A Deep Learning Approach for Automating Concrete Bridge Defect Assessment Using Computer Vision

by

Evan McLaughlin

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Civil Engineering

Waterloo, Ontario, Canada, 2020

Author's Declaration

Abstract

Current bridge inspection practices are outdated compared to the advanced technologies available today, and there is significant room for improvement. For example, spalls are inspected by visual assessment and delaminations are inspected by sounding for hollow areas in the concrete. This yields coarse size estimation and subjective measuring, which is exacerbated by limited funding. These limitations severely restrict inspection information provided to an engineer, making adequate bridge management difficult and bridge repairs expensive. Current inspection researchers are aware of this problem, and therefore there is significant focus on applying advanced technologies to improve the accuracy and economic efficiency of routine bridge inspections for improved bridge management.

The Structural Dynamics Identification and Control (SDIC) research lab at the University of Waterloo has been working to develop a process for automated end-to-end inspection of spalls and delaminations in reinforced concrete bridges that tightens size estimation, removes subjectivity, and improves accessibility. This process combines the accessibility benefits of robotics with the detailed 3D structural modelling of state-of-the-art simultaneous localization and mapping (SLAM), and the accurate and objective object labeling of state-of-the-art convolutional neural networks (CNN). Major steps required for this automated end-to-end inspection can be broadly divided into five components: 1) a mobile data collection platform complete with lidar and camera sensors, 2) a mapping component to fuse data from various sensors into a common reference frame, 3) a defect labeling component to automatically label defects in images, 4) a map labeling component to semantically enrich the 3D map with pixel information from images, and 5) a non-subjective and automated defect quantification component.

The work in this thesis focuses specifically on components 3), 4), and 5). These three components assume that data is collected by lidar and camera sensors (Component 1) and a 3D map of the bridge structure has been generated by SLAM (Component 2). To achieve component 3, this thesis presents an implementation of MobileNetV2/Deeplab V3, which is a state-of-the-art pixel-wise CNN, for fully automated pixel-wise labeling of spalls and delaminations in visual and infrared images respectively. Spalls are labeled with 71.4% mean intersection over union (mIoU) and delaminations with 82.7% mIoU, which is reasonable compared to same CNN's score of 77.3% on benchmark datasets. For component 4, an algorithm is developed based on the pinhole camera model and ray-tracing to intelligently fuse the CNN and colour data stored in pixels with the generated 3D point cloud. This yields a spatially accurate 3D map of the scanned structure that is colourized and semantically enriched with defect information. This enables the last component, which implements an algorithm to automatically extract, organize, and quantify areas for both spall and delamination defects present in the semantically labeled 3D map. A comparison is performed to test the difference of using manual ground truth defect labels in the image versus automated CNN labels, with all else held constant. The comparison showed a defect size error of 25.9% for spalls and 13.6% for delaminations, which is proportional to the 28.6% and 17.3% mIoU errors reported for spalls and delamintions respectively at the image labeling step. This is evidence that this pipeline can be used for any defect area quantification, where the optimal CNN can be chosen for automated labeling based on a tradeoff of accuracy vs. computation requirements. Future work involves extending this pipeline to include more defect quantification, such as crack length and crack width.

Acknowledgements

I would like to thank my supervisor, Dr. Sriram Narasimhan, for the support and motivation to push myself towards producing contributions to research that will leave a lasting sense of accomplishment. I would also like to thank Dr. Chul Min Yeum and Dr. Trevor Hrynyk for being on my thesis review committee and for evaluating my oral defence.

A big thank you also goes to all my fellow lab members for being supportive of each other's successes as graduate students. A special thank you goes to lab members Nicholas Charron and Stephen Phillips for working with me to learn new software everyday in the lab, Dirk Friesen for providing invaluable hardware support, and Stan Fong and Dylan Dowling for always being willing to engage in thought-provoking conversations to improve the quality of research produced by the lab.

I would also like to thank the Natural Sciences and Engineering Research Council (NSERC), the Ontario Graduate Scholarship (OGS) program, and the University of Waterloo for providing funding so I could focus more of my time and effort on producing high-quality research.

Dedication

This is dedicated to my parents, Robert and Luisa, and to the rest of my family who supported me throughout all of my endeavours.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In North America, bridges are required to undergo routine inspections while following standardized guidelines, e.g. the Ontario Structure Inspection Manual (OSIM) [39], to ensure that they are in adequate structural and serviceable condition. Such inspections are typically performed by qualified inspectors for visual assessment of common defects such as cracks, spalls, or delaminations. The sheer number of bridges which fall under such regulatory purview combined with the routine inspection frequency (typically every two years) has resulted in a significant burden on the inspection labor force. As a majority of routine inspection tasks are currently undertaken visually, the subjective nature of such routine inspections calls for more repeatable, reliable, and automated methods to complement or even replace many inspection tasks.

1.1 Routine Concrete Bridge Assessment

The goal of routine structural concrete bridge inspections is "to ensure, within an economic framework, an acceptable standard for structures in terms of public safety, comfort and convenience" [39]. To achieve these goals, OSIM outlines a set of defects that can be detected visually by a qualified inspector within arms reach of the defect. Common defects assessed during routine bridge inspection include delamination, spalling, cracks, erosion, scaling, disintegration, etc. These defects are detected, measured for pertinent dimensions (e.g. area, length, width), and then reported in severity. This thesis focuses on improving the inspection method for delamination and spalling, which are two critical defects that are both related to each other in the way they manifest in the structure.

1.1.1 Spalling Assessment

OSIM describes a spall as ”a fragment, which has been detached from a larger concrete mass” [39]. Therefore, when searching for spalling an inspector must search for areas where concrete has completely detached from the bridge structure, leaving a noticeable volume of missing concrete (Fig. 1.1). Spalling is considered a problem for a few reasons: 1) the concrete falling from the structure can pose a hazard to people or structures below, and 2) the loss of concrete reduces concrete cover that protects the steel rebar from corroding. Spalling does not often represent a direct loss to strength, as it generally occurs at locations where concrete is in tension, which is usually ignored in design calculations.

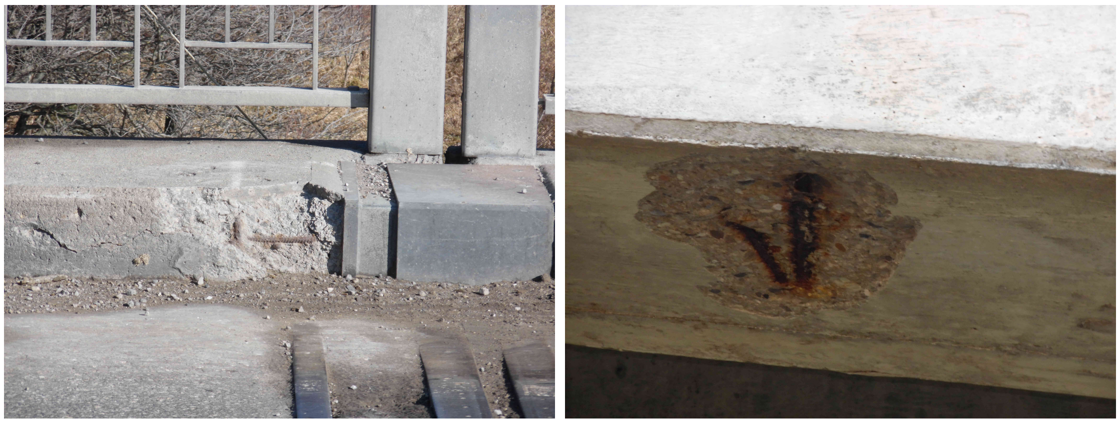


Figure 1.1: Example photos of spalls. (Photos by: Region of Waterloo)

Spalls are visually detectable as the the location of the spall will have a noticeably different texture (e.g. aggregate concrete exposed, rough surface), will have a noticeable depth change, and in extreme cases will have locations of exposed rebar. During inspection, spalls are classified for severity by their area and depth of concrete removed. Following [39], spalls can be classified as:

- LIGHT: Spalled area measuring less than 150 mm in any direction or less than 25 mm in depth.
- MEDIUM: Spalled area measuring between 150 mm to 300 mm in any direction or between 25 mm and 50 mm in depth.
- SEVERE: Spalled area measuring between 300 mm to 600 mm in any direction or between 50 mm and 100 mm in depth.

- VERY SEVERE: Spalled area measuring more than 600 mm in any direction or greater than 100 mm in depth.

1.1.2 Delamination Assessment

OSIM describes a delamination as, ”a discontinuity of the surface concrete which is substantially separated but not completely detached from concrete below or above it” [39]. This means that to detect a delamination an inspector looks for an area where concrete is internally cracked, but not yet fully breaking off from the larger structural concrete mass. Dalamination is a problem for a few reasons: 1) delaminations are likely to grow until they become a spall if left without maintenance intervention, and 2) delaminations signify internal cracking, which is likely caused by the expanding volume of the rebar that results from corrosion.

Delaminations are often not visually detectable, as they are a subsurface defect. They instead tend to be detected by hammer sounding or chain dragging. These methods require the inspector to use the distinct hollow sound of delaminated concrete to estimate the extent of the delamination. Another method is to use the change in thermal patterns at a delaminated area to estimate the delamination area. This is possible as the smaller delaminated mass of concrete heats/cools faster than the larger mass, resulting in different thermal patterns when subjected to day/night heating and cooling cycle. Fig. 1.2 shows examples of delaminations revealed by infrared imagery. Following [39] delaminations can be classified as:

- LIGHT: Delaminated area measuring less than 150 mm in any direction.
- MEDIUM: Delaminated area measuring 150 mm to 300 mm in any direction.
- SEVERE: Delaminated area measuring 300 mm to 600 mm in any direction.
- VERY SEVERE: Delaminated area measuring more than 600 mm in any direction.

1.2 Motivation

A key aspect of current delamination and spall assessment that could be greatly improved is area size quantification. Area size quantification can be cumbersome and fraught with

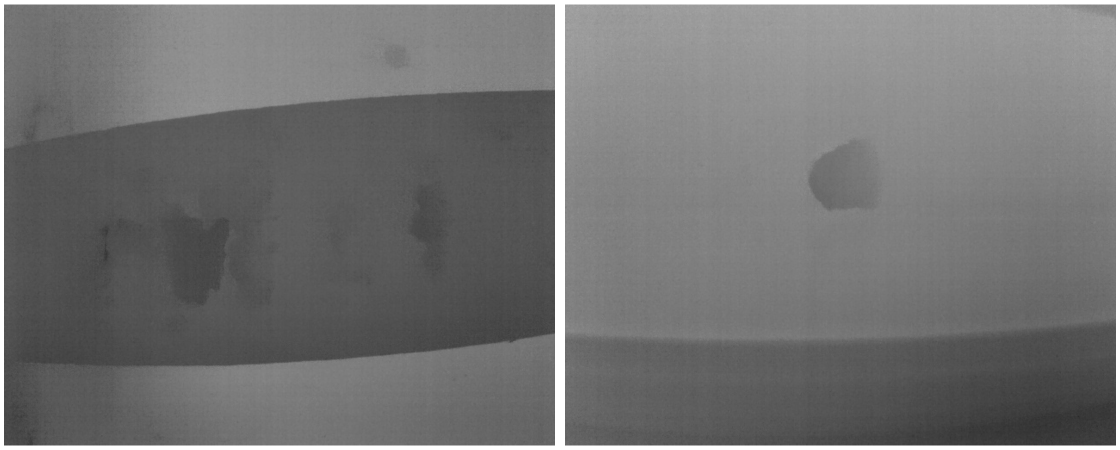


Figure 1.2: Example delaminations showing different thermal patterns in infrared images.

human error when access is limited due to long spans, water bodies, traffic conditions, and safety requirements. Such limitations lead inspectors to estimate the dimensions and quantities without supporting measurements, or could lead to neglecting the defects all together. As a result, subjectivity and variability between inspections is common [22]. Such variability in delamination/spall areas makes comparing areas between routine inspections (e.g. two years apart as per [39]) challenging, and prevents quantitative tracking of defect growth over time. As well, lack of quantitative information makes asset management decisions more challenging and hence there is an urgent need for new methods that can produce objective, repeatable, and quantifiable defect identification in the field.

Developments in modern sensors such as high-resolution cameras and laser scanners (lidars) can significantly assist in overcoming current defect inspection practices. Computer vision techniques and machine learning algorithms can be used to automatically analyze images for defects, while lidar information can provide reference-free scale when the image information is fused together with lidar data. Combined with robotics technologies, specifically simultaneous localization and mapping (SLAM), lidar scans and images obtained at different locations can be fused to build 3D representations of bridge assets. The core work of this thesis is therefore focused on how state-of-the-art computer vision techniques can be applied to process camera and lidar data, and to automate and remove subjectivity from the area size quantification of delaminations and spalls.

Chapter 2

Background

Traditionally, an inspector will capture photos of a detected defect to provide more information to the inspection report. These images are supplemental, and are not the main method of obtaining information about the severity of the defect. However, the images do contain information necessary to automate the defect detection, localization, and quantification process.

2.1 Images

Images are structured grid-like data of pixels that provide information about the location photographed at a specific snapshot in time. The size, density, quality, lighting, and more can all vary for an image, but the method in which image data is stored primarily remains the same. To understand the characteristics that enable detection of spalls and delaminations in images, it must first be understood how both visual images and infrared images represent the data collected by the camera.

2.1.1 Light

Broadly speaking, images are generated by capturing light information from the 3D world at a specific point in time. Further details on how to model this are elaborated when discussing the Pinhole Camera Model. However, it is first necessary to cover a brief overview of light waves, and specifically the difference between the light wavelength for the visual spectrum of light and for infrared light.

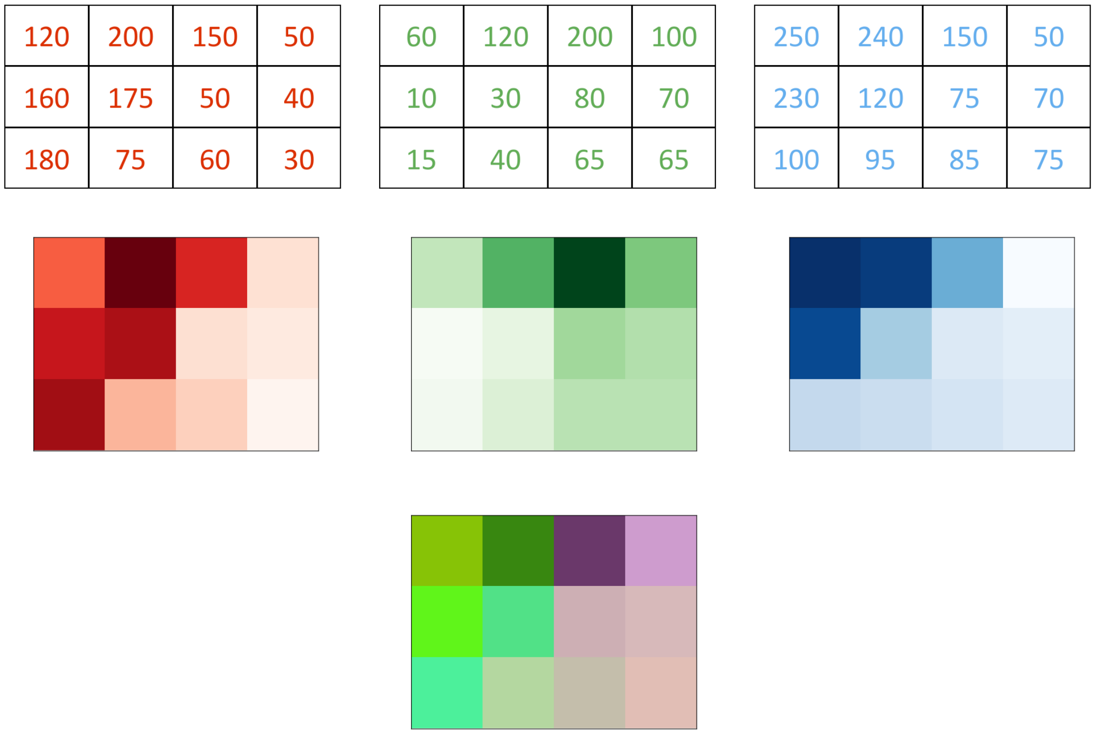
The visual spectrum of light consists of light wavelengths between $400nm$ and $750nm$. This range is named the visible spectrum because it is the range of light wavelengths that are visible to the human eye, with a $400nm$ wavelength being violet, a $750nm$ wavelength being red, and with a gradient of the colours of the rainbow making the values in between. Most cameras operate in the visible spectrum for ease of human interpretation.

There exist cameras to capture other wavelengths that are be useful for certain applications, one of these wavelengths being infrared. Infrared light has slightly less energy than red visible light. Less energy means a larger wavelength, and so infrared cameras are generally capturing light wavelengths from $8000nm$ to $15000nm$. This is beneficial, as an infrared camera can provide information about the thermal emissivity of an object. This thermal emissivity does not give absolute temperature, but can give an idea of which objects are hotter/colder relative to a scene.

2.1.2 Visual Images

A visual image is the most common type of image, in which it captures light information in the visible spectrum. These are typical to how one would capture an image of their family, their pet, etc. with any standard camera (e.g. phone camera). A visual image is typically stored as an $m \times n \times 3$ matrix of pixels, where m is the height of the image and n is the width. The depth of the matrix is 3, as image pixels are typically stored such that colour is represented using the red, green, blue (RGB) colour spectrum, where the colour of each individual pixel is represented as a combination of varying intensities of red, green, and blue. The RGB values are represented by an integer from 0 to 255, which is chosen so that each pixel can be stored using only $8c$ bits of memory, where c is the number of colour channels. It is possible to capture more detail by allocating more memory, but for the majority of images 8 bits is adequate. Fig. 2.1 provides an example of how an RGB visual image is built.

A spall is visible in a typical RGB visual image because the image captures the different appearance (e.g. exposed aggregate, rough concrete, lighting changes) that are caused by the detached concrete (Fig. 1.1). The depth discontinuity caused by a spall is not visible in the image, as an image does not contain depth information. However, lighting effects (e.g. shadows) caused by the discontinuity may be visible in some cases.



120	200	150	50
160	175	50	40
180	75	60	30

60	120	200	100
10	30	80	70
15	40	65	65

250	240	150	50
230	120	75	70
100	95	85	75

Figure 2.1: Example of how a visual RGB image is stored as a combination of red, green, and blue intensities. These intensities are combined to make the output $3 \times 4 \times 3$ visual image at the bottom of the figure.

2.1.3 Infrared Images

An infrared image is less common than a visual image, as it captures light in the infrared range. Infrared images are often referred to as thermal images, as a thermal image explicitly indicates a passive infrared sensor (i.e. does not emit any infrared energy itself, and only sense energy from the environment). For the case of this thesis only passive infrared sensors are used, and the terms thermal and infrared image are assumed to be the same. Infrared images differ from a visual image, as it is typically stored as an $m \times n \times 1$ matrix of pixels. The depth of the matrix is only one, because each pixel is a single integer between 0 and 255 that represents the thermal intensity reading of the camera. The intensities are often stored as either white-hot (255 indicates hottest and 0 indicates coldest), or black-hot (0 indicates hottest and 255 indicates coldest). Fig. 2.2 provides an example of how an infrared image is built.

A delamination is noticeable in an infrared image because a delamination has a different heating/cooling pattern compared to the large concrete mass (Fig. 1.2). This is because the internal discontinuities create an insulating layer, making the delaminated concrete a smaller, separate mass. This causes the delamination to heat/cool at a different rate than the larger mass and therefore produces a different thermal intensity when captured by an infared camera. Because the delamination is detected by distinct local changes in thermal intensity, it is possible to detect the heating/cooling effects the same way in either a white-hot or black-hot infrared image.

255	180	200	55
195	175	40	30
190	45	35	0

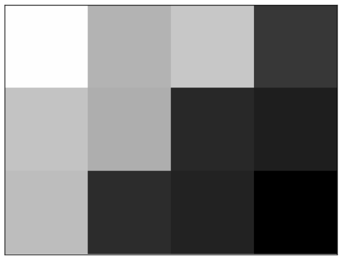


Figure 2.2: Example of how grayscale intensities are stored to make a $3 \times 4 \times 1$ grayscale infrared image.

2.1.4 Pinhole Camera Model

With knowledge of how visual and infrared images are stored, it is also important to know how a camera collects such data and projects a 3D world object or a feature into a 2D grid-like array of pixels (note even though the matrix may be 3D in the case of RGB, the third dimension is only to store colour data). This can be done by understanding the pinhole camera model [24, 61, 4], which is a sufficient model for all cameras used in this thesis. The pinhole camera model is derived by assuming that light from the external environment travels through a pinhole point and then continues until it comes into contact with the sensor of the camera. This sensor reads the light information, and after some internal data processing it can be saved and stored in an RGB format. The way objects in an environment (represented in world coordinates, X, Y, Z) may be projected to the sensor (represented by u, v) is shown in Fig. 2.3. The location at which the light information is read is often called the image plane, and the distance from this plane to the camera pinhole is known as the focal length of the camera. The virtual image plane is the location in front of the center that is a focal length distance in front of the pinhole, and the central point on this plane is the principal point. The purpose of the virtual image plane is to visually simplify the pinhole camera model, while keeping the mathematics valid.

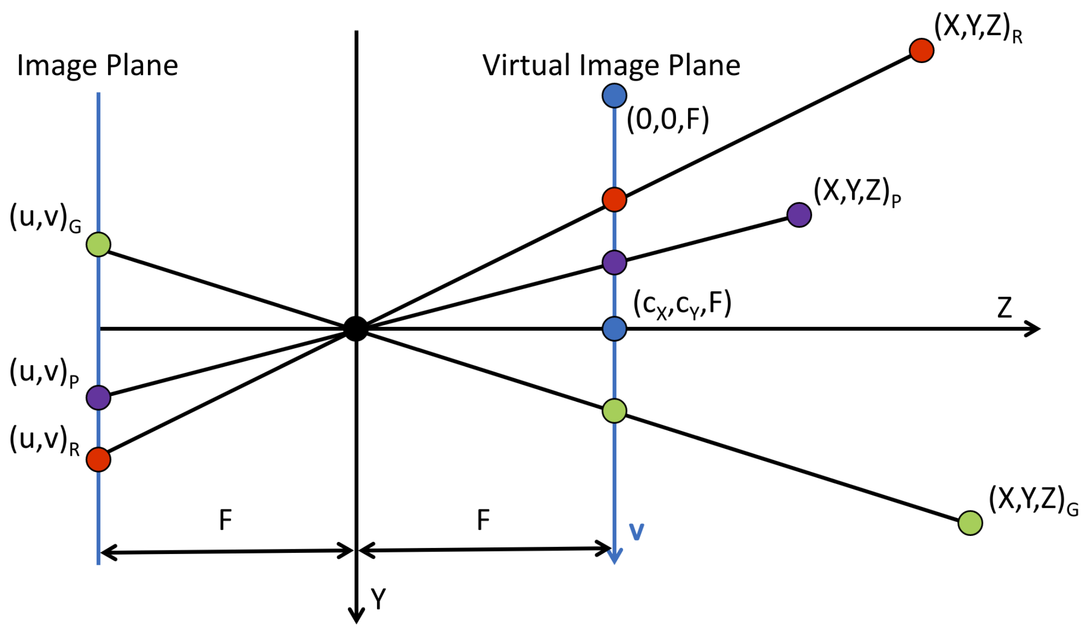


Figure 2.3: Pinhole camera model used to show high-level how a camera collects light information, and to derive the equations for projecting 3D world coordinates to a 2D virtual image plane. The model shown assumes no lens distortion. The X and u axes are assumed to be directed out of the page. The R, P, G subscripts are used to label the red, purple, green points to help follow which 3D point is stored where in the image plane. The point (c_x, c_y, F) is known as the principal point, and is the center of the virtual image plane.

The image plane in Fig. 2.3 represents the location of the sensor in the case of a physical camera. This sensor reads the light it receives and can store it as a matrix of pixels, where each pixel contains the color information (e.g. RBG or infrared intensity). This pinhole model is used to derive the equations necessary to project a 3D object onto a 2D grid of pixels. To simplify this example, assume that the global points are in the same coordinate frame as the camera (X, Y, Z coordinate frame where the pinhole of the camera is $(0, 0, 0)$), and the goal is to covert a point (X, Y, Z) to a (u, v) pixel.

Consider the v coordinate of a 3D point in Fig. 2.3. We can construct a formula using similar the triangles property and isolate for v,

$$\frac{v - c_y}{f_y} = \frac{Y}{Z} \tag{2.1}$$

$$v = \frac{Y}{Z} * f_y + c_y \tag{2.2}$$

where v is vertical location of the pixel on the image, f_y is the focal length of the camera, c_y is the offset of the principal point in the vertical direction, and Y, Z describe location of the object relative to the camera in 3D space. In the horizontal direction, the same process can be done to get the vertical u component using

$$\frac{u - c_x}{f_x} = \frac{X}{Z} \tag{2.3}$$

$$u = \frac{X}{Z} * f_x + c_x \tag{2.4}$$

where f_x is the focal length of the camera (note that in all applications in this thesis $f_x = f_y = F$), c_x is the offset of the principal point in the horizontal direction, and X, Z describe the location of the object relative to the camera in 3D space. These two equations can be combined into a matrix,

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

that can then be applied to any 3D point in the camera coordinate frame using the equation,

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K} \left(\frac{1}{Z} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \right) \tag{2.6}$$

to convert a 3D point to a 2D pixel. The matrix **K** is known as the intrinsic camera matrix, as it contains the internal properties of the camera that remain constant assuming no damage or mechanical adjustments. It is often the case that $f_x = f_y$, $c_x = \frac{width}{2}$, and $c_y = \frac{height}{2}$, meaning a manufacturer could provide these parameters. However, in practice this matrix is obtained through checkerboard camera calibration and is essential for combining camera data with data from other sensors such as a lidar. See appendix A.1 for details on calibrating a camera with a checkerboard to determine its intrinsic **K** matrix. To use the matrix, the 3D points must be in the camera coordinate frame, which can be done using extrinsic properties. These extrinsic properties describe how the camera is positioned in the world. Further details on how these properties are extracted are covered in 3D mapping section.

The similar triangle formulation of these equations also highlights the limitations of information collected using a camera. Specifically, it shows how it is relatively simple to project from 3D to 2D, but impossible to transform 2D to 3D using a single image because of the scale ambiguity that is caused by the Z coordinate. That is, all 3D points along a linear ray extending from the camera center and through a (u, v) pixel will have the same values for $\frac{X}{Z}$ and $\frac{Y}{Z}$, therefore mapping the same (u, v) pixel location in 2D. For this reason, the 3D point corresponding to a u, v pixel can be determined up to a scale factor of s, where the 3D point is $s[X\ Y\ 1]^T$. This is why it is not possible to estimate depth from a single image alone.

2.2 Computer Vision

With knowledge of how data is collected by a camera and stored in an image, the next step is to automate the extraction of information within the image. This is done using image processing and computer vision. Computer vision is different from image processing, as computer vision involves analyzing the image for certain information (e.g. object detection). Image processing is a broader concept that covers all techniques to process an image (e.g. sharpening, denoising, etc.), and in many cases can work as part of a computer vision algorithm (e.g. noise removal before object detection). In this thesis, some image processing is used but the focus is specifically on computer vision techniques for detecting and localizing a defect in an image. More broadly, the goal is automatic image-based object detection and localization.

In the computer vision community, there are two types of approaches that are used to automatically process image data for detecting and locating objects of interest: knowledge-driven methods and data-driven methods. Both these methods make use of image convolution—a core concept in computer vision—but use it in a different manner. Knowledge-driven methods are based on the known characteristics of the object of interest. For example, this knowledge may include known colours (e.g. RGB pixel values to search for are known) and shapes of the objects of interest in the images. Knowledge based methods will often include human-input rules for determining which features in images correspond to which objects. The second approach falls under data-driven methods, which rely on large quantities of labeled image data to train deep prediction networks. Once these networks have been trained to detect and localize the objects of interest based on the input data, they can then be used to detect and localize those objects on new data sets. These methods require large labeled datasets and significant processing resources to train the networks. Data-driven methods have become the method of choice for many computer vision based object recognition applications, and have have consistently dominated computer vision competitions since a data-driven network first won the Imagenet challenge in 2012 [28]. However, it is still important to understand knowledge-driven methods to understand how data-driven methods work.

2.2.1 Image Convolution

Before discussing knowledge-driven versus data-driven methods it is important to understand their main building block —image convolution. Image convolution is the process of moving a kernel over each valid pixel in the image and calculating a new value for that pixel based on the properties of the kernel. Mathematically, the image convolution for a location (i, j) in an image can be given by equation

$$S(i,j) = (K \circledast I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{2.7}$$

where $\circledast$ is the convolution operation, S is the output of the convolution, I is the original image and K is the $m \times n$ convolution kernel. However, in this formula it is evident that the indices $(i - m, j - n)$ into the input image I decrease as the indices (m, n) into the kernel K will increase. This provides no benefit to filtering an image, and complicates the implementation. As a result, while the term is often still called convolution, cross-correlation is actually what is implemented in most computer vision programming libraries. It is given by the equation

$$S(i,j) = (K \circledast I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{2.8}$$

where the only change is the subtraction of m, n from i, j turns into addition. This makes the location for which to index into the image and the kernel both grow as values m, n increase. This is more intuitive, and therefore most libraries implement cross-correlation. However, these libraries will still call the operation convolution even though cross-correlation is what is implemented [21]. In this thesis, image convolution will refer to the method of cross correlation presented in Equation 2.8. Fig. 2.4 shows an example this cross-correlation being applied to an small sample image tensor. Notice that the filter decreases the size of the original image, but this is often negligible in practice as image have large dimensions relative to the decrease in size. The size reduction can also be prevented by padding additional values (e.g. zeros, or reflected values) around the edges of the image before convolution.

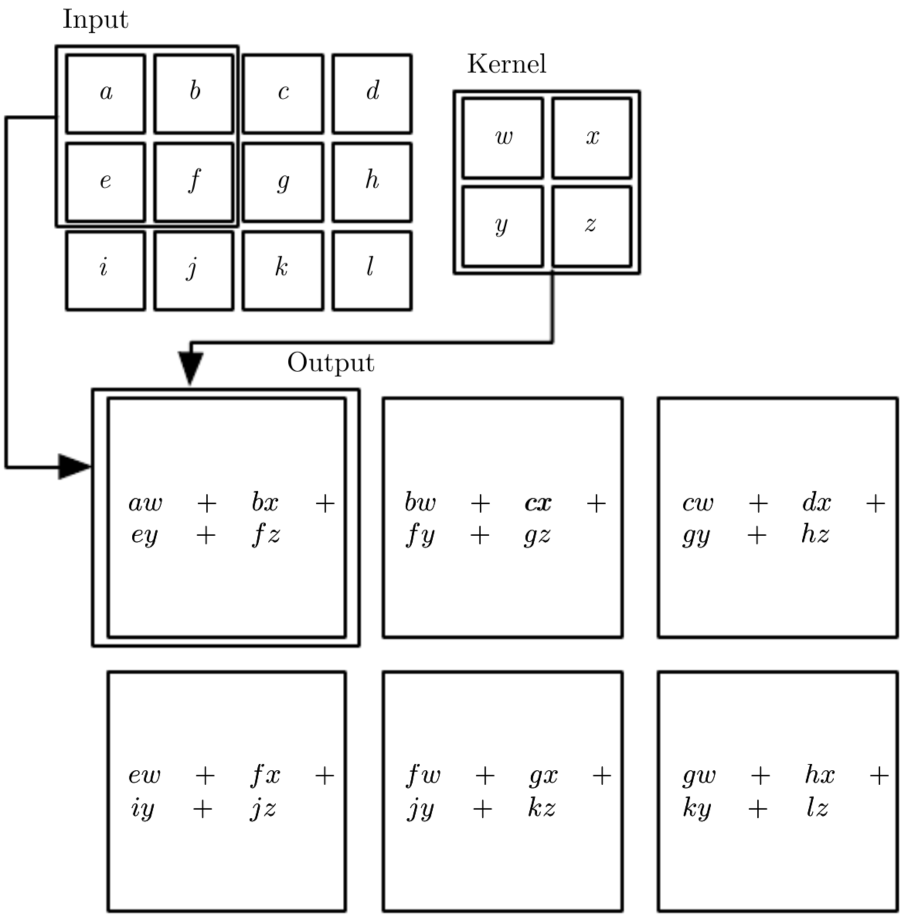


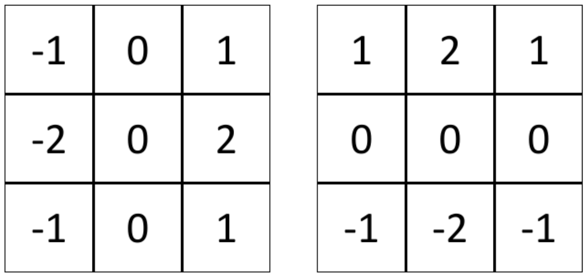
Figure 2.4: Image convolution with a 2×2 kernel and a 3×4 image. (Courtesy of [21])

Image convolution is the core building block of many knowledge-driven and data-driven methods of computer vision. The key difference is that in knowledge-driven methods the kernel weights are carefully constructed by computer vision experts to extract some information of interest, whereas in data-driven methods the kernel weights are randomly initialized and then learned by the algorithm using an iterative approach. More detail is provided when discussing these specific methods.

2.2.2 Knowledge-Driven Computer Vision

There are numerous knowledge-driven methods for object detection and localization within computer vision. For understanding these methods, a good algorithm to analyze is edge detection and localization using canny edge detection [6]. This algorithm makes use of image convolution and other mathematical techniques to detect edges, making it a good example for understanding how knowledge-drive methods use image convolution to extract information from an image.

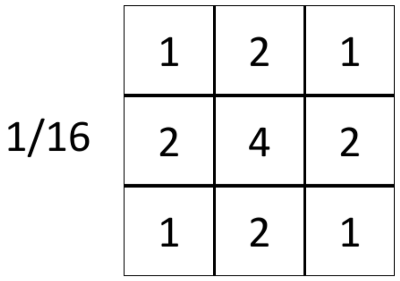
Recall in the case of knowledge-driven methods, the convolution kernel weights are defined based on known desirable properties or information to extract. In this case, it is known that an edge in an image appears as a sharp change in image intensity causing a clear disparity between neighbouring pixels. This can be due to lighting change (e.g. shadow), colour change (e.g. red to blue), or depth change (e.g. foreground to background object). For this reason, the kernel to detect edges is a kernel that estimates the horizontal (G_x) or vertical (G_y) derivative of the image at each valid pixel location, as a high value of the derivative is a likely indicator of an edge. For canny edge detection, the common kernel used for derivative calculation is the sobel kernel (Fig. 2.5). When using a kernel such as the sobel, it is likely that the image will also contain noise that may cause random high derivatives to be present at areas that are not edges. Therefore, in canny edge detection the image is first filtered using a Gaussian kernel (e.g. Fig 2.6) to reduce noise effects. A Gaussian kernel essentially takes a Gaussian-weighted average around the pixel of interest, which can smooth out noise. The size of this filter and the rate of weight dropoff from the center to the outer edges can affect the amount of noise removed, but can also reduce the change in intensity locations that should be classified as edges in the original image, so the decision on filter size and weights is best made by an expert in computer vision.



-1	0	1
-2	0	2
-1	0	1

1	2	1
0	0	0
-1	-2	-1

Figure 2.5: Sobel derivative kernels. Left: horizontal derivative (vertical edge detection). Right: vertical derivative (horizontal edge detection).



	1	2	1
1/16	2	4	2
	1	2	1

Figure 2.6: Common 3×3 Gaussian filter for image noise removal.

After noise removal and gradient calculations, there are two convolved images left over —the convolved image representing the gradient in the x direction (G_x) and the convolved image representing the gradient in the y direction (G_y). The overall edge gradient matrix, G, and the gradient direction matrix, Θ, can then be calculated using these two images by using equations,

$$G = \sqrt{G_x^2 + G_y^2} \tag{2.9}$$

$$\Theta = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{2.10}$$

where all operations are performed element-wise (i.e. on each set of corresponding pixels at location (u, v) in G_x and G_y. With G and Θ calculated, the gradient magnitude and direction are now estimated at each pixel location.

The next steps for canny edge detection are to use non-maximum suppression to thin the edges, and then a hysteresis thresholding technique to segment out the edges. These are advanced techniques are not important to understanding the rest of the work presented in this thesis so explicit detail is left out. However, the central idea is that the high derivative values could indicate an edge. The issue then arises, how large a derivative is high enough to indicate an edge? This threshold is a human input (hysteresis thresholding requires two human inputs), and the optimal value can vary depending on the image. Therefore, this knowledge-driven method could be considered as only semi-automated as there is need for human intervention. This requirement of human input is true for a large amount of knowledge-driven methods.

After all steps of the canny edge detection algorithm are complete, the result is a binary mask that detects and localizes edges in the image. It is clear from the example, Fig. 2.7, that even with a Gaussian filter there is still significant noise causing erroneous edge predictions. In addition, it is clear that it is extremely difficult to distinguish an edge caused by a crack, a shadow, a depth change, etc, because intensity disparity between neighbouring pixels contains no further detail other than that an edge is present. This limitation makes it difficult to use canny edge detection alone in applications, as it does not distinguish the cause of the edge. It is common for knowledge-driven methods to adjust and/or augment such an algorithm to only detect edges caused by a specific phenomenon (e.g. cracks). However, many of these methods are shown to work on a specific dataset, but fail when the data does not fit specific boundary conditions. This limits the use of knowledge-driven methods in application, and motivates the use of data-driven methods as an alternative.

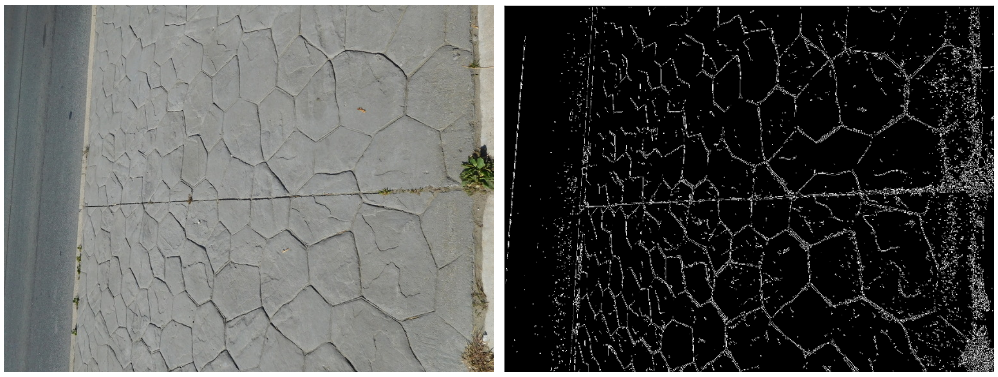


Figure 2.7: Canny edge detection example. Left: original image. Right: result.

2.2.3 Data-Driven Computer Vision

The current state-of-the-art for data-driven object detection and localization in images is a Convolutional Neural Network (CNN). A CNN is a type of neural network that uses the image convolution (Fig. 2.4) in place of a typical fully connected layer. The main difference between the image convolution used in a CNN vs a knowledge-driven method is that instead of using pre-programmed kernel weights, the kernel weights for a CNN are randomly initialized and then learned by the network to extract the necessary information.

In a CNN, numerous different kernels are stacked with other image processing techniques such as maximum pooling filters (a form of image filtering, where the pixel is assigned the maximum value in a pixel area around it, e.g. 3x3). This enables a CNN to extract features and combine different feature information to foster object detection and localization. For example, a CNN to classify if an image contains a dog might see the first few layers of the network learn to extract features such as edges, corners, circles, etc. Subsequent layers can then learn how these features combine to depict common broader features of a dog such as fur, tail, four legs, and snout. The final layers of the network then learn a liklihood that an image contains a dog based on the broader features that it found. If the likelihood is above a threshold (often 0.5), then the image contains a dog. AlexNet [28] provides one of the first and most basic illustrations of CNN in Fig. 2.8 that was used for such object classification.

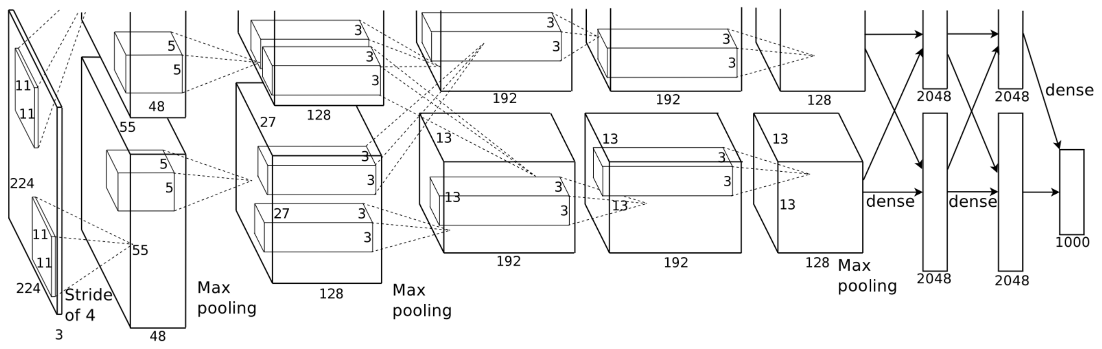


Figure 2.8: AlexNet network architecture showing how a $224 \times 224 \times 3$ input image is first filtered and max pooled then classified in 1 of 1000 categories using fully-connected dense layers. (Courtesy of [28])

The next key question for these networks is how are the network weights (specifically Kernel weights for a CNN) actually learned? The kernel weights used for convolution operations in a CNN are learned using a technique known as back-propagation, which makes use of the fact that the convolutional operator is a linear operation and is therefore differentiable. The back-propagation algorithm compares the output of the network generated with its current weights to the correct output. The correct output is provided by manually labeled examples (e.g. giving a network a picture with a dog and indicating to it that it should label the image as having dog), which makes this method of training supervised learning. After comparison, the next step is to update the weights of the kernels in the network to achieve a prediction that is closer to the correct output. This method is called back-propagation because the updates to the weights propagate backwards through the network using partial derivatives. Back-propagation is generally combined with a gradient descent method, which is an iterative method that steps the error function value towards a minimum at each step.

For a mathematical example of back-propagation and gradient descent, consider a two step process of updating the weights in a kernel. First, consider mean squared error (MSE) as a loss function

$$J(\theta) = \frac{1}{2} \sum \left(f(\mathbf{x}, \theta) - \mathbf{y}\right)^2 \tag{2.11}$$

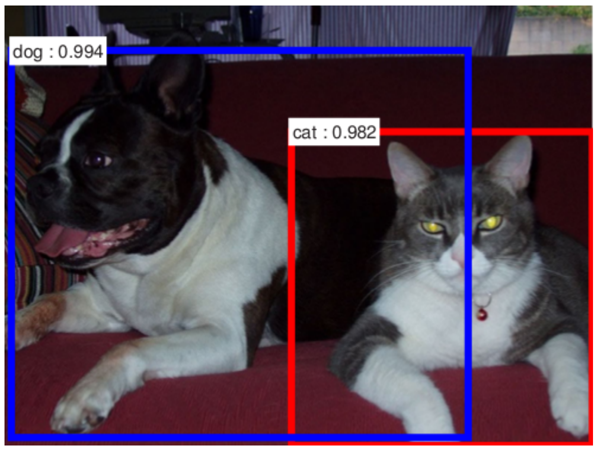
where $J(\theta)$ is the loss, θ are the weights of the convolution kernels, $\mathbf{x}$ is the input image and $\mathbf{y}$ is the correct output. This function gives the sum of the squared errors between the output from example inputs $\mathbf{x}$ and corresponding labels $\mathbf{y}$. Second, The derivative of this function can be taken with respect to each θ_i value to find the direction of fastest movement. This direction can then be used to apply the best calculated update to each kernel weight using the formula

$$\theta_i = \theta_i - \alpha \frac{\delta}{\delta \theta_i} J(\theta) \tag{2.12}$$

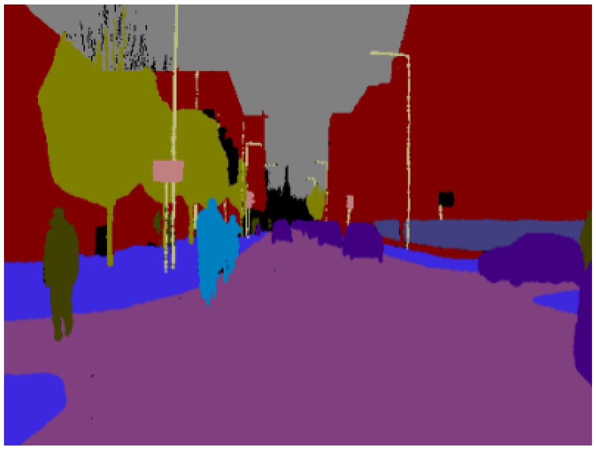
where θ_i is a specific kernel weight, α is a constant that scales the learning rate, and $\frac{\delta}{\delta \theta_i} J(\theta)$ is the partial derivative of the MSE loss function with respect to θ_i. This two step process is repeated until the total error does not change with updates or a set number of maximum iterations is reached.

This process of back-propagation and gradient descent can be extended to more complex loss functions or update functions to improve training time and accuracy. Currently, issues such as avoiding local minimums, improving training time, and improving weight distributions are active areas of research. However, the methods always build upon the general concepts of of back-propagation and gradient descent updating. Further information about robust training methods can be found in [21].

To train a CNN using supervised learning following back-propagation and gradient-descent, one must know the required structure of the network output to be able to produce labeled example data. Specifically, for the case of object detection and localization, it is important to know how the detection and location information is represented. There are two common CNN-based techniques to achieve this in an image: a region-based convolutional neural network (R-CNN) technique and a semantic pixel-wise segmentation (sometimes referred to as semantic segmentation CNN technique). R-CNNs [48, 47] detect and locate objects by outputting the coordinates of a labeled bounding box around the detected object of interest (Fig. 2.9a). Semantic segmentation CNNs [10, 2] present a structured grid-like data output, with the same pixel density of the original image. The output is a mask where each pixel is individually classified into a category, which yields pixel-level location accuracy (Fig. 2.9b). Clearly, such pixel-wise segmentation information is significantly richer compared to R-CNNs, however at the cost of added computational complexity and effort. To get a better picture of this, an overview of popular R-CNNs and semantic segmentation CNNs is provided.



(a) Bounding Boxes [48]



(b) Segmented Image [2]

Figure 2.9: R-CNN and CNN for Segmentation Examples.

Region-based Convolutional Neural Networks

There exist many R-CNN algorithms that produce good results on standardized data sets for object detection and localization. [48] presented a R-CNN algorithm called Faster R-CNN, which works by generating rectangular regions of interest (ROIs) in an image using a Region Proposal Netowork (RPN), and running the classification algorithm from [20] on each ROI. This yields object detection and localization via bounding boxes. ROI generation is important to this method, as too many regions mean low computational efficiency. The RPN algorithm in [48] improves on traditional methods of region proposals, namely sliding window methods. Sliding window region proposal involves running the classification portion of an R-CNN on each cell in a sub-grid of the image. For example, a 300×300 image can be made into a 10×10 grid of 30×30 cells, and then the classification can be run on each 30×30 cell to see if an object is present. The sliding window technique yields excessive iterations of the classification algorithm, and is therefore computationally inefficient. Faster R-CNN allows for real-time bounding box classification and localization, as the RPN suggests less sub-image regions for the classification algorithm to be run on. The main drawback of Faster R-CNN and similar R-CNN networks is they only provides bounding box level localization, which may not accurate enough for some applications.

Semantic Segmentation Convolutional Neural Networks

Fully Convolutional Networks (FCN) [52] introduced the use of convolutional networks for semantic segmentation (Fig. 2.10). FCNs contain no densely connected layers in their architecture, which are the matrix multiplication layers of a basic neural network. This makes every layer a convolutional layer, yielding the name FCN. [52] presented three different networks for segmentation: FCN-32s, FCN-16s, and FCN-8s. These networks combine coarse information from higher layers with fine information from lower layers, with FCN-32s incorporating the least fine data from higher layers, and FCN-8s incorporating the most. Intuitively, it would seem that FCN-8s would have higher accuracy but worse computation, as it incorporates more data than FCN-16s and FCN32s. This trade-off between accurate, dense pixel-wise classification, and computational efficiency was demonstrated empirically. When analyzing the affects of maintaining fine-detailed from previous layers, FCN-8s achieved a 3.3% better mean intersection over union (mIoU) score than FCN-32s, but trained 40% slower than FCN-32s. This evidence supports the intuitive trade off of computation versus accuracy when using CNNs. The more detail that is in the network, the more accurate the network can potentially be. However, the more detail that is in the network, the more computational power that is required to train it.

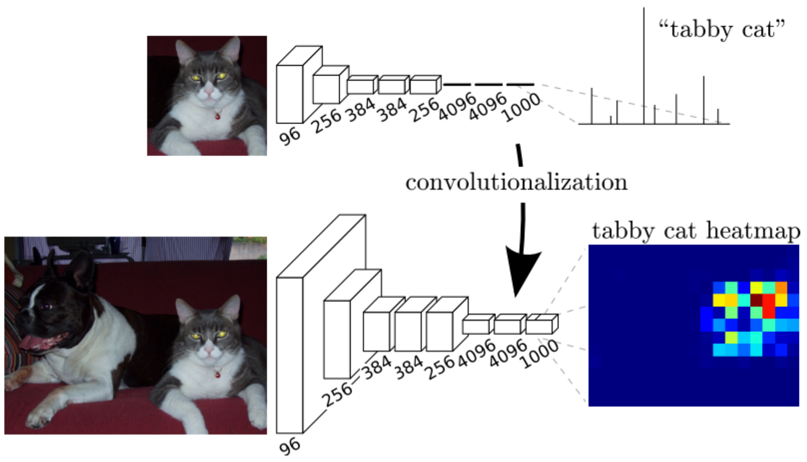


Figure 2.10: FCN architecture overview. (Courtesy of [52])

This trade off between computation and accuracy led the dominant research in CNNs for pixel-wise segmentation towards improving accuracy, or improving computation efficieny while maintatining high accuracy. [2] presents another CNN for pixel-wise segmentation, which uses an encoder-decoder structure to attempt to improve accuracy while maintaining computational efficiency (Fig. 2.11). The main idea is that pertinent information from the input image can be encoded in a tensor with smaller spatial resolution, pixel-wise labeled, and then decoded back to the original image resolution. In SegNet, max-pooling indices are saved during the encoder layers and transferred to decoder layers to improve the segmentation resolution and performance. This structure enables SegNet to better capture smaller objects compared to other methods. However, many of the object boundaries produced by SegNet are not smooth, showing that improvements are still possible.

4

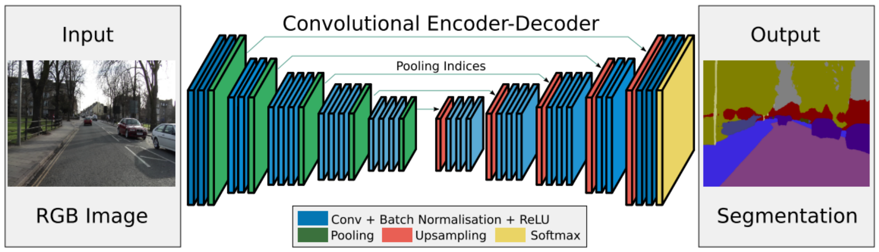


Figure 2.11: SegNet architecture overview. (Courtesy of [2])

In terms of achieving the best possible accuracy, there are many algorithms that achieve close to state-of-the-art. One example is [44], who developed an improved network structure called Global Convolutional Network (GCN) (Fig. 2.12). GCN boasted state of the art results on the two most dominant datasets. It produced a mean intersection over union of 82.2% on PASCAL VOC 2012 and 76.9% on Cityscapes. This is not the highest accuracy at the time of writing, but is typical of what accuracy is produced by state-of-the-art networks for semantic pixel-wise segmentation. Hundreds of different architectures exist, but for purposes of applying CNNs these numbers give an estimate on the accuracy values that can be expected.

CNN Selection

As illustrated above, much CNN research focuses on incrementally improving the state-of-the-art accuracy or computational efficiency benchmarks on popular data sets. However, many of the networks are nearly indistinguishable by these metrics, and thus selecting a CNN to implement for a given application is difficult. The main takeaway is that all of the popular networks that have shown good results on benchmark data sets will likely produce

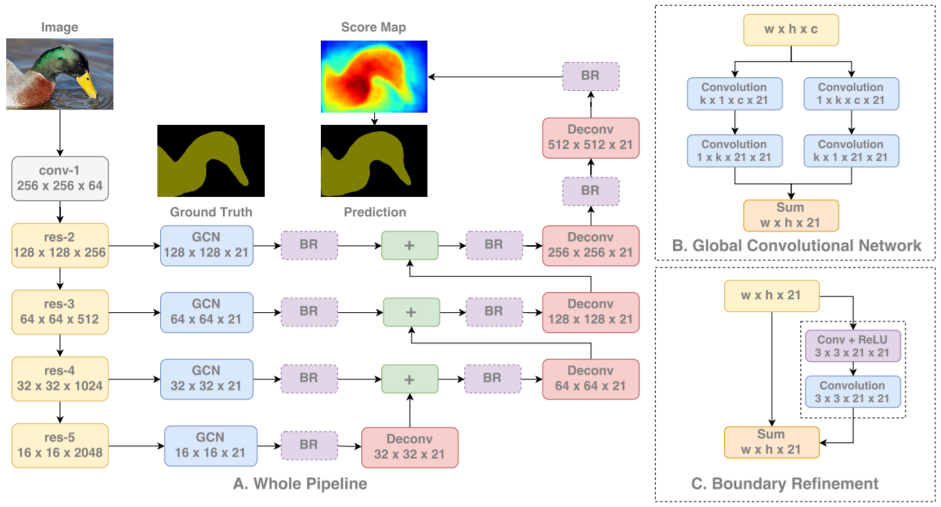


Figure 2.12: GCN architecture overview. (Courtesy of [44])

similar results for a specific application. They key is to select any reasonable network that suits the requirements of the application and focus on proper implementation of the network.

Chapter 3

Literature Review

Significant research work currently exists related to the application of computer vision techniques to augment defect detection and quantification in images. Specifically, crack detection has been researched extensively. This literature review focuses on computer vision for detection of cracks, spalls, and delaminations as there are common elements to the process irrespective of defect type. In general, the algorithms for detecting defects are not compared on an accepted standardized data set, as such a data set does not exists at the time of writing this thesis. Therefore, the reviews focus on algorithm limitations rather than the actual reported accuracy numbers.

Localization and quantification have been studied relatively less compared to detection. This is because images lack physical scale required to localize the defect on the structure (e.g. is it on a beam? girder? etc.) and to calculate a physical measurement of the defect (e.g. length in meters, area in square meters). However, state-of-the-art in sensing and robotics has made integrating localization and quantification with cameras more easily accessible, and these methods are also covered.

3.1 Knowledge-Driven Defect Analysis

Knowledge-driven methods for defect analysis have been researched longer than data-driven methods due to fewer computational requirements. Until 2012, these methods were the primary way of assessing defects from images. As a result of the extensive work, knowledge-driven methods may be considered a solved problem for most common defects, as it is currently considered that their accuracy results have approached an upper limit that is not able to compete with data-driven methods. However, it is extremely important to study how these methods can be applied to assess defects to be able to understand how data-driven methods can be properly implemented.

Cracks are the most heavily researched defect from a knowledge-driven point of view. Therefore, various algorithms exist, and can identify cracks with reasonable accuracy and precision. For example, [41] implemented a crack detection and crack tracking algorithm to detect and localize cracks in images. Images were first processed using a median filter, then the median filtered image was subtracted from the original image with the idea that the areas of largest difference were crack candidates. Morphological operations (i.e. processing images based on shapes) were then applied to crack candidates to facilitate crack segment connectivity. The next step was the tracing algorithm, in which a "seed" pixel with maximum probability of being a crack is selected, then traced bi-directionally using pixel intensities to determine which nearby pixel is most likely also part of a crack. The major flaw with this method is that the median filter and tracing algorithm rely heavily on pixel intensities, making it likely to fail in images with artifacts that can interfere with intensity values. For example, the algorithm could miss a crack that is hidden under a dark shadow, or classify a thin shadow as a crack.

Spall detection is less researched than cracks, as they are not as obvious to detect in images. [18] proposed a method for spall detection and assessment on concrete columns. They first isolated the spall region by a local entropy-based thresholding algorithm. Next, a global adaptive thresholding algorithm, template matching, and morphological operations were used in measuring the depth of spalling into the column and length of spalling along the column. This approach was found to overestimate the extent of a spall, as sometimes it considers large cracks as continuation of spall region. In addition, this method is limited to low-entropy photos, as background noise could interfere with detection. Despite this, limited further research has been done to detect spalls using knowledge-driven image processing approaches.

Little research exists to assess delaminations from a knowledge-driven point of view. This is likely because assessment requires infrared imagery, which is less of a developed field of research. However, it would be possible to develop a knowledge-based algorithm for delamination detection, the simplest being to just use a grayscale threshold to segment out the lighter/darker delaminations. Similar to other defect detection methods, this leaves a lot of room for potential error as other entities may appear lighter/darker in the image as well.

3.2 Hybrid Defect Analysis

As computation and efficiency of data-driven methods improved, computer vision research started to move towards machine learning. This boom happened around 2012 when [28] showed that a deep learning algorithm could achieve significantly better results on image classification. The structural inspection computer vision application community started working shortly after with hybrid methods utilizing both knowledge-driven and data-driven techniques for defect analysis. These methods provide a balance between understanding the characteristics of the defect, while also being able to remove many of the scene-dependent input threshold values that limit the ability for pure knowledge-driven methods to generalize to many applications.

For cracks, [46] proposed a hybrid method that used an image pixel intensity histogram-based analysis to identify regions of the image with cracks, as it was observed that histograms for 30x30 pixel regions containing cracks were wider. This is due to the large number of dark crack pixels contrasting light grey concrete pixels. Generating localized pixel intensity histograms is the knowledge-driven component. For the data-driven component, instead of using user-inputted rules to classify region histograms as a crack (e.g. classifying histograms wider than a certain threshold as a crack), a support vector machine algorithm was trained to automatically classify the region histograms as cracked/not cracked. This method was shown to perform better than a pure knowledge-based canny edge detection, but is still highly dependent on the environment (for example, a shadow over half of the region could also cause a histogram to be wide and resemble a crack). The authors admit that research needs to be done to test the ability of the algorithm to generalize.

[32] developed a state of the art solution for general damage detection that works well under varied lighting conditions. Their system first applies an automatic clustering method for segmentation. Clustering was tried using two different methods: Canny's edge detection method (knowledge-driven), and K-Means clustering technique (data-driven). K-means clustering was found to have better results. After segmentation, a machine learning classifier was shown to perform better than previous classification benchmarks. This method is more robust to shadows and changing environments, as the machine learning algorithm can be trained to recognize common sections incorrectly classified by the segmentation algorithm. For example, machine learning algorithms can learn that a shadow may have straighter edges compared to jagged cracks, after both were detected during segmentation. This algorithm provides evidence that the problem of general damage detection of can be considered a solved problem, depending on the size of damage and quality of image. Accuracy could be marginally improved, but the larger need is for an overall

inspection algorithm that can take the damage estimates from general images and present location and size statistics.

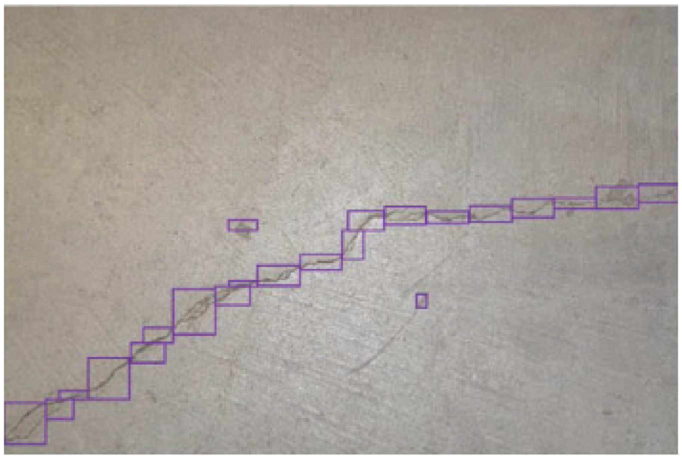
[42] presented an automated procedure for detecting delaminations in concrete bridge decks using infrared thermography. They first created a mosaic of thermal images of a bridge deck using a stitching algorithm (knowledge-driven). This mosaic was then segmented and objective thresholds identified using a K-Means clustering method (data-driven), to identify delamination locations based on changes in heating patters. This algorithm can create a condition map that effectively identifies delaminations within the bridge decks, but, although data-driven, the k-means clustering algorithm still requires human input for selecting the number of categories for clustering. This limits the algorithm to being only semi-automated.

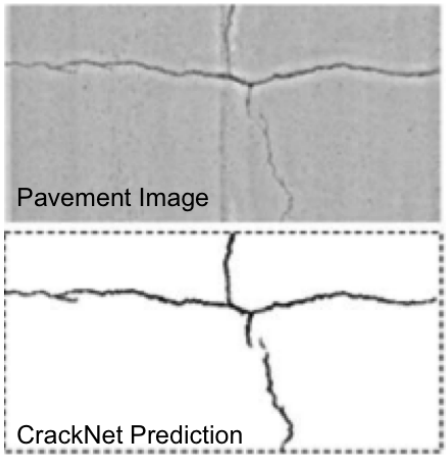
3.3 Data-Driven Defect Analysis

Most recently, data-driven CNNs have received traction amongst structural inspection community predominately for classifying images as containing a defect/no defect, but also for defect localization in an image. These networks are being widely used for augmenting civil infrastructure condition assessment in applications such as inspection and monitoring [54]. Specifically with regards to structural defect detection and/or localization, R-CNNs have been used more frequently for infrastructure inspection applications compared to semantic segmentation CNNs. These networks yield accurate prediction results, particularly when high-quality images of a defect are taken with minimal background noise, but provide only bounding-box localization of the area of interest. Research in semantic segmentation CNNs is more limited, but these provide more detailed localization predictions of the defect in the image.

As with knowledge-driven methods, cracks are the most researched defect for visual assessment using data-driven computer vision. [7] developed an R-CNN algorithm for detecting cracks in high-resolution images. This method is computationally inefficient, as it uses 256x256 pixel sliding window approach to classify each region as cracked/not cracked. The algorithm reported a high accuracy, but results were only tested on similar images, and thus it was not proven to generalize to varying environments. [8] also used Faster R-CNN [48] for assessing five defect types (four steel defects and concrete cracking). This algorithm is more efficient than [7], and showed good test results for concrete cracks (Fig. 3.1a). However, it was limited to images of two bridges and a building complex. In addition, this study primarily focused on steel bridges.

[60] proposed a method of semantic segmentation of cracks using CNNs (CrackNet), which automates crack detection on 3D asphalt surfaces (Fig. 3.1b) using a fully convolutional network. This network reported good accuracy, but the method is less efficient than common state-of-the-art for semantic segmentation as it does not incorporate any data encoding/decoding techniques. [59] implemented a similar CNN that improved upon the computational efficiency by using convolutions and transposed convolutions for the encoder/decoder architecture, but had worse accuracy than CrackNet. [1, 40] also present semantic segmentation CNNs for crack detection and localization, but again merely present improved accuracy and/or efficiency on their specific dataset. This highlights one of the current issues with machine machine learning research: the battle for the highest accuracy deters from the application (in this case crack inspection), which all of these networks are arguably sufficient for.


(a) Faster R-CNN on cracks [7]



(b) CrackNet [60]

Figure 3.1: Crack assessment results.

Applying CNNs to defects other than cracks is still uncommon. [34] presents a pixel-wise CNN that is trained to label spalls along with cracks, holes, and efflorescence. Focusing on spalls, the network achieves reasonable accuracy ($> 80\%$ mean intersection over union) but is trained and tested on carefully taken images of defects that contain few potential adversarial examples (based on examples shown). This method therefore works well if images are taken by trained inspectors, but will likely not perform as well with lower-quality images of defects. Larger data sets with more adversarial examples (examples that are likely to cause incorrect classification) are still required for spalls.

At the time of writing this thesis, the only work done to automate the detection of delaminations in infrared images using a CNN is the work presented here. [37] presents a lightweight CNN algorithm to automatically pixel-wise label delaminations in infrared images collected of bridge girders and soffits. Similar to the spalling research, this method obtains good results but is limited in data set size and network size (due to computational limitations), so will likely not generalize to all different delaminations unless retrained. Larger data sets with more adversarial examples are still required for delaminations.

3.4 Robotics and Visual Defect Assessment

The majority of the work done in assessing defects using computer vision focuses on obtaining high accuracy for labeling the defects in images. It is less common that the work considers the localization of the defect (e.g. is it on a girder? where along the girder is it?), and the quantification of a defect (e.g. if it is a spall, how big is the spall?). However, some work does attempt to address these issues using robotics and neural networks specifically, or with robotics and just general computer vision techniques.

3.4.1 Robotics and Neural Networks

[26] attempted to solve the localization problem by proposing the use of unmanned aerial vehicles (UAV) to collect images, and a R-CNN for data processing and damage analysis. UAVs are often equipped with a GPS, which creates the potential to geo-tag each image to provide an better estimate of defect location. This is limited, as many concrete structures (specifically the underside of bridges) create a GPS denied environment, thus preventing the UAV from properly navigating and from producing geo-tagged images. Hence, the inspection coverage would be limited to areas where GPS is available.

[34] attempted to solve the quantification problem by pre-calibrating a laser range finder with a camera using linear regression models to estimate the conversion of image pixel scale to physical area. This was shown to provide area estimates for cracks, spalls, efflorescence, and holes. However, such an approach is not practical when used for mapping large areas with mobile platforms, where it is difficult to satisfy the bounds or conditions implied in such pre-calibrated models. As well, if there is more than one surface in the image the range-finder cannot detect the distance to both surfaces, so the calibration will only be valid for one surface.

3.4.2 General Computer Vision and Robotics

Terrestrial laser scanners (TLS) have often been used to quantify defects in previous studies, e.g., [33, 35, 31, 57, 58]. TLS provide high-density colourized 3D representations of a structure, which can be useful for some aspects of structural engineering. For example, a stationary scanner can be placed on a construction site and run each day for accurate daily progress inspection. However, TLS are expensive and time consuming to employ and do not offer the same advantages in terms of mobility and speed such as drones or mobile ground vehicles. In a bridge inspection setting, these mobility limitations make full coverage difficult and time consuming. In addition, TLS are often proprietary products that do not provide access to raw data, therefore making it difficult to customize and automate defect inspection processes.

In terms of using mobile data collection platforms, much of the published literature focuses on the use of drones to collect image data, e.g., [15, 11] and creating 3D models using photogrammetric techniques, mostly structure from motion (SfM), e.g., [25]. However, defect quantification for bridge inspection using using SfM is fraught with challenges as images of bridges tend to be difficult to stitch as the structure lacks distinct features (e.g. flat and monotone concrete or asphalt), and the majority of the surrounding area is not static (e.g. vegetation, water, vehicles), which makes it difficult to match features between images. Mobile ground vehicles have been used in the following studies related to bridge inspections, e.g., [29, 19, 43, 45]. The process of building colourized and semantically labelled maps and with defect quantification have not been the focus of these studies. Other mobile platforms such as bridge climbing robot have been shown to be effective for carrying out inspections in steel bridges [30]; in the aforementioned study, the primary focus is on magnetic platform development and defect detection in images and not on building colourized or semantically enriched maps, and automated defect quantification.

3.5 Thesis Contributions

Significant work has been undertaken applying computer vision to structural inspections, but this does not mean that using computer vision for concrete defect inspection is a solved problem. Computer vision itself is still rapidly improving in terms of accuracy, computational efficiency, and adding physical meaningful/scale to images. This thesis aims to take the state-of-the-art in computer vision and apply it to structural assessment of spalls and delaminations to improve the traditional practices of routine bridge inspection. The contributions of this thesis are:

- A semi-automated defect labeling tool for rapid labeling of images for training semantic segmentation CNNs (Section 4.3.1)
- Implementation of MobileNetV2 and Deeplab V3 semantic segmentation CNN for pixel-wise labeling of delaminations and spalls (Section 4.3.2)
- Occlusion-aware labeling of a 3D point cloud using image colour data and CNN defect labels to enrich defect detection with defect localization (Section 4.4)
- An objective and repeatable procedure for automatic defect area quantification from a semantically labeled point cloud (Section 4.5)
- Analysis of the effects of errors in image defect labeling on real-scale defect size quantification in a point cloud (Section 5.3)

Chapter 4

Methodology

The proposed methodology for automating delamination and spall area size quantification can be broadly divided into five components, represented by the block diagram shown in Fig. 4.1. These components consist of a mobile data collection platform complete with multiple sensors, a mapping component to fuse data from various sensors into a common reference frame in order to generate a 3D map and pose trajectory, a defect labeling component to automatically detect and label defects in images, a map labeling component to semantically enrich the map 3Dwith colour and pixel information from images, and finally the defect quantification component. The data collection platform and the the mapping are outside the scope of this thesis. However, the general overview of these components with key details required to understand the aspects relevant to this thesis, namely defect labeling, map labeling, and defect quantification, are explained for the sake of completeness.

4.1 Data Collection

The primary data required to implement the proposed automated defect assessment procedure is spatial data from a laser scanner (e.g. lidar) for mapping and visual data from cameras for defect labeling. Fusing the data from these sensors via map labeling provides the necessary geometry and colourization to be able to detect, locate, and quantify defects.

To enable the collection of data for inspection, The Structural Dynamics Identification and Control (SDIC) research lab developed a mobile platform capable of operating in harsh terrains, including under bridges [9, 38]. The platform used is an unmanned ground vehicle

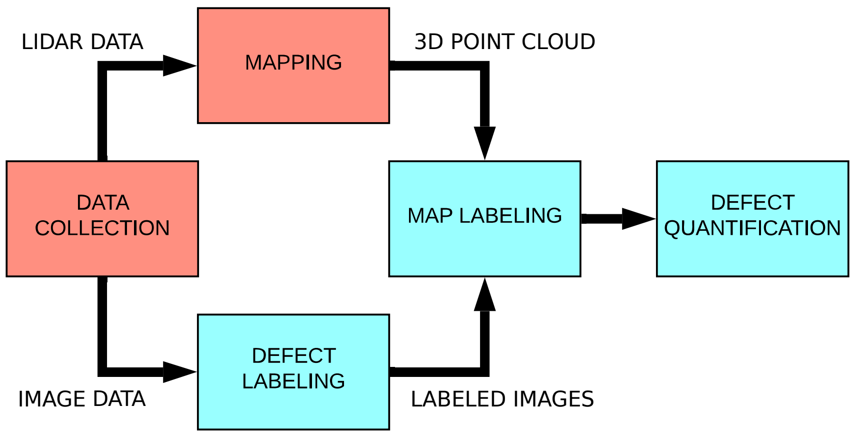


Figure 4.1: Block diagram of the automated inspection pipeline. Red boxes represent work done primarily by other lab members. Blue boxes represent work presented in this thesis.

(UGV) produced by Clearpath Robotics, from Kitchener, Ontario. The platform is called the Husky (Fig. 4.2), and is capable of reaching 1 m/s velocity with four rubber pneumatic tires, a self-weight of 50 kg and payload of 75 kg.

To enable data collection, the husky platform is equipped with:

- two Flir Blackfly red-green-blue (RGB) cameras with wide FOV (185 degrees)
- one Flir Blackfly RGB camera with narrow FOV (40 degrees) lens
- one Flir ADK Infrared (IR) camera
- two Velodyne VLP16 lidars
- one Swift Nav Real-time Kinematic (RTK) GPS
- one Xsens MTI-30 inertial measurement unit (IMU)

For the work in this thesis, the Flir Blackfly RGB camera with narrow FOV is used for visual image data, the Flir ADK Infrared (IR) camera is used for infrared image data, and the two Velodyne VLP16 lidars are used for spatial data. An on-board computer is used to control the data collection, to run the sensor drivers, and to control the husky UGV.

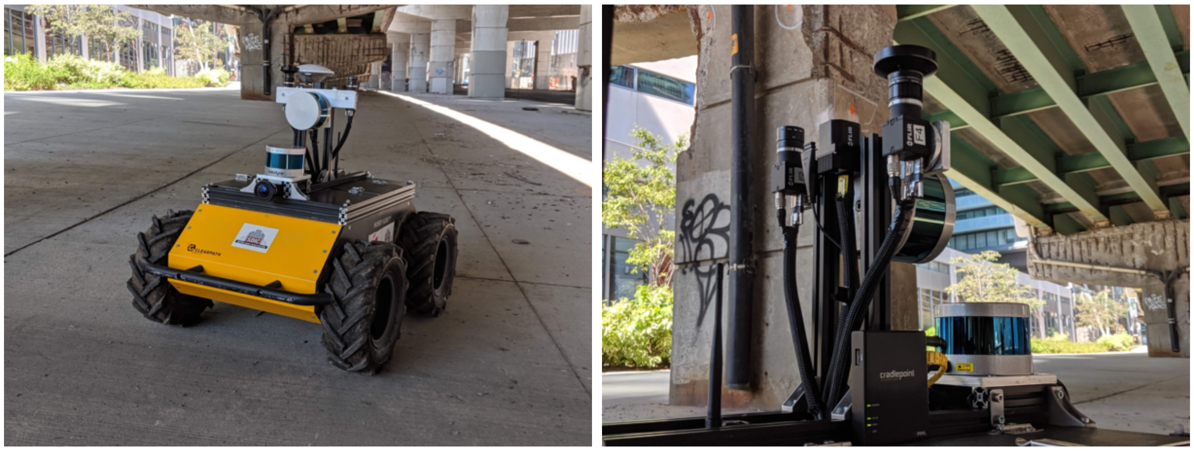


Figure 4.2: Husky Inspection platform. (Photos by: Nicholas Charron)

Automated inspections require the data collected from different sensors to be integrated, both spatially and temporally. Key aspects to being able to fuse the information obtained from different sensors are synchronous sensor measurements and accurate calibrations. To achieve these, hardware for time-synchronization hardware and sensor calibrations (intrinsic and extrinsic parameters) have been undertaken. Time-synchronization is achieved by a custom printed circuit board (PCB) developed to control the sensor capture rates to consistent time intervals. This is done by sending a hardware trigger signal from the PCB to different sensors synchronously. For example, consider a lidar scanning at 10Hz and a camera capturing photos at 1Hz. Time-synchronization in this situation is achieved by sending the hardware trigger signal at time intervals to ensure that the camera consistently captures an image every 10th lidar scan. Therefore, every 10th lidar scan has an identical timestamp to a captured image. This time-synchronization is necessary for fusing data from different sensors that are mounted on a mobile platform. The custom PCB was developed by other lab members, and further details are available in [9].

Calibration is separated into extrinsic and intrinsic calibration. Intrinsic calibration focuses on calibrating the internal parameters of a single sensor. These intrinsic parameters are used to abstract the raw sensor data into a format that is more intuitive to a user. The Velodyne VLP16 lidars are intrinsically calibrated on purchase, and their collected data is provided as 3D points relative to the lidar center. The Flir Blackfly RGB camera and the Flir ADK Infrared (IR) camera must be calibrated for their intrinsic camera matrix 2.5 to understand how the (u, v) pixels were obtained from a 3D environment. These intrinsic parameters are calculated using the Matlab camera calibration toolbox [4] for calibration of a single camera using the common checkerboard target method. Camera intrinsic calibration details are outlined in appendix A.1. The Blackfly RGB camera is calibrated using the basic checkerboard target shown in 4.3a. The Flir ADK Infrared camera is calibrated using a custom checkerboard shown Fig. 4.3b as a target. This target can be used for infrared calibration as the flat black paint (high emissivity) and aluminum (low emissivity) have high contrast in emitted energy in the infrared band.

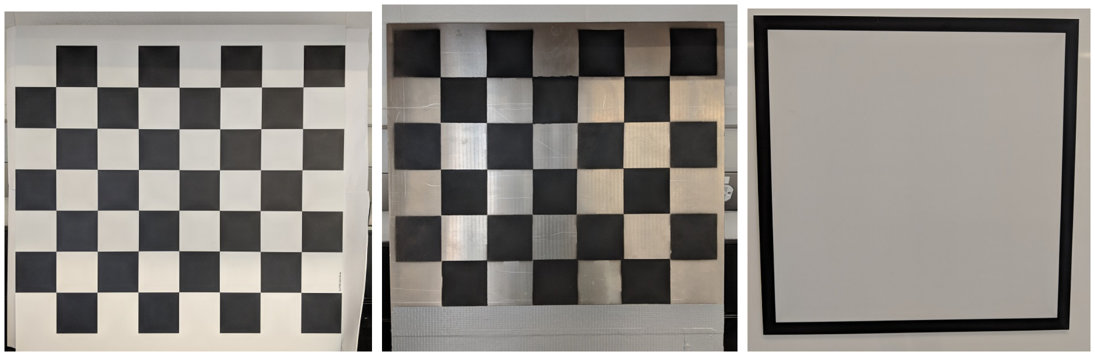

(a) Vision camera target (b) IR camera target. (c) Lidar target.

Figure 4.3: Custom calibration targets.

Extrinsic calibration involves finding the transformation from points in one sensor's coordinate frame to another coordinate frame. This is calculated using a transformation matrix, given by

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_A = \mathbf{T}_{AB} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_B \quad ; \quad \mathbf{T}_{AB} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{33} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

where r_{ij} are the 3x3 rotation matrix coefficients for transforming the point and t_x, t_y, t_z represent the translation information for transforming the point. This transformation matrix enables the transformation of data in one sensor's coordinate frame to another sensor's. The extrinsic calibration between the RGB and the infrared cameras is refined using the Matlab stereo camera calibration toolbox [4]. Further details on stereo camera calibration are presented in appendix A.2. For this thesis, the same checkerboard for calibration of the infrared camera intrinsic data is also used for stereo calibration (Fig. 4.3b), as it displays the checkerboard pattern in both the infrared and vision images, taken synchronously.

The vision camera to lidar calibration is first estimated visually and then refined using the square target shown in Fig. 4.3c. Scans are taken with the camera and the vertical lidar directed at the stationary square target. The points in the 3D point cloud generated from the individual lidar scans are then projected onto the image plane using the camera to lidar transformation matrix, the camera intrinsic matrix, and the pinhole camera model. These points are overlaid onto the image to visually determine the accuracy of the extrinsic calibration. The transformation matrix is then manually adjusted for rotation and translation until the location of the target in the lidar scan visually aligns precisely with the location of the target in the image. However, more information than just the transformation matrix is required to adequately transform points from an image to a lidar, since the image data is 2D and the lidar data is 3D. This is one of the contributions of this thesis, and will be covered in Section 4.4.

4.2 3D Mapping

The 3D mapping portion of this work involves taking all the spatial data collected from the two Velodyne VLP16 lidars and aggregating it into one 3D representation of the scanned structure. The main technical challenge is that the vehicle is often moving while the scans are collected, so the data during each scan is collected in different coordinate frames. In order to combine all the scans, the scans need to be transformed into the same coordinate frame. This can be achieved by knowing the exact location that each scan was taken, and the transformation matrices (in the form of 4.1) required to transform the data between different locations.

The algorithms used to solve the 3D mapping problem in this automated inspection procedure come from the concept of simultaneous localization and mapping (SLAM), for which comprehensive studies are available [14, 60, 56, 53]. The SLAM algorithms used for this work are outside the scope of the work undertaken by the author and are therefore not described in detail in this thesis. Readers can refer to [9, 38] for in-depth review of how 3D mapping is performed for this application. However, it is important to describe a conceptual overview of SLAM and 3D mapping to understand how image information (pixel information and output from computer vision techniques) can be fused with a 3D map for reference-free scale in images. This understanding can be achieved with a high-level overview of coordinate transformations, and an understanding of how a 3D map is stored after completion.

The first aspect to understand is coordinate transformations. These are the same as the coordinate transformation discussed in calibration (Equation 4.1), and are represented as transformation matrices. In calibration this is used to transform between the coordinate frames of two different sensors (e.g. lidar to camera), but in the case of localization it can also be used to transform data from the same sensor into the same coordinate frame if that sensor has moved (e.g. transform data from the lidar obtained at point B to the coordinate frame of the lidar when it was at point A). Performing this transformation is relatively simple, but the complexity arises in the calculation of the transformation matrix. This is where SLAM is required, as it estimates these transformation matrices for the robot platform (and therefore the lidar, since its pose is static with respect to the platform) as it moves. This enables every lidar scan to be transformed into the coordinate frame of the original scan. Once all the scans are converted to the same coordinate frame, an aggregated 3D map of the scanned structure can be obtained. More post-processing is usually undertaken to remove noise and to further refine the maps, and information is presented in [9].

The second important aspect is to understand how a 3D map is stored. The output of SLAM in this application is a 3D map of a concrete structure (mapping aspect) and the location of the robot at the time of each sensor measurement (localization aspect). The 3D map is a collection of points with just (x, y, z) spatial data that represents the scanned structure. Any additional semantic data (e.g. colour, defect presence) is added during map labeling. The localization is a set of poses (position and orientation of the robot) with $(x, y, z, \phi, \theta, \psi)$ points at each time a sensor reading was taken. Using the calibration and localization it is possible to know the pose of the camera in the same coordinate frame as the generated 3D map at each time an image was captured. The 3D map and the pose of the camera at the time an image is captured are subsequently used to fuse image data with lidar data for automated defect assessment.

4.3 Defect Labeling

Defect labeling is a critical step for estimating the area/size of a defect and its accuracy determines the quality of area measurement results. It is possible to consider defect labeling using camera data, lidar data, or both. However, given the significant research in computer vision for object detection/localization, and the fact the the camera data contains more dense and valuable information for detection of visual defects (specifically delaminations and spalls), defects were chosen to be labeled using image data alone.

Defect labeling in the image requires a method of labeling which pixels represent a defect (e.g. spall/delamination) and which represent background (e.g. sound concrete, sky, ground). This is achieved by using a binary image mask, which is a binary image with the same pixel density (e.g. same height and width) as the original image, but with binary pixels (1 or 0) to represent defect (1) or background (0). There are many ways to obtain these masks, and this thesis will cover both manual labeling and automated labeling via a CNN.

4.3.1 Manual Labeling

One option for labeling a defect is to manually select which pixels in the image represent a defect or no defect. Manual labeling can be used with the proposed inspection pipeline, but eliminates full-automation from the defect labeling step. Therefore, manual labeling is not used in the final procedure, but is a key step towards training a state-of-the-art CNN to automate defect labeling.

To train a CNN to detect objects in images at a pixel level, each training image needs all the pixels to be labeled as one of the classes. In this case, only two classes are used and therefore a binary image mask can be used for each case of:

- Infrared Images: (0) sound concrete, and (1) delamination
- Visual Images: (0) sound concrete, and (1) spall

Semantic pixel-wise image labeling is a tedious task, even with only two classes. To assist with fast labeling, a graphical user interface (GUI) was created in MATLAB [36]. The goal of this GUI is to be generic enough that it can be used to label images for delamination and spall defects, but could be extended to label any number of defect types on concrete bridges (e.g. cracks). This GUI makes use of basic image processing techniques to provide an initial estimate of the defect locations, then provides tools to manually fine-tune the labeling.

Fig. 4.4 shows the process used to provide the initial estimate of the pixel labels for the delamination defect, prior to manual editing. First, the image is converted to grayscale ($m \times n \times 1$ image with pixel values between 0 and 255) if it is not already in that format. This is required as a grayscale thresholding technique is used to generate the binary image mask that estimates the defect segmentation. The grayscale thresholding value can be adjusted between 0 and 255 by using the slidebar to select the desired thresholds of grayscale intensities to include. By default, the thresholding sets all pixel values below a certain grayscale threshold to 1, and the rest to 0 (e.g. if the threshold is 100, all grayscale values below 100 are set to 1, and all grayscale values above 100 are set to 0). For example, spalls may appear as darker than surrounding concrete, so setting a low threshold to include only dark pixels as a defect will give a good initial estimate of which pixels are spalls.

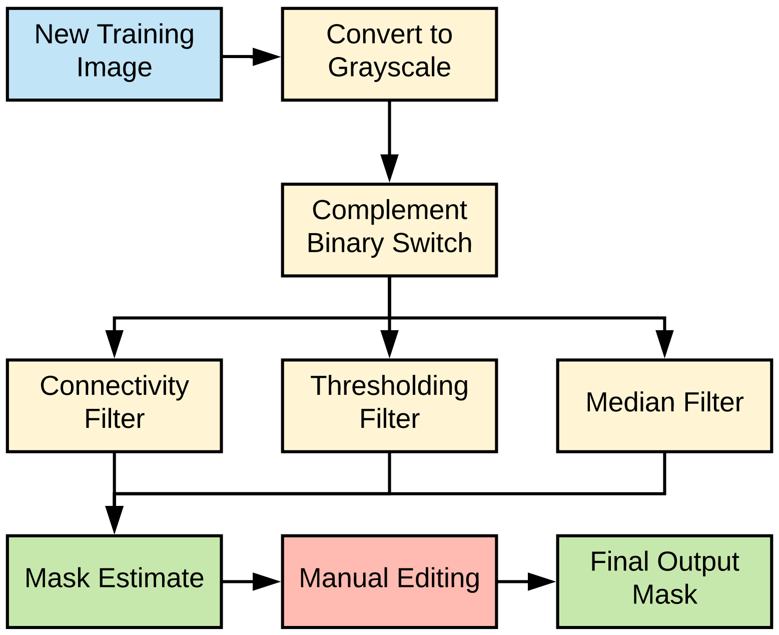


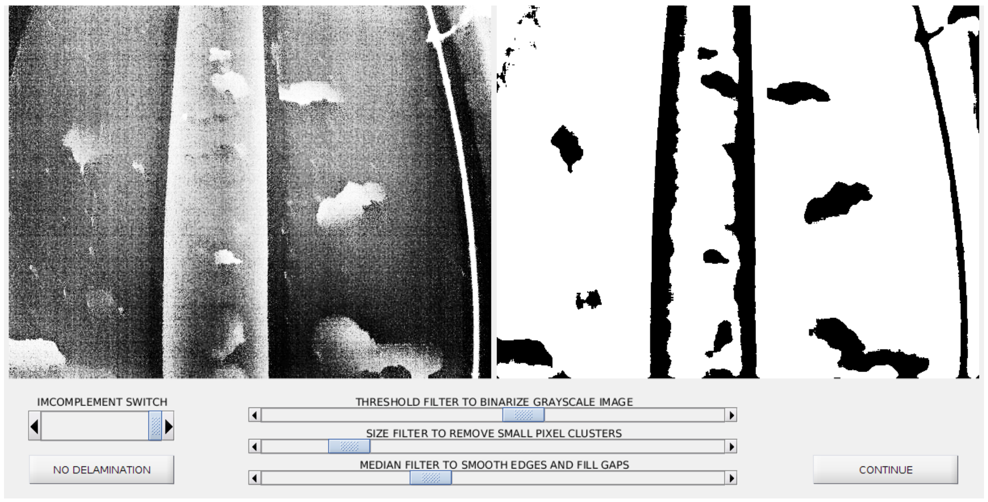
Figure 4.4: Flowchart of the image processing techniques used to simplify manual pixel-wise image labeling. Blue blocks represent inputs, yellow blocks represent image processing techniques, green blocks represent outputs, and red blocks represent manual tasks.

However, not all defects are always the darkest in the image. The GUI's complement binary switch option allows the user to switch the thresholding to set the grayscale values above the set threshold to a value of 1 in the binary mask estimate. For example, a delamination may appear lighter than the surrounding concrete, so the user can activate the complement binary switch and set a high threshold to segment only the lightest delamination pixels to be part of the mask estimate representation of the defects in the image.

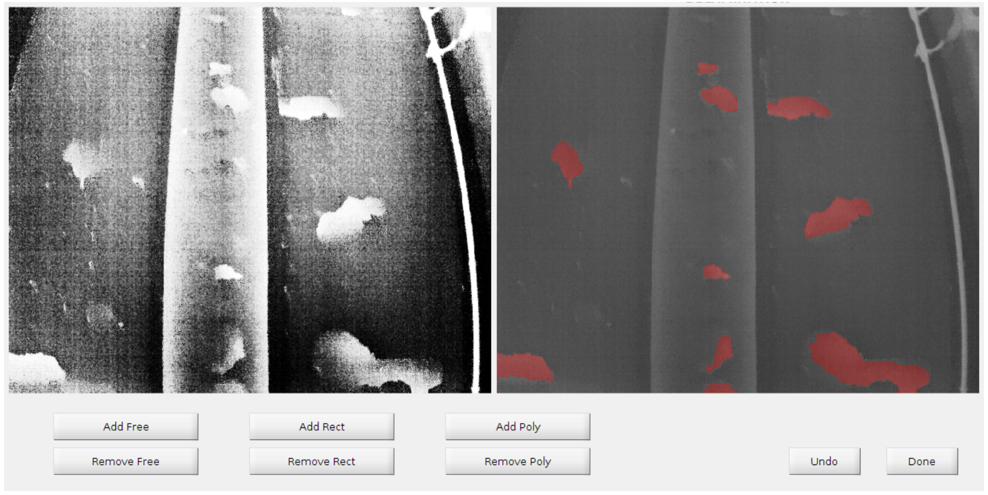
The other two image processing filters used for refining the binary mask are the connectivity, and median filters. Each of these filters is applied using a sliding adjuster to change the controlling attribute of the filter. The connectivity filter removes pixel clusters below a certain threshold value (e.g. if the filter is set to 100 pixels, all pixel clusters smaller than 100 are removed). This is because noise tends to be scattered, and not a part of large connected clusters, whereas defects tend to be large blocks of connected pixels. The median filter passes a filter over the image that replaces the pixel with the median value within a $n \times n$ window around the pixel. The sliding adjuster changes the size, n, of the filter. The median filter removes jagged edges and makes for a more smooth image mask, with a larger value of n providing more smoothing. The value of n is therefore adjusted to find a desirable balance of smoothed defect boundaries while still maintaining the original defect shape. Fig. 4.5a shows the graphical interface for the image labeler. The top image shows the use of the thresholding, connectivity, and median filter on an infrared image to estimate the delamination mask.

These image processing techniques provide a good pixel-wise estimate of defect locations (right side of Fig. 4.5a), but have several shortcomings. First, artifacts in the image will be included during thresholding and cannot be removed by other filters. For example, shadows may cause certain sections of the image to appear as dark or as light as the defect, causing them to be included as a defect in the binary mask. Shadows are an example of an object that often requires manual removal from the mask. Second, small gaps in the detected defect area cannot be corrected by the connectivity or median filters, but should be closed in order to make the defect a single entity. These examples outline the limitations of these image processing methods and outline the need for manual alterations to the mask (4.5b). The manual alteration step allows for areas to be added or removed by drawing freehand, rectangular, or polygonal shapes. This is typical as to what is provided by a generic labeling tool, but the fact that the binary mask estimate is already provided significantly reduces the amount of time spent on the manual drawing step. Appendix C provides an in-depth example of labeling an image from start to finish.

The described labeling approach allows for accurate and rapid labeling, which is necessary to be able to generate a large and accurately labeled data set. This approach has


IMCOMPLEMENT SWITCH
THRESHOLD FILTER TO BINARIZE GRAYSCALE IMAGE
SIZE FILTER TO REMOVE SMALL PIXEL CLUSTERS
MEDIAN FILTER TO SMOOTH EDGES AND FILL GAPS
NO DELAMINATION
CONTINUE

(a) Sliders to adjust the threshold used for each image processing technique. Adjust until desired delamination mask estimate is obtained.



(b) GUI tools to manually adjust image mask after estimate. Areas can be added or removed using freehand, rectangular, or polygonal shapes.

Figure 4.5: Matlab labeling GUI interface

also been tested for labeling cracks, spalls, corrosion stains, and exposed reinforcement in concrete structures. It can be used as part of the inspection process if manual labeling is preferred for its high accuracy at the cost of loss of automation, but can also be used to develop training data for a fully automated labeling method for a more rapid and automated inspection procedure.

4.3.2 Automated Labeling

To fully automate the inspection procedure of spalls and delaminations, a fully automated method of labeling spalls and delaminations in images is required. A concrete spall is easily detectable in an image as it contains clear visual features, which can be extracted using image processing and machine learning techniques. Delaminations, however, are not easy to detect in the visible spectrum, but can be detected in the infrared spectrum images as subsurface voids create noticeable variations in thermal conductivity.

A convolutional neural network (CNN) is a state-of-the-art machine learning tool for automatic labeling of data in an image. A CNN is a type of neural network that uses the convolution operator in place of typical fully connected network layers. CNNs are commonly applied to analyze inputs that have a grid-like data structure, such as images [21], which are used in this application. A CNN for pixel-wise segmentation of an image takes a single image as an input, and outputs a mask with the same pixel density as the original image. The pixel elements in the output mask are integers that represent the defect label (e.g. for spall or delamination the mask is: 0 - no defect, 1 - defect). There is no mathematical limit to the number of classification categories, but the amount chosen can affect the quality of results.

The proposed inspection methodology is arranged such that any manual or automated defect labeling method that produces a pixel-wise mask can be used. Therefore, at the defect labeling step it is possible to integrate with any pixel-wise labeling technique of choice, including manual labeling using the presented MATLAB labeling tool. It would also be possible to integrate with any of the defect labeling CNNs presented in [59, 40, 1, 34]. The choice of image labeling method depends on the specific computation and accuracy requirements of the desired defect assessment. For example, one could use a computationally efficient CNN with slightly lower accuracy for frequent routine inspections and a more robust, computationally expensive CNN that achieves high accuracy for a more detailed infrequent inspection.

For proof of the entire inspection process concept, it is necessary to implement a fully-automated network to show that the inspection process as a whole can be fully-automated.

This network must obtain reasonable accuracy, and foster reasonable quantification sizes. In this thesis, the CNN implemented is MobileNetV2 [51] as a feature extractor with Deeplab V3 [10] to perform semantic pixel-wise segmentation of defects in images. This network architecture is a pixel-wise CNN, which was developed to prioritize time-efficiency and minimize hardware requirements, while maintaining reasonable accuracy. This network architecture is chosen for this application since it is computationally efficient, but still competitive with the deeper networks that produce the current state-of-the-art in accuracy. Additionally, the results from the CNN are intended to be fused with lidar data, which downsamples the resolution. In other words, the fine details of the defect geometry are limited by the lidar resolution, not the image resolution.

The main network details are explained in Appendix B, and full details are available in [51, 10]. Some specific adjustments are made to suit this specific inspection application, and are outlined here. First, the input size is adjusted to a 512x640 image for the delamination labeling and a 512x640x3 image for the spall labeling. Second, the final output layer of the network is programmed to be a pixel-wise softmax activation, given by the equation

$$z_i = \frac{exp(x_i)}{\sum_{j=1}^{n} exp(x_j)} \tag{4.2}$$

This is used to convert the $i \in \{1, ..., n\}$ output values (x_i) for each label category to a score between 0 and 1 (z_i) [21]. This is applied to each pixel individually. The closer the value is to 1 for a given category (e.g. defect or no defect), the more confident the network is that the pixel belongs to that specific category.

Restricting the scores of each pixel to be between 0 and 1 enables the model to be trained using the categorical cross entropy (CCE) loss function, given by the equation

$$J = -\sum_{i=1}^{n} y_i log(z_i) \tag{4.3}$$

This is used to minimize the loss (J) across all label categories $(i \in \{1, ..., n\})$ given the output of the network with the current weights (z_i). This also depends on y_i, which is a binary label indicating if the pixel does $(y_i = 1)$ or does not $(y_i = 0)$ belong to class i [21]. The loss is calculated on a pixel-wise basis. CCE is chosen because it harshly penalizes areas that should be labeled as defect but aren't. This is critical, as since defects tend to be sparse in the image, labeling all pixels as no defect could easily give the appearance of high accuracy. This loss function specifically prevents that error from occurring.

For this thesis, the network is trained using the ADAM optimizer [27], with a learning rate of 0.003, a β_1 of 0.9, and β_2 of 0.99. This optimizer is similar to stochastic gradient descent, but with a more intelligent update to avoid local minimums. The ADAM optimizer algorithm is detailed in appendix A.3. The batch size used is 4 (this is limited by computer hardware), and the number of training epochs is 250. Random image transformations are used at the start of each training epoch to help prevent overfitting. Before each epoch, the training images are shuffled and random transformations are applied to the training images. These randomized image transformations include: flip horizontal, flip vertical, rotation, width and height shift, and zoom. Random transformations ensures a high likelihood that the network will not see the exact same image twice during training, which is crucial for limiting overfitting when training on smaller datasets.

Computer Hardware

A CNN approach to pixel-wise segmentation requires significant computational resources to ensure training the CNN is possible at a reasonable speed. Specifically, the graphics processing unit (GPU) is crucial for training a neural network. A GPU is preferred over a typical central processing unit (CPU) for training a neural network because a GPU contains thousands of small computing cores, making them specialized at performing parallel operations. A typical CPU in an average computer contains 4 to 8 large computing cores, making them specialized at rapidly performing single ordered operations. A CPU therefore can perform a single operation faster, but a GPU can perform more operations synchronously. Training a neural network involves a large amount of simple operations, such as matrix multiplications and additions, that can be performed in parallel. The ability to perform these operations in parallel results in the GPU significantly outperforming the CPU for training neural networks. Additional software is also developed for GPUs to make them specialized for the parallel operations required in training a neural network. For this work, a custom built computer with an Nvidia GTX 1080 TI GPU programmed to take advantage of Nvidia's CUDA and cuDNN software for accelerated training of neural networks was used. This is a significant improvement over the majority of computers, but still has limitations in training time, network size, and batch size.

The first limitation is training time. Most large networks have access to multiple GPUs or to GPU clusters for training which further accelerates training time. With only one GPU, testing a change takes long periods of time, but does not place hard limitations on the quality of the results. The next two limitations - network size and batch size - are based on the limitation of the GPU memory. This is because when training, the batch size and network size are limited by the GPU memory. In this case the GTX 1080 TI has 11GB of memory, which is one of the main factors in choosing to use the MobileNet V2 and Deeplab V3 CNN, as it requires less computational load relative to state-of-the-art CNNs.

In addition, the training batch size was limited to only 4 images. This is a limitation, as a larger batch size often yields faster training convergence, and better avoidance of local minimums and overfitting when training. However, it is sufficient for the proof of concept presented, as the goal is not to achieve the most accurate network but rather to show that automated inspections can be performed with any desired network.

4.4 Map Labeling

The literature review makes it evident that most research applying CNNs to defect inspection focuses on defect detection in images, rather than on the quantification of the defect in physical scale. This is because the scale ambiguity resulting from images being a projection of a 3D scene onto a 2D plane makes it difficult to estimate real-world scale from images alone. This work focuses on combining defect information in images with lidar spatial data to enable reference-free defect scale-determination from data obtained using the mobile data collection platform. The proposed method for combining image and lidar data uses ray-tracing for map labeling that is able to distinguish which pixel should be labeled with image data when an occlusion is present. This algorithm uses the robot extrinsic calibrations, camera intrinsic calibrations, and robot trajectory coordinate transformations provided from SLAM to add colour and CNN information to a generated 3D map. This information can be transferred to the cloud using the continuous $(x, y, z, \phi, \theta, \psi)$ pose estimate from the SLAM process, and the camera positions relative to the robot from the calibration process. The transformation between the robot frame and the map frame is exported automatically during the mapping process each time an image is captured.

To label the cloud, coordinate transformation can be used to map 3D cloud coordinate points to an image pixel that captured that 3D point. The colour data from the image pixel can then be assigned to the corresponding point in the point cloud. One basic method for mapping cloud points to image pixels is the projection method, where the 3D cloud points are projected to the image plane (i.e. to u, v coordinates). This would be done using the pinhole camera model [55] combined with the transformation from the map frame to the camera frame. Once converted, the projected points can be labeled based on their location relative to the pixels on the image plane. The projection from 3D coordinates to a pixel coordinate can be done using Eq. 4.4:

$$s \begin{bmatrix} u & v & 1 \end{bmatrix}^T = \mathbf{K} \begin{bmatrix} \mathbf{R}_{CM} & \mathbf{t}_{CM} \end{bmatrix} \mathbf{P}_M^k \tag{4.4}$$

where s is a scaling constant, u, v are the integer pixel coordinates on the image plane,

$\mathbf{K}$ is the intrinsic matrix (Equation 2.5), $\mathbf{R}_{CM}$ and $\mathbf{t}_{CM}$ are the respective 3x3 rotation and 3x1 translation matrices that correspond to the transformation from the map frame ($\mathcal{F}_{\mathcal{M}}$) to the camera frame ($\mathcal{F}_{\mathcal{C}}$), and $\mathbf{P}_M^k$ is a homogeneous coordinate $(X, Y, Z, 1)$. $\mathbf{P}_M^k$ is the k^{th} homogeneous coordinate point in the point cloud, presented in the map frame $\mathcal{F}_{\mathcal{M}}$.

A few problems arise when using this projection process alone for map labeling. First, this method does not account for occlusions, which becomes an increasingly bigger issue as the cloud size and complexity increase. For example, points on different surfaces (e.g. point on girder and soffit) may project to the same pixel location on the image plane, but the image can only see one of those planes if one is behind the other (Fig. 4.6). This occurs because of the scale ambiguity of the pinhole camera model. Thus, the projection method is extended to an alternative approach, where points on the image plane are mapped to corresponding points on the 3D point cloud using the inverse of the pinhole camera model.

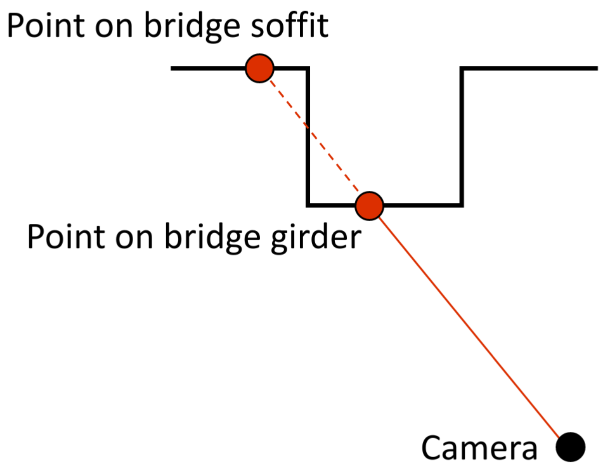


Figure 4.6: Example showing how occlusions can cause incorrect labels. Both the bridge soffit and girder points will project to the same u, v pixel, causing the soffit to be erroneously labeled with the colour information of the girder. Therefore, the method of projecting a 3D point to the 2D plane is not adequate for transferring colour information from the image to the point cloud.

This map labeling process iterates through all (u, v) pixels of all images taken. The first step of this process is to remove distortion in the current image. This is done using

Eq. 4.5 and the distortion parameters obtained during camera intrinsic calibration [24].

$$\begin{aligned}\widetilde{u} &= u(1 + k_1r^2 + k_2r^4) + 2p_1uv + p_2(r^2 + 2u^2) \\ \widetilde{v} &= v(1 + k_1r^2 + k_2r^4) + 2p_2uv + p_1(r^2 + 2v^2) \\ r^2 &= u^2 + v^2\end{aligned} \tag{4.5}$$

where, $\widetilde{u}, \widetilde{v}$ are the distorted image coordinates, k_1, k_2 are the radial distortion coefficients, and p_1, p_2 are the tangential distortion coefficients.

The second step is to convert the un-distorted pixel coordinates on the current image plane to points in the camera coordinate frame. To do so, the inverse of the camera matrix, $\mathbf{K}$, is taken and pre-multiplied on both sides of equation 4.6 to isolate $\mathbf{P}_C^{u,v}$ - the coordinates of the $(u, v)^{th}$ pixel point in the camera frame $\mathcal{F}_\mathcal{C}$ for the current image. Note that equation 4.6 differs from equation 4.4 since the points are being calculated in the camera frame, therefore $\mathbf{R}_{CM}$ and $\mathbf{t}_{CM}$ are not applied. Isolating for $\mathbf{P}_C^{u,v}$ yields equation 4.7. Also, notice that equation 4.6 is iterated through each $(u, v)^{th}$ pixel as opposed to equation 4.4, which is iterated through each individual point in the cloud.

$$s \begin{bmatrix} u & v & 1 \end{bmatrix}^T = \mathbf{K}\mathbf{P}_C^{i,j} \tag{4.6}$$

$$\mathbf{P}_C^{i,j} = \begin{bmatrix} x & y & z \end{bmatrix}^T = \mathbf{K}^{-1} s \begin{bmatrix} u & v & 1 \end{bmatrix}^T = s \begin{bmatrix} \frac{u - c_x}{f_x} & \frac{v - c_y}{f_y} & 1 \end{bmatrix}^T \tag{4.7}$$

The scaling constant, s, is equal to the distance from the camera optical center to the image plane in this application, and is obtained by multiplying the focal length in pixels by the physical pixel size, which is generally known from the camera manufacturer.

Once the u, v points are represented in the camera coordinate frame, it is possible to convert the coordinates of the camera optical center—located at (0,0,0) in the camera frame—and the pixel 3D coordinates ($\mathbf{P}_C^{u,v}$) to the map coordinate frame. This is done using the transformation between the camera and the robot frame, $\mathbf{T}_{CR}$, and the transformation from the robot frame to the map frame, $\mathbf{T}_{MR}$. A linear ray direction vector can then be calculated by subtracting the camera optical center from the pixel point on the image plane. The ray is extended in the calculated direction until this contacts with (or comes within a specified threshold distance of) a point on the point cloud, as illustrated in Fig. 4.7.

For each pixel, the calculated ray is extended incrementally and a KD tree search algorithm [3] is performed at each iteration to check if the endpoint of the ray is within

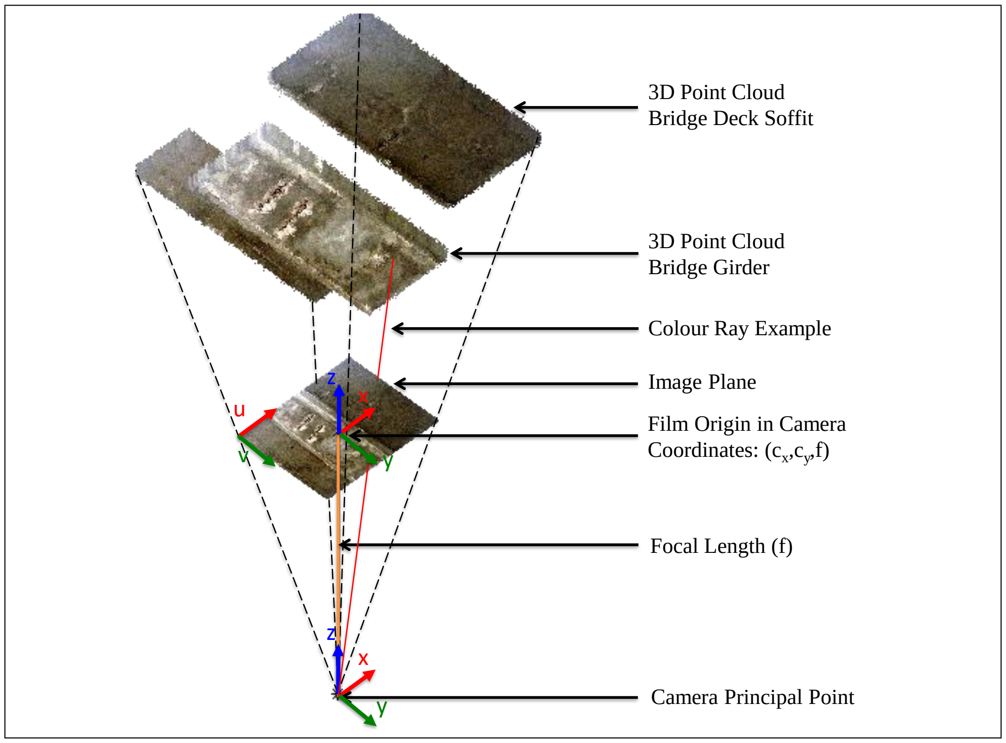
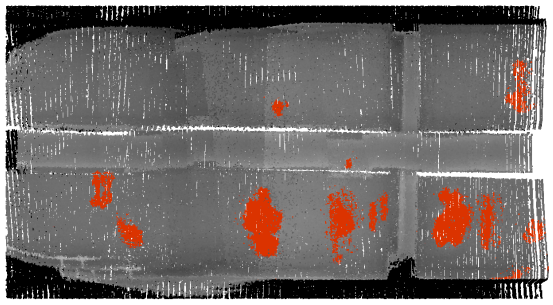


Figure 4.7: Point cloud colourization illustration

a specified distance threshold from a point in the point cloud. This process is accelerated in two ways. First, by eliminating all points in the cloud which are not located within the image plane (i.e. have u, v coordinates outside of the active area) when projected using the camera model in equation 4.4. This ensures that the search does not check points irrelevant to the current image. Second, the rays are incrementally extended by a distance equal to the distance between the ray endpoint and the closest point on the reduced cloud. After each ray extension, if a point is found to be within the threshold distance from the ray endpoint, then that point is assigned the colour and CNN information of the pixel. Otherwise, the ray is extended. This process continues until the current pixel is successfully mapped to a cloud point, or until a set number of maximum iterations is reached. This incremental ray extension and contact check process is necessary, instead of immediately colourizing the closest point to a pixel, because the closest point may not necessarily be along the path of the ray. This algorithm thus ensures that the cloud point coloured is the first point that the ray contacts, which mimics the path light would actually take from the point to the camera sensor.

The last aspect of the point cloud labeling method is an image blending technique. Since the image is significantly more dense than the point cloud, it likely that multiple pixels may map to the same point. Hence, the algorithm should have a method of determining which pixel contains the most relevant information (e.g. colour, CNN labele) when multiple pixels ray-tace to the same 3D point in the point cloud. In the case of this algorithm, the technique used is to take the pixel information that comes from when the principal point of the virtual image plane is closest to the 3D point being labeled in the point cloud. In other words, the image information captured when the camera was physically closest to the 3D point being labeled will be given priority when assigning pixel information. The idea behind this approach is that the highest quality image data should come from the closest camera pose. This also helps remove distortion effects, as the closest points to the camera pose are often the points near the center of the image where there is less distortion. Fig. 4.8 shows the improvement obtained by using closest distance blending. Algorithms 1 (setup), 2 (functionality) summarize the of the ray-tracing colourization process.

This colourization process can be done automatically with both the vision and the infrared images. Fig. 4.9 shows a bottom view of the coloured point cloud using both cameras. Note that the vision images are in colour, but the lack of exposure under the bridge and the fact that only concrete is photographed makes the colourized point cloud primarily gray. In both sets of colourized clouds, the colourized portions depict a concrete girder running in the approximate centre of the colourized portion, with concrete deck soffit on either side.


(a) With closest distance blending

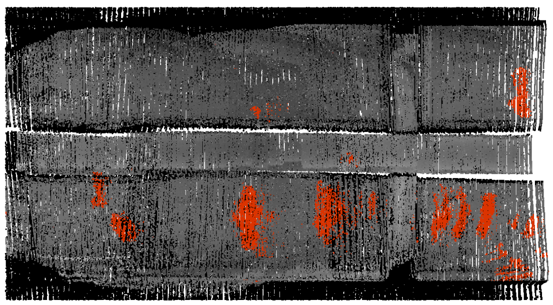

(b) With no blending

Figure 4.8: Comparison of the ray-tracing technique with blending vs. no blending.

Algorithm 1 Raytrace Map Labeling - Setup

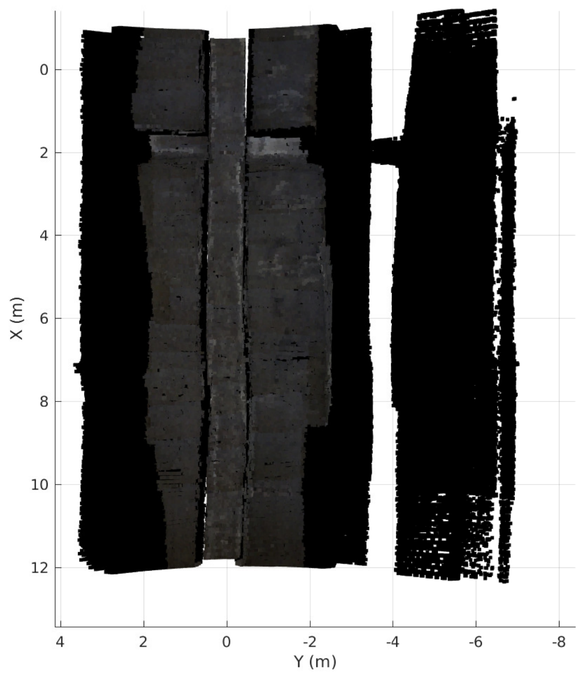
Input	**Description**
img	original image matrix
$mask$	binary image mask
i	image number
I	set of all image, mask, index groups, $(img, mask, i)$
T^i_{MC}	transformation at time image i was captured, from camera frame to map frame
O^i_C	camera optical center in camera frame coordinates for image i
K, s	camera intrinsic matrix and scale coefficient (camera focal length in meters)
M_f	n by 6 matrix of map points, $(x, y, z, dist, colour, defect)$
α	distance threshold for determining contact
β	maximum number of iterations per pixel

```
1: for $point \in M_f$ do
2:     $point[dist] \leftarrow \infty$              ▷ initialize closest point distances to infinity
3:     $point[colour] \leftarrow (0, 0, 0)$         ▷ initialize colour of points to black
4:     $point[defect] \leftarrow (0)$               ▷ initialize defect presence to no defect
5: end for
```

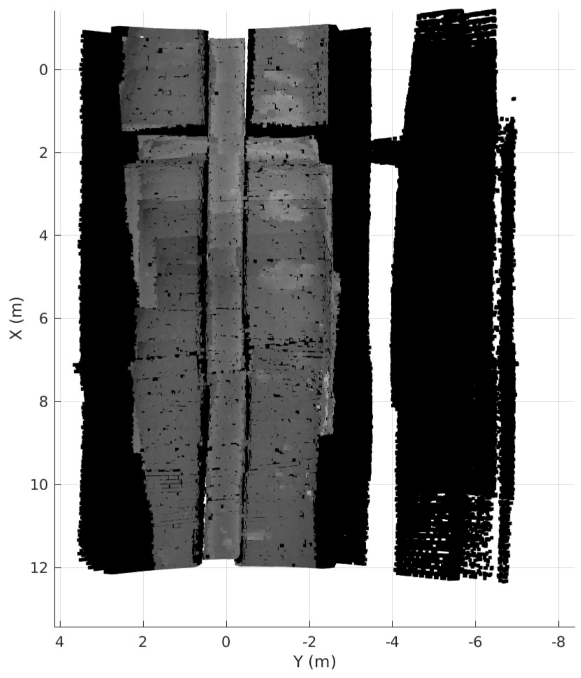
Algorithm 2 Raytrace Map Labeling - Functionality

See Algorithm 1 for input descriptions and setup

```
 1: for $(img, mask, i) \in I$ do
 2:     $O^i_M \leftarrow T^i_{MC} * O^i_C$    ▷ convert camera optical center in $F_c$ to $F_M$ for image $i$
 3:     for $u \leftarrow 1$ to width($img$) do
 4:         for $v \leftarrow 1$ to height($img$) do
 5:             $P^{u,v}_M \leftarrow T^i_{MC} * K^{-1} * s\begin{bmatrix}u\ v\ 1\end{bmatrix}$    ▷ convert pixel point from $F_c$ to $F_M$
 6:             $\phi_r \leftarrow P^{u,v}_M - O^i_M$    ▷ calculate ray direction unit vector
 7:             $P^E_M \leftarrow P^{u,v}_M$    ▷ initialize the ray endpoint in the map frame
 8:             while $true$ do
 9:                 $point \leftarrow \arg\min(\text{distance}(P^E_M, M_f))$    ▷ calculate the closest point
10:                 $D_c \leftarrow \min(\text{distance}(P^E_M, M_f))$    ▷ calculate the closest point distance
11:                 if $D_c < \alpha$ then
12:                     $cur\_dist \leftarrow |\text{euclidean}(O^i_M, point)|$
13:                     if $cur\_dist < point[dist]$ then
14:                         $point[dist] \leftarrow cur\_dist$    ▷ update closest point distance
15:                         $point[colour] \leftarrow img[u, v]$    ▷ update colour
16:                         $point[defect] \leftarrow mask[u, v]$    ▷ update defect info
17:                     else continue    ▷ continue to next pixel
18:                     end if
19:                 else if $iteration > \beta$ then continue    ▷ continue to next pixel
20:                 end if
21:                 $P^E_M \leftarrow P^E_M + \phi_r * D_c$    ▷ extend ray by distance to closest point
22:             end while
23:         end for
24:     end for
25: end for
```



(a) Cloud colourized with vision camera



(b) Cloud colourized with IR camera

Figure 4.9: 3D point clouds showing bottom up view of a bridge girder and soffit.

4.5 Defect Quantification

Given a 3D point cloud map labeled with semantic defect information, it is possible to automate the size quantification of the defects in the point cloud. This size estimation can be any metric, but the metric of interest for spalls and delaminations considered is the defect area. The defect area quantification process presented follows a five-step process outlined in Fig. 4.10.

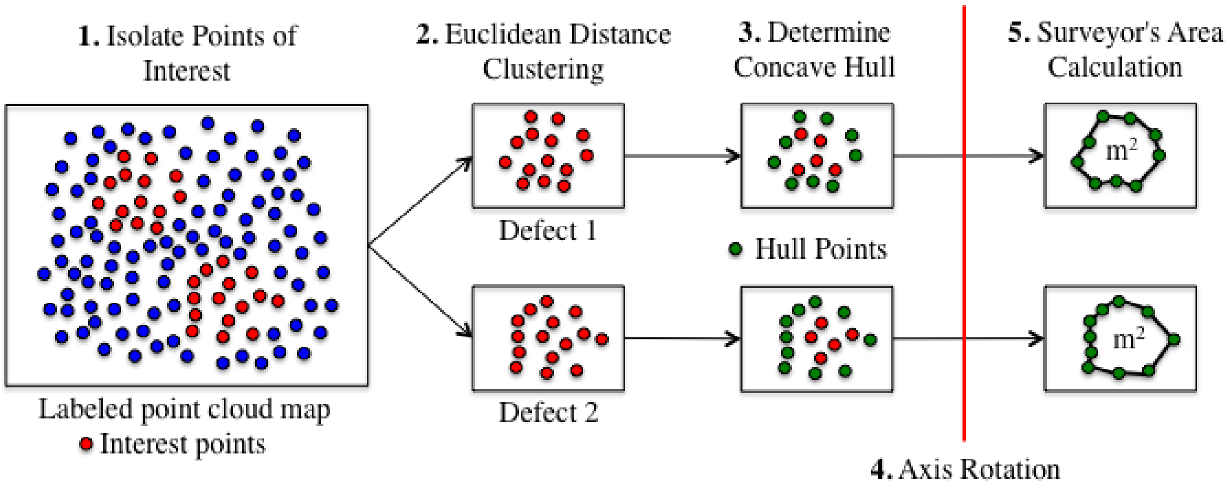
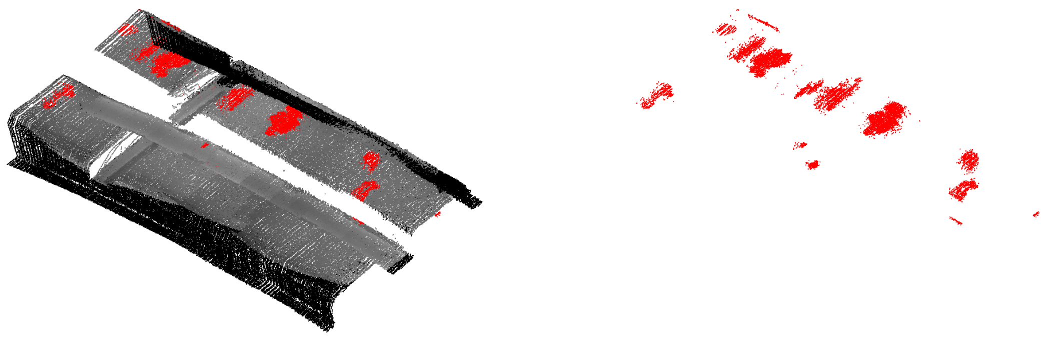

1. Isolate Points of Interest
2. Euclidean Distance Clustering
3. Determine Concave Hull
5. Surveyor's Area Calculation
Defect 1
● Hull Points
m²
Defect 2
m²
Labeled point cloud map
● Interest points
4. Axis Rotation

Figure 4.10: Five-step process for quantifying individual defect areas.

The first step involves identifying and extracting points within the point cloud that are labeled representing defects of interest. This is done by creating a new point cloud with only points representing defect geometry. For example, if delamination is the defect of interest then the new point cloud will be only points labeled as delamination (Fig. 4.11). The second step segments the remaining points representing defects into individual point clouds. That is, each defect is associated with its own distinct sub-point cloud to represent its geometry. This segmentation process is done using an euclidean cluster extraction algorithm presented in [49]. Fig. 4.12a shows an example of a segmented delamination point cloud.

The third step estimates the defect boundary by calculating the 2D concave hull bordering the defect. To do this, the sub-cloud is first filtered to remove noise using random sampling consensus (RANSAC) [17]. RANSAC removes noise by randomly selecting 3 points from the data, fitting a 3D plane to the 3 points, and counting the number of data points within a user-specified threshold distance from the plane. The plane that has the



(a) Semantically labeled point cloud (b) Delamination-only extract point cloud

Figure 4.11: Example of extracting defect information from semantically labeled point cloud.

most inliers is considered the plane that best fits the defect area, and all outlier points for that plane are removed. Once outliers are removed, a 2D concave hull can then be extracted using the inlier points. For this work, the concave hull is extracted using the Point Cloud Library implementation [50]. Fig. 4.12b shows an example of a concave hull for a delamination point cloud.

Step four represents the generated concave hull in a new coordinate frame so that the main object geometry (i.e. the defect area to be calculated) is in the X, Y coordinate plane. This is done by transforming the coordinate system using the properties of the dot and cross products to reconstruct the Z axis such that it is parallel to the normal vector of the plane fit using RANSAC (Algorithm 3).

The final step in the process is calculating the defect area using the transformed 3D defect point cloud. Since the sub-cloud is in a coordinate frame where the normal vector is the same direction as the Z axis, the geometry representing the defect area is represented in the X, Y plane. This enables calculating a tight area of the defect of interest (e.g. delamination or spall) using the surveyor's area formula [5], given by the equation

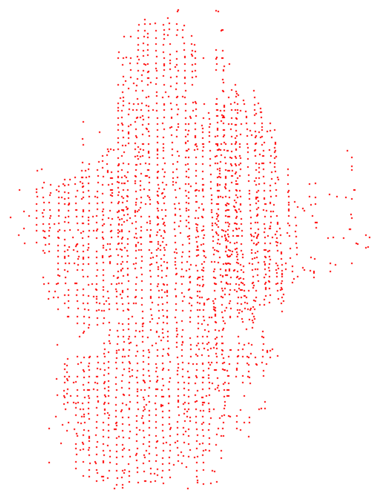
$$Area = \frac{1}{2}\left(\begin{vmatrix} x_0 & x_1 \\ y_0 & y_1 \end{vmatrix} + \begin{vmatrix} x_1 & x_2 \\ y_1 & y_1 \end{vmatrix} + \cdots + \begin{vmatrix} x_{n-2} & x_{n-1} \\ y_{n-2} & y_{n-1} \end{vmatrix} + \begin{vmatrix} x_{n-1} & x_0 \\ y_{n-1} & y_0 \end{vmatrix}\right) \tag{4.8}$$

where each point on the hull is a point (x_n, y_n). The final area calculated will be in the same units as used to define the point cloud. For example, if a point (x, y, n) in the point cloud is defined using meters, then this method will report the area of the point cloud in meters.

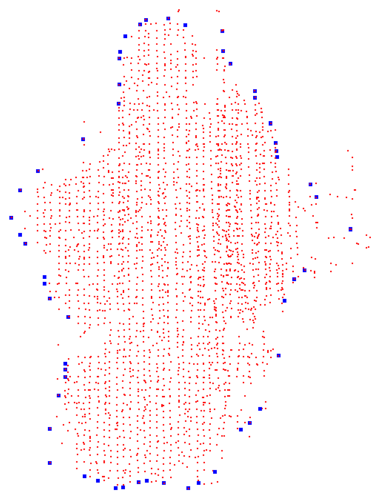
Algorithm 3 Coordinate Frame Transformation

Input	**Description**
$\mathbf{C}$	Matrix of points representing the 3D defect point cloud
$\mathbf{n}$	The normal vector of the plane fit to the 3D defect point cloud, $\mathbf{C}$

```
1: $\mathbf{x}_0 \leftarrow [1, 0, 0]$                                      ▷ Initialize the original $X$ axis
2: $\mathbf{y}_{new} \leftarrow \frac{\mathbf{x}_0 \times \mathbf{n}}{||\mathbf{x}_0 \times \mathbf{n}||}$          ▷ Calculate and normalize new $Y$ axis
3: $\mathbf{x}_{new} \leftarrow \frac{\mathbf{n} \times \mathbf{y}_{new}}{||\mathbf{n} \times \mathbf{y}_{new}||}$      ▷ Calculate and normalize new $Y$ axis
4: $\mathbf{z}_{new} \leftarrow \frac{\mathbf{n}}{||\mathbf{n}||}$                  ▷ Calculate and normalize new $Z$ axis
5:
6: for $\mathbf{p} \in \mathbf{C}$ do
7:     $\mathbf{p}_t \leftarrow \mathbf{p}$                                  ▷ Create temporary point
8:     $\mathbf{p}[x] \leftarrow \mathbf{p}_t \cdot \mathbf{x}_{new}$                       ▷ Update $X$ component
9:     $\mathbf{p}[y] \leftarrow \mathbf{p}_t \cdot \mathbf{y}_{new}$                       ▷ Update $Y$ component
10:    $\mathbf{p}[z] \leftarrow \mathbf{p}_t \cdot \mathbf{z}_{new}$                       ▷ Update $Z$ component
11: end for
12:
13: return $\mathbf{C}$                        ▷ Return transformed 3D defect point cloud
```



(a) Point cloud delamination geometry



(b) Blue points represent 2D boundary hull

Figure 4.12: Example of segmented defect and its 2D boundary hull.

Chapter 5

Results

No public data sets exist to validate the proposed inspection pipeline. Therefore, an entire inspection data set is collected for testing the effectiveness of the automated defect labeling, and the entire automated inspection process as a whole. The defects of concern for testing are area defects commonly found in concrete bridges, namely spalls and delaminations. Other defects such as cracks can also be addressed using the same approach, but due to the nature of defects in the available dataset, the methodology is limited to spall and delamination defects only.

5.1 Data Sets

The data set used for proof of the presented inspection methodology concept as a whole was collected on April 12, 2018 at the Northfield Drive bridge over the Conestogo River (Woolwich, Ontario, Canada). This is a beam/girder reinforced concrete highway bridge that is partially over water and partially over land (Fig. 5.1). It was constructed in 1960 and supports two lanes of traffic in a rural area. The bridge is in relatively poor condition due to its age, and contains significant sections of spalling and delamintion on the concrete girders and soffit. The data was collected on a sunny afternoon after a relatively cold and wet morning, making for good conditions for detecting delaminations with an infrared camera (due to large temperature variations in a short period of time).

The data set was collected by driving the mobile platform underneath the the bridge in a straight path parallel to one of the above bridge girders to collect lidar and image data. This data collection took approximately 60 seconds and resulted in a 15.6m long portion of the concrete girder and surrounding soffit being mapped for inspection purposes. The data set contains no information about other parts of the bridge (e.g. piers or top of deck). The data set contained sufficent lidar data for dense 3D mapping of the girder and soffit. In addition, a total of 60 infrared spectrum images and 60 visual spectrum images were captured of the girder and soffit area for colourization and defect detection. These images contained primarily delamination and spall defects, respectively, as they are the dominant defects present on the bridge girders and soffit. The presence of delaminations and spalls in these images was verified by comparing the defects in the images to a 2018 inspection of the same bridge.

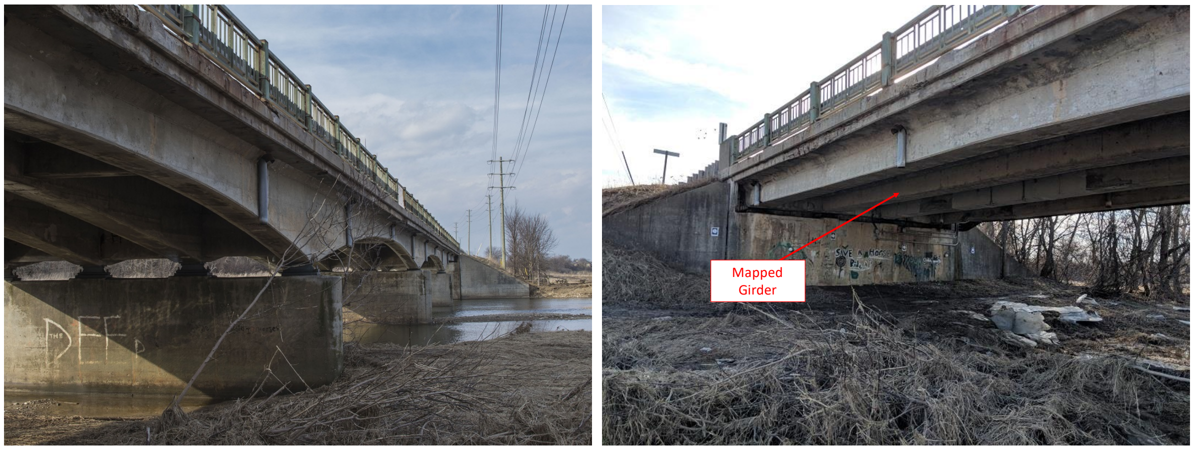


Figure 5.1: Northfield Drive bridge over the Conestogo River. (Photos by: Nicholas Charron)

5.1.1 Defect Labeling Data sets

The data sets for defect labeling are broken into two subsets —a training set and a validation set. The training set consists of images used to train the weights of the CNN, and the validation set consists of different images that are used to validate that the network can make accurate predictions on images not seen during training. CNNs are usually implemented with a random organization of training/validation/test data set split (e.g. 60% train / 20%validation / 20% test). However, in this application the data sets are specifically set up so that the training data is a collection of bridge images and the validation data is a collection of images of a specific bridge (i.e. the Northfield drive bridge), not included in the training data set. No test set is included, as the amount of images is limited, and the focus is not on testing the accuracy capabilities of the network but rather on proving the network can be used for spalling and delamination labeling. The split of training and validation data sets is specifically organized to show that the network can be training on images of random bridges, and then adequately generalize to a new bridge.

For training the CNN to label delaminations, 436 infrared images were captured of four other beam/girder reinforced concrete bridges. All images were labeled using the aforementioned manual labeling tool for use as the training set. These images were captured in variable conditions, at varying times of day, at different times in the year, and under different weather conditions to ensure a variety of appearances of the delaminations. The 60 infrared spectrum images captured by the mobile platform during the girder scan are used as the validation set for CNN. The manual pixel-wise delamination labels of these 60 images are verified by a ground truth delamination inspection report performed by professional inspectors (independently), and made available to the research lab by the owner. The ground-truth image labels are used here to generate a target value to achieve based on inspection results (in terms of defects) and it is assumed that the inspectors measured and reported the results accurately. Furthermore, estimating the true values for delamination under field conditions is extremely difficult and hence the statements regarding errors and what is meant by ground-truth must be interpreted in this context. The images in both the training and validation datasets were collected using a FLIR VUE Pro and Flir ADK cameras with a spectral bands in the range of of 8 - 15 μm ($8000 - 15000$ nm, as outlined as the wavelength of interest for infrared cameras).

For training the CNN to label spalls, 540 visual spectrum images were captured from several beam/girder reinforced concrete bridges in the regions of Waterloo and Toronto, Ontario. Some images in this training set were obtained by professional inspectors during routine visual inspections, while others were obtained by the members of the SDIC research lab. This dataset consists of images captured in variable conditions, and at different times of the year to ensure a variety of spall conditions. All images were labeled using the aforementioned manual labeling tool for use as the training set. The 60 visual spectrum images captured by the mobile platform during the girder scan described previously are used as the validation set for the CNN. The pixel-wise spall labels of these 60 images are verified by a ground truth spall inspection report performed by professional inspectors (independently), and made available to the authors by the owner. The images in the spall training dataset were taken using various cameras, including phone cameras, and the Flir cameras on-board the described platform.

CNNs for semantic pixel-wise segmentation are often evaluated based on mean intersection over union (mIoU), given by the equation

$$mIoU = \frac{TP}{TP + FP + FN} \tag{5.1}$$

where TP is true positives, FP is false positives, and FN is false negatives. To the knowledge of the authors, no standardized data set or previous studies exist to compare mIoU or other CNN accuracy metrics to automatically label spalls or delaminations. Therefore, an alternative method is proposed, where the defect labeling method is validated by comparing its mIoU accuracy metric achieved when implementing the network on a delamination and spall data set compared to its performance implementing it on a benchmark data set. The presented network methodology has been tested on the Pascal VOC 2012 [16] benchmark data set for pixel-wise object segmentation, and achieved a mIoU score of 77.3%. Therefore, for this application the CNN is considered validated if it is able to achieve approximately 77.3% mIoU accuracy for pixel-wise segmentation of delaminations and spalls.

5.1.2 Defect Quantification Dataset

For validating the automated defect quantification process as a whole, all of the collected data was used to generate a 3D point cloud map of the scanned 15.6m length of the concrete bridge girder and soffit. The lidar data was used with SLAM to generate the unlabelled 3D map. The final map contains semantic labels from using the map labeling method to fuse information from the pixel-wise defect labeling step (i.e. using either ground truth or CNN image labels) to the map. A labeled 3D map enables the use of the presented area calculation procedure for automated quantification of defect sizes.

To assess the accuracy of the automated defect quantification, the quantification methodology is performed twice on the generated map. One iteration uses the ground truth image labels (i.e. manual defect labels using the MATLAB labeling GUI) and the other uses the CNN image labels (i.e. CNN predicted defect labels). In this work, the ground truth image labels are considered to yield ground truth defect areas with this quantification process. This consideration is justified by: (1) the true defect areas can never be known due to subjectivity and limitations with measurements, (2) an analysis of accuracy of the 3D point cloud shows approximately 1cm point location error at the girder of interest [9], which is negligible compared to defect sizes, and (3) a comparison performed between the delamination and spall areas from a 2018 inspection report of the bridge performed by professional inspectors and the areas from ground truth image labels. The inspection comparison showed that for both inspections the defects were in the same location on the bridge girder/soffit, and that area estimates were within the same order of magnitude. Further detail in the comparison is not possible, as the 2018 inspection report only reports coarse bounding-box areas for both delaminations and spalls due to the time and accessibility limitations of human inspection. Therefore, the areas obtained from fusing ground truth image labels with the 3D map are the best possible estimate of ground truth areas for validating the defect areas from fusing CNN image labels with the 3D map.

5.2 Defect Labeling

In this thesis the target of the defect labeling step is: 1) to show that the network can be trained on images of random bridges and generalize to a bridge not seen in the training data, and 2) to achieve similar object segmentation mIoU accuracy as when the network is trained and evaluated on the Pascal VOC 2012 [16] benchmark data set, but with delaminations and spalls as the object classes to be segmented. These targets are chosen as the network has already proven success on benchmark data sets, so in this thesis the focus is on application of the network towards defect inspection.

Table 5.1 summarizes the training and validation results for pixel-wise labeling of delaminations. The comparison shows that the delamination object segmentation results slightly exceed the results achieved when implementing the network on the Pascal VOC 2012 object segmentation data set. This is likely because of a few reasons. First, all images in the delamination training and validation data sets are obtained using infrared cameras with similar characteristics (FLIR VUE Pro and Flir ADK). This causes there to be little variability in mechanical characteristics that affect how an image captures scene data, and no variability in the spatial density that the images are stored. This yields less variability between all images in the data set than normally seen in a benchmark data set. Second, infrared images contain less potentially adversarial features compared to visual images. Infrared images of concrete bridges generally only contain mute concrete background with some hot/cold areas due objects such as pipes, or delaminations. The objects that are present tend to not resemble a delamination patch, making it easier for the algorithm to distinguish what hot/cold areas should be labeled as a delamination.

Fig. 5.2 displays the results of the network on four of the images in the validation set. These examples show that the network performs well at finding delaminated areas, and the error is largely confined to the finer details around the boundaries of delaminations. This boundary error is likely due to the shallowness/simplicty of the network used. A more deep/complex CNN may be able to improve on these boundary errors by maintaining more information from earlier layers at the cost of more computation (e.g. the tradeoff of using FCN-8s versus FCN-32s). However, this is likely not necessary for defect area calculations as 1) it is difficult to adequately label boundaries, so some errors may be from human label error, and 2) conventional inspection techniques being used today (e.g. hammer sounding, chain dragging) tend to report even coarser details (e.g. only bounding box areas). In terms of the ultimate application of defect assessment the delamination predictions by the CNN are considered adequate.

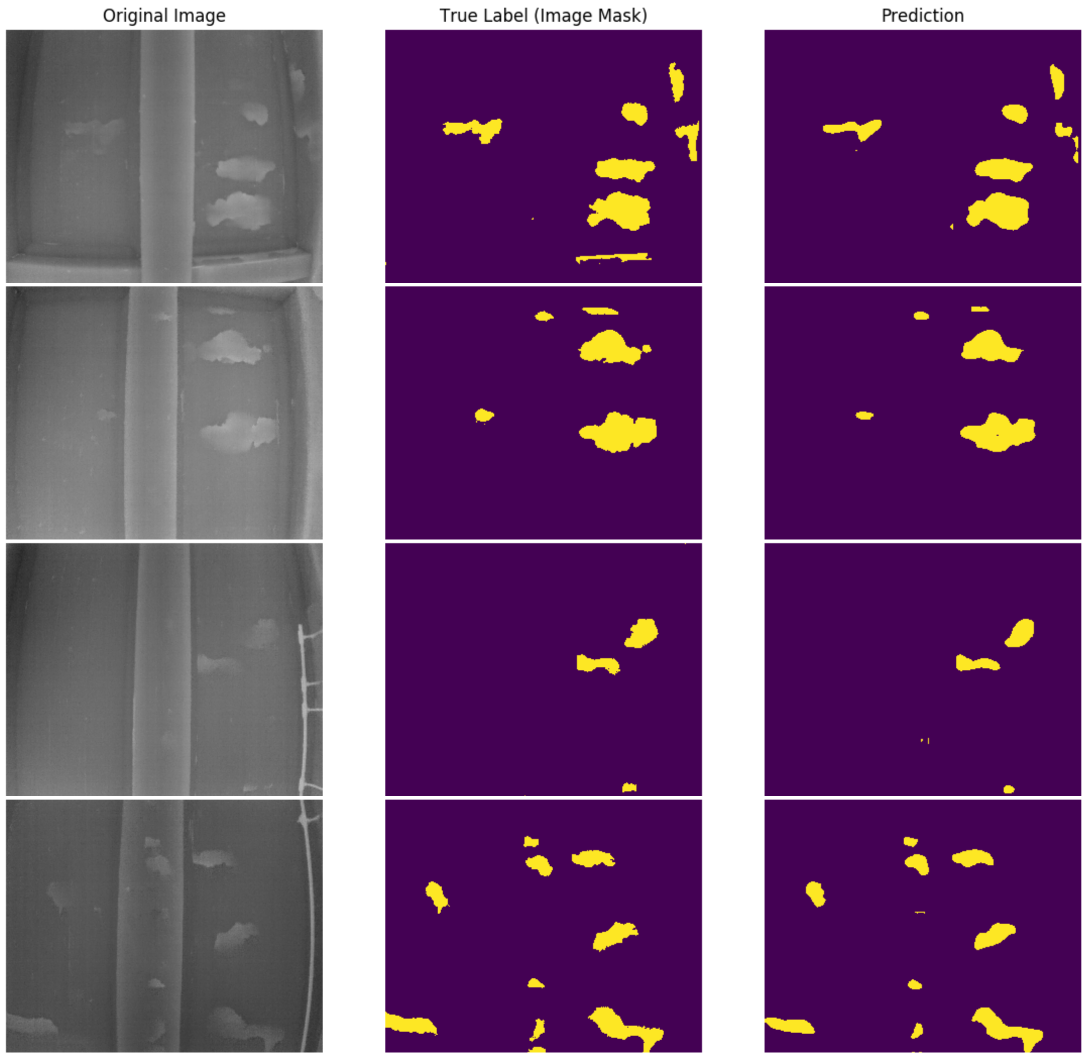


Figure 5.2: Infrared delamination prediction results. All rows show images with a large amount of delaminations. In general, the network does well to distinguish what is a delamination, but struggles with fine details and small delaminations.

Table 5.1 also shows the results of training and validation on the spall data set compared to the benchmark performance achieved by the network on the Pascal VOC 2012 data set. The mIoU shows that the spall network achieves comparable results on the validation set when compared to the performance of the same network on benchmark data sets, but the training set achieved better results than the validation. This indicates there is some overfitting to the spall training data set, despite using the same setup as for the delamination training. This is evidence that the data set is causing overfitting for a few reasons. First, the spalls data set did not consist of images only of bridges. Many of the images in the training dataset were of spalls on other concrete structures. Second, spalls have large range of appearances. The spall can have smooth or jagged edges, have exposed rebar or no exposed rebar, be under low or high lighting conditions, etc. This variability of appearances makes it more difficult for the network to learn the characteristics of a spall. This problem detecting true positives compounds into more false positives, as there is a large set of objects that may look similar to a spall (e.g. dirty patch on concrete). To alleviate these accuracy limitations data set should be larger and more diverse to cover all the varieties of spalls that may be observed by the network.

Fig. 5.3 displays the results of the network on four of the images in the validation set. Similar to the delaminations, the errors in these images are confined to the fine details at the boundaries of the spalls, with only a few small areas of spalls being erroneously classified entirely. As is the same case with delamination inspection, the fine details are often ignored during routine spall inspections, so this still presents a significant improvement over conventional visual approaches being used today. Therefore, for the ultimate application of defect assessment the spall predictions by the CNN are considered adequate. If higher accuracy is required for a specific application, it is always possible to implement a deeper network in the future.

It is worth noting that there exist CNNs for pixel-wise labeling of structural defects that report similar or better mIoU (80-85%) than reported here for delaminations and spalls [1, 34]. However, the reasons for such differences can be attributed to a number of factors such as difference in the network complexity, difference in data sets, or difference in the complexity of defect appearance. For example, in the case of [34], the data set consists of images where the defect is central and large in the image. This creates easier conditions for labeling than if the defect images are captured by a mobile platform. For these reasons, making direct comparisons between different implementations and defects could be misleading if using only accuracy metrics. Additionally, the accuracy of the network should be viewed in terms of the application requirements for which the results above present significant improvement over current practice.

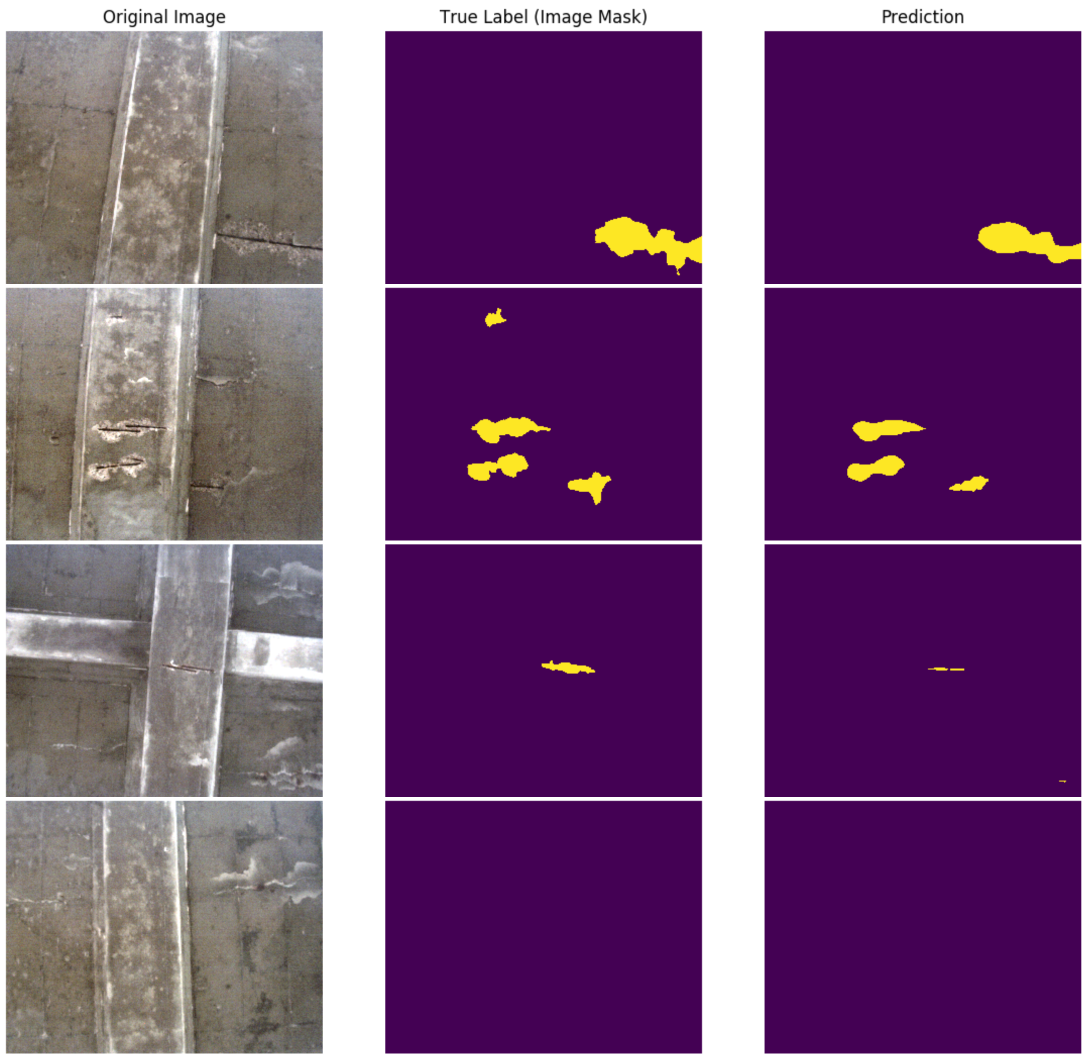


Figure 5.3: Visual spall prediction results. The first two rows show examples of clear and large spalls. The third row shows an example of a thin/small spall area. The fourth row shows an example where there is significant potential for false positives. In general, the network does well to distinguish what is a spall, but struggles with fine details and small spalls.

Table 5.1: Network evaluation metric results.

Dataset	Split	CCE	mIoU
Delamination	Train	0.013	83.7%
	Val	0.056	82.7%
Spall	Train	0.018	81.4%
	Val	0.019	71.4%
Pascal VOC 2012 [16]	Benchmark	n/a	77.3% [51]

5.3 Defect Quantification

One of the main contributions of this thesis is the ability to provide quantitative information about defects, which requires all steps presented in the methodology. However, a major limitation in the quantification information is the quality of image labeling results. To date, there has been little effort directed towards studying how errors from using an automated image labeling method (e.g. CNN) transfer to errors in calculating quantitative defect information. To this end, this thesis presents a comparison of the quantification information provided using the presented methodology for assessing delaminations and spalls when using ground truth labels (i.e. manual lables with MATLAB GUI) versus CNN predicted labels at the image labeling step. The target of this approach is to show that errors at the image labeling step are consistent with errors in defect quantification. This will show that the automated inspection process provides accurate results that are limited only by the accuracy of the defect image labels.

To compare the results of ground truth labels and CNN labels, the quantification methodology is performed twice on the created validation data set for each defect; once using the 60 ground truth labeled images for map labeling, and once using the 60 CNN labeled images for map labeling (these 60 images are from the robot scan of the bridge girder and soffit, and are the validation set for the automated spall and delamination image labeling), while all other variables remained constant. This results in a consistent 3D map (i.e. same point locations and colors), but different defect labels. The inspection process is run for four iterations (two for each defect).

The 3D point cloud and the area extraction method parameters are held constant at euclidean clustering threshold = 7.5cm and RANSAC outlier threshold = 1cm across all iterations. The clustering threshold of 7.5cm in this application is user-defined and problem-specific. With this threshold, 3D points closer than 7.5cm from each other are considered to represent the same defect. This threshold provided a good balance between not falsely excluding points from clusters, while still sufficiently separating the defects for these data sets. The input of 7.5cm can be adjusted (and should be adjusted) based on specific inspection needs and repair procedures for the case under study. For example, it may need to be set smaller (e.g. 5cm) for crack defects. The RANSAC outlier threshold of 1cm is based on the accuracy of the sensors, which is +/- 1cm for a 3D point. Therefore, anything further than 1cm away is considered an outlier.

Fig. 5.4 shows the difference between quantifying delaminations using ground truth labels and using CNN labels for the defect labeling step. The figure plots the intersection points (points labeled delamination by both manual and automated labeling) on top of the union points (points labaled delamination by manual or automated labeling). From this, it is possible to qualitatively see that the delamination locations are nearly identical in both iterations. Table 5.2 quantitatively outlines the comparison between the delaminations obtained in the two cases. These comparisons are based on automated matching, and only delamination areas with greater than 50 points representing its geometry are considered. Empirical testing showed clusters of less than 50 points do not contain enough information to be assessed. This limit could be reduced if the point cloud has a higher point density. Recall the mIoU of the infrared images in the validation set is 82.7% (error = 17.3%). Many of the calculated defects reported similar error rates between ground truth and CNN label area calculations. The overall defect area error of 13.6% is similar to the overall mIoU error, with the minor difference likely due to discrepancy between how error is calculated for pixels versus for an area metric. Minor differences in the error may also be caused by the point cloud causing down sampling of the dense image data.

The main outlier in area calculations is delamination 4, with a percentage error of 50.6%. On closer observation, this error is primarily due to the defect segmentation step. In this case, the points representing delamination 4 in the 3D map labeled with CNN predictions are sparse. As a result, the segmentation threshold of 7.5cm caused a significant number of points to not be included in the geometry of delamination 4 for the CNN inspection. This is the danger of using a hard threshold for segmentation, as the selected value may not be optimal for all cases. However, better defect labeling or a more dense point cloud would make the results of the segmentation step more consistent. The remaining defects associated with relatively large errors are small defects. This is not surprising, however, as the absolute errors of these smaller defects are still small, and the CNN label information still provides valuable information. The error rate of the smaller defects could likely be improved by increasing the spatial density of the point cloud, but using a deeper/more accurate CNN would be the best option for improving these small defects. Recall that the network implemented is good at general localization of the defect but struggles with the fine labeling around borders. When defects are small, the fine labeling becomes far more important. This presents a decision for the inspector, as it may be okay that smaller delaminations are not as accurately depicted during routine inspection, but may be critical for a in-depth inspection.

Table 5.2: Delamination size results (sizes in cm^2).

Label ID	**Ground Truth Size**	**CNN Size**	**error**	**% error**
1	7877	6986	891	11.3%
2	6922	6257	665	9.6%
3	4681	4082	599	12.8%
4	2013	1836	177	8.8%
5	1452	717	735	50.6%
6	2203	1898	305	13.8%
7	1887	1857	30	1.6%
8	2154	1676	478	22.2%
9	1329	1356	-26	2.0%
10	737	753	-15	2.1%
11	1374	1077	297	21.6%
12	921	957	-37	4.0%
13	724	404	320	44.2%
14	622	503	119	19.1%
15	709	302	407	57.5%
16	200	243	-43	21.7%
17	318	320	-2	0.8%
Total	**36122**	**31224**	**4898**	**13.6%**

Fig. 5.5 shows the difference between spall areas using ground truth labels and CNN labels at the image labeling step. Similar to delaminations, the intersection spall labels are plotted on top of the union spall labels. The figure shows qualitatively that the locations of the spalls are nearly identical for both labeling methods. The only notable error appears at defect 5, where the CNN labeled data appears much sparser than the manual labeled data.

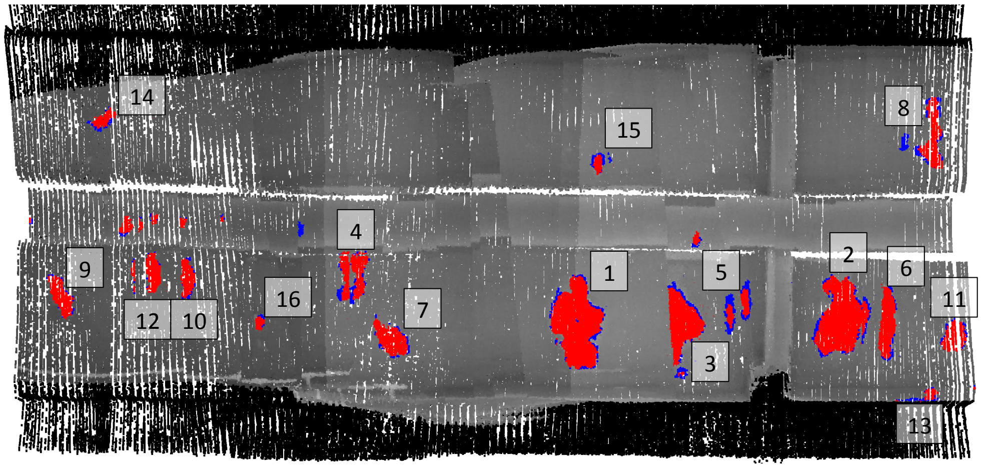


Figure 5.4: Extracted delaminations labeled on the 3D point cloud representation of the 15.6m long section of the bridge girder (horizontal across the middle of the image) and soffit scanned by the robot platform. Red points represent points labeled as delamination in both ground truth and CNN labeling. Blue points represent points represent points labeled in only one of the methods.

The quantitative results of the automated defect quantification algorithm for assessing spalling are reported in Table 5.3. Only spalls with geometry representation of greater than 50 points are listed in the table. Recall the mIoU of the infrared images in the validation set is 71.4% (error = 28.6%). Similar to the delamination results, most spalls have an error comparable to the mIoU error of the validation set of the CNN. However, the overall error of this dataset is large, primarily because of spall 5. This error is contributed by an insufficient number of points (less than 50) in the CNN label inspection iteration to represent the geometry of spall 5. This could be seen as a limitation of setting a minimum number of points threshold for which cluster are considered spalls. However, even lowering the cluster threshold would likely result in significant error. Therefore, the best solution to alleviate errors of this nature is still a more dense 3D map or a deeper CNN. This is again an inspector decision, as spall 5 is small enough that it could be seen as low importance. Additionally, in Fig. 5.5 it is still evident that there are some points labeled as spall in this area from the CNN, so a review of the visual data still enables an inspector to indicate an abnormality. An inspector could therefore rerun the inspection after manual fine-tuning the image label of that spall, or mark that a small spall exists that cannot be accurately measured at this time but should be checked in subsequent inspections. If spall 5 is removed for error calculations, the overall area error is 25.9%, which is again comparable to the validation set error.

Table 5.3: Spall size results (sizes in cm^2).

Label ID	Ground Truth Size	CNN Size	error	% error
1	3277	2573	705	21.5%
2	531	473	58	10.9%
3	533	387	145	27.3%
4	531	178	353	66.5%
5	394	0	394	100%
Total	**5266**	**3611**	**1655**	**34.0%**
Total-5	**4872**	**3611**	**1261**	**25.9%**

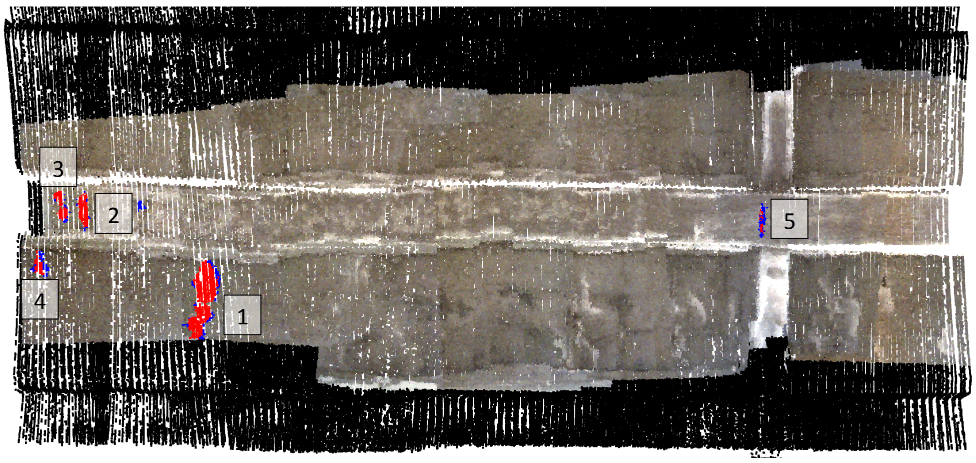


Figure 5.5: Extracted spalls labeled on the 3D point cloud representation of the 15.6m long section of the of the bridge girder (horizontal across the middle of the image) and soffit scanned by the robot platform. Red points represent points labeled as spalling in both hand labeling and CNN labeling. Blue points represent points represent points labeled in only one of the methods.

The results of the two analyses show that the mIoU results of the CNN image labeling method transfers to similar quantification accuracy of delamination and spall areas when using the presented methodology. Compared to current routine inspection methods, the accuracy of the quantitative inspection results could be considered adequate. However, this conclusion should not be generalized to all defects, such as cracks, which contain more fine details for quantification (e.g. crack width) than delaminations and spalls. In this case, it is unclear if errors in the CNN will be comparable to quantification errors, as limitations in the map density may cause other significant errors. This could be a subject of a separate research study.

Chapter 6

Conclusions

The work presented in this thesis outlines how state-of-the-art computer vision techniques can be applied to processed camera and lidar data to semi- or fully-automate the area size quantification of delaminations and spalls and remove some of the subjectivity of traditional inspections. This is made possible through a five-step process of:

1. Data collection using an unmanned ground vehicle mounted with well calibrated sensors, specifically lidar and camera sensors.

2. Processing of lidar data with SLAM algorithms to create a spatial representation of the scanned structure via a 3D map.

3. Semi- or fully-automated labeling of defects in the collected images using a manual labeling tool (for best accuracy) or semantic segmentation CNN (for full automation and improved time efficiency).

4. Transfer of image data (pixel colour and CNN label) onto the 3D point cloud using an occlusion-aware ray-tracing technique to augment the pinhole projection method.

5. An automated area calculation algorithm for calculating defect areas in real-world scale given a 3D point cloud enriched with semantic defect information.

The overall methodology is shown to be capable of performing delamination and spall detection and quantification to a level of accuracy that can compete with current inspection practices. Several key insights to success can be learned from developing the overall inspection process:

- The semi-automated defect labeling tool developed in MATLAB is essential for building adequately large labeled data sets. The method increases the pace of labeling, and also provides an initial guess of defect locations based on image properties, which can potentially limit the subjectivity of human labeling.

- It is possible to replicate the accuracy of state-of-the-art CNNs (specifically MobileNetV2 and Deeplab V3 semantic segmentation CNN) for pixel-wise labeling of delaminations and spalls. The main limitation is the ability to develop a large enough data set for adequate training. However, this is evidence that the structural engineering discipline can save time by implementing any state-of-the-art CNN, rather than developing entirely novel neural networks.

- Labeling a 3D point by projecting to 2D and assigning it with the information contained in the corresponding 2D pixel is not adequate because of occlusions. This could cause defects to appear multiple times on different parts of a structure. Alternatively, a ray-tracing method that extends a ray from the camera through the virtual image until contact with the 3D point cloud is able to label 3D points with pixel information in a manner robust to occlusions, which is essential to accurate results.

- Use of a semantically labeled 3D point cloud creates the opportunity for repeatable and non-subjective defect area quantification. Some parameter settings (e.g. euclidean clustering threshold) may cause slightly different results, but these parameters can still be held constant during subsequent inspections to maintain consistency. This enables better tracking of defect growth over time, and presents a significant improvement on traditional practices, where defect area is shown to be subjective based on inspector. The inspector does not need to be entirely removed, as the inspector can also visually review the semantically labeled 3D point cloud to qualitatively verify the results of the automated inspection process.

- The main source of error of the proposed inspection process is in the automated labeling of the images, as error rates in image defect labeling are consistent with error rates of the physical defect areas. Error rates in image defect labeling can always be improved with more cost investment (more computation or manual labour cost), giving the engineer a decision the amount of resources to allocate to image labeling based on the severity/importance of the inspection.

Despite the improvement on the repeatability and accuracy of current defect inspection, the presented inspection process has limitations. First, The presented process is only shown to be accurate for delaminations and spalls and not for other defects such as cracks. However, extending this approach to other defects can follow the same process provided appropriate data sets are available. Second, there is a lack of extensive defect (specifically delamination and spall) data sets available for training and inference, limiting the ability to adequately train neural networks. Pixel-wise CNNs are still relatively new in the field of defect assessment, and the tedious nature of data labeling makes training data limited. Over time this can be resolved through open source data sharing from other researchers and performance comparisons with other networks. From a practical standpoint, this can also be improved by having inspectors manually fine-tune CNN labels, as this still presents an improvement in time-efficiency until a large enough data set is obtained. Finally, the data collection platform used to collect the pertinent data for this thesis is a ground vehicle, which was chosen primarily based on risk and regulatory constraints, and payload restrictions. Therefore no testing has been done on data acquired from other platforms such as drones and hand-held units. However, all of the algorithms developed and applied in this thesis (CNN, map labeling, area extraction), as well as the SLAM algorithm are designed to be platform agnostic. That is, they depend only on the form of the collected data, and can be implemented regardless of the platform employed to collect the data. The main limitation is the expense of mounting lidar and camera sensors to other platforms.

There are success and limitations of the work detailed in this thesis, but overall the automated inspection process presents stepping stone towards bringing state-of-the-art robotics and computer vision into bridge inspection. The current state of bridge inspection is outdated and economically inefficient, as visual inspections are fraught with human errors and are limited by accessibility, and manual inspection labour is expensive. A move towards these technologies presents a significant opportunity to improve accuracy, while also cutting costs of inspection. In addition, reducing the effort of manual inspection frees inspectors and engineers to allocate more of their time towards intelligent deterioration prevention and maintenance techniques. More research should continue to be done to push automated bridge inspections towards being a mainstream practice.

References

[1] Mohamad Alipour, Devin K. Harris, and Gregory R. Miller. Robust Pixel-Level Crack Detection Using Deep Fully Convolutional Neural Networks. *Journal of Computing in Civil Engineering*, 33(6):1–14, 2019.

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1505.07293*, 2015.

[3] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[4] Jean-Yves Bouguet. Camera calibration toolbox for matlab. *Computational Vision at the California Institute of Technology.*, 2015. http://www.vision.caltech.edu/bouguetj/calib_doc/.

[5] Bart Braden. The surveyor's area formula. *The College Mathematics Journal*, 17(4):326–337, 1986.

[6] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[7] Young Jin Cha, Wooram Choi, and Oral Büyüköztürk. Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks. *Computer-Aided Civil and Infrastructure Engineering*, 32(5):361–378, 2017.

[8] Young Jin Cha, Wooram Choi, Gahyun Suh, Sadegh Mahmoudkhani, and Oral Büyüköztürk. Autonomous Structural Visual Inspection Using Region-Based Deep Learning for Detecting Multiple Damage Types. *Computer-Aided Civil and Infrastructure Engineering*, 33(9):731–747, 2018.

[9] Nicholas Charron, Evan McLaughlin, Stephen Phillips, Kevin Goorts, Sriram Narasimhan, and Steven L. Waslander. Automated Bridge Inspection Using Mobile Ground Robotics. *Journal of Structural Engineering (United States)*, 145(11):1–18, 2019.

[10] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017.

[11] Siyuan Chen, Debra F Laefer, Eleni Mangina, SM Iman Zolanvari, and Jonathan Byrne. Uav bridge inspection through evaluated 3d reconstructions. *Journal of Bridge Engineering*, 24(4):05019001, 2019.

[12] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., USA, 1st edition, 2017.

[13] Sattar Dorafshan, Robert J. Thomas, and Marc Maguire. Comparison of deep convolutional neural networks and edge detectors for image-based crack detection in concrete. *Construction and Building Materials*, 186:1031–1045, 2018.

[14] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping (SLAM): part I The Essential Algorithms. *Robotics & Automation Magazine*, 2:99–110, 2006.

[15] Andrew Ellenberg, Ahsas Kontsos, Franklin Moon, and Ivan Bartoli. Bridge deck delamination identification from unmanned aerial vehicle infrared imagery. *Automation in Construction*, 72:155–165, 2016.

[16] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html, 2012.

[17] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[18] Stephanie German, Ioannis Brilakis, and Reginald Desroches. Rapid entropy-based detection and properties measurement of concrete spalling with machine vision for post-earthquake safety assessments. *Advanced Engineering Informatics*, 26(4):846–858, 2012.

[19] Spencer Gibb, Hung Manh La, Tuan Le, Luan Nguyen, Ryan Schmid, and Huy Pham. Nondestructive evaluation sensor fusion with autonomous robotic system for civil infrastructure inspection. *Journal of Field Robotics*, 35(6):988–1004, 2018.

[20] Ross Girshick. Fast R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[22] Benjamin A. Graybeal, Brent M. Phares, Dennis D. Rolander, Mark Moore, and Glenn Washer. Visual inspection of highway bridges. *Journal of Nondestructive Evaluation*, 21(3):67–83, 2002.

[23] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[24] Janne Heikkila and Olli Silven. A four-step camera calibration procedure with implicit image correction. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1106–1112, 1997.

[25] Ali Khaloo, David Lattanzi, Keith Cunningham, Rodney Dell'Andrea, and Mark Riley. Unmanned aerial vehicle inspection of the Placer River Trail Bridge through image-based 3D modelling. *Structure and Infrastructure Engineering*, 14(1):124–136, 2018.

[26] In Ho Kim, Haemin Jeon, Seung Chan Baek, Won-Hwa Hong, and Hyung-Jo Jung. Application of crack identification techniques for an aging concrete bridge inspection using an unmanned aerial vehicle. *Sensors*, 18(6):1881, 2018.

[27] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference for Learning Representations*, 2015.

[28] Alex Krizhevsky, IIya Sulskever, and Geoffret E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information and Processing Systems (NIPS)*, 60(6):84–90, 2012.

[29] Hung M La, Nenad Gucunski, Kristin Dana, and Seong-Hoon Kee. Development of an autonomous bridge deck inspection robotic system. *Journal of Field Robotics*, 34(8):1489–1504, 2017.

[30] Hung Manh La, Tran Hiep Dinh, Nhan Huu Pham, Quang Phuc Ha, and Anh Quyen Pham. Automated robotic monitoring and inspection of steel structures and bridges. *Robotica*, 37(5):947967, 2019.

[31] Debra F. Laefer, Linh Truong-Hong, Hamish Carr, and Manmeet Singh. Crack detection limits in unit based masonry with terrestrial laser scanning. *NDT and E International*, 62:66–76, 2014.

[32] David Lattanzi and Gregory R. Miller. Robust Automated Concrete Damage Detection Algorithms for Field Applications. *Journal of Computing in Civil Engineering*, 28(2):253–262, 2014.

[33] David W. Law, David Silcock, and Lucas Holden. Terrestrial laser scanner assessment of deteriorating concrete structures. *Structural Control and Health Monitoring*, 25(5):1–15, 2018.

[34] Shengyuan Li, Xuefeng Zhao, and Guangyi Zhou. Automatic pixel-level multiple damage detection of concrete structure using fully convolutional network. *Computer-Aided Civil and Infrastructure Engineering*, pages 616–634, 2019.

[35] W. Liu, S. Chen, and E. Hauser. LiDAR-based bridge structure defect detection. *Experimental Techniques*, 35(6):27–34, 2011.

[36] MATLAB. *version 9.2.0 (R2017a)*. The MathWorks Inc., Natick, Massachusetts, 2017.

[37] E McLaughlin, N Charron, and S Narasimhan. Combining deep learning and robotics for automated concrete delamination assessment. In *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*, volume 36, pages 485–492. IAARC Publications, 2019.

[38] Evan Mclaughlin, Nicholas Charron, Sriram Narasimhan, and M Asce. Automated Defect Quantification in Concrete Bridges Using Robotics and Deep Learning. *Journal of Computing in Civil Engineering*, 34(5):1–12, 2020.

[39] Ministry of Transportation Ontario, St. Catharines, ON. *Ontario Structural Inspection Manual (OSIM)*, 2008.

[40] Fu Tao Ni, Jian Zhang, and Zhi Qiang Chen. Pixel-level crack delineation in images with convolutional feature fusion. *Structural Control and Health Monitoring*, 26(1):1–18, 2019.

[41] Je Keun Oh, Giho Jang, Semin Oh, Jeong Ho Lee, Byung Ju Yi, Young Shik Moon, Jong Seh Lee, and Youngjin Choi. Bridge inspection robot system with machine vision. *Automation in Construction*, 18(7):929–941, 2009.

[42] Tarek Omar, Moncef L. Nehdi, and Tarek Zayed. Infrared thermography model for automated detection of delamination in RC bridge decks. *Construction and Building Materials*, 168(April):313–327, 2018.

[43] H Peel, S Luo, AG Cohn, and R Fuentes. Localisation of a mobile robot for bridge bearing inspection. *Automation in Construction*, 94:244–256, 2018.

[44] Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel mattersimprove semantic segmentation by global convolutional network. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition*, pages 1743–1751. IEEE, 2017.

[45] Stephen Phillips and Sriram Narasimhan. Automating data collection for robotic bridge inspections. *Journal of Bridge Engineering*, 24(8):04019075, 2019.

[46] Prateek Prasanna, Kristin Dana, Nenad Gucunski, and Basily Basily. Computer-vision based crack detection and analysis. In *SPIE Smart Structures and Materials + Nondestructive Evaluation and Health Monitoring*, volume 8345, 2012.

[47] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[48] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39, 06 2015.

[49] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments.* PhD thesis, Computer Science department, Technische Universitaet Muenchen, Germany, October 2009.

[50] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). *Proceedings - IEEE International Conference on Robotics and Automation*, pages 1 – 4, 2011.

[51] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

[52] Evan Shelhamer, Jonathan Long, Trevor Darrell, Evan Shelhamer, Trevor Darrell, Jonathan Long, Trevor Darrell, Evan Shelhamer, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[53] Roland Siegwart and Illah R Nourbakhsh. *Introduction to Autonomous Mobile Robots*. The MIT Press, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2004.

[54] Billie F. Spencer, Vedhus Hoskere, and Yasutaka Narazaki. Advances in Computer Vision-Based Civil Infrastructure Inspection and Monitoring. *Engineering*, 5(2):199–222, 2019.

[55] Peter Sturm. Pinhole Camera Model. In *Computer Vision*. Springer US, 2014.

[56] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[57] Yelda Turkan, Simon Laflamme, and Liangyu Tan. Terrestrial Laser Scanning-Based Bridge Structural Condition Assessment. *InTrans Project Reports*, 199, 2016.

[58] J. Valença, I. Puente, E. Júlio, H. González-Jorge, and P. Arias-Sánchez. Assessment of cracks on concrete bridges using image processing supported by laser scanning survey. *Construction and Building Materials*, 146:668–678, 2017.

[59] Xincong Yang, Heng Li, Yantao Yu, Xiaochun Luo, Ting Huang, and Xu Yang. Automatic Pixel-Level Crack Detection and Measurement Using Fully Convolutional Network. *Computer-Aided Civil and Infrastructure Engineering*, 33(12):1090–1109, 2018.

[60] Allen Zhang, Kelvin CP Wang, Baoxian Li, Enhui Yang, Xianxing Dai, Yi Peng, Yue Fei, Yang Liu, Joshua Q Li, and Cheng Chen. Automated pixel-level pavement crack detection on 3d asphalt surfaces using a deep-learning network. *Computer-Aided Civil and Infrastructure Engineering*, 32(10):805–819, 2017.

[61] Zhengyou Zhang. A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2002.

APPENDICES

Appendix A

Further Algorithm Details

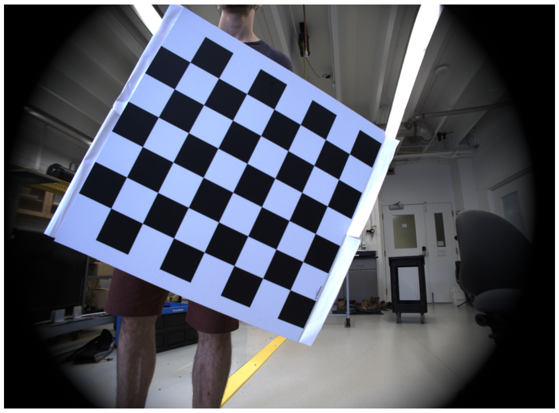
A.1 Checkerboard Single Camera Calibration

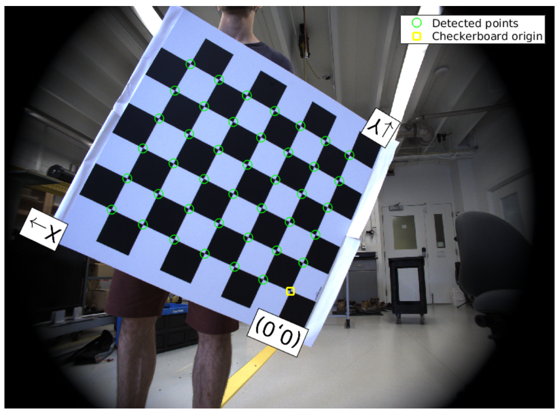
A.1.1 Process Overview

An overview of the process of checkerboard camera calibration using [4] can motivate the derivation of the mathematics. To calibrate a camera with a checkerboard, the user must provide images of the checkerboard target at numerous angles and skews (e.g. Fig. A.1a). In each image, a corner detection technique can be used (e.g. Harris corner detection [23]) to determine the actual (u, v) location of all the corners on the checkerboard (Fig. A.1b). This is a relatively easy knowledge-driven computer vision task, as the checkerboard has distinct features to distinguish it from background noise.

The next step is to set the coordinate frame of the checkerboard in world coordinates. This is done by setting one of the outside corner points as $(0, 0, 0)$ and then assuming the checkerboard is flat along the Z axis such that $Z = 0$ for all points. The real-world distance of the squares and the number of squares is an input to the calibration, and therefore the coordinates of all corners can be calculated.

This setup outlines the inputs to the the checkerboard camera calibration process, which are the (u, v) coordinates of the checkerboard corners in all calibration images and the (X, Y, Z) world coordinates of the checkerboard corners (which are defined to be the same for every image). The goal of the mathematics is therefore to solve for the rotation, $\mathbf{R}$, and translation ,$\mathbf{t}$, that describes the location of the camera relative to the checkerboard in each image, and the intrinsic matrix, $\mathbf{K}$, that minimizes the error between the (u, v)


(a) Checkerboard calibration image



(b) Detected checkerboard corners

Figure A.1: Example of a typical image used during checkerboard camera calibration.

corners detected using computer vision and the (u, v) coordinates of the corresponding 3D (X, Y, Z) points projected to the 2D image plane.

The output of the camera calibration provides the calculated locations of the checkerboard relative to the image via $\mathbf{R}, \mathbf{t}$, as shown in Fig. A.2a. The main metric for evaluating the calibration is the reprojection error, which is the pixel distance between the the computer vision detected corners and the 3D projected corners. This can be shown for each image to help determine which images best contribute to the calibration (Fig. A.2b). In general, the lower the reprojection error, the more accurate the calculated K matrix is.

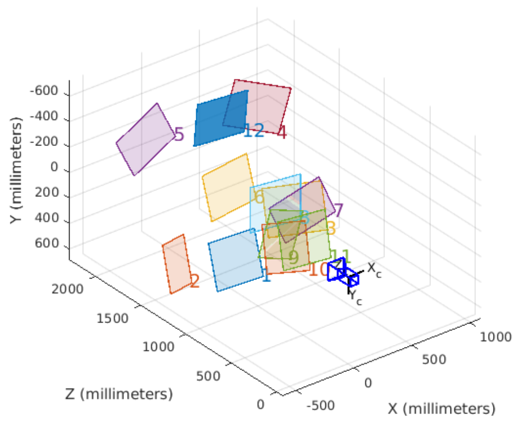
This overview describes camera calibration as a black box by only explaining the required inputs and the expected outputs. The next section details the mathematics on how the $\mathbf{K}$ matrix is actually calculated.

A.1.2 Derivation

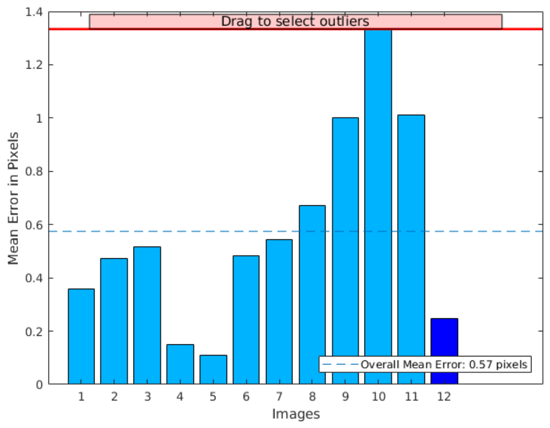
[61] presents the derivation for checkerboard camera calibration. To begin, recall that the pinhole camera model for projecting a 3D world coordinate point to the 2D image plane is given by the equation

$$s \begin{bmatrix} u & v & 1 \end{bmatrix}^T = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \begin{bmatrix} X & Y & Z & 1 \end{bmatrix}^T \tag{A.1}$$

where the goal is to project a world coordinate point, $\begin{bmatrix} X & Y & Z & 1 \end{bmatrix}^T$, to an image pixel, $\begin{bmatrix} u & v & 1 \end{bmatrix}^T$. $\mathbf{R}$ and $\mathbf{t}$ are the extrinsic rotation (3×3) and translation (3×1) that define



(a) Estimated checkerboard locations



(b) Calibration reprojection errors

Figure A.2: Example of a typical image used during checkerboard camera calibration.

how the camera is oriented relative to the 3D space around it. $\mathbf{K}$, is known as the camera intrinsic matrix and is given by

$$\mathbf{K} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{A.2}$$

where f_x, f_y describe the focal length of the camera and c_x, c_y describe the location of the principal point. Note that γ is also included. This represents the skew coefficient between the x and y axes, but is 0 for all applications in the thesis and therefore not included. γ is only included here for derivation completeness. The parameters of the intrinsic matrix are static (assuming no internal mechanical changes of the camera). These parameters ate the subject of interest during checkerboard camera calibration.

[61] presents the mathematics for intrinsic camera calibration using a flat checkerboard target. The first step is to assume that the target is flush with the Z axis of the world coordinates such that $Z = 0$ for all points. This simplifies the pinhole equation to

$$s \begin{bmatrix} u & v & 1 \end{bmatrix}^T = \mathbf{K} \begin{bmatrix} \mathbf{r_1} & \mathbf{r_2} & \mathbf{r_3} & \mathbf{t} \end{bmatrix} \begin{bmatrix} X & Y & 0 & 1 \end{bmatrix}^T \tag{A.3}$$

where $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3$ are column vectors representing the three columns of the rotation matrix. It is evident that $\mathbf{r}_3$ can be ignored by setting the checkerboard to be on the plane such that $Z = 0$ for all points, therefore simplifying the equation to

$$s \begin{bmatrix} u & v & 1 \end{bmatrix}^T = \mathbf{K} \begin{bmatrix} \mathbf{r_1} & \mathbf{r_2} & \mathbf{t} \end{bmatrix} \begin{bmatrix} X & Y & 1 \end{bmatrix}^T \tag{A.4}$$

This now resembles a 3×3 homography transformation, $\mathbf{H}$, where

$$s\begin{bmatrix}u & v & 1\end{bmatrix}^T = \mathbf{H}\begin{bmatrix}X & Y & 1\end{bmatrix}^T; \quad \mathbf{H} = \mathbf{K}\begin{bmatrix}\mathbf{r_1} & \mathbf{r_2} & \mathbf{t}\end{bmatrix} \tag{A.5}$$

Given an image of the checkerboard, a homography, denoted as $\mathbf{H} = \begin{bmatrix}\mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3\end{bmatrix}$, can be estimated. Equation A.5 can then be written as

$$\begin{bmatrix}\mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3\end{bmatrix} = \lambda\mathbf{K}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r_2} & \mathbf{t}\end{bmatrix} \tag{A.6}$$

where λ is an arbitrary scalar. By properties of a rotation matrix, $\mathbf{r}_1$ and $\mathbf{r}_2$ must be orthonormal, leaving the fundamental constraints on the intrinsic parameters as

$$\mathbf{h}_1^T\mathbf{K}^{-T}\mathbf{K}^{-1}\mathbf{h}_2 = 0 \tag{A.7}$$

$$\mathbf{h}_1^T\mathbf{K}^{-T}\mathbf{K}^{-1}\mathbf{h}_1 = \mathbf{h}_2^T\mathbf{K}^{-T}\mathbf{K}^{-1}\mathbf{h}_2 \tag{A.8}$$

where $\mathbf{K}^{-T} = (\mathbf{K}^{-1})^T = (\mathbf{K}^T)^{-1}$. In practice, K is solved for by using a closed-form solution solution, followed by a nonlinear optimization technique.

Closed-Form Solution

For the closed form-solution, let

$$\mathbf{B} = \mathbf{K}^{-T}\mathbf{K}^{-1} = \begin{bmatrix}B_{11} & B_{12} & B_{13}\\ B_{12} & B_{22} & B_{23}\\ B_{13} & B_{23} & B_{33}\end{bmatrix} = \begin{bmatrix}\frac{1}{f_x^2} & -\frac{\gamma}{f_x^2 f_y} & \frac{c_y\gamma - c_x f_y}{f_x^2 f_y}\\ -\frac{\gamma}{f_x^2 f_y} & \frac{\gamma^2}{f_x^2 f_y^2} + \frac{1}{f_y^2} & -\frac{\gamma(c_y\gamma - c_x f_y)}{f_x^2 f_y^2} - \frac{c_y}{f_y^2}\\ \frac{c_y\gamma - c_x f_y}{f_x^2 f_y} & -\frac{\gamma(c_y\gamma - c_x f_y)}{f_x^2 f_y^2} - \frac{c_y}{f_y^2} & \frac{(c_y\gamma - c_x f_y)^2}{f_x^2 f_y^2} + \frac{c_y^2}{f_y^2} + 1\end{bmatrix} \tag{A.9}$$

Note that $\mathbf{B}$ is symmetric, and can be defined by a 6D vector

$$\mathbf{b} = \begin{bmatrix}B_{11} & B_{12} & B_{22} & B_{13} & B_{23} & B_{33}\end{bmatrix}^T \tag{A.10}$$

Let the i^{th} column vector of $\mathbf{H}$ be $\mathbf{h}_i = \begin{bmatrix}h_{i1} & h_{i2} & h_{i3}\end{bmatrix}^T$. Then,

$$\mathbf{h}_i^T\mathbf{B}\mathbf{h}_j = \mathbf{v}_{ij}^T\mathbf{b} \tag{A.11}$$

where

$$\mathbf{v}_{ij} = \begin{bmatrix}h_{i1}h_{j1} & h_{i1}h_{j2} + h_{i2}h_{j1} & h_{i2}h_{j2} & h_{i3}h_{j1} + h_{i1}h_{j3} & h_{i3}h_{j2} + h_{i2}h_{j3} & h_{i3}h_{j3}\end{bmatrix}^T \tag{A.12}$$

The two fundamental constrains in equations A.7 and A.8 can be rewritten as

$$\begin{bmatrix} \mathbf{v}_{12}^T \\ (\mathbf{v}_{11} - \mathbf{v}_{22})^T \end{bmatrix} \mathbf{b} = 0 \tag{A.13}$$

If n images of the checkerboard are observed, stacking equation A.13 n times yields

$$\mathbf{Vb} = 0 \tag{A.14}$$

where $\mathbf{V}$ is a $2n \times 6$ matrix. The solution to equation A.14 is the eigenvector of $\mathbf{VV}^T$ associated with the smallest eigenvalue. Once $\mathbf{b}$ is estimated, it is possible to compute intrinsic matrix $\mathbf{K}$. See [61] for details. Once $\mathbf{K}$ is known, the extrinsic parameters can be computed from equation A.5.

Maximum Likelihood Estimation

The solution from the closed-form solution can then be refined using a maximum likelihood equation. This equation is used to solve for $\mathbf{K}$ given n images of the checkerboard with p relevant points (i.e detectable corners). To solve for $\mathbf{K}$, a maximum likelihood estimate is obtained by minimizing the function,

$$\sum_{i=1}^{n} \sum_{j=1}^{p} ||\mathbf{m}_{i,j} - \hat{\mathbf{m}}(\mathbf{K}, \mathbf{R}_i, \mathbf{t}_i, \mathbf{M}_j)||^2 \tag{A.15}$$

Where $\mathbf{m} = \begin{bmatrix} u & v \end{bmatrix}^T$, $\mathbf{M} = \begin{bmatrix} X & Y & Z \end{bmatrix}^T$, and $\hat{\mathbf{m}}$ is equation A.5 used for projecting a 3D point on the flat checkerboard plane to a 2D pixel. This equation requires an initial estimate, which is the $\mathbf{K}, \{\mathbf{R}_i, \mathbf{t}_i | i = 1, ..., n\}$ from the closed-form solution. The correct u, v values for $m_{i,j}$ are found by using any corner detection algorithm to detect the corners of the checkerboard pattern.

Distortion

[61] also presents how to augment this calibration to include lens distortion, specifically radial distortion, which is present in the cameras used in this thesis. In general, the first two terms of radial distortion are adequate, where

$$\begin{aligned} \widetilde{u} &= u(1 + k_1 r^2 + k_2 r^4) \\ \widetilde{v} &= v(1 + k_1 r^2 + k_2 r^4) \\ r^2 &= u^2 + v^2 \end{aligned} \tag{A.16}$$

where $\widetilde{u}, \widetilde{v}$ the distorted image coordinates and $k1, k2$ are the coefficients of the radial distortion. This can be incorporated into A.15, to yield a new maximum likelihood estimate of

$$\sum_{i=1}^{n} \sum_{j=1}^{p} ||\mathbf{m}_{i,j} - \hat{\mathbf{m}}(\mathbf{K}, k_1, k_2, \mathbf{R}_i, \mathbf{t}_i, \mathbf{M}_j)||^2 \tag{A.17}$$

where $\hat{\mathbf{m}}(\mathbf{K}, k_1, k_2, \mathbf{R}_i, \mathbf{t}_i, \mathbf{M}_j)$ is the projection of point M_j in image i according to equation A.5 followed by distortion correction via A.16.

A.2 Checkerboard Stereo Camera Calibration

A.2.1 Process Overview

To calibrate a stereo camera with a checkerboard, the user must provide image pairs of the checkerboard target at numerous angles and skews captured by each camera (Fig. A.3). It is critical for stereo calibration that the checkerboard does not move between the capturing image pairs (i.e. the global coordinates of the checkerboard must be identical for image pairs). Similar to single camera calibration, a corner detection technique is used to determine the actual (u, v) location of all the corners on the checkerboard in both images.

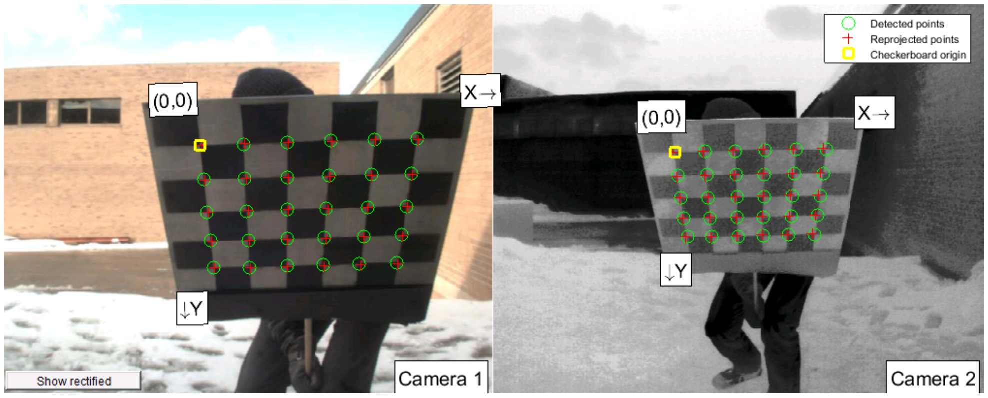


Figure A.3: Example of a typical image pair used during checkerboard stereo camera calibration. Note the identical posture of the man holding the checkerboard indicates the images were taken synchronously, ensuring consistent checkerboard location.

The next step is to set the coordinate frame of the checkerboard in world coordinates, which is done similar to single camera calibration. For each checkerboard location, an outside corner point is set as $(0,0,0)$ and the checkerboard is assumed flat along the Z axis such that $Z = 0$ for all points. The real-world distance of the squares and the number of squares is an input to the calibration, and therefore the coordinates of all corners can be calculated. The world coordinates do not depend on the camera, and are set once per stereo image pair.

Stereo calibration also requires the intrinsic matrix of the cameras to be provided. If not provided, the algorithm can calculate the intrinsic information by first performing single camera calibration on the images for each camera independently. However, the setup of the checkerboard to optimize single camera calibration versus stereo camera calibration may differ, so it is best to provide the intrinsic matrices.

This setup outlines the inputs to the the checkerboard stereo camera calibration process, which are the (u, v) coordinates of the checkerboard corners, the (X, Y, Z) world coordinates of the checkerboard corners for all image pairs, and the intrinsic matrices of the cameras. The goal of the mathematics is therefore to solve for the rotation, $\mathbf{R}$, and translation ,$\mathbf{t}$, that describes the location of camera 2 relative to camera 1 that minimizes the error between the (u, v) corners detected using computer vision on an image captured by camera 1 and the (u, v) coordinates from transforming a corresponding 3D (X, Y, Z) point in the coordinate frame of camera 2 to the 2D image plane of camera 1.

The output of the camera calibration provides the static transformation $T_{c1,c2}$, which outlines the rotation and translation information from camera 2 to camera 1 in a single matrix

$$\mathbf{T}_{c1,c2} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{33} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.18}$$

This is the matrix that the mathematics seeks to solve for.

A.2.2 Derivation

The mathematics of stereo camera calibration build upon single camera calibration. For stereo camera calibration, the checkerboard is still flat, but is captured while in the same location by two different cameras. The world coordinates of the checkerboard are defined the same as the single camera calibration, and the intrinsic matrix, $\mathbf{K}$, is known for both the cameras (via previous single camera calibration). The equation to solve for the transformation between cameras therefore becomes

$$s \begin{bmatrix} u & v & 1 \end{bmatrix}_{c1}^{T} = \mathbf{K}_{c1} \mathbf{T}_{c1,c2} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}_{c2} \begin{bmatrix} X & Y & Z & 1 \end{bmatrix}_{c2}^{T} \tag{A.19}$$

where $\mathbf{T}_{c1,c2}$ is the transformation matrix from a 3D point in camera 2's coordinate frame to camera 1's coordinate frame. Since the intrinsic parameters of both cameras are known, it is possible to calculate $\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}_{c2}$ using equation A.5 based on camera 2's intrinsic calibration. The only unknown in the equation is $\mathbf{T}_{c1,c2}$ (see equation A.18), which represents the rotation and translation information for converting a point in the coordinate frame of camera 2 to the coordinate frame of camera 1. This transformation matrix can be solved for for a checkerboard with p corners by minimizing equation

$$\sum_{i=1}^{n}\sum_{j=1}^{p}||\mathbf{m}_{c1,i,j} - \hat{\mathbf{m}}(\mathbf{T}_{c1,c2}, \mathbf{M}_j)||^2 \tag{A.20}$$

where $\hat{\mathbf{m}}$ is equation A.19 with world checkerboard coordinate $\mathbf{M}_j$ in image i being used to solve for $\mathbf{T}_{c1,c2}$. This equation can be solved for a transformation in either direction, but is generally solved converting from camera 2 to camera 1. The correct u, v values for $m_{i,j}$ are found by using any corner detection algorithm to detect the corners of the checkerboard pattern in camera 1.

A.3 ADAM Optimizer

This work used an implementation of the ADAM optimizer [27] to train the CNN. This optimizer is similar to the basic gradient descent method for training a neural network. The ADAM optimizer builds on basic gradient descent by computing individual adaptive learning rates using estimates of the first and second moments of the gradients. These first and second moments of the gradients work by incorporating previous calculated gradients into the update equation to help avoid local minimums. The full algorithm for ADAM is summarized in algorithm 4.

Algorithm 4 ADAM Optimizer [27]

Input	**Description**
α	Stepsize
β_1, β_2	Exponential decay rates for the moment estimates
$f(\theta)$	Stochastic objective function with parameters θ
θ_0	Initial parameter vector
ϵ	10^{-7}, used to avoid divide by zero error

```
 1: $m_0 \leftarrow 0$                                  ▷ Initialize first moment vector
 2: $v_0 \leftarrow 0$                                  ▷ Initialize second moment vector
 3: $t \leftarrow 0$                                    ▷ Initialize timestep
 4:
 5: while $\theta_t$ not converged do
 6:     $t \leftarrow t + 1$
 7:     $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$      ▷ Get gradients w.r.t. stochastic objective timestep $t$
 8:     $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$      ▷ Update biased first moment estimate
 9:     $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$      ▷ Update biased second raw moment estimate
10:     $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$      ▷ Compute bias-corrected first moment estimate
11:     $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$      ▷ Compute bias-corrected second raw moment estimate
12:     $\theta \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$      ▷ Update parameters
13: end while
14:
15: return $\theta_t$                                   ▷ Resulting parameters
```

Appendix B

Network Architecture

The convolutional neural network implemented in this work is MobileNetV2 [51] as a feature extractor with Deeplab V3 [10] to perform semantic pixel-wise segmentation of defects in images. The complete details of each network are outlined in their respective papers, but details are also outlined here.

B.1 MobileNetV2

MobileNetV2 uses depthwise separable convolutions, which are a key building block for an efficient neural network. These are used in tandem with a pointwise convolution to replace full convolutional layers with a more efficient layer. A depthwise convolution applies a kernel to only one channel of the tensor such that the resulting image is the same size. For example, if the input is a $50 \times 50 \times 6$ tensor, the depthwise convolution may be 6 $k \times k$ kernels that are applied to each channel of the tensor separately. The output is therefore 6 50×50 tensors that are stacked to be $50 \times 50 \times 6$. A pointwise convolution is a $1 \times 1 \times d_i$ kernel that is applied over the entire depth, d_i, of the input tensor. The resulting tensor will have a depth, d_j, depending on the number of kernel filters used. For example, if 10 $1 \times 1 \times 6$ kernels are applied to a $50 \times 50 \times 6$ tensor, the output is $50 \times 50 \times 10$.

Standard convolution on an $h_i \times w_i \times d_i$ tensor with d_j $k \times k$ kernels results in a computational cost of

$$h_i \cdot w_i \cdot d_i \cdot d_j \cdot k \cdot k \tag{B.1}$$

as all d_j kernels will required $k \cdot k$ operations at each location in the original tensor. Conversely, depthwise seperable convolutions empirically work as well as standard convolution but with less operations. The depthwise convolution involves $k^2(h_i \cdot w_i \cdot d_i)$ operations and yields a $h_i \times w_i \times d_i$ tensor. The pointwise convolution is $d_j(h_i \cdot w_i \cdot d_i)$ operations, which when added to the depthwise operations yields a total of number of operations defined as

$$h_i \cdot w_i \cdot d_i(k^2 + d_j) \tag{B.2}$$

The reduction of computational cost is proportional to k^2. In MobileNetV2, $k = 3$ for all depthwise convolutions, and therefore is estimated to reduce computional cost by approximately 8 or 9 times.

MobileNetV2 also uses linear bottleneck layers, as empirical evidence shows that using linear layers prevents non-linearities (e.g. relu) from destroying too much information [51]. The linear bottleneck can be viewed as a pointwise convolution, but with a linear activation function. The basic convolutional block of MobileNetV2 is called a bottleneck residual block, and is summarized in Table B.1 and illustrated in Fig. B.1.

Table B.1: Bottleneck residual block used in MobileNetV2 [51]. Transforms an input tensor with k channels, to k' channels using a stride s and expansion factor t. Note that it is common for $s = 1$, which makes the input and ouput tensors of the depthwise operation the same size.

Input	**Operator**	**Output**
$h \times w \times k$	1x1 conv2d, ReLU6	$h \times w \times (tk)$
$h \times w \times (tk)$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times (tk)$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

B.2 Deeplab V3

Deeplab V3 uses a convolutional layer know as an atrous convolution to foster accurate pixel-wise labeling. This atrous convolution uses a dilation rate to indicate a separation between learnable parameters in the kernel. That is, an atrous convolution skips pixels in the kernel to get a larger field of view when applying a a smaller convolution filter. For example, Fig. B.2 shows the difference between a 3×3 convolution and a 3×3 atrous convolution with a dilation rate of 2. The atrous convolution covers a 5×5 area for the same amount of computation as the regular convolution.

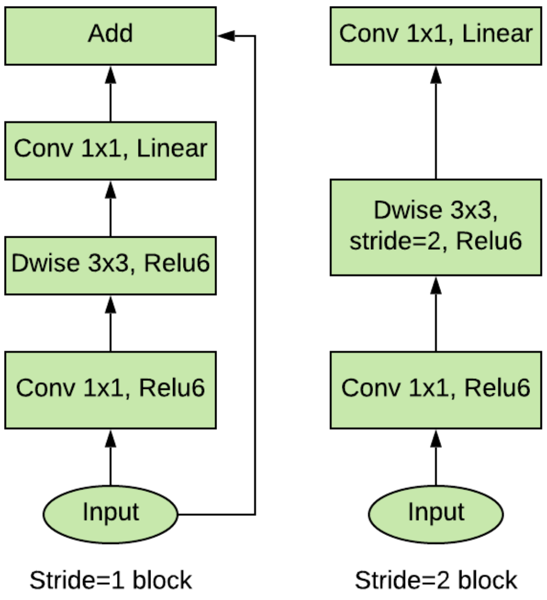


Figure B.1: MobilenetV2 Bottleneck residual block architecture.

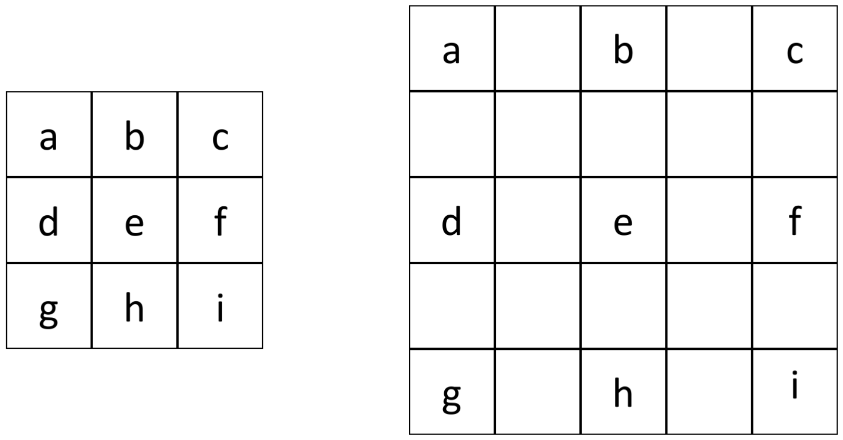


Figure B.2: Left: Regular 3×3 convolution kernel. Right: 3×3 atrous convolution with dilation rate=2. Blank squares are ignored by the kernel.

B.3 Combined Final CNN

The bottleneck residual block and atrous convolution are the building blocks to be able to implement MobileNetV2 and Deeplab V3 for pixel-wise labeling of spalls and delaminations. There are a few other important concepts to note before understanding the full network architecture. First, the rectified linear unit (relu) activation function given by

$$\text{relu}(x) = \max(0, x) \tag{B.3}$$

and the relu6 actication function given by

$$\text{relu6}(x) = \max(6, \text{relu}(x)) \tag{B.4}$$

are used to provide non-linearities to the CNN. In addition, batch normalization (BN) and bilinear upsampling are also used at locations throughout the network architecture. Batch normalization applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1 [12]. This batch normalization is used to prevent gradients from ”blowing up” to large values, as this can limit the ability of the neural network to work as an aggregate of functions.

Bilinear upsampling upscales the spatial density of the image using bilinear interpolation to fill in the missing data between pixels. This is required as, even though all tensors are padded so the output size is the same as the input size, a stride of 2 causes the dimensions of the input tensor to be halved. This reduction in dimensions improves computation by reducing the number of operations performed at each layer. The bilinear upscaling is used to ensure the output tensor has the same density as the input image, which is mandatory for pixel-wise labeling. With this knowledge, it is possible to describe the entire network, as is done in Table B.2

Table B.2: Full summary of the neural network implemented for classifying spalls and delaminations. The network begins with MobileNetV2 as a feature extractor, where t is the expansion ratio, c is the number of filters, n is the number of times the layer is used, and s is the stride. Note that when $n > 1, s > 2$, the stride is only used the first iteration, and then reverted to 1. The layers with a dilation rate perform atrous depthwise convolution instead of regular depthwise convolution. This is for the purpose of pixel-wise classification. The network then splits into two branches that extract information from different scales. These branches are then concatenated and each pixel is classified as a defect (spall/delamination) or no defect using a softmax activation.

MobileNetV2 feature extraction

Input	**Operator**	t	c	n	s
$512 \times 640 \times C$	conv2d, BN, relu6	-	32	1	2
$256 \times 320 \times 32$	bottleneck res block	1	16	1	1
$256 \times 320 \times 16$	bottleneck res block	6	24	2	2
$128 \times 160 \times 24$	bottleneck res block	6	32	3	2
$64 \times 80 \times 32$	bottleneck res block	6	64	4	1
$64 \times 80 \times 64$	bottleneck res block with dilation=2	6	96	4	1
$64 \times 80 \times 96$	bottleneck res block with dilation=2	6	160	3	1
$64 \times 80 \times 160$	bottleneck res block with dilation=4	6	320	1	1

Branches for atrous spatial pyramid pooling

Input	**Operator**	**kernel**	c
$64 \times 80 \times 320$ (branch 1)	average pooling	32×40	-
$2 \times 2 \times 320$	conv2d, BN, relu	1×1	256
$2 \times 2 \times 256$	bilinear upsampling	32×40	-
$64 \times 80 \times 320$ (branch 2)	conv2d, BN, relu	1×1	256

Branch concatenation and final classification

Input	**Operator**	**kernel**	c
$2 - (64 \times 80 \times 256)$	concatenate branches 1,2	-	-
$64 \times 80 \times 512$	conv2d	1×1	256
$64 \times 80 \times 256$	bilinear upsampling	8×8	2
$512 \times 640 \times 2$	softmax	-	-

Appendix C

Labeling Example

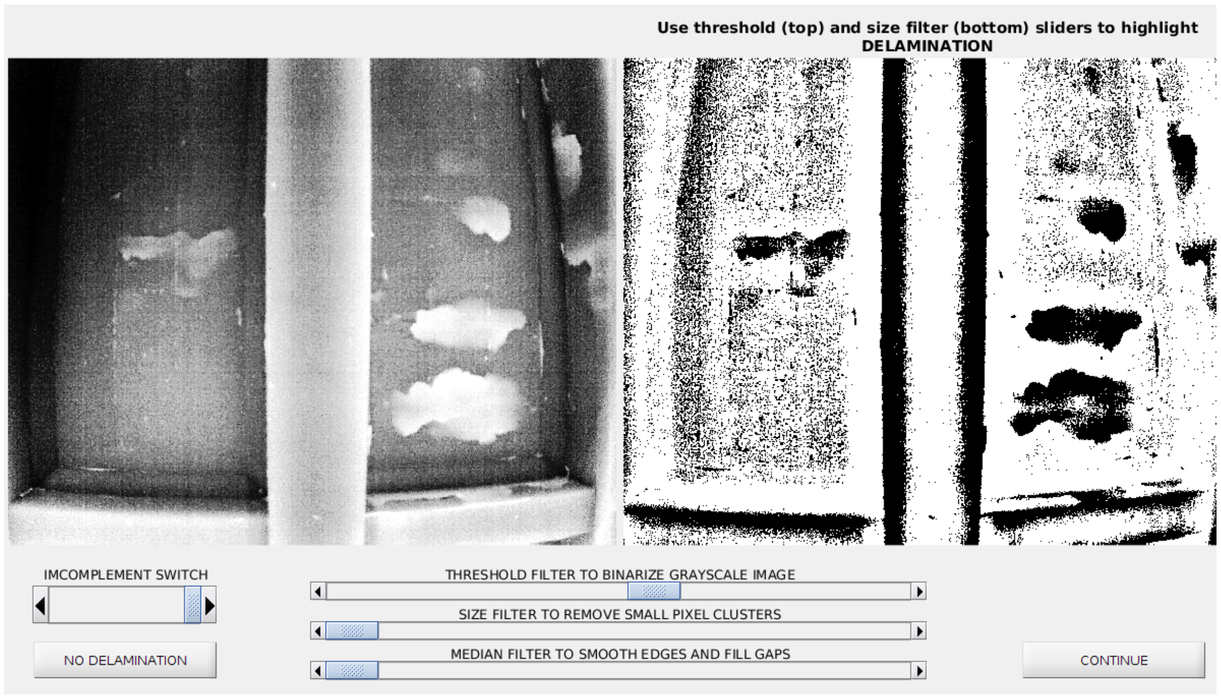

Use threshold (top) and size filter (bottom) sliders to highlight DELAMINATION
IMCOMPLEMENT SWITCH
NO DELAMINATION
THRESHOLD FILTER TO BINARIZE GRAYSCALE IMAGE
SIZE FILTER TO REMOVE SMALL PIXEL CLUSTERS
MEDIAN FILTER TO SMOOTH EDGES AND FILL GAPS
CONTINUE

Figure C.1: Figure showing the thresholding step for manual labeling of delaminations in an infrared photo (left). The complement switch has been activated to set the label of pixels lighter than the grayscale threshold as 1 (i.e. defect). The delaminations appear strong in the mask (right), but there is significant noise and other objects still labeled as defect using thresholding alone.

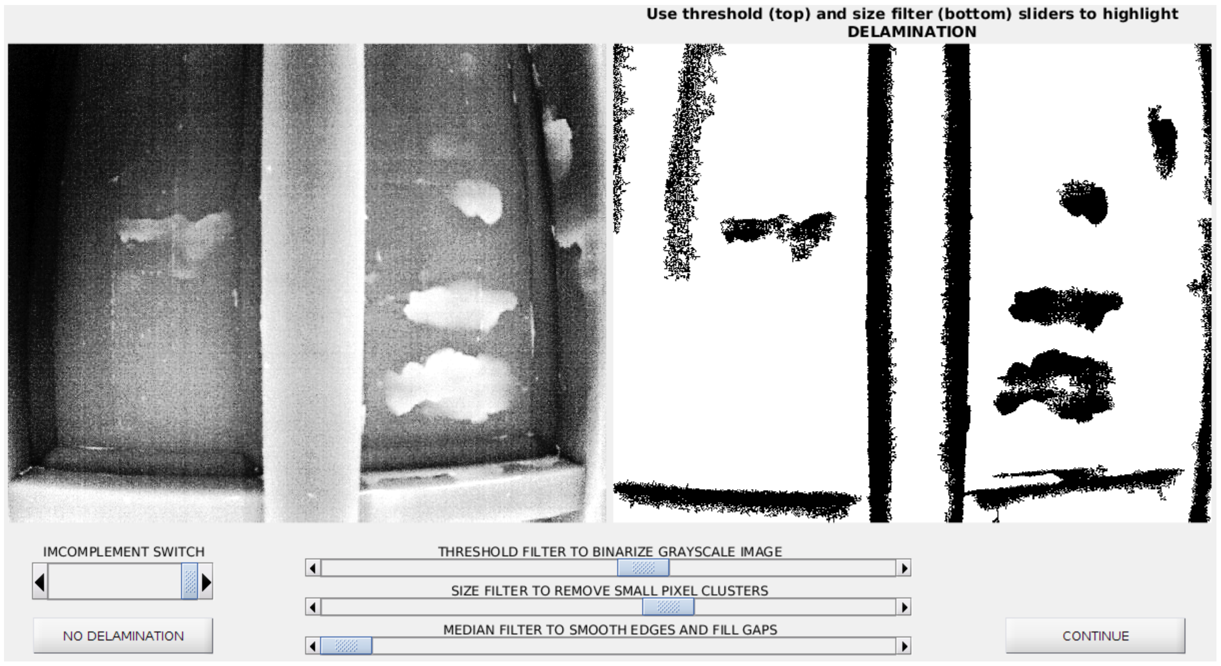

Use threshold (top) and size filter (bottom) sliders to highlight
DELAMINATION
IMCOMPLEMENT SWITCH
THRESHOLD FILTER TO BINARIZE GRAYSCALE IMAGE
SIZE FILTER TO REMOVE SMALL PIXEL CLUSTERS
NO DELAMINATION
MEDIAN FILTER TO SMOOTH EDGES AND FILL GAPS
CONTINUE

Figure C.2: Figure showing the size filter step for removing small pixel clusters from the binary mask. All the noise has been removed from the background of the image, but there are still large objects (e.g. edge of concrete girder) that are labeled as defect in the image mask.

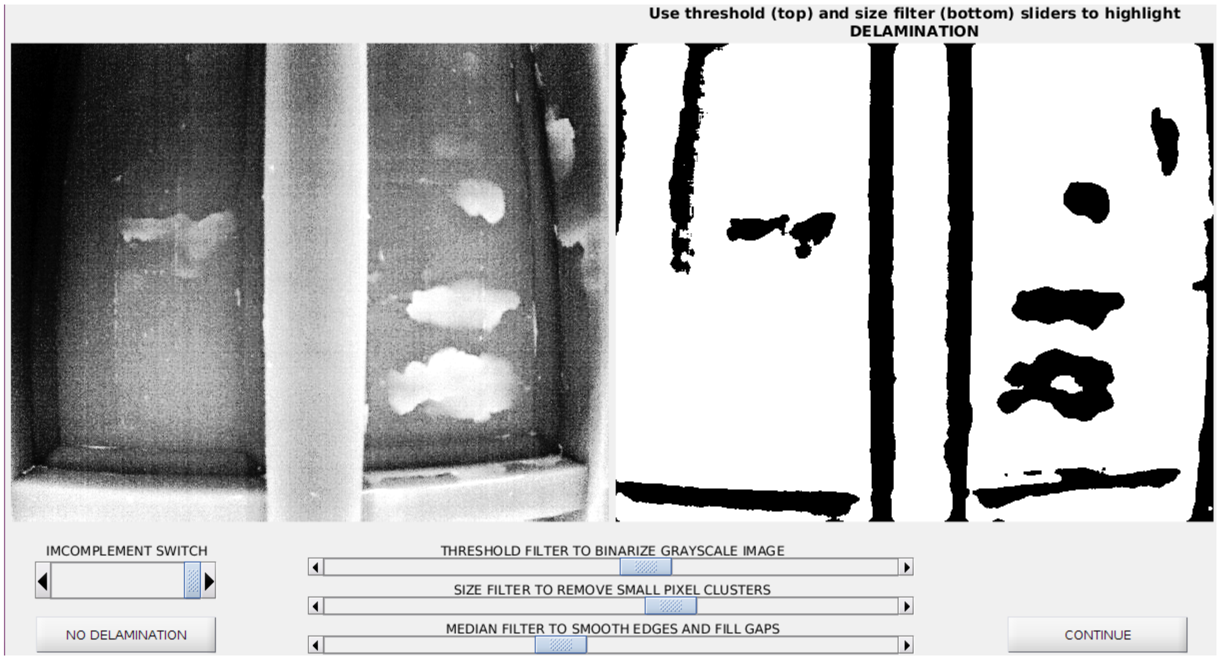

Use threshold (top) and size filter (bottom) sliders to highlight DELAMINATION
IMCOMPLEMENT SWITCH
THRESHOLD FILTER TO BINARIZE GRAYSCALE IMAGE
SIZE FILTER TO REMOVE SMALL PIXEL CLUSTERS
NO DELAMINATION
MEDIAN FILTER TO SMOOTH EDGES AND FILL GAPS
CONTINUE

Figure C.3: Figure showing the median filter step for smoothing out edges in the binary mask. This smoothing also fills in any small noise that is within the defect area. However, manual intervention is still needed in some cases.

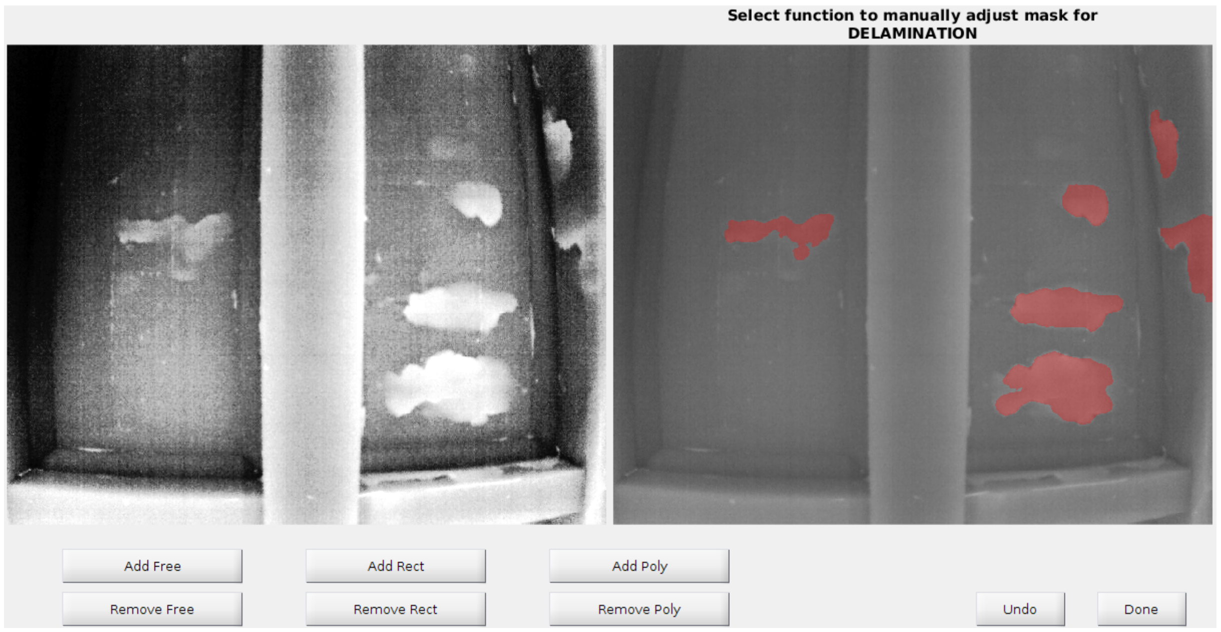


Figure C.4: Figure showing the final mask (red shows defect pixel labels) overlayed over the original image after manual intervention. Manual intervention was able to remove objects that appeared bright but were not a delamination and was also able to fill in any remaining gaps in the delamination areas.