

NORTHWEST NAZARENE UNIVERSITY

Fire Monitoring and Assessment Platform: Image Post-processing and Image Manipulation

THESIS

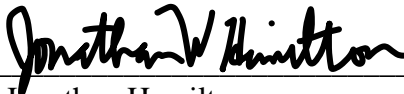
Submitted to the Department of Mathematics and Computer Science  
in partial fulfillment of the requirements  
for the degree of  
BACHELOR OF SCIENCE


Jonathan W Hamilton  
2017

THESIS  
Submitted to the Department of Mathematics and Computer Science  
in partial fulfillment of the requirements  
for the degree of  
BACHELOR OF SCIENCE

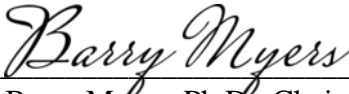
Jonathan W Hamilton  
2017

Fire Monitoring and Assessment Platform: Image Post-processing and Image Manipulation

Author:   
Jonathan Hamilton

Approved:   
Dale Hamilton, M.S., Professor of Computer Science,  
Department of Mathematics and Computer Science, Faculty Advisor

Approved:   
Brice Allen, Engineering Lab Manager,  
Department of Physics and Engineering

Approved:   
Barry Myers, Ph.D., Chair,  
Department of Mathematics and Computer Science

## Abstract

Fire Monitoring and Assessment Platform: Image Post-processing and Image Manipulation.

HAMILTON, JONATHAN (Department of Mathematics and Computer Science),  
DR. MYERS, BARRY (Department of Mathematics and Computer Science).

The Fire Monitoring and Assessment Platform (FireMAP) uses post-fire, aerial imagery to determine fire severity. Traditionally, post-fire analysis is done by on site wildland firefighters, satellites, or manned aircraft. Because traditional post-fire image acquisition is often dangerous for firefighters and too expensive and low resolution from satellites and aircraft, FireMAP plans to use drones for safer and higher resolution post-fire image acquisition. The purpose of this section of the FireMAP project is to transform classified imagery into a form more usable to end users. By the time the aerial imagery has reached this section of the project, each pixel from a post-fire image has been placed into a raster and classified as white ash, black ash, dirt, surface vegetation or canopy vegetation. Within the classified image, the burn area boundaries contain sub-object in size spatio-spectral clusters resulting in unclear and incorrectly classified pixels leaving a salt-and-peppered effect across the image. The open source program OpenCV's open and close morphological functions fix these problems by smoothing burn area boundaries making them clearly defined. Because high severity burns leave areas of white ash smaller than the burnt vegetation, the same morphological functions dilate the high severity burn areas and delete misclassified burn areas.

## Acknowledgments

First of all, I would like to thank the external individuals and organizations that made the FireMAP project possible. The funding the project received from NASA, the Idaho Space Grant Consortium, and the IDeA Network of Biomedical Research Excellence made it financially possible. Their contribution is deeply appreciated. I would like to thank the United States Forest Service (USFS) and United States Department of the Interior (DOI) and other agencies that are interested in the FireMAP project. Their interest in the FireMAP product gave the project direction. Without them, there would be no reason to make the project. All the individual employees of the USFS and DOI who showed interest in the FireMAP project are particularly appreciated. They spent their time and resources to ensure the FireMAP project fit their needs and was the best it could be.

I would also like to thank all those within FireMAP that did quality work for me to build on. Mikhail Bowerman, Joshua Gambill, Nicholas Hamilton, Llewellyn Johnston, Glen Lungen, Patrick Richardson, and Greg Smith all worked on sections of FireMAP that this section heavily relied on, so their good work is greatly appreciated. Along with mentors Dale Hamilton and Dr. Barry Myers, Llewellyn and Patrick often gave help on this section of the project and were exceptional resources.

## Table of Contents

Title Page	i
Signature Page	ii
Abstract	iii
Acknowledgments	iv
Table of Figures and Tables	vi
Fire Monitoring and Assessment Platform	1
Morphology	4
Inputting and Interpreting Classified Imagery	7
Removing Salt-and-Pepper and Smoothing Boundaries	10
Connected Components	12
High Severity Burn Areas	15
Conclusion	17
Future Work	18
Works Cited	24
<b>Appendix A: Source Code</b>	<b>20</b>

## Table of Figures

Figure 1: Morphological Erosion.....	6
Figure 2: Morphological Dilation .....	6
Figure 3: Morphological Open and Close Operations .....	7
Figure 4: Classified Inputted Image.....	10
Figure 5: Classified Image to Burn Area Image .....	12
Figure 6: Smoothing Burn Area.....	14
Figure 7: White Ash Tendrils .....	19
Figure 8: White Ash and Burn Area Overlay .....	20

## Table of Tables

<i>Table 1: Input Raster Classification</i> .....	3
Table 2: Classified to Burn Area Conversion .....	11

## Fire Monitoring and Assessment Platform

This project is part of the larger Fire Monitoring and Assessment Platform (FireMAP) project. Traditionally, assessment of post-fire effects is done on-site by wildland firefighters, satellites, or manned aircraft. Because of the instability of a forest after a fire, this task is often highly dangerous for wildland firefighters. To adequately assess a burn and its boundaries, they would have to walk among trees that could fall or reignite, endangering firefighter safety. Additionally, wildland firefighters often have difficulty accurately representing and finding all burn boundaries. Insignificantly small non-burned areas with the burn boundary are especially difficult to represent. Wildland firefighters monitoring and assessing post-fire conditions is dangerous and often inaccurate.

Alternatively, satellites or aircraft could be used. Attaining aerial imagery provides a more spatially complete representation of the burn extent and severity than ground assessments done by wildland firefighters; however, it does come with its disadvantages. Satellites and aircraft are often expensive to operate and replace. Satellites have to be put into orbit and aircraft have to be fueled, neither of which are cheap. If either crash due to leaving orbit or some unforeseen flight error, an expensive replacement will be needed.

Besides being costly to operate, satellites are not reliable to attain timely data. Satellites orbit the Earth and can only see the area directly below them, so to use imagery taken from a satellite, it would first need to continue its orbit until it is in a location where it is over the fire. Landsat, the satellite often used to attain satellite wildland fire data,

takes approximately sixteen days to orbit the earth which is too slow of a turnaround because a lot happens in the first couple of days in a burn area following a fire and that data would be lost while waiting for the satellite to get into position. The time to get satellite imagery could be even longer if cloud cover or smoke obscures the burn area.

Satellites are so high in orbit that they have a very low resolution of around 30 square meters a pixel. While this resolution is sufficient for multi-thousand-acre fires, the majority of fires are small enough that they would only be a few pixels (Hamilton, 2015).

Because aircraft operation is expensive and satellites are often unreliable for smaller burn areas, FireMAP plans to use drones to attain aerial imagery instead of using traditional fire monitoring and assessment techniques. Using drones is safer than using firefighters because pilots do not need to walk in the burn areas and can instead fly the drone from a safer location outside the perimeter of the burned area. If a drone crashes, they are comparatively cheap and easily replaced.

Drones have a high spatial resolution compared to satellites. While the spatial resolution of imagery taken from a drone is dependent on the its height above ground level when the imagery is taken, even at the maximum legal flying height for drones of 300 meters produces higher spatial resolution imagery than satellites. Flying at 120 meters (400 feet), the spatial resolution of the imagery is approximately six centimeters, increasing or decreasing depending on the altitude. The spatial resolution of drone imagery is so high that it might contain more data than is necessary, so it can be resampled to a lower resolution so that unneeded burn data is not stored and analyzed.

FireMAP takes the post-fire imagery taken from a Phantom 3 drone, combines



them into an orthomosaic and processes the data in the imagery to make useful post-burn information. The area of a single image from a drone flight (approximately 72 hectares at a height of 120 meters) may not be large enough to contain the entirety of a fire, so multiple images need to be captured to contain the entire burn area. FireMAP uses the photogrammetry orthomosaic software Pix4D<sup>1</sup> to stitch together the image segments to create a complete orthomosaic of the burn area. Pix4D autonomously flies a user specified path over a post-burn area taking frequent enough images that they overlap. A GPS latitude and longitude coordinate is assigned to each image which Pix4D uses to stitch together the imagery. When Pix4D is finished, there is one image containing the whole of the burn area.

After Pix4D makes an orthomosaic of the burn area, the orthomosaic is classified into one of seven classes: black ash, white ash, surface vegetation (herbaceous or shrub), canopy vegetation (conifer or deciduous) and bare soil (or rock). The classes and their equivalent values are shown in Table 1.

*Table 1: Input Raster Classification*

Class Value	Class Name
0	White Ash
1	Black Ash
2	Herbaceous
3	Shrub
4	Conifer Trees
5	Deciduous Trees
6	Dirt/Soil

---

<sup>1</sup> Pix4D is documented and downloadable at [pix4d.com](http://pix4d.com)

Each of the classes has different attributes assigned to them for assessing fire severity. Black ash and white ash depict fire severity where black ash is low severity and white ash is high severity (Lentile, 2006). Surface vegetation, canopy vegetation, and bare soil all represent non-burn areas in different ways. Dirt and surface vegetation are similar and are only differentiated to make classification easier. Canopy vegetation is different because canopy cover in aerial imagery obscures the ground under it forcing assumptions to be made about the burn severity under the canopy. If more information on the image classification is wanted, the other FireMAP projects are good resources on Class Separability<sup>2</sup>, Object Identification using Clustering<sup>3</sup>, and Object Identification using Machine Learning Algorithms<sup>4</sup>.

---

<sup>2</sup> Bowerman, M. Data Collection, Analysis, and Class Separability

<sup>3</sup> Richardson, P. J. Object Identification in Imagery Using Cluster Analysis

<sup>4</sup> Johnston, L. B. Detecting Burn Severity in Post Wildland Fire Imagery through k-Dimensional Trees and k-Nearest Neighbours Machine Learning Algorithms

## Morphology

This section of the FireMAP project heavily relies on morphological image processing functions. Since morphology is most commonly related to biology an understanding of morphology as it applies to the FireMAP project is needed. For this project, morphology can be described as mathematics for binary imagery.

If addition and subtraction are the most basic mathematical functions, then erosion and dilation are the corresponding most basic morphological functions. Both erosion and dilation are binary morphological functions manipulating images with pixel values of either 255 (positive value) or 0 (negative value). Erosion of set A by set B is “a set of all points x such that B translated by x is still contained in A” (Morphological Operations: An Overview, 1996). A pictorial representation of the erosion process is shown in Figure 1. Set B, often called the structuring element, has an origin point depicted by a black dot in Figure 1. Set B’s origin is compared to each individual pixel in set A. If not all the overlapping pixels in set A where set B is currently checking are positive, then the pixel in set A where set B’s origin is currently located is made negative. Erosion has a shrinking effect on the boundaries of objects in a binary image making the objects have smaller areas.

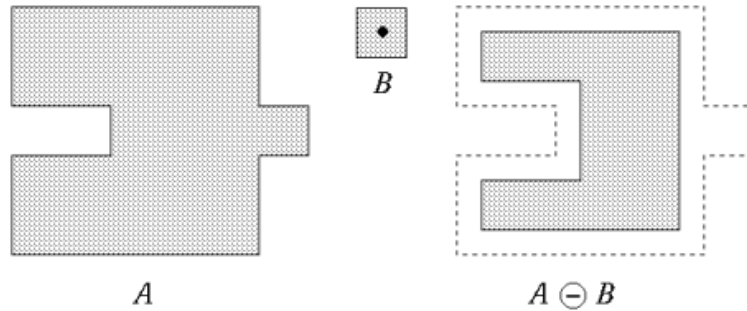


Figure 1: Morphological Erosion<sup>5</sup>

Morphological dilation is like erosion but expands instead shrinks boundaries of objects in a binary image. A pictorial representation of dilation is shown in Figure 2. Set B's origin is compared to each individual pixel in set A. If there are any overlapping positive pixels in A where B is currently checking, then the pixel in set A where set B's origin is currently located is made positive.

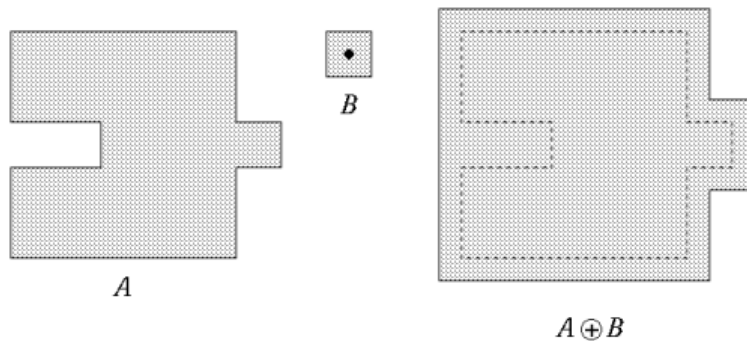


Figure 2: Morphological Dilation<sup>6</sup>

This project primarily uses open and close morphological operations which are

<sup>5</sup> (Morphological Operations: An Overview, 1996)

<sup>6</sup> (Morphological Operations: An Overview, 1996)

combinations of erosion and dilation. An open morphological operation of set A by set B first erodes then dilates set A by set B. (Morphological Operations: An Overview, 1996) Opening gets rid of small islands and extrusions of positive pixels. A close morphological operation of set A by set B first dilates then erodes set A by set B (Morphological Operations: An Overview, 1996). A closing gets rid of small holes and extrusions of negative pixels. A pictorial representation of the open and close morphological operations are shown in Figure 3.

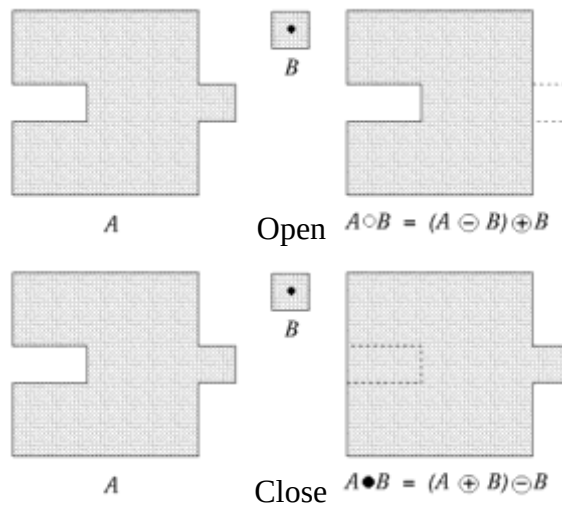


Figure 3: Morphological Open and Close Operations<sup>7</sup>

Some other morphological functions were considered for use in this project, but the better improved results would not have been enough to justify the added complexity and run time. The implementation of a thickening morphological function would create a more accurate representation of small islands of high burn severity, but a simpler dilate

<sup>7</sup> (Morphological Operations: An Overview, 1996)

morphological function is used in the final product. The thickening morphological operation is still in the code, but is incomplete because it was not used in the final product. It is more important to understand that the functions exist and that they were considered than how they work.

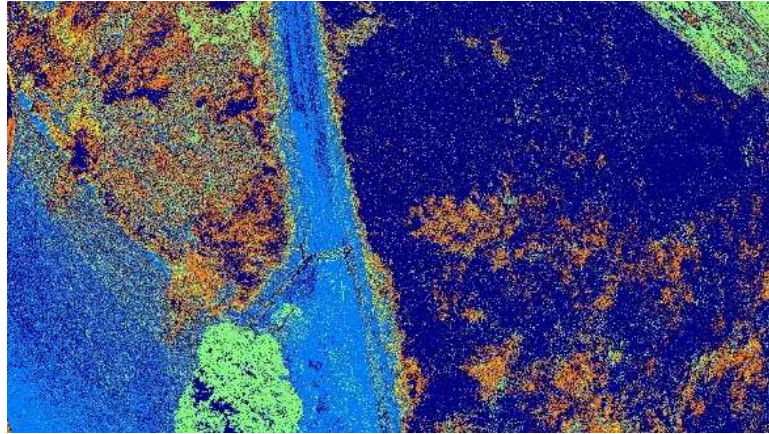
## Inputting and Interpreting Classified Imagery

By the time the post-burn imagery gets to this section of the project, it has already been classified and stored in raster format. The imagery would have been passed to this section of the project in one of the usual image formats like JPG, but those formats distort the imagery for a smaller storage size which in this case would result in the loss of data. To avoid data loss, a raster was used instead of traditional image formats.

A raster is a two-dimensional matrix where each cell holds a numerical value related to the pixel the raster represents. The values in the raster ranged between one and seven representing the pixel's classification. Rasters are stored in a text format instead of an image, so before morphological operations could be done on the imagery, the raster had to be converted back into an image.

The open source OpenCV library was used to store the post-fire burn classification data in RAM. OpenCV's Mat variable is a grayscale image which means that instead of storing red, green, and blue values for each pixel as is done in a color image, one value is stored where a value of 0 is black, a value of 255 is white, and the values in-between are shades of grey scaling from black to white. The Mat variable is like other variables, except that it is stored on RAM instead of on disk allowing faster access for manipulation. Additionally, MAT is like other image types, like JPEG, but it does not compress the imagery data resulting in data corruption. The program converted the numerical values in the raster to pixel values in the Mat cell by cell. The results of this process are shown in Figure 4. While the Mat is a grayscale image and could still be

shown as a black-and-white image, a color map was applied to make the classes more distinct and make visualizing the data easier. Each grayscale value in the Mat was given a color representation. For example, the dark blue is low severity burn, the light blue is dirt, the orange is brush, and the light green is conifer canopy.



*Figure 4: Classified Inputted Image*

Besides the two-dimensional matrix in the raster, there is also metadata giving useful information about the data. While a wide range of metadata was brought in with the metadata, the only ones inputted to this project was the column and row dimensions of the matrix.

While the data is useful, there is too much information. While classes are all useful for processing the data throughout the FireMAP project, the current user requirements care more about where and how severely it burned than all the information currently included, so post-processing makes the information better match user requirements. This process is done by making a burn area image, a high-severity area



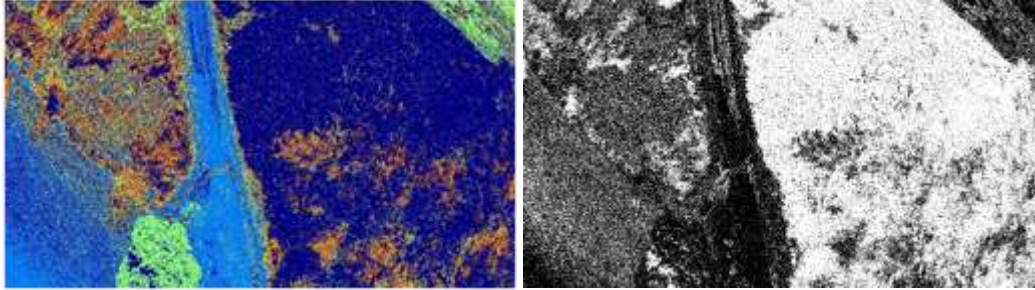
image, and combining the two images to make the final result. Finding high severity areas will be talked about in a later section.

The burn area image is made from the inputted classified imagery. If a cell is classified as high or low severity, then the equivalent burn area image cell is positive. If a cell is any other of the classified values, then the equivalent burn area image cell is negative. Table 2 shows what each of the classes in the classified imagery converts to in the burn area imagery.

*Table 2: Classified to Burn Area Conversion*

Class Value	Class Name	Burn Value	Burn Name
0	White Ash	1	Unburned
1	Black Ash	1	Unburned
2	Herbaceous	0	Burned
3	Shrub	0	Burned
4	Conifer Trees	0	Burned
5	Deciduous Trees	0	Burned
6	Dirt/Soil	0	Burned

While there are eight classes in the classified imagery, they are all combined into two separate classes when converted, so the burn area imagery contains two classes: burned and unburned. A pictorial representation of the conversion from eight classes to two classes is shown in Figure 5. Again, a color map was applied to the eight-class grayscale image to give it the appearance of color for



*Figure 5: Classified Image to Burn Area Image*

In the classified image, the dark blue areas are white ash and are put in the burned class in the burn area imagery. While the white ash is too small to be seen in the classified imagery at this point, it is also put in the burned class in the burn area imagery.

## Removing Salt-and-Pepper and Smoothing Boundaries

Even with how accurately the data classification is, classification errors and real-world problems like shadows result in misclassified clusters of pixels belonging to the same class smaller than the typical object size in that class. The misclassified clusters create a salt-and-pepper effect on the image. The largest source for this problem is because the color of black ash and shadow are so similar, the classification section of the FireMAP project has difficulty distinguishing them. While the problem is in the classification section of FireMAP, its effects can be significantly reduced in this section of FireMAP using morphological operations.

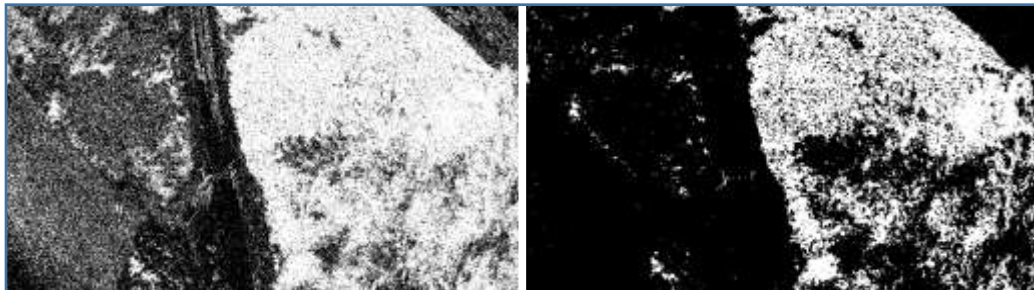
A morphological open and close are used to mitigate the salt-and-pepper effect. When combined, these operations get rid of both positive and negative islands and extrusions. As mentioned in the section on morphology, a set B acts on set A. In this case, set A is the burned class and set B is a structuring element with the origin at the center. The shape of the structuring element had little effect on the burned class, so a circular one was used. The big question that had to be answered was what radius the structuring element should be.

If removing incorrectly classified islands of data and smoothing the boundaries were the only relevant parameters in determining the structuring element's radius, then a larger radius would be better. With a larger structuring element, boundaries would be smoother and larger islands of salt-and-pepper would be taken care of; however, the radius couldn't be too large because the burn areas would be transformed into blobs

which would inaccurately represent the burn area boundaries. Even with the quality of a higher structuring element radius, the run time of a larger radius makes large radii impractical.

The structuring element's radius could not be too big because the run time increases exponentially as the radius increases. Because timely output matters to this project, slightly improved accuracy of burn area boundaries does not justify drastically increased run times. This section of the project's runtime would be a couple of seconds with a radius of a couple of pixels and would rapidly increase to minutes as the radius increased. The largest radius checked was a radius of 9 which took tens of minutes. To get the best results, the radius must be large enough that the salt-and-pepper is removed and small enough that it does not take an unacceptable amount of time to operate.

It was found that a radius anywhere between two and five pixels produced adequate results. Currently, this section operates using a structuring element with a radius of four, but that number could easily be changed if the end user wanted. Figure 6 shows the result of applying the open and closed operations on the burn area imagery where the left image is before the operations and the right image is after.



*Figure 6: Smoothing Burn Area*

The open and close operations, using a structuring element with a radius of four, removes most of the salt-and-pepper and smooths the burn area boundaries. There are still some islands of positive and negative data that were too large for the open and close to catch, but as stated before, these islands are acceptable at this point of the project so as not to make too long of a run time. A function was made to take care of the remaining islands of false data using the *Connected Components* functionality in the OpenCV library.

## Connected Components

The purpose of this section of the program is to remove islands of data too large for an open and close operation to adequately get rid of. Upon completion of this project, this section did not meet its desired functionality. Some problems were encountered using OpenCV's *Connected Components with Stats* function. Even with this section not working, the final result still meets the project requirements, but there are still insignificantly small clusters of data that are sub-object in size that would be removed if this section of the project functioned properly.

Connected components can be described as contiguous pixels in the same class being spatially clustered into groups. OpenCV's *Connected Components* takes a binary image – like the smoothed burn area one made previously – and splits all the positive pixels into connected components where each component is given a numeric value. Negative, non-burned pixels are all part of the first connected component, and connected components are added as they are found.

There are two uses for this section of the project: delete false clusters of data that a morphological open and close operation could not get rid of and delete sub-object in size clusters of correctly classified burned or unburned. If there was an area that was completely burned except for a small patch of dirt, then the entire area would be classified as burned including the small patch of dirt. Likewise, if an area was completely unburnt except for a small ember start outside the burn boundary, then the entire area would be considered unburned. The spatial extent of a burned island considered to be

insignificant is different than that of an insignificantly small non-burned area. This section of the project takes the difference into account and contains two changeable values for the minimum spatial extent for burned and non-burned islands.

Theoretically, that is how this section would have worked, but there were some errors that were not fixed by the completion of the project. One potential source of the problem is while *Connected Components* is an established function in the OpenCV library, *Connected Components with Stats* was released in the newest update to the library. While new functionality in a purchased product usually works, OpenCV is an open-source product and is more likely to have bugs in its newer. Further work will be done to use *Connected Components with Stats* in this section of the project, but due to its newness, it will not be assumed functional.

The connected components section of this project removes small positive clusters of data but has not yet been implemented to remove negative clusters. Implementation of this functionality would not be difficult to achieve. The image can be morphologically inverted then the same code that deletes small positive islands of data can also delete small negative islands of data. After deleting small islands of negative data, the image would be re-inverted. This had not yet been implemented because there was no need for added complexity debugging as the original process to remove small positive islands was not working properly.

## High Severity Burn Areas

As previously stated, the one of the desired results of this project is the production of an image containing the high-severity burn area. High severity burn areas are identified by white ash, so a binary image can be made using the classified imagery where white ash is a positive value (white) and anything else is a negative value (black). Also, high severity areas only occur within low severity areas, so any white ash outside the burn area is assumed to be false and is marked as negative in the white ash imagery. After the white ash is found, these areas need to be dilated out to match the actual spatial extent of the high-severity burn.

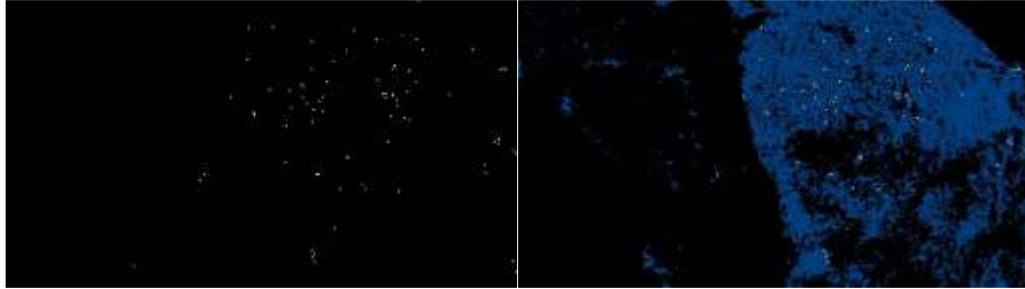
It is reasonable to assume that a white ash region would accurately depict the spatial extent of a high severity burn area, but it does not. For example, as a campfire burns down to white ash, the burned material takes up less volume than it did before it was burned. As shown in Figure 7, when a bush completely burns, all that is left is white tendrils branching out from the center of the bush, the remainder of completely combusted larger branches. Because the white ash left behind is smaller than the vegetation's original spatial extent, it must be dilated out to the actual high severity burn spatial extent.





*Figure 7: White Ash Tendrils*

Like with the other classes in the classification, there are falsely classified white ash pixels in the classified image. Like how salt-and-pepper is removed in the burn extent imagery, small clusters of high severity burn areas are removed using an erode morphological function. The remaining white ash is dilated to four times the original area. The high severity imagery is overlaid onto the burn area imagery. As before, high severity areas can only occur within the burn area bounds, so positives in the high severity imagery are marked as non-burned in the overlaid image. The final result is shown in Figure 8 where the left image is high severity burn extent and the right image is the overlaid image. The actual overlay image is grayscale. A color map was applied to it for easier visualization.



*Figure 8: White Ash and Burn Area Overlay*

Due to the lack of specifications by the end user requirements, both the dilation radius and the dilation radius were decided on based off of what looked best to the creator of the project. Alterations to the erosion and dilation are easily made within the code. Currently, the white ash is dilated with a structuring element with a radius of two, but should the user requirements change, the radius could easily be changed. The program was designed to always dilate the white ash in proportion to the erosion amount.

## Conclusion

This project has been successful in meeting many of its goals. Most notably, it reads in an image raster and converts it to an image containing classification information, makes a high severity map by eroding then dilating the white ash areas within the burn area, overlays high severity burn areas onto the burn area imagery creating a final product that adequately represents where the fire burned and with what severity. Several sections of the project, though meeting the goals of the project, can be further enhanced to produce higher quality results and it is recommended that further work be done in these areas. Additional focus should be placed on more accurately interpreting the classified image to find burned areas and removing salt-and-pepper regions of images. Finally, as mentioned before, successfully implementing *Connected Components with Stats* would remove the sub-object sized positive and negative areas that are too small for the open and close morphological operations to catch. Correctly implementing *Connected Components with Stats* remains the portion of this project that will require the most effort to include.

## Future Work

The *Connected Components with Stats* section of the project could use the most work. The code is close to completion; it just needs some debugging to fix it. The source of the problem has been narrowed down, and with some effort, connected components could become operational.

Identifying burn areas can be improved by taking into account canopy cover. As mentioned previously, a fire can burn under a canopy, so the project could be improved by determining if areas under canopies are burned or not based on if the area around the canopy is burned or not. Because the aerial imagery's view of under the canopy is obscured, there is no way to be sure if it is burned or not, but an educated guess can be made by observing the area around the canopy. If all the area around the tree is burned, it is likely also burned under the canopy. Likewise, if all the area around the tree is not burned, it is likely not burned under the canopy. It would be tricky to determine the burn area under a canopy when there is a combination of burned and not burned around the canopy, but a solution to that problem can be found upon further research. The additional functionality would not be guaranteed to be completely accurate, but it would be more accurate than classifying all canopies as unburned which is what is currently done.

One last area of improvement is adjusting the resolution to better fit end user's needs. True, traditional methods of acquiring fire data are not high enough resolution to get to as much information as drones, but using drones to attain the imagery likely creates information with too high a resolution to be useful. The images that the FireMAP team is working with are massive even for small fires where an orthomosaic of a 40 hectare (100

acre) scene flown at 60 meters containing approximately two billion pixels. If potential end users want all the collected information and are willing to deal with the massive storage requirement, there will be no problem, but they have requested that the project have functionality implemented that automatically or manually adjusts the resolution of the imagery to a resolution that is more manageable for the end user's needs.

Because the end user has never had the problem of too much information before, they do not know what resolution would be best, but further research could be done to choose a value or enable the end user to adjust the resolution to what they want for each individual case.

## Works Cited

- Bowerman, M. (2017). Data Collection, Analysis, and Class Separability.
- Gonzalez, R. C., & Woods, R. E. (2008). *Digital Image Processing* (Third ed.). Upper Saddle River, NJ: Paerson Education, Inc.
- Johnston, L. B. (2017). Detecting Burn Severity in Post Wildland Fire Imagery through k-Dimensional Trees and k-Nearest Neighbours Machine Learning Algorithms.
- Morphological Operations: An Overview*. (1996, July 17). Retrieved March 30, 2017, from <http://www.inf.u-szeged.hu/ssip/1996/morpho/morphology.html>
- OpenCV*. (2017, April 20). Retrieved from <http://opencv.org/>
- Pix4D*, 3.2. (n.d.). Retrieved from <https://pix4d.com/>
- Richardson, P. J. (2017). Object Identification in Imagery Using Cluster Analysis.

## Appendix A: Source Code

### A.1 Source.cpp

```
//Denoising_v1.0.cpp
// Jon Hamilton

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/photo.hpp"
#include "ImgManipulate.h"
#include <stdio.h>
#include <iostream>           //std::ifstream
#include <fstream>
#include <ctime>

cv::Mat open(cv::Mat source, cv::Mat structElem);
cv::Mat close(cv::Mat source, cv::Mat structElem);

int main(int, char** argv)
{
    //Variables
    clock_t start;
    const int size = 3, //Erode + Dilate Amount
             posMin = 5000,
             negMin = 2000;

    cv::Mat src, burnArea, dst, disp,
            element = getStructuringElement(
                cv::MORPH_ELLIPSE,
                cv::Size(2 * size + 1, 2 * size + 1),
                cv::Point(size, size));
    start = clock();

    /// Load an image
    std::ifstream imgASCII("../data/LowellDamASCII.txt");
    cv::String temp;
    std::string dummy;
    int rows, cols;

    // Read in Metadata
    imgASCII >> dummy;
    imgASCII >> cols; // NCols
    imgASCII >> dummy;
    imgASCII >> rows; // NRows
    imgASCII >> dummy;
    imgASCII >> dummy; // XLLCorner
    imgASCII >> dummy;
    imgASCII >> dummy; // YLLCorner
    imgASCII >> dummy;
    imgASCII >> dummy; // CellSize
    imgASCII >> dummy;
```

```

imgASCII >> dummy; // NoData_Value

// Convert roster to MAT
src.create(rows, cols, CV_8U);
for (int x = 0; x < rows; x++) {
    for (int y = 0; y < cols; y++) {
        int group;
        imgASCII >> group;
        src.at<uchar>(x, y) = group * 32;
    }
}

// Convert to output roster format
cv::applyColorMap(src, disp, cv::COLORMAP_JET);
imwrite("../output/0 Original.jpg", disp);

// Create binary image
burnArea = binary(src);
cv::applyColorMap(burnArea, disp, cv::COLORMAP_OCEAN);
imwrite("../output/1 Burn Area.jpg", disp);

// Smooth binary image
burnArea = open(burnArea, element);
burnArea = close(burnArea, element);
cv::applyColorMap(burnArea, disp, cv::COLORMAP_OCEAN);
imwrite("../output/2 Smooth Burn Area.jpg", disp);

// Remove small islands of data
//deleteIslands(burnArea, burnArea, posMin, negMin);
//cv::applyColorMap(burnArea, disp, cv::COLORMAP_OCEAN);
//imwrite("../output/4 Islandless Burn Area.jpg", burnArea);

// Add white ash to output
cv::Mat highSeverity = whiteAsh(src, burnArea);
dst = combine(burnArea, highSeverity);

// Output Processed Image
cv::applyColorMap(dst, disp, cv::COLORMAP_OCEAN);
imwrite("../output/5 Output.jpg", disp);
return 0;
}

//*****
/** Functions

cv::Mat open(cv::Mat img, cv::Mat structElem) {
    erode(img, img, structElem);
    dilate(img, img, structElem);
    return img;
}
cv::Mat close(cv::Mat img, cv::Mat structElem) {
    dilate(img, img, structElem);
    erode(img, img, structElem);
    return img;
}

```



## A.2 ImgManipulate.h

```
#pragma once
#include "opencv2/highgui/highgui.hpp"

//void thicken(cv::Mat src, cv::Mat& dst);
//void hitmiss(cv::Mat src, // Source image, 8 bit single-channel matrix
// cv::Mat& dst, // Destination image
// cv::Mat& kernel); // Kernel. 1=foreground, -1=background, 0=don't
care
cv::Mat binary(cv::Mat src);
void deleteIslands(cv::Mat src, cv::Mat& dst, int posMin, int negMin);
cv::Mat whiteAsh(cv::Mat src, cv::Mat boundary);
cv::Mat combine(cv::Mat boundary, cv::Mat white);
bool matIsEqual(cv::Mat img1, cv::Mat img2);
```

### A.3 imgManipulate.cpp

```
#include "ImgManipulate.h"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/photo.hpp"
#include <iostream>

/*
// Applies thickening morphological operation on src
// **Contemplated using this instead of the open and close operation
// **But decided not to because the added complecity was not worth the slightly
better output
void thicken(cv::Mat src, cv::Mat& dst)
{
    const int size = 1;
    cv::Mat temp,
        elm = getStructuringElement(
            cv::MORPH_RECT,
            cv::Size(2 * size + 1, 2 * size + 1),
            cv::Point(size, size));
    cv::bitwise_not(src, dst);          // Take complement of src

    //Set beginning shape of the structuring element

    // Thin the complement of src
    for (int sameCount = 0; sameCount <= 8; sameCount++) {
        hitmiss(dst, temp, elm);
        if (!matIsEqual(dst, temp))
            sameCount = 0;
        dst = temp;

        // Rotate structuring element 45 degrees
        cv::Point2f elm_center(elm.cols / 2.0F, elm.rows / 2.0F);
        cv::Mat rot_mat = cv::getRotationMatrix2D(elm_center, 45, 1.0);
        cv::Mat dst;
        cv::warpAffine(elm, dst, rot_mat, elm.size());
    }
    cv::bitwise_not(dst, dst);        // Complement thinned image

    // Post processing to remove outlier pixels
}
*/

/*
// Hit-or-miss transform function
// I didn't write this function. I found it online.
void hitmiss(cv::Mat src, cv::Mat& dst, cv::Mat& kernel) {
    CV_Assert(src.type() == CV_8U && src.channels() == 1);

    cv::Mat k1 = (kernel == 1) / 255;
    cv::Mat k2 = (kernel == -1) / 255;

    cv::normalize(src, src, 0, 1, cv::NORM_MINMAX);
}
```

```

        cv::Mat e1, e2;
        cv::erode(src, e1, k1);
        cv::erode(1 - src, e2, k2);

        dst = e1 & e2;
    }
    */

// Turn image into a binary image
// 1-Burned, 0-Unburned
cv::Mat binary(cv::Mat src) {
    cv::Mat dst;
    dst.create(src.rows, src.cols, CV_8U);
    for (int x = 0; x < src.rows; x++) {
        for (int y = 0; y < src.cols; y++) {
            uint8_t assignment = src.at<uchar>(x,y);
            if (assignment <= 32) // Values below 32 are white ash
and black ash
                dst.at<uint8_t>(x, y) = 255;
            else
                dst.at<uint8_t>(x, y) = 0;
        }
    }
    return dst;
}

/**Did not get working. Error within ConnectedComponentsStats()
// Gets rid of islands of data that are
// smaller than (int) size and greyscale shade (uint8_t) color
void deleteIslands(cv::Mat src, cv::Mat& dst, int posMin, int negMin) {
    int size;
    cv::Mat stats, components, centroid,
        disp;

    components.create(src.rows, src.cols, CV_8U);
    dst = src;

    /*******
    /** Get rid of islands of data smaller than posMin
    /*******
    /*******
    // dst:      Input image
    // components: Output image
    // stats: Statistics array for each component
    // centroid: (x,y) coordinate array for each component
    // connection: 8 or 4 for 8-way or 4-way connectivity
    const int connection = 4;
    size = connectedComponentsWithStats(dst, components, stats, centroid,
connection);

    //cv::applyColorMap(components, disp, cv::COLORMAP_JET);
    imwrite("../output/3 Burn Area Components.jpg", disp);

    std::cout << components.cols << ", " << components.rows << std::endl;

```

```

// Cycle through all positive connected components and check size
for (int label = 0; label < size; label++) {

    // Find if the connected component is too small
    if (stats.at<uint8_t>(label, cv::CC_STAT_AREA) < posMin) {
        int top = stats.at<uint8_t>(label, cv::CC_STAT_TOP),
            height = stats.at<uint8_t>(label, cv::CC_STAT_HEIGHT),
            left = stats.at<uint8_t>(label, cv::CC_STAT_LEFT),
            width = stats.at<uint8_t>(label, cv::CC_STAT_WIDTH),
            area = stats.at<uint8_t>(label, cv::CC_STAT_AREA);

        std::cout << "Component " << label << ": " << std::endl
            << "\tOrigin: " << left << ", " << top << std::endl
            << "\tBounds: " << width << ", " << height << std::endl
            << "\tArea: " << area << std::endl;

        // Look at area within bounds of component[label]
        for (int y = top; y <= top + height; y++)
        {
            for (int x = left; x <= left + width; x++)
            {
                //std::cout << (x - left) << ", " << (y - top)
                // Change pixels part of connected
                int grouping = components.at<uint8_t>(x, y);
                if (grouping == label)
                    dst.at<uint8_t>(x, y) = 0;
            }// for(x)
            if (y == top + height)
                system("PAUSE");
        }// for (y)
        std::cout << std::endl;
    }// if(small)
} // for(label)

}

// Extracts the white ash out of an image
// and dilates it to match the actual area of the burned material
cv::Mat whiteAsh(cv::Mat src, cv::Mat boundary) {
    cv::Mat dst;
    dst.create(src.rows, src.cols, CV_8U);
    for (int x = 0; x < src.rows; x++) {
        for (int y = 0; y < src.cols; y++) {
            uint8_t pixel = src.at<uchar>(x, y);
            uint8_t burned = boundary.at<uchar>(x, y);
            // If the pixel is white ash in the source image
            // and within the burned area bounds,...
            if (pixel == 64 && burned == 255)
                dst.at<uint8_t>(x, y) = (uint8_t) 255;
            else
                dst.at<uint8_t>(x, y) = (uint8_t) 0;
        }
    }
}

```

```

    }
}
cv::imwrite("../output/3 White Ash.jpg", dst);

// Adjust variable "size" to change dilation amount
int size = 2; // White ash dilation amount
cv::Mat element = cv::getStructuringElement(
    cv::MORPH_ELLIPSE,
    cv::Size(size, size));
cv::erode(dst, dst, element); // Get rid of too small white ash

// Dilate remaining white ash to four times the original size
size *= 8;
element = cv::getStructuringElement(
    cv::MORPH_ELLIPSE,
    cv::Size(size, size));
cv::dilate(dst, dst, element);

cv::imwrite("../output/4 White Ash Smoothed.jpg", dst);
return dst;
}

// Combines the mid and high-severity burn area maps
// high severity only recorded within the mid severity bounds
cv::Mat combine(cv::Mat boundary, cv::Mat white)
{
    cv::Mat dst;
    dst.create(boundary.rows, boundary.cols, CV_8U);
    for (int x = 0; x < boundary.rows; x++) {
        for (int y = 0; y < boundary.cols; y++) {
            uint8_t burned = boundary.at<uchar>(x, y);
            if (burned == 0)
                dst.at<uint8_t>(x, y) = 0;
            else {
                uint8_t severity = white.at<uchar>(x, y);
                if (severity == 0)
                    dst.at<uint8_t>(x, y) = 127;
                else
                    dst.at<uint8_t>(x, y) = 255;
            }
        }
    }
    return dst;
}

// Compares two Mats to see if they are the same images
// Assumes the Mats are the same size and not empty
bool matIsEqual(cv::Mat img1, cv::Mat img2) {
    cv::Mat Mdiff;
    cv::compare(img1, img2, Mdiff, cv::CMP_NE);
    int idiff = cv::countNonZero(Mdiff);
    return idiff == 0;
}

```