PIPE CRACK DETECTION IN CIVIL STRUCTURES

A Thesis submitted to the faculty of
San Francisco State University
In partial fulfillment of
the requirements for
the Degree

Master of Science

In

Engineering: Structural/Earthquake Engineering

by

Esteban Ricardo Kim

San Francisco, California

December 2022

Certification of Approval

I certify that I have read Pipe Crack Detection in Civil Structures by Esteban Ricardo Kim, and that in my opinion this work meets the criteria for approving a thesis submitted in partial fulfillment of the requirement for the degree Master of Science in Engineering: Structural/Earthquake Engineering at San Francisco State University.

_____

Cheng Chen, Ph.D.
Professor,
Thesis Committee Chair

_____

Jenna Wong, Ph.D.
Associate Professor

Abstract

Structural health monitoring systems in civil structures is widely researched topic that is regarded of high interested but that encounters many challenges. There is interest to further develop these systems to help maintain structures and help prevent any major damages. However, the issues faced range from engineering feasibility, cost effectives, and a defined methodology that can produce measurable results. This paper will explore previously researched methods for crack detection in civil structures and the feasibility to build a system that can implement these methods as well as the effectiveness of the results it can produce. Image processing methods such as the Canny, Sobel, Prewitt and Gaussian are used on different types of images of pipes and other civil structures. An algorithm is built combing the image processing methods mentioned and the crack widths and lengths are calculated for evaluation. The effectiveness of the algorithm is discussed and compared to previously used methods.

Acknowledgements

This research project was undertaken after review of a couple of research topics recommended by Dr. Chen. The research was difficult and invited to explore methodologies that I had not reviewed in depth within the graduate program study courses. Fortunately, this pushed my boundaries and helped prepare me for life after graduate school. I would like to thank my professor and counselor Dr. Cheng Chen for his guidance, support and advise during this process. I'd also like to thank all the professors who were part of the process with a special mention to Dr. Jenna Wong, without their supplemental academic support this would not have been the same. Lastly, I'd like to thank my family for their unconditional support as this research was done during a time of global uncertainty and personal hardship.

Table of Contents

List of Tables

List of Figures

Chapter 1. Introduction

There are roughly about 2.5 million miles of pipelines in the U.S. which makes it the largest network of energy pipelines in the world [1]. This extensive system carries with it extremely valuable energy resources within it and is essential for everyday life as it transports natural gas, crude oil, and volatile liquids. However, this system comes with a high cost and high-risk factor as failure of such system can be catastrophic. In the U.S., the cost to deal with corrosion problems is $100 billion each year [1]. These problems naturally occur due to the environmental effects pipes are subjected coupled with the loading and deterioration over their life span. Taking these factors into consideration it is still difficult to effectively and efficiently evaluate the health of an entire pipeline as they can be several hundred miles long.

Since this is a such a big task to handle, it is imperative for there to be a proper system of procedures in place that can inspect these pipelines and provide accurate data with regards to the health of the pipe. This is easier said than done and there only exists a slim tolerable limit that if surpassed can lead to catastrophe, as PG&E found out when a pipe ruptured in San Bruno, CA. This study will discuss the nature of the events that occurred on the 10^{th} of September in 2010 and the reasons behind the failure of this pipeline. The methods and tests that could have been used to prevent these events will also be discussed further in detail in order to provide a route to the most feasible means of failure detection [3].

This is a topic of great interest for both the public and private sector since the detection of deteriorating pipes is essential for the industry. However, detecting and assessing cracks that will lead to failure, is not as straight forward as it may seem like randomness and irregularities of the cracks can prove challenging to model. The approaches that will be presented in this paper were proposed by S.K. Sinha, and P.W. Feiguth [9], and were analyzed for their effectiveness. They were able to develop an image of a crack in a pipe by eliminating noise and extracting the crack.

Their objective was to segmentize the image and get rid of distresses from the background. They found that their morphological approach to the segmentation of an image can be effective but proves difficult when classifying the cracks due to irregularities in the image. They understood

similar approaches existed for detection of linear features as other fields of study and industries require this type of detection. Since the medical field, for example, require this type of technology when viewing images of bone structures, cell boundaries and things of that nature, Sinha and Feiguth, looked to expand on these methods in their study [9]

The latter part of this study will look to investigate possible applications of the methods developed by Sinha, et. al [9] have used for image crack detection. The automation process to be more precise will be analyzed as it is desired to achieve a method that can be reliable, objective and time efficient to improve on current experience in-field operators who evaluate images based on their field expertise.

Background

1.1 Catastrophic Event in San Bruno CA

In the fall of 2010 a Pacific Gas & Electric Company's natural gas transmission pipeline ruptured in San Bruno California. This pipeline rupture unfortunately caused severe damage as it killed eight people, caused multiple injuries, and destroyed 38 homes . Figure 1 shows the damage [3]. The magnitude of severity of this prompted for the National Transportation Safety Board (NTSB) to come out to investigate to further determine the roots of this system failure. They took a 28-ft section of the ruptured pipe along with the exposed ends of the pipe still underground were sent to the NTSB Materials Laboratory in Washington, D.C., for examination [3].

Figure 1 - San Bruno pipe rupture [3].

The line that rupture was constructed in the 1940's in two phases and the material used were double submerged arc welded (DSAW) pipes. The pipes used were roughly 31 feet in length, had a diameter of 30 inches and a minimum yield strength of 52 ksi. Due to a subdivision development project PG&E had to relocate part of this pipeline a few years after it had initially been installed. It was during this relocation that the problem could be traced back to. This section was made from excess or leftover refurbished DSAW pipes. At the specific location to which the rupture could be linked to it was noted that, that portion of the pipeline was constructed with six short pipes that are referred to as "pups", Figure 2 shows the sections of these pups [3]. However, five of those pups, including the pup where the rupture would initiate, had properties that differed from DSAW pipe [3].

Figure 2 – Pipe Rupture [3]

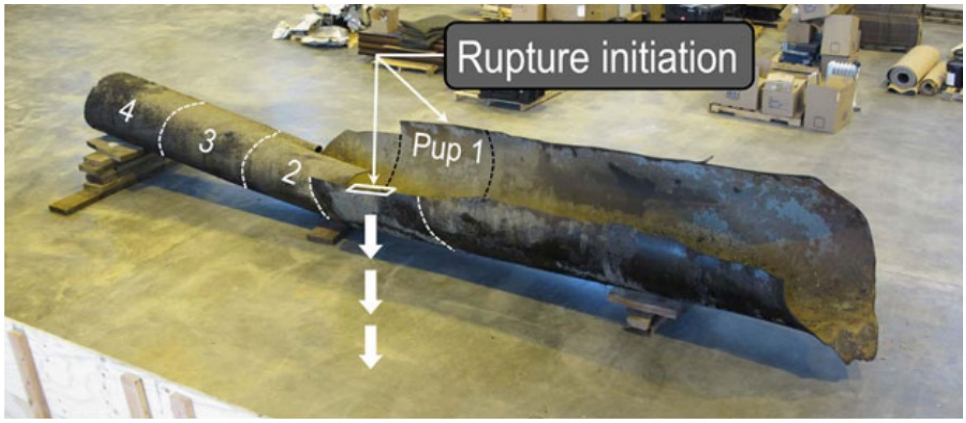
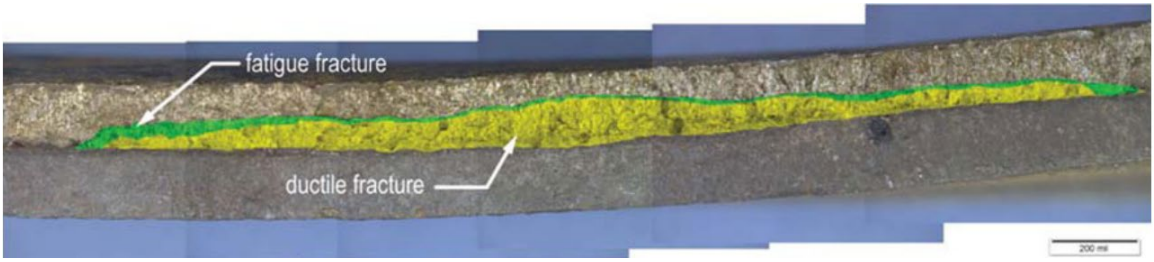


Figure 3 – Pipe fracture [3]



The main property differences about the pups when compared to a typical DSWA pipe were the welds, which even varied between themselves. More specifically they only had a weld on the outside and they were not welded on the inside which left a sort of splinter or slight fracture that was half the thickness of the wall, Figures 4 and 5 illustrate these welds [3]. The investigating team further determined from finite element analysis how the design and construction of these pieces negatively reduced the strength of the connection and increased the stresses it experienced. In addition to the welds being weaker, the actual tensile yield strength of the pups was found to be less than the 52 ksi minimum required proving to have inconsistent qualities compared to the typical DSAW pipe.

Figure 4 – Pup welds [3]

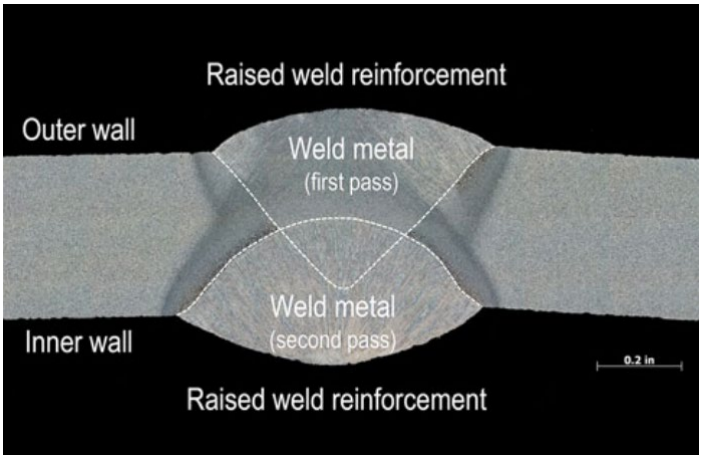
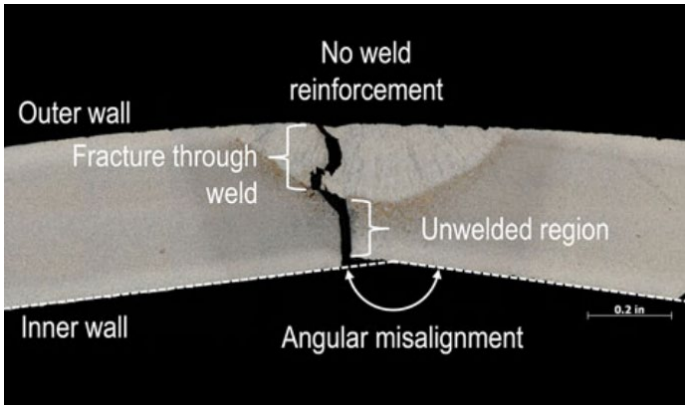


Figure 5 – Weld fracture [3].



The qualities of the pups that were used, to replace the section of the pipe that needed to be relocated, in hindsight left this section of the pipeline at a high risk of failure. So, the day the failure occurred it was due to electrical work that was being done on a transmission network. This caused a failure in the pressure regulation of the system which led to an increase of pressure and when this reached the pup with the incompetent weld this section of the pipeline ruptured. If the pipeline had been adequately replaced with a consistent DSAW pipe in accordance with the rest of the pipeline system, it is safe to say the rupture could have been prevented. This is because Line 32 was able to legally operate at 400 psi, and typical ran below 375 psi, the failure occurred at 386 psi, and records showed that in the previous decade before the rupture two instances had been

recorded for pressures above the 375psi mark. The third instance proved to be fatal as the pipes crack grew from ductile fracture and eventually burst from fatigue.

It is clear, that there was negligence from those responsible for ensuring the quality of the pipe being used was up to standards. There were several methods and tests in place to utilize that would have ensured that the specified pipe was capable to withstand the service loads that it would eventually be subjected to. The DSAW pipe has its own proof test, that it must pass, within its material specification which requires the pass withstand 1170 psi. Incredibly enough a proper simple visual inspection would have also been more than enough to have realized that the longitudinal seam of the pipe had irregularities when compared to a typical DSAW. Hydrostatic testing was also available at the time and could have also been used to identify the fact that the pipe sections in question would have not passed this test and therefore not gone into service. This type of test generally stresses pipes at 10% above capacity, for the city of San Bruno the pipes must be stressed at 50% above capacity unfortunately this was not a required test at the time of construction [3]. Radiographic inspection is also method of testing commonly used today but that was also not required when this pipeline was constructed.

1.2 Pipeline Network

The pipeline network in the U.S. is very extensive and extremely important for many reasons. As previously mentioned, the pipeline network carries natural resources that provide society with a source for energy, which when means that it is forever married with today's modern economy. Until further sources are energy are fully developed and replace or surpass naturals gas and oil as the main providers of energy, the consumption of these two fuels will be directly tied to the growth of the economy. As of 2018 the main source of energy in the U.S., is petroleum accounting for 36% of the total consumption for all energy sources, with natural gas being the next source at 31% [1]. Taken this into account, this highlights the importance of needing to have a dependable pipeline network that can operate at optimal levels. Crude oil has 80,740 miles of pipes and Refined Petroleum accounts for 62,719 miles, all stats are as of 2018 according to statistics provided by the U.S. Department of Transportation Pipeline and Hazardous Materials Safety

Administration. Natural Gas on the other hand has approximately 2.3 million miles of gas distribution pipelines, so as demonstrated these two sources are the lifeblood for energy in today's world [5].

Table 1 – Pipelines Miles by Commodity: Crude Oil [5].

Hazardous Liquid Pipeline Miles and Tanks by

Time run: 3/19/2020 4:05:32 PM

Data Source: US DOT Pipeline and Hazardous Materials Safety Administration
Portal Data as of 3/18/2020 10:34:13 PM

State: (All Column Values) **Fed/State:** (All Column Values) **Regulated By:** (All Column Values)

Commodity	Calendar Year	Interstate Miles	Intrastate Miles	Total Miles	Total Miles Miles of Gathering	Breakout Tanks
CRUDE OIL	2018	55,466.9	25,273.0	80,740.0	3,129.7	3,183
	2017	54,351.5	24,859.6	79,211.1	3,526.3	2,964
	2016	52,229.9	23,479.9	75,709.8	3,617.7	2,845
	2015	50,245.8	22,809.6	73,055.4	3,823.1	2,650
REFINED PP	2018	51,904.2	10,814.5	62,718.6	0.0	4,771
	2017	51,872.2	10,496.7	62,368.9	0.0	4,846
	2016	51,998.4	10,462.4	62,460.8	0.0	4.766
	2015	52,143.5	10,490.7	62,634.2	0.0	4,659

Table 2 – Pipelines Miles by Commodity: Natural Gas [5].

Pipeline Miles by Commodity – Gas Distribution

Time run: 3/19/2020 1:41:45 PM

Data Source: US DOT Pipeline and Hazardous Materials Safety Administration
Portal Data as of 3/18/2020 10:34:13 PM

State: (All Column Values) **Fed/State:** (All Column Values) **Regulated By:** (All Column Values)

Commodity	Calendar Year	Main Miles	# of Services	Service Miles
Natural Gas	2019	1,322,856.6	69,565,658.0	935,084
	2018	1,305,698.8	69,241,428.0	929,692
	2017	1,294,570.1	68,534,655.0	925,792
	2016	1,284,700.3	67,903,635.0	922,377

There is growing concern that these pipelines are subjected to heavy constraints as the demand for energy increases. If they are limited in the capacity they can carry, congestions issues arise which cause a domino effect on transportation costs. If resources become limited in one area due to low supply but on going or even increasing demand, then prices will rise. Whereas other areas may have the opposite effect in which supply remains constant and therefore prices do not drastically vary. This exacerbates the need for large expansion projects for pipelines to reach more regions. However as analyzed by Oliver in "Economies of scale and scope in expansion of the U.S. natural gas pipeline network", he found that large projects would create diseconomies and foresaw those smaller projects would be preferred due to having more manageable costs. The costs incurred including building and maintaining of all required facilities, so if a method to reduce these costs can be achieved it would help alleviate some of the pressure faced by the pipeline network.

These numbers and realities placed some perspective on the importance of these energy sources, but in order to find a solution pipes and the material they are made be closely evaluated. The two most used types of pipes are steel pipes and plastic pipes, making up roughly 98% of the miles in the pipeline network, with steel accounting for about 40% and plastic accounting for about 58% of the pipes [6].

1.3 Materials

Low-carbon or low-alloy steel is the main material used in the pipeline network for the transport of oil and gas. The microstructure of this material consists mainly of ferrite matter with satellite areas of pearlite [6]. This means that these pipes primarily consist of iron with small amounts of carbon and manganese [7]. These characteristics make this type of pipe very convenient due to their durability and ability to withstand the loads they will experience over their service life. Since these properties provide the ideal array of strength, toughness, ductility, and weldability they are preferred over other iron-based materials, like cast iron, or high-alloy steels. Additionally, a great advantage of low-carbon steels is that its properties, when utilized over a set range of temperature (-20^{o} F to 250^{o} F), will be the same when subjected to a tensile test as it would have been back in 1910 [6]. This material is not perfect however, and it still has its weaknesses as it can be vulnerable to oxidation when exposed to the elements of a natural environment. This issue can be overcome by applying a protective coat which can help combat corrosion, it is referred to as cathodic protection.

To further understand the behavior of pipes over their service life and their ability to perform at optimal conditions, the cause of pipe corrosion must be understood. Therefore, a pipe experiences corrosion when at a subatomic level, electrons begin to move away from the exposed surface of the pipe, hence beginning the corrosion process. This by consequence effects the pipes' ability to effectively carry the load as its strength is reduced, due to the iron becoming weaker from the oxidation process. For this reason, a protective coat that can help dampen the rate at which the pipe will corrode is necessary and periodic evaluations of the protective coat are required to ensure pipes are maintained. However even if proper maintenance is performed failure can still occur from fatigue and can lead to leaks from cracks or in the worst-case scenario a complete structural failure.

Chapter 2: Methodology

This chapter presents a summary of existing methodologies used in pipeline inspection.

2.1 Iyer and Sinha

As discussed in the Pipeline Network section, the vastly extensive pipeline network is especially important, and states and cities want to ensure investments made on them are protected with limited interruption to the service they provide. The pipes conditions are checked regularly to assess and determine the state they are in. This is assessment is by either CCTV, closed circuit television, or by using SSET, scanner evaluation technology, in order to obtain video and images of the pipes. The video and images collected would then be evaluated by an experienced operator. A few issues with this method are that it heavily relies on the subjectivity of the operator, who in contrast is at the mercy of the quality of the image. There is a void of an automated crack detection and classification process that can be objective and improve upon the in-field operators' subjective interpretation. This would allow gathering data of the pipes that is more in line with their reality, and therefore the most adequate method to restore the pipe can be applied.

Iyer and Sinha, proposed a crack detection algorithm, that would first improve the contrast of the image by the dark pixels from the "background", would then perform crack enhancement, to set up a detection process by applying filters with linear structuring elements to remove noise patterns. They adapted this method to different types of pipes of varying material and color. This is important as the existing pipeline network consists pipes of different types of material, additionally the pipes conditions can range from new too old to having residue or vegetation. Given these variances, Iyer and Sinha understood the challenges colors and background features posed in identifying cracks within images. They performed their study by comparing automatically detected cracks with manually plotted cracks (ground truth) [8]. They use what they refer to as the "buffer method", to match the automatically extracted cracks to the ground truth image. This method is further discussed in this paper, but it is essentially a predefined width around a crack that provides a reference for the analysis. Iyer and Sinha, established parameters that would allow them to obtain the probability of detecting cracks in images with varying crack patterns, background and color,

by creating a range of the size of the element and degree of rotation. This allowed them to find the optimal combination for these parameters. Their method performed well, they still experienced some issues, but when compared against other conventional methods used, it outperformed them. The analysis performed by Iyer and Sinha is very similar to those performed by Sinha and Feiguth, and their study is an expansion of this one.

Figure 6 – Original and Processed Images: Different cracks [8]

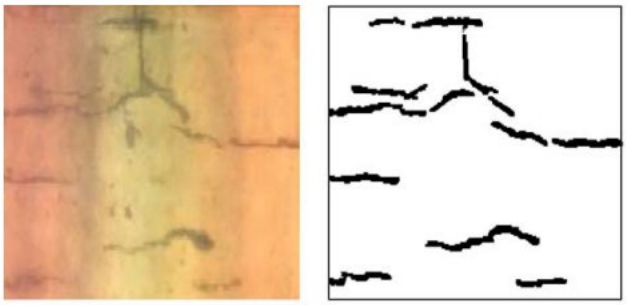


Figure 7 - Original and Processed Images: Linear Cracks [8]

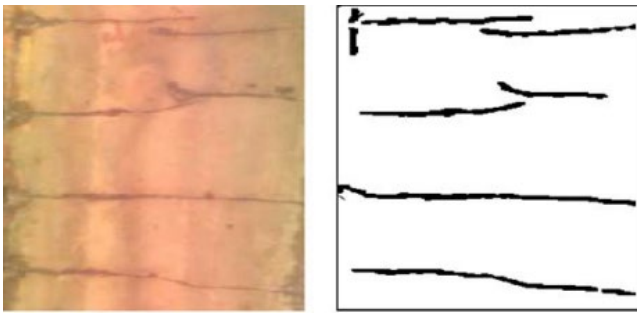
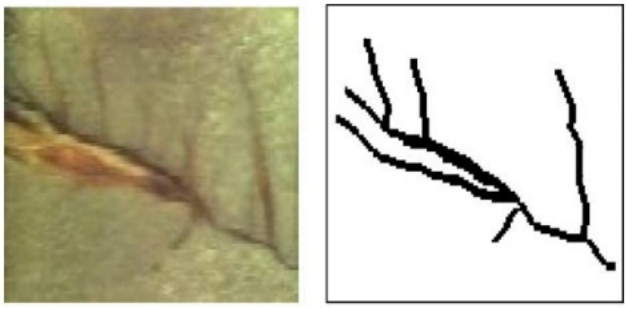


Figure 8 - Original and Processed Images: Branch Cracks [8]



2.2 Sinha and Fieguth

Sinha and Fieguth proposed to use to two-step algorithm to detect crack features in the segmented underground pipe images. The first step is to use statistical properties to extract a crack detail from the image. After this a "global cleaning" is conducted and is done so to merge sections to form a crack. This is a simplified explanation of how the process works but the algorithm Sinha and Fieguth implemented did prove to show successful results.

The initial crack detector that is used is based on a ratio edge detector, while the second detector arose from their research. The outcome of both detections is combined to create a response which then undergoes post processing procedure in order to produce a potential crack section. To further define the first crack detector is said to be the ratio of the average pixel values of two non-overlapping adjacent neighborhoods [9]. The response is defined as, one minus, the minimum coordinates of one regions sample mean over the other regions sample mean. This results in an overall response of the minimum coordinates of a region. Refer to equations one and two for these responses. D_1 is essentially a function of location and its parameters and when the response is significant the pixel is treated as being part of a crack. Sinha and Feiguth, found that this detector is accurate, however they noted that it was only the mean comparison component that performed adequately when the operator was not centered on an edge interface [9]. Yoram Yakimovsky from the California Institute of Technology wrote "Boundary and object detection in real world images", from which Siha and Feiguth used the variance comparison. This methodology assumed that cracks were junctions between points and this comparison is used to form a function of crack strength in the area [8]. If there is a large enough response, then the pixel is treated as part of the crack. This response can be seen in equation 3, Sinha and Feiguth had presented in their paper.

$$r_{ij} : r_{ij} = 1 - \min\left(\frac{\mu_i}{\mu_j}, \frac{\mu_j}{\mu_i}\right) \quad (1)$$

$$D_1 = \min\left(r_{12}, r_{13}\right) \quad (2)$$

$$D_2 = \frac{(\sigma_0^2)^{R_0}}{(\sigma_1^2)^{|R_1|}(\sigma_2^2)^{|R_2|}} \quad (3)$$

Figure 9– Original and Detected "major" crack before linking operations [9]

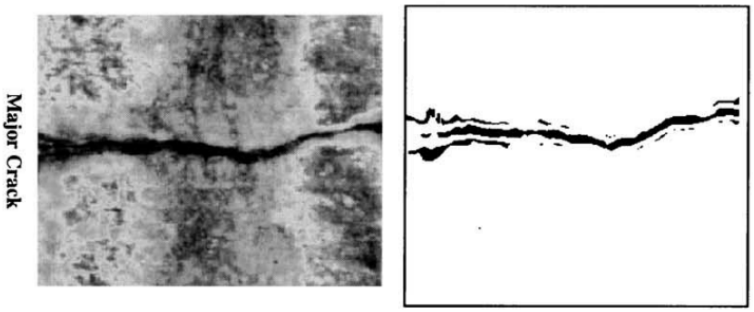
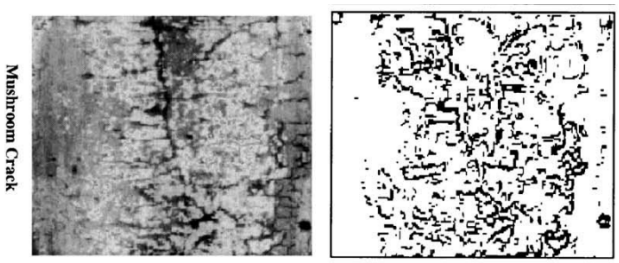


Figure 10 – Original and Detected "mushroom" crack before linking operations [9]



There are problems that arouse from utilizing just equations 2 and 3 to detect cracks within pipes. Sinha and Feiguth realized there were not an equation that was directly applicable to detect cracks in a dependable manner, they decided to fuse equations 2 and 3. They derived a fused response that was "chosen because of its indulgent disjunctive behavior for high values, its severe conjunctive behavior for small values, and its adaptive behavior in other cases" [8]. They determined that in order to detect as many cracks as possible these operators had to be applied in all directions. They also set conditions in which thresholds were kept constant even if the operator was being applied separately on different directions. Sinha and Feiguth limited the directions for detection and only conducted their study on the following directions, $0^o, 45^o, 90^o, and\ 135^o$. The result is a binary image, which still contains some noise, but the image can now be taken into a linking process.

Noise appears in the form of undesirable and irregular short edges while cracks can also still be disconnected from other edges. The linking process and cleaning up of the image aids in fixing this problem in order to create a better-defined image and therefore show cracks in a clearer manner. As with the initial crack detector methods, the linking process also has local and global

methods that are applied. Similarly, the local will focus on small regions and utilized the information previously gathered to continue finding pixels that share similar traits. The global will at the overall shapes and pattern of the cracks to piece together features that are shared. Sinha and Feiguth used a local linking process in their study with a focus on pixels that represented cracks. They identified the cracks and their characteristics, its length, starting and end points, and the direction in which it traveled. This allowed them to connect the points and link the tiny gaps between cracks. By doing so it also eliminated noise from outlying clusters with very minimal pixels. This process can still be improved as some cracks may be missed, however if enough cracks are identified when first detecting the cracks this process does provide sufficient results.

Sinha and Feiguth applied their crack detection method to concrete sewer pipe images from various regions of North America and evaluated the images by comparing the automatically detected cracks with ground truth from manual crack extraction [9]. They used what they refer to as the “buffer method” to examine the similarities between hand labeling and automatic extraction. This method predefines a width, first on the ground data using a morphological dilation, and then on the extracted data. If the data lies within the predetermined buffer, a match will have been identified. Sinha and Feiguth set up a series of equations establishing parameters from which they would use to determine the probability of detecting a crack. They used these parameters to assess the effectiveness of the detectors and their probability of detecting a positive result. They found that large regions reduced noise sensitivity but increased the probability of small cracks being missed [9]. In order to avoid missing small cracks, they implemented their crack detection over different size regions and were classified into three categories, minor, major, and multiple cracks.

Figure 11 – Major crack: Original image, truth image and filtered response [9].

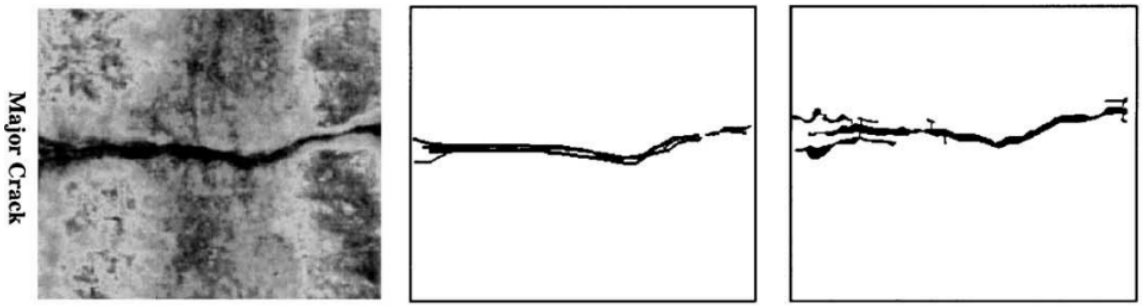
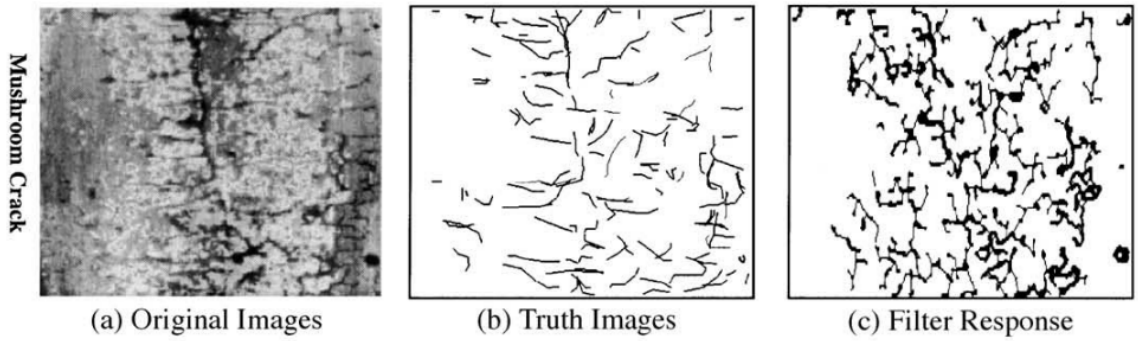


Figure 12 – Mushroom crack: Original image, truth image and filtered response [9].



To measure their methods capability to deliver positive results, they compared it against common crack detection methods, such as canny edge detection, and Otsu's thresholding. The Canny edge detector is an operator design to detect a wide scope of edges on images. Otsu's thresholding is an algorithm that will iterate through all possible thresholds to determine whether the pixels are in the foreground or in the background. Sinha's and Feiguth's results proved their method was more effective in identifying minor, major and multiple cracks. These results prove that their method can be used to detect cracks in images from underground sewer pipes. Utilizing the method, they proposed to implement it on PVC pipes would then be an achievement this paper will be looking to build for.

2.3 Survey and Evaluation of promising approaches

As brought up in the studies by Iyer, Sinha and Feiguth, there is a need for a time effective and inexpensive method to monitor and asses' structures, and they have proposed techniques to do so. Janashahi, Kelly, Masri and Sukhatme in "A survey and evaluation of promising approaches for automatic image-base defect detection of bridge structures" present an array of approaches that can potentially also be applied to other civil structures, such as underground pipes. They essentially studied the feasibility for image-based health monitoring systems, with their goal being to develop a robot that could independently go underneath bridges and capture images [10]. The robotic system coupled with an image acquisition system, would be designed to detect defects, classify them, and localize their position. Jahanshahi et.al [10] also discuss the importance of preprocessing and preparing the image for the extensive process it will undergo.

The following are found to be crucial for the purpose of this paper as it will set reference points from which to build on. Image registration, feature matching and feature detection, will all provide important steps for crack detection. Image registration can provide a process to compare two or more images of the same landscape or location, taken at different times and of different angles, and be able to analyze them for differences. Feature detection would be critical for the robotic system as it provides control points for identification. Feature matching will allow the detection of objects so it's extremely important that the appropriate feature is detected. Jahanshahi et.al expand on these topics in their studies and provide algorithms that have been successful for their desired application. They go on to discuss edge detection techniques as well as morphological approaches for crack detection in a similar manner in which Iyner, Sinha, and Feiguth investigated. However, their study proved inconclusive as to which was the better approach.

2.4 In-Pipe Robot

The in-pipe robot is an alternate method used to detect cracks and faults within pipelines. This methodology uses robots to maneuver and pass through the pipelines in order to visually detect faults. As previously discussed, the importance of pipeline infrastructure is extremely important as they provide vital supplies that act as a bloodline to all urban areas. However, this vast network is difficult to consistently and effectively maintained as pipes can easily be

overlooked and this is the area in-pipe robots look to fill. This method of inspection would be more advanced than the conventional methods currently used in practice and will be able to meet the detection requirements the current and future systems will demand.

The robots that are currently being used are not all created equally though, and the different types provide different advantages and disadvantages, these types are broken out into “Smart PIGs” (Pipeline Inspection Gauges), wheeled robots, tracked robots, and telescopic robots [11]. The difference these types of robots serve are their capabilities in the different type of pipe in which they can perform. For example, a wheeled robot may be able to move faster and provide service to a medium or large diameter pipe however they would be inadequate for a smaller diameter pipe. Another example of advantages and disadvantages can be seen in the wheel robot, as you can imagine this type of robot depends on its wheels and the traction in can get within a pipeline environment. This comes with issues though as they can become very robust have big surfaces areas that can impede its movement within a pipe and the added features needed for traction can cause control issues. A telescopic in-pipe robot is a motor-screw-driven structure with supporting wheel mechanism (SWM) that can provide large traction force under a stable motion state [11].

The telescopic robot being presented by Hua et. al, as one of the better options it still comes with challenges it still needs to overcome. Starting with the detection, surveying, and positioning of the pipe track line, the errors presented include positioning errors that begin to accumulate. The solution presented to resolve these issues was to use accelerometers, gyroscopes, odometers, and the attitude calculation method to gather all information required to determine the movement of the robot. Given these tools the telescopic robot is set out to achieve pipeline maintenance without damaging the pipe and help identify if a repair is needed due to a crack or rupture. In addition to the previously mentioned features this robot was also designed to carry a CCTV monitoring system and uses the images it captures to highlight fault areas. It then uses the K-mean clustering method to classify the pipeline fault [11].

A few key details to understand of the robot’s functions is that they can travel about 218 yards, its adaptable to different pipe materials and diameters, and it has a waterproof sealant that

can perform well under large amounts of sludge. A critical function it can perform is that it can measure its location under disturbances at variable pipe depths. The mechanical structure of the robot can be broken down into a handful of systems. The supporting wheel mechanism (SWM) that supports the robot and there is also a locking system that will lock the legs when they keep locking with the pipe wall. The control module (MB) which obviously houses the controls and positioning systems of the structure and is protected by the waterproof sealant. Lastly the structure that is referred to as the TM performs the telescopic action, which makes the robot move.

There was a special software that was designed accordingly to this model structure. This software is comprised of a several components, the control module, the video capture module, a calculation module, the display module, and the data management module. There are several functions, related to the operation of the robot, are ran and determined by this software but for this report the pipeline fault detection is the function of interest. As mentioned already a CCTV monitoring system is used for this purpose and transmits the video the robot is capturing. This feed of information passes through a fault a detection algorithm to help identify the fault. There is a system in place for image processing that can help filter bad data originating from noise and distortion caused by the image generation and transmission.

The robot uses the video it captures of the interior of the pipe and send it to the monitoring computer. Through image pre-processing any noise that is present is minimized or eliminated and a feature extraction process is then conducted. A potential issue that arises from obtaining these images and then subjecting them to a pre-process and post-processing procedures is that if the pipe has hazy or non-transparent fluids the CCTV module will capture unclear images. This creates unwanted effects when conducting a feature extraction process.

2.5 Roberts, Prewitt, Canny and Sobel

There were other studies conducted on other type of civil structures, such as concrete walls in which image edge detection. These studies were analyzed and reviewed to find applicable approaches that could be implanted on edge detection on pipe walls. Hoang et. al reviewed and analyzed, concrete wall cracks, with the Roberts, Prewitt, Canny, and Sobel algorithms. They utilize several functions and operations to enhance the image and help recognize the crack. Hoang

et. al describe the median filter approach as an effective method to smooth a digital image [12]. The authors describe the median filter approach as effective when analyzing unwanted noise in concrete walls. Hoang et. al explain how the median filter replaces each pixel with the median pixel within its neighborhood, which is set as a parameter to define the window size of the neighborhood [12]. The authors further explore other methods one of which is the *Roberts Edge Detection Method,* which they describe as an algorithm that can calculate the spatial gradient measurement of a digital image. It's an algorithm that can identify crack objects by locating regions of an image with high spatial frequencies [12]. This is done by first converting the image from the RGC format to the grayscale format. The authors further explain how once the image is processed, pixel values at each location are the approximated absolute magnitude of the spatial gradient of the original greyscale image [12]. Further this method requires a threshold parameter which functions by comparing the gradient values of a pixel with the threshold value. The absolute gradient value is obtained by combing two filters with an image neighborhood with the size of 2x2 pixels. If the gradient values of pixels in an image are smaller than the threshold value, then it is replaced with the threshold values [12]. This then provides an image that has identifies the edges. However, the Roberts method is very dependent on the use a hyperparameter, and the output of the crack detection is greatly affected on the parameter defined for the algorithm iteration [12]. Hoang et. al describe the *Prewitt Edge Detection Method,* as an edge detection method that uses two filters to estimate the derivatives of each location within an image. Similarly, to the Roberts method, the Prewitt method also requires a threshold parameter be defined in order to detect edges [12]. The difference with the Prewitt method is that the filters used are 3x3 matrices and the image neighborhood with the size of 3x3 pixels. The authors explain the *Sobel Edge Detection Method and* describe how it highlights edges by smoothing an image before calculating its derivatives. The Sobel method also uses a threshold value, which is used if the Sobel gradient values are smaller than the defined threshold value [12]. However, different from the Roberts and Prewitt methods, the Sobel method calculates the gradient magnitude by combing the partial derivatives of the x and y direction, with an image neighborhood. The partial derivatives are 3x3 matrices and the size of the pixels are 3x3. The Canny method is described by the authors as a multistep algorithm to detect edges. The first step is to apply a gaussian convolution to the image and then non-maximum suppression is performed. The Gaussian filter is presented as the convolution of the Gaussian

function and the function for an image neighborhood. The gradient is then calculated with the gaussian filter and the image neighborhood matrices. Unlike the previous methods described, the Canny method relies on two parameters. These parameters apply a threshold in which, the edges that are stronger than the upper threshold are determined as strong edges, and those that are weaker the lower threshold are suppressed [12]. Furthermore, edges that are between the upper and lower threshold are determined to be weak. The edge pixels that are not connected to strong edge pixels and are also part of the weak edge category, are also suppressed [12].

Figure 13 – Edge detection using Sobel method [12]

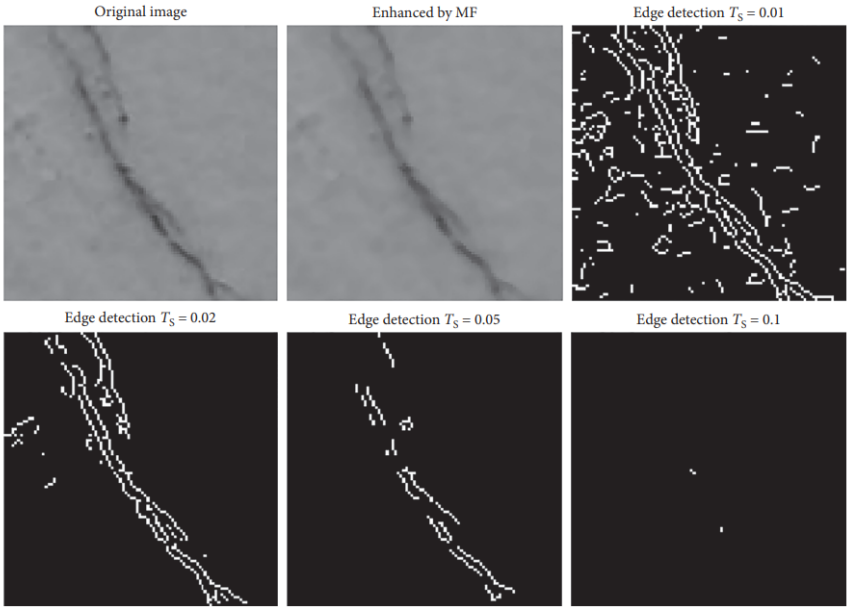
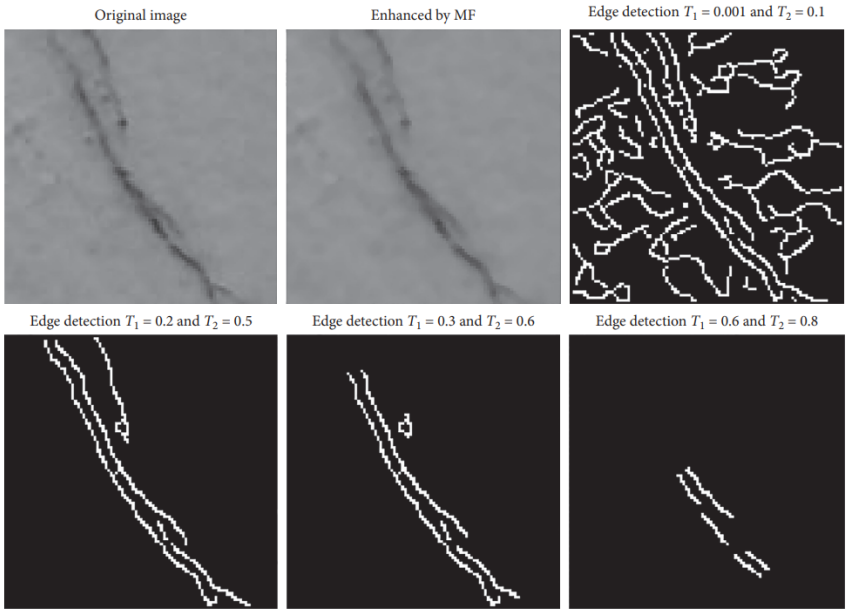


Figure 14 - Edge detection using Canny method [12]



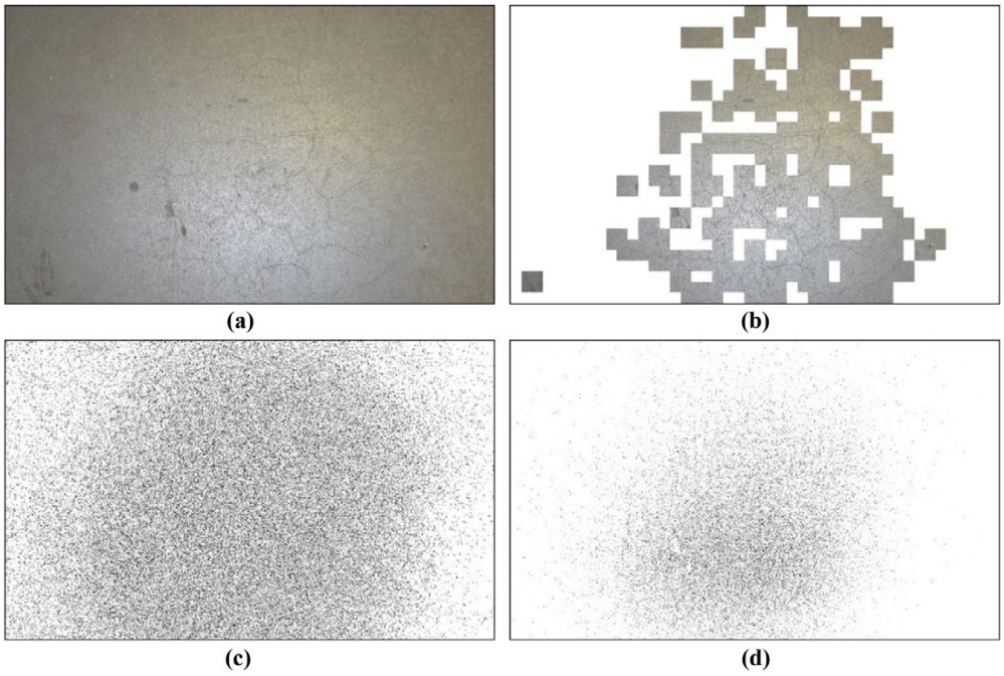
Hoang et. al experimental approach is based on detecting cracks and assessing those cracks. Hoang et. al (2018) explain the proposed "Metaheuristic Optimized Edge Detection model for concrete wall Crack Recognition" (MO-EDCR) and the three steps of the model, which are application of the median filter for noise suppression, edge identification using algorithms, and morphological operations to remove unwanted noise. The model is constructed by setting up three groups of parameters, the window size parameter, the edge detector threshold, and the threshold parameter used for removing small objects. The authors obtain results that showed that the best performing method for crack classification was the Prewitt method. The results provided an analysis of the effectiveness of each type of edge detection algorithm although the authors discovered some of them performed well under certain parameters but ultimately the Prewitt method was the best suited method for crack detection.

2.6 Convolutional Neural Networks (CNNs)

Cha et. al present a method to detect cracks by using convolution neural networks (CNNs). CNNs effectively capture topology images and can differentiate through large numbers of data sets [14]. Cha et. al explain how they us CNNs to build classifiers that would detect concrete cracks from images. The authors created a CNN using multiple layers, of inputs and outputs, which included, convolution, pooling, and activation. Cha et. al explain how the convolution layer performs three operations throughout an input array, the pooling layer reduces the spatial size of an input array, the rectified liner unit (ReLu) layer is used as a nonlinear activation function [14]. After the multiple layers are defined a data bank is generated and hyperparameters are defined for the CNN [14]. The authors describe how defining the hyperparameter is difficult, and a trial-and-error approach is used to define it as it is necessary for crack detection.

Cha et. al CNN method was tested by applying it on images taken of a concrete tunnel [14]. The authors obtained results that were highly accurate and almost identical to the validation process. Cha et. al explain how they compared their method to common methods being tested in the field, such as image processing techniques as described by Hoang et al. and describe how the CNN method can overcome difficulties these techniques faced.

Figure 15 – Thin crack, (a) original image, (b), proposed CNN, (c) Canny method, (d) Sobel method [14].



2.7 Action Camera

Lydon et. al developed a system in which a camera is used to record a section of a bridge under loading conditions at a minim frame rate of 25 frames per second [14]. The authors followed a three-step procedure to their study in which they would calibrate the camera, and then follow a feature extraction and feature tracking process. Lydon et al. used the SURF algorithm to process the extracting of features in images that needed to be tracked [14]. The authors use points detected from the extracting process to filter outliers and improve the output data.

Lydon et. al performed one laboratory test and two field tests to examine their proposed method of structural health monitoring [14]. The laboratory test was conducted by applying a central load to a simply supported beam in order to allow to determine the accuracy of the hardware system and the post processing methodology proposed [14]). The authors describe the setup of the laboratory test as a way to validate the measurements of the camera in a controlled environment and the results showed enough evidence that would allow for an accurate assessment in the field.

This validation from the authors was essential for them to prove, before they performed their field test as capturing good data was critical for their study. The first field test was performed on the Governors Bridge, on the west span of the bridge, and the loading conditions were performed under non rush hour traffic loading [15]. The authors explain how the displacements obtained from the data acquisition systems agreed and in line with the percent error found from the lab test. The field test showed positives results that correlated with the laboratory test which provides further evidence that this system can be effective.

The studies presented in this paper will provide the starting point from which this investigation will begin. As observed in several of the reviewed papers much research has acknowledged and tested edge detection methods, like Canny Edge detector, or have used a morphological approach to conduct their studies. They have found positive results utilizing these methods but have also expressed their limitations. Sinha has processed a method that performed better than the conventional methods and it is an approach that the continuation of this paper will further explore. The other challenges faced in the reviewed papers was also that acquisition of the images and creating a faster, more efficient automized method to obtain images for further processing. Robotic systems are the main solution for this issue and is an area of great interest for the field as in-pipe robots can be highly beneficial. These will be the areas of focus as this study continuous in order to find feasible solutions to the current issues faced.

Chapter 3 Analysis

3.1 Python

There are several image processing methods available within python that can help achieve desired post processed images. The libraries explored to process images were OpenCV, SciPy, scikit-image, NumPy, matplotlib and a few others, which are open-source libraries which are used for computer vision and machine learning. Open CV provides a platform in which computers can identify the objects within images and videos. This process helps understand the content of an image and can facilitate the classification and labeling of such objects. These are the two main task OpenCV performs classification and identification, in the former the program can be fed data of

certain objects and will subsequently classify new objects as part of that set [36]. The SciPy library is used to provide a high-level of algorithms built into functions which allow for faster computational power. These functions and commands can be applied across several domains and builds on extensions like NumPy. Scikit-image is another image processing library that contains several algorithms that assist with analysis, manipulation, object detection, and data reduction, of an image. The NumPy library has many of the fundamental packages for scientific computation, which provide fast operations for arrays, shape manipulation, sorting, Fourier transforms and several more [16]. This library is great for creating structures that will produce efficient outputs. The matplotlib library is also a great package that helps with plotting and visualizing data. These are a few other libraries used and those will be explored in the section below.

3.2 Method

The method of analysis that will be explored using Python will be to apply the libraries referenced above in conjunction with the theories reviewed in the sections above. That is the analysis conducted by Iyer and Sinha in which they performed operations that would identify joints, voids, and features that would represent a crack in an image. A similar processed which they employed is also how the Python code was structured. A color image would be loaded, which would then get processed and transformed into a grayscale helping provide an enhanced contrast of the image. After this process the grey scale image would then be run through a thresholding process and will provide desired segmented features of the image. From these features that are derived then the next step would be to identify the objects within the image which will then provide the desired properties in order to quantify the object and obtain a value for the object. The final goal being to obtain a set of values that provide measurements of the cracks that are being identified within the image.

3.3 Python libraries

3.3.1 Cv2:

The “cv2” module is used to import the OpenCV libraries, which is a library designed to assist with computer vision problems [17]. It provides several commands which can load images, perform differentiation, normal distribution, and morphology functions, rescale real values and

display output images. These functions will be reviewed further in depth in the following sections to understand their main purpose.

3.3.2. NumPy:

As previously mentioned, NumPy is good at storing and accessing multidimensional arrays for computation. It incorporates several fundamental array concepts, such as data structure, indexing, vectorization, and broadcasting [18]. It allows the operations such as Fourier transforms, linear algebra, random number generation, routines for shape manipulation amongst other operations. This library is often used in conjunction with other python packages like SciPy and Matplotlib. Since this library primary purpose, it to work with multidimensional arrays the most import object is called a *ndarray.* This object describes a collection of items of the same type and can be accessed using a zero-base index [19]. Each item in a ndarray is the same size as the block in the memory and each element in the ndarray is an object of the data-type object. Subsequently any item that is extracted can be represented by an object of one array scalar. A ndarray is an object that has a contiguous one-dimensional segment with an index scheme that maps each item to a location in the block.

3.3.3 Skimage:

Skimage is another open-source package in Python used for image processing, with several modules, sub-modules and functions aimed to facilitate image manipulation. These functions can be used to open, read, resize, rescale, alter the brightness, and many other capabilities that are used as part of the process to achieve the desired goal for your image.

3.3.4 Matplotlib.plylot

Matplotlib.plylot is a “MATLAB-like” collection of functions that provides different ways of plotting, the ability to create figures and generally acts as a GUI [20]. The “NumPy. Round” command is a function that rounds an array to the given number of decimals [21].

3.3.5 np. array

The NumPy array is a grid of values, all of the same type and is indexed by a tuple of nonnegative integers [22]. The shape of the array will be the tuple of integers given the size of

the array along each dimension. There are several ways to index arrays offered by the NumPy library, and several functions are available to perform different tasks depending on the type of manipulation needed.

3.4 Summary of Proposed Python Code

The python code is initiated by importing the libraries that will be being used, for this analysis the above-mentioned libraries were imported. Once these are brought into the editor, then all the functions that will define the analysis can be structured. The initial functions defined is non maximum suppression. Non maximum suppression is a method used to detect objects and select the best possible bounding box out of a set of overlapping boxes. The parameters used are an array of boxes and an overlapping threshold, in which the array is set up such that different bounding boxes are identified, and the threshold determines the overlapping area allowed of the bounding boxes. If the overlap exceeds the threshold, then one of the boxes is discarded. This process helps convert the image to zeros and ones before it is processed through edge detection functions.

The “im.read” function in OpenCV is one most common way, to load an image into the program and returns a NumPy array of n-dimensions. It requires two parameters to be defined in order to run, and those are the filename and the flag. The filename is written first and provides a string value which represents the path to the file. The flag is the second and is an optional parameter and defines the mode in which the image should be read [23]. So this function is used to load the desired image. The next few lines are used as defined threshold values that could be adjusted accordingly as needed. The following step taken was to apply a gaussian blur to the image. The gaussian blur filter essentially will smooth an image by averaging pixels values with neighboring pixels, in which the “target” pixel will have the highest value of influence on the average than pixels that are further away [24]. This filter helps remove high frequency components, or noise, in an image and is typically used before proceeding with other processing functions. The “cv.2guassianblur” function has the following input parameters: the input image, the output image, the Gaussian Kernel, the standard deviation in the x-axis and y-axis, and a border type. For the analysis the parameters that were used were the input image, the gaussian kernel, and the standard deviation. The input image was simply the image that is being worked

on, the gaussian kernel parameter is the predefined gaussian kernel function, and the standard deviation parameter is also predefined.

After processing the image through the gaussian filter, the processed image and the original image will be “subtracted”. Using “cv2.subtract” the pixels from two images can be subtracted and subsequently merge the two images. Once this function is applied then the Sobel operator is applied, this function will calculate the derivates from the image. It will compute the gradient of the image in the x and y direction and will compute its magnitude and direction. The magnitude is calculated by taking the square root of the summation of the gradient in the x-direction squared plus the gradient in the y-direction squared. The direction is calculated by taking the arctangent of the gradient in the y-direction divided by the gradient in x-direction. From this function it could be inferred that the edge direction would be positive for the transition from dark to white and negative otherwise [25]. So, the Sobel operator parameters are the input image, the depth of the output image, the order of the derivative x, the order of derivative y, the kernel size, a scale factor, a delta value, and a border type, the last three being optional [26]. In the analysis both the x and y directions are computed and returns a NumPy array of “float64”, which is a 64-bit floating point number.

After computing these values, the hypotenuse and the arctan of these two functions are calculated. The “np. hypot” function is used to find the hypotenuse and the parameters for it are the “x1” and “x2”, which in this analysis are the functions of the Sobel operator in the x and y directions. The resulting value is also a NumPy array which represents the hypotenuse. The arctan function in the analysis also uses the Sobel operators in the x and y direction to calculate the trigonometric inverse tangent and the result is also an array of “float64”. Once these functions are defined there is a threshold value that is established and is used to compare against the hypotenuse value prior to proceeding to obtain the image that identifies the edges.

The following step in the program was to create an if-else statement which executes both the true part and false part of a given condition and if the condition is true then the “if” block is executed but if it is false then the “else” block would be executed [27]. The intent of using the if-else statement was so the program would either produce a processed image with detection of the

cracks or it would apply a non-maximal suppression to obtain the desired result. For the “if” block the functions cv2.normalize, np. ones, and cv2.morphology are used to compare against the given statement. The cv2.normalize function changes the range of pixel intensity and helps increase the contrast features of an image [28]. The parameters of the function are the input image, output image, an alpha and beta values, and the normalization type. For the program the image that was computed with the Sobel process was used as the input image. The alpha and beta values were 0 and 255 respectively and the normalization type used was the “NORM_MINMAX”. The np.ones function will return an array if similar shape and size where the elements value is set to 1 [29]. The parameters for this function are the shape, order, and datatype of the returning array. Lastly the “cv2.morphology” function performs an opening, closing operations which are used to remove noise from the image [31]. The parameters for this function are the input image, the morphological operation, and the kernel matrix. The input image used was the processed normalized image, the morphological operation was “cv2.MORPH_CLOSE” and the kernel matrix was the “np.ones” output array. The “else” block contained the non-maximum suppression function, defined at the beginning of the program, and also contains the “np.ones” and “cv2.morphology functions. For most images used in the analysis it was the “else” block that returned the processed image with the identification of the cracks, that yielded the best results.

After the image has been processed it is loaded again and remains in pixels therefor a scale is written to provide the conversion from pixels to inches since ultimately the goal is to obtain results in inches. The scale is determined by the dot per pixels of the image and a simple conversion is applied to obtain the scale. The scale has to be adjusted depending on the image properties. The images used in this analysis varied therefor the scale was adjusted accordingly to each image. Once the scale is set the next step is to use the skimage library to import the “threshold Otsu” command.

The threshold Otsu function is used to find a threshold via Otsu’s method, which is a binarization algorithm. This function uses a greyscale image to find a thresholds value between the peaks of a grayscale histogram [31]. Otsu’s thresholding method iterates through all possible threshold values and calculates a measure of spread for the pixel levels on each side of the

threshold for the pixels that fall on either the foreground or background [33]. The goal of the function is to identify the threshold value for which the sum of the foreground and background spreads is at its minimum. As mentioned, the algorithm will iterate in search for the threshold that minimizes the within-class variance, which is defined as a weight sum of variances of the foreground and background [33]. Within the python code once this function is ran it will produce a threshold value which provides the basis on which a thresholder can be generated.

The thresholder image will equal the input image, where the pixel values of the image are greater than or equal to the threshold that was just determined using the threshold Otsu function. The pixel values that are of interest are the light pixels as those are the pixels that represent a crack. This function will create an array in python of, “true” and “false” values essentially, zeros and ones since a logical operator is being used. The threshold image, in most cases of the analysis had the threshold value being greater than the input image. The thresholder image is then plotted and shows how the pixels that weren’t within the threshold are no longer shown, which provides a cleaner background and image to work with.

The next step in this process is to remove any excessive regions or objects that are touching the edge of the image. This is done so that the end results are not skewed as these objects can potentially generate inaccurate results. The general idea is that an image will capture an entire crack, if only a portion of the crack is captured within the image, then it will be difficult to quantify the true measure of the crack. Therefor this process is done by using the skimage library and the segmentation module to import the “clear border” function. This function does just that, it will clear objects that are connected to the border of the input image. Similar to the thresholder image, this function will return an array of “true” and “false” values.

After the thresholding process is conducted and the objects that were touching the edge are cleared then the next step is to label the objects within the image. The method in which this will be achieved is to label connected regions of an array, that is all the pixels within a feature or object will be given a label. This will be achieved by using the “measure.label” function which is also a function from the skimage library. Essentially this function will label connected regions of an integer array [34]. This is done when two pixels are neighbors and have the same value, in

which the value refers to the maximum number of orthogonal hops that will consider a pixel a neighbor [34]. The required parameters of this function are the input image, the background, the return number of labels and the connectivity, the latter three being optional. For the analysis only the input image and the connectivity were used. The connectivity parameter, as mentioned above, is the maximum number of orthogonal hops to consider a pixel a neighbor. The return value is a ndarray, a labeled array, in which all the regions that were connected have been assigned the same integer value [34]. This labeled image is then plotted in order to distinguish the different labels that have been identified.

After finishing labeling the next part of the process is the most critical as now computation of the all the properties is analyzed. The function that will be used to obtain all the measurement is “measure.regionprops”, which is function that will measure properties of an input image. This is a great function as it has the capability of measuring several things such as area, bounding box areas, axis lengths, perimeters, centroids, coordinates, eccentricity, diameters, image intensity, moments, amongst other things. For the analysis the desired measurements that needed to be obtained were the area, the bounding box area, the equivalent diameter, the major axis length, the minor axis length, the mean intensity, and the perimeter. The most critical measurements were the area, the major and minor axis lengths and the perimeter and the reason for that will be further discussed in the results section. The parameters of the function are the input image, the intensity image, a cache, the coordinates, and the properties. The cache and the coordinates are optional so for the analysis only the input image, the intensity image and the desire properties were used. The area parameter will return an area of the labeled region scaled to the measure of pixel are found within the region. The bounding box area is similar to the area parameter, but this parameter will return the area of bounding box. The major axis length will return the length of the major axis length of an ellipse that has the same normalized second central moments as the labeled region. By contrast the minor axis length is the minor axis of the ellipse that has the second normalized second central moments as the region. The equivalent diameter is calculated by measuring the diameter of a circle with the same area as the labeled region. Lastly the perimeter is found by approximating the contour of a line

that goes through the centers of the border pixels using a 4-connectivity. These parameters will be the ones that will be saved to the output date tables for analysis.

Once the parameters are set the following portion of the python code is set up to format and capture the data in organized tables to make it simple to read the results. The panda library is imported, and this library is a tool used for data analysis and works well with the matplotlib and NumPy libraries. Pandas has several structures and features, such as date frames, time series analysis, data visualization, amongst a few others which allow for different methods of data analysis. For the analysis conducted in this research a data frame was used, which store data in a table format of row and columns similar to an Excel spreadsheet. Therefor the function "pd.DataFrame" is used and it's a two-dimensional tabular data with the following parameters: data, index, columns, dtype, copy. For the analysis only the data parameter was used and is defined as the output of the measure.regionprops function. Now that the data is formatted the next step that was taken was to filter out unwanted data or in this case data of small regions that weren't of interest. Therefor a filtering function is written so that any region with a small area is deleted from the tabulated data. This is done simply by setting up a greater than or equal to function with the desired size of area set as the benchmark. After these unwanted regions are deleted, the next step taken is to convert the data into inches since it is all shown in pixels. This is simply done by multiplying each parameter accordingly by the scale, such that areas are represented in inches squared, and length and diameter measurements are represented in inches. Lastly the tabulated table is exported and saved as an excel file, this is done so that results could be saved and compared if any parameters, functions, or thresholds are changed in the code.

Chapter 4. Results

The python code was tested on several images, both from a data base of images and from images taken with an iPhone XR under control conditions. The images acquired from a data set are obtained from the METU Campus Buildings by Caglar First Ozgenel [36]. The images are of cracks in concrete and are high resolution images and the entire data set consisted of 4000 images with size of 227x227 pixels. The controlled images were taken of a 6" PVC pipe with hand crafted valleys to represent cracks as well as images of a plastic school binder with hand-

made cuts to exaggerate the representation of a crack in an image. These images varied in the properties they each had some were larger than other, however the DPI (dots per inch) were all the same since the same camera was used. All images were evaluated for their crack patterns and their background features as this determined how effective the first portion of the python code was. For the data set of images there were several that had a lot of noise in the background when converted to the gray scale, so it was difficult to identify which were the "true" cracks. This was in spite of adjusting thresholding factors, therefor these were discarded for the analysis.

4.1 Controlled Images

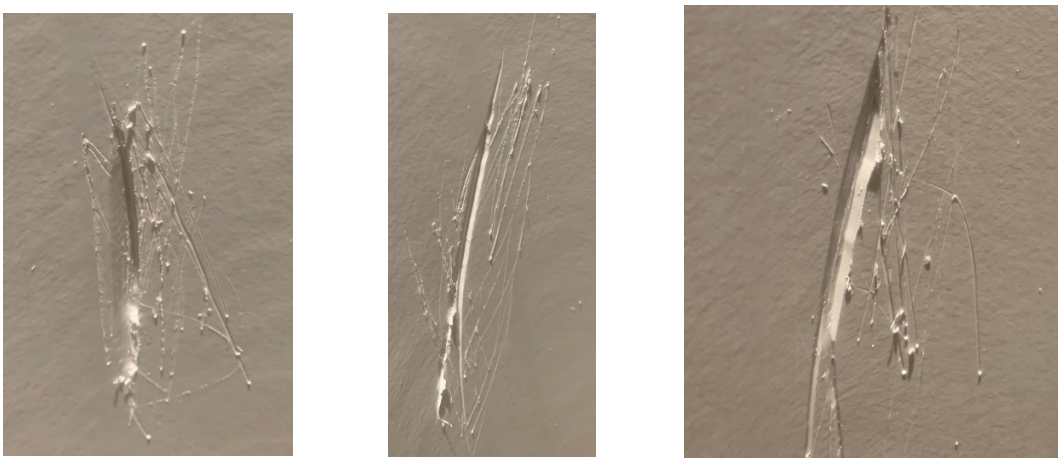
The control images used were that of a PVC pipe and of a plastic binder. The PVC pipe is standard pipe used for sewage pipes, water main lines and other applications for both commercial and residential use. The intent for the use of this type of pipe is that a similar situation from the "real world" could be applied to simulate failure. The challenges faced using this pipe, brought forth the need to explore other image options for use in this analysis. The plastic binder used is the standard plastic binder that could be easily found at a typical retail store that sells school and office supplies.

4.1.1 PVC Pipe Image

As mentioned, the PVC pipe used was a 6" inch pipe and the valleys in the pipe were created using a combination of a pocket-knife and metal carving tool were used. Using these tools, a series of "cracks" were created to provide areas within the pipe for use in the analysis. Several cracks were created in order to have a several samples to experiment however not all were useful once processed through the program. The challenges encountered were that due to the nature in which the cracks were created the images captured would have a lot of noise and require a lot of processing. This was very inefficient as the program would have to be readjusted or recalibrated for each image to process an image with the most minimal noise as possible. If the noise was deleted in the first portion of the program in which the non-maximum suppression, Gaussian blur and Sobel process were executed then the second portion of the program would either generate too much data or not enough. This was due to the set of thresholds that are built into the program, so if the threshold keeps the noise that wasn't processed in the first part of the

program, then the crack would not be clearly identified. If the thresholds were set such that the noise was blurred, then there were several images in which even the crack was blurred and therefore not identified.

Figure 16 - PVC pipe with manufactured "cracks". (a) PVC1.1, (b) PVC2.1, (c) PVC3.2-2

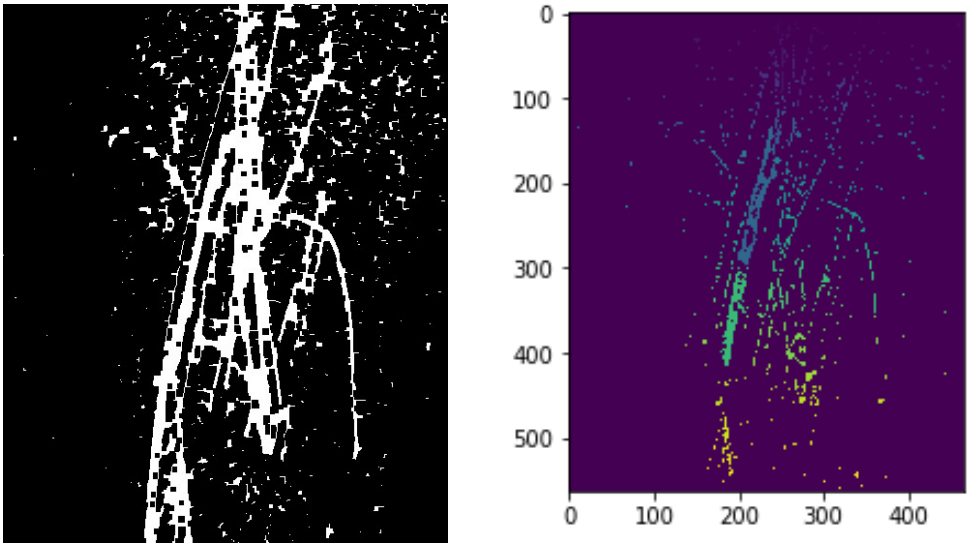


(a) (b) (c)

The image labeled PVC3.2-2 was an image of a manufactured crack that had been created that was approximately 1/4" in width and approximately 2 ½" in length. This was considered to be a good size crack as previous cracks made were either too small or too big which presented other challenges that will be further discussed below. The PVC3.2-2 image when processed through the first part of the program the resulting gray scale image does identify the crack that was created, however it does still contain noise. This noise can be addressed by adjusting the first threshold value within the program. This threshold was the product of the hypotenuse of the Sobel Response output and a value that could be adjusted such that the output image had the right contrasting features. If this threshold was a value approximately 0.2 or smaller the contrasting features of the image would no longer be of value as the cracks would be lost. If the threshold value was greater than approximately 0.6 than a similar situation would occur with the contrasting features of the image, and the cracks would no longer be identified. Therefor there was a very small range of the threshold that obtained the desired gray scale image and that was found in the range from approximately 0.3-0.5.

The 2nd part of the program would look for the crack as well in order to quantify it and produce results that would provide a measure of the crack. The image would also go through additional thresholds such that further noise reduction could be conducted. The resulting image was that of an image that had the desired crack identified with a bit of smaller cracks and noise also identified in the surrounding regions. The major crack that was had been hand crafted was split into two major areas once processed through the program. The resulting minor axis length, which in this case was the width, of one of those regions which will be called region “90” was 0.264 inches while region “232” was 0.217 inches. The former result had a 5 percent error, and the latter had a 12 percent error, while the average of the two results had a 3 percent error. The major axis length, which in this case represented the length, of region 90 was 2.66 inches while region 232 was 1.75 inches. These results had bigger percent errors as the overall length of the crack was 2.5 inches therefor the results obtained for this feature of the crack were inaccurate. This is partially due to the small areas that are lumped as part of these regions and incorporated into the results.

Figure 17 - Filtered images, (a) Sobel edge detection, (b) Otsu edge detection.



(a) (b)

Figure 18 - Crack dimensions (a) width dimension, (b) length dimension.

(a) (b)

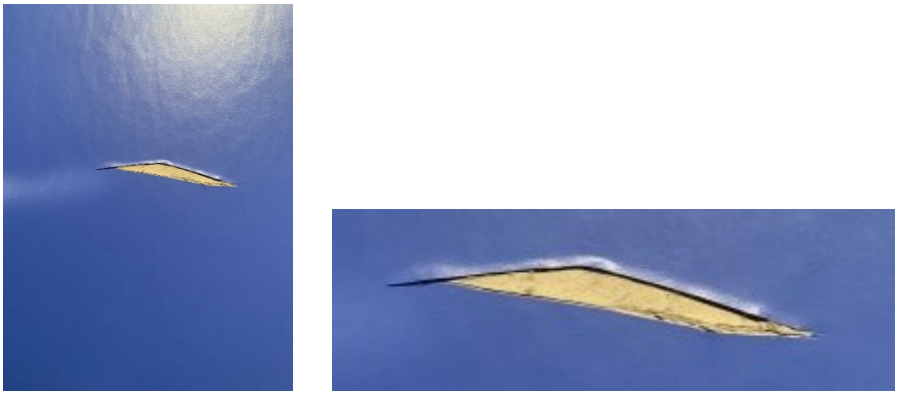
Table 3 - Results of PVC image obtained from Python code

	Label	Major Axis Length (in.)	Minor Axis Length (in.)	Perimeter (in.)
47	48	1.085117404	0.064888524	1.771348807
298	299	2.441209598	0.285918575	10.27263611
480	481	0.983384048	0.168143646	1.757458807
610	611	1.722150604	0.221992987	6.365174173
726	727	0.669390914	0.331078307	1.686203908
837	838	0.677877088	0.099635392	1.533329225

4.1.2 Plastic Binder Image

Since the PVC pipe image was the first image that was evaluated and mixed results were obtained, a secondary object was used for observation and experimentation. The program needed to be verified for its effectiveness before proceeding on the data set of images. Therefore, a plastic binder was used with the idea an image of truly flat surface could be captured using the camera available. Any curvatures in the pipe when the image was being captured using the camera available for the experiment could have potentially been causing some of the error experienced in the PVC pipe images. A higher quality camera could have the ability to incorporate any curvature to images of pipe walls. The plastic binder image represented something closer to a concrete or wood shear wall in which a flat surface is being captured for observation.

Figure 19 – Binder image (a) original image, (b) cropped image

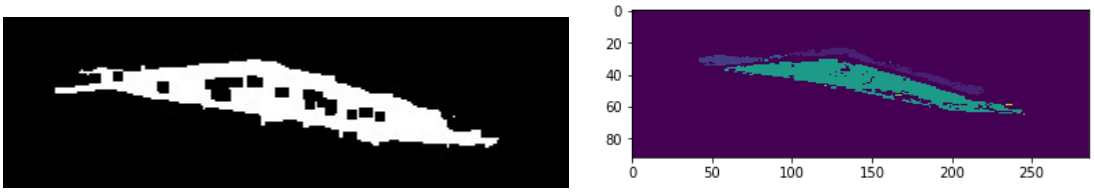


(a) (b)

The image labeled Binder (1) was an image of a manufactured crack on a plastic binder in the shape of a triangle with its major length being approximately 2 3/8” and major width being approximately 1/4”. The crack was made of simple shape with the intent that the program should find it simple and easy to process and produce accurate results. The image when processed through the first part of the program had much less noise than the PVC pipe image. The noise that it did contain was addressed by calibrating the threshold, however this did not have to be too extensive as the range in which good contrasting features were larger than with the PVC pipe image. The threshold range in this case was from approximately 0.1-0.7, which meant that the constraints faced with the PVC pipe image were no longer present. Those being having to modify the threshold rigorously, such that the crack remained visible with the most minimal noise possible. Consequently, the second part of the program also had better results.

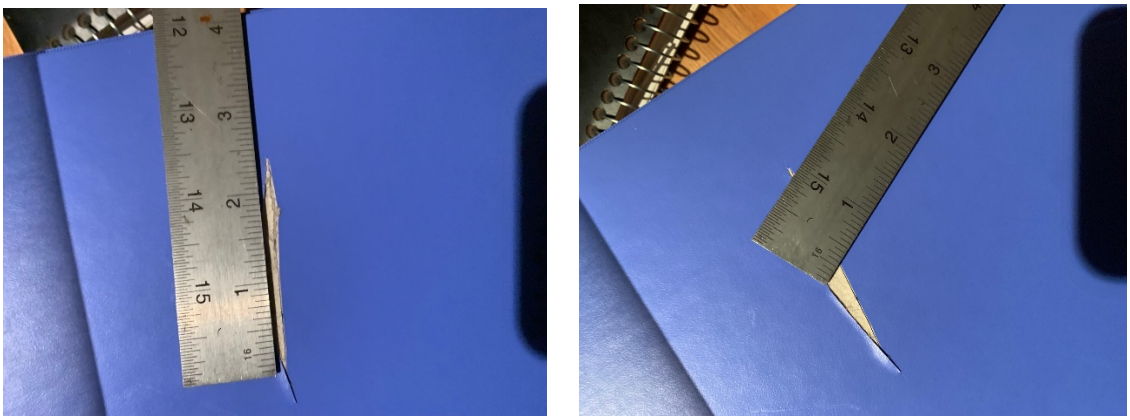
The 2nd part of the program produced much better results for this image than the PVC pipe image. The resulting image had the desired crack identified in one large region and only a couple other regions were identified. These aren’t attributed to noise but rather features of the object caused by creating the crack with a pocket-knife. The edges of the crack had plastic that was deformed due to the cutting and therefore the program also identified these areas and quantified them. The major region, which is labeled as region 6, had a minor axis length of 0.236 inches. This meant it had a 5 percent error, producing identical result values as the previous image. However, in this case the major axis length, had much better results. The resulting value for the major axis length was 2.357 inches which meant it had a 0.7 percent error. These results were positive for the research as it validated the program and proved it could be effective for use on images. The next step would be to test the program on the data set of images that were obtained from Ozgenel.

Figure 20 – Binder Filtered images, (a) Sobel edge detection, (b) Otsu edge detection



(a) (b)

Figure 21 - Crack dimensions, (a) length dimension, (b) width dimension.



(a) (b)

Table 4 - Results of Binder image obtained from Python code

	Label	Major Axis Length (in.)	Minor Axis Length (in.)	Perimeter (in.)
0	1	2.047500161	0.128149	3.694404
1	2	0.546047764	0.08729	1.27549
2	3	0.132139726	0.037126	0.192063
5	6	2.356844119	0.235975	8.55836

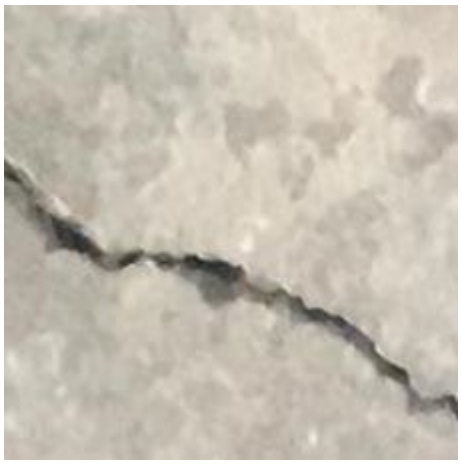
4.2 Data Set Images

The data set of images used for the analysis were chosen by image type, as there were several images that were very similar to each other. This meant that instead of running the program on all images, an image from a group of images that shared characteristics and features was chosen for the experiment. This set of images could represent a realistic data set captured for processing when evaluating pipe health as it is being monitored by a private or public agency. Therefore, it was important to run them through the program to understand if it was feasible for real life application. Four images from the data set were processed and the results obtained were mixed.

Figure 22 - Data set of images used. (a) Image 00001, (b) Image 00026, (c) 00007, (d) Image 00068.

(a)

(c)



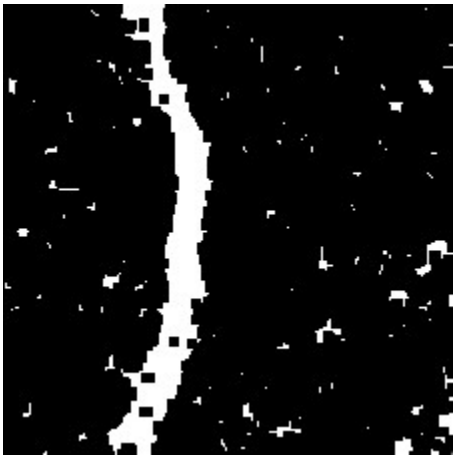
(b) (d)

The first image processed is labeled “00001” and the features, of this image has a crack running north-south within the image and with average crack size width relative to the rest of the data set. Unlike the controlled images, the crack size is unknown beforehand and could not be checked with a measuring device. This first image faced similar constraints with the thresholding value used within the non-max suppression portion of the program. If the threshold value was too small the crack would be identified however the edges of the crack would lose their definition, so they were no longer clearer marked. If the threshold value was increased the edges of the crack would be clearly identified. However due to the depth of the crack has a darker shade or darker pixels, when processed through this part of the program the area of the crack in which the pixels were at their darkest then when processed this area would be blurred. This meant that the program would not recognize it as part of the crack since it was far away from the edge.

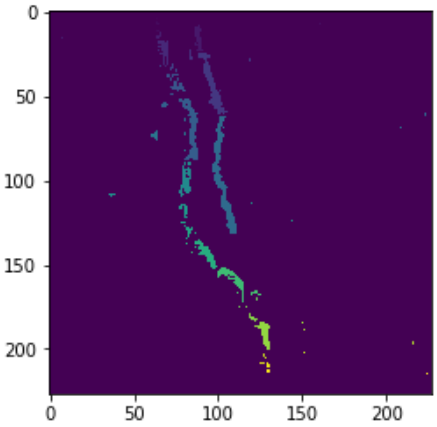
Given this first filtering process didn’t clearly outline the crack the second part of the program also did not fully recognize the crack. However, it did outline the crack edges very clearly, but the regions were broken apart and weren’t continuous so instead of the program providing a single output value several output values were given. So, for this image the crack width could not be determine directly from the program nor the crack length. These values could be obtained by performing post processing operations by analyzing the output data and finding the corresponding values that represent the crack length and widths. But doing this wouldn’t not

be efficient as the goal would be to provide these values by identifying one or two large areas that capture most of the crack.

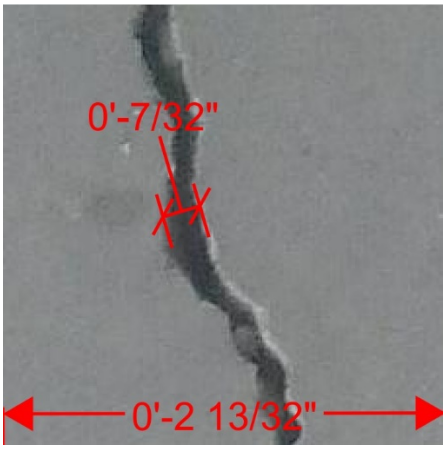
Figure 23 - Filtered images, (a) Sobel edge detection, (b)Bluebeam scaled dimensions (c) Otsu edge detection.



(a)



(c)



(b)

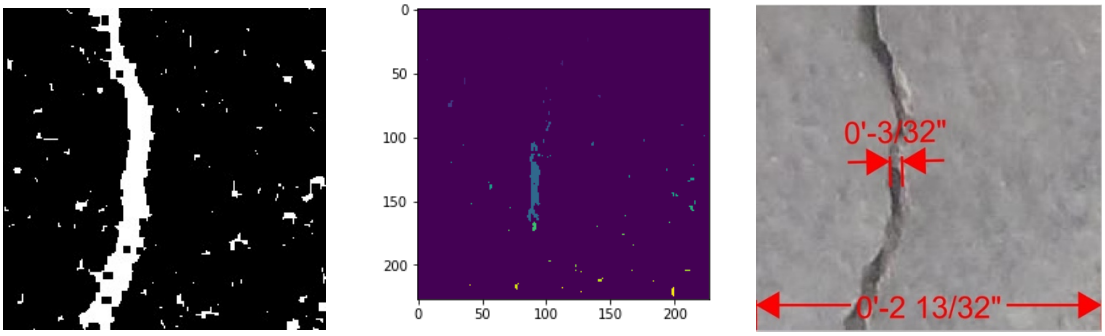
Table 5 - Results of image 00001 obtained from Python code.

	Label	Major Axis Length (in.)	Minor Axis Length (in.)	Perimeter (in.)
14	15	0.07811842	0.045101307	0.118901863
15	16	0.088174506	0.067671576	0.256850451
17	18	0.482731872	0.081522115	1.065188606
20	21	0.848388508	0.093806656	1.647863801
23	24	0.062950618	0.52885154	0.167042157
28	29	0.200001725	0.66724498	0.528802921
37	38	0.229191217	0.060657764	0.554692097
41	42	0.267187289	0.082181265	0.653130116

A second image was process labeled 00017 and part of the same type of image from the data set as the previously processed image 00001. A similar image was processed to further explore if similar results would be observed or if better results could be obtained. The 00017 image produced a better gray scale image as the crack depth for this image was not a significant factor as the previous image. The threshold range is 0.25-0.5 and the output image within this range clearly identifies the crack within the image. However, the Otsu threshold wasn't as successful at identifying the crack, it does identify a portion of the crack that was sufficient to provide valuable results. The crack width for an area of the crack that was identified was 0.0929 inches. In order to check the validity of this value Bluebeam software was used to get an approximation of the crack width. It's an approximation due to the limit Bluebeam sets for how accurate the scale can be, which is 1/32 of an inch. So, to check the value obtained with the Python program the DPI of the image is identified, in this case DPI value for all the data set images was 96 DPI. Therefor this value is used is to convert to inches and then the value is used to scale the image within Bluebeam. Once the image is calibrated within Bluebeam the same region that was identified through the program is measured to check the value for accuracy. For the case of the 00017 image the Bluebeam measured value of the crack width was 3/32" which is 0.09375 inches. This meant that the crack width length obtained relative to the Bluebeam value obtained had a percent error of 0.8. This represented a very good result for this image as the crack width output value can be assumed to be accurate. However, a similar limitation still

remained which was the crack length was not clearly identified with the program therefor an output value was not obtained for the full length of the crack.

Figure 24 - Filtered images, (a) Sobel edge detection, (b) Otsu edge detection, (c) Bluebeam scaled dimensions.



(a) (b) (c)

(b)

Table 6 - Results of image 00017 obtained from Python code.

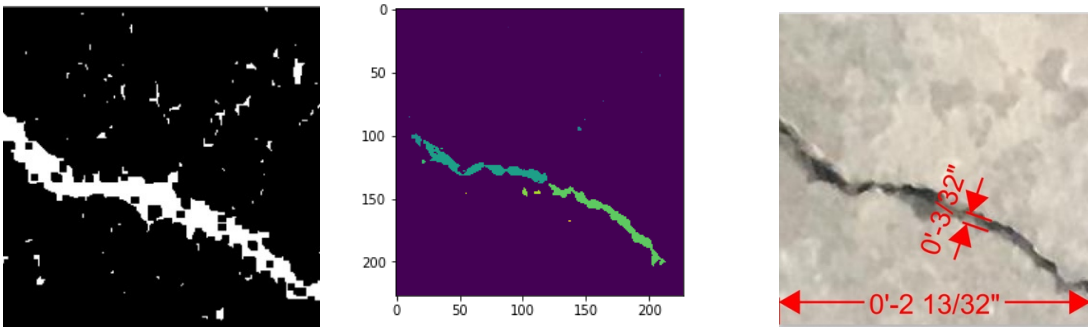
	Label	Major Axis Length (in.)	Minor Axis Length (in.)	Perimeter (in.)
10	11	0.062427047	0.030425374	0.106327431
15	16	0.095134926	0.018811776	0.125005
20	21	0.693142299	0.092951344	1.722199919
38	39	0.058327608	0.028173756	0.117114588
42	43	0.076072858	0.03164821	0.142263451

Given the results for these two images, the next image that was processed was image 00026 which was similar in characteristics to image 00001 and 00017 but this image had a crack that ran east-west in the image. The crack width was similar in size to the previous two images so the expectation was that the crack width value could be obtained. The threshold range for the gray scale image was also in the region of 0.25-0.5 similar to image 00017, and the output gray scale image did produce an outlined identification of the crack. Similarly, the Otsu threshold produced an outlined image of the crack with two major regions representing the crack. This was

a different result from the previous data set images and was in turn a positive result when compared to the other two.

The output value for the crack width of the image was 0.16 inches and the crack length was 2.549 inches. The crack width that was scaled from Bluebeam was 5/32 inches which meant that the relative percent error was 2.4. This result was consistent for the crack width when compared to previous images. The crack length was 2.549 inches but was obtained from adding the two major regions that were identified and spanned the length of the image. The Bluebeam value was approximately 2 13/32 inches which meant the relative percent error value was 5.9. This value can be used for reference but is not a true result as the crack length identified had the areas at the edge of the image deleted, this was a function in the program. Secondly the processed image had two different major areas which meant the crack had a break from one area to the other, and although minute there is a portion of the crack that isn't accounted for.

Figure 25 - Filtered images, (a) Sobel edge detection, (b) Otsu edge detection, (c) Bluebeam scaled dimensions.



(a) (b) (c)

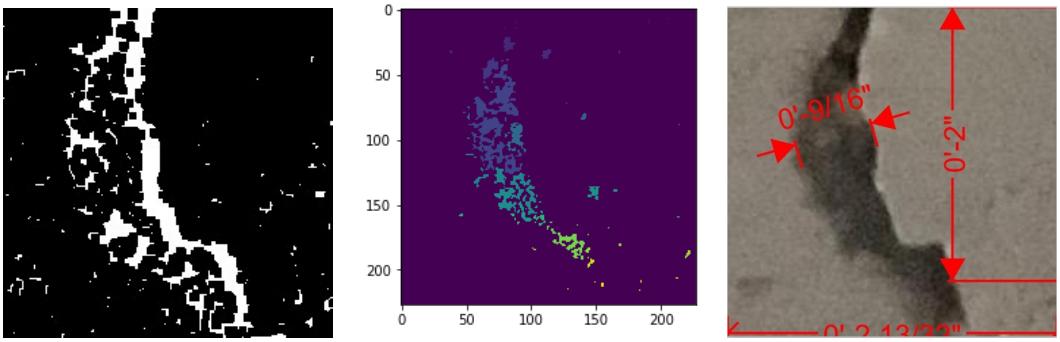
Table 7 - Results of image 00026 obtained from Python code.

	Label	Major Axis Length (in.)	Minor Axis Length (in.)	Perimeter (in.)
7	8	0.108857425	0.05809096	0.278424764
8	9	1.291044553	0.160050545	2.80403468
11	12	1.257691894	0.170155227	2.819877013
12	13	0.068132104	0.037839079	0.150523019

Since the third image provided improved results a fourth image was processed but now an image with different characteristics from the previous images was used for observation. Image 00068 was a crack that ran north-south in the image but had a significantly wider crack than the previously processed images. Given the programs ability to identify the crack width of the previous images the expectation was that similar results could be obtained. However, the crack depth, or darkened pixels of the crack could potentially distort the output values, similar to the results obtained for image 00001. The output gray scale produced an image that does identify the crack, but the crack depth issue was present as regardless of how the range of the threshold was adjusted part of the crack width would be blurred and not fully captured.

The Otsu threshold provided an output image that did capture a larger region of the crack width as well as the crack length. In this case the crack depth, the dark pixels are not completely omitted and are captured as part of an area. The major area identified was in the northern region of the image and the output crack width value was 0.386 inches. The Bluebeam value for the crack width at its widest was 9/16 inches so this was a significant difference from the output value of the program. This can be attributed to how the program calculates the minor axis length of the identified areas. The crack length could not be obtained directly from one or two values since the output values were broken out into ten major areas and therefor require these areas to be added. The summation of these values is 2.57 inches which relative to the image size and crack length, which is approximately 2.41 inches, which has a 7 percent error. These approximations are good starting point to evaluate these images, but further and more granular crack identifications would be needed to obtain more accurate results.

Figure 26 - Filtered images, (a) Sobel edge detection, (b) Bluebeam scaled dimensions. (c) Otsu edge detection.



(a) (b) (c)

Table 8 - Results of image 00068 obtained from Python code.

	Label	Major Axis Length (in.)	Minor Axis Length (in.)	Perimeter (in.)
7	8	0.110471835	0.08376525	0.334824627
11	12	0.198944951	0.087437781	0.571211234
14	15	0.885044581	0.386293031	6.421948969
24	25	0.2509549	0.133922578	0.948444174
32	33	0.136940176	0.084087145	0.443309489
33	34	0.371333827	0.155714604	1.613408409
35	36	0.213270636	0.162196772	1.062661018

Conclusion

The results obtained using the program were very similar to the results previous studies had achieved. The strength of the program was the filtering process of running the image through non-max suppressions, Gaussian, Sobel, and Otsu filters. The images were put through enhancing procedures to enhance the contrast of the image, which provide identification of the edges, which then provided a pixel map with labeled areas representing the cracks in the image. This map was used to assign values to regions of the crack for evaluation and validate if the program was processing accurate results. The images chosen for this evaluation were broken into two categories of controlled images and data set images. These images were selected based on the characteristics, features and properties seen within the image in order to experiment on an ample set of images to explore different conditions that can be realistically encountered. It can be concluded that the proposed algorithm had mixed performance results when set under varying background, color, and crack patterns. The program performed well for the controlled images as it was able to evaluate all the features of the images and provide results that were consistent with the measurable values that were obtained using measuring tools. The performance for these images could be attributed to the controlled variables that could be monitored and independent of any external factors. That is the camera used, the lighting, the crack formation, the material where all know and additionally the crack could be measured for validity using a measuring tool such as a ruler or measuring tape. The program performance for the data set of images had mixed results, as it provided relatively good results for a few features of the images but provided inconsistent values for other features. These strengths and weakness are discussed to help highlight areas that would need to improve for any further assessments of this program. The goal of this research was focused on evaluating and combing algorithms and methods used for image segmentation in pipes for use in a reasonable and urgent application for pipeline health evaluation. Given the urgency that exists to find a more efficient method for these evaluations a more complete, accurate and automated set up would be required to fully implement in real life application.

References

[1] "Where are Liquid Pipelines Located?" Pipeline 101, https://pipeline101.org/Where-Are-Pipelines-Located

[2] Chun-Qing Li, Hassan Baji, Wei Yan, "Optimal Inspection Plan for Deteriorating Structural Members Using Stochastic Models with Application to Buried Pipelines" J. Struct., 2019, 145(11)

[3] Frances Richards, "Failure Analysis of a Natural Gas Pipeline Rupture" J Fail. Anal. And Prevn. (2013) 1:65-657

[4] U.S. Energy Information Administration, "U.S. Energy Facts Explained", August 28, 2019, https://www.eia.gov/energyexplained/us-energy-facts/.

[5] U.S. Department of Transportation, Pipeline and Hazardous Materials Safety Administration, "Pipeline Mileage and Facilities", January 28, 2020, https://www.phmsa.dot.gov/data-and-statistics/pipeline/pipeline-mileage-and-facilities.

[6] W. Visser, Y. Sun, O. Gregory, G. Plume, C-E. Rousseau, H. Ghonen, "Deformation characteristics of low carbon steel subjected to dynamic impact loading" Material Science and Engineering A 528 (2011) 7857-7866.

[7] John F. Kiefner, Cheryl J. Trench, "Oil Pipeline Characteristics and Risk Factors: Illustrations from the Decade of Construction" Kiefner & Associates Inc., Allegro Energy Group, December 2001.

[8] Shivprakash Iyer, Sunil K. Sinha, "A robust approach for automatic detection and segmentation of cracks in underground pipeline images" Image and Vision Computing 23 (2005) 921-933.

[9] Sunil K. Sinha, Paul W. Feiguth, "Automated detection of cracks in buried concrete pipe images" Automation in Construction 15(2006) 58-72.

[10] Mohammad R. Jahanshahi, Jonathan S. Kelly, Sami F. Masri, Gaurav S. Sukhatme, "A survey and evaluation of promising approaches for automatic image-base defect detection of bridge structures" Structure and Infrastructure Engineering Vol.5, No.6 December 2009, 455-486

[11] Hua Song, Kunshan Ge, Di Qu, Huapu Wu, Jing Yang, “Design of in-pipe robot based on inertial positioning and visual detection” Advances in Mechanical Engineering 2016, Vol 8(9) 1-22.

[12] Nhat-Duc Hoang, Quoc-Lam Nguyen. “Concrete Wall Cracks: A Comparative Study on the Performances of Roberts, Prewitt, Canny and Sobel Algorithms.” Advances in Civil Engineering, Volume 2018, Article ID 7163580, 16 pages. https://doi.org/10.1155/2018/7163580

[13] Young-Jin Cha, Wooram Choi and Oral Buyukozturk. (2017). “Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks.” Computers-Aided Civil and Infrastructure Engineering, 32 (2017) 361-378.

[14] Darragh Lydon, Myra Lydon, Su Taylor, Jesus Martinez Del Rincon, David Hester, James Brownjohn. (2019). “Development and field testing of a vision-base displacement system using a low-cost wireless action camera.” Journal of Mechanical Systems and Signal Processing 121, 343-358.

[15] F. Frigui, J.P. Faye, C. Martin, O. Dalverny, F. Peres, S. Judenherc. (2018). “Global methodology for damage detection and localization in civil engineering structures.” Engineerings Structures 171, 686-695.

[16] “What is NumPy?”, https://numpy.org/doc/stable/user/whatisnumpy.html

[17] “Open CV on Wheels”, https://pypi.org/project/opencv-python/

[18] Charles R. Harris, K. Jarrod Millman, Stefan J. van der Walt, Ralf Gommers, Paul Virtanen, David Cournapeau, Eric Weiser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernandez del Rio, Mark Wiebe, Pearu Peterson, Pierre Gerard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, Travis E. Oliphant. (2020). “Array Programming with NumPy.” arXiv: 2006. 10256v1.
https://www.researchgate.net/publication/342302317_Array_Programming_with_NumPy

[19] Great Learning Team (2022). “What is NumPy in Python” https://www.mygreatlearning.com/blog/python-numpy-tutorial/#:~:text=NumPy%2C%20which%20stands%20for%20Numerical,stands%20for%20'Numerical%20Python'.

[20] John Hunter, Darren Hale, Eric Firring, Michael Droettboom and the Matplotlib development team (2012). “Matlplotlib.pyplot”. https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html#module-matplotlib.pyplot

[21] Mohit Gupta_OMG (2020). “numpt.round_() in Python”. https://www.geeksforgeeks.org/numpy-round_-python/#:~:text=The%20numpy.,the%20given%20number%20of%20decimals.&text=Parameters%20%3A,%3A%20%5Barray_like%5D%20Input%20array.

[22] Justin Johnson. “Python NumPy Tutorial (with Juniper and Collab)”. https://cs231n.github.io/python-numpy-tutorial/#:~:text=started%20with%20Numpy.-,Arrays,the%20array%20along%20each%20dimension.

[23] “Python imread (): Different ways to load an image using the OpenCV.imread() method”. https://www.askpython.com/python-modules/python-imread-opencv

[24] Krunal (2020). “Python Cv2: Filtering Image Using GaussianBlur() Method”. https://appdividend.com/2020/09/19/python-cv2-filtering-image-using-gaussianblur-method/#:~:text=In%20cv2.,of%20the%20OpenCV%2DPython%20library.

[25] The All Learner (2020). “First-order Derivative kernels for Edge Detection”. https://theailearner.com/tag/cv2-sobel/

[26] OpenCV development team (2014). “Image Filtering”. https://docs.opencv.org/3.0-beta/modules/imgproc/doc/filtering.html#cv2.Sobel

[27] Avijeet Biswal (2022). “Understanding Python If-Else Statement”. https://www.simplilearn.com/tutorials/python-tutorial/python-if-else-statement#:~:text=The%20if%2Delse%20statement%20is,else%20block%20code%20is%20executed.

[28] Python Pool (2021). “CV2 Normalize () in Python Explained with Examples”. https://www.pythonpool.com/cv2-normalize/

[29] Ankit Lathiya (2022). “Np.Ones: Understanding NumPy Ones () Function”. https://appdividend.com/2022/01/19/np-ones/

[30] Edubca (2022). “Introduction to OpenCV Morphology”. https://www.educba.com/opencv-morphology/

[31] Data Carpentry (2018). “Image Processing with Python”. https://datacarpentry.org/image-processing/07-thresholding/

[32] Andrew Greensled (2010). “Otsu Thresholding”. http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html

[33] Muthukrishman (2020). “Otsu’s method for image thresholding explained and implemented”. https://muthu.co/otsus-method-for-image-thresholding-explained-and-implemented/

[34] Scikit-image development team. “Module”. https://scikit-image.org/docs/dev/api/skimage.measure.html#skimage.measure.label

[35] Özgenel, Ç.F., Gönenç Sorguç, A. (2018). “Performance Comparison of Pretrained Convolutional Neural Networks on Crack Detection in Buildings”, ISARC 2018, Berlin.

[36] Adrien Rosebrock (2017). “Object Detection with Deep learning and OpenCV”. https://pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/