    imshow(imageCM)
    title([filename, ' ', num2str(size(targetRows,2)), 'x', ...
        num2str(size(targetCols,2)), ...
        ' CMWIR Pixels Around Fire Region'])
    colormap(colorMap);
end

dim = size(CLWIRFireOverTime);
CLWIRFireOverTime = reshape(CLWIRFireOverTime, [dim(1)*dim(2) dim(3)]);

dim = size(CMWIRFireOverTime);
CMWIRFireOverTime = reshape(CMWIRFireOverTime, [dim(1)*dim(2) dim(3)]);

minMaxDynRangeFireOverTimeCLWIR = minmax(CLWIRFireOverTime(:)');
rangeFireOverTimeCLWIR = minMaxDynRangeFireOverTimeCLWIR(2)-...
    minMaxDynRangeFireOverTimeCLWIR(1)+1;
disp(['CL Fire Min, Max Values: ', ...
    num2str(minMaxDynRangeFireOverTimeCLWIR)])
disp(['CL Fire Range: ', num2str(rangeFireOverTimeCLWIR)])

minMaxDynRangeFireOverTimeCMWIR = minmax(CMWIRFireOverTime(:)');
rangeFireOverTimeCMWIR = minMaxDynRangeFireOverTimeCMWIR(2)-...
    minMaxDynRangeFireOverTimeCMWIR(1)+1;
disp(['CM Fire Min, Max Values: ', ...
    num2str(minMaxDynRangeFireOverTimeCMWIR)])
disp(['CM Fire Range: ', num2str(rangeFireOverTimeCMWIR)])

disp(['Mean of the Average CM/CL Ratio of Non-Fire Areas: ', ...
    num2str(meanMeanRatioNoFireCMCL)])

```

```

disp(['Mean of the Min CM/CL Ratio of Non-Fire Areas: ',...
    num2str(meanMinRatioNoFireCMCL)])
disp(['Mean of the Max CM/CL Ratio of Non-Fire Areas: ',...
    num2str(meanMaxRatioNoFireCMCL)])

if plot_enable
    subplot(2,2,3)
    line(1:procFrames,CLWIRFireOverTime);
    title([filename, ' CLWIR Pixel Intensity Around Fire Region Over Time'])
    ylabel('Intensity')
    xlabel('Frame')
    YLim = ylim;

    subplot(2,2,4)
    line(1:procFrames,CMWIRFireOverTime);
    title([filename, ' CMWIR Pixel Intensity Around Fire Region Over Time'])
    ylabel('Intensity')
    xlabel('Frame')
    set(gca,'YLim',YLim);

    f3 = figure(3);
    set(f3,'units','normalized','outerposition',[0 0 1 1])
    subplot(1,3,1)
    hold on
    plot(meanRatioNoFireCMCL);
    plot(1:procFrames,meanMeanRatioNoFireCMCL,'Color','red','LineWidth',3)
    hold off
    title([filename, ' Average CM/CL Ratio of Non-Fire Areas Over Time'])
    xlabel('Frame')
    ylabel('Intensity Ratio (CM/CL)')

    subplot(1,3,2)
    hold on
    plot(minRatioNoFireCMCL);
    plot(1:procFrames,meanMinRatioNoFireCMCL,'Color','red','LineWidth',3)
    hold off
    title([filename, ' Min CM/CL Ratio of Non-Fire Areas Over Time'])
    xlabel('Frame')
    ylabel('Intensity Ratio (CM/CL)')

    subplot(1,3,3)
    hold on
    plot(maxRatioNoFireCMCL);
    plot(1:procFrames,meanMaxRatioNoFireCMCL,'Color','red','LineWidth',3)
    hold off
    title([filename, ' Max CM/CL Ratio of Non-Fire Areas Over Time'])
    xlabel('Frame')
    ylabel('Intensity Ratio (CM/CL)')
end
%% End of Script -----%
```



```

%% Matlab Script -----
% Script : STEP_1b_generate_validVideoFrames_script.m
% Author : Raul-Alexander Aldama
% Date : 2/11/13
%
% Description : Creates a compilation of unsaturated frames (valid Frames)
% from a sequence of original frames
%
% Notes :   Video Sequences   Frames   Saturation Threshold (CLWIR, CMWIR)
%           'setup1'           3753
%           'charcoal1'        4000      65535, 65535
%           'oats1'            8000      1.38e4, 1.15e4
%           'oats2'            395
%           'oats2b'           8000
%           'oats3'            8000
%           'pine_needles'     14934
%           'pine_cones'       8000
%           'palm'             4405
%           'thistle'          6670
%           'wet_leaves'       6063
%           'wet_dry1'         6117
%           'wet_dry2'         6364
%           'rosemary'         5609
%           'applewood1'       6100
%           'applewood2'       6100
%           'extinguish'       6100
%
% Revision(s) :
%-----
clear all; close all;

path = 'K:\Thesis\Data\';

filename = 'oats1';
startFrame = 1;
endFrame = 8000;
frameLength = 1;

write_enable = 0;
plot_enable = 1;

maxCLWIRLevel = 1.38e4; %UPDATE VALUE FOR EACH VIDEO SEQUENCE
maxCMWIRLevel = 1.15e4; %UPDATE VALUE FOR EACH VIDEO SEQUENCE

targetUpperLeftCorner = [305 311]; % [X Y]
targetBottomRightCorner = [390 375]; % [X Y]
targetRows = targetUpperLeftCorner(2):targetBottomRightCorner(2);
targetCols = targetUpperLeftCorner(1):targetBottomRightCorner(1);

procFrames = frameLength*floor((endFrame-startFrame+1)/frameLength);
endFrame = startFrame + procFrames - 1;
disp(['Processing ', num2str(procFrames), ' ', filename, ' Frames...']);
iterations = (endFrame-startFrame+1)/frameLength;

keptFrames = zeros(1, procFrames);

```

```

originalIndex = zeros(1, procFrames);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INPUT DATA PATHS
path_CLWIR = strcat(path, 'ORIGINAL_DATA\CLWIR\', filename, '\');
filename_CLWIR = strcat(path_CLWIR, filename, '_');

path_CMWIR = strcat(path, 'ORIGINAL_DATA\CMWIR\', filename, '\');
filename_CMWIR = strcat(path_CMWIR, filename, '_');

% OUTPUT DATA PATHS
path_CL = strcat(path, 'VALID_DATA\CLWIR\', filename, '\');
if write_enable
    [~,~,~] = mkdir(path_CL);
end
filename_CL = strcat(path_CL, filename, '_');

path_CM = strcat(path, 'VALID_DATA\CMWIR\', filename, '\');
if write_enable
    [~,~,~] = mkdir(path_CM);
end
filename_CM = strcat(path_CM, filename, '_');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
i=1;
for n=1:iterations
    currentStart = startFrame+(n-1)*frameLength;
    currentEnd = n*frameLength + startFrame-1;

    list_CLWIR = generateFrameList(filename_CLWIR, currentStart, currentEnd,
    '.tif');
    list_CMWIR = generateFrameList(filename_CMWIR, currentStart, currentEnd,
    '.tif');

    CLWIR = tif2Video(list_CLWIR);
    CMWIR = tif2Video(list_CMWIR);

    CLWIRFireOverTime = CLWIR(targetRows, targetCols, :);
    CMWIRFireOverTime = CMWIR(targetRows, targetCols, :);

    maxCLWIRFire = max(CLWIRFireOverTime(:));
    maxCMWIRFire = max(CMWIRFireOverTime(:));

    if (maxCLWIRFire < maxCLWIRLevel) && (maxCMWIRFire < maxCMWIRLevel)
        if write_enable
            list = generateFrameList(filename_CL, i, i, '.tif');
            writeTifs(CLWIR, list);

            list = generateFrameList(filename_CM, i, i, '.tif');
            writeTifs(CMWIR, list);
        end
        keptFrames(n) = 1;
        originalIndex(i) = n;
        i = i + 1;
    end
end
end

```

```

originalIndex = originalIndex(1:i-1);

if plot_enable
    figure()
    imagesc(keptFrames, [0 1])
    set(gca, 'YTick', [])
    xlabel('Frame')
    title([filename, ' Raw Valid Frames (Red), Invalid Frames (Blue)'])

    figure()
    scatter(originalIndex, 1:length(originalIndex), '.')
    title([filename, ' Valid Frame Number vs. Total Frame Number'])
    xlabel('Total Frame Number')
    ylabel('Valid Frame Number')
end
%% WRITE VIDEO FROM IMAGES ON HD
if write_enable
    save(strcat(path_CL, 'keptFrames'), 'keptFrames');
    save(strcat(path_CL, 'originalIndex'), 'originalIndex');

    list = generateFrameList(filename_CL, 1, i-1, '.tif');
    writeMPEG4(filename_CL, list, 1);

    list = generateFrameList(filename_CM, 1, i-1, '.tif');
    writeMPEG4(filename_CM, list, 1);
end
%%
%% End of Script -----%
```

```

%% Matlab Script -----
% Script : STEP_1c_analyze_validVideoFrames_script.m
% Author : Raul-Alexander Aldama
% Date : 2/11/13
%
% Description : Analyzes valid frames and generates statistics
%
% Notes :
%         Video Sequences   Frames   Valid Frames
%         'setup1'          3753
%         'charcoal1'        4000     4000
%         'oats1'            8000     2124
%         'oats2'            395
%         'oats2b'           8000
%         'oats3'            8000
%         'pine_needles'     14934
%         'pine_cones'       8000
%         'palm'             4405
%         'thistle'          6670
%         'wet_leaves'       6063
%         'wet_dry1'         6117
%         'wet_dry2'         6364
%         'rosemary'         5609
%         'applewood1'       6100
%         'applewood2'       6100
%         'extinguish'       6100
%
% Revision(s) :
%-----
clear all; close all;

path = 'K:\Thesis\Data\';

filename = 'oats1';
startFrame = 1;
endFrame = 2124;
frameLength = 1;

colorMap = 'hot';
viewFrame = 1;

write_enable = 0;
plot_enable = 1;

targetUpperLeftCorner = [305 311]; % [X Y]
targetBottomRightCorner = [390 375]; % [X Y]
targetRows = targetUpperLeftCorner(2):targetBottomRightCorner(2);
targetCols = targetUpperLeftCorner(1):targetBottomRightCorner(1);

procFrames = frameLength*floor((endFrame-startFrame+1)/frameLength);
endFrame = startFrame + procFrames - 1;
disp(['Processing ', num2str(procFrames), ' ', filename, ' Frames...']);
iterations = (endFrame-startFrame+1)/frameLength;

dynRangeOverTimeCLWIR = zeros(2,procFrames,'uint16');
dynRangeOverTimeCMWIR = zeros(2,procFrames,'uint16');

```

```

meanRatioNoFireCMCL = zeros(1, procFrames);
minRatioNoFireCMCL = zeros(1, procFrames);
maxRatioNoFireCMCL = zeros(1, procFrames);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INPUT DATA PATHS
path_CLWIR = strcat(path, 'VALID_DATA\CLWIR\', filename, '\');
filename_CLWIR = strcat(path_CLWIR, filename, '_');

path_CMWIR = strcat(path, 'VALID_DATA\CMWIR\', filename, '\');
filename_CMWIR = strcat(path_CMWIR, filename, '_');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
originalIndex = load(strcat(path_CLWIR, 'originalIndex'));

for n=1:iterations
    currentStart = startFrame+(n-1)*frameLength;
    currentEnd = n*frameLength + startFrame-1;

    list_CLWIR = generateFrameList(filename_CLWIR, currentStart, ...
        currentEnd, '.tif');
    list_CMWIR = generateFrameList(filename_CMWIR, currentStart, ...
        currentEnd, '.tif');

    CLWIR = tif2Video(list_CLWIR);
    CMWIR = tif2Video(list_CMWIR);

    dynRangeOverTimeCLWIR(:,n) = minmax(CLWIR(:)');
    dynRangeOverTimeCMWIR(:,n) = minmax(CMWIR(:)');

    CLWIRFireOverTime(:, :, currentStart:currentEnd) = CLWIR(targetRows, ...
        targetCols, :);
    CMWIRFireOverTime(:, :, currentStart:currentEnd) = CMWIR(targetRows, ...
        targetCols, :);

    noFireCLWIR = CLWIR;
    noFireCLWIR(targetRows, targetCols) = zeros(length(targetRows), ...
        length(targetCols), 'uint16');
    dim = size(noFireCLWIR);
    noFireCLWIR = reshape(noFireCLWIR, [1 dim(1)*dim(2)]);
    noFireIndex = noFireCLWIR > 0;

    noFireCMWIR = CMWIR;
    noFireCMWIR(targetRows, targetCols) = zeros(length(targetRows), ...
        length(targetCols), 'uint16');
    dim = size(noFireCMWIR);
    noFireCMWIR = reshape(noFireCMWIR, [1 dim(1)*dim(2)]);
    noFireIndex = noFireIndex & noFireCMWIR > 0;

    noFireCLWIR = noFireCLWIR(noFireIndex);
    noFireCMWIR = noFireCMWIR(noFireIndex);

    meanRatioNoFireCMCL(n) = mean(double(noFireCMWIR) ./ ...
        double(noFireCLWIR));

```

```

minRatioNoFireCMCL(n) = min(double(noFireCMWIR)./double(noFireCLWIR));
maxRatioNoFireCMCL(n) = max(double(noFireCMWIR)./double(noFireCLWIR));
end

if plot_enable
    f1 = figure(1);
    set(f1,'units','normalized','outerposition',[0 0 1 1])
    subplot(1,2,1)
    hold on
    h1 = line(1:procFrames,dynRangeOverTimeCLWIR(1,:), 'Color','blue',...
        'LineWidth',4);
    h2 = line(1:procFrames,dynRangeOverTimeCLWIR(2,:), 'Color','red');
    hold off
    title([filename, ' Valid CLWIR Min and Max Values Over Time'])
    ylabel('Intensity')
    xlabel('Frame')
    YLim = ylim;
    legend([h1,h2],{'Min','Max'})

    subplot(1,2,2)
    hold on
    h1 = line(1:procFrames,dynRangeOverTimeCMWIR(1,:), 'Color','blue',...
        'LineWidth',4);
    h2 = line(1:procFrames,dynRangeOverTimeCMWIR(2,:), 'Color','red');
    hold off
    title([filename, ' Valid CMWIR Min and Max Values Over Time'])
    ylabel('Intensity')
    xlabel('Frame')
    set(gca,'YLim',YLim);
    legend([h1,h2],{'Min','Max'})
end

minDynRangeOverTimeCLWIR = mean(dynRangeOverTimeCLWIR(1,:));
maxDynRangeOverTimeCLWIR = max(dynRangeOverTimeCLWIR(2,:));

minDynRangeOverTimeCMWIR = mean(dynRangeOverTimeCMWIR(1,:));
maxDynRangeOverTimeCMWIR = max(dynRangeOverTimeCMWIR(2,:));

minMaxDynRangeOverTimeCLWIR = [minDynRangeOverTimeCLWIR, ...
    maxDynRangeOverTimeCLWIR];
minMaxDynRangeOverTimeCMWIR = [minDynRangeOverTimeCMWIR, ...
    maxDynRangeOverTimeCMWIR];
rangeOverTimeCLWIR = maxDynRangeOverTimeCLWIR-minDynRangeOverTimeCLWIR+1;
rangeOverTimeCMWIR = maxDynRangeOverTimeCMWIR-minDynRangeOverTimeCMWIR+1;

disp(['Valid CL Min(Averaged), Max Values: ',...
    num2str(minMaxDynRangeOverTimeCLWIR)])
disp(['Valid CL Range: ', num2str(rangeOverTimeCLWIR)])
disp(['Valid CM Min(Averaged), Max Values: ',...
    num2str(minMaxDynRangeOverTimeCMWIR)])
disp(['Valid CM Range: ', num2str(rangeOverTimeCMWIR)])

meanMeanRatioNoFireCMCL = mean(meanRatioNoFireCMCL);
meanMinRatioNoFireCMCL = mean(minRatioNoFireCMCL);
meanMaxRatioNoFireCMCL = mean(maxRatioNoFireCMCL);

```

```

dynRange.dynRangeCLWIR = minMaxDynRangeOverTimeCLWIR;
dynRange.dynRangeCMWIR = minMaxDynRangeOverTimeCMWIR;
CMCLRatio = meanRatioNoFireCMCL;
if write_enable
    save(strcat(path_CLWIR, 'dynRange'), '-struct', 'dynRange', '-v7.3');
    save(strcat(path_CLWIR, 'CMCLRatio'), 'CMCLRatio', '-v7.3');
end

if plot_enable
    imageCL = CLWIRFireOverTime(:, :, viewFrame);
    imageCLRRange = max(imageCL(:)) - min(imageCL(:));
    imageCL = setHist(imageCL, minmax(imageCL(:)'), [0 65535]);

    imageCM = CMWIRFireOverTime(:, :, viewFrame);
    imageCMRange = max(imageCM(:)) - min(imageCM(:));
    imageCM = setHist(imageCM, minmax(imageCM(:)'), [0 imageCMRange]);
    imageCM = setHist(imageCM, [0 imageCLRRange], [0 65535]);

    f2 = figure(2);
    set(f2, 'units', 'normalized', 'outerposition', [0 0 1 1])
    subplot(2,2,1)
    imshow(imageCL)
    title([filename, ' ', num2str(size(targetRows,2)), 'x', ...
        num2str(size(targetCols,2)), ...
        ' Valid CLWIR Pixels Around Fire Region'])
    colormap(colorMap);

    subplot(2,2,2)
    imshow(imageCM)
    title([filename, ' ', num2str(size(targetRows,2)), 'x', ...
        num2str(size(targetCols,2)), ...
        ' Valid CMWIR Pixels Around Fire Region'])
    colormap(colorMap);
end
dim = size(CLWIRFireOverTime);
CLWIRFireOverTime = reshape(CLWIRFireOverTime, [dim(1)*dim(2) dim(3)]);

dim = size(CMWIRFireOverTime);
CMWIRFireOverTime = reshape(CMWIRFireOverTime, [dim(1)*dim(2) dim(3)]);

minMaxDynRangeFireOverTimeCLWIR = minmax(CLWIRFireOverTime(:));
rangeFireOverTimeCLWIR = minMaxDynRangeFireOverTimeCLWIR(2)-...
    minMaxDynRangeFireOverTimeCLWIR(1)+1;
disp(['Valid CL Fire Min, Max Values: ', ...
    num2str(minMaxDynRangeFireOverTimeCLWIR)])
disp(['Valid CL Fire Range: ', num2str(rangeFireOverTimeCLWIR)])

minMaxDynRangeFireOverTimeCMWIR = minmax(CMWIRFireOverTime(:));
rangeFireOverTimeCMWIR = minMaxDynRangeFireOverTimeCMWIR(2)-...
    minMaxDynRangeFireOverTimeCMWIR(1)+1;
disp(['Valid CM Fire Min, Max Values: ', ...
    num2str(minMaxDynRangeFireOverTimeCMWIR)])
disp(['Valid CM Fire Range: ', num2str(rangeFireOverTimeCMWIR)])

```

```

disp(['Mean of the Valid Average CM/CL Ratio of Non-Fire Areas: ',...
    num2str(meanMeanRatioNoFireCMCL)])
disp(['Mean of the Valid Min CM/CL Ratio of Non-Fire Areas: ',...
    num2str(meanMinRatioNoFireCMCL)])
disp(['Mean of the Valid Max CM/CL Ratio of Non-Fire Areas: ',...
    num2str(meanMaxRatioNoFireCMCL)])

if plot_enable
    subplot(2,2,3)
    line(1:procFrames,CLWIRFireOverTime);
    title([filename,...
        ' Valid CLWIR Pixel Intensity Around Fire Region Over Time'])
    ylabel('Intensity')
    xlabel('Frame')
    YLim = ylim;

    subplot(2,2,4)
    line(1:procFrames,CMWIRFireOverTime);
    title([filename,...
        ' Valid CMWIR Pixel Intensity Around Fire Region Over Time'])
    ylabel('Intensity')
    xlabel('Frame')
    set(gca,'YLim',YLim);

    f3 = figure(3);
    set(f3,'units','normalized','outerposition',[0 0 1 1])
    subplot(1,3,1)
    hold on
    plot(meanRatioNoFireCMCL);
    plot(1:procFrames,meanMeanRatioNoFireCMCL,'Color','red','LineWidth',3)
    hold off
    title([filename,...
        ' Valid Average CM/CL Ratio of Non-Fire Areas Over Time'])
    xlabel('Frame')
    ylabel('Intensity Ratio (CM/CL)')

    subplot(1,3,2)
    hold on
    plot(minRatioNoFireCMCL);
    plot(1:procFrames,meanMinRatioNoFireCMCL,'Color','red','LineWidth',3)
    hold off
    title([filename, ' Valid Min CM/CL Ratio of Non-Fire Areas Over Time'])
    xlabel('Frame')
    ylabel('Intensity Ratio (CM/CL)')

    subplot(1,3,3)
    hold on
    plot(maxRatioNoFireCMCL);
    plot(1:procFrames,meanMaxRatioNoFireCMCL,'Color','red','LineWidth',3)
    hold off
    title([filename, ' Valid Max CM/CL Ratio of Non-Fire Areas Over Time'])
    xlabel('Frame')
    ylabel('Intensity Ratio (CM/CL)')
end
%% End of Script -----%
```



```

%% Matlab Script -----%
% Script : STEP_1d_generate_validIntensityNormFigures_script.m
% Author : Raul-Alexander Aldama
% Date : 2/25/13
%
% Description : Illustrates the affect of CLWIR and CMWIR normalization for
% a specific frame pair
%
% Notes :
%
% Revision(s) :
%-----%
close all; clear all;

oatsCL = imread('K:\Thesis\Data\VALID_DATA\CLWIR\oats1\oats1_1201.tif');
oatsCM = imread('K:\Thesis\Data\VALID_DATA\CMWIR\oats1\oats1_1201.tif');
dim = size(oatsCL);
CL_CMCL = reshape(oatsCL, [1 dim(1)*dim(2)]);
index = CL_CMCL > 0;
dim = size(oatsCM);
CM_CMCL = reshape(oatsCM, [1 dim(1)*dim(2)]);
index = index & CM_CMCL > 0;
CL_CMCL = CL_CMCL(index);
CM_CMCL = CM_CMCL(index);
CMCL = mean(double(CM_CMCL)./double(CL_CMCL));
disp(num2str(CMCL));
disp(num2str(minmax(oatsCL(:))))
disp(num2str(minmax(oatsCM(:))))
[count, bins] = imhist(oatsCL,2^16);
numtones = numel(find(count));
disp(num2str(numtones))
[count, bins] = imhist(oatsCM,2^16);
numtones = numel(find(count));
disp(num2str(numtones))

f1 = figure(1);
set(f1,'units','normalized','outerposition',[0 0 1 1])
subtightplot(2,3,1)
imshow(oatsCL)
colormap(hot)
title('oats1 CLWIR Frame 1201 Before Normalization')
subplot(2,3,4)
[oatsCL_count, bins] = imhist(oatsCL,2^16);
stem(bins, oatsCL_count,'Marker','None')
set(gca,'XLim',[0 65535])
title('oats1 CLWIR Frame 1201 Histogram Before Normalization')
xlabel('Intensity')
ylabel('Count')

f2 = figure(2);
set(f2,'units','normalized','outerposition',[0 0 1 1])
subtightplot(2,3,1)
imshow(oatsCM)
colormap(hot)
title('oats1 CMWIR Frame 1201 Before Normalization')
subplot(2,3,4)

```

```

[oatsCM_count, bins] = imhist(oatsCM,2^16);
stem(bins, oatsCM_count,'Marker','None')
set(gca,'XLim', [0 65535])
title('oats1 CMWIR Frame 1201 Histogram Before Normalization')
xlabel('Intensity')
ylabel('Count')

oatsCLInv = oatsCL;
oatsCMInv = oatsCM./CMCL;
dim = size(oatsCLInv);
CL_CMCL = reshape(oatsCLInv, [1 dim(1)*dim(2)]);
index = CL_CMCL > 0;
dim = size(oatsCMInv);
CM_CMCL = reshape(oatsCMInv, [1 dim(1)*dim(2)]);
index = index & CM_CMCL > 0;
CL_CMCL = CL_CMCL(index);
CM_CMCL = CM_CMCL(index);
CMCL = mean(double(CM_CMCL)./double(CL_CMCL));
disp(num2str(CMCL));
disp(num2str(minmax(oatsCLInv(:))))
disp(num2str(minmax(oatsCMInv(:))))
[count, bins] = imhist(oatsCLInv,2^16);
numtones = numel(find(count));
disp(num2str(numtones))
[count, bins] = imhist(oatsCMInv,2^16);
numtones = numel(find(count));
disp(num2str(numtones))

figure(1)
subtightplot(2,3,2)
imshow(oatsCLInv)
colormap(hot)
title('oats1 CLWIR Frame 1201 After CM/CL Correction')
subplot(2,3,5)
[oatsCLShift_count, bins] = imhist(oatsCLInv,2^16);
stem(bins, oatsCLShift_count,'Marker','None')
set(gca,'XLim', [0 65535])
title('oats1 CLWIR Frame 1201 Histogram After CM/CL Correction')
xlabel('Intensity')
ylabel('Count')

figure(2)
subtightplot(2,3,2)
imshow(oatsCMInv)
colormap(hot)
title('oats1 CMWIR Frame 1201 After CM/CL Correction')
subplot(2,3,5)
[oatsCMShift_count, bins] = imhist(oatsCMInv,2^16);
stem(bins, oatsCMShift_count,'Marker','None')
set(gca,'XLim', [0 65535])
title('oats1 CMWIR Frame 1201 Histogram After CM/CL Correction')
xlabel('Intensity')
ylabel('Count')

oatsCLAdj = setHist(oatsCLInv, [3886 13788], [0 65535]);
oatsCMAdj = setHist(oatsCMInv, [3886 13788], [0 65535]);

```

```

dim = size(oatsCLAdj);
CL_CMCL = reshape(oatsCLAdj, [1 dim(1)*dim(2)]);
index = CL_CMCL > 0;
dim = size(oatsCMAdj);
CM_CMCL = reshape(oatsCMAdj, [1 dim(1)*dim(2)]);
index = index & CM_CMCL > 0;
CL_CMCL = CL_CMCL(index);
CM_CMCL = CM_CMCL(index);
CMCL = mean(double(CM_CMCL)./double(CL_CMCL));
disp(num2str(CMCL));
disp(num2str(minmax(oatsCLAdj(:))))
disp(num2str(minmax(oatsCMAdj(:))))
[count, bins] = imhist(oatsCLAdj,2^16);
numtones = numel(find(count));
disp(num2str(numtones))
[count, bins] = imhist(oatsCMAdj,2^16);
numtones = numel(find(count));
disp(num2str(numtones))

figure(1)
subtightplot(2,3,3)
imshow(oatsCLAdj)
colormap(hot)
title('oats1 CLWIR Frame 1201 After Dynamic Range Expansion')
subplot(2,3,6)
[oatsCLAdj_count, bins] = imhist(oatsCLAdj,2^16);
stem(bins, oatsCLAdj_count, 'Marker', 'None')
set(gca, 'XLim', [0 65535])
title('oats1 CLWIR Frame 1201 Histogram After Dynamic Range Expansion')
xlabel('Intensity')
ylabel('Count')

figure(2)
subtightplot(2,3,3)
imshow(oatsCMAdj)
colormap(hot)
title('oats1 CMWIR Frame 1201 After Dynamic Range Expansion')
subplot(2,3,6)
[oatsCMAdj_count, bins] = imhist(oatsCMAdj,2^16);
stem(bins, oatsCMAdj_count, 'Marker', 'None')
set(gca, 'XLim', [0 65535])
title('oats1 CMWIR Frame 1201 Histogram After Dynamic Range Expansion')
xlabel('Intensity')
ylabel('Count')

f3 = figure(3);
set(f3, 'units', 'normalized', 'outerposition', [0 0 1 1])
subplot(2,3,1:2)
hold on
s1 = stem(bins, oatsCL_count, 'Marker', 'None', 'Color', 'red');
s2 = stem(bins, oatsCLAdj_count, 'Marker', 'None', 'Color', 'blue');
hold off
set(gca, 'XLim', [0 65535])
legend([s1 s2], {'Before Normalization', 'After Normalization'})
title('oats1 CLWIR Frame 1201 Histograms')
xlabel('Intensity')

```

```

ylabel('Count')

subplot(2,3,4:5)
hold on
s1 = stem(bins, oatsCM_count, 'Marker', 'None', 'Color', 'red');
s2 = stem(bins, oatsCMAAdj_count, 'Marker', 'None', 'Color', 'blue');
hold off
set(gca, 'XLim', [0 65535])
legend([s1 s2], {'Before Normalization', 'After Normalization'})
title('oats1 CMWIR Frame 1201 Histograms')
xlabel('Intensity')
ylabel('Count')

dim = size(oatsCL);
oatsCLmap = reshape(oatsCL, [1 dim(1)*dim(2)]);
dim = size(oatsCLAdj);
oatsCLAdjmap = reshape(oatsCLAdj, [1 dim(1)*dim(2)]);
dim = size(oatsCM);
oatsCMmap = reshape(oatsCM, [1 dim(1)*dim(2)]);
dim = size(oatsCMAAdj);
oatsCMAAdjmap = reshape(oatsCMAAdj, [1 dim(1)*dim(2)]);

subplot(2,3,3)
scatter(oatsCLmap, oatsCLAdjmap, '.')
title('oats1 CLWIR Frame 1201 Intensity Mapping')
xlabel('Before Normalization')
ylabel('After Normalization')
subplot(2,3,6)
scatter(oatsCMmap, oatsCMAAdjmap, '.')
title('oats1 CMWIR Frame 1201 Intensity Mapping')
xlabel('Before Normalization')
ylabel('After Normalization')

f4 = figure(4);
subplot(2,1,1)
hold on
s1 = stem(bins, oatsCL_count, 'Marker', 'None', 'Color', 'blue');
s2 = stem(bins, oatsCM_count, 'Marker', 'None', 'Color', 'green');
hold off
set(gca, 'XLim', [3885 7224])
legend([s1 s2], {'CLWIR', 'CMWIR'})
title('oats1 Frame 1201 Histograms Before Normalization')
xlabel('Intensity')
ylabel('Count')

subplot(2,1,2)
hold on
s1 = stem(bins, oatsCLAdj_count, 'Marker', 'None', 'Color', 'blue');
s2 = stem(bins, oatsCMAAdj_count, 'Marker', 'None', 'Color', 'green');
hold off
set(gca, 'XLim', [0 23416])
legend([s1 s2], {'CLWIR', 'CMWIR'})
title('oats1 Frame 1201 Histograms After Normalizaiton')
xlabel('Intensity')
ylabel('Count')

```

```

f5 = figure(5);
set(f5, 'units', 'normalized', 'outerposition', [0 0 1 1])
subtightplot(2,5,1)
imshow(oatsCL, [3885 7424])
title('Full Region')
subtightplot(2,5,2)
oatsCL1 = applyThreshVideo(oatsCL, 3885, 4146);
[oatsCL1_count, bins] = imhist(oatsCL1, 2^16);
imshow(oatsCL1, [3885 7424])
title('First Sub-Region')
subtightplot(2,5,3)
oatsCL2 = applyThreshVideo(oatsCL, 4147, 4402);
[oatsCL2_count, bins] = imhist(oatsCL2, 2^16);
imshow(oatsCL2, [3885 7424])
title('Second Sub-Region')
subtightplot(2,5,4)
oatsCL3 = applyThreshVideo(oatsCL, 4403, 4727);
[oatsCL3_count, bins] = imhist(oatsCL3, 2^16);
imshow(oatsCL3, [3885 7424])
title('Third Sub-Region')
subtightplot(2,5,5)
oatsCL4 = applyThreshVideo(oatsCL, 4728, 65535);
[oatsCL4_count, bins] = imhist(oatsCL4, 2^16);
imshow(oatsCL4, [3885 7424])
title('Fourth Sub-Region')
colormap(hot)
subplot(2,5,6)
stem(bins, oatsCL_count, 'Marker', 'none')
xlabel('Intensity')
ylabel('Count')
yLim = ylim;
set(gca, 'Xlim', [3885 7424])
set(gca, 'Ylim', yLim)
title('Full Region Histogram')
subplot(2,5,7)
stem(bins, oatsCL1_count, 'Marker', 'none')
xlabel('Intensity')
ylabel('Count')
set(gca, 'Xlim', [3885 7424])
set(gca, 'Ylim', yLim)
title('First Sub-Region Histogram')
subplot(2,5,8)
stem(bins, oatsCL2_count, 'Marker', 'none')
xlabel('Intensity')
ylabel('Count')
set(gca, 'Xlim', [3885 7424])
set(gca, 'Ylim', yLim)
title('Second Sub-Region Histogram')
subplot(2,5,9)
stem(bins, oatsCL3_count, 'Marker', 'none')
xlabel('Intensity')
ylabel('Count')
set(gca, 'Xlim', [3885 7424])
set(gca, 'Ylim', yLim)
title('Third Sub-Region Histogram')
subplot(2,5,10)
stem(bins, oatsCL4_count, 'Marker', 'none')

```

```

xlabel('Intensity')
ylabel('Count')
set(gca,'Xlim',[3885 7424])
set(gca,'Ylim',yLim)
title('Fourth Sub-Region Histogram')
suprtitle(...)
    'oats1 CLWIR Frame 1201 Histogram Dissection Before Normalization')

f6 = figure(6);
set(f6,'units','normalized','outerposition',[0 0 1 1])
subtightplot(2,5,1)
imshow(oatsCM,[3885 7424])
title('Full Region')
subtightplot(2,5,2)
oatsCM1 = applyThreshVideo(oatsCM,5541,5839);
[oatsCM1_count, bins] = imhist(oatsCM1, 2^16);
imshow(oatsCM1,[3885 7424])
title('First Sub-Region')
subtightplot(2,5,3)
oatsCM2 = applyThreshVideo(oatsCM,5840,6287);
[oatsCM2_count, bins] = imhist(oatsCM2, 2^16);
imshow(oatsCM2,[3885 7424])
title('Second Sub-Region')
subtightplot(2,5,4)
oatsCM3 = applyThreshVideo(oatsCM,6288,6571);
[oatsCM3_count, bins] = imhist(oatsCM3, 2^16);
imshow(oatsCM3,[3885 7424])
title('Third Sub-Region')
subtightplot(2,5,5)
oatsCM4 = applyThreshVideo(oatsCM,6572,65535);
[oatsCM4_count, bins] = imhist(oatsCM4, 2^16);
imshow(oatsCM4,[3885 7424])
title('Fourth Sub-Region')
colormap(hot)
subplot(2,5,6)
stem(bins,oatsCM_count,'Marker','none')
xlabel('Intensity')
ylabel('Count')
yLim = ylim;
set(gca,'Xlim',[3885 7424])
set(gca,'Ylim',yLim)
title('Full Region Histogram')
subplot(2,5,7)
stem(bins,oatsCM1_count,'Marker','none')
xlabel('Intensity')
ylabel('Count')
set(gca,'Xlim',[3885 7424])
set(gca,'Ylim',yLim)
title('First Sub-Region Histogram')
subplot(2,5,8)
stem(bins,oatsCM2_count,'Marker','none')
xlabel('Intensity')
ylabel('Count')
set(gca,'Xlim',[3885 7424])
set(gca,'Ylim',yLim)
title('Second Sub-Region Histogram')
subplot(2,5,9)

```

```

stem(bins,oatsCM3_count,'Marker','none')
xlabel('Intensity')
ylabel('Count')
set(gca,'Xlim',[3885 7424])
set(gca,'Ylim',yLim)
title('Third Sub-Region Histogram')
subplot(2,5,10)
stem(bins,oatsCM4_count,'Marker','none')
xlabel('Intensity')
ylabel('Count')
set(gca,'Xlim',[3885 7424])
set(gca,'Ylim',yLim)
title('Fourth Sub-Region Histogram')
suptitle(...
    'oats1 CMWIR Frame 1201 Histogram Dissection Before Normalization')

f7 = figure(7);
set(f7,'units','normalized','outerposition',[0 0 1 1])
subtightplot(2,5,1)
imshow(oatsCLAdj,[0 23416])
title('Full Region')
subtightplot(2,5,2)
oatsCLAdj1 = applyThreshVideo(oatsCLAdj,0,1721);
[oatsCLAdj1_count, bins] = imhist(oatsCLAdj1, 2^16);
imshow(oatsCLAdj1,[0 23416])
title('First Sub-Region')
subtightplot(2,5,3)
oatsCLAdj2 = applyThreshVideo(oatsCLAdj,1722,3415);
[oatsCLAdj2_count, bins] = imhist(oatsCLAdj2, 2^16);
imshow(oatsCLAdj2,[0 23416])
title('Second Sub-Region')
subtightplot(2,5,4)
oatsCLAdj3 = applyThreshVideo(oatsCLAdj,3416,5784);
[oatsCLAdj3_count, bins] = imhist(oatsCLAdj3, 2^16);
imshow(oatsCLAdj3,[0 23416])
title('Third Sub-Region')
subtightplot(2,5,5)
oatsCLAdj4 = applyThreshVideo(oatsCLAdj,5785,65535);
[oatsCLAdj4_count, bins] = imhist(oatsCLAdj4, 2^16);
imshow(oatsCLAdj4,[0 23416])
title('Fourth Sub-Region')
colormap(hot)
subplot(2,5,6)
stem(bins,oatsCLAdj_count,'Marker','none')
xlabel('Intensity')
ylabel('Count')
yLim = ylim;
set(gca,'Xlim',[0 23416])
set(gca,'Ylim',yLim)
title('Full Region Histogram')
subplot(2,5,7)
stem(bins,oatsCLAdj1_count,'Marker','none')
xlabel('Intensity')
ylabel('Count')
set(gca,'Xlim',[0 23416])
set(gca,'Ylim',yLim)
title('First Sub-Region Histogram')

```

```

subplot(2,5,8)
stem(bins,oatsCLAdj2_count,'Marker','none')
xlabel('Intensity')
ylabel('Count')
set(gca,'Xlim',[0 23416])
set(gca,'Ylim',yLim)
title('Second Sub-Region Histogram')
subplot(2,5,9)
stem(bins,oatsCLAdj3_count,'Marker','none')
xlabel('Intensity')
ylabel('Count')
set(gca,'Xlim',[0 23416])
set(gca,'Ylim',yLim)
title('Third Sub-Region Histogram')
subplot(2,5,10)
stem(bins,oatsCLAdj4_count,'Marker','none')
xlabel('Intensity')
ylabel('Count')
set(gca,'Xlim',[0 23416])
set(gca,'Ylim',yLim)
title('Fourth Sub-Region Histogram')
suptitle('oats1 CLWIR Frame 1201 Histogram Dissection After Normalization')

f8 = figure(8);
set(f8,'units','normalized','outerposition',[0 0 1 1])
subtightplot(2,5,1)
imshow(oatsCMAdj,[0 23416])
title('Full Region')
subtightplot(2,5,2)
oatsCMAdj1 = applyThreshVideo(oatsCMAdj,285,1760);
[oatsCMAdj1_count, bins] = imhist(oatsCMAdj1, 2^16);
imshow(oatsCMAdj1,[0 23416])
title('First Sub-Region')
subtightplot(2,5,3)
oatsCMAdj2 = applyThreshVideo(oatsCMAdj,1761,3819);
[oatsCMAdj2_count, bins] = imhist(oatsCMAdj2, 2^16);
imshow(oatsCMAdj2,[0 23416])
title('Second Sub-Region')
subtightplot(2,5,4)
oatsCMAdj3 = applyThreshVideo(oatsCMAdj,3820,5500);
[oatsCMAdj3_count, bins] = imhist(oatsCMAdj3, 2^16);
imshow(oatsCMAdj3,[0 23416])
title('Third Sub-Region')
subtightplot(2,5,5)
oatsCMAdj4 = applyThreshVideo(oatsCMAdj,5501,65535);
[oatsCMAdj4_count, bins] = imhist(oatsCMAdj4, 2^16);
imshow(oatsCMAdj4,[0 23416])
title('Fourth Sub-Region')
colormap(hot)
subplot(2,5,6)
stem(bins,oatsCMAdj_count,'Marker','none')
xlabel('Intensity')
ylabel('Count')
yLim = ylim;
set(gca,'Xlim',[0 23416])
set(gca,'Ylim',yLim)
title('Full Region Histogram')

```



```

subplot(2,5,7)
stem(bins,oatsCMAdj1_count,'Marker','none')
xlabel('Intensity')
ylabel('Count')
set(gca,'Xlim',[0 23416])
set(gca,'Ylim',yLim)
title('First Sub-Region Histogram')
subplot(2,5,8)
stem(bins,oatsCMAdj2_count,'Marker','none')
xlabel('Intensity')
ylabel('Count')
set(gca,'Xlim',[0 23416])
set(gca,'Ylim',yLim)
title('Second Sub-Region Histogram')
subplot(2,5,9)
stem(bins,oatsCMAdj3_count,'Marker','none')
xlabel('Intensity')
ylabel('Count')
set(gca,'Xlim',[0 23416])
set(gca,'Ylim',yLim)
title('Third Sub-Region Histogram')
subplot(2,5,10)
stem(bins,oatsCMAdj4_count,'Marker','none')
xlabel('Intensity')
ylabel('Count')
set(gca,'Xlim',[0 23416])
set(gca,'Ylim',yLim)
title('Fourth Sub-Region Histogram')
suptitle('oats1 CMWIR Frame 1201 Histogram Dissection After Normalization')

figure(9)
hold on
s1 = scatter(oatsCLmap, oatsCMmap, '.', 'red');
s2 = scatter(oatsCLAdjmap, oatsCMAdjmap, '.', 'blue');
hold off
title('oats1 Frame 1201 Intensity Scatter Plot')
xlabel('CLWIR')
ylabel('CMWIR')
legend([s1 s2], {'Before Normalization', 'After Normalization'})
%% End of Script -----%
```

```

%% Matlab Script -----%
% Script : STEP_2a_preProcessValidVideo_script.m
% Author : Raul-Alexander Aldama
% Date : 2/26/13
%
% Description : Pre-processes Valid Frames, processes frames in sets of
% 30-frame length or 1 second
%
% Notes :   Video Sequences   Frames   Valid Frames (multiple of 30)
%           'setup1'           3753
%           'charcoal1'        4000      3990
%           'oats1'            8000      2100
%           'oats2'            395
%           'oats2b'           8000
%           'oats3'            8000
%           'pine_needles'     14934
%           'pine_cones'       8000
%           'palm'             4405
%           'thistle'          6670
%           'wet_leaves'       6063
%           'wet_dry1'         6117
%           'wet_dry2'         6364
%           'rosemary'         5609
%           'applewood1'       6100
%           'applewood2'       6100
%           'extinguish'       6100
%
% Revision(s) :
%-----%
clear all; close all; tic;

path = 'K:\Thesis\Data\';

filename = 'oats1';
startFrame = 1201;
endFrame = 1230;
frameLength = 30;

write_enable = 0;
plot_enable = 0;
viewFrame = 1;

intensityLim = 65535;
register_video = 0;
rPerc = 100/100;

noisy_frames = 0;
noisePerc = 10/100;
noiseMu = 0;
noiseSigma = intensityLim/48;
minNoiseVal = -intensityLim;
maxNoiseVal = intensityLim;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% INPUT DATA PATHS
path_CLWIR = strcat(path, 'VALID_DATA\CLWIR\ ', filename, '\ ');

```

```

filename_CLWIR = strcat(path_CLWIR,filename,'_');

path_CMWIR = strcat(path,'VALID_DATA\CMWIR\ ',filename,'\ ');
filename_CMWIR = strcat(path_CMWIR,filename,'_');

% OUTPUT_DATA_PATHS
path_CL = strcat(path,'INPUT_DATA\VALID\CL\ ',filename,'\ ');
if ~register_video
    path_CL = strcat(path_CL,'UNREG\ ');
end
if write_enable
    [~,~,~] = mkdir(path_CL);
end
filename_CL = strcat(path_CL,filename,'_');

path_CM = strcat(path,'INPUT_DATA\VALID\CM\ ',filename,'\ ');
if ~register_video
    path_CM = strcat(path_CM,'UNREG\ ');
end
if write_enable
    [~,~,~] = mkdir(path_CM);
end
filename_CM = strcat(path_CM,filename,'_');

if noisy_frames
    path_noisyCL = strcat(path,'INPUT_DATA\VALID\noisyCL\ ',filename,'\ ');
    if ~register_video
        path_noisyCL = strcat(path_noisyCL,'UNREG\ ');
    end
    if write_enable && noisy_frames
        [~,~,~] = mkdir(path_noisyCL);
    end
    filename_noisyCL = strcat(path_noisyCL,filename,'_');

    path_noisyCM = strcat(path,'INPUT_DATA\VALID\noisyCM\ ',filename,'\ ');
    if ~register_video
        path_noisyCM = strcat(path_noisyCM,'UNREG\ ');
    end
    if write_enable && noisy_frames
        [~,~,~] = mkdir(path_noisyCM);
    end
    filename_noisyCM = strcat(path_noisyCM,filename,'_');
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load(strcat(path_CLWIR,'dynRange'));
load(strcat(path_CLWIR,'CMCLRatio'));

[s1,s2,s3,s4,s5,s6,s7] = RandStream.create('mrg32k3a','NumStreams',7,...
    'seed',0);

%%
procFrames = frameLength*floor((endFrame-startFrame+1)/frameLength);
diagOffset = zeros(1,procFrames);
endFrame = startFrame + procFrames - 1;
disp(['Processing ',num2str(procFrames),' Frames...']);

```

```

iterations = (endFrame-startFrame+1)/frameLength;
%%
for n=1:iterations
    currentStart = startFrame+(n-1)*frameLength;
    currentEnd = n*frameLength + startFrame-1;

    list_CLWIR = generateFrameList(filename_CLWIR,currentStart,...
        currentEnd, '.tif');
    list_CMWIR = generateFrameList(filename_CMWIR,currentStart,...
        currentEnd, '.tif');

    CLWIR = tif2Video(list_CLWIR);
    CMWIR = tif2Video(list_CMWIR);

    if n == 1 && plot_enable
        plot = 1;
    else
        plot = 0;
    end

    dim = size(CMWIR);

    CMCL = CMCLRatio(currentStart:currentEnd);
    CMCL = repmat(CMCL,[dim(1) 1 dim(2)]);
    CMCL = permute(CMCL,[1 3 2]);

    [CL, CM, refCM, baseCM, borderOffset, regPoints] = preProcessVideo(...
        CLWIR, dynRangeCLWIR,...
        CMWIR, dynRangeCMWIR, CMCL, 0, 0, 0, intensityLim,...
        register_video, rPerc, s1,...
        frameLength, plot, viewFrame, filename, currentStart, currentEnd);

    diagOffset(1,(n-1)*frameLength+1:n*frameLength) = borderOffset;

    if noisy_frames
        noisyCL = awgnVideo(CL,noisePerc,noiseMu,noiseSigma,...
            minNoiseVal,maxNoiseVal,s2,s3,s4);
        noisyCM = awgnVideo(CM,noisePerc,noiseMu,noiseSigma,...
            minNoiseVal,maxNoiseVal,s5,s6,s7);
    end

    if write_enable
        list = generateFrameList(filename_CL,currentStart,currentEnd,...
            '.tif');
        writeTifs(CL, list);

        list = generateFrameList(filename_CM,currentStart,currentEnd,...
            '.tif');
        writeTifs(CM, list);

        if noisy_frames
            list = generateFrameList(filename_noisyCL,currentStart,...
                currentEnd, '.tif');
            writeTifs(noisyCL, list);
        end
    end
end

```

```

        list = generateFrameList(filename_noisyCM,currentStart,...
            currentEnd, '.tif');
        writeTifs(noisyCM, list);
    end
end

if write_enable
    save(strcat(path_CL, 'diagOffset'), 'diagOffset');
    save(strcat(path_CM, 'diagOffset'), 'diagOffset');

    if noisy_frames
        save(strcat(path_noisyCL, 'diagOffset'), 'diagOffset');
        save(strcat(path_noisyCM, 'diagOffset'), 'diagOffset');
    end
end

%% WRITE VIDEO FROM IMAGES ON HD
if write_enable
    list = generateFrameList(filename_CL,startFrame,endFrame, '.tif');
    writeMPEG4(filename_CL, list, startFrame);

    list = generateFrameList(filename_CM,startFrame,endFrame, '.tif');
    writeMPEG4(filename_CM, list, startFrame);

    if noisy_frames
        list = generateFrameList(filename_noisyCL,startFrame,endFrame,...
            '.tif');
        writeMPEG4(filename_noisyCL, list, startFrame);

        list = generateFrameList(filename_noisyCM,startFrame,endFrame,...
            '.tif');
        writeMPEG4(filename_noisyCM, list, startFrame);
    end
end
%%
toc;
%% End of Script -----%
```

```

%% Matlab Script -----%
% Script : STEP_2b_analyze_preProcessValidVideo_script.m
% Author : Raul-Alexander Aldama
% Date : 2/26/13
%
% Description : Analyzes pre-processed valid frames and generates
% statistics
%
% Notes :   Video Sequences   Frames   Valid Frames (multiple of 30)
%           'setup1'           3753
%           'charcoal1'        4000      3990
%           'oats1'            8000      2100
%           'oats2'            395
%           'oats2b'           8000
%           'oats3'            8000
%           'pine_needles'     14934
%           'pine_cones'       8000
%           'palm'             4405
%           'thistle'          6670
%           'wet_leaves'       6063
%           'wet_dry1'         6117
%           'wet_dry2'         6364
%           'rosemary'         5609
%           'applewood1'       6100
%           'applewood2'       6100
%           'extinguish'       6100
%
% Revision(s) :
%-----%
clear all; close all;

path = 'K:\Thesis\Data\';

filename = 'oats1';
startFrame = 1;
endFrame = 2100;
frameLength = 1;

register_video = 1;

colorMap = 'hot';
colorLim = [0 20000];
viewFrame = 1;

plot_enable = 1;

targetUpperLeftCorner = [305 311]; % [X Y]
targetBottomRightCorner = [390 375]; % [X Y]
targetRows = targetUpperLeftCorner(2):targetBottomRightCorner(2);
targetCols = targetUpperLeftCorner(1):targetBottomRightCorner(1);

procFrames = frameLength*floor((endFrame-startFrame+1)/frameLength);
endFrame = startFrame + procFrames - 1;
disp(['Processing ', num2str(procFrames), ' Frames...']);
iterations = (endFrame-startFrame+1)/frameLength;

```

```

dynRangeOverTimeCLWIR = zeros(2,procFrames,'uint16');
dynRangeOverTimeCMWIR = zeros(2,procFrames,'uint16');

CLWIRFireOverTime = zeros(length(targetRows),length(targetCols),...
    procFrames,'uint16');
CMWIRFireOverTime = zeros(length(targetRows),length(targetCols),...
    procFrames,'uint16');

meanRatioNoFireCMCL = zeros(1, procFrames);
minRatioNoFireCMCL = zeros(1, procFrames);
maxRatioNoFireCMCL = zeros(1, procFrames);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INPUT DATA PATHS
path_CLWIR = strcat(path,'INPUT_DATA\VALID\CL\',filename,'\');
if ~register_video
    path_CLWIR = strcat(path_CLWIR,'UNREG\');
end
filename_CLWIR = strcat(path_CLWIR,filename,'_');

path_CMWIR = strcat(path,'INPUT_DATA\VALID\CM\',filename,'\');
if ~register_video
    path_CMWIR = strcat(path_CMWIR,'UNREG\');
end
filename_CMWIR = strcat(path_CMWIR,filename,'_');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for n=1:iterations
    currentStart = startFrame+(n-1)*frameLength;
    currentEnd = n*frameLength + startFrame-1;

    list_CLWIR = generateFrameList(filename_CLWIR,currentStart,...
        currentEnd, '.tif');
    list_CMWIR = generateFrameList(filename_CMWIR,currentStart,...
        currentEnd, '.tif');

    CLWIR = tif2Video(list_CLWIR);
    CMWIR = tif2Video(list_CMWIR);

    dynRangeOverTimeCLWIR(:,n) = minmax(CLWIR(:)')';
    dynRangeOverTimeCMWIR(:,n) = minmax(CMWIR(:)')';

    CLWIRFireOverTime(:, :, currentStart:currentEnd) = CLWIR(targetRows,...
        targetCols,:);
    CMWIRFireOverTime(:, :, currentStart:currentEnd) = CMWIR(targetRows,...
        targetCols,:);

    noFireCLWIR = CLWIR;
    noFireCLWIR(targetRows,targetCols) = zeros(length(targetRows),...
        length(targetCols),'uint16');
    dim = size(noFireCLWIR);
    noFireCLWIR = reshape(noFireCLWIR, [1 dim(1)*dim(2)]);
    noFireIndex = noFireCLWIR > 0;

```

```

noFireCMWIR = CMWIR;
noFireCMWIR(targetRows,targetCols) = zeros(length(targetRows),...
length(targetCols),'uint16');
dim = size(noFireCMWIR);
noFireCMWIR = reshape(noFireCMWIR, [1 dim(1)*dim(2)]);
noFireIndex = noFireIndex & noFireCMWIR > 0;

noFireCLWIR = noFireCLWIR(noFireIndex);
noFireCMWIR = noFireCMWIR(noFireIndex);

meanRatioNoFireCMCL(n) = mean(double(noFireCMWIR)./...
double(noFireCLWIR));
minRatioNoFireCMCL(n) = min(double(noFireCMWIR)./double(noFireCLWIR));
maxRatioNoFireCMCL(n) = max(double(noFireCMWIR)./double(noFireCLWIR));
end
if plot_enable
    f1 = figure(1);
    set(f1,'units','normalized','outerposition',[0 0 1 1])
    subplot(1,2,1)
    hold on
    h1 = line(1:procFrames,dynRangeOverTimeCLWIR(1,:), 'Color','blue',...
'LineWidth',4);
    h2 = line(1:procFrames,dynRangeOverTimeCLWIR(2,:), 'Color','red');
    hold off
    title([filename, ' Pre-Processed CLWIR Min and Max Values Over Time'])
    ylabel('Intensity')
    xlabel('Frame')
    YLim = ylim;
    legend([h1,h2],{'Min','Max'})

    subplot(1,2,2)
    hold on
    h1 = line(1:procFrames,dynRangeOverTimeCMWIR(1,:), 'Color','blue',...
'LineWidth',4);
    h2 = line(1:procFrames,dynRangeOverTimeCMWIR(2,:), 'Color','red');
    hold off
    title([filename, ' Pre-Processed CMWIR Min and Max Values Over Time'])
    ylabel('Intensity')
    xlabel('Frame')
    set(gca,'YLim',YLim);
    legend([h1,h2],{'Min','Max'})
end
minDynRangeOverTimeCLWIR = min(dynRangeOverTimeCLWIR(1,:));
maxDynRangeOverTimeCLWIR = max(dynRangeOverTimeCLWIR(2,:));

minDynRangeOverTimeCMWIR = min(dynRangeOverTimeCMWIR(1,:));
maxDynRangeOverTimeCMWIR = max(dynRangeOverTimeCMWIR(2,:));

minMaxDynRangeOverTimeCLWIR = [minDynRangeOverTimeCLWIR,...
maxDynRangeOverTimeCLWIR];
minMaxDynRangeOverTimeCMWIR = [minDynRangeOverTimeCMWIR,...
maxDynRangeOverTimeCMWIR];
rangeOverTimeCLWIR = maxDynRangeOverTimeCLWIR-minDynRangeOverTimeCLWIR+1;
rangeOverTimeCMWIR = maxDynRangeOverTimeCMWIR-minDynRangeOverTimeCMWIR+1;

```



```

disp(['Pre-Processed CL Min, Max Values: ',...
    num2str(minMaxDynRangeOverTimeCLWIR)])
disp(['Pre-Processed CL Range: ', num2str(rangeOverTimeCLWIR)])
disp(['Pre-Processed CM Min, Max Values: ',...
    num2str(minMaxDynRangeOverTimeCMWIR)])
disp(['Pre-Processed CM Range: ', num2str(rangeOverTimeCMWIR)])

meanMeanRatioNoFireCMCL = mean(meanRatioNoFireCMCL);
meanMinRatioNoFireCMCL = mean(minRatioNoFireCMCL);
meanMaxRatioNoFireCMCL = mean(maxRatioNoFireCMCL);
CMCLRatio = meanRatioNoFireCMCL;

if plot_enable
    imageCL = CLWIRFireOverTime(:,:,viewFrame);
    imageCM = CMWIRFireOverTime(:,:,viewFrame);

    f2 = figure(2);
    set(f2,'units','normalized','outerposition',[0 0 1 1])
    subplot(2,2,1)
    imshow(imageCL,colorLim)
    title([filename,...
        ' ',num2str(size(targetRows,2)), 'x',...
        num2str(size(targetCols,2)),...
        ' Pre-Processed CLWIR Pixels Around Fire Region ',...
        '(Contrast Enhanced)'])
    colormap(colorMap);

    subplot(2,2,2)
    imshow(imageCM,colorLim)
    title([filename,...
        ' ',num2str(size(targetRows,2)), 'x',...
        num2str(size(targetCols,2)),...
        ' Pre-Processed CMWIR Pixels Around Fire Region ',...
        '(Contrast Enhanced)'])
    colormap(colorMap);
end

dim = size(CLWIRFireOverTime);
CLWIRFireOverTime = reshape(CLWIRFireOverTime,[dim(1)*dim(2) dim(3)]);

dim = size(CMWIRFireOverTime);
CMWIRFireOverTime = reshape(CMWIRFireOverTime,[dim(1)*dim(2) dim(3)]);

minMaxDynRangeFireOverTimeCLWIR = minmax(CLWIRFireOverTime(:));
rangeFireOverTimeCLWIR = minMaxDynRangeFireOverTimeCLWIR(2)-...
    minMaxDynRangeFireOverTimeCLWIR(1)+1;
disp(['Pre-Processed CL Fire Min, Max Values: ',...
    num2str(minMaxDynRangeFireOverTimeCLWIR)])
disp(['Pre-Processed CL Fire Range: ',num2str(rangeFireOverTimeCLWIR)])

minMaxDynRangeFireOverTimeCMWIR = minmax(CMWIRFireOverTime(:));
rangeFireOverTimeCMWIR = minMaxDynRangeFireOverTimeCMWIR(2)-...
    minMaxDynRangeFireOverTimeCMWIR(1)+1;
disp(['Pre-Processed CM Fire Min, Max Values: ',...

```

```

        num2str(minMaxDynRangeFireOverTimeCMWIR)])
disp(['Pre-Processed CM Fire Range: ',num2str(rangeFireOverTimeCMWIR)])

disp(['Mean of Pre-processed Average CM/CL Ratio of Non-Fire Areas: ',...
    num2str(meanMeanRatioNoFireCMCL)])
disp(['Mean of Pre-processed Min CM/CL Ratio of Non-Fire Areas: ',...
    num2str(meanMinRatioNoFireCMCL)])
disp(['Mean of Pre-processed Max CM/CL Ratio of Non-Fire Areas: ',...
    num2str(meanMaxRatioNoFireCMCL)])

if plot_enable
    subplot(2,2,3)
    line(1:procFrames,CLWIRFireOverTime);
    title([filename,...
        ' Pre-Processed CLWIR Pixel Intensity ',...
        'Around Fire Region Over Time'])
    ylabel('Intensity')
    xlabel('Frame')
    YLim = ylim;

    subplot(2,2,4)
    line(1:procFrames,CMWIRFireOverTime);
    title([filename,...
        ' Pre-Processed CMWIR Pixel Intensity ',...
        'Around Fire Region Over Time'])
    ylabel('Intensity')
    xlabel('Frame')
    set(gca,'YLim',YLim);

    f3 = figure(3);
    set(f3,'units','normalized','outerposition',[0 0 1 1])
    subplot(1,3,1)
    hold on
    plot(meanRatioNoFireCMCL);
    plot(1:procFrames,meanMeanRatioNoFireCMCL,'Color','red','LineWidth',3)
    hold off
    title([filename,...
        ' Pre-Processed Average CM/CL Ratio of Non-Fire Areas Over Time'])
    xlabel('Frame')
    ylabel('Intensity Ratio (CM/CL)')

    subplot(1,3,2)
    hold on
    plot(minRatioNoFireCMCL);
    plot(1:procFrames,meanMinRatioNoFireCMCL,'Color','red','LineWidth',3)
    hold off
    title([filename,...
        ' Pre-Processed Min CM/CL Ratio of Non-Fire Areas Over Time'])
    xlabel('Frame')
    ylabel('Intensity Ratio (CM/CL)')

    subplot(1,3,3)
    hold on
    plot(maxRatioNoFireCMCL);
    plot(1:procFrames,meanMaxRatioNoFireCMCL,'Color','red','LineWidth',3)
    hold off

```

```
title([filename,...
      ' Pre-Processed Max CM/CL Ratio of Non-Fire Areas Over Time'])
xlabel('Frame')
ylabel('Intensity Ratio (CM/CL) ')
end
%% End of Script -----%
```

```

%% Matlab Script -----%
% Script : STEP_3a_generate_validProcess_script.m
% Author : Raul-Alexander Aldama
% Date : 3/1/13
%
% Description : Performes Sequential Multispectral Difference frame
% processing, processes frames in sets of 30-frame length or 1 second
%
% Notes :   Video Sequences   Frames   Valid Frames (multiple of 30)
%           'setup1'           3753
%           'charcoal1'        4000      3990
%           'oats1'            8000      2100
%           'oats2'            395
%           'oats2b'           8000
%           'oats3'            8000
%           'pine_needles'     14934
%           'pine_cones'       8000
%           'palm'             4405
%           'thistle'          6670
%           'wet_leaves'       6063
%           'wet_dry1'         6117
%           'wet_dry2'         6364
%           'rosemary'         5609
%           'applewood1'       6100
%           'applewood2'       6100
%           'extinguish'       6100
%
% Revision(s) :
%-----%
clear all; close all; tic;

path = 'K:\Thesis\Data\';

filename = 'oats1';
startFrame = 1;
endFrame = 2100;
frameLength = 30;

registered_video = 1;
noisy_frames = 0;

write_enable = 0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% INPUT DATA PATHS
path_CL = strcat(path, 'INPUT_DATA\VALID\CL\', filename, '\');
if ~registered_video
    path_CL = strcat(path_CL, 'UNREG\');
end
filename_CL = strcat(path_CL, filename, '_');

path_CM = strcat(path, 'INPUT_DATA\VALID\CM\', filename, '\');
if ~registered_video
    path_CM = strcat(path_CM, 'UNREG\');
end

```

```

end
filename_CM = strcat(path_CM,filename,'_');

if noisy_frames
    path_noisyCL = strcat(path,'INPUT_DATA\VALID\noisyCL\',filename,'\');
    if ~registered_video
        path_noisyCL = strcat(path_noisyCL,'UNREG\');
    end
    filename_noisyCL = strcat(path_noisyCL,filename,'_');

    path_noisyCM = strcat(path,'INPUT_DATA\VALID\noisyCM\',filename,'\');
    if ~registered_video
        path_noisyCM = strcat(path_noisyCM,'UNREG\');
    end
    filename_noisyCM = strcat(path_noisyCM,filename,'_');
end
% OUTPUT_DATA PATHS
path_difCLCM = strcat(path,'PROCESS_DATA\VALID\difCLCM\',filename,'\');
if ~registered_video
    path_difCLCM = strcat(path_difCLCM,'UNREG\');
end
if write_enable
    [~,~,~] = mkdir(path_difCLCM);
end
filename_difCLCM = strcat(path_difCLCM,filename,'_');

if noisy_frames
    path_noisydifCLCM = strcat(path,'PROCESS_DATA\VALID\noisydifCLCM\',...
        filename,'\');
    if ~registered_video
        path_noisydifCLCM = strcat(path_noisydifCLCM,'UNREG\');
    end
    if write_enable
        [~,~,~] = mkdir(path_noisydifCLCM);
    end
    filename_noisydifCLCM = strcat(path_noisydifCLCM,filename,'_');
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
procFrames = frameLength*floor((endFrame-startFrame+1)/frameLength);
endFrame = startFrame + procFrames - 1;

disp(['Processing ',num2str(procFrames),' Frames...']);

iterations = (endFrame-startFrame+1)/frameLength;

%%
for n=1:iterations
    currentStart = startFrame+(n-1)*frameLength;
    currentEnd = n*frameLength + startFrame-1;

    list_CL = generateFrameList(filename_CL,currentStart,currentEnd,...
        '.tif');
    list_CM = generateFrameList(filename_CM,currentStart,currentEnd,...
        '.tif');

```

```

CL = tif2Video(list_CL);
CM = tif2Video(list_CM);

if noisy_frames
    list_noisyCL = generateFrameList(filename_noisyCL,currentStart,...
        currentEnd, '.tif');
    list_noisyCM = generateFrameList(filename_noisyCM,currentStart,...
        currentEnd, '.tif');

    noisyCL = tif2Video(list_noisyCL);
    noisyCM = tif2Video(list_noisyCM);
end

%%
difCLCM = CL-CM;

if noisy_frames
    noisydifCLCM = noisyCL-noisyCM;
end

if write_enable
    list = generateFrameList(filename_difCLCM,currentStart,...
        currentEnd, '.tif');
    writeTifs(difCLCM, list);

    if noisy_frames
        list = generateFrameList(filename_noisydifCLCM,currentStart,...
            currentEnd, '.tif');
        writeTifs(noisydifCLCM, list);
    end
end
end

%% WRITE VIDEO FROM IMAGES ON HD
if write_enable
    list = generateFrameList(filename_difCLCM,startFrame,endFrame, '.tif');
    writeMPEG4(filename_difCLCM, list, startFrame);

    if noisy_frames
        list = generateFrameList(filename_noisydifCLCM,startFrame,...
            endFrame, '.tif');
        writeMPEG4(filename_noisydifCLCM, list, startFrame);
    end
end

%%
toc;
%% End of Script -----%
```

```

%% Matlab Script -----%
% Script : STEP_3b_generate_altValidProcess_script.m
% Author : Raul-Alexander Aldama
% Date : 3/2/13
%
% Description : Performes alternative frame processing techniques,
% processes frames in sets of 30-frame length or 1 second
%
% Notes :   Video Sequences   Frames   Valid Frames (multiple of 30)
%           'setup1'           3753
%           'charcoal1'        4000      3990
%           'oats1'            8000      2100
%           'oats2'            395
%           'oats2b'           8000
%           'oats3'            8000
%           'pine_needles'     14934
%           'pine_cones'       8000
%           'palm'             4405
%           'thistle'          6670
%           'wet_leaves'       6063
%           'wet_dry1'         6117
%           'wet_dry2'         6364
%           'rosemary'         5609
%           'applewood1'       6100
%           'applewood2'       6100
%           'extinguish'       6100
%
% Revision(s) :
%-----%
clear all; close all; tic;

path = 'K:\Thesis\Data\';

filename = 'oats1';
startFrame = 1;
endFrame = 2100;
frameLength = 30;
temporalLength = 10;

noisy_frames = 0;
write_enable = 0;

videoDimension = [640 480];
intensityLim = 65535;
thresh = 4500;
adapThreshRes = 2^15;

threshCL_enable = 0;
adapthreshCL_enable = 0;
pcadifCLCM_enable = 0;
pcathreshCL_enable = 0;
pcaadapthreshCL_enable = 0;
pcaCL_enable = 0;
pcaCM_enable = 0;
pcaCLCM_enable = 0;
pcaTCLCM_enable = 1;

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% INPUT DATA PATHS
```

```
path_CL = strcat(path, 'INPUT_DATA\VALID\CL\', filename, '\');  
filename_CL = strcat(path_CL, filename, '_');
```

```
path_CM = strcat(path, 'INPUT_DATA\VALID\CM\', filename, '\');  
filename_CM = strcat(path_CM, filename, '_');
```

```
if noisy_frames  
    path_noisyCL = strcat(path, 'INPUT_DATA\VALID\noisyCL\', filename, '\');  
    filename_noisyCL = strcat(path_noisyCL, filename, '_');  
  
    path_noisyCM = strcat(path, 'INPUT_DATA\VALID\noisyCM\', filename, '\');  
    filename_noisyCM = strcat(path_noisyCM, filename, '_');  
end
```

```
% PROCESS_DATA PATHS
```

```
path_threshCL = strcat(path, 'PROCESS_DATA\VALID\threshCL\', filename, '\');  
if write_enable  
    [~,~,~] = mkdir(path_threshCL);  
end  
filename_threshCL = strcat(path_threshCL, filename, '_');
```

```
path_adapthreshCL = strcat(path, ...  
    'PROCESS_DATA\VALID\adapthreshCL\', filename, '\');  
if write_enable  
    [~,~,~] = mkdir(path_adapthreshCL);  
end  
filename_adapthreshCL = strcat(path_adapthreshCL, filename, '_');
```

```
path_pcaCL = strcat(path, 'PROCESS_DATA\VALID\pcaCL\', filename, '\');  
if write_enable  
    [~,~,~] = mkdir(path_pcaCL);  
end
```

```
path_pcaCM = strcat(path, 'PROCESS_DATA\VALID\pcaCM\', filename, '\');  
if write_enable  
    [~,~,~] = mkdir(path_pcaCM);  
end
```

```
path_pcadifCLCM = strcat(path, ...  
    'PROCESS_DATA\VALID\pcadifCLCM\', filename, '\');  
if write_enable  
    [~,~,~] = mkdir(path_pcadifCLCM);  
end
```

```
path_pcathreshCL = strcat(path, ...  
    'PROCESS_DATA\VALID\pcathreshCL\', filename, '\');  
if write_enable  
    [~,~,~] = mkdir(path_pcathreshCL);  
end
```

```
path_pcaadapthreshCL = strcat(path, ...  
    'PROCESS_DATA\VALID\pcaadapthreshCL\', filename, '\');
```



```

if write_enable
    [~,~,~] = mkdir(path_pcaadapthreshCL);
end

path_pcaCLCM = strcat(path,'PROCESS_DATA\VALID\pcaCLCM\',filename,'\');
if write_enable
    [~,~,~] = mkdir(path_pcaCLCM);
end

path_pcaTCLCM = strcat(path,'PROCESS_DATA\VALID\pcaTCLCM\',filename,'\');
if write_enable
    [~,~,~] = mkdir(path_pcaTCLCM);
end

if noisy_frames
    path_noisythreshCL = strcat(path,...
        'PROCESS_DATA\VALID\noisythreshCL\',filename,'\');
    if write_enable
        [~,~,~] = mkdir(path_noisythreshCL);
    end
    filename_noisythreshCL = strcat(path_noisythreshCL,filename,'_');

    path_noisyadapthreshCL = strcat(path,...
        'PROCESS_DATA\VALID\noisyadapthreshCL\',filename,'\');
    if write_enable
        [~,~,~] = mkdir(path_noisyadapthreshCL);
    end
    filename_noisyadapthreshCL = strcat(...
        path_noisyadapthreshCL,filename,'_');

    path_noisypcaCL = strcat(path,...
        'PROCESS_DATA\VALID\noisypcaCL\',filename,'\');
    if write_enable
        [~,~,~] = mkdir(path_noisypcaCL);
    end

    path_noisypcaCM = strcat(path,...
        'PROCESS_DATA\VALID\noisypcaCM\',filename,'\');
    if write_enable
        [~,~,~] = mkdir(path_noisypcaCM);
    end

    path_noisypcadifCLCM = strcat(path,...
        'PROCESS_DATA\VALID\noisypcadifCLCM\',filename,'\');
    if write_enable
        [~,~,~] = mkdir(path_noisypcadifCLCM);
    end

    path_noisypcathreshCL = strcat(path,...
        'PROCESS_DATA\VALID\noisypcathreshCL\',filename,'\');
    if write_enable
        [~,~,~] = mkdir(path_noisypcathreshCL);
    end

    path_noisypcaadapthreshCL = strcat(path,...

```

```

        'PROCESS_DATA\VALID\noisypcaadapthreshCL\',filename,'\');
    if write_enable
        [~,~,~] = mkdir(path_noisypcaadapthreshCL);
    end

    path_noisypcaCLCM = strcat(path,...
        'PROCESS_DATA\VALID\noisypcaCLCM\',filename,'\');
    if write_enable
        [~,~,~] = mkdir(path_noisypcaCLCM);
    end

    path_noisypcaTCLCM = strcat(path,...
        'PROCESS_DATA\VALID\noisypcaTCLCM\',filename,'\');
    if write_enable
        [~,~,~] = mkdir(path_noisypcaTCLCM);
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if pcaCL_enable
    pcaImages_CL = cell(1, temporalLength);
    pcaFrameBuffer_CL = zeros(videoDimension(2),...
        videoDimension(1),2*temporalLength);
end
if pcaCM_enable
    pcaImages_CM = cell(1, temporalLength);
    pcaFrameBuffer_CM = zeros(videoDimension(2),...
        videoDimension(1),2*temporalLength);
end
if pcadifCLCM_enable
    pcaImages_difCLCM = cell(1, temporalLength);
    pcaFrameBuffer_difCLCM = zeros(videoDimension(2),...
        videoDimension(1),2*temporalLength);
end
if pcatreshCL_enable
    pcaImages_threshCL = cell(1, temporalLength);
    pcaFrameBuffer_threshCL = zeros(videoDimension(2),...
        videoDimension(1),2*temporalLength);
end
if pcaadapthreshCL_enable
    pcaImages_adapthreshCL = cell(1, temporalLength);
    pcaFrameBuffer_adapthreshCL = zeros(videoDimension(2),...
        videoDimension(1),2*temporalLength);
end
if pcaCLCM_enable || pcaTCLCM_enable
    dataCell_CLCM = cell(1, 2);
    pcaImages_CLCM = cell(1, 2);
end
if pcaTCLCM_enable
    pcaImages_TCLCM = cell(1, temporalLength);
    pcaFrameBuffer_TCLCM = zeros(videoDimension(2),...
        videoDimension(1),2*temporalLength);
end
if noisy_frames
    pcaImages_noisyCL = cell(1, temporalLength);
    pcaFrameBuffer_noisyCL = zeros(videoDimension(2),...
        videoDimension(1),2*temporalLength);
end

```

```

pcaImages_noisyCM = cell(1, temporalLength);
pcaFrameBuffer_noisyCM = zeros(videoDimension(2), ...
    videoDimension(1), 2*temporalLength);

pcaImages_noisydifCLCM = cell(1, temporalLength);
pcaFrameBuffer_noisydifCLCM = zeros(videoDimension(2), ...
    videoDimension(1), 2*temporalLength);

pcaImages_noisythreshCL = cell(1, temporalLength);
pcaFrameBuffer_noisythreshCL = zeros(videoDimension(2), ...
    videoDimension(1), 2*temporalLength);

pcaImages_noisyadapthreshCL = cell(1, temporalLength);
pcaFrameBuffer_noisyadapthreshCL = zeros(videoDimension(2), ...
    videoDimension(1), 2*temporalLength);

dataCell_noisyCLCM = cell(1, 2);
pcaImages_noisyCLCM = cell(1, 2);

pcaImages_noisyTCLCM = cell(1, temporalLength);
pcaFrameBuffer_noisyTCLCM = zeros(videoDimension(2), ...
    videoDimension(1), 2*temporalLength);
end
%%
procFrames = frameLength*floor((endFrame-startFrame+1)/frameLength);
endFrame = startFrame + procFrames - 1;

disp(['Processing ', num2str(procFrames), ' Frames...']);

iterations = (endFrame-startFrame+1)/frameLength;
temporalIterations = frameLength/temporalLength;

adapthreshCLOverTime = zeros(1, procFrames, 'uint16');
if noisy_frames
    noisyadapthreshCLOverTime = zeros(1, procFrames, 'uint16');
end
%%
for n=1:iterations
    currentStart = startFrame+(n-1)*frameLength;
    currentEnd = n*frameLength + startFrame-1;

    list_CL = generateFrameList(filename_CL, currentStart, ...
        currentEnd, '.tif');
    list_CM = generateFrameList(filename_CM, currentStart, ...
        currentEnd, '.tif');

    CL = tif2Video(list_CL);
    CM = tif2Video(list_CM);

    if noisy_frames
        list_noisyCL = generateFrameList(filename_noisyCL, currentStart, ...
            currentEnd, '.tif');
        list_noisyCM = generateFrameList(filename_noisyCM, currentStart, ...

```

```

        currentEnd, '.tif');

        noisyCL = tif2Video(list_noisyCL);
        noisyCM = tif2Video(list_noisyCM);
    end

    if pcadifCLCM_enable
        difCLCM = CL-CM;
    end
    if threshCL_enable || pcatreshCL_enable
        threshCL = applyThreshVideo(CL, thresh, intensityLim);
    end
    if adapthreshCL_enable || pcaadapthreshCL_enable
        [adapthresh, pks_CL,...
         loc_CL, adapthresh_count,...
         adapthresh_bins] = findThreshVideo(CL,...
         adapThreshRes, adapThreshRes, adapThreshRes, 0);
        adapthreshCLOverTime(...
            currentStart:currentEnd) = adapthresh;
        adapthreshCL = applyThreshVideo(...
            CL, adapthresh, intensityLim);
    end
    if noisy_frames
        noisydifCLCM = noisyCL-noisyCM;

        noisythreshCL = applyThreshVideo(noisyCL, thresh, intensityLim);

        [noisyadapthresh, pks_noisyCL,...
         loc_noisyCL, noisyadapthresh_count,...
         noisyadapthresh_bins] = findThreshVideo(noisyCL,...
         adapThreshRes, adapThreshRes, adapThreshRes, 0);
        noisyadapthreshCLOverTime(...
            currentStart:currentEnd) = noisyadapthresh;
        noisyadapthreshCL = applyThreshVideo(...
            noisyCL, noisyadapthresh, intensityLim);
    end

    if write_enable

        list = generateFrameList(filename_threshCL,...
            currentStart,currentEnd, '.tif');
        writeTifs(threshCL, list);

        list = generateFrameList(filename_adapthreshCL,...
            currentStart,currentEnd, '.tif');
        writeTifs(adapthreshCL, list);

        if noisy_frames
            list = generateFrameList(filename_noisythreshCL,...
                currentStart,currentEnd, '.tif');
            writeTifs(noisythreshCL, list);

            list = generateFrameList(filename_noisyadapthreshCL,...
                currentStart,currentEnd, '.tif');
            writeTifs(noisyadapthreshCL, list);
        end
    end
end

```

```

        end
    end

    if pcaCLCM_enable || pcaTCLCM_enable
        dataCell_CLCM{1} = CL;
        dataCell_CLCM{2} = CM;
        [pcaImages_CLCM] = spectralPCA(dataCell_CLCM, pcaImages_CLCM);
    end
    if noisy_frames
        dataCell_noisyCLCM{1} = noisyCL;
        dataCell_noisyCLCM{2} = noisyCM;
        [pcaImages_noisyCLCM] = spectralPCA(dataCell_noisyCLCM, ...
            pcaImages_noisyCLCM);
    end

    if write_enable
        writeTifs_PCA(pcaImages_CLCM, path_pcaCLCM, currentStart, ...
            currentEnd, 0, filename, 1);

        if noisy_frames
            writeTifs_PCA(pcaImages_noisyCLCM, path_noisypcaCLCM, ...
                currentStart, currentEnd, 0, filename, 1);
        end
    end

    for i = 1:temporalIterations
        if n == 1 && i == 1
            enable = 0;
        else
            enable = 1;
        end

        startIndex = 1 + (i-1)*temporalLength;
        endIndex = startIndex + temporalLength - 1;

        if write_enable
            startWrite = currentStart + (i-1)*temporalLength;
            endWrite = startWrite + temporalLength - 1;
        end

        if pcaCL_enable
            [pcaImages_CL, pcaFrameBuffer_CL] = temporalPCA(...
                CL(:, :, startIndex:endIndex), ...
                pcaImages_CL, pcaFrameBuffer_CL, enable);
        end
        if write_enable
            writeTifs_PCA(pcaImages_CL, path_pcaCL, ...
                startWrite, endWrite, temporalLength, filename, enable);
        end

        if pcaCM_enable
            [pcaImages_CM, pcaFrameBuffer_CM] = temporalPCA(...
                CM(:, :, startIndex:endIndex), ...
                pcaImages_CM, pcaFrameBuffer_CM, enable);
        end
    end
end

```

```

if write_enable
    writeTifs_PCA(pcaImages_CM, path_pcaCM,...
        startWrite, endWrite, temporalLength, filename, enable);
end

if pcadifCLCM_enable
    [pcaImages_difCLCM, pcaFrameBuffer_difCLCM] = temporalPCA(...
        difCLCM(:, :, startIndex:endIndex), ...
        pcaImages_difCLCM, pcaFrameBuffer_difCLCM, enable);
end
if write_enable
    writeTifs_PCA(pcaImages_difCLCM, path_pcadifCLCM,...
        startWrite, endWrite, temporalLength, filename, enable);
end

if pcatreshCL_enable
    [pcaImages_threshCL, pcaFrameBuffer_threshCL] = temporalPCA(...
        threshCL(:, :, startIndex:endIndex), ...
        pcaImages_threshCL, pcaFrameBuffer_threshCL, enable);
end
if write_enable
    writeTifs_PCA(pcaImages_threshCL, path_pcatreshCL,...
        startWrite, endWrite, temporalLength, filename, enable);
end

if pcaadapthreshCL_enable
    [pcaImages_adapthreshCL, ...
        pcaFrameBuffer_adapthreshCL] = temporalPCA(...
        adapthreshCL(:, :, startIndex:endIndex), ...
        pcaImages_adapthreshCL, pcaFrameBuffer_adapthreshCL, ...
        enable);
end
if write_enable
    writeTifs_PCA(pcaImages_adapthreshCL, path_pcaadapthreshCL,...
        startWrite, endWrite, temporalLength, filename, enable);
end

if pcaTCLCM_enable
    [pcaImages_TCLCM, pcaFrameBuffer_TCLCM] = temporalPCA(...
        pcaImages_CLCM{2}(:, :, startIndex:endIndex), ...
        pcaImages_TCLCM, pcaFrameBuffer_TCLCM, enable);
end
if write_enable
    writeTifs_PCA(pcaImages_TCLCM, path_pcaTCLCM,...
        startWrite, endWrite, temporalLength, filename, enable);
end

if noisy_frames
    [pcaImages_noisyCL, pcaFrameBuffer_noisyCL] = temporalPCA(...
        noisyCL(:, :, startIndex:endIndex), ...
        pcaImages_noisyCL, pcaFrameBuffer_noisyCL, enable);
    if write_enable
        writeTifs_PCA(pcaImages_noisyCL, path_noisypcaCL,...
            startWrite, endWrite, temporalLength, filename, ...
            enable);
    end
end

```

```

[pcaImages_noisyCM, pcaFrameBuffer_noisyCM] = temporalPCA(...
    noisyCM(:, :, startIndex:endIndex), ...
    pcaImages_noisyCM, pcaFrameBuffer_noisyCM, enable);
if write_enable
    writeTifs_PCA(pcaImages_noisyCM, path_noisypcaCM, ...
        startWrite, endWrite, temporalLength, filename, ...
        enable);
end

[pcaImages_noisydifCLCM, ...
    pcaFrameBuffer_noisydifCLCM] = temporalPCA(...
    noisydifCLCM(:, :, startIndex:endIndex), ...
    pcaImages_noisydifCLCM, pcaFrameBuffer_noisydifCLCM, ...
    enable);
if write_enable
    writeTifs_PCA(pcaImages_noisydifCLCM, ...
        path_noisypcadifCLCM, startWrite, endWrite, ...
        temporalLength, filename, enable);
end

[pcaImages_noisythreshCL, ...
    pcaFrameBuffer_noisythreshCL] = temporalPCA(...
    noisythreshCL(:, :, startIndex:endIndex), ...
    pcaImages_noisythreshCL, pcaFrameBuffer_noisythreshCL, ...
    enable);
if write_enable
    writeTifs_PCA(pcaImages_noisythreshCL, ...
        path_noisypcathreshCL, startWrite, endWrite, ...
        temporalLength, filename, enable);
end

[pcaImages_noisyadapthreshCL, ...
    pcaFrameBuffer_noisyadapthreshCL] = temporalPCA(...
    noisyadapthreshCL(:, :, startIndex:endIndex), ...
    pcaImages_noisyadapthreshCL, ...
    pcaFrameBuffer_noisyadapthreshCL, enable);
if write_enable
    writeTifs_PCA(pcaImages_noisyadapthreshCL, ...
        path_noisypcaadapthreshCL, startWrite, endWrite, ...
        temporalLength, filename, enable);
end

[pcaImages_noisyTCLCM, ...
    pcaFrameBuffer_noisyTCLCM] = temporalPCA(...
    pcaImages_noisyCLCM{2}(:, :, startIndex:endIndex), ...
    pcaImages_noisyTCLCM, pcaFrameBuffer_noisyTCLCM, enable);
if write_enable
    writeTifs_PCA(pcaImages_noisyTCLCM, ...
        path_noisypcaTCLCM, startWrite, endWrite, ...
        temporalLength, filename, enable);
end
end
end
end
end

```

```

%% WRITE VIDEO FROM IMAGES ON HD
if write_enable
    list = generateFrameList(filename_threshCL,startFrame,endFrame,...
        '.tif');
    writeMPEG4(filename_threshCL, list, startFrame);

    list = generateFrameList(filename_adapthreshCL,startFrame,endFrame,...
        '.tif');
    writeMPEG4(filename_adapthreshCL, list, startFrame);

    writeMPEG4_PCA(path_pcaCLCM,startFrame,endFrame,...
        length(dataCell_CLCM),0,filename);

    writeMPEG4_PCA(path_pcaCL,startFrame,endFrame,...
        temporalLength,temporalLength,filename);

    writeMPEG4_PCA(path_pcaCM,startFrame,endFrame,...
        temporalLength,temporalLength,filename);

    writeMPEG4_PCA(path_pcadifCLCM,startFrame,endFrame,...
        temporalLength,temporalLength,filename);

    writeMPEG4_PCA(path_pcathreshCL,startFrame,endFrame,...
        temporalLength,temporalLength,filename);

    writeMPEG4_PCA(path_pcaadapthreshCL,startFrame,endFrame,...
        temporalLength,temporalLength,filename);

    writeMPEG4_PCA(path_pcaTCLCM,startFrame,endFrame,...
        temporalLength,temporalLength,filename);

if noisy_frames
    list = generateFrameList(filename_noisythreshCL,...
        startFrame,endFrame,'.tif');
    writeMPEG4(filename_noisythreshCL, list, startFrame);

    list = generateFrameList(filename_noisyadapthreshCL,...
        startFrame,endFrame,'.tif');
    writeMPEG4(filename_noisyadapthreshCL, list, startFrame);

    writeMPEG4_PCA(path_noisypcaCLCM,startFrame,endFrame,...
        length(dataCell_noisyCLCM),0,filename);

    writeMPEG4_PCA(path_noisypcaCL,startFrame,endFrame,...
        temporalLength,temporalLength,filename);

    writeMPEG4_PCA(path_noisypcaCM,startFrame,endFrame,...
        temporalLength,temporalLength,filename);

    writeMPEG4_PCA(path_noisypcadifCLCM,startFrame,endFrame,...
        temporalLength,temporalLength,filename);

    writeMPEG4_PCA(path_noisypcathreshCL,startFrame,endFrame,...
        temporalLength,temporalLength,filename);

```



```

        writeMPEG4_PCA(path_noisypcaadapthreshCL,startFrame,endFrame,...
            temporalLength,temporalLength,filename);

        writeMPEG4_PCA(path_noisypcaTCLCM,startFrame,endFrame,...
            temporalLength,temporalLength,filename);
    end
end
%%
toc;
%% End of Script -----%
```

```

%% Matlab Script -----%
% Script : STEP_4_generate_validTrainingVideo_script.m
% Author : Raul-Alexander Aldama
% Date : 3/3/13
%
% Description : Generates frames for MLP Neural Network Training
%
% Notes :   Video Sequences   Frames   Valid Frames (multiple of 30)
%           'setup1'          3753
%           'charcoal1'       4000      3990
%           'oats1'           8000      2100
%           'oats2'           395
%           'oats2b'          8000
%           'oats3'           8000
%           'pine_needles'    14934
%           'pine_cones'      8000
%           'palm'            4405
%           'thistle'         6670
%           'wet_leaves'      6063
%           'wet_dry1'        6117
%           'wet_dry2'        6364
%           'rosemary'        5609
%           'applewood1'      6100
%           'applewood2'      6100
%           'extinguish'      6100
%
%           Frame Processing Technique
%           'difCLCM'
%           'threshCL'
%           'adapthreshCL'
%           'CL'
%           'CM'
%           'pcadifCLCM'
%           'pcathreshCL'
%           'pcaCL'
%           'pcaCM'
%           'pcaCLCM'
%           'pcaTCLCM'
%
% Revision(s) :
%-----%

clear all; close all;

path = 'K:\Thesis\Data\';

filename = 'oats1';

inputSource = 'CM';
noisySource = 1;
pcaInputNumber = 0;

refSource = inputSource;
pcaRefNumber = 0;

startFrame = 1;
endFrame = 2100;

```

```

frameLength = 30;

write_enable = 1;
plot_enable = 1;
viewFrame = 1;

vidRes = [640,480];
targetUpperLeftCorner = [305 311]; % [X Y]
targetBottomRightCorner = [390 375]; % [X Y]
targetRows = targetUpperLeftCorner(2):targetBottomRightCorner(2);
targetCols = targetUpperLeftCorner(1):targetBottomRightCorner(1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INPUT DATA PATHS
if noisySource
    inputSource = strcat('noisy',inputSource);
end
path_inputVideo = strcat(path,'PROCESS_DATA\VALID\',inputSource,'\',...
    filename,'\');
if pcaInputNumber > 0
    path_inputVideo = strcat(path_inputVideo,'PCA_',...
        num2str(pcaInputNumber),'\');
end
filename_inputVideo = strcat(path_inputVideo,filename,'_');

path_refVideo = strcat(path,'PROCESS_DATA\VALID\',refSource,'\',...
    filename,'\');
if pcaRefNumber > 0
    path_refVideo = strcat(path_refVideo,'PCA_',...
        num2str(pcaRefNumber),'\');
end
filename_refVideo = strcat(path_refVideo,filename,'_');

% OUTPUT DATA PATHS
path_trainInput = strcat(path,'TRAIN_DATA\VALID\trainInput\',...
    inputSource,'\ ',filename,'\');
if pcaInputNumber > 0
    path_trainInput = strcat(path_trainInput,'PCA_',...
        num2str(pcaInputNumber),'\');
end
if write_enable
    [~,~,~] = mkdir(path_trainInput);
end
filename_trainInput = strcat(path_trainInput,filename,'_');

%%

procFrames = frameLength*floor((endFrame-startFrame+1)/frameLength);
endFrame = startFrame + procFrames - 1;
disp(['Processing ',num2str(procFrames),' Frames...']);
iterations = (endFrame-startFrame+1)/frameLength;

for n=1:iterations
    currentStart = startFrame+(n-1)*frameLength;
    currentEnd = n*frameLength + startFrame-1;

```

```

frameList = generateFrameList(filename_inputVideo,...
    currentStart,currentEnd,'.tif');
inputVideo = tif2Video(frameList);

frameList = generateFrameList(filename_refVideo,...
    currentStart,currentEnd,'.tif');
refVideo = tif2Video(frameList);

trainVideo = zeros(size(inputVideo),'uint16');

if noisySource
    for i=1:frameLength
        image = inputVideo(:,:,i);
        refImage = refVideo(:,:,i);

        image(targetRows,targetCols) = refImage(targetRows,targetCols);
        trainVideo(:,:,i) = image;
    end
else
    trainVideo = inputVideo;
end

if plot_enable && n == 1
    figure()
    subplot(1,3,1)
    imshow(inputVideo(:,:,viewFrame));
    subplot(1,3,2)
    imshow(refVideo(:,:,viewFrame));
    subplot(1,3,3)
    imshow(trainVideo(:,:,viewFrame));

    figure()
    imshowpair(inputVideo(:,:,viewFrame),...
        trainVideo(:,:,viewFrame),'Scaling','none');
end

if write_enable
    list = generateFrameList(filename_trainInput,...
        currentStart,currentEnd,'.tif');
    writeTifs(trainVideo, list);
end
end
%% WRITE VIDEO FROM IMAGES ON HD
if write_enable
    list = generateFrameList(filename_trainInput,startFrame,endFrame,...
        '.tif');
    writeMPEG4(filename_trainInput, list, startFrame);
end
%%
%% End of Script -----%
```

```

%% Matlab Script -----%
% Script : STEP_5_extract_validTrainingFeatures_script.m
% Author : Raul-Alexander Aldama
% Date : 3/11/13
%
% Description : Extracts Feature Points for MLP Neural Network Training
%
% Notes :   Video Sequences   Frames   Valid Frames (multiple of 30)
%           'setup1'           3753
%           'charcoal1'        4000      3990
%           'oats1'            8000      2100
%           'oats2'            395
%           'oats2b'           8000
%           'oats3'            8000
%           'pine_needles'     14934
%           'pine_cones'       8000
%           'palm'             4405
%           'thistle'          6670
%           'wet_leaves'       6063
%           'wet_dry1'         6117
%           'wet_dry2'         6364
%           'rosemary'         5609
%           'applewood1'       6100
%           'applewood2'       6100
%           'extinguish'       6100
%
%           Frame Processing Technique   Gaussian Sigma
%           'difCLCM'                   0.25
%           'threshCL'                   0.218
%           'adapthreshCL'               0.218
%           'CL'                         0.24
%           'CM'
%           'pcadifCLCM'
%           'pcathreshCL'
%           'pcaCL'
%           'pcaCM'
%           'pcaCLCM'
%           'pcaTCLCM'
%
% Revision(s) :
%-----%

clear all;
close all;

path = 'K:\Thesis\Data\';

filename = 'oats1';

inputSource = 'difCLCM';
noisySource = 0;
pcaInputNumber = 0;

refSource = 'CM';

startFrame = 1;
endFrame = 2100;

```

```

frameLength = 30;

trainLengthLim = 50000;

write_enable = 0;
plot_enable = 1;
viewFrame = 10;

filtSqrSize = 25;
unsharpStrength = 1;
gausSigma = 0.25;

maxCoefVariationLim = 3.5;
maxNeighborDist = 5;
maxCorrCoefLim = 0.8;
maxCorrLagLim = 5;

distNH = 12;

intensityLim = 65535;

targetUpperLeftCorner = [305 311]; % [X Y]
targetBottomRightCorner = [390 375]; % [X Y]
targetRows = targetUpperLeftCorner(2):targetBottomRightCorner(2);
targetCols = targetUpperLeftCorner(1):targetBottomRightCorner(1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INPUT DATA PATHS
if noisySource
    inputSource = strcat('noisy',inputSource);
    refSource = strcat('noisy',refSource);
end

path_inputVideo = strcat(path,'TRAIN_DATA\VALID\trainInput\',...
    inputSource,'\ ',filename,'\ ');
if pcaInputNumber > 0
    path_inputVideo = strcat(path_inputVideo,'PCA_',...
        num2str(pcaInputNumber),'\ ');
end
filename_inputVideo = strcat(path_inputVideo,filename,'_');

path_refVideo = strcat(path,'INPUT_DATA\VALID\ ',...
    refSource,'\ ',filename,'\ ');
diagOffset = strcat(path_refVideo,'diagOffset.mat');
filename_refVideo = strcat(path_refVideo,filename,'_');

% OUTPUT DATA PATHS
path_trainOutput = strcat(path,'TRAIN_DATA\VALID\trainOutput\ ',...
    inputSource,'\ ',filename,'\ ');
if pcaInputNumber > 0
    path_trainOutput = strcat(path_trainOutput,'PCA_',...
        num2str(pcaInputNumber),'\ ');
end
if write_enable
    [~,~,~] = mkdir(path_trainOutput);
end

```

```

filename_trainOutput = strcat(path_trainOutput,filename,'_');
%%

if write_enable
    trainIndex = 1;
    trainMatrix = NaN.*ones(6, trainLengthLim);
    breakFlag = 0;
end

procFrames = frameLength*floor((endFrame-startFrame+1)/frameLength);
endFrame = startFrame + procFrames - 1;
disp(['Processing ',num2str(procFrames),' Frames...']);
iterations = (endFrame-startFrame+1)/frameLength;

try
    load(diagOffset);
catch exception
    diagOffset = 10.*ones(1, procFrames);
end

for n=1:iterations
    currentStart = startFrame+(n-1)*frameLength;
    currentEnd = n*frameLength + startFrame-1;

    frameList = generateFrameList(filename_inputVideo,...
        currentStart,currentEnd,'.tif');
    originalVideo = tif2Video(frameList);
    inputVideo = originalVideo;

    frameList = generateFrameList(filename_refVideo,...
        currentStart,currentEnd,'.tif');
    refVideo = tif2Video(frameList);

    if noisySource
        inputVideo = medFilterVideo(inputVideo, 3);
    end

    [~, ~, loc, ~, ~] = findThreshVideo(refVideo,2^16,10,10,1);
    bottomClip = mean(loc);

    [featureTable, feature1Title, feature2Title,...
        feature3Title, featureInputVideo, featureOutputVideo,...
        featureMask] = extractFireFeatures(...
        inputVideo, refVideo, filename,...
        inputSource, refSource, frameLength,...
        currentStart, currentEnd, intensityLim,...
        bottomClip, diagOffset,...
        targetRows, targetCols, unsharpStrength,...
        filtSqrSize, gausSigma,...
        maxCoefVariationLim, maxNeighborDist,...
        maxCorrCoefLim, maxCorrLagLim,...
        distNH, plot_enable, viewFrame, 1 );

    if write_enable
        if n == 1

```

```

        [writerObj_featureSpace] = objectFigMPEG4(strcat(...
            filename_trainOutput,inputSource,'_featureSpace'), 1);
        figHandle_featureSpace = 27;
    end
    if n ~= iterations
        writeFigMPEG4(writerObj_featureSpace,figHandle_featureSpace,0)
    end
    if n == iterations
        writeFigMPEG4(writerObj_featureSpace,figHandle_featureSpace,1)
    end

    if n == iterations
        view(45,15)
        [featureSpaceFrame,~] = getframe(figHandle_featureSpace);
        imwrite(featureSpaceFrame,[filename_trainOutput,inputSource,...
            '_featureSpaceFrame.tif'],'tif','Compression','none')
    end
end
if write_enable
    for i=1:size(featureTable,1)
        if trainIndex <= trainLengthLim
            trainMatrix(:,trainIndex) = featureTable(i,2:7)';
        else
            breakFlag = 1;
            disp('Generation of Training Data Complete!')
            break;
        end
        trainIndex = trainIndex + 1;
    end
    disp(['Generated ',num2str(trainIndex),' Samples...'])
    if breakFlag
        break;
    end
end
end

if write_enable
    trainNum = size(trainMatrix,1);
    trainMatrix = trainMatrix(isfinite(trainMatrix));
    trainMatrix = reshape(trainMatrix, trainNum,...
        length(trainMatrix)/trainNum);
    save([filename_trainOutput,inputSource,...
        '_trainMatrix'],'trainMatrix','-v7.3');
end
%% End of Script -----%
```



```

%% Matlab Script -----%
% Script : STEP_6_generate_validNN_script
% Author : Raul-Alexander Aldama
% Date : 3/23/13
%
% Description : Generates and trains the MLP Neural Network used for
% feature point classification
%
% Notes :    Video Sequences
%            'setup1'
%            'charcoal1'
%            'oats1'
%            'oats2'
%            'oats2b'
%            'oats3'
%            'pine_needles'
%            'pine_cones'
%            'palm'
%            'thistle'
%            'wet_leaves'
%            'wet_dry1'
%            'wet_dry2'
%            'rosemary'
%            'applewood1'
%            'applewood2'
%            'extinguish'
%
%            Frame Processing Technique
%            'difCLCM'
%            'threshCL'
%            'adapthreshCL'
%            'CL'
%            'CM'
%            'pcadifCLCM'
%            'pcathreshCL'
%            'pcaCL'
%            'pcaCM'
%            'pcaCLCM'
%            'pcaTCLCM'
%
% Revision(s) :
%-----%
clear all; close all;

path = 'K:\Thesis\Data\';

write_enable = 0;

% INPUT/OUTPUT DATA PATHS
path_trainOutput = strcat(path, 'TRAIN_DATA\VALID\trainOutput\');

trainData = [];
filenames = {'oats1'};
inputSources = {'difCLCM', 'threshCL', 'adapthreshCL', 'CL'};
for j=1:length(filenames)
    filename = filenames{j};

```

```

    for i=1:length(inputSources)
        inputSource = inputSources{i};
        trainMatrix = strcat(path_trainOutput,inputSource,'\',...
            filename,'\ ',filename,'_',inputSource,'_trainMatrix.mat');
        load(trainMatrix);
        trainData = cat(2,trainData, trainMatrix);
    end
end

featureInput = trainData(3:5,:);
featureInput_1 = trainData(3,:);
featureInput_2 = trainData(4,:);
featureInput_3 = trainData(5,:);
featureTarget = trainData(6,:);

positive_feature1 = featureInput_1(featureTarget == 1);
negative_feature1 = featureInput_1(featureTarget == 0);

positive_feature2 = featureInput_2(featureTarget == 1);
negative_feature2 = featureInput_2(featureTarget == 0);

positive_feature3 = featureInput_3(featureTarget == 1);
negative_feature3 = featureInput_3(featureTarget == 0);

figure(1)
hold on
h1 = scatter3(positive_feature1,positive_feature2,positive_feature3,...
    '+','red');
h2 = scatter3(negative_feature1,negative_feature2,negative_feature3,...
    '.','blue');
hold off
grid on
xlabel('NH STD DEV')
ylabel('NH MEAN')
zlabel('NH AVG AUTOCORR')
view(45,15)
legend([h1 h2],{'Fire', 'Noise'},'Location','NorthEast')
title('Feature Space of Training Samples')

if write_enable
    inputs = featureInput;
    targets = featureTarget;

    % Create a Pattern Recognition Network
    hiddenLayerSize = 10;
    net = patternnet(hiddenLayerSize);

    % Setup Division of Data for Training, Validation, Testing
    % For a list of all data division functions type: help nndivide
    net.divideFcn = 'dividerand'; % Divide data randomly
    net.divideMode = 'sample'; % Divide up every sample
    net.divideParam.trainRatio = 50/100;
    net.divideParam.valRatio = 25/100;
    net.divideParam.testRatio = 25/100;
end

```

```

% For help on training function 'trainlm' type: help trainlm
% For a list of all training functions type: help nntrain
net.trainFcn = 'trainlm'; % Levenberg-Marquardt
net.trainParam.goal = 1e-3;

% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
net.performFcn = 'mse'; % Mean squared error

% Choose Plot Functions
% For a list of all plot functions type: help nnplot
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
    'plotregression', 'plotfit', 'plotconfusion', 'plotinerrcorr',...
    'ploterrcorr','plotresponse', 'plotwb','plotroc'};

% Train the Network
[net,tr] = train(net,inputs,targets);

% Test the Network
outputs = net(inputs);
errors = gsubtract(targets,outputs);
performance = perform(net,targets,outputs);

% Recalculate Training, Validation and Test Performance
trainTargets = targets .* tr.trainMask{1};
valTargets = targets .* tr.valMask{1};
testTargets = targets .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,outputs);
valPerformance = perform(net,valTargets,outputs);
testPerformance = perform(net,testTargets,outputs);

% View the Network
view(net)

% Save Variables
save([path_trainOutput, 'net.mat'],'net','-v7.3');
save([path_trainOutput, 'inputs.mat'],'inputs','-v7.3');
save([path_trainOutput, 'outputs.mat'],'outputs','-v7.3');
save([path_trainOutput, 'targets.mat'],'targets','-v7.3');
save([path_trainOutput, 'tr.mat'],'tr','-v7.3');
save([path_trainOutput, 'errors.mat'],'errors','-v7.3');
else
    load([path_trainOutput, 'net.mat']);
    load([path_trainOutput, 'inputs.mat']);
    load([path_trainOutput, 'outputs.mat']);
    load([path_trainOutput, 'targets.mat']);
    load([path_trainOutput, 'tr.mat']);
    load([path_trainOutput, 'errors.mat']);
end

% Plots
figure(2), plotperform(tr)
figure(3), plottrainstate(tr)
figure(4), ploterrhist(errors)

```

```
figure(5), plotregression(targets,outputs)
figure(6), plotconfusion(targets,outputs)
figure(7), plotinerrcorr(inputs,errors)
figure(8), ploterrcorr(errors)
figure(9), plotresponse(targets,outputs)
figure(10), plotwb(net)
figure(11), plotroc(targets,outputs)
%% End of Script -----%
```

```

%% Matlab Script -----%
% Script : STEP_7_classify_validVideoFeatures_script.m
% Author : Raul-Alexander Aldama
% Date : 4/1/13
%
% Description : Classifies extracted Feature Points with MLP Neural Network
%
% Notes :   Video Sequences   Frames   Valid Frames (multiple of 30)
%           'setup1'           3753
%           'charcoal1'        4000      3990
%           'oats1'            8000      2100
%           'oats2'            395
%           'oats2b'           8000
%           'oats3'            8000
%           'pine_needles'     14934
%           'pine_cones'       8000
%           'palm'             4405
%           'thistle'          6670
%           'wet_leaves'       6063
%           'wet_dry1'         6117
%           'wet_dry2'         6364
%           'rosemary'         5609
%           'applewood1'       6100
%           'applewood2'       6100
%           'extinguish'       6100
%
%           Frame Processing Technique   Gaussian Sigma
%           'difCLCM'                   0.25
%           'threshCL'                  0.218
%           'adapthreshCL'              0.218
%           'CL'                        0.24
%           'CM'                        0.24
%           'pcadifCLCM'                0.25
%           'pcathreshCL'               0.216
%           'pcaadapthreshCL'          0.218
%           'pcaCL'                    0.24
%           'pcaCM'                    0.24
%           'pcaCLCM'                  0.24
%           'pcaTCLCM'                 0.25
%
% Revision(s) :
% -----%

clear all; close all; tic;

path = 'K:\Thesis\Data\';

filename = 'oats1';

inputSource = 'difCLCM';
pcaInputNumber = 0;
noisySource = 0;

refSource = 'CM';
compareSource = 'CL';

registered_video = 1;

```

```

startFrame = 1;
endFrame = 2100;
frameLength = 30;

alarm_enable = 0;
write_enable = 0;
plot_enable = 1;
viewFrame = 1;

gausSigma = 0.25;
filtSqrSize = 25;
unsharpStrength = 1;

maxCoefVariationLim = 3.5;
maxNeighborDist = 5;
maxCorrCoefLim = 0.8;
maxCorrLagLim = 5;

distNH = 12;

enhancedLim = 2*4500;
intensityLim = 65535;

hotColorMap = 'hot';
coolColorMap = 'bone';

classifyThresh = 0.5;
classifyLengthLim = 50000;

targetUpperLeftCorner = [305 311]; % [X Y]
targetBottomRightCorner = [390 375]; % [X Y]
targetRows = targetUpperLeftCorner(2):targetBottomRightCorner(2);
targetCols = targetUpperLeftCorner(1):targetBottomRightCorner(1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INPUT DATA PATHS
if noisySource
    inputSource = strcat('noisy',inputSource);
    refSource = strcat('noisy',refSource);
    compareSource = strcat('noisy',compareSource);
end

path_inputVideo = strcat(path,'PROCESS_DATA\VALID\',inputSource,'\',...
    filename,'\');
if pcaInputNumber > 0
    path_inputVideo = strcat(path_inputVideo,'PCA_',...
        num2str(pcaInputNumber),'\');
end
if ~registered_video
    path_inputVideo = strcat(path_inputVideo,'UNREG\');
end
filename_inputVideo = strcat(path_inputVideo,filename,'_');

path_refVideo = strcat(path,'INPUT_DATA\VALID\',refSource,'\',...
    filename,'\');
diagOffset = strcat(path_refVideo,'diagOffset.mat');
if ~registered_video
    path_refVideo = strcat(path_refVideo,'UNREG\');

```

```

end
filename_refVideo = strcat(path_refVideo,filename,'_');

path_compareVideo = strcat(path,'INPUT_DATA\VALID\',compareSource,...
    '\',filename,'\');
if ~registered_video
    path_compareVideo = strcat(path_compareVideo,'UNREG\');
end
filename_compareVideo = strcat(path_compareVideo,filename,'_');

path_trainOutput = strcat(path,'TRAIN_DATA\VALID\trainOutput\');
net = strcat(path_trainOutput,'net.mat');

% OUTPUT_DATA PATHS
path_output = strcat(path,'OUTPUT_DATA\VALID\',inputSource,'\',...
    filename,'\');
if pcaInputNumber > 0
    path_output = strcat(path_output,'PCA_',num2str(pcaInputNumber),'\');
end
if ~registered_video
    path_output = strcat(path_output,'UNREG\');
end
if write_enable
    [~,~,~] = mkdir(path_output);
end
filename_output = strcat(path_output,filename,'_');
%%
if write_enable
    classifyIndex = 1;
    classifyMatrix = NaN.*ones(8, classifyLengthLim);
    breakFlag = 0;
end

procFrames = frameLength*floor((endFrame-startFrame+1)/frameLength);
endFrame = startFrame + procFrames - 1;
disp(['Processing ',num2str(procFrames),' ', filename, ' ', inputSource,...
    ' Frames...']);
iterations = (endFrame-startFrame+1)/frameLength;

classifyPositiveFrames = zeros(1, iterations);
classifyNegativeFrames = zeros(1, iterations);
classifySuccessfulPositiveFrames = zeros(1, iterations);
classifySuccessfulNegativeFrames = zeros(1, iterations);
classifyFalsePositiveFrames = zeros(1, iterations);
classifyFalseNegativeFrames = zeros(1, iterations);

try
    load(diagOffset);
catch exception
    diagOffset = 10.*ones(1, procFrames);
end

load(net);

for n=1:iterations
    currentStart = startFrame+(n-1)*frameLength;
    currentEnd = n*frameLength + startFrame-1;

```

```

classifyPositiveSampleNum = 0;
classifyNegativeSampleNum = 0;
classifySuccessfulPositiveSampleNum = 0;
classifySuccessfulNegativeSampleNum = 0;
classifyFalsePositiveSampleNum = 0;
classifyFalseNegativeSampleNum = 0;

frameList = generateFrameList(filename_inputVideo,...
    currentStart,currentEnd,'.tif');
inputVideo = tif2Video(frameList);

frameList = generateFrameList(filename_refVideo,...
    currentStart,currentEnd,'.tif');
refVideo = tif2Video(frameList);

frameList = generateFrameList(filename_compareVideo,...
    currentStart,currentEnd,'.tif');
compareVideo = tif2Video(frameList);

[~, ~, loc, ~, ~] = findThreshVideo(refVideo,2^16,10,10,1);
bottomClip = mean(loc);

[featureTable, feature1Title, feature2Title,...
    feature3Title, featureInputVideo, featureOutputVideo,...
    featureMask] = extractFireFeatures(...
    inputVideo, refVideo, filename,...
    inputSource, refSource, frameLength,...
    currentStart, currentEnd, intensityLim,...
    bottomClip, diagOffset,...
    targetRows, targetCols, unsharpStrength,...
    filtSqrSize, gausSigma,...
    maxCoefVariationLim, maxNeighborDist,...
    maxCorrCoefLim, maxCorrLagLim,...
    distNH, plot_enable, viewFrame, 1 );

if plot_enable && write_enable
    if n == 1
        [writerObj_featureSpace] = objectFigMPEG4(...
            strcat(filename_output,inputSource,'_featureSpace'), 1);
        figHandle_featureSpace = 27;
    end
    if n ~= iterations
        writeFigMPEG4(writerObj_featureSpace,figHandle_featureSpace,0)
    end
    if n == iterations
        writeFigMPEG4(writerObj_featureSpace,figHandle_featureSpace,1)

        view(45,15)
        [featureSpaceFrame,~] = getframe(figHandle_featureSpace);
        imwrite(featureSpaceFrame,[filename_output,inputSource,...
            '_featureSpaceFrame.tif'],'tif','Compression','none')
    end
end
classifyInput = featureTable(:,4:6)';
classifyTarget = featureTable(:,7)';

```



```

classifyOutput = net(classifyInput);
featureTable(:,8) = classifyOutput';
classifyThreshOutput = zeros(size(classifyOutput));
for i=1:length(classifyOutput)
    if classifyOutput(i) >= classifyThresh
        classifyThreshOutput(i) = 1;
    else
        classifyThreshOutput(i) = 0;
    end
end
featureTable(:,9) = classifyThreshOutput';

classifySampleNum = size(featureTable,1);

dimFeatureVideo = size(featureOutputVideo);

featureVideo = zeros(dimFeatureVideo, 'uint16');
featureMaskIndecies = logical(repmat(featureMask, [1 1 frameLength]));
featureVideo(featureMaskIndecies) = inputVideo(featureMaskIndecies);

classifyPositiveMask = false(dimFeatureVideo(1), dimFeatureVideo(2));
classifyPositiveVideo = zeros(dimFeatureVideo, 'uint16');
for i=1:classifySampleNum
    if classifyThreshOutput(i) == 1
        classifyPositiveSampleNum = classifyPositiveSampleNum + 1;
        classifyPositiveMask(featureTable(i,2), featureTable(i,3)) = 1;
        classifyPositiveVideo(featureTable(i,2), featureTable(i,3), :) ...
            = inputVideo(featureTable(i,2), featureTable(i,3), :);
    elseif classifyThreshOutput(i) == 0
        classifyNegativeSampleNum = classifyNegativeSampleNum + 1;
    end

    if classifyTarget(i) == classifyThreshOutput(i)
        if classifyThreshOutput(i) == 1
            classifySuccessfulPositiveSampleNum = ...
                classifySuccessfulPositiveSampleNum + 1;
        elseif classifyThreshOutput(i) == 0
            classifySuccessfulNegativeSampleNum = ...
                classifySuccessfulNegativeSampleNum + 1;
        end
    elseif classifyTarget(i) ~= classifyThreshOutput(i)
        if classifyThreshOutput(i) == 1
            classifyFalsePositiveSampleNum = ...
                classifyFalsePositiveSampleNum + 1;
        else
            classifyFalseNegativeSampleNum = ...
                classifyFalseNegativeSampleNum + 1;
        end
    end
end
end

classifyPositiveFrames(n) = classifyPositiveSampleNum;
classifyNegativeFrames(n) = classifyNegativeSampleNum;
classifySuccessfulPositiveFrames(n) = ...
    classifySuccessfulPositiveSampleNum;
classifySuccessfulNegativeFrames(n) = ...
    classifySuccessfulNegativeSampleNum;

```

```

classifyFalsePositiveFrames(n) = classifyFalsePositiveSampleNum;
classifyFalseNegativeFrames(n) = classifyFalseNegativeSampleNum;

compareIndex = find(classifyPositiveMask);
compareMask = uint16(featureMask);
compareMask(compareIndex) = 2;
compareFrame = setHist(compareMask,[0 2],[0 intensityLim]);
compareFrameEnhanced = setHist(compareMask,[0 2],[0 enhancedLim]);

if plot_enable || (alarm_enable && classifyPositiveSampleNum > 0)
    if alarm_enable && classifyPositiveSampleNum > 0
        beep;
    end
    figure(28)
    imshowpair(featureVideo(:,:,viewFrame),...
        classifyPositiveVideo(:,:,viewFrame));
    title([filename, ' ', inputSource,...
        ' Feature Frame vs. Classified Positive Frame',...
        num2str(currentStart+viewFrame-1)])
    tightfig;

    figure(29)
    imshow(classifyPositiveMask);
    title([filename, ' ', inputSource,...
        ' Samples Classified as Positive for Frames ',...
        num2str(currentStart), ' to ', num2str(currentEnd)])
    tightfig;

    figure(30)
    imshowpair(compareVideo(:,:,viewFrame),classifyPositiveMask);
    title([filename, ' ', inputSource,...
        ' Classified Positive Frame vs. ', compareSource, ' Frame ',...
        num2str(currentStart+viewFrame-1)])
    tightfig;
end
if plot_enable
    for i=1:frameLength
        [inputVideovsEnhancedCL, inputVideovsEnhancedCL_zoom,...
            figHandle_inputVideovsEnhancedCL] = ...
            compareVideos(inputVideo(:,:,i),...
                hotColorMap,['Enhanced ', filename, ' ',inputSource,...
                    ' Video'],compareVideo(:,:,i),...
                coolColorMap,['Enhanced ', filename, ' ', compareSource,...
                    ' Video'],...
                [0 enhancedLim; 0 enhancedLim], targetRows, targetCols,...
                currentStart+i-1, 31);
        if write_enable
            if n == 1 && i == 1
                [writerObj_inputVideovsEnhancedCL] = objectFigMPEG4(...
                    [filename_output, '_enhanced_',inputSource,...
                        '_vs_', compareSource], 30);
            end
            if n ~= iterations || (n == iterations && i < frameLength)
                writeFigMPEG4(writerObj_inputVideovsEnhancedCL,...
                    figHandle_inputVideovsEnhancedCL, 0)
            end
            if n == iterations && i == frameLength

```

```

        writeFigMPEG4(writerObj_inputVideovsEnhancedCL,...
            figHandle_inputVideovsEnhancedCL, 1)
    end
end

[inputVideovsCL, inputVideovsCL_zoom,...
    figHandle_inputVideovsCL] = ...
compareVideos(inputVideo(:, :, i), ...
    hotColorMap, [filename, ' ', inputSource, ' Video'], ...
    compareVideo(:, :, i), ...
    coolColorMap, [filename, ' ', compareSource, ' Video'], ...
    [0 intensityLim; 0 intensityLim], targetRows, ...
    targetCols, currentStart+i-1, 34);
if write_enable
    if n == 1 && i == 1
        [writerObj_inputVideovsCL] = objectFigMPEG4(...
            [filename_output, ' ', inputSource, '_vs_', ...
            compareSource], 30);
    end
    if n ~= iterations || (n == iterations && i < frameLength)
        writeFigMPEG4(writerObj_inputVideovsCL,...
            figHandle_inputVideovsCL, 0)
    end
    if n == iterations && i == frameLength
        writeFigMPEG4(writerObj_inputVideovsCL,...
            figHandle_inputVideovsCL, 1)
    end
end

[featureVideovsEnhancedInput,...
    featureVideovsEnhancedInput_zoom,...
    figHandle_featureVideovsEnhancedInput] = ...
compareVideos(featureVideo(:, :, i), ...
    hotColorMap, ['Enhanced ', filename, ...
    ' Features of ', inputSource, ' Video'], inputVideo(:, :, i), ...
    coolColorMap, ['Enhanced ', filename, ' ', ...
    inputSource, ' Video'], ...
    [0 enhancedLim; 0 enhancedLim], targetRows, targetCols, ...
    currentStart+i-1, 37);
if write_enable
    if n == 1 && i == 1
        [writerObj_featureVideovsEnhancedInput] = ...
            objectFigMPEG4([filename_output, ...
            '_enhanced_feature_', inputSource], 30);
    end
    if n ~= iterations || (n == iterations && i < frameLength)
        writeFigMPEG4(writerObj_featureVideovsEnhancedInput,...
            figHandle_featureVideovsEnhancedInput, 0)
    end
    if n == iterations && i == frameLength
        writeFigMPEG4(writerObj_featureVideovsEnhancedInput,...
            figHandle_featureVideovsEnhancedInput, 1)
    end
end

[featureVideovsInput, featureVideovsInput_zoom,...
    figHandle_featureVideovsInput] = ...

```

```

        compareVideos(featureVideo(:,:,i),...
        hotColorMap,[filename, ' Features of ',...
        inputSource, ' Video'],inputVideo(:,:,i),...
        coolColorMap,[filename, ' ', inputSource, ' Video'],...
        [0 intensityLim; 0 intensityLim],...
        targetRows, targetCols, currentStart+i-1, 40);
    if write_enable
        if n == 1 && i == 1
            [writerObj_featureVideovsInput] = ...
                objectFigMPEG4([filename_output, '_feature_',...
                inputSource], 30);
        end
        if n ~= iterations || (n == iterations && i < frameLength)
            writeFigMPEG4(writerObj_featureVideovsInput,...
                figHandle_featureVideovsInput, 0)
        end
        if n == iterations && i == frameLength
            writeFigMPEG4(writerObj_featureVideovsInput,...
                figHandle_featureVideovsInput, 1)
        end
    end
end

classifyInput_1 = featureTable(:,4);
classifyInput_2 = featureTable(:,5);
classifyInput_3 = featureTable(:,6);

positiveKnown_feature1 = classifyInput_1(classifyTarget == 1);
negativeKnown_feature1 = classifyInput_1(classifyTarget == 0);

positiveKnown_feature2 = classifyInput_2(classifyTarget == 1);
negativeKnown_feature2 = classifyInput_2(classifyTarget == 0);

positiveKnown_feature3 = classifyInput_3(classifyTarget == 1);
negativeKnown_feature3 = classifyInput_3(classifyTarget == 0);

positiveClassify_feature1 = ...
    classifyInput_1(classifyThreshOutput == 1);
negativeClassify_feature1 = ...
    classifyInput_1(classifyThreshOutput == 0);

positiveClassify_feature2 = ...
    classifyInput_2(classifyThreshOutput == 1);
negativeClassify_feature2 = ...
    classifyInput_2(classifyThreshOutput == 0);

positiveClassify_feature3 = ...
    classifyInput_3(classifyThreshOutput == 1);
negativeClassify_feature3 = ...
    classifyInput_3(classifyThreshOutput == 0);

figHandle_classifyFeatureSpace=figure(43);
set(figHandle_classifyFeatureSpace,...
    'units','normalized','outerposition',[0 0 1 1])

a = subplot(2,4,1:2);
hold on

```

```

h1 = scatter3(...
    positiveKnown_feature1,positiveKnown_feature2,...
    positiveKnown_feature3,'+', 'red');
h2 = scatter3(...
    positiveClassify_feature1,positiveClassify_feature2,...
    positiveClassify_feature3,'o', 'red');
h3 = scatter3(...
    negativeKnown_feature1,negativeKnown_feature2,...
    negativeKnown_feature3,'.', 'blue');
h4 = scatter3(...
    negativeClassify_feature1,negativeClassify_feature2,...
    negativeClassify_feature3,'s', 'blue');
hold off
grid on
xlabel(feature1Title)
ylabel(feature2Title)
zlabel(feature3Title)
view(45+(n-1)*6,15)
xlim = get(gca, 'XLim');
ylim = get(gca, 'YLim');
zlim = get(gca, 'ZLim');
legend([h1 h2 h3 h4],{'Known Fire', 'Classified Fire',...
    'Known Noise', 'Classified Noise'}, 'Location',...
    'NorthEastOutside');
seconds = [(currentEnd-frameLength)/30, currentEnd/30];
title([filename, ' ',inputSource,...
    ' Target Sample Features vs. ',...
    'Classified Sample Features (after ',...
    num2str(currentEnd), ' Frames - ',num2str(seconds(2)), 's)'])

subplot(2,4,3:4)
cla;
hold on
h1 = scatter3(...
    positiveKnown_feature1,positiveKnown_feature2,...
    positiveKnown_feature3,'+', 'red');
h2 = scatter3(...
    positiveClassify_feature1,positiveClassify_feature2,...
    positiveClassify_feature3,'o', 'red');
h3 = scatter3(...
    negativeKnown_feature1,negativeKnown_feature2,...
    negativeKnown_feature3,'.', 'blue');
h4 = scatter3(...
    negativeClassify_feature1,negativeClassify_feature2,...
    negativeClassify_feature3,'s', 'blue');
hold off
grid on
xlabel(feature1Title)
ylabel(feature2Title)
zlabel(feature3Title)
view(45,15)
set(gca, 'XLim', xlim, 'YLim', ylim, 'ZLim', zlim);
legend([h1 h2 h3 h4],{'Known Fire', 'Classified Fire',...
    'Known Noise', 'Classified Noise'}, 'Location',...
    'NorthEastOutside');
seconds = [(currentEnd-frameLength)/30, currentEnd/30];
title([filename, ' ',inputSource, ' Target Sample Features vs. ',...

```

```

'Classified Sample Features (' , num2str(currentStart), ':' , ...
num2str(currentEnd), ' Frames - ' , num2str(seconds(1)) , ':' , ...
num2str(seconds(2)) , 's' '])

for i = 1:frameLength
    [classifyImageEnhanced, classifyImageEnhancedZoom, ...
     figHandle_classifyImageEnhanced] = ...
     compareVideos(compareFrameEnhanced, hotColorMap, ...
     ['Enhanced ' , filename, ' Classified ' , inputSource, ...
     ' Samples'], ...
     compareVideo(:, :, i), coolColorMap, ...
     ['Enhanced ' , filename, ' ' , compareSource, ' Video'], ...
     [0 enhancedLim; 0 enhancedLim], targetRows, targetCols, ...
     currentStart+i-1, figHandle_classifyFeatureSpace+1);
    if write_enable
        if n == 1 && i == 1
            [writerObj_classifyImageEnhanced] = objectFigMPEG4(...
            [filename_output, '_enhanced_classified_', ...
            inputSource, '_vs_', compareSource], 30);
        end
        if n ~= iterations || (n == iterations && i < frameLength)
            writeFigMPEG4(writerObj_classifyImageEnhanced, ...
            figHandle_classifyImageEnhanced, 0)
        end
        if n == iterations && i == frameLength
            writeFigMPEG4(writerObj_classifyImageEnhanced, ...
            figHandle_classifyImageEnhanced, 1)
        end
    end
end

[classifyImage, classifyImageZoom, ...
 figHandle_classifyImage] = compareVideos(compareFrame, ...
 hotColorMap, [filename, ' Classified ' , inputSource, ...
 ' Samples'], ...
 compareVideo(:, :, i), ...
 coolColorMap, [filename, ' ' , compareSource, ' Video'], ...
 [0 intensityLim; 0 intensityLim], targetRows, targetCols, ...
 currentStart+i-1, figHandle_classifyFeatureSpace+4);
if write_enable
    if n == 1 && i == 1
        [writerObj_classifyImage] = objectFigMPEG4(...
        [filename_output, '_classified_', inputSource, ...
        '_vs_', compareSource], 30);
    end
    if n ~= iterations || (n == iterations && i < frameLength)
        writeFigMPEG4(writerObj_classifyImage, ...
        figHandle_classifyImage, 0)
    end
    if n == iterations && i == frameLength
        writeFigMPEG4(writerObj_classifyImage, ...
        figHandle_classifyImage, 1)
    end
end
end

figure(figHandle_classifyFeatureSpace)
subtightplot(2,4,5)
imshow(classifyImage)

```

```

xlabel([filename, ' Classified ', inputSource, ' Samples'];...
      ['vs. ',filename, ' ', compareSource, ' Video']]);
subtightplot(2,4,6)
imshow(classifyImageZoom)
xlabel(['Zoomed-in Region of ', filename, ' Classified ',...
      inputSource, ' Samples'];['vs. ',filename,...
      ' ', compareSource, ' Video']]);
subtightplot(2,4,7)
imshow(classifyImageEnhanced)
xlabel(['Enhanced ', filename, ' Classified ', inputSource,...
      ' Samples'];['vs. Enhanced ', filename,...
      ' ', compareSource, ' Video']]);
subtightplot(2,4,8)
imshow(classifyImageEnhancedZoom)
xlabel(['Zoomed-in Region of Enhanced ',...
      filename, ' Classified ',inputSource, ' Samples'];...
      ['vs. Enhanced ',filename, ' ',compareSource, ' Video']]);

if write_enable
    if n == 1 && i == 1
        [writerObj_classifiedFeatures] = objectFigMPEG4(...
            strcat(filename_output,inputSource,...
                '_classifyFeatureSpace'), 30);
    end
    if n ~= iterations || (n == iterations && i < frameLength)
        writeFigMPEG4(writerObj_classifiedFeatures,...
            figHandle_classifyFeatureSpace, 0)
    end
    if n == iterations && i == frameLength
        writeFigMPEG4(writerObj_classifiedFeatures,...
            figHandle_classifyFeatureSpace, 1)
    end
end
end
if write_enable
    if n == iterations
        view(a,45,15)
        [classifyFeatureSpaceFrame,~] =getframe(...
            figHandle_classifyFeatureSpace);
        imwrite(classifyFeatureSpaceFrame,...
            [filename_output,inputSource,...
                '_classifyFeatureSpaceFrame.tif'],...
            'tif','Compression','none')
    end
end
end

if write_enable
    for i=1:size(featureTable,1)
        if classifyIndex <= classifyLengthLim
            classifyMatrix(:,classifyIndex) = featureTable(i,2:9)';
        else
            breakFlag = 1;
            disp('Classification of Data Complete!')
            break;
        end
        classifyIndex = classifyIndex + 1;
    end
end

```

```

        end
        if breakFlag
            break;
        end
    end
end

if plot_enable
    f = figure(50);
    set(f, 'units', 'normalized', 'outerposition', [0 0 1 1])
    subplot(2,1,1)
    title([filename, ' ', inputSource, ' Classification Over Time'])
    hold on
    xSeconds = (linspace(frameLength,...
        frameLength*iterations,iterations))./frameLength;
    p1 = plot(xSeconds,classifyPositiveFrames,...
        'LineStyle','-', 'Color','red', 'LineWidth',2);
    p2 = plot(xSeconds,classifyNegativeFrames,...
        'LineStyle','--', 'Color','blue', 'MarkerFaceColor',...
        'blue', 'LineWidth',2);
    hold off
    xlabel('Seconds')
    ylabel('Samples Count')
    legend([p1 p2],...
        {'Fire', 'Noise'});
    subplot(2,1,2)
    hold on
    p3 = plot(xSeconds,classifyFalsePositiveFrames,...
        'LineStyle','-.', 'Color','red', 'MarkerFaceColor',...
        'red', 'LineWidth',2);
    p4 = plot(xSeconds,classifyFalseNegativeFrames,...
        'LineStyle',':', 'Color','blue', 'MarkerFaceColor',...
        'blue', 'LineWidth',2);
    hold off
    xlabel('Seconds')
    ylabel('Samples Count')
    legend([p3 p4],...
        {'False-Positive',...
        'False-Negative'});

    if write_enable
        [classifyStatsFrame,~] = getframe(f);
        imwrite(classifyStatsFrame,[filename_output,inputSource,...
            '_classifyStatsFrame.tif'],'tif','Compression','none')
    end
end

classifyPositiveSampleNum = sum(classifyPositiveFrames);
classifyNegativeSampleNum = sum(classifyNegativeFrames);
classifySuccessfulPositiveSampleNum = sum(...
    classifySuccessfulPositiveFrames);
classifySuccessfulNegativeSampleNum = sum(...
    classifySuccessfulNegativeFrames);
classifyFalsePositiveSampleNum = sum(classifyFalsePositiveFrames);
classifyFalseNegativeSampleNum = sum(classifyFalseNegativeFrames);
totalSampleNum = sum(classifyPositiveSampleNum) + sum(...
    classifyNegativeSampleNum);

```



```

falsePositiveError = classifyFalsePositiveSampleNum/totalSampleNum;
falseNegativeError = classifyFalseNegativeSampleNum/totalSampleNum;
totalClassifyErrors = (classifyFalsePositiveSampleNum + ...
    classifyFalseNegativeSampleNum);
totalError = totalClassifyErrors/totalSampleNum;

disp(['Samples Classified: ', num2str(totalSampleNum)])
disp(['Samples Classified as Fire: ', num2str(classifyPositiveSampleNum)])
disp(['Samples Classified as Noise: ', num2str(classifyNegativeSampleNum)])
disp(['Samples Successfully Classified as Fire: ', ...
    num2str(classifySuccessfulPositiveSampleNum)])
disp(['Samples Successfully Classified as Noise: ', ...
    num2str(classifySuccessfulNegativeSampleNum)])
disp(['False Positives: ', num2str(classifyFalsePositiveSampleNum)])
disp(['False Positive Error: ', num2str(falsePositiveError)])
disp(['False Negatives: ', num2str(classifyFalseNegativeSampleNum)])
disp(['False Negative Error: ', num2str(falseNegativeError)])
disp(['Total Classification Errors: ', num2str(totalClassifyErrors)])
disp(['Total Sample Classification Error: ', num2str(totalError)])

classifyStats.filename = filename;
classifyStats.inputSource = inputSource;
classifyStats.startFrame = startFrame;
classifyStats.endFrame = endFrame;
classifyStats.frameLength = frameLength;
classifyStats.registered_video = registered_video;
classifyStats.filename_output = filename_output;
classifyStats.filtSqrSize = filtSqrSize;
classifyStats.unsharpStrength = unsharpStrength;
classifyStats.gausSigma = gausSigma;
classifyStats.maxCoefVariationLim = maxCoefVariationLim;
classifyStats.maxNeighborDist = maxNeighborDist;
classifyStats.maxCorrCoefLim = maxCorrCoefLim;
classifyStats.maxCorrLagLim = maxCorrLagLim;
classifyStats.distNH = distNH;
classifyStats.intensityLim = intensityLim;
classifyStats.targetUpperLeftCorner = targetUpperLeftCorner;
classifyStats.targetBottomRightCorner = targetBottomRightCorner;
classifyStats.totalSampleNum = totalSampleNum;
classifyStats.classifyPositiveSampleNum = classifyPositiveSampleNum;
classifyStats.classifyNegativeSampleNum = classifyNegativeSampleNum;
classifyStats.classifySuccessfulPositiveSampleNum = ...
    classifySuccessfulPositiveSampleNum;
classifyStats.classifySuccessfulNegativeSampleNum = ...
    classifySuccessfulNegativeSampleNum;
classifyStats.classifyFalsePositiveSampleNum = ...
    classifyFalsePositiveSampleNum;
classifyStats.falsePositiveError = falsePositiveError;
classifyStats.classifyFalseNegativeSampleNum = ...
    classifyFalseNegativeSampleNum;
classifyStats.falseNegativeError = falseNegativeError;
classifyStats.totalClassifyErrors = totalClassifyErrors;
classifyStats.totalError = totalError;
classifyStats.classifyPositiveFrames = classifyPositiveFrames;
classifyStats.classifyNegativeFrames = classifyNegativeFrames;
classifyStats.classifySuccessfulPositiveFrames = ...

```

```

        classifySuccessfulPositiveFrames;
classifyStats.classifySuccessfulNegativeFrames = ...
        classifySuccessfulNegativeFrames;
classifyStats.classifyFalsePositiveFrames = classifyFalsePositiveFrames;
classifyStats.classifyFalseNegativeFrames = classifyFalseNegativeFrames;

if write_enable
    save([filename_output,inputSource,'_classifyStats'],'-struct',...
        'classifyStats','-v7.3');

    classifyNum = size(classifyMatrix,1);
    classifyMatrix = classifyMatrix(isfinite(classifyMatrix));
    classifyMatrix = reshape(classifyMatrix, classifyNum,...
        length(classifyMatrix)/classifyNum);
    save([filename_output,inputSource,'_classifyMatrix'],...
        'classifyMatrix','-v7.3');
end

toc;
%% End of Script -----%
```

```

function [ outputVideo ] = applyDiagOffset( inputVideo, diagOffset,...
    currentStart )
%% Matlab Function -----%
% Function : applyDiagOffset.m
% Author : Raul-Alexander Aldama
% Date : 3/27/2013
%
% Description : Masks border edges in input masking process
%
% Notes :
% Revision(s) :
%-----%
vidDim = size(inputVideo);
frameLength = vidDim(3);

outputVideo = zeros(vidDim,'uint16');

for i=1:frameLength
    inputFrame = inputVideo(:,:,i);
    offset = diagOffset(currentStart+i-1);
    row = 1+offset:vidDim(1)-offset;
    col = 1+offset:vidDim(2)-offset;
    outputFrame = zeros(vidDim(1), vidDim(2));
    outputFrame(row,col) = inputFrame(row,col);
    outputVideo(:,:,i) = outputFrame;
end
%% End of Function -----%

```

```

function [ featureRow, featureCol, featureTarget, featureSamples,...
    numFeatureSamples, coefVariationMask, coefVariationMaskNeighbors,...
    corrMask, corrMaskNeighbors ] = applyFeatureMask(temporalSamples, ...
    maxCoefVariationLim, maxNeighborDist, maxCorrCoefLim, maxCorrLagLim,...
    inputRow, inputCol, trainTarget, plot_enable, inputMask)
%% Matlab Function -----%
% Function : applyFeatureMask.m
% Author : Raul-Alexander Aldama
% Date : 3/28/2013
%
% Description : Applies feature mask
%
% Notes :
% Revision(s) :
%-----%
featureRow = inputRow;
featureCol = inputCol;
featureTarget = trainTarget;
featureSamples = temporalSamples;

coefVariation = (std(temporalSamples)./mean(temporalSamples));
keepSampleMarker = coefVariation < maxCoefVariationLim & coefVariation > 0;
featureRow = featureRow(keepSampleMarker');
featureCol = featureCol(keepSampleMarker');
featureTarget = featureTarget(keepSampleMarker');
featureSamples = featureSamples(:,keepSampleMarker);
numFeatureSamples = size(featureSamples,2);
keepSampleMarker = keepSampleMarker(keepSampleMarker);

if plot_enable
    coefVariationMask = false(size(inputMask));
    for i=1:numFeatureSamples
        coefVariationMask(featureRow(i),featureCol(i)) = 1;
    end
else
    coefVariationMask = inputMask;
end

keepSampleMarker = false(size(keepSampleMarker));
for i=1:numFeatureSamples
    refPoint = [featureRow(i), featureCol(i)];
    for j=1:numFeatureSamples
        if j > i
            point = [featureRow(j), featureCol(j)];
            if abs(point(1)-refPoint(1)) <= maxNeighborDist && abs(point(2)-
refPoint(2)) <= maxNeighborDist
                keepSampleMarker(i) = 1;
                keepSampleMarker(j) = 1;
                break;
            end
        end
    end
end
featureRow = featureRow(keepSampleMarker');
featureCol = featureCol(keepSampleMarker');
featureTarget = featureTarget(keepSampleMarker');

```

```

featureSamples = featureSamples(:,keepSampleMarker);
numFeatureSamples = size(featureSamples,2);
keepSampleMarker = keepSampleMarker(keepSampleMarker);

if plot_enable
    coefVariationMaskNeighbors = false(size(inputMask));
    for i=1:numFeatureSamples
        coefVariationMaskNeighbors(featureRow(i),featureCol(i)) = 1;
    end
else
    coefVariationMaskNeighbors = inputMask;
end

keepSampleMarker = false(size(keepSampleMarker));
for i=1:numFeatureSamples
    refPoint = [featureRow(i), featureCol(i)];
    for j=1:numFeatureSamples
        if j > i
            point = [featureRow(j), featureCol(j)];
            if abs(point(1)-refPoint(1)) <= maxNeighborDist && abs(point(2)-
refPoint(2)) <= maxNeighborDist
                [pixelCorr, pixelLag] =
xcorr(featureSamples(:,i),featureSamples(:,j),'coef');
                pixelLag = pixelLag';
                [maxCorr, maxCorrIndex] = max(pixelCorr);
                maxCorrInst = pixelLag(maxCorrIndex);
                if maxCorr <= maxCorrCoefLim && abs(maxCorrInst) <=
maxCorrLagLim
                    keepSampleMarker(i) = 1;
                    break;
                end
            end
        end
    end
end
featureRow = featureRow(keepSampleMarker');
featureCol = featureCol(keepSampleMarker');
featureTarget = featureTarget(keepSampleMarker');
featureSamples = featureSamples(:,keepSampleMarker);
numFeatureSamples = size(featureSamples,2);
keepSampleMarker = keepSampleMarker(keepSampleMarker);

if plot_enable
    corrMask = false(size(inputMask));
    for i=1:numFeatureSamples
        corrMask(featureRow(i),featureCol(i)) = 1;
    end
else
    corrMask = inputMask;
end

keepSampleMarker = false(size(keepSampleMarker));
for i=1:numFeatureSamples
    refPoint = [featureRow(i), featureCol(i)];
    for j=1:numFeatureSamples
        if j > i

```

```

        point = [featureRow(j), featureCol(j)];
        if abs(point(1)-refPoint(1)) <= maxNeighborDist && abs(point(2)-
refPoint(2)) <= maxNeighborDist
            keepSampleMarker(i) = 1;
            keepSampleMarker(j) = 1;
            break;
        end
    end
end
end
featureRow = featureRow(keepSampleMarker');
featureCol = featureCol(keepSampleMarker');
featureTarget = featureTarget(keepSampleMarker');
featureSamples = featureSamples(:,keepSampleMarker);
numFeatureSamples = size(featureSamples,2);
keepSampleMarker = keepSampleMarker(keepSampleMarker);

if plot_enable
    corrMaskNeighbors = false(size(inputMask));
    for i=1:numFeatureSamples
        corrMaskNeighbors(featureRow(i),featureCol(i)) = 1;
    end
else
    corrMaskNeighbors = inputMask;
end
%% End of Function -----%
```

```

function [ outputVideo ] = applyThreshVideo( inputVideo, minLevel,...
    maxLevel )
%% Matlab Function -----%
% Function : applyThreshVideo.m
% Author : Raul-Alexander Aldama
% Date : 2/24/2013
%
% Description : Applies a minimum and maximum intensity level threshold to
% video frames
%
% Notes :
% Revision(s) :
%-----%
outputVideo = inputVideo;
index = (inputVideo<minLevel);
outputVideo(index) = 0;
index = (inputVideo>maxLevel);
outputVideo(index) = 0;
%% End of Function -----%

```

```

function [ outputVideo ] = awgnVideo( inputVideo, noisePerc, mu, sigma,...
    minVal, maxVal, s1, s2, s3)
%% Matlab Function -----%
% Function : awgnVideo.m
% Author : Raul-Alexander Aldama
% Date : 3/25/2013
%
% Description : Adds additive white gaussian noise to video
%
% Notes : Experimental
% Revision(s) :
%-----%
assert(mu>=minVal && mu<=maxVal);
assert(sigma>0);

rows = size(inputVideo,1);
cols = size(inputVideo,2);
frameLength = size(inputVideo,3);

outputVideo = zeros(size(inputVideo),'uint16');
for i=1:frameLength
    image = inputVideo(:,:,i);

    row = randi(s1,rows,1,round(rows*cols*noisePerc));
    col = randi(s2,cols,1,round(rows*cols*noisePerc));
    noise = mu + sigma*randn(s3,1,round(rows*cols*noisePerc));
    outsideRange = noise<minVal | noise>maxVal;
    while sum(outsideRange)>0
        noise(outsideRange) = mu + sigma*randn(sum(outsideRange),1);
        outsideRange = noise<minVal | noise>maxVal;
    end

    for j=1:length(noise)
        oldVal = image(row(j),col(j));
        newVal = oldVal+noise(j);
        image(row(j),col(j)) = newVal;
    end

    outputVideo(:,:,i) = image;
end
%% End of Function -----%

```



```

function [image, imageZoom, figureHandle ] = compareVideos(hotFrame,...
    hotColorMap, hotTitle, coolFrame, coolColorMap, coolTitle,...
    CLim, zoomRows, zoomCols, currentStart, figHandleStart)
%% Matlab Function -----%
% Function : compareVideos.m
% Author : Raul-Alexander Aldama
% Date : 3/30/2013
%
% Description : Creates composite video to compare frames, adds scale and
% frame number to video
%
% Notes :
% Revision(s) :
%-----%
scaleLength = 20;

figure(figHandleStart)
imshow(hotFrame)
colormap(hotColorMap)
set(gca, 'CLim', CLim(1,:));
hotFrame_color = getframe;
hotFrame_colorZoom = hotFrame_color.cdata(zoomRows, zoomCols,:);

figure(figHandleStart+1)
imshow(coolFrame)
colormap(coolColorMap)
set(gca, 'CLim', CLim(2,:));
coolFrame_color = getframe;
coolFrame_colorZoom = coolFrame_color.cdata(zoomRows, zoomCols,:);

f=figure(figHandleStart+2);
figureHandle = f;
set(f, 'units', 'normalized', 'outerposition', [0 0 1 1])

subtightplot(2,2,1);
imshow(hotFrame_color.cdata);
xLim = get(gca, 'XLim');
hold on
line([xLim(2)-scaleLength xLim(2)-1],[1 1], 'Color', 'red', 'LineWidth', 2);
text(2,1,num2str(currentStart), 'Color', 'white',...
    'VerticalAlignment', 'Top', 'HorizontalAlignment', 'Left')
hold off
title(hotTitle);

subtightplot(2,2,2);
imshow(coolFrame_color.cdata);
xLim = get(gca, 'XLim');
hold on
line([xLim(2)-scaleLength xLim(2)-1],[1 1], 'Color', 'red', 'LineWidth', 2);
text(2,1,num2str(currentStart), 'Color', 'white',...
    'VerticalAlignment', 'Top', 'HorizontalAlignment', 'Left')
hold off
title(coolTitle);

subtightplot(2,2,3);
imshowpair(coolFrame_color.cdata, hotFrame_color.cdata, 'blend', ...

```

```

        'scaling','none')
xLim = get(gca, 'XLim');
hold on
line([xLim(2)-scaleLength xLim(2)-1],[1 1], 'Color','red', 'LineWidth',2);
text(2,1,num2str(currentStart), 'Color','white',...
    'VerticalAlignment','Top', 'HorizontalAlignment','Left')
hold off
xlabel({[hotTitle, ' vs. ', coolTitle]});
image = getframe;
image = image.cdata;

subtightplot(2,2,4);
imshowpair(coolFrame_colorZoom,hotFrame_colorZoom,'blend','scaling','none')
xLim = get(gca, 'XLim');
hold on
line([xLim(2)-scaleLength xLim(2)-1],[1 1], 'Color','red', 'LineWidth',2);
text(1,1,num2str(currentStart), 'Color','white',...
    'VerticalAlignment','Top', 'HorizontalAlignment','Left')
hold off
xlabel({'Zoomed-In Region of ', hotTitle, ' vs. ', coolTitle});
imageZoom = getframe;
imageZoom = imageZoom.cdata;
%% End of Function -----%

```

```

function [featureTable, feature1Title, feature2Title, feature3Title,...
    featureInputVideo, featureOutputVideo, featureMask] = ...
    extractFireFeatures( inputVideo, refVideo,...
        filename, inputSource, refSource, frameLength,...
        currentStart, currentEnd, intensityLim, bottomClip, diagOffset,...
        targetRows, targetCols, unsharpStrength, filtSqrSize, gaussSigma,...
        maxCoefVariationLim, maxNeighborDist, maxCorrCoefLim, maxCorrLagLim,...
        distNH, plot_enable, viewFrame, figHandleStart )
%% Matlab Function -----%
% Function : extractFireFeatures.m
% Author : Raul-Alexander Aldama
% Date : 3/11/2013
%
% Description : Extracts forest fire heat plume features and feature points
%
% Notes :
% Revision(s) :
%-----%
vidDim = [size(inputVideo,1) size(inputVideo,2)];

refVideo = medFilterVideo(refVideo, 5);

if plot_enable
    inputFrame = inputVideo(:,:,viewFrame);
    inputZero = (inputVideo == 0);
    inputZeroFrame = inputZero(:,:,viewFrame);
    inputFrameFFT = fft2(inputFrame);
    inputFrameFFT = fftshift(inputFrameFFT);
    inputFrameFFT = abs(inputFrameFFT);
    inputFrameFFT = uint16(20.*log(inputFrameFFT+1));

    refFrame = refVideo(:,:,viewFrame);

    inputFrame = setHist(inputFrame,minmax(inputFrame(:)'),...
        [0 intensityLim]);
    refFrame = setHist(refFrame,minmax(refFrame(:)'),[0 intensityLim]);

    figure(figHandleStart)
    imshowpair(refFrame,inputFrame,'Scaling','none')
    title([filename,' ',refSource,' Reference vs. ', inputSource,...
        ' Frame ',num2str(currentStart+viewFrame-1)])
    tightfig;
end

inputVideo = unsharpFilterVideo(inputVideo, unsharpStrength,...
    filtSqrSize, gaussSigma);

if plot_enable
    filtFrame = inputVideo(:,:,viewFrame);
    filtZero = (inputVideo == 0);
    filtZeroMask = zeros(vidDim);
    for i=1:frameLength
        filtZeroMask = filtZeroMask | inputVideo(:,:,i) > 0;
    end
    filtZeroFrame = filtZero(:,:,viewFrame);

```

```

filtFrameFFT = fft2(filtFrame);
filtFrameFFT = fftshift(filtFrameFFT);
filtFrameFFT = abs(filtFrameFFT);
filtFrameFFT = uint16(20.*log(filtFrameFFT+1));

filtFrame = setHist(filtFrame,minmax(filtFrame(:)'),[0 intensityLim]);

figure(figHandleStart+1)
subtightplot(1,2,1)
imshow(imcomplement(inputZeroFrame), 'border', 'tight')
title([filename, ' ',inputSource, ' Non-Zero Pixels of Frame ',...
    num2str(currentStart+viewFrame-1)])
subtightplot(1,2,2)
imshow(imcomplement(filtZeroFrame), 'border', 'tight')
title([filename, ' ',inputSource,...
    ' Non-Zero Pixels of Unsharp Filtered Frame ',...
    num2str(currentStart+viewFrame-1)])

figure(figHandleStart+2)
subtightplot(1,2,1)
imshow(inputFrameFFT,[], 'border', 'tight')
title([filename, ' ',inputSource, ' Frame FFT ',...
    num2str(currentStart+viewFrame-1)])
subtightplot(1,2,2)
imshow(filtFrameFFT,[], 'border', 'tight')
title([filename, ' ',inputSource, ' Unsharp Filtered Frame FFT ',...
    num2str(currentStart+viewFrame-1)])
end

refVideo = applyThreshVideo(refVideo,bottomClip,intensityLim);
zeroRef = (refVideo == 0);

inputVideo(zeroRef) = 0;

mask = zeros(vidDim);
inputMask = zeros(vidDim);
targetMask = zeros(vidDim);

for i=1:frameLength
    mask = mask | (inputVideo(:, :, i) > 0);
end

if plot_enable
    threshZeroInput = (inputVideo == 0);
    zeroRefMask = zeros(vidDim);
    for i=1:frameLength
        zeroRefMask = zeroRefMask | zeroRef(:, :, i) == 1;
    end
    zeroRefFrame = zeroRef(:, :, viewFrame);
    zeroThreshFrame = threshZeroInput(:, :, viewFrame);

    inputThreshFrame = inputVideo(:, :, viewFrame);
    inputThreshFrame = setHist(inputThreshFrame,...
        minmax(inputThreshFrame(:)'),[0 intensityLim]);

```

```

figure(figHandleStart+3)
imshow(filtZeroMask)
title([filename, ' ', inputSource, ...
    ' Non-Zero Pixels of Unsharp Filtered Frames ', ...
    num2str(currentStart), ' to ', num2str(currentEnd)])
tightfig;

topPerc = 100 - round(bottomClip/intensityLim*100);
figure(figHandleStart+4)
subplot(2,2,1)
imshow(refFrame, 'border', 'tight')
title([filename, ' ', refSource, ' Frame ', ...
    num2str(currentStart+viewFrame-1)])
subplot(2,2,3)
imshow(imcomplement(filtZeroFrame), 'border', 'tight')
xlabel([filename, ' ', inputSource, ...
    ' Non-Zero Pixels of Unsharp Filtered Frame ', ...
    num2str(currentStart+viewFrame-1)])
subplot(2,2,2)
imshow(imcomplement(zeroRefFrame), 'border', 'tight')
title([filename, ' ', refSource, ' Top ', num2str(topPerc), ...
    '% of Frame ', num2str(currentStart+viewFrame-1)])
subplot(2,2,4)
imshow(imcomplement(zeroThreshFrame), 'border', 'tight')
xlabel([filename, ' ', inputSource, ...
    ' Non-Zero Pixels of Unsharp Filtered Frame after ', ...
    refSource, ' Top ', num2str(topPerc), ...
    '% Retained from Frame ', num2str(currentStart+viewFrame-1)])

figure(figHandleStart+5)
imshowpair(filtZeroMask, imcomplement(zeroRefMask), 'Scaling', 'none')
title([filename, ' ', refSource, ' Top ', num2str(topPerc), '% vs. ', ...
    inputSource, ' Non-Zero Pixels of Unsharp Filtered Frames ', ...
    num2str(currentStart), ' to ', num2str(currentEnd)])
tightfig;

figure(figHandleStart+6)
imshowpair(filtFrame, inputThreshFrame, 'Scaling', 'none')
title([filename, ' ', inputSource, ...
    ' Unsharp Filtered Frame vs. Unsharp Filtered Frame after ', ...
    refSource, ' Top ', num2str(topPerc), '% Retained from Frame ', ...
    num2str(currentStart+viewFrame-1)])
tightfig;
end

inputRows = 1+max(diagOffset):vidDim(1)-max(diagOffset);
minMaxInputRows = minmax(inputRows);
inputCols = 1+max(diagOffset):vidDim(2)-max(diagOffset);
minMaxInputCols = minmax(inputCols);
inputMask(inputRows,inputCols) = mask(inputRows,inputCols);

inputMaskIndecies = logical(repmat(inputMask,[1 1 frameLength]));
featureInputVideo = zeros(size(inputVideo), 'uint16');
featureInputVideo(inputMaskIndecies) = inputVideo(inputMaskIndecies);

```

```

[inputRow, inputCol] = find(inputMask);
numSamples = length(inputCol);
temporalSamples = zeros(frameLength,numSamples);
trainTarget = zeros(numSamples, 1);

minmaxTargetRows = minmax(targetRows);
minmaxTargetCols = minmax(targetCols);

if minmaxTargetRows(1) >= minMaxInputRows(1) && ...
    minmaxTargetRows(2) <= minMaxInputRows(2) ...
    && minmaxTargetCols(1) >= minMaxInputCols(1) && ...
    minmaxTargetCols(2) <= minMaxInputCols(2)
    targetMask(targetRows,targetCols) = inputMask(targetRows,targetCols);
else
    targetMask = inputMask;
end

for i=1:numSamples
    sample = featureInputVideo(inputRow(i),inputCol(i),:);
    temporalSamples(:,i) = permute(sample, [3 2 1]);
    trainTarget(i) = targetMask(inputRow(i),inputCol(i));
end

if plot_enable
    afterInputMask = (featureInputVideo == 0);
    afterInputMaskFrame = afterInputMask(:,:,viewFrame);
    targetRegionMask = zeros(size(targetMask),'uint16');
    targetRegionMask(targetRows,targetCols) = intensityLim;

    figure(figHandleStart+7)
    imshowpair(filtZeroMask,inputMask,'Scaling', 'independent')
    title([filename, ' ',inputSource,...
        ' Non-zero Pixels Before Input Mask vs. ',...
        'Non-zero Pixels After Input Mask for Frames ',...
        num2str(currentStart), ' to ',num2str(currentEnd)])
    tightfig;

    figure(figHandleStart+8)
    imshowpair(imcomplement(filtZeroFrame),...
        imcomplement(afterInputMaskFrame),'Scaling', 'independent')
    title([filename, ' ',inputSource,...
        ' Non-Zero Pixels of Unsharp Filtered Frame vs. ',...
        'After Input Mask ',num2str(currentStart+viewFrame-1)])
    tightfig;

    figure(figHandleStart+9)
    imshowpair(inputMask,targetMask,'Scaling', 'independent')
    title([filename, ' ',inputSource, ' Input Mask vs. ',...
        'Target Mask for Frames ',num2str(currentStart), ' to ',...
        num2str(currentEnd)])
    tightfig;

    figure(figHandleStart+10)
    imshowpair(refFrame, targetRegionMask,'Scaling', 'independent')

```

```

title([filename, ' ', refSource, ' Frame vs. Target Region for Frame ', ...
       num2str(currentStart+viewFrame-1)])
tightfig;

figure(figHandleStart+11)
imshowpair(imcomplement(afterInputMaskFrame), targetRegionMask, ...
           'Scaling', 'independent')
title([filename, ' ', inputSource, ' After Input Mask vs. ', ...
       'Target Region for Frame ', num2str(currentStart+viewFrame-1)])
tightfig;

figure(figHandleStart+12)
imshowpair(inputMask, targetRegionMask, 'Scaling', 'independent')
title([filename, ' ', inputSource, ' Input Mask vs. ', ...
       'Target Region for Frames ', num2str(currentStart), ' to ', ...
       num2str(currentEnd)])
tightfig;
end

if numSamples > 0
    [featureRow, featureCol, featureTarget, featureSamples, ...
     numFeatureSamples, coefVariationMask, ...
     coefVariationMaskNeighbors, corrMask, corrMaskNeighbors] = ...
        applyFeatureMask(temporalSamples, maxCoefVariationLim, ...
                          maxNeighborDist, maxCorrCoefLim, maxCorrLagLim, ...
                          inputRow, inputCol, trainTarget, plot_enable, inputMask);

    [featureTable, featureSamples, feature1Title, feature2Title, ...
     feature3Title] = generateFeaturePoints(featureRow, featureCol, ...
                                             featureTarget, featureSamples, numFeatureSamples, distNH);

    if numFeatureSamples > 0
        featureRows = featureTable(:,2);
        featureCols = featureTable(:,3);
        featureInput_1 = featureTable(:,4);
        featureInput_2 = featureTable(:,5);
        featureInput_3 = featureTable(:,6);

        positive_feature1 = featureInput_1(featureTarget == 1);
        negative_feature1 = featureInput_1(featureTarget == 0);

        positive_feature2 = featureInput_2(featureTarget == 1);
        negative_feature2 = featureInput_2(featureTarget == 0);

        positive_feature3 = featureInput_3(featureTarget == 1);
        negative_feature3 = featureInput_3(featureTarget == 0);
    else
        coefVariationMask = zeros(size(inputMask));
        coefVariationMaskNeighbors = zeros(size(inputMask));
        corrMask = zeros(size(inputMask));
        corrMaskNeighbors = zeros(size(inputMask));

        featureRows = [];
        featureCols = [];
    end
end

```

```

        positive_feature1 = [];
        negative_feature1 = [];

        positive_feature2 = [];
        negative_feature2 = [];

        positive_feature3 = [];
        negative_feature3 = [];
    end
else
    coefVariationMask = zeros(size(inputMask));
    coefVariationMaskNeighbors = zeros(size(inputMask));
    corrMask = zeros(size(inputMask));
    corrMaskNeighbors = zeros(size(inputMask));
    featureRow = inputRow;
    featureCol = inputCol;
    featureTarget = trainTarget;
    featureSamples = temporalSamples;
    numFeatureSamples = numSamples;

    [featureTable, featureSamples, feature1Title, feature2Title,...
     feature3Title ] = generateFeaturePoints(featureRow, featureCol,...
     featureTarget, featureSamples, numFeatureSamples, distNH );

    positive_feature1 = [];
    negative_feature1 = [];

    positive_feature2 = [];
    negative_feature2 = [];

    positive_feature3 = [];
    negative_feature3 = [];
end
featureMask = corrMaskNeighbors;
featureMaskIndecies = logical(repmat(featureMask,[1 1 frameLength]));
featureOutputVideo = zeros(size(featureInputVideo),'uint16');
featureOutputVideo(featureMaskIndecies) = ...
    featureInputVideo(featureMaskIndecies);

featurePixelsPositive = zeros(vidDim,'uint16');
for i=1:numFeatureSamples
    if featureTarget(i) == 1
        featurePixelsPositive(featureRows(i),featureCols(i))= 1;
    end
end
end

if plot_enable
    figure(figHandleStart+13)
    imshowpair(inputMask, coefVariationMask,'Scaling','independent')
    title([filename, ' ',inputSource, ' Input Mask vs. ',...
        'Coefficient of Variation Mask for Frames ',...
        num2str(currentStart),' to ',num2str(currentEnd)])
    tightfig;

    figure(figHandleStart+14)

```



```

imshowpair(coefVariationMask, coefVariationMaskNeighbors,...
    'Scaling','independent')
title([filename, ' ',inputSource, ' Coefficient of Variation Mask ',...
    num2str(2*maxNeighborDist+1),'-by-',num2str(2*maxNeighborDist+1),...
    ' Neighbors for Frames ',num2str(currentStart),' to ',...
    num2str(currentEnd)])
tightfig;

figure(figHandleStart+15)
imshowpair(coefVariationMaskNeighbors, corrMask, 'Scaling','independent')
title([filename, ' ',inputSource, ' Coefficient of Variation Mask ',...
    num2str(2*maxNeighborDist+1),'-by-',num2str(2*maxNeighborDist+1),...
    ' Neighbors vs. Cross Correlation Mask for Frames ',...
    num2str(currentStart),' to ',num2str(currentEnd)])
tightfig;

figure(figHandleStart+16)
imshowpair(corrMask, featureMask, 'Scaling','independent')
title([filename, ' ',inputSource, ' Cross Correlation Mask ',...
    num2str(2*maxNeighborDist+1),'-by-',num2str(2*maxNeighborDist+1),...
    ' Neighbors for Frames ',num2str(currentStart),' to ',...
    num2str(currentEnd)])
tightfig;

figure(figHandleStart+17)
imshowpair(inputMask, featureMask, 'Scaling','independent')
title([filename, ' ',inputSource,...
    ' Input Mask vs. Feature Mask for Frames ',num2str(currentStart),...
    ' to ',num2str(currentEnd)])
tightfig;

afterFeatureMaskFrame = featureOutputVideo(:,:,viewFrame);
afterFeatureMaskFrame = setHist(afterFeatureMaskFrame,...
    minmax(afterFeatureMaskFrame(:)'),[0 intensityLim]);
afterFeatureMasking = (featureOutputVideo == 0);
afterFeatureMask = afterFeatureMasking(:,:,viewFrame);
figure(figHandleStart+18)
imshowpair(imcomplement(afterInputMaskFrame),...
    imcomplement(afterFeatureMask), 'Scaling','independent')
title([filename, ' ',inputSource, ' After Input Mask vs. ',...
    'After Feature Mask for Frames ',num2str(currentStart),' to ',...
    num2str(currentEnd)])
tightfig;

figure(figHandleStart+19)
imshowpair(inputFrame, afterFeatureMaskFrame, 'Scaling','independent')
title([filename, ' ',inputSource, ' vs. Feature Frame ',...
    num2str(currentStart+viewFrame-1)])
tightfig;

figure(figHandleStart+20)
imshowpair(refFrame, afterFeatureMaskFrame, 'Scaling','independent')
title([filename, ' ',refSource, ' Frame vs. Feature Frame ',...
    num2str(currentStart+viewFrame-1)])
tightfig;

```

```

figure(figHandleStart+21)
imshowpair(inputFrame, featureMask, 'Scaling', 'independent')
title([filename, ' ', inputSource, ' Frame ', ...
       num2str(currentStart+viewFrame-1), ' vs. Feature Mask '])
tightfig;

figure(figHandleStart+22)
imshowpair(refFrame, featureMask, 'Scaling', 'independent')
title([filename, ' ', refSource, ' Frame ', ...
       num2str(currentStart+viewFrame-1), ' vs. Feature Mask '])
tightfig;

figure(figHandleStart+23)
imshowpair(featureMask, featurePixelsPositive, 'Scaling', 'independent')
title([filename, ' ', inputSource, ' Feature Mask vs. ', ...
       'Positive Targets for Frames ', num2str(currentStart), ' to ', ...
       num2str(currentEnd)])
tightfig;

figure(figHandleStart+24)
imshowpair(featureMask, targetRegionMask, 'Scaling', 'independent')
title([filename, ' ', inputSource, ' Feature Mask vs. ', ...
       'Target Region for Frames ', num2str(currentStart), ' to ', ...
       num2str(currentEnd)])
tightfig;

figure(figHandleStart+25)
imshowpair(refFrame, featurePixelsPositive, 'Scaling', 'independent')
title([filename, ' ', refSource, ' Frame vs. ', ...
       'Positive Targets for Frames ', num2str(currentStart), ' to ', ...
       num2str(currentEnd)])
tightfig;
end

if plot_enable
    f = figure(figHandleStart+26);
    set(f, 'units', 'normalized', 'outerposition', [0 0 1 1])
    subplot(4,3,1)
    cla;
    hold on
    for i=1:numSamples
        if trainTarget(i) == 1
            plot(1:frameLength, temporalSamples(:,i), 'Color', 'red');
        else
            plot(1:frameLength, temporalSamples(:,i), 'Color', 'blue');
        end
    end
    hold off
    axis('auto')
    ylim = get(gca, 'YLim');
    set(gca, 'XLim', [1 frameLength], 'YLim', ylim);
    xlabel('Frame')
    ylabel('Intensity')
    title('Temporal Signals');
end

```

```

subplot(4,3,2)
cla;
hold all
for i=1:numSamples
    if trainTarget(i) == 1
        plot(1:frameLength,temporalSamples(:,i));
    end
end
hold off
set(gca,'XLim',[1 frameLength],'YLim',ylim);
xlabel('Frame')
ylabel('Intensity')
seconds = [(currentEnd-frameLength)/30, currentEnd./30];
title([filename, ' ', inputSource, ' Features (',...
        num2str(currentStart),':', num2str(currentEnd), ' Frames - ',...
        num2str(seconds(1)),':',num2str(seconds(2)),'s)'];'Fire'));

subplot(4,3,3)
cla;
hold all
for i=1:numSamples
    if trainTarget(i) == 0
        plot(1:frameLength,temporalSamples(:,i));
    end
end
hold off
set(gca,'XLim',[1 frameLength],'YLim',ylim);
xlabel('Frame')
ylabel('Intensity')
title('Noise');

subplot(4,3,4)
cla;
hold on
for i=1:numFeatureSamples
    if featureTarget(i) == 1
        plot(1:frameLength,featureSamples(:,i),'Color','red');
    else
        plot(1:frameLength,featureSamples(:,i),'Color','blue');
    end
end
hold off
set(gca,'XLim',[1 frameLength],'YLim',ylim);
xlabel('Frame')
ylabel('Intensity')
title('Temporal Signals With Fire Characteristics');

subplot(4,3,5)
cla;
hold all
for i=1:numFeatureSamples
    if featureTarget(i) == 1
        plot(1:frameLength,featureSamples(:,i));
    end
end

```

```

hold off
set(gca,'XLim',[1 frameLength],'YLim',ylim);
xlabel('Frame')
ylabel('Intensity')
title('Fire');

subplot(4,3,6)
cla;
hold all
for i=1:numFeatureSamples
    if featureTarget(i) == 0
        plot(1:frameLength,featureSamples(:,i));
    end
end
hold off
set(gca,'XLim',[1 frameLength],'YLim',ylim);
xlabel('Frame')
ylabel('Intensity')
title('Noise');

a = subplot(4,3,11);
hold on
h1 = scatter3(positive_feature1,positive_feature2,positive_feature3,...
    '+','red');
h2 = scatter3(negative_feature1,negative_feature2,negative_feature3,...
    '.','blue');
hold off
grid on
xlabel(feature1Title)
ylabel(feature2Title)
zlabel(feature3Title)
n = currentEnd/frameLength;
view(45+(n-1)*6,15)
xlim = get(gca,'XLim');
ylim = get(gca,'YLim');
zlim = get(gca,'ZLim');
legend([h1 h2],{'Fire', 'Noise'},'Location','NorthEast');
title([filename, ' ', inputSource, ' Features (after ',...
    num2str(currentEnd), ' Frames - ', num2str(seconds(2)), 's)']);

subplot(4,3,7)
cla;
hold on
scatter3(positive_feature1,positive_feature2,positive_feature3,...
    '+','red')
scatter3(negative_feature1,negative_feature2,negative_feature3,...
    '.','blue')
hold off
grid on
xlabel(feature1Title)
ylabel(feature2Title)
zlabel(feature3Title)
view(0,0)
set(gca,'XLim',xlim,'YLim',ylim,'ZLim',zlim);

subplot(4,3,8)

```

```

cla;
hold on
scatter3(positive_feature1,positive_feature2,positive_feature3,...
        '+','red')
scatter3(negative_feature1,negative_feature2,negative_feature3,...
        '.','blue')
hold off
grid on
xlabel(feature1Title)
ylabel(feature2Title)
zlabel(feature3Title)
view(0,90)
set(gca,'XLim',xlim,'YLim',ylim,'ZLim',zlim);

subplot(4,3,10)
cla;
hold on
scatter3(positive_feature1,positive_feature2,positive_feature3,...
        '+','red')
scatter3(negative_feature1,negative_feature2,negative_feature3,...
        '.','blue')
hold off
grid on
xlabel(feature1Title)
ylabel(feature2Title)
zlabel(feature3Title)
view(90,0)
set(gca,'XLim',xlim,'YLim',ylim,'ZLim',zlim);

subtightplot(4,3,[9;12])
cla;
imshowpair(featureMask, featurePixelsPositive,'Scaling','independent')
xlabel({'Target Samples vs. Feature Samples'});

axes(a);
end
%% End of Function -----%

```

```

function [ thresh, pks, loc, count, bins ] = findThreshVideo(...
    inputVideo, vidRes, horizRes, vertRes, num )
%% Matlab Function -----%
% Function : findThreshVideo.m
% Author : Raul-Alexander Aldama
% Date : 2/22/2013
%
% Description : Find peaks in histogram for thresholding
%
% Notes :
% Revision(s) :
%-----%
[count, ~] = imhist(inputVideo(:), vidRes);
peakDistance = round((max(inputVideo(:))-min(inputVideo(:)))/horizRes);
if peakDistance < 1
    peakDistance = 1;
end
peakThresh = round(max(count)/vertRes);
if peakThresh < 1
    peakThresh = 1;
end
[pks,loc] = findpeaks(count, 'MINPEAKDISTANCE', peakDistance,...
    'THRESHOLD', peakThresh);
if num == 0
    num = length(loc);
end
thresh = loc(num);
[count, bins] = imhist(inputVideo(:), 2^16);
end
%% End of Function -----%

```

```

function [ featureTable, featureSamples, feature1Title, feature2Title,...
    feature3Title ] = generateFeaturePoints(featureRow, featureCol,...
    featureTarget, featureSamples, numFeatureSamples, distNH )
%% Matlab Function -----%
% Function : generateFeaturePoints.m
% Author : Raul-Alexander Aldama
% Date : 3/31/2013
%
% Description : Calculates the feature points from extracted features
%
% Notes :
% Revision(s) :
%-----%
if numFeatureSamples > 0
    featureTable = zeros(numFeatureSamples,7);

    if numFeatureSamples > 0
        indecies = 1:numFeatureSamples;
        featureTable(:,1) = indecies;
        featureTable(:,2) = featureRow;
        featureTable(:,3) = featureCol;

        feature1 = zeros(1, numFeatureSamples);
        for i=1:numFeatureSamples
            sumFeature1 = 0;
            refPoint = [featureRow(i), featureCol(i)];
            for j=1:numFeatureSamples
                point = [featureRow(j), featureCol(j)];
                if abs(point(1)-refPoint(1)) <= distNH && ...
                    abs(point(2)-refPoint(2)) <= distNH
                    sumFeature1 = sumFeature1 + std(featureSamples(:,j));
                end
            end
            feature1(i) = sumFeature1;
        end
        feature1 = feature1';
        featureTable(:,4) = feature1;
        %
        feature2 = zeros(1, numFeatureSamples);
        for i=1:numFeatureSamples
            sumFeature2 = 0;
            refPoint = [featureRow(i), featureCol(i)];
            for j=1:numFeatureSamples
                point = [featureRow(j), featureCol(j)];
                if abs(point(1)-refPoint(1)) <= distNH && ...
                    abs(point(2)-refPoint(2)) <= distNH
                    sumFeature2 = sumFeature2 + mean(featureSamples(:,j));
                end
            end
            feature2(i) = sumFeature2;
        end
        feature2 = feature2';
        featureTable(:,5) = feature2;
        %
        feature3 = zeros(1, numFeatureSamples);
        for i=1:numFeatureSamples

```

```

sumFeature3 = 0;
refPoint = [featureRow(i), featureCol(i)];
for j=1:numFeatureSamples
    point = [featureRow(j), featureCol(j)];
    if abs(point(1)-refPoint(1)) <= distNH && ...
        abs(point(2)-refPoint(2)) <= distNH
        pixelAutoCorr = xcorr(featureSamples(:,j),...
            featureSamples(:,j), 'coef');
        meanPixelAutoCorr = mean(pixelAutoCorr);
        sumFeature3 = sumFeature3 + meanPixelAutoCorr;
    end
end
feature3(i) = sumFeature3;
end
feature3 = feature3';
featureTable(:,6) = feature3;
%
featureTable(:,7) = featureTarget;
end
else
    featureTable = zeros(0,7);
end
feature1Title = 'NH STD DEV';
feature2Title = 'NH MEAN';
feature3Title = 'NH AVG AUTOCORR';
%% End of Function -----%

```



```

function [ frameList ] = generateFrameList( fileName, ...
    startFrame, endFrame, type )
%% Matlab Function -----%
% Function : generateFrameList.m
% Author : Raul-Alexander Aldama
% Date : 2/1/2013
%
% Description : Generates a list of frames from which a video matrix is
% formed
%
% Notes :
% Revision(s) :
%-----%
frameList = cell(1,endFrame-startFrame+1);
for i=1:(endFrame-startFrame+1)
    str = [char(fileName),num2str(i+startFrame-1),char(type)];
    frameList(i) = {str};
end
%% End of Function -----%

```

```

function [filteredVideo] = medFilterVideo(inputVideo, sqrSize)
%% Matlab Function -----%
% Function : medFilterVideo.m
% Author : Raul-Alexander Aldama
% Date : 3/2/2013
%
% Description : Performs sqrSize-by-sqrSize median filtering on a video
% matrix
%
% Notes :
% Revision(s) :
%-----%
frameCount = size(inputVideo,3);

filteredVideo = uint16(zeros(480,640,frameCount));

for i = 1:frameCount
    image = inputVideo(:,:,i);
    imageFilt = medfilt2(image,[sqrSize sqrSize],'symmetric');
    filteredVideo(:,:,i) = imageFilt;
end
%% End of Function -----%

```

```

function [writerObj] = objectFigMPEG4(fileName, frameRate)
%% Matlab Function -----%
% Function : objectFigMPEG4.m
% Author : Raul-Alexander Aldama
% Date : 3/28/2013
%
% Description : Creates a video object for recording video
%
% Notes :
% Revision(s) :
%-----%
writerObj = VideoWriter(char(fileName), 'MPEG-4');
writerObj.FrameRate = frameRate;
%% End of Function -----%

```

```

function [image] = openRawImage(filename, nRow, nColumn, bType)
%% Matlab Function -----%
% Function : openRawImage.m
% Author : Raul-Alexander Aldama
% Date : 2/1/2013
%
% Description : Imports individual raw video frames
%
% Notes :
% Revision(s) :
%-----%
fid = fopen(filename, 'r', 'l');
if (fid==1)
    error('Cannot open image file...press CTRL-C to exit ');
end
nSize = nRow*nColumn;
image = fread(fid, nSize, bType);
image = cast(image, bType);
image = reshape(image, nColumn, nRow);
image = image';
fclose(fid);
%% End of Function -----%

```

```

function [CL, CM, refCM, baseCM, diagOffset, regPoints] = ...
    preProcessVideo(CL, dynRangeCLWIR, CM, dynRangeCMWIR, CMCLratio,...
        bottomClipPerc, topClipPerc, filtSqrSize, intensityLim,...
        register_video, rPerc, s, frameLength, plot_enable, viewFrame,...
        filename, currentStart, currentEnd)
%% Matlab Function -----%
% Function : preProcessVideo.m
% Author : Raul-Alexander Aldama
% Date : 3/3/2013
%
% Description : Performs pre-processing on a n-frame set of CLWIR and CMWIR
% video
%
% Notes :
% Revision(s) :
%-----%
bottomClip = uint16(bottomClipPerc * intensityLim);
topClip = uint16((1-topClipPerc) * intensityLim);

if filtSqrSize > 0
    CL = medFilterVideo(CL, filtSqrSize);
    CM = medFilterVideo(CM, filtSqrSize);
end

dynRange = minmax([dynRangeCMWIR dynRangeCLWIR]);

CL = setHist(CL, dynRange, [0 intensityLim]);
CL = applyThreshVideo(CL,bottomClip,topClip);

CM = uint16(double(CM)./CMCLratio);
CM = setHist(CM, dynRange, [0 intensityLim]);
CM = applyThreshVideo(CM,bottomClip,topClip);

if register_video
    refCM = setHist(CM,minmax(CM(:)'),[0 intensityLim]);
    if rPerc < 1
        r = randperm(s,frameLength,round(rPerc*frameLength));
        refCMW = refCM(:, :, r);
    else
        refCMW = refCM;
    end
    baseCM = uint16(median(double(refCMW),3));

    [~, H, regPoints, diagOffset] = videoRegistration(refCMW, baseCM,...
        plot_enable, viewFrame);

    CL = transformVideo(CL, H);
    CM = transformVideo(CM, H);
else
    refCM = zeros(size(CM), 'uint16');
    baseCM = refCM(:, :, 1);
    diagOffset = zeros(size(frameLength));
    regPoints = {};
end

```

```

if plot_enable
    figure()
    imshow(CL(:,:,viewFrame))
    colormap('hot')
    title([filename, ' Pre-Processed CLWIR Frame Number ',...
            num2str(currentStart+viewFrame-1)])

    figure()
    imshow(CM(:,:,viewFrame))
    colormap('hot')
    title([filename, ' Pre-Processed CMWIR Frame ',...
            num2str(currentStart+viewFrame-1)])

    figure()
    imshow(baseCM)
    title([filename, ' Contrast Enhanced Composite CMWIR Reference Frame',...
            ' for Frames ', num2str(currentStart), ' to ', num2str(currentEnd)])

    figure()
    varRefCM = var(double(refCM),0,3);
    imshow(varRefCM,[])
    colormap('hot')
    title([filename, ' Variance of CMWIR Unregistered Frames ',...
            num2str(currentStart), ' to ', num2str(currentEnd)])

    figure()
    varCM = var(double(CM),0,3);
    imshow(varCM,[])
    colormap('hot')
    title([filename, ' Variance of CMWIR Registered Frames ',...
            num2str(currentStart), ' to ', num2str(currentEnd)])
end
%% End of Function -----%

```

```

function [ outputVideo ] = raw2Video( frameList, nRow, nColumn, bType )
%% Matlab Function -----%
% Function : raw2Video.m
% Author : Raul-Alexander Aldama
% Date : 2/1/2013
%
% Description : Creates a video matrix from raw frames
%
% Notes :
% Revision(s) :
%-----%
frameLength = length(frameList);
outputVideo = zeros(nRow,nColumn,frameLength,bType);
for i = 1:frameLength
    image = openRawImage(char(frameList(i)), nRow, nColumn, bType);
    outputVideo(:, :, i) = image;
end
%% End of Function -----%

```

```

function [diagOffset] = reborderVideo(inputVideo)
%% Matlab Function -----%
% Function : reborderVideo.m
% Author : Raul-Alexander Aldama
% Date : 3/17/2013
%
% Description : Identifies the video registration border edge artifacts
%
% Notes :
% Revision(s) :
%-----%
dim = size(inputVideo);
diagOffset = zeros([1 dim(3)]);

for i=1:dim(3)
    count1 = 0;
    count2 = 0;
    count3 = 0;
    count4 = 0;

    rowOffset = 0;
    colOffset = 0;
    pixel = -1;
    while pixel < 0
        pixel = inputVideo(1+rowOffset,1+colOffset,i);
        if pixel < 0
            count1 = count1 + 1;
        end
        rowOffset = rowOffset + 1;
        colOffset = colOffset + 1;
    end
    rowOffset = 0;
    colOffset = 0;
    pixel = -1;
    while pixel < 0
        pixel = inputVideo(1+rowOffset,dim(2)+colOffset,i);
        if pixel < 0
            count2 = count2 + 1;
        end
        rowOffset = rowOffset + 1;
        colOffset = colOffset - 1;
    end
    rowOffset = 0;
    colOffset = 0;
    pixel = -1;
    while pixel < 0
        pixel = inputVideo(dim(1)+rowOffset,1+colOffset,i);
        if pixel < 0
            count3 = count3 + 1;
        end
        rowOffset = rowOffset - 1;
        colOffset = colOffset + 1;
    end
    rowOffset = 0;
    colOffset = 0;
    pixel = -1;

```



```

while pixel < 0
    pixel = inputVideo(dim(1)+rowOffset,dim(2)+colOffset,i);
    if pixel < 0
        count4 = count4 + 1;
    end
    rowOffset = rowOffset - 1;
    colOffset = colOffset - 1;
end
diagOffset(i) = max([count1 count2 count3 count4]);
end
%% End of Function -----%

```

```

function [outputVideo] = setHist(inputVideo, histRange, valueRange)
%% Matlab Function -----%
% Function : setHist.m
% Author : Raul-Alexander Aldama
% Date : 2/27/2013
%
% Description : Applies a linear transformation to a scale a video matrix
%
% Notes :
% Revision(s) :
%-----%
min = histRange(1);
max = histRange(2);
a = valueRange(1);
b = valueRange(2);
outputVideo = uint16((((b-a)./(max-min)).*(inputVideo-min)+a));
%% End of Function -----%

```

```

function [pcaImages] = spectralPCA(dataCell, pcaImages)
%% Matlab Function -----%
% Function : spectralPCA.m
% Author : Raul-Alexander Aldama
% Date : 3/15/2013
%
% Description : Performs multi-spectral principal component analysis
%
% Notes :
% Revision(s) :
%-----%
frameNum = length(dataCell);
rows = size(dataCell{1}, 1);
cols = size(dataCell{1}, 2);
frameLength = size(dataCell{1}, 3);
data = zeros(rows,cols,frameLength.*frameNum);
for i=1:frameLength
    p = 1;
    for m=1:frameLength
        for n=1:frameNum
            data(:, :, p) = dataCell{n}(:, :, m);
            p = p + 1;
        end
    end
end

index = (i-1)*frameNum+1;
block = data(:, :, index:index+frameNum-1);
block = reshape(block, rows*cols, frameNum);
[block, ~, sigma] = zscore(block);
sigma = repmat(sigma, [size(block,1), 1]);
[~, pcaScoreBlock] = princomp(block);
pcaImagesBlock = abs(pcaScoreBlock).*sigma;
pcaImagesBlock = uint16(reshape(pcaImagesBlock, rows, cols, frameNum));

for m=1:frameNum
    pcaImages{m}(:, :, i) = pcaImagesBlock(:, :, m);
end
end
%% End of Function -----%

```

```

function h=subtightplot(m,n,p,gap,marg_h,marg_w,varargin)
%function h=subtightplot(m,n,p,gap,marg_h,marg_w,varargin)
%
% Functional purpose: A wrapper function for Matlab function subplot. Adds
the ability to define the gap between
% neighbouring subplots. Unfortunately Matlab subplot function lacks this
functionality, and the gap between
% subplots can reach 40% of figure area, which is pretty lavish.
%
% Input arguments (defaults exist):
%   gap- two elements vector [vertical,horizontal] defining the gap between
neighbouring axes. Default value
%       is 0.01. Note this value will cause titles legends and labels to
collide with the subplots, while presenting
%       relatively large axis.
%   marg_h margins in height in normalized units (0...1)
%         or [lower upper] for different lower and upper margins
%   marg_w margins in width in normalized units (0...1)
%         or [left right] for different left and right margins
%
% Output arguments: same as subplot- none, or axes handle according to
function call.
%
% Issues & Comments: Note that if additional elements are used in order to be
passed to subplot, gap parameter must
%       be defined. For default gap value use empty element- [].
%
% Usage example: h=subtightplot((2,3,1:2,[0.5,0.2]))

if (nargin<4) || isempty(gap),    gap=0.01; end
if (nargin<5) || isempty(marg_h), marg_h=0.05; end
if (nargin<5) || isempty(marg_w), marg_w=marg_h; end
if isscalar(gap),    gap(2)=gap; end
if isscalar(marg_h), marg_h(2)=marg_h; end
if isscalar(marg_w), marg_w(2)=marg_w; end
gap_vert    = gap(1);
gap_horz    = gap(2);
marg_lower  = marg_h(1);
marg_upper  = marg_h(2);
marg_left   = marg_w(1);
marg_right  = marg_w(2);

%note n and m are switched as Matlab indexing is column-wise, while subplot
indexing is row-wise :(
[subplot_col,subplot_row]=ind2sub([n,m],p);

% note subplot supports vector p inputs- so a merged subplot of higher
dimensions will be created
subplot_cols=1+max(subplot_col)-min(subplot_col); % number of column elements
in merged subplot
subplot_rows=1+max(subplot_row)-min(subplot_row); % number of row elements in
merged subplot

% single subplot dimensions:
%height=(1-(m+1)*gap_vert)/m;
%axh = (1-sum(marg_h)-(Nh-1)*gap(1))/Nh;

```

```

height=(1-(marg_lower+marg_upper)-(m-1)*gap_vert)/m;
%width =(1-(n+1)*gap_horz)/n;
%axw = (1-sum(marg_w)-(Nw-1)*gap(2))/Nw;
width =(1-(marg_left+marg_right)-(n-1)*gap_horz)/n;

% merged subplot dimensions:
merged_height=subplot_rows*( height+gap_vert )- gap_vert;
merged_width= subplot_cols*( width +gap_horz )- gap_horz;

% merged subplot position:
merged_bottom=(m-max(subplot_row))*(height+gap_vert) +marg_lower;
merged_left=(min(subplot_col)-1)*(width+gap_horz) +marg_left;
pos_vec=[merged_left merged_bottom merged_width merged_height];

% h_subplot=subplot(m,n,p,varargin{:},'Position',pos_vec);
% Above line doesn't work as subplot tends to ignore 'position' when same mnp
is utilized
h=subplot('Position',pos_vec,varargin{:});

if (nargout < 1), clear h; end

end

```

```

function [pcaImages, frameBuffer] = temporalPCA(data, pcaImages,...
    frameBuffer, enable)
%% Matlab Function -----%
% Function : temporalPCA.m
% Author : Raul-Alexander Aldama
% Date : 3/15/2013
%
% Description : Performs multi-temporal principal component analysis
%
% Notes :
% Revision(s) :
%-----%
rows = size(data,1);
cols = size(data,2);
temporalLength = size(data,3);
for i=1:temporalLength
    if i==1
        frameBuffer(:, :, 1:temporalLength) = ...
            frameBuffer(:, :, temporalLength+1:2*temporalLength);
    end
    frameBuffer(:, :, temporalLength+1:2*temporalLength) = data;
    if enable==1
        block = frameBuffer(:, :, i:i+temporalLength-1);
        block = reshape(block, rows*cols, temporalLength);
        [block, ~, sigma] = zscore(block);
        sigma = repmat(sigma, [size(block,1), 1]);
        [~, pcaScoreBlock] = princomp(block);
        pcaImagesBlock = abs(pcaScoreBlock).*sigma;
        pcaImagesBlock = uint16(reshape(pcaImagesBlock, rows, cols, ...
            temporalLength));

        for m=1:temporalLength
            pcaImages{m}(:, :, i) = pcaImagesBlock(:, :, m);
        end
    end
end
%% End of Function -----%

```

```

function [outputVideo] = tif2Video(frameList)
%% Matlab Function -----%
% Function : tif2Video.m
% Author : Raul-Alexander Aldama
% Date : 2/7/2013
%
% Description : Creates a video matrix from tif frames
%
% Notes :
% Revision(s) :
%-----%
frameLength = length(frameList);
image = imread(char(frameList(1)), 'tif');
rows = size(image,1);
cols = size(image,2);
bType = class(image);
outputVideo = zeros(rows, cols, frameLength, bType);
for i = 1:frameLength
    image = imread(char(frameList(i)), 'tif');
    outputVideo(:,:,i) = image;
end
%% End of Function -----%

```

```

function hfig = tightfig(hfig)
% tightfig: Alters a figure so that it has the minimum size necessary to
% enclose all axes in the figure without excess space around them.
%
% Note that tightfig will expand the figure to completely encompass all
% axes if necessary. If any 3D axes are present which have been zoomed,
% tightfig will produce an error, as these cannot easily be dealt with.
%
% hfig - handle to figure, if not supplied, the current figure will be used
% instead.

    if nargin == 0
        hfig = gcf;
    end

    % There can be an issue with tightfig when the user has been modifying
    % the contents manually, the code below is an attempt to resolve this,
    % but it has not yet been satisfactorily fixed
    origwindowstyle = get(hfig, 'WindowStyle');
    set(hfig, 'WindowStyle', 'normal');

    % 1 point is 0.3528 mm for future use

    % get all the axes handles note this will also fetch legends and
    % colorbars as well
    hax = findall(hfig, 'type', 'axes');

    % get the original axes units, so we can change and reset these again
    % later
    origaxunits = get(hax, 'Units');

    % change the axes units to cm
    set(hax, 'Units', 'centimeters');

    % get various position parameters of the axes
    if numel(hax) > 1
        fsize = cell2mat(get(hax, 'FontSize'));
        ti = cell2mat(get(hax, 'TightInset'));
        pos = cell2mat(get(hax, 'Position'));
    else
        fsize = get(hax, 'FontSize');
        ti = get(hax, 'TightInset');
        pos = get(hax, 'Position');
    end

    % ensure very tiny border so outer box always appears
    ti(ti < 0.1) = 0.15;

    % we will check if any 3d axes are zoomed, to do this we will check if
    % they are not being viewed in any of the 2d directions
    views2d = [0,90; 0,0; 90,0];

    for i = 1:numel(hax)

        set(hax(i), 'LooseInset', ti(i,:));
    end

```



```

%         set(hax(i), 'LooseInset', [0,0,0,0]);

% get the current viewing angle of the axes
[az,el] = view(hax(i));

% determine if the axes are zoomed
iszoomed = strcmp(get(hax(i), 'CameraViewAngleMode'), 'manual');

% test if we are viewing in 2d mode or a 3d view
is2d = all(bsxfun(@eq, [az,el], views2d), 2);

if iszoomed && ~any(is2d)
    error('TIGHTFIG:haszoomed3d', 'Cannot make figures containing
zoomed 3D axes tight.')
end

end

% we will move all the axes down and to the left by the amount
% necessary to just show the bottom and leftmost axes and labels etc.
moveleft = min(pos(:,1) - ti(:,1));

movedown = min(pos(:,2) - ti(:,2));

% we will also alter the height and width of the figure to just
% encompass the topmost and rightmost axes and labels
figwidth = max(pos(:,1) + pos(:,3) + ti(:,3) - moveleft);

figheight = max(pos(:,2) + pos(:,4) + ti(:,4) - movedown);

% move all the axes
for i = 1:numel(hax)
    set(hax(i), 'Position', [pos(i,1:2) - [moveleft,movedown],
pos(i,3:4)]);
end

origfigunits = get(hfig, 'Units');

set(hfig, 'Units', 'centimeters');

% change the size of the figure
figpos = get(hfig, 'Position');

set(hfig, 'Position', [figpos(1), figpos(2), figwidth, figheight]);

% change the size of the paper
set(hfig, 'PaperUnits','centimeters');
set(hfig, 'PaperSize', [figwidth, figheight]);
set(hfig, 'PaperPositionMode', 'manual');
set(hfig, 'PaperPosition',[0 0 figwidth figheight]);

```

```

% reset to original units for axes and figure
if ~iscell(origaxunits)
    origaxunits = {origaxunits};
end

for i = 1:numel(hax)
    set(hax(i), 'Units', origaxunits{i});
end

set(hfig, 'Units', origfigunits);

%     set(hfig, 'WindowStyle', origwindowstyle);

end

```

```

function [ outputVideo ] = transformVideo( inputVideo, H )
%% Matlab Function -----%
% Function : transformVideo.m
% Author : Raul-Alexander Aldama
% Date : 2/7/2013
%
% Description : Applies an affine transformation (H) to a video matrix
%
% Notes :
% Revision(s) :
%-----%
rows = size(inputVideo, 1);
cols = size(inputVideo, 2);
frameLength = size(inputVideo, 3);
outputVideo = zeros(rows,cols,frameLength);
hGTPrj = vision.GeometricTransformer;
for i = 1:frameLength
    img = single(inputVideo(:,:,i));
    h = H{i};
    imgBp = step(hGTPrj, img, h);
    outputVideo(:,:,i) = imgBp;
end

outputVideo = uint16(outputVideo);
%% End of Function -----%

```

```

function [filteredVideo] = unsharpFilterVideo(inputVideo, strength,...
    sqrSize, sigma)
%% Matlab Function -----%
% Function : unsharpFilterVideo.m
% Author : Raul-Alexander Aldama
% Date : 3/22/2013
%
% Description : Applies unsharp mask filter to a set of video frames
%
% Notes :
% Revision(s) :
%-----%
H = fspecial('gaussian',[sqrSize sqrSize],sigma);

rows = size(inputVideo,1);
cols = size(inputVideo,2);
frameCount = size(inputVideo,3);
filteredVideo = uint16(zeros(rows,cols,frameCount));

for i = 1:frameCount
    image = inputVideo(:,:,i);
    lowPass = imfilter(image,H);
    imageFilt = image - strength*lowPass;
    filteredVideo(:,:,i) = imageFilt;
end
%% End of Function -----%

```

```

function [registeredVideo, H, points, diagOffset] = videoRegistration(...
    inputVideo, baseFrame, plot_enable, viewFrame)
%% Matlab Function -----%
% Function : videoRegistration.m
% Author : Raul-Alexander Aldama, MathWorks
% Date : 2/4/2013
%
% Description : Determines the affine transformation for video registration
%
% Notes :
% Revision(s) :
%-----%
rows = size(inputVideo, 1);
cols = size(inputVideo, 2);
frameCount = size(inputVideo, 3);

registeredVideo = zeros(rows,cols,frameCount);
H = cell(1,frameCount);
points = cell(1,frameCount);
imgA = single(baseFrame);

maxPts = 1000;
ptThresh = 1e-3;
hCD = vision.CornerDetector( ...
    'Method','Harris corner detection (Harris & Stephens)', ...
    'MaximumCornerCount', maxPts, ...
    'CornerThreshold', ptThresh, ...
    'NeighborhoodSize', [7 7]);
pointsA = step(hCD, imgA);
blockSize = 15; % Block size.

hGTE = vision.GeometricTransformEstimator;
hGT = vision.GeometricTransformer;
hGTPrj = vision.GeometricTransformer('BackgroundFillValue',-1);

nRansacTrials = 1;
Ts = cell(1,nRansacTrials);
costs = zeros(1,nRansacTrials);
inliers = cell(1,nRansacTrials);

for i = 1:frameCount
    imgB = single(inputVideo(:, :, i));
    if plot_enable && i == viewFrame
        imgA_u16 = uint16(imgA);
        imgB_u16 = uint16(imgB);
        cvexShowImagePair(imgA_u16, imgB_u16, 'Reference Frame',...
            'Video Frame');
        figure; imshow(cat(3,imgA_u16,imgB_u16,imgB_u16),[]);
        title('Color composite (Reference Frame = red, ',...
            'Unregistered Frame = cyan)');
    end
    pointsB = step(hCD, imgB);

    if plot_enable && i == viewFrame
        imgA_u16 = uint16(imgA);
        imgB_u16 = uint16(imgB);

```

```

        cvexShowImagePair(imgA_u16, imgB_u16,...
            'Corners in Reference Frame',...
            'Corners in Unregistered Frame',...
            'SingleColor', pointsA, pointsB);
    end
    % Extract features for the corners
    [featuresA, pointsA] = extractFeatures(imgA, pointsA, ...
        'BlockSize', blockSize);
    [featuresB, pointsB] = extractFeatures(imgB, pointsB, ...
        'BlockSize', blockSize);

    % Match features which were found in the current and previous frames
    indexPairs = matchFeatures(featuresA, featuresB, 'Metric', 'SSD');
    pointsA = pointsA(indexPairs(:, 1), :);
    pointsB = pointsB(indexPairs(:, 2), :);
    if plot_enable && i == viewFrame
        imgA_u16 = uint16(imgA);
        imgB_u16 = uint16(imgB);
        cvexShowMatches(imgA_u16, imgB_u16, pointsA, pointsB,...
            'Reference Point', 'Unregistered Point');
    end

    % Run multiple RANSAC trials
    for j=1:nRansacTrials
        % Estimate affine transform
        [Ts{j}, inliers{j}] = step(hGTE, pointsB, pointsA);

        % Warp image and compute error metric.
        imgBp = step(hGT, imgB, Ts{j});
        costs(j) = sum(sum(imabsdiff(imgBp, imgA)));
    end
    % Take best result.
    [~,ix] = min(costs);

    pointsBp = [single(pointsB), ones(size(pointsB,1), 1)] * Ts{ix};
    h = [Ts{ix} [0 0 1]'];

    if plot_enable && i == viewFrame
        imgA_u16 = uint16(imgA);
        imgBp_u16 = uint16(imgBp);
        cvexShowMatches(imgA_u16, imgBp_u16, pointsA(inliers{ix}, :),...
            pointsBp(inliers{ix}, :), 'Reference Point',...
            'Unregistered Point');
    end
    imgBp = step(hGTPrj, imgB, h);
    registeredVideo(:, :, i) = imgBp;
    H{i} = h;
    points{i} = [pointsA, pointsB];
end
diagOffset = reborderVideo(registeredVideo);
registeredVideo = uint16(registeredVideo);
%% End of Function -----%
```

```

function [] = writeFigMPEG4(writerObj, figHandle, stopFlag)
%% Matlab Function -----%
% Function : writeFigMPEG4.m
% Author : Raul-Alexander Aldama
% Date : 3/28/2013
%
% Description : Writes MPEG-4 video of a figure to a video object and
% closes the object when finished
%
% Notes :
% Revision(s) :
%-----%
open(writerObj);
frame = getframe(figHandle);
writeVideo(writerObj, frame);

if stopFlag
    close(writerObj);
end
%% End of Function -----%

```

```

function [] = writeMPEG4(fileName, frameList, startFrame)
%% Matlab Function -----%
% Function : writeMPEG4.m
% Author : Raul-Alexander Aldama
% Date : 2/21/2013
%
% Description : Writes MPEG-4 video from images on hard drive, adds scale
% and frame number
%
% Notes :
% Revision(s) :
%-----%
intensityLim = 65535;
scalePos = 10;
scaleLength = 20;
frameLength = length(frameList);
writerObj = VideoWriter(char(fileName), 'MPEG-4');
open(writerObj);
for i=1:frameLength
    image = double(imread(char(frameList(i)), 'tif'))/intensityLim;
    dim = size(image);
    rgbImage = cat(3, image, image, image);
    htxtins = vision.TextInserter(num2str(startFrame+i-1));
    htxtins.Color = [255, 255, 255]; % [red, green, blue]
    htxtins.FontSize = 12;
    htxtins.Location = [1 1]; % [x y]
    rgbImage = step(htxtins, rgbImage);
    rgbImage(scalePos, ...
        dim(2)-scalePos-scaleLength+1:dim(2)-scalePos, 1) = 1;
    rgbImage(scalePos, ...
        dim(2)-scalePos-scaleLength+1:dim(2)-scalePos, 2) = 0;
    rgbImage(scalePos, ...
        dim(2)-scalePos-scaleLength+1:dim(2)-scalePos, 3) = 0;
    writeVideo(writerObj, rgbImage);
end
close(writerObj);
%% End of Function -----%

```



```

function [ ] = writeMPEG4_PCA( folderPath, startFrame, endFrame,...
    pcaNumber, temporalLength, filename )
%% Matlab Function -----%
% Function : writeMPEG4_PCA.m
% Author : Raul-Alexander Aldama
% Date : 3/5/2013
%
% Description : Writes MPEG-4 video from PCA images on hard drive, adds
% scale and frame number
%
% Notes :
% Revision(s) :
%-----%
for i=1:pcaNumber
    frameList = generateFrameList(strcat(folderPath,'PCA_',num2str(i),...
        '\',filename,'_'),...
        startFrame,endFrame-temporalLength, '.tif');
    writeMPEG4(strcat(folderPath,'PCA_',num2str(i),'\',filename,'_'),...
        frameList, startFrame);
end
%% End of Function -----%

```

```

function [ ] = writeTifs( inputFrames, frameList)
%% Matlab Function -----%
% Function : writeTifs.m
% Author : Raul-Alexander Aldama
% Date : 2/21/2013
%
% Description : Writes .tif images to hard drive
%
% Notes :
% Revision(s) :
%-----%
frameCount = length(frameList);

for i=1:frameCount
    imwrite(inputFrames(:, :, i), char(frameList(i)), 'tif', ...
        'Compression', 'none')
end
%% End of Function -----%

```

```

function [ ] = writeTifs_PCA(pcaScore, folderPath,...
    currentStart, currentEnd, temporalLength, filename, enable)
%% Matlab Function -----%
% Function : writeTifs_PCA.m
% Author : Raul-Alexander Aldama
% Date : 3/5/2013
%
% Description : Writes .tif PCA images to hard drive
%
% Notes :
% Revision(s) :
%-----%
if enable == 1
    for i = 1:length(pcaScore)
        [~,~,~] = mkdir(folderPath, strcat('PCA_', num2str(i)));
        frameBlock = pcaScore{i};
        frameList = generateFrameList(...
            strcat(folderPath, 'PCA_', num2str(i), '\', filename, '_'), ...
            currentStart-temporalLength, currentEnd-temporalLength, '.tif');
        writeTifs(frameBlock, frameList);
    end
end
%% End of Function -----%

```

```

%% Matlab GUI -----%
% GUI : liveVideoViewer.m
% Author : Raul-Alexander Aldama
% Date : 2/4/2013
%
% Description : Allows user to view a video matrix from the workplace
% and scale, threshold, view pixel information, calculate distances,
% crop video, and write the resulting matrix back to the workplace
%
% Notes :
% Revision(s) :
%-----%
function varargout = liveVideoViewer(varargin)
% LIVEVIDEOVIEWER MATLAB code for liveVideoViewer.fig
%     LIVEVIDEOVIEWER, by itself, creates a new LIVEVIDEOVIEWER or raises
the existing
%     singleton*.
%
%     H = LIVEVIDEOVIEWER returns the handle to a new LIVEVIDEOVIEWER or the
handle to
%     the existing singleton*.
%
%     LIVEVIDEOVIEWER('CALLBACK',hObject,eventData,handles,...) calls the
local
%     function named CALLBACK in LIVEVIDEOVIEWER.M with the given input
arguments.
%
%     LIVEVIDEOVIEWER('Property','Value',...) creates a new LIVEVIDEOVIEWER
or raises the
%     existing singleton*. Starting from the left, property value pairs are
%     applied to the GUI before liveVideoViewer_OpeningFcn gets called. An
%     unrecognized property name or invalid value makes property application
%     stop. All inputs are passed to liveVideoViewer_OpeningFcn via
varargin.
%
%     *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%     instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help liveVideoViewer

% Last Modified by GUIDE v2.5 24-Feb-2013 09:43:04

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',   gui_Singleton, ...
                  'gui_OpeningFcn', @liveVideoViewer_OpeningFcn, ...
                  'gui_OutputFcn',  @liveVideoViewer_OutputFcn, ...
                  'gui_LayoutFcn',   [] , ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

```

```

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before liveVideoViewer is made visible.
function liveVideoViewer_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject      handle to figure
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
% varargin     command line arguments to liveVideoViewer (see VARARGIN)

%opengl SOFTWARE;
global stopReq;
global currentWindow;
global updateTime;
global minLevel;
global maxLevel;
global xPoints;
global yPoints;
global dataStream;
global dataStreamEQ;
global dataStreamPT;
global avg;
global sigma;
global minValue;
global maxValue;
global image;
global histWindow;
global imageWindow;
global imageXLim;
global imageYLim;
global histXLim;
global histYLim;
global currentFrame;
global hImageLimitTable;
global hImageWindow;
global length;

avg = 0;
sigma = 0;
minValue = 0;
maxValue = 65535;
length = 0;
imageXLim = [1 640];
imageYLim = [1 480];
stopReq = 0;
updateTime = 1;
minLevel = 0;
maxLevel = 65535;
xPoints = [0,0];
yPoints = [0,0];
dataStream = uint16(zeros(480,640,1));

```

```

dataStreamEQ = uint16(zeros(480,640,1));
dataStreamPT = uint16(zeros(480,640,1));
image = uint16(zeros(480,640));
histWindow = handles.histWindow;
imageWindow = handles.imageWindow;
currentWindow = imageWindow;
currentFrame = 1;
hImageLimitTable = handles.imageLimitTable;
hImageWindow = handles.imageWindow;

handles.frameCount = 1;
handles.inputData = dataStream;
imshow(image, [0 65535], 'Parent', handles.imageWindow);
hold(handles.imageWindow, 'on');
plot(handles.imageWindow, linspace(double(imageXLim(1)+1),
double(imageXLim(1)+1+20)), ...

linspace(double(imageYLim(1)+1), double(imageYLim(1)+1), 'Color', 'red', 'LineWi
dth', 3);
hold(handles.imageWindow, 'off');
set(zoom, 'ActionPostCallback', @window_ActionPostCallback);
set(pan, 'ActionPostCallback', @window_ActionPostCallback);
set(zoom, 'ActionPreCallback', @window_ActionPreCallback);
set(pan, 'ActionPreCallback', @window_ActionPreCallback);

histXLim = [0 65535];
histYLim = [0 640*480];

[count, bins] = imhist(image, 2^16);
stem(handles.histWindow, bins, count, 'Marker', 'none')
axis(handles.histWindow, 'tight');
% set(handles.histWindow, 'XTickLabel', []);
% stem(handles.colorbarWindow, bins, count, 'Marker', 'none')
% axis(handles.colorbarWindow, 'tight');
% set(handles.colorbarWindow, 'YTickLabel', [], 'Clim', [0 65535]);
% colormap('Gray');
% c = colorbar('Location', 'North');
% p = get(handles.colorbarWindow, 'Position');
% set(c, 'Position', p);
% Choose default command line output for liveVideoViewer
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes liveVideoViewer wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = liveVideoViewer_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

```

```

% Get default command line output from handles structure
varargout{1} = handles.output;

function minText_Callback(hObject, eventdata, handles)
% hObject      handle to minText (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of minText as text
%         str2double(get(hObject,'String')) returns contents of minText as a
double
global minLevel;
textValue = uint16(str2double(get(handles.minText,'String')));
set(handles.minSlider,'Value',textValue);
minLevel = textValue;
adjustFrames(hObject, eventdata, handles);
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function minText_CreateFcn(hObject, eventdata, handles)
% hObject      handle to minText (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function maxText_Callback(hObject, eventdata, handles)
% hObject      handle to maxText (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of maxText as text
%         str2double(get(hObject,'String')) returns contents of maxText as a
double
global maxLevel;
textValue = uint16(str2double(get(handles.maxText,'String')));
set(handles.maxSlider,'Value',textValue);
maxLevel = textValue;
adjustFrames(hObject, eventdata, handles);
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function maxText_CreateFcn(hObject, eventdata, handles)
% hObject      handle to maxText (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.

```

```

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on slider movement.
function minSlider_Callback(hObject, eventdata, handles)
% hObject    handle to minSlider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of
slider
global minLevel;
sliderValue = num2str( uint16( get(handles.minSlider,'Value') ) );
set(handles.minText,'String', sliderValue);
minLevel = uint16( get(handles.minSlider,'Value') );
adjustFrames(hObject, eventdata, handles);
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function minSlider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to minSlider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
sliderMin = 0;
sliderMax = 65535;
sliderStep = [(16/65535), (16/65535)]; % major and minor steps of 1

set(hObject, 'Min', sliderMin);
set(hObject, 'Max', sliderMax);
set(hObject, 'SliderStep', sliderStep);
set(hObject, 'Value', sliderMin);
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function maxSlider_Callback(hObject, eventdata, handles)
% hObject    handle to maxSlider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of
slider
global maxLevel;
sliderValue = num2str( uint16( get(handles.maxSlider,'Value') ) );
set(handles.maxText,'String', sliderValue);
maxLevel = uint16( get(handles.maxSlider,'Value') );
adjustFrames(hObject, eventdata, handles);

```



```

guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function maxSlider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to maxSlider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
sliderMin = 0;
sliderMax = 65535;
sliderStep = [(16/65535), (16/65535)]; % major and minor steps of 1

set(hObject, 'Min', sliderMin);
set(hObject, 'Max', sliderMax);
set(hObject, 'SliderStep', sliderStep);
set(hObject, 'Value', sliderMax);
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
stopButton_Callback(hObject, eventdata, handles);
delete(hObject);

% --- Executes on button press in startButton.
function startButton_Callback(hObject, eventdata, handles)
% hObject    handle to startButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global stopReq;
global updateTime;
global dataStream;
global dataStreamEQ;
global dataStreamPT;
global image;
global currentFrame;
global minLevel;
global maxLevel;
global xPoints;
global yPoints;
global imageXLim;
global imageYLim;
global histXLim;
global histYLim;
global avg;
global sigma;
global minValue;
global maxValue;

```

```

global length;

stopReq = 0;

try
    textValue = get(handles.inputFile, 'String');
    handles.inputData = evalin('base',textValue);
    set(handles.inputFile, 'ForegroundColor', 'black');
    dataStream = handles.inputData;
    dataStreamPT = dataStream;
    avg = uint16(mean(dataStream(:)));
    sigma = uint16(sqrt(var(double(dataStream(:)))));
    handles.frameCount = size(dataStream, 3);
    minSlider_Callback(hObject, eventdata, handles);
    maxSlider_Callback(hObject, eventdata, handles);
    imageLimitTable_CellEditCallback(hObject, eventdata, handles);
    [count, ~] = imhist(handles.inputData(:, :, 1), 2^16);
    histYLim = [0 max(count)];
    guidata(hObject, handles);
    while (stopReq == 0)
        for i = 1:handles.frameCount
            if (stopReq == 0)
                image = dataStream(:, :, i);
                currentFrame = i;
                imshow(image, [0 65535], 'Parent', handles.imageWindow);
                set(handles.imageWindow, 'XLim', imageXLim, 'YLim', imageYLim);
                hold(handles.imageWindow, 'on');
                plot(handles.imageWindow, linspace(double(imageXLim(1)+1),
double(imageXLim(1)+1+20)), ...

linspace(double(imageYLim(1)+1), double(imageYLim(1))+1), 'Color', 'red', 'LineWi
dth', 3);

                if (~isequaln(xPoints, [0,0]) && ~isequaln(yPoints, [0,0]))

plot(handles.imageWindow, linspace(xPoints(1), xPoints(2)), linspace(yPoints(1),
yPoints(2)), 'Color', 'green', 'LineWidth', 2);

plot(handles.imageWindow, linspace(double(imageXLim(1)+1), double(imageXLim(1)+
length)), ...

linspace(double(imageYLim(1)+1), double(imageYLim(1))+1), 'Color', 'green', 'Line
Width', 2);

text(double(imageXLim(1)), double(imageYLim(2)), strcat(num2str(length), 'px'), '
Color', 'green', ...

'VerticalAlignment', 'Bottom', 'HorizontalAlignment', 'Left')
end
hold(handles.imageWindow, 'off');
figure1_WindowButtonMotionFcn(hObject, eventdata, handles);
set(handles.imageLimitTable, 'data', [imageXLim; imageYLim]);
[imageCount, bins] = imhist(image, 2^16);
stem(handles.histWindow, bins, imageCount, 'Marker', 'none');
hold(handles.histWindow, 'on');
toggleStatus = get(handles.histeqButton, 'Value');
if ~toggleStatus

```

```

        [inputCount, ~] = imhist(handles.inputData(:, :, i), 2^16);
        h = stem(handles.histWindow, bins, inputCount,
'Marker', 'none');
        set(h, 'Color', 'cyan');
    else
        [inputCountEQ, ~] = imhist(dataStreamEQ(:, :, i), 2^16);
        h = stem(handles.histWindow, bins, inputCountEQ,
'Marker', 'none');
        set(h, 'Color', 'cyan');
    end
    set(handles.histWindow, 'XLim', histXLim, 'YLim', histYLim);
    vertLimit = max(count);
    avg = uint16(mean(dataStream(:)));
    sigma = uint16(sqrt(var(double(dataStream(:)))));
    minValue = uint16(min(dataStream(:)));
    maxValue = uint16(max(dataStream(:)));

    plot(handles.histWindow, linspace(double(minLevel), double(minLevel)), linspace(
0, vertLimit), 'Color', 'green', 'LineWidth', 2);

    plot(handles.histWindow, linspace(double(maxLevel), double(maxLevel)), linspace(
0, vertLimit), 'Color', 'red', 'LineWidth', 2);

    plot(handles.histWindow, linspace(double(avg), double(avg)), linspace(0, vertLimit),
'Color', 'black', 'LineStyle', ':');
    n = avg + sigma;
    while n < maxValue

    plot(handles.histWindow, linspace(double(n), double(n)), linspace(0, vertLimit), '
Color', 'yellow', 'LineStyle', ':');
    n = n + sigma;
    end
    n = avg - sigma;
    while n > minValue

    plot(handles.histWindow, linspace(double(n), double(n)), linspace(0, vertLimit), '
Color', 'yellow', 'LineStyle', ':');
    n = n - sigma;
    end

    plot(handles.histWindow, linspace(double(minValue), double(minValue)), linspace(
0, vertLimit), 'Color', 'magenta', 'LineStyle', ':');

    plot(handles.histWindow, linspace(double(maxValue), double(maxValue)), linspace(
0, vertLimit), 'Color', 'magenta', 'LineStyle', ':');
    hold(handles.histWindow, 'off');
    pause(updateTime);
    end
end
end
catch exception
    handles.inputData = zeros(480, 640, 1);
    set(handles.inputFile, 'ForegroundColor', 'red');
    % rethrow(exception);
end

% --- Executes on button press in stopButton.

```

```

function stopButton_Callback(hObject, eventdata, handles)
% hObject      handle to stopButton (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

global stopReq;
stopReq = 1;
%guidata(hObject,handles);

% --- Executes on button press in outputButton.
function outputButton_Callback(hObject, eventdata, handles)
% hObject      handle to outputButton (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
global dataStream;
global imageXLim;
global imageYLim;
global minLevel;
global maxLevel;
stopButton_Callback(hObject, eventdata, handles);
outputFilename = get(handles.outputFile, 'String');
if (~isequaln(imageXLim, [1,640]) && ~isequaln(imageYLim, [1,480]))
    image = imresize(dataStream(imageYLim(1):imageYLim(2),
imageXLim(1):imageXLim(2), :), [480 640]);
else
    image = dataStream;
end
assignin('base',outputFilename,image);
assignin('base','minLevel',minLevel);
assignin('base','maxLevel',maxLevel);
guidata(hObject,handles);
close(handles.figure1);

function inputFile_Callback(hObject, eventdata, handles)
% hObject      handle to inputFile (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of inputFile as text
%        str2double(get(hObject,'String')) returns contents of inputFile as a
double
stopButton_Callback(hObject, eventdata, handles);
startButton_Callback(hObject, eventdata, handles);
% pause(1);
% stopButton_Callback(hObject, eventdata, handles);
% guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function inputFile_CreateFcn(hObject, eventdata, handles)
% hObject      handle to inputFile (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.

```

```

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function outputFile_Callback(hObject, eventdata, handles)
% hObject    handle to outputFile (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of outputFile as text
%        str2double(get(hObject,'String')) returns contents of outputFile as
a double
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function outputFile_CreateFcn(hObject, eventdata, handles)
% hObject    handle to outputFile (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function updateText_Callback(hObject, eventdata, handles)
% hObject    handle to updateText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of updateText as text
%        str2double(get(hObject,'String')) returns contents of updateText as
a double
global updateTime;
updateValue = str2double(get(handles.updateText,'String'));
updateTime = updateValue;
fpsValue = 1/updateTime;
fpsText = num2str(fpsValue);
set(handles.fpsText,'String',fpsText);
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function updateText_CreateFcn(hObject, eventdata, handles)
% hObject    handle to updateText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

```

end

function fpsText_Callback(hObject, eventdata, handles)
% hObject      handle to fpsText (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of fpsText as text
%         str2double(get(hObject,'String')) returns contents of fpsText as a
double
global updateTime;
fpsValue = str2double(get(handles.fpsText, 'String'));
updateTime = 1/fpsValue;
updateText = num2str(updateTime);
set(handles.updateText, 'String', updateText);
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function fpsText_CreateFcn(hObject, eventdata, handles)
% hObject      handle to fpsText (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject, 'BackgroundColor'),
get(0, 'defaultUicontrolBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end

function adjustFrames(hObject, eventdata, handles)
global dataStream;
global dataStreamEQ;
global dataStreamPT;
global maxLevel;
global minLevel;
guidata(hObject, handles);
toggleStatusHist = get(handles.histeqButton, 'Value');
toggleStatusMode = get(handles.modeButton, 'Value');

if toggleStatusHist && ~toggleStatusMode
    dataStream = uint16((65535./(maxLevel-minLevel)).*(uint16(dataStreamEQ)-
minLevel));
elseif ~toggleStatusHist && ~toggleStatusMode
    dataStream = uint16((65535./(maxLevel-
minLevel)).*(uint16(handles.inputData)-minLevel));
elseif toggleStatusHist && toggleStatusMode
    dataStream = dataStreamEQ;
    index = (dataStream<minLevel);
    dataStream(index) = 0;
    index = (dataStream>maxLevel);
    dataStream(index) = 0;

elseif ~toggleStatusHist && toggleStatusMode
    dataStream = dataStreamPT;

```

```

        index = (dataStream<minLevel);
        dataStream(index) = 0;
        index = (dataStream>maxLevel);
        dataStream(index) = 0;
end

guidata(hObject,handles);

% --- Executes on mouse press over figure background, over a disabled or
% --- inactive control, or over an axes background.
function figure1_WindowButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global minLevel;
global maxLevel;
global xPoints;
global yPoints;
global length;
global imageXLim;
global imageYLim;
global image;
if gca == handles.histWindow
    [x] = ginput(2);
    minLevel = uint16(x(1));
    maxLevel= uint16(x(2));
    textValue = num2str(minLevel);
    sliderValue = minLevel;
    set(handles.minText,'String', textValue);
    set(handles.minSlider,'Value',sliderValue);
    textValue = num2str(maxLevel);
    sliderValue = maxLevel;
    set(handles.maxText,'String', textValue);
    set(handles.maxSlider,'Value',sliderValue);
    adjustFrames(hObject, eventdata, handles)
elseif gca == handles.imageWindow
    [p, q] = ginput(2);
    points = [p';q'];
    if points(1,1) < 0
        points(1,1) = 0;
    end
    if points(1,1) > 640
        points(1,1) = 640;
    end

    if points(1,2) < 0
        points(1,2) = 0;
    end
    if points(1,2) > 640
        points(1,2) = 640;
    end

    if points(2,1) < 0
        points(2,1) = 0;
    end
end

```

```

    if points(2,1) > 480
        points(2,1) = 480;
    end

    if points(2,2) < 0
        points(2,2) = 0;
    end
    if points(2,2) > 480
        points(2,2) = 480;
    end
    x = (points(1,:));
    y = (points(2,:));
    length = sqrt((x(2)-x(1))^2+(y(2)-y(1))^2);
    xPoints = x;
    yPoints = y;

    imshow(image, [0 65535], 'Parent', handles.imageWindow);
    hold(handles.imageWindow, 'on');
    plot(handles.imageWindow, linspace(double(imageXLim(1))+1,
double(imageXLim(1)+1+20)), ...

linspace(double(imageYLim(1))+1, double(imageYLim(1))+1), 'Color', 'red', 'LineWi
dth', 3);
    set(handles.imageWindow, 'XLim', imageXLim, 'YLim', imageYLim);

plot(handles.imageWindow, linspace(xPoints(1), xPoints(2)), linspace(yPoints(1),
yPoints(2)), 'Color', 'green', 'LineWidth', 2);

plot(handles.imageWindow, linspace(double(imageXLim(1))+1, double(imageXLim(1)+
length)), ...

linspace(double(imageYLim(1))+1, double(imageYLim(1))+1), 'Color', 'green', 'Line
Width', 2);

text(double(imageXLim(1)), double(imageYLim(2)), strcat(num2str(length), 'px'), '
Color', 'green', ...
    'VerticalAlignment', 'Bottom', 'HorizontalAlignment', 'Left')
    hold(handles.imageWindow, 'off');
end
guidata(hObject, handles);

% --- Executes on mouse press over axes background.
function histWindow_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to histWindow (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on mouse press over figure background.
function figure1_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on mouse motion over figure - except title and menu.
function figure1_WindowButtonMotionFcn(hObject, eventdata, handles)

```



```

% hObject      handle to figure1 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% [x, y] = ginput(1);
% intensity = num2str(image(uint16(y),uint16(x)));
% location = strcat(num2str(uint16(y)),':',num2str(uint16(x)));
% set(handles.intensityText,'String',strcat(intensity,' at ',location));
global image;
global imageXLim;
global imageYLim;
global currentFrame;
pixel = uint16(get(handles.imageWindow,'CurrentPoint'));
pixel = pixel(1,(1:2));
if pixel(1) < imageXLim(1)
pixel(1) = imageXLim(1);
end
if pixel(1) > imageXLim(2)
    pixel(1) = imageXLim(2);
end
if pixel(2) < imageYLim(1)
    pixel(2) = imageYLim(1);
end
if pixel(2) > imageYLim(2)
    pixel(2) = imageYLim(2);
end
intensity = num2str(double(image(pixel(2),pixel(1))));
location =
strcat(num2str(double(pixel(1))),':',num2str(double(pixel(2))),':',num2str(double(currentFrame)));
set(handles.intensityText,'String',strcat(intensity,' at [',location,']'));

function window_ActionPreCallback(obj, evd)
global currentWindow
currentWindow = gca;

function window_ActionPostCallback(obj, evd)
global currentWindow;
global imageWindow;
global histWindow;
global imageXLim;
global imageYLim;
global histXLim;
global histYLim;
global hImageLimitTable;
global hImageWindow;
global image;
global xPoints;
global yPoints;
global length;

if currentWindow == imageWindow
    XLim = uint16(get(evd.Axes,'XLim'));
    YLim = uint16(get(evd.Axes,'YLim'));
    if XLim(1) < 1
        XLim(1) = 1;

```

```

end
if XLim(2) > 640
    XLim(2) = 640;
end
if YLim(1) < 1
    YLim(1) = 1;
end
if YLim(2) > 480
    YLim(2) = 480;
end
imageXLim = XLim;
imageYLim = YLim;
set(hImageLimitTable, 'data', [imageXLim;imageYLim]);

imshow(image, [0 65535], 'Parent', hImageWindow);
hold(hImageWindow, 'on');
plot(hImageWindow, linspace(double(imageXLim(1))+1,
double(imageXLim(1)+1+20)), ...

linspace(double(imageYLim(1))+1, double(imageYLim(1))+1), 'Color', 'red', 'LineWi
dth', 3);
set(hImageWindow, 'XLim', imageXLim, 'YLim', imageYLim);
if (~isequaln(xPoints, [0,0]) && ~isequaln(yPoints, [0,0]))

plot(hImageWindow, linspace(xPoints(1), xPoints(2)), linspace(yPoints(1), yPoints
(2)), 'Color', 'green', 'LineWidth', 2);

plot(hImageWindow, linspace(double(imageXLim(1))+1, double(imageXLim(1)+length)
), ...

linspace(double(imageYLim(1))+1, double(imageYLim(1))+1), 'Color', 'green', 'Line
Width', 2);

text(double(imageXLim(1)), double(imageYLim(2)), strcat(num2str(length), 'px'), '
Color', 'green', ...
    'VerticalAlignment', 'Bottom', 'HorizontalAlignment', 'Left')
end
hold(hImageWindow, 'off');
elseif currentWindow == histWindow
    XLim = uint16(get(evd.Axes, 'XLim'));
    YLim = uint16(get(evd.Axes, 'YLim'));
    if XLim(1) < 0
        XLim(1) = 0;
    end
    if XLim(2) > 65535
        XLim(2) = 65535;
    end
    if YLim(1) < 0
        YLim(1) = 0;
    end
    if YLim(2) > 640*480
        YLim(2) = 640*480;
    end
    histXLim = XLim;
    histYLim = YLim;
end
end

```

```

% --- Executes when entered data in editable cell(s) in imageLimitTable.
function imageLimitTable_CellEditCallback(hObject, eventdata, handles)
% hObject      handle to imageLimitTable (see GCBO)
% eventdata    structure with the following fields (see UITABLE)
%   Indices: row and column indices of the cell(s) edited
%   PreviousData: previous data for the cell(s) edited
%   EditData: string(s) entered by the user
%   NewData: EditData or its converted form set on the Data property. Empty
if Data was not changed
%   Error: error string when failed to convert EditData to appropriate value
for Data
% handles      structure with handles and user data (see GUIDATA)
global imageXLim;
global imageYLim;
newLim = get(handles.imageLimitTable, 'data');
if newLim(1,1) < 1
    newLim(1,1) = 1;
end
if newLim(1,2) > 640
    newLim(1,2) = 640;
end
if newLim(2,1) < 1
    newLim(2,1) = 1;
end
if newLim(2,2) > 480
    newLim(2,2) = 480;
end

    imageXLim = newLim(1,:);
    imageYLim = newLim(2,:);

% --- Executes on button press in histeqButton.
function histeqButton_Callback(hObject, eventdata, handles)
% hObject      handle to histeqButton (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hint: get(hObject, 'Value') returns toggle state of histeqButton
global dataStream;
global dataStreamEQ;
global avg;
global sigma;
global minValue;
global maxValue;

toggleStatus = get(handles.histeqButton, 'Value');
if toggleStatus
    for i = 1:handles.frameCount
        dataStreamEQ(:, :, i) = histeq(handles.inputData(:, :, i));
    end
    dataStream = dataStreamEQ;
    avg = uint16(mean(dataStreamEQ(:)));
    sigma = uint16(sqrt(var(double(dataStreamEQ(:)))));
    minValue = uint16(min(dataStreamEQ(:)));
    maxValue = uint16(max(dataStreamEQ(:)));

```

```

        adjustFrames(hObject, eventdata, handles);
else
    dataStream = handles.inputData;
    avg = uint16(mean(dataStream(:)));
    sigma = uint16(sqrt(var(double(dataStream(:)))));
    minValue = uint16(min(dataStream(:)));
    maxValue = uint16(max(dataStream(:)));
    adjustFrames(hObject, eventdata, handles);
end

% --- Executes on button press in clearMarks.
function clearMarks_Callback(hObject, eventdata, handles)
% hObject      handle to clearMarks (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
global xPoints;
global yPoints;
global image;
global imageXLim;
global imageYLim;

xPoints = [0,0];
yPoints = [0,0];

imshow(image, [0 65535], 'Parent', handles.imageWindow);
hold(handles.imageWindow, 'on');
plot(handles.imageWindow, linspace(double(imageXLim(1)+1),
double(imageXLim(1)+1+20)), ...

linspace(double(imageYLim(1)+1), double(imageYLim(1)+1), 'Color', 'red', 'LineWi
dth', 3);
set(handles.imageWindow, 'XLim', imageXLim, 'YLim', imageYLim);
hold(handles.imageWindow, 'off');

% --- Executes on button press in modeButton.
function modeButton_Callback(hObject, eventdata, handles)
% hObject      handle to modeButton (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
global dataStream;
global dataStreamPT;
% Hint: get(hObject, 'Value') returns toggle state of modeButton
toggleStatus = get(handles.modeButton, 'Value');
if toggleStatus
    set(handles.modeButton, 'BackgroundColor', 'red', 'String', 'Threshold
Mode');
    dataStreamPT = dataStream;
    adjustFrames(hObject, eventdata, handles);
else
    set(handles.modeButton, 'BackgroundColor', 'blue', 'String', 'Scaling Mode');
    dataStream = dataStreamPT;
    adjustFrames(hObject, eventdata, handles);
end
%% End of GUI -----%
```