UNIVERSITY OF MASSACHUSETTS, AMHERST

BACHELORS THESIS

Nuclear Test Film Analysis

Author:
Jack Moody

Supervisor:
Dr. Rory Miskimen

A thesis submitted in fulfillment of the requirements
for the degree of Bachelors of Science in Physics

in the

Research Group Name
Department of Physics

March 22, 2021

Study hard what interests you the most in the most undisciplined, irreverent and original manner possible.

Richard P. Feynman

Abstract

Jack Moody

Nuclear Test Film Analysis

Nuclear ejecta are the earth particulates launched into the air by the cratering of a nuclear detonation that eventually hit the ground. This phenomenon within nuclear weapons effects has eluded detailed description for decades due to limitations of understanding compacted inhomogeneous material fracturing. Understanding the evolution of nuclear ejecta as a function of angle, size, and velocity over time is critical to the advancement of nuclear weapons effects research. With the recent digitization of nuclear testing films, scientists are investigating if image analysis techniques can be applied to these films to characterize the time evolution of three coupled variables that were previously thought to be unquantifiable. This investigation employs image analysis methods such as RGB color separation, Otsu and Yen thresholding, and information entropy calculations to enhance the images and isolate the ejecta. We expect to see the ejecta follow a ballistic trajectory with drag and erosive surface shedding affecting the velocity.

THIS PAGE IS INTENTIONALLY LEFT BLANK

Acknowledgements

I thank Dr. David Bacon, Dr. John Cockayne, Dr. Sarma Ananthakrisna, Dr. Joanna Ingraham, MAJ Nathaniel Kaminski, COL Chad Schools, Dr. Rory Miski-men, Dr. Andrea Pocar, and Dr. Scott Auerbach for their training and guidance throughout this process and my overall time as an undergraduate. I also must thank the UMass iCons Program, the UMass Physics Department, along with UMass Army ROTC for their support and mentorship.

THIS PAGE IS INTENTIONALLY LEFT BLANK

Contents

Chapter 1

Introduction

We currently live in an age where massive ordnance air blast (MOAB) and nuclear weapons exist. Due to this, it is possible that colossal craters could be created by the detonation. This cratering process can create an immense amount of ejecta that can cause devastating effects. Ejecta are an important area of study within the discipline of nuclear weapons effects research that falls under the sub-iscipline of ground motion research.

During the time when atmospheric nuclear testing was conducted, it was quite difficult to accurately observe the trajectory of nuclear ejecta. In this investigation, the focus is on the largest ejecta earth fragments that behave mainly ballistically. Contrary to a large chemical explosion or meteor impact, nuclear bursts have high energy to mass ratios. Thus, when detonated on or near flat surfaces, crater radius relative to fireball size is small. However, the higher speed ballistic soil clumps can negate this relationship for small yield bursts. Given constraints and priorities, there was limited research regarding major dynamic ejecta during the atmospheric nuclear testing era due to the fireball masking the ejecta.

In recent years United States research initiatives have digitized many of the atmospheric nuclear test films. This allowed for the rebirth of many research projects in nuclear weapons effects that were previously left unanswered. The goal of the digitization effort was not just to preserve the aging and decaying film from the nuclear testing era, but to also see if new tools brought about by the computer age could answer questions previously thought to be unquantifiable.

The goal of this report is to tackle one of these questions: the dynamic ejecta problem, with the new computational tools available to us now that old films have been digitized.

Now, after all of this, one might be wondering why it is worth investigating the ejecta problem further in this way. The answer, is that ejecta poses a number of threats to those assets around it if a detonation were ever to occur, and it is useful to prepare in case those problems ever arose.

Understanding the ejecta problem will bolster the understanding of nuclear weapons effects. For example, it can help inform processes like consequence management, the area of study focused on mitigating effects of weapons of mass destruction (WMD) to a larger extent based on the initial conditions of how a weapon was used. It can also help inform decisions on fratricide and survivability.

It is critical that we understand how a detonation could affect weapon systems, the people operating them, or the people and structures nearby. If a detonation produces ejecta, this research could help make informed decisions.

Chapter 2

Background

2.1 A Quick History of Nuclear Testing in the United States

The first nuclear detonation, "The Trinity Test" was conducted on July 16, 1945. Since then over 1,000 nuclear tests have been conducted by the United States [DOE/NV—209-REV 16]. United States nuclear weapon testing was halted by the veto-proof Exon-Hatfield-Mitchell Amendment in 1992.

2.2 Types of Nuclear Weapon Explosions

In broad terms, there are four main types of nuclear weapons explosions depicted in Figure 2.1 through Figure 2.4

1. Atmospheric

   Detonations that take place in the atmosphere

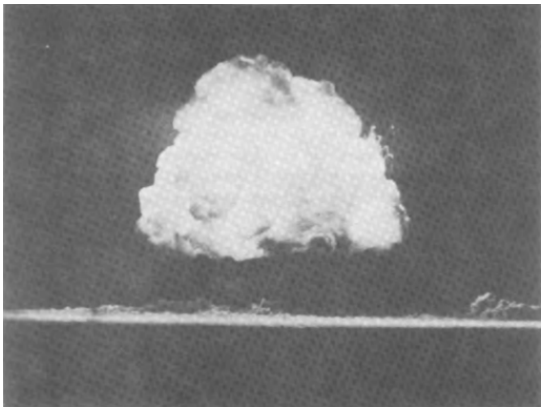


FIGURE 2.1: Example of Atmospheric Detonation [1]

2. High Altitude

   Detonations that occur at heights high well into the atmosphere

FIGURE 2.2: Rocket Used for High Altitude Burst [6]

3. Underwater

   Detonations that take place underwater or close to the surface of the water

FIGURE 2.3: Example of Underwater Detonation, with naval vessel in foreground [6]

4. Underground

   Detonations that occur at varying depth under the surface of the Earth

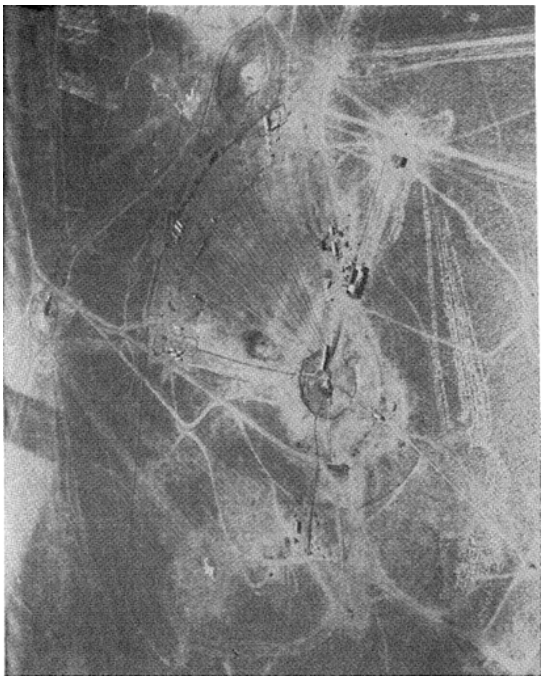


FIGURE 2.4: Aerial View Post Underground Detonation [2]

Reasons these tests were conducted include to answer questions within the field of nuclear weapons effects research.

2.3 Nuclear Weapons Effects

2.3.1 Introduction to Nuclear Weapon Effects

Nuclear weapons effects is the branch of nuclear research dedicated to understanding immediate and long term products that come from a nuclear detonation. Some atmospheric burst example effects are blast, thermal radiation, and ionizing radiation. For specifics, these can include radioactive fallout, thermal flash effects, cratering, electromagnetic pulse effects, or human survivability [7].

One of the more challenging aspects of nuclear weapons effects research is making measurements of the numerical values are not, and cannot, be exact, even for replicated cratering bursts. There are substantial inherent differences: from the natural difficulties that arise from the type of soil a weapon is detonated on, to the weather conditions present at the time of detonation, etc. Even if two weapons are designed to have the same explosive yield, they can have drastically different effects. This accounting is very important because despite how much effort can be put into the study of nuclear weapons effects; there could always be factors that were not accounted for or predicted that would change the way weapon effects happen if it was every used.

2.3.2 The Differences between a Nuclear and High Explosive (HE) Detonations

There are major differences between high explosive (HE) and nuclear detonations beyond the expected yield. The paramount differences are the different mechanisms that affect the detonation and the mass to yield ratio.

In an HE detonation, the detonation mechanism is a chemical reaction. This chemical reaction causes a change in temperature and pressure creating excess energy that is used up during the detonation. Part of that excess energy can also be absorbed by the additional mass present in the reaction, further reducing explosive effect. In contrast, nuclear detonations are caused by the energy generated by the result of formation of derivative atomic nuclei by redistribution of the protons and neutrons within the modified nuclei. Due to the forces between atomic nuclei being far far greater than that of the forces within atoms, the energy produced in a nuclear detonation is of a much larger magnitude than that of a conventional chemical explosive, even if the masses of the two weapons are equal [7]. Both HE and nuclear detonations are used to study ejecta.

2.4 Cratering and Ejecta

2.4.1 Cratering

As seen in Figures 2.5 and 2.9) and permanently deforms the soil due to it being compressed so much that the common soil has a higher density (known as plastic deformation). After this process, some of the material falls back into the crater, with most of the remainder being deposited (see Figure 2.9) around the edges of the new crater to form the lip of the crater. Materials that go past this lip are scattered outwards and known as high speed ballistic ejecta.

2.4.2 Ejecta

As discussed in the last section, certain emplacements of detonations of either nuclear or high explosive causation can create a crater in the earth. The fragmented material that is displaced by this detonation gets thrown in various sizes into the atmosphere and eventually hits the ground. The larger "throw out" material that behaves semi ballistically as it is ejected from the surface and redeposits on the ground is called ejecta. A depiction of this phenomena can be seen in Figure 2.10.

The material that is thrown out consists of either fragmented soil or rock debris; the radial motion of the compressed material causes hoop stresses to produce individual fragments. Once thrown, the ejecta will either fall within the lip of the crater as shown in Figure 2.5, land past the lip of the crater, or stay suspended in the atmosphere for some period of time and eventually land back on the ground.

The angle and velocity the ejecta is able to be thrown out at varies drastically depending on the type of soil the weapon is detonated around because the ejecta originates from the material pushed up and out from the radial force pressing down into the ground, if a crater is being formed at all. For example, for a hard material such as rock, the ejecta would go farther because the material is unable to absorb the shock as well as a softer material such as uncompacted dirt or sand. If the weapon was detonated over sand or loose soil, there would be a relatively large amount of ejecta. But it would not go far because of the shearing force would interact with the ejecta as it moves through the atmosphere, slowing it down immensely and causing the material to shed until it comes to rest. Harder material has a bit of an advantage against the shearing force because it has more compact material to shed than loose soil, so it can retain its relative shape more easily and keep moving against the shearing force.

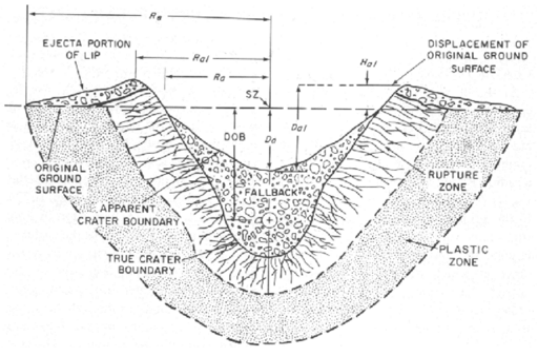


FIGURE 2.5: Cross section of a crater formed by a subsurface nuclear detonation [7]

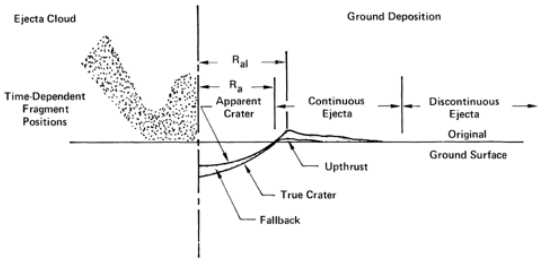


FIGURE 2.6: A more detailed description of the ejecta model [18]

FIGURE 2.7: Photographic depiction of a crater produced by a nuclear detonation. [10]

FIGURE 2.8: Photographic depiction of a crater produced by a nuclear detonation from the inside. A person is standing in the center for depth reference [10]

FIGURE 2.9: Example of the rupture zone [7]

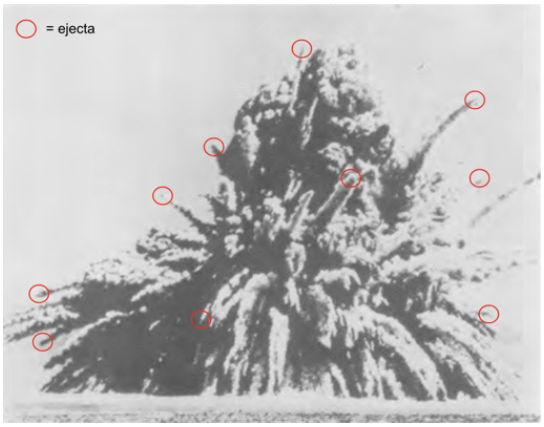


FIGURE 2.10: Photographic depiction of ejecta produced by the same detonation that created the above crater [10]

2.4.3 Underlying Physics

When dealing with the slower ballistic ejecta, it is useful to have an approximation of the radial limit where the ejecta that lands around the lip is going to fall. That radial limit is the outer edge of the crater lip and is usually two to three times the apparent crater radius. A satisfactory approximation of this main mass of ejecta can be described as:

$$R_e \simeq 2.15 R_a \tag{2.1}$$

Where R_e is the radius of the continuous ejecta coverage and R_a is the crater radius. It is also important to note that the discontinuous ejecta that goes past the lip of the crater does not obey the above equation.

Understanding how ejecta that goes past the lip of the crater evolves over time to predict where it lands is a harder effort. Dr. John Wistoski tried to solve this problem with some interesting conclusions for cohesive and soft-rock soils.

2.5 Previous Studies of Ejecta

In the 1970s, Dr. John Wisotski looked at this type of ejecta, or dynamic ejecta, from his specialized high-explosive detonation photographs [20]. The detonations were recorded using multiple cameras that captured both the left and right sides of the detonation. Painstaking effort was made to ensure that the calibration of the cameras that captured the ejecta, the type of cameras used to record the detonation, the frame rate of said cameras, the HOB of the detonation, and the yield of the detonation were all recorded so that accurate data analysis could be performed.

Due to the numerous variables in Dr. Wisotski's study, he devised a computer routine that adjusted the acquired data to ensure the results were accurate. That procedure is as follows:

1. Scale of the cameras so there was a Standard Ground Zero (SGZ), in essence a common axis for all the films
2. Scale in the plane-of-view
3. Correction of the Y coordinate due to the tilt of the camera axis
4. Determination of interim least-square parabolic curve fits of the ejecta
5. Determination of the average drag constant (K) for a fixed increment of space so that the change in velocity could be calculated
6. Inverse-drag routine that started from outside the fireball towards SGZ
7. Determined of the drag coefficient (C_D) from the combined particle's estimated weight and presented area, air density, and drag constant (K)
8. Determination of the Reynolds number based on the particle's characteristic length or dimension, estimated surface escape velocity, air density, and its dynamic velocity
9. Determination of the final least-squares parabolic equations for the ejecta
10. Determination of all initial (escape) and terminal (impact) parameters

An example of the final trajectories of the individual ejecta can be seen in Figure 2.11 and Figure 2.12.

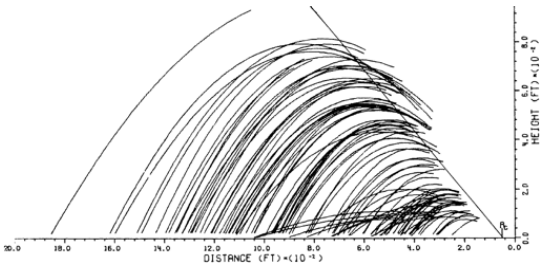


FIGURE 2.11: Final trajectory of ejecta produced by the left side of a detonation. The ejection center line was at 51° [20]

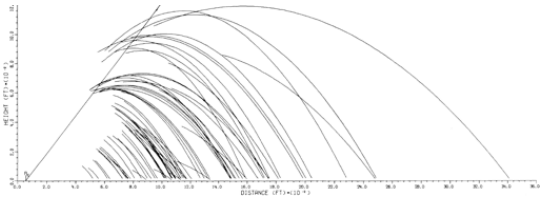


FIGURE 2.12: Final trajectory of ejecta produced by the right side of a detonation. The ejection center line was at 52° [20]

As the study went on, it was realized that the Reynolds numbers of the ejecta were generally so large, due to both the size and changes in velocities of the ejecta, that the K drag constants approached zero. Therefore, there is reason to assume that the in-flight particles were very near ballistic as seen in Figure 2.11 and Figure 2.12.

Given that the trajectories were observed as near ballistic outside the fireball, the equation used to calculate the maximum range of the ejecta (neglecting drag forces) was:

$$S_{max} = (V_E^2 \sin 2\theta_E)/g \tag{2.2}$$

Where $V_E \simeq$ Escape Velocity, $\theta_E \simeq$ escape angle, and g $\simeq$ acceleration due to gravity.

Analysis established that the discontinuous (or dynamic) ejecta traveled with near ballistic trajectories and that the ejecta weight scale directly to the charged weight at scaled distances to the charge weight to the $\frac{1}{3}$ power (i.e. $(\frac{w_2}{w_1})^{\frac{1}{3}}$).

This is an important result, and my investigation will see if dynamic ejecta produced by an actual nuclear detonation agree with the relationship proposed by Wisotski [20]. More work has not been done on studying film data of ejecta due to technical limitations at the time and the prioritization of other studies related to nuclear weapons effects being deemed of more urgent investigation.

2.6 What is the problem at hand and how is it being fixed?

The films that captured nuclear detonations have since been digitized and with current computational tools to provide some new answers to the dynamic ejecta problem. Specifically, this reports investigates the use of image analysis techniques to perform the data acquisition, and the Python coding to perform the analysis on the digitized film of a number of nuclear detonations to study their ejecta.

2.6.1 How This Study Will Differ

There are a few differences between Dr. Wisotski film data that will cause the current investigation to have a few different steps to reach a similar conclusion.

1. The data is extrapolated from digitized nuclear detonation films

2. The type of camera and lenses used are not known. Which leaves conditional variables un-answerable such as if there was any astigmatism present in the lenses or if the camera was angled at all and thus not in the same plane as the detonation
3. There is only one directional viewpoint available for analysis, there is not access to the same detonation from multiple view points or angles.
4. The film was left untouched for decades, prior to digitization, so artifacts might have developed in the film that could cause degradation.
5. There is known jitter in the film that is caused by the mechanical movement of the film being moved between sprocket holes that can cause vertical movement that impact the calibration and the calculation of a SGZ

With all of the difficulties being known, it is believed this is a solvable problem because the amount of computational tools available should allow us to obtain to at least a first order approximation of the velocity and angle of the ejecta. These tools include image analysis techniques/ software and coding scripts that should be able to perform the needed computational routine devised by Dr. Wisotski much faster.

Chapter 3

Image Analysis

3.1 Definition of Image Analysis

Image analysis can be described as the extraction of meaningful information from images by means of image processing techniques. Specifically this analysis involves digital image processing techniques on digitized images using either the open source software packages ImageMagick, Scikit-Image, and PILLOW. These software are used to extract the needed data from the basic digitized frames of the film (i.e., enhancing the ejecta that are coming from the detonation).

3.2 Languages and Packages Used

3.2.1 Python

Python is the main language used in this investigation. It is an open source object oriented programming language that is commonly used for image analysis. It has many packages written in it that are helpful for this investigation [14].

3.2.2 ImageMagick

ImageMagick [9] is an open-source software suite that provides easily accessible command line tools to enhance images as needed. It can help provide some image enhancements that are easier to perform here than in Python because it can all be performed in one line. While technically not written in Python, its commands can be imported into the Python scripts used in this investigation using the os.system package [11].

However, it is important to note that ImageMagick performs its commands in order of how they are written, so it is very important to think about what each command will do and in what order steps should be put in before performing them on an image to ensure the appropriate enhancement you want is being completed.

3.2.3 Scikit Image

Scikit-Image, [17], "is a collection of algorithms for image processing that are free of charge and free of restriction". It was by far the most used package in this report and many of the functions used in this analysis derive from using Scikit-Image.

3.2.4 PILLOW

An image processing suite for Python. It was used frequently for color channel separation, saving arrays to files, and exporting images [13].

3.3 Basics of Understanding Image Analysis

Behind all of the cropping, filters, and random image changes one makes to the photos on their phones or computers, is a surprisingly large amount of mathematics. Topics include linear algebra, discrete mathematics, matrix theory, calculus, and statistics. This section is dedicated to introducing some of the needed underlying mathematical tools and concepts.

3.3.1 What is a digitized image

In essence, once an image is digitized, it is a matrix of discrete intensity values where each individual pixel is an element within that matrix and its particular value is the intensity of the color of that pixel in the range of 0 to 255 (256 = 2^8 because it is in bytes, but it is indexed at 0 so the highest value is 255). 0 is mapped to the color black and 255 is mapped to the color white. Here is an example:

First, let us assume that the image is a simple 2x2 matrix, A. Let $A_{11} = 0, A_{12} = 255, A_{21} = 255$, and $A_{22} = 0$. Here is what the matrix would look like first abstractly, then as a digitized image:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} 0 & 255 \\ 255 & 0 \end{bmatrix} =$$

This passes a simple sanity check because the elements that are equal to 0 appear black in the image and the elements equal to 255 appear white. It is important to note though that this is a very simple example just to get the basic concept down. Figure 3.1 shows a more complex example, however this is still yet more complicated for a color image and one that is of much larger size.

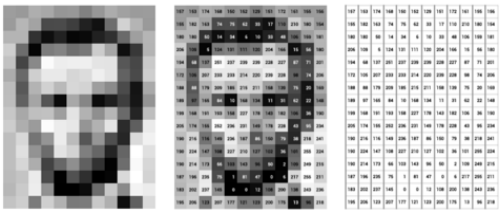


FIGURE 3.1: Digitized Portrait of Abraham Lincoln shown in matrix representation from Stanford [8]

All of the images used in this analysis have been digitized and can be viewed as a giant matrix much like the one seen in Figure 3.1. Some of the images processed in this paper are in standard RGB (Red, Green, Blue) coloration, but are then changed to a grayscale version (much like the examples herein) so that the functions and operators being performed on them can be conducted more accurately.

3.3.2 Red, Green, Blue, and Grayscale Channels

While this might seem on the surface like a topic that doesn't need much discussion, it is important to visit. As discussed above, all digital images can be translated into a matrix of values related to that matrix elements pixel intensity. But how is that intensity determined? It is determined by the intensity of that pixel!

It is easiest to start off with describing the grayscale channel. A grayscale image only needs to know intensity information, i.e. how bright a particular pixel is. The higher the intensity, the lighter the color. So again, 0 is typically mapped to be white and 255 is mapped to be black, all other values in between are various shades of gray. Grayscale images are easier to perform manipulations on because they contain only one channel. This channel is comprised of a two dimensional array of bits, where the size of the array is equal to the height and width of the image. It might not seem like that is an important factor, but it will become evident that things are far more complicated for RGB images.

Moving on to color images, also known as RGB images, contain 3 bytes of data for each pixel that is then split into three parts: one byte for red intensity, one byte for green intensity, and one byte for blue intensity. From there, each pixel will contain a mix of intensity of each individual color. However, since there is a value for red, green, and blue for each pixel in the entire image, it is natural to group those intensities together like what is done in the grayscale image. Thus a dedicated individual red, green, and blue channels can be created! This is still likely quite abstract, so here is an example to demonstrate each channel:

FIGURE 3.2: Albert Einstein and his friend David Rothman at the beach in 1939. The original image is in the center, grayscale on top left, red channel on top right, green channel on bottom left, and blue channel on bottom right [16]

It is clear that the center color image in Figure 3.2 can be separated into individual color channels. Depending on the desired results, certain channels are more helpful than others. Color separation is a paramount part of image analysis.

It is worth introducing here that if an image is in RGB, the conversion to a grayscale image requires an algorithm that approximates the intensity of the combined red / green / blue light.

3.3.2.1 Color Channel Separation

As discussed in previous sections, one of the first steps performed on the images was separating the colored images into just the red channel for further analysis. This was done using PILLOW.

3.3.3 Convolution

Now that basics of digitized images has been covered, it is time to go over some more mathematical tools used to get data from these images.

An important part of image analysis is the use of convolutions. A convolution is a mathematical operation that "multiplies together" two arrays of numbers, typically of different sizes but of the same dimensionality, to produce a third array of values of the same dimensionality. The first input array is usually the original image that needs to be analyzed, and it is usually grayscaled. The second input array is much smaller and is the Structure Element (SE) that interacts with the first input array to give the desired outcome for the outputted third array. Most SEs are known as kernels, and will be discussed at length in the following section. The mathematical

representation of a convolution is as follows:

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x+dx, y+dy)$$

Where g(x,y) is the outputted array, ω is the kernel, and f(x,y) is the original image. This is a very abstract version of the formula though, and does not show what happens for boundary problems.

Perhaps a more holistic way to denote the convolution formula would be:

$$V = \frac{\sum_{i=1}^{q} (\sum_{j=1}^{q} f_{ij} d_{ij})}{F}$$

Here:
f_{ij} = the coefficient of a convolution kernel at position i,j within the kernel
d_{ij} = the data value of the pixel that corresponds to f_{ij}
q = the dimension of the kernel, assuming that is is a square kernel (i.e. for q = 2 then the kernel is a 2x2 matrix)
F = either the sum of the coefficients of the kernel or 1 if the sum of the coefficients is 0
V = the output pixel value that goes into the output array. Keep in mind for cases where V is less than 0, V is mapped to 0. Kernel Eq

Convolution operations can blur, sharpen, erode, dilate, or even detect the edges of an image. However, the size and shape of the kernel used on the image will have a massive impact on the outputted array/ image.

3.3.4 Kernel

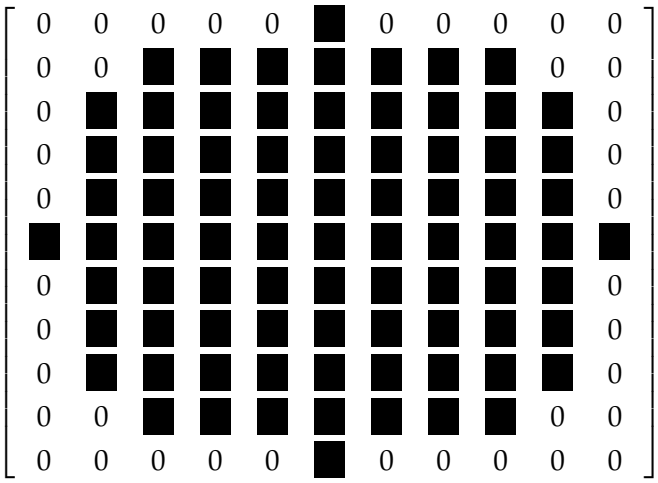
In computer vision and image analysis, a kernel is essentially a small convolution matrix. this small matrix can be used in a variety of shapes and sizes to assist in an operator function that would manipulate an image. It does this by performing a convolution between the kernel and the original image. For example, if you want to perform an edge detection on an image with a kernel of 3, you would use a kernel like this:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

In this analysis though, the specific kernel used was a disk:

1. Disk

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



Mostly through experimental trial and error these were decided to be the main kernels for the analysis. These were the best because they optimized the desired image outcome without increasing the already massive amount of noise present in the images. It is also important to note here that the 1s are mapped to black and 0s to white for this kernel example, it is merely a matter of convention to make visualization easier.

3.3.5 Filters and Thresholding:

Filters use some convolution method, apply it to either the entire domain of the image or just a subset, and outputs the image with that applied filter. The purpose of applying a filter is to modify the original image in such a way that the original image is manipulated to get a more favorable image that can be used in the analysis. Example filters include sharpening, eroding, dilating, edge detection, segmentation, or thresholding.

To avoid further abstraction, here is an example of a sharpening filter:

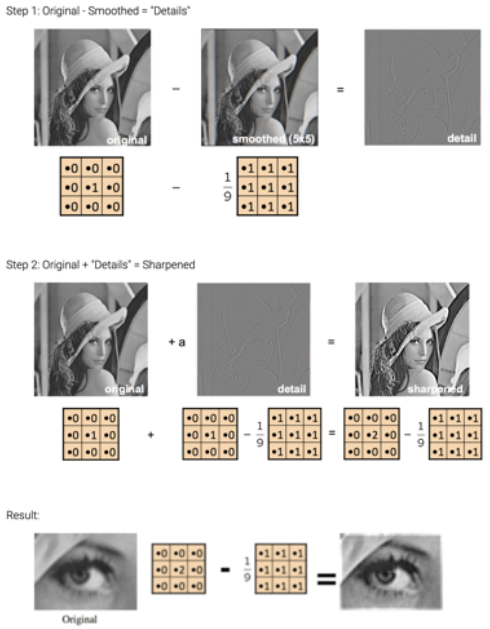


FIGURE 3.3: Example of a sharpening filter algorithm [8]

Perhaps an even easier example to visualize than a sharpening filter, is thresholding. This is one of the simplest parts filters to impose on an image. Speaking mathematically, a thresholding method replaces each pixel in an image with a black pixel if the image intensity is less than the fixed intensity constant chosen, or sets the pixel to white if the intensity is greater than that fixed constant. Here is an example:

Let's allow for every pixel in an image to have an intensity I_{ij} and that there is an arbitrary fixed intensity constant, T, that all pixels in an image will be thresholded by. Then:

$$\begin{cases} \text{For } I_{ij} < T, I_{ij} = 0 \text{ a black pixel} \\ \text{For } I_{ij} > T, I_{ij} = 1 \text{ a white pixel} \end{cases}$$

This is a very simple definition of thresholding, and more complex versions that are used in this analysis are discussed next.

3.3.5.1 Otsu Thresholding

The Otsu method, named after Nobuyuki Otsu, is an image binarization/ thresholding algorithm that designates a single image intensity threshold constant and separates the image into foreground or background based on that constant. That constant is determined by either maximizing or minimizing the inter-class or intra-class intensity variance respectively. In this analysis, the inter-class variance is maximized. All pixels set to black are background and all pixels set to white are considered foreground. The Otsu method is classified as a bilevel threshold [12].

3.3.5.2 Yen Thresholding

While the Otsu Method is considered to be a bilevel threshold, the Yen Method is a multilevel threshold. This method was first introduced in 1995 by Jui-Cheng Yen, et al. In concise terms, the Yen threshold performs an automatic thresholding criterion (ATC) that determines the optimal constant for multilevel thresholding and including an additional term that accounts for the difference between the original and bilevel threshold and includes that into the threshold constant. As a result, you get a more nuanced threshold for the outputted image [22] [21].

3.3.5.3 Contrast Stretching

Contrast stretching is utilized in this study to shrink the intensity range of the input image so that it can be manipulated more easily by previously discussed functions. It does this by analyzing the normalized histogram of intensity values for a grayscaled version of the input image, then takes an inputted minimum and maximum intensity value and sets all values outside of that domain range equal to 0 or 1 depending on desired outcome. For this investigation, those values are set to 0. In other words, the pixels outside of the set intensity domain become black pixels [3].

3.3.5.4 Entropy

Information Entropy, discovered by Dr. Claude Shannon, the American mathematician and inventor of modern information theory, devised a scheme for information entropy, which can be described as computing the base 2 logarithm of the bits within the image. The function returns the minimum number of bits needed to encode the local gray level distribution. This function can act as a noise reducer [4].
Mathematical formalism: Given that there is a discrete random variable X, where there are possible outcomes $x_1, ...x_n$ which occur with probability $P(x_1), ...P(x_n)$, then the entropy, X, can be defined as:

$$H(X) = -\sum_{i=1}^{n} P(x_i) log P(x_i) \tag{3.1}$$

where the $\sum$ denotes the sum over the variables possible values. It is important to note here that the log is in base 2 because pixels are in units of bytes, the log is not in base e like one would normally expect with an entropy calculation in say statistical physics, which although analogous, are not the same entropy.

In this analysis, this entropy function is used as a both a smoothing function thresholding method. As a smoothing function, it reduces the amount of noise present in the image, and reduces total pixel intensity, thus acting like a thresholding method.

3.3.6 Cross Correlation for Template Matching

The correlation between two signals (also known as the cross correlation) is a common approach for feature detection. This method differs from the other convolution techniques previously discussed because this algorithm requires the use of the spatial domain, instead of the frequency domain. The type of cross correlation used in this report is to compute a template match. The goal is to find the correlation coefficient that indicates when the inputted kernel, has the highest correlation somewhere on the larger image the kernel is being matched to. The normalized correlation coefficient (the main coefficient used in this investigation) can be described by using this formula by [5]:

$$\gamma = \frac{\sum_{x,y}[f(x,y) - \bar{f}_{u,v}][t(x-u,y-v) - t]}{(\sum_{x,y}[f(x,y) - \bar{f}_{u,v}]^2 \sum_{x,y}[t(x-u,y-v) - t]^2)^{\frac{1}{2}}}$$

3.3.6.1 Template Matching

Template Matching is a fast, normalized, cross-correlation algorithm that takes a an input image, and uses that as a kernel to then scan a larger image to find the highest correlation coefficient (γ) between that kernel and the same domain sized piece on the larger image. It then identifies that as the "match" with the kernel. The algorithm used in this study is from Scikit-Image [19]. Here is a template matching algorithm along with the output:

FIGURE 3.4: Example of a Template Match Algorithm. The Galaxy is the kernel and Universe is the Image

```
Extracted the Galaxy at location:   368,   92
A    60 x    60 pixel image
Max Correlation from appended list is: 1.0
```

FIGURE 3.5: Output From Hubble Match Template Algorithm

3.3.7 Image Segmentation

Now it is time to digress from the mathematics behind what is being done in this analysis quickly to discuss why these functions are being performed. To understand how and why these functions are used on digitized images, image segmentation needs to be defined.

Image segmentation is the process of partitioning a digitized image into multiple

segments or regions of interest. Image segmentation is done to simplify the original image down into smaller parts so that the analysis can be performed faster and more clearly. It can be performed by applying either a singular or multiple filters as described above. This is an iterative process that takes experimentation to fully understand what the best outcome will be. It requires not only trying multiple functions on the images, but trying them in different order to achieve the best result. Most of the methods and procedure that are discussed in the following section fall under the image segmentation process.

3.4 General Procedure for Image Analysis

There is no one size fits all approach to image analysis. The procedure one follows can vary drastically from project to project or even image to image depending on the needed output or given noise of an image. The exact procedure used in this study varied greatly over the months, but it will be discussed in depth in 4. There is also a discussion on how a different procedure might have worked if more time was allowed in 6.

Due to the variability of a procedure, it is important that one understand all the tools available for image analysis. The introduction provided above was just a small snapshot into this very extensive and rapidly growing field. Please refer to FIgure 3.6 for a more comprehensive list of all the tools available as described in [15].

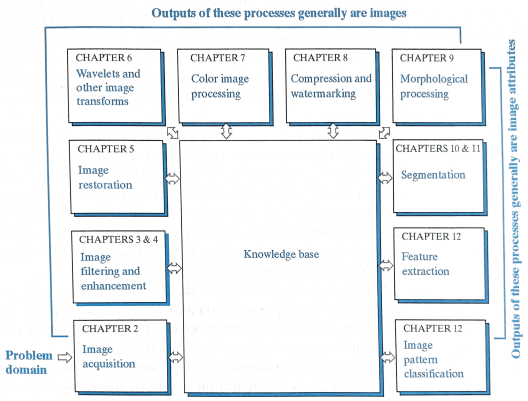


FIGURE 3.6: Procedure for Image Analysis Outlined in [15]

Chapter 4

Methods

4.1 Setting the Stage

It is important to note, that although not an excuse, the images utilized in this study are very old, and although they are of amazing quality for the time, they are not up to par with current state of the art photography. Thus, the analysis of these images was more difficult because there was an intense amount of noise, jitter, and light intensity, and a myriad of other problems that had to be tackled before image segmentation could truly begin.

It was also difficult to choose the time step needed for this analysis. There were many frames available to process, but not all were needed for the actual analysis. It was decided that the time step size for this analysis would be to take a data point once every 10 frames.

4.2 Procedure for Image Segmentation

Please refer to Section 7 to see the scripts that performed the steps about to be outlined. As discussed previously, the exact procedure for this analysis changed over time, but the most effective procedure will be discussed at length.

1. **Importing Frames**
   The script begins by importing 5 frames. The chosen frame is the center frame of interest in this five frame array, then the two time steps before and after that particular frame are also imported. So if we are interested in frame 9, frames 7, 8, 9, 10, and 11 are also imported. Those 5 frames are then weighted until there are 11 total frames being imported. they are weighted such that frame 7 is imported once, frame 8 is imported twice, frame 9 is imported five times, frame 10 is imported twice, and frame 11 is imported once. This might seem odd at first, but this weighting was adopted because there was a large amount of noise when each frame was being processed individually. The weighting allows for the new image that will eventually be processed to have a more "robust" amount of edges to be detected due to the large number of frames being put together. It also assists in reducing noise. However, no frames have

been added together yet, this step is just importing all of the frames needed for the rest of the procedure.

FIGURE 4.1: Entire Original Image

```
frame = ["7", "8","8", "9", "9", "9","9","9","10", "10", "11"]
```

FIGURE 4.2: Example Input Frame Array

2. **Crop to Half Fireball**
   After the 11 total frames are imported, they are cropped down from the original size down to just half of the fireball. Specifically, the right hand side of the fireball. Cropping the image down was done to reduce the total number of pixels that need to be analyzed so that only the area of interest (AOI) was being analyzed. The right side of the fireball was chosen because the spires are most clearly defined on that side.

   The AOI was chosen to be the right side of the fireball and not just the right spires because it was found experimentally that if the half fireball was used for the following steps, the outputted image was more accurate than just using the spires from the start. This is likely due to the additional pixels the functions get to operate on provide more information for the function to process than just including the spires from the start. This cropping was done using a PILLOW function.

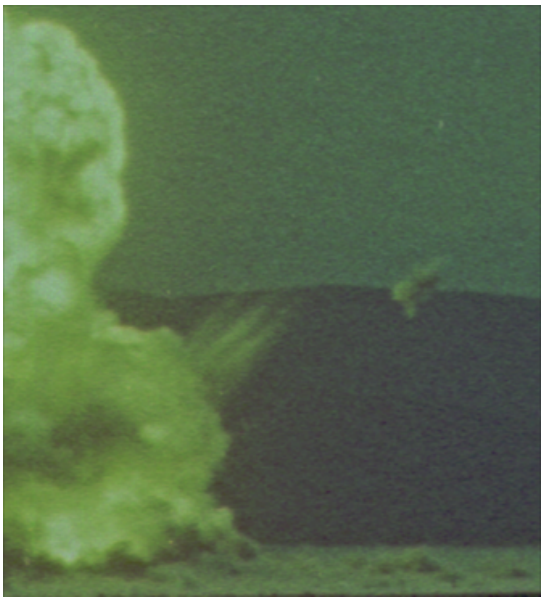


FIGURE 4.3: Right Half of Fireball for Single Image

3. **Extract Out All But the Red Channel**
   After the cropping of these 11 images, the images are turned into an array of data, then the blue and green channels are set to zero so that only the red channel remains for the images. This was doing using a PILLOW function.

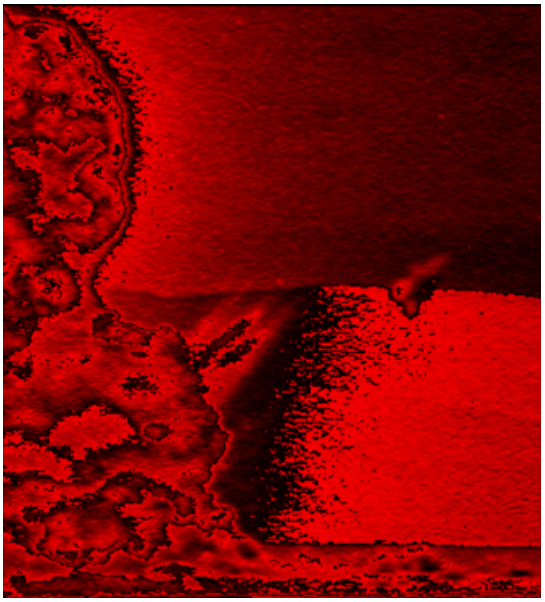


FIGURE 4.4: Red Channel of Half Fireball

4. **Addition of Frames**
   After setting the other color channels to 0, the red channel arrays are then added together to get one "master array". This master array is used to perform the following few steps in the procedure. See Figure 7.2 to see the script that performs the functions discussed over the last four steps.

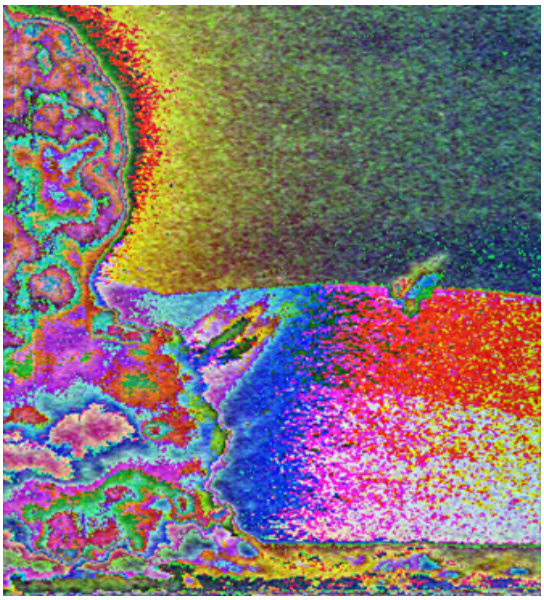


FIGURE 4.5: Half 11 Frame Weighted Fireball

5. **Contrast Stretch**
   This master array now has a contrast stretch performed on it. The Scikit-Image function makes a normalized histogram of the intensity values of the master array, then takes intensity values less than 2 and greater than 98 and sets them equal to 0 (a black pixel).

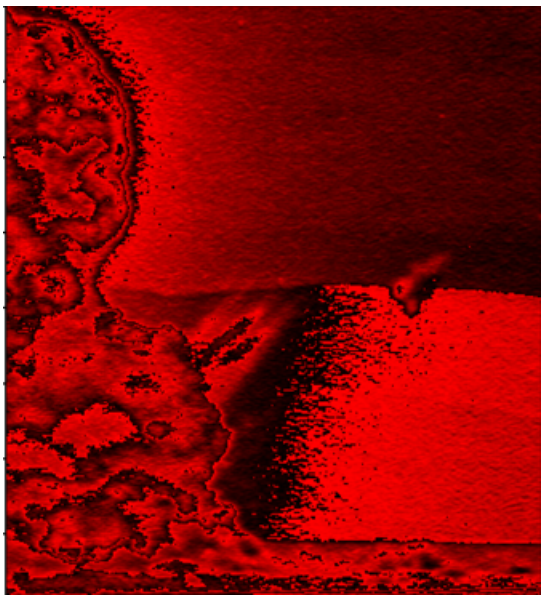


FIGURE 4.6: Contrast Stretch of Half Fireball

6. **Entropy Calculation**
   After the contrast stretch has been performed, an additional smoothing function is applied in the form of an entropy function. This is done by using the entropy function from Scikit-Image with a disk shape kernel with a diameter

of 5 pixels. The entropy function then uses this kernel to compare the information entropy of the pixel it is analyzing with its neighbors and smooths out the additional noise. This helps prepare the fireball for the thresholding method that is about to be preformed on it so that the thresholding method can be as accurate as possible.

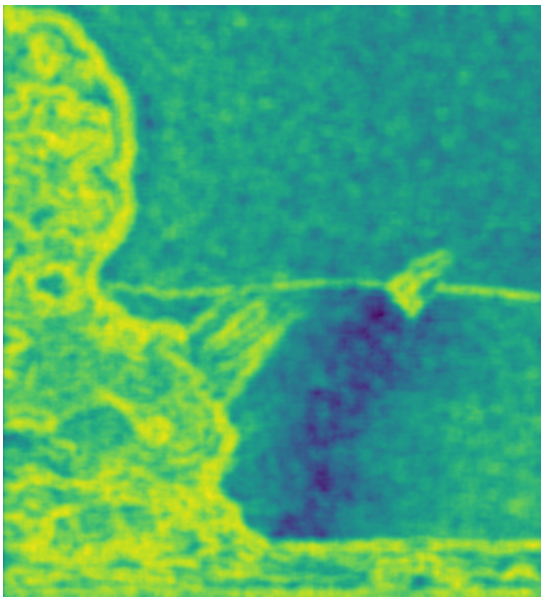


FIGURE 4.7: Entropy of CS Half Fireball

7. **Otsu Threshold**
   Next, the Otsu Threshold is performed on the half fireball. This essentially "binearizes" the image into background and foreground. Specifically, the foreground would be the fireball which would be converted to white (or 0 in terms of pixel intensity) and the background would be converted to black (or 1 in terms of pixel intensity). Sadly, this algorithm is not perfect and there is still a decent amount of noise present that the algorithm detects as foreground. This failure is not just a fault of the algorithm, due to the varied pixel intensities caused by the light coming from the fireball, the algorithm can struggle in separating foreground from background. An example can be seen in the large white bar that crosses from left to right across all of Figure 4.8, along with the white spots in the upper left hand corner.

   Now that the image is binarized, the half fireball is cropped down to just a (100,100) pixel sub-image that contains just the binarized spires. See Figure 7.3 for the script that performs steps five, six, and seven.

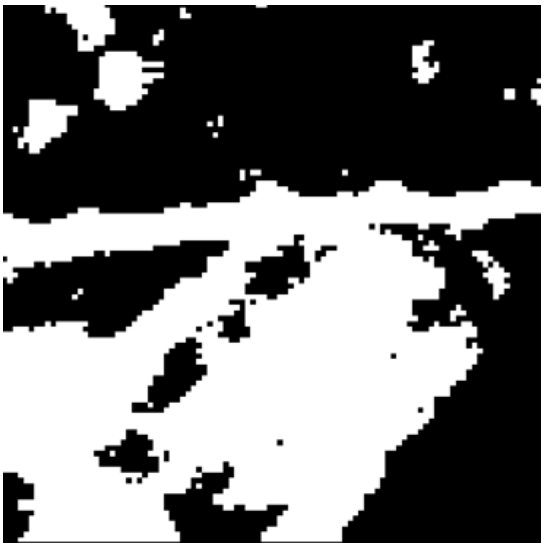


FIGURE 4.8: Otsu Threshold of Spires

8. **Crop Out Spires of Next Time Step**
   Critical to the rest of the procedure is the acquisition of a (100,100) pixel cut out of the spires from the next time step. This is done in a new function within python that imports the next time step frame then crops it down to the same (100,100) pixel cut out as the otsu thresholded spires. For example, if the otsu threshold spires are from frame 9, the unaltered spires would be cropped from frame 10. This counter intuitive step was taken from experimental observation that showed this was the best way to overlap the two images and get consistent thresholded results. See Figure 4.9 for an example of the unaltered (100,100) pixel cut out of the spires.

FIGURE 4.9: (100,100) Cut Out of Spire ROI

9. **Make Otsu Threshold Spires Transparent and Overlay**
   The next step is to take the otsu thresholded spire image and make the white

pixels transparent. Since that outline is white, and it eventually needs to be overlayed on top of the unaltered spires, the image has a "-fill" command and a "-opaque" command performed on it to make the white pixels transparent. After this, the now transparent Otsu spires are layered on top of the original unaltered spires one time step later using a -layers flatten command. The in between steps are not well suited for a physical examples, but here is the final result of this process in Figure 4.10. See Figure 7.4 to see the script that performs steps eight and nine.

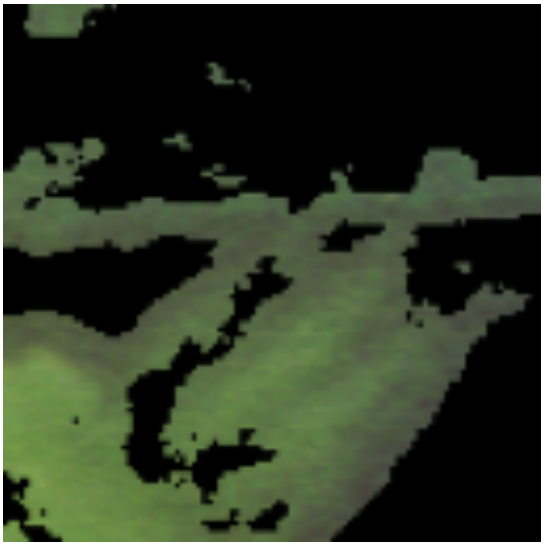


FIGURE 4.10: Transparent Otsu Spires Overlayed on Next Time Step Un-altered Spires

10. **Yen Threshold**

    With Figure 4.10 complete, it is clear that the Otsu filter left some area around the spires that should have been in the background. The Yen Threshold will now be applied tighten the threshold and leave only the brightest pixels behind. These pixels are what will be used to determine the tip of the spire. The result can be seen in Figure 4.11.

FIGURE 4.11: Yen Threshold

11. **Repeat Step 9 for Yen Threshold**
    Much like the bullet point says, the final step in the procedure for image segmentation was to repeat step 9 for then Yen Threshold version of the spires. The result is shown in Figure 4.12. See Figure 7.5 to see the script that performs steps ten and eleven.

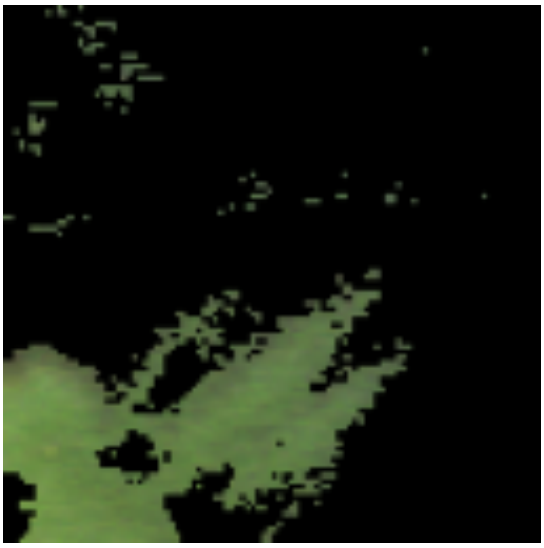


FIGURE 4.12: Final Result

4.3 Data Acquisition

Using matplotlib's function plot, two images are placed side by side as seen in Figure 4.13. Next the cursor is used to find the farther tip of the spire and the (x,y) coordinate would be recorded. This was repeated till all frames and each respective spire per frame had tip data points collected. Since the spires evolve over the time steps of this analysis, the spires start off as a singular "blob" then evolve into three

distinct tips, then they disintegrate until only the middle spire, the largest "finger", remained. Due to this, the most data was able to be collected for the middle spire, the second most for the lower spire, and the least for the upper spire.

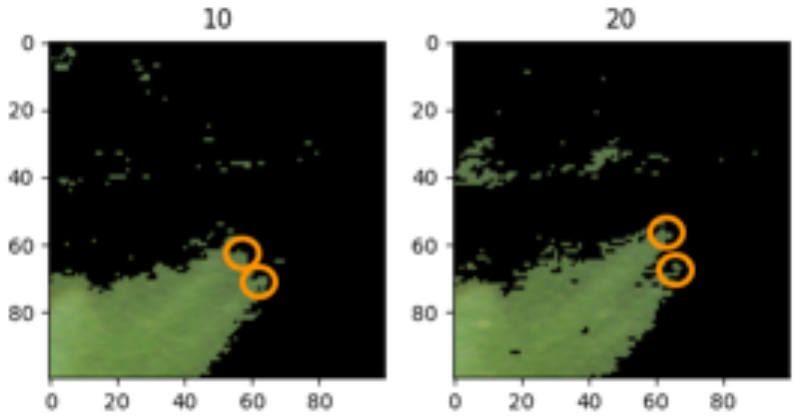


FIGURE 4.13: Two Sets of Spires Side by Side With Circles to Indicate Spire Tips at Early Time Step

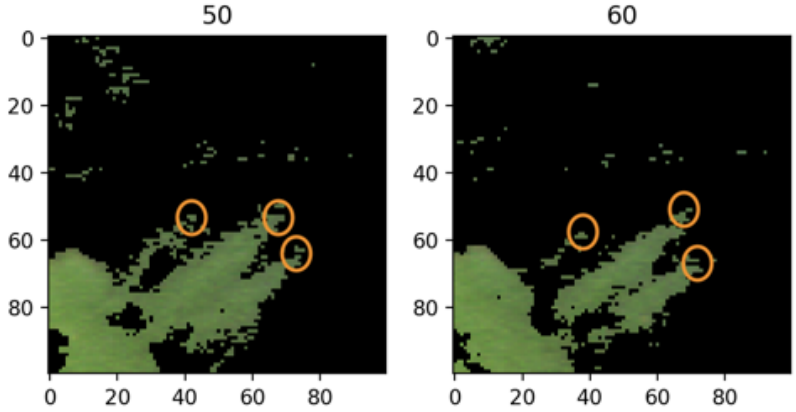


FIGURE 4.14: Two Sets of Spires Side by Side With Circles to Indicate Spire Tips at Later Time Step

4.4 Data Visualization and Validation

These data points were then visualized on the original spires by using Image.Draw and plotting all collected points and place them onto the covered yen photos. This can be seen in Figure 4.15

From there, a *.gif file was made to track the evolution of the spires over time to make sure the data passed the first sanity check, thankfully it did. As the spires progressed, they follow along the trajectory of the farthest points as identified.

Finally, the spire tip locations were converted from the small cropped images and placed on the original images. This was done by figuring out the small pixel location and mapping it back onto the original image, then scaling all the collected points. The difficult thing is that images do not start at the origin, instead they start at (0, ymax) where ymax is the number of pixels the image has in the y direction. This is perhaps counter intuitive because we are used to the origin being (0,0).

However, there is another step that is required before this. It is necessary to find where the (100,100) section of spire falls on the original image. Specifically, the location of the bottom left corner of the (100,100) on the original image. This was done using a template matching function from Scikit-Image. From there, an extra 100 pixels was subtracted from y values (found through experimentation). Now even after all of this, the y values are still not quite accurate. The reason comes again from the inverted y axis. So, to get a proper pair of coordinates, take the new y value and subtract that value from ymax. After that, it is fairly simple to plot all the points onto the image and see that the points follow the trajectory of the spires again over time. The final outcome is seen in Figure 4.16

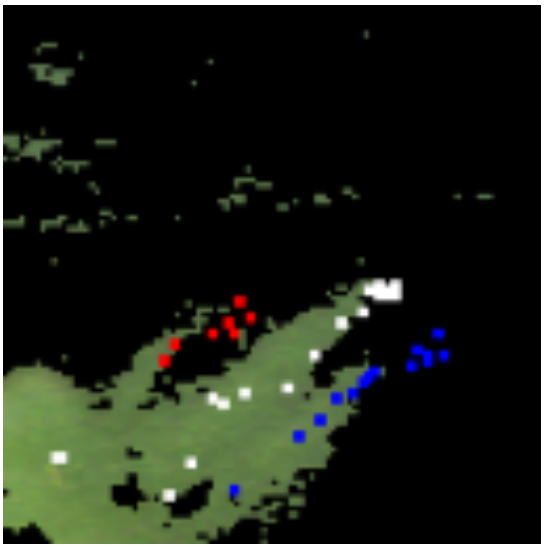


FIGURE 4.15: All Data Points Collected on One Spire Set

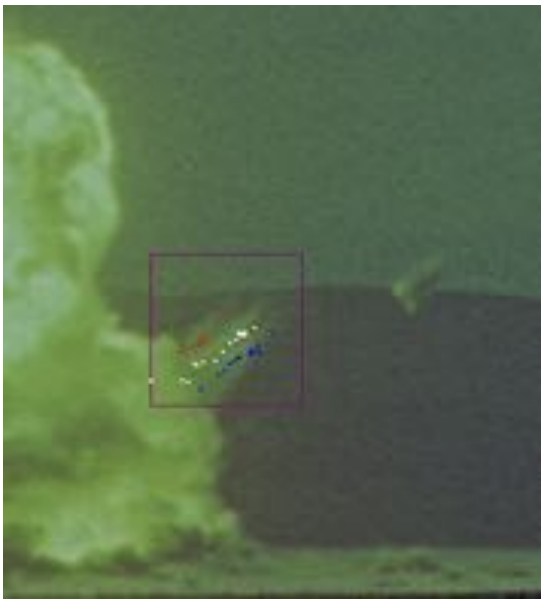


FIGURE 4.16: All Data Points on Half Fireball

4.5 Data Analysis

In essence, the data points collected are being converted into distances using basic Euclidean geometry and pixel to meter measurements. After those distances are found, the velocity is being calculated using the time step of the frames to get the velocity over time and a polynomial curve fitting plot. The angle will be calculated from that same curve fit and the time step. r_0 is calculated based on adding the 10% of the fireball radius at time of detonation to the center point of the detonation. This is discussed in further detail in the next section.

THIS PAGE IS INTENTIONALLY LEFT BLANK

Chapter 5

Results

As discussed in earlier sections, ejecta cannot come from SGZ, the origin/ intersection of the film plane and center of the fireball. Any material that is at the center of the detonation is vaporized and is expelled vertically upwards. For ejecta to be created, it must come from an origin point, r_0, some distance away from SGZ and be expelled at an angle for it to survive. For this particular kind of detonation, r_0 can be determined by taking 10% of the fireball radius and adding it to the origin because at that point the material would no longer be vaporized. The fireball radius at the time of detonation was determined to be approximately $100\sqrt[3]{2} \approx 126$ meters, so the r_0 is 12.6 meters off of SGZ.

After detonation, the fireball goes through a period of rapid expansion and growth, which can be seen in Figure 5.1. The initial detonation occurred at frame 3337, so at frame 3336 there was no height to the fireball (hence the point at the origin), and at the very next frame, the height of the fireball was 468 pixels high. Shortly after, at frame 3376, the ejecta began to be noticed and the ejecta heights are the rest of the data points seen in Figure 5.1. Again, the rest of the points past frame 3376 are the heights of the spires, not the height of the fireball, which is still growing at this time but that data was not collected.

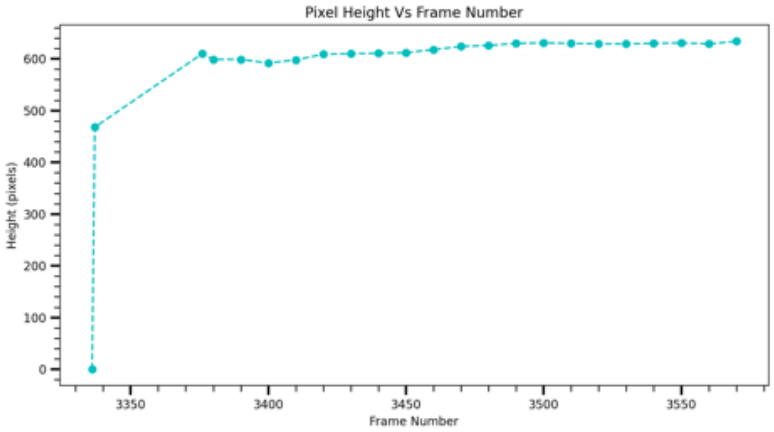


FIGURE 5.1: Pixel Height of Middle Spire Vs Frame Number

Next is converting pixels and frames to meters and time. The pixel to meter conversion occurs by taking the pixel measurement and multiplying it by 0.8, due to the focal length of the camera lens used to capture the film. The frame rate of the camera was 100 frames per second, so the time step in between each frame is 0.01 seconds. Taking these unit conversions into account, Figure 5.1 can be shown as Figure 5.2. It is important to note that the 3337 point was not used in 5.2 because the height of that measurement was used to create the difference in length between all of the following spire measurements, so subtracting its own value from itself gives a height of 0. It will also be mentioned again more explicitly that frame 3336 was used as time = 0 because that is the frame prior to the detonation occurring. The film recording was started prior to detonation, but since this investigation is focused on post detonation and nothing occurs prior to 3337, only one frame before the detonation is needed to create an origin.

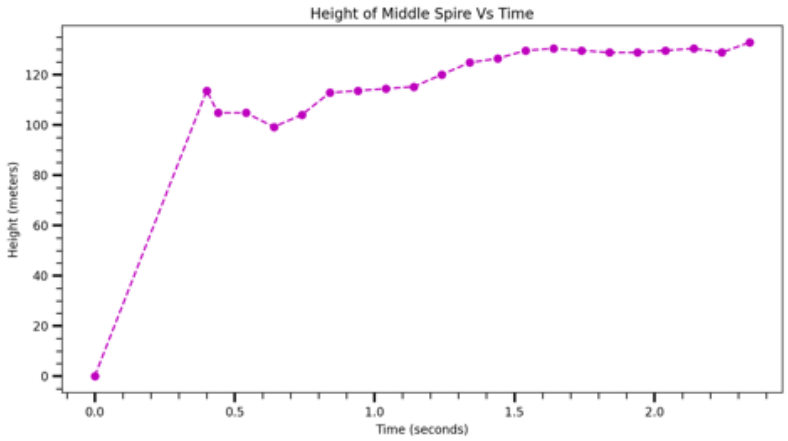


FIGURE 5.2: Height of Middle Spire Vs Time

Now that units have been converted properly, the velocity needs to be found for the spire. The data collected for the change in length of the spire over time, seen in Figure 5.3, had a polynomial fit of $d = -14.475t^2 + 68.768t + 100.2$, since this is in units of t^2, this fit can be viewed as the deceleration of the spire over time. It is important to note here that this fit is approximating the real path length with straight distances to each point because the velocities collected were not accurate nor precise enough to build the correct curved path using $\sqrt{(\frac{dy}{dt})^2 + (\frac{dx}{dt})^2}$ from calculus. This latter method was attempted but the result was far too chaotic to be accurate. So the derivative of this curve fit was used to get the velocity.

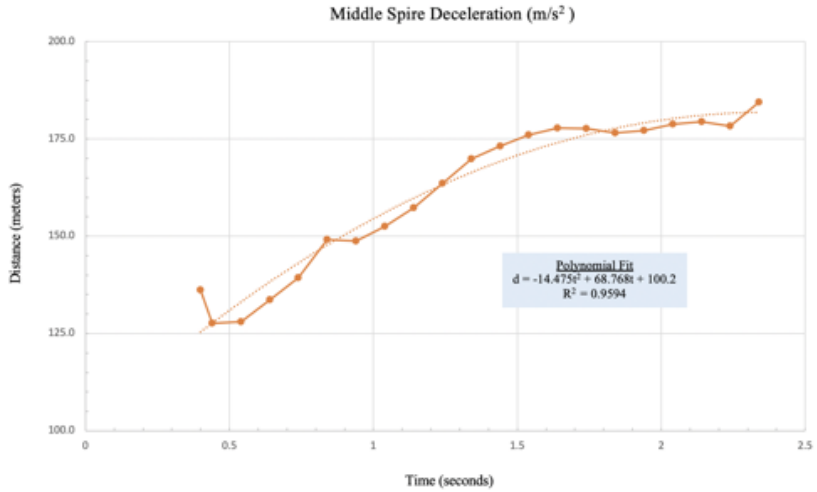


FIGURE 5.3: Deceleration of Middle Spire Vs Time

Taking the derivative of this fit with respect to time, $d = -28.95t + 68.768$, and plugging in the time evolution provides the change in velocity over time curve seen in Figure 5.4. The two data points seen at velocity = 350m/s occur because that is the average velocity needed to go from r_0 to point where the first spire was noticed at frame 3376/ time = 0.4s after detonation. The intense deceleration from 350 m/s to just 57.18 m/s at time = 0.4s occurs due to the ejecta moving essentially through a vacuum while still inside the fireball which allows it to move incredibly fast, but as it emerges from the fireball the ejecta hits a density wall created by ambient air that is at several ATM, forcing it to slow down and the aerodynamic drag force causes it to break apart, further slowing the ejecta down. This causes the linear decrease in velocity seen in Figure 5.4.

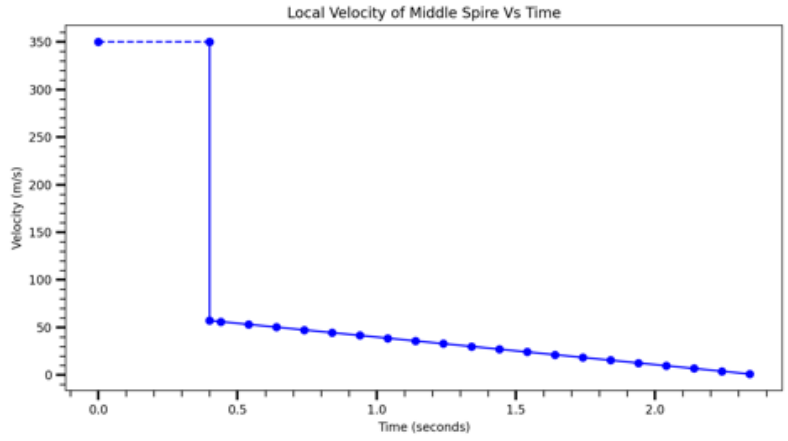


FIGURE 5.4: Local Velocity of Middle Spire Vs Time

The local angles (Figure 5.5) were calculated based on the polynomial fit obtained from graphing the (x,y) tip locations for the middle spire over time as seen in Figure 5.6. The arc tangent of the derivative of the polynomial fit was used to calculate local change in angle for all but the first three points. The first three angles were calculated by taking the arc tangent of their raw coordinates because those points did not match the fit. This decreasing angle agrees with the fact that ejecta cannot come from the center of the detonation because it would be vapor, so it can only survive if it is expelled at an angle and would be a liquid or solid depending on the angle of ejection. This decrease also agrees with the type of soil this detonation occurred around and also decreases as the velocity decreases.

There is still some discrepancy for the first three angle data points that require validation. It is believed that the reason why these points do not align with the polynomial fit is because these measurements occurred so early in time for the observation of the ejecta that the collected data points could be showing the same spire breaking apart, causing those changes in angle. Thus, the angles can be checked for accuracy by adding in $\frac{1}{2}gt^2$, where g is gravity and t is time, to the y values for the tip location, taking the linear fit, and finding the arc tangent to get the angle, as seen in Figure 5.7. That arc tangent produces an angle of 54°, which validates that while the first few angles might not have fit the curve, the main clump of ejecta is followed overall in this plot because 54° is consistent with the initial angles observed in Figure 5.5.

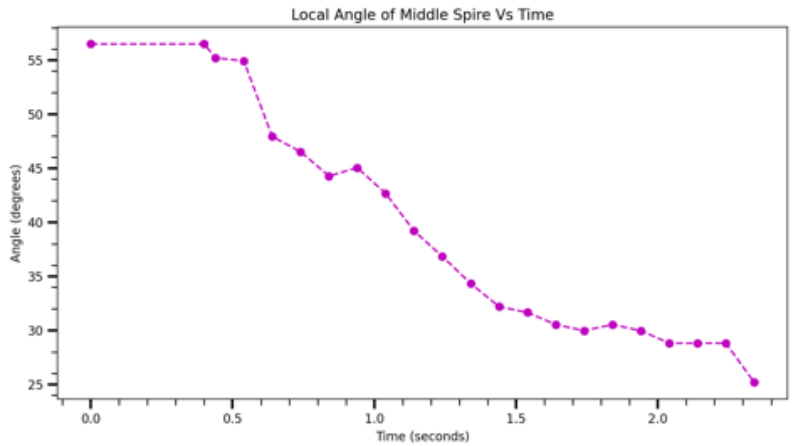


FIGURE 5.5: Local Angle of Middle Spire Over Time

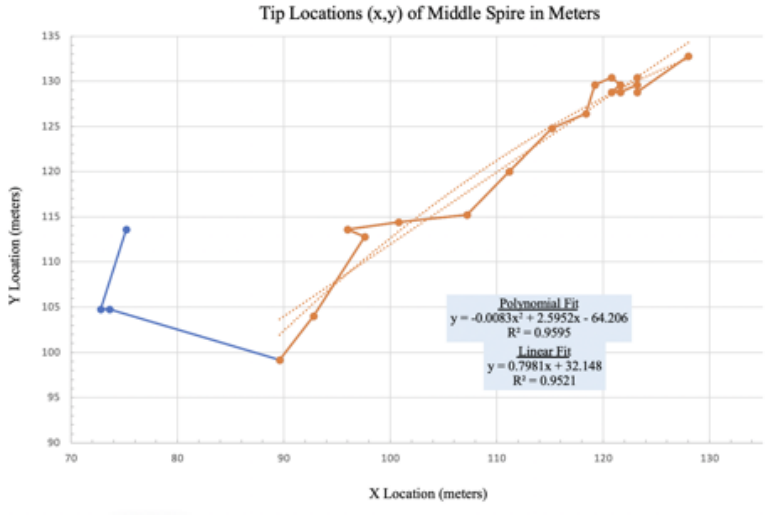


FIGURE 5.6: Middle Spire Tip Location

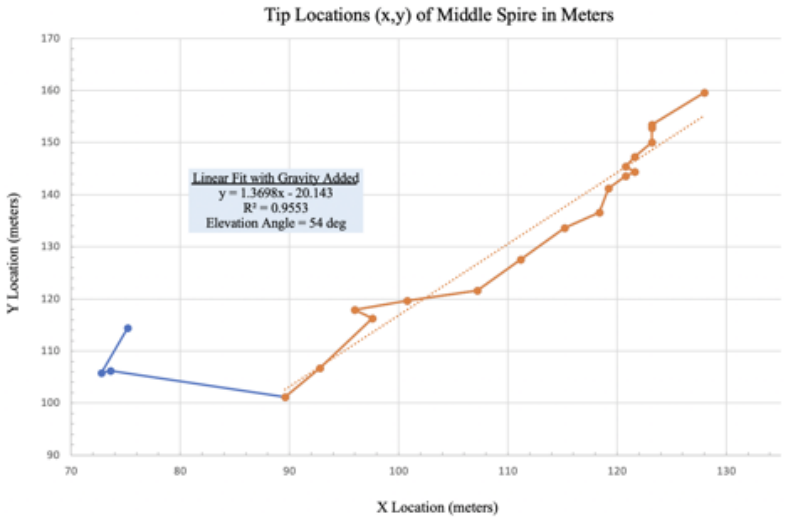


FIGURE 5.7: Initial Angle of Middle Spire with Gravity Added

THIS PAGE IS INTENTIONALLY LEFT BLANK

Chapter 6

Conclusion and Recommendations

6.1 Conclusion

Overall, this investigation can be seen as a success. A schema was devised to properly identify the tips of spires, and was used to calculate their velocities. The angles and r_0 were also able to be calculated. It is hoped that this investigation helps inspire further investigations into nuclear weapons effects using similar image analysis techniques. With this in mind, here are some recommendations for anyone who wishes to continue either with this effort, or a similar one.

6.2 Recommendations

It is important to note again that the procedure for image analysis is often fluid. With this in mind, if there were more time to perform this analysis, there are a few things that could possible change the results for the better.

1. After conducting the Otsu Threshold and putting the original spires back onto the transparent version, a entropy calculation could be done again before performing the Yen threshold to see if that would preserve more pixels.

2. Morphological Active Contours without Edges (ACWE) Snakes was a method used earlier in the study but was eventually abandoned due to the lack of ability to clearly identify the spires, although it was accurate in identifying the fireball. The recommendation is that after the original spires were put ton the Otsu version, to then perform the ACWE snake because there would be far less noise for it to deal with, perhaps yielding a more accurate result. ACWE Snakes are designed to work on low contrast images, much like the ones used in this investigation. If high contrast images are available, Morphological Geodesic Active Contours (GAC) snakes can be pursued instead.

3. The template match method should not be abandoned as a way to actually identify the tips of the spires. As more sophistication is built into the model to isolate the tips and get rid of the background and noise, the template match could easily identify the spire tips and provide the location and correlation/ confidence in the measurement which can help with error propagation. This

algorithm even works for rotated versions of the kernel, so if there is an application where the spires arc heavily, this could be a worthwhile algorithm to utilize.

4. the Morphological Edgeout Algorithm from ImageMagick can also be utilized. It was eventually abandoned due to the Otsu Threshold performing better in this study, but it is highly capable of detecting edges and contours in high contrast images, which this investigation does not have. The edgeout could be used to outline the spires to use as a mask to get rid of excess noise and isolate the spires.

Chapter 7

Appendixes

7.1 Appendix 1, Scripts

```
#Step 0, import general packages
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import os
from skimage.filters.rank import entropy
from skimage.morphology import disk
from skimage import exposure
from skimage import color
from skimage.filters import threshold_otsu
from skimage.filters import threshold_yen
```

FIGURE 7.1: Packages Used in Functions

```
def imp_images_weighting11_1():

    frame = ["7", "8","8", "9", "9", "9","9","9","10", "10", "11"]
    arrays=[]

    dir = "~/Directory/"
    for i in frame:
        input = dir + i +".tif"
        frame_i = Image.open(input)#.convert("L")

    #for partial fireball

        left = 1395
        top = 1338
        right = 1750
        bottom = 1730

    # Cropped image of above dimension

        frame_i = frame_i.crop((left, top, right, bottom))
        print(i, frame_i.size)
        frame_i = np.asarray(frame_i)
        red_image_i = frame_i.copy() # Make a copy
        red_image_i[:,:,1] = 0 #Get rid of green channel
        red_image_i[:,:,2] = 0 #Get rid of blue channel
        arrays.append(red_image_i)
    global array4
    array4 = sum(arrays)
    plt.imshow(array4)
    plt.show()
```

FIGURE 7.2: Import Images, Red Channel Separation, and Summation Function

```
def cs_en_otsu():
    imp_images_weighting11_1()

    p2, p98 = np.percentile(array4, (2, 98)) #sets up contrast stretch
    img_rescale = exposure.rescale_intensity(array4, in_range=(p2, p98))
    #above performs contrast stretch
    entr_img = entropy(color.rgb2gray(img_rescale), disk(5)) #entropy function

    thresh = threshold_otsu(entr_img) #establishes otsu threshold
    binary = entr_img > thresh #performs otsu threshold

    left = 100
    top = 150
    right = 200
    bottom = 250

    # Cropped image of above dimension
    binary2 = Image.fromarray(binary)
    binary2 = binary2.crop((left, top, right, bottom))  #performs crop
    binary2.save("~/Directory/9_otsu.gif")

cs_en_otsu()
```

FIGURE 7.3: Contrast Stretch, Entropy, and Otsu Functions

```
def single_img_IM():
    cs_en_otsu()
    dir = "~/Directory/"

    input = dir + "10.tif"
    frame_base = Image.open(input)

    #for partial fireball mask

    left = 1395
    top = 1338
    right = 1750
    bottom = 1730
    # Cropped image of above dimension

    frame_base = frame_base.crop((left, top, right, bottom))
    print(frame_base, frame_base.size)

    left2 = 100
    top2 = 150
    right2 = 200
    bottom2 = 250

    frame_i2 = frame_base.crop((left2, top2, right2, bottom2))
    #Above crops to spires

    frame_i2.save("~/Directory/10_crop_spire.tif")
    #Saves spires
    cmd = "convert 9_otsu.gif -opaque white -fill transparent otsu_t.tif"
    os.system(cmd)
    cmd2 = "convert otsu_t.tif 10_crop_spire.tif -layers flatten 10_otsu.tif"
    os.system(cmd2)

single_img_IM()
```

FIGURE 7.4: Create Single Cropped Spires and Perform ImageMagick Commands

```
def yen_threshold():

    dir = "/Users/jackmoody/Desktop/"

    input = dir + "10_otsu.tif"
    frame_i = Image.open(input).convert("L")
    global frame_2
    frame_2 = np.asarray(frame_i)

    thresh = threshold_yen(frame_2)
    binary3 = frame_2 > thresh

    binary4 = Image.fromarray(binary3)
    binary4.save("/Users/jackmoody/Desktop/10_yen.tif")
    cmd = "convert 10_yen.tif -opaque white -fill transparent yen_t.tif"
    os.system(cmd)
    cmd2 = "convert yen_t.tif 10_crop_spire.tif -layers flatten 10_yen_F.tif"
    os.system(cmd2)

yen_threshold()
```

FIGURE 7.5: Yen Threshold and ImageMagick Commands

7.2 Appendix 2, Additional Functions Pursued

1. Morphological Snakes from Scikit Image Morphological Snakes
2. Morphological EdgeOut Morphological Snakes

THIS PAGE IS INTENTIONALLY LEFT BLANK

Bibliography

[1] Steve Rohrer & Carl Maag. "Project Trinity 1945 - 1946". In: (1982). URL: https://www.dtra.mil/Portals/61/Documents/NTPR/2-Hist_Rpt_Atm/1945_DNA_6028F.pdf.

[2] Joe A. Stinson & Carole R. Schoengold. "Operations Nougat, Flintlock, Latchkey, Bowline, and Emery". In: (1997). URL: https://www.dtra.mil/Portals/61/Documents/NTPR/3-Hist_Rpt_UGT/1962-1972_DNA6326F_Operations_Nougat_etc.pdf.

[3] *Contrast Stretch by Scikit-Image*. URL: https://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_equalize.html.

[4] *Entropy by Scikit-Image*. URL: https://scikit-image.org/docs/dev/api/skimage.filters.rank.html#skimage.filters.rank.entropy.

[5] *Fast Normalized Cross-Correlation by JP Lewis*. URL: http://scribblethink.org/Work/nvisionInterface/nip.html.

[6] J.H. Hallowell, E.J. Martin F.W., McMullan R.A., Miller M.J., Osborn C.F. & Shelton Berkhouse F.S. Calhoun F.R. Gladeck K.G. Gould. "Operation Hardtack I -1958". In: (1982). URL: https://www.dtra.mil/Portals/61/Documents/NTPR/2-Hist_Rpt_Atm/1958_DNA_6038F.pdf.

[7] Samuel Glasstone & Philip J. Dolan. *The Effects of Nuclear Weapons*. United States Department of Defense and the United States Department of Energy.

[8] *Image Filtering Tutorial By Stanford*. URL: https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html.

[9] *ImageMagick*. URL: https://imagemagick.org/.

[10] Carl Maag, Robert Shepanek, Inara Gravitis & Stephen Rohrer. "Projects Gnome and Sedan, The PLOWSHARE Program". In: (1983). URL: https://www.dtra.mil/Portals/61/Documents/NTPR/2-Hist_Rpt_Atm/1961_DNA_6029F.pdf.

[11] *OS*. URL: https://docs.python.org/3/library/os.html.

[12] *Otsu Threshold by Scikit-Image*. URL: https://scikit-image.org/docs/dev/api/skimage.filters.html#skimage.filters.threshold_otsu.

[13] *Pillow*. URL: https://pypi.org/project/Pillow/.

[14] *Python*. URL: https://www.python.org/.

[15] Richard E. Woods & Rafael C. Gonzales. *Digital Image Processing*. Pearson.

[16] *Rare Historical Photos*. URL: https://rarehistoricalphotos.com/einstein-at-the-beach-1939/.

[17] *Scikit-Image*. URL: https://scikit-image.org/.

[18] William R. Seebaugh. In: (1976).

[19] *Template Matching by Sci-kit*. URL: https://scikit-image.org/docs/dev/api/skimage.feature.html#skimage.feature.match_template.

[20] John Wisotski. In: *Impact and Explosion Cratering* (1977).

[21] Chang F.J., Yen J.C., & Chang S. In: *IEEE* ().

[22] *Yen Threshold by Scikit-Image*. URL: https://scikit-image.org/docs/dev/api/skimage.filters.html#skimage.filters.threshold_yen.