Crack Detection and Measurement Utilizing Image-Based Reconstruction

---

Paul Zheng

Project and Report

Committee Members:

Dr. Cris Moen

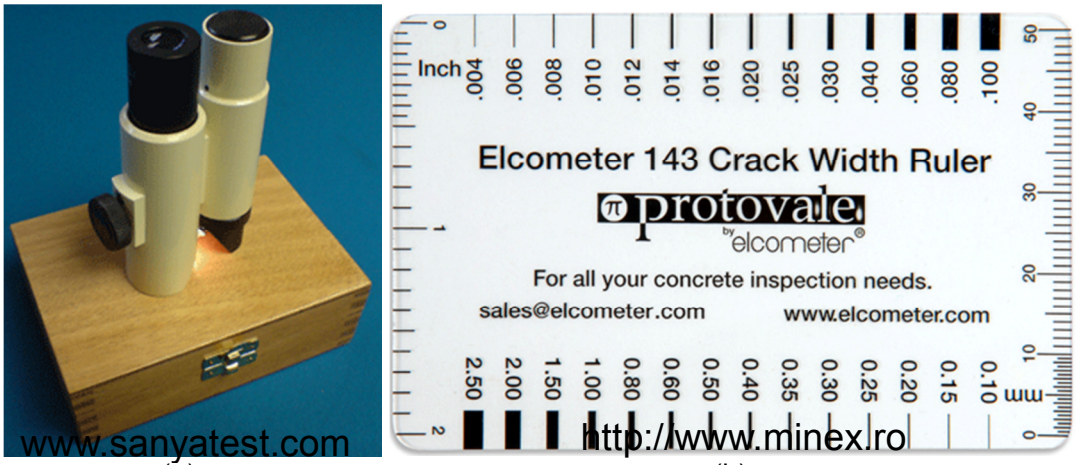
Dr. Carin Roberts-Wollmann

Dr. Tommy Cousins

Table of Contents

INTRODUCTION

Image-based reconstruction for automated crack detection has been on the rise for the past decade or so; this new technology can be applicable to many different areas such as laboratory testing, field inspections, construction quality control and quality assurance, and post disaster reconnaissance. An added feature to automated crack detection is the ability to perform digital crack measurements with increased safety. Crack detection during experimental testing may require researchers to mark cracks on the specimens, whereas researchers can take photographs of the specimens from a safe distance and have the reconstructed model digital crack detection. Automated crack detection along with digital crack measurements will increase the quantity of cracks observed and measured. Increased quantity could reduce cost of field inspections by reducing inspection time. Compared to traditional crack measurement techniques such as a crack detection pocket microscope (crack-scope) or crack width card (also referred to as a crack width gauge), safety would be less of a concern since photographs for image reconstruction can be taken at a distance rather than having to be directly against the structure; both of these traditional tools can be seen in Figure 1. Safety is a major concern in post disaster reconnaissance; after an event such as an earthquake or tsunami, structures have to be examined to determine the extent of damage. By utilizing image based reconstruction, assessments can be made without placing the inspector or engineer in dangerous situations.



(a) (b)

Figure 1: (a) Crack-scope, (b) Crack Width Card

In order for image based reconstruction for automated crack detection and digital crack measurements to be implemented into practice, it must be evaluated with real cracks from laboratory experiments. The goal of this paper is to demonstrate how this new technology can be applied to reinforced concrete experiments by verifying crack width measurements retrieved from image based reconstruction based on an automated crack detection algorithm and compare it to manual crack measurements from reinforced concrete slab tests investigating the performance of corrosion resistant reinforcing bars (CRR) (Bowen et al. 2013). Figure 2 represents a common crack width plot that can be found in Bowen et al. which contains crack measurements that were compared to crack measurements from this project. Also, Figure 3 demonstrates the tight space between the frame columns and the constant moment region of the specimens where crack measurements were made.

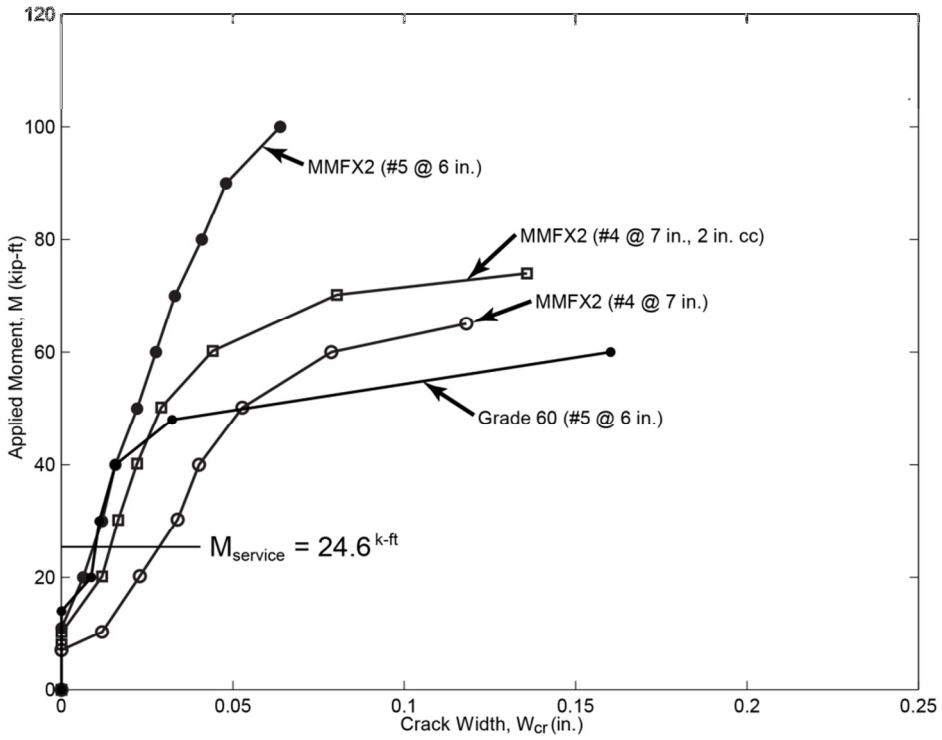


Figure 2: Example of Crack Width Plot from Bowen et al.

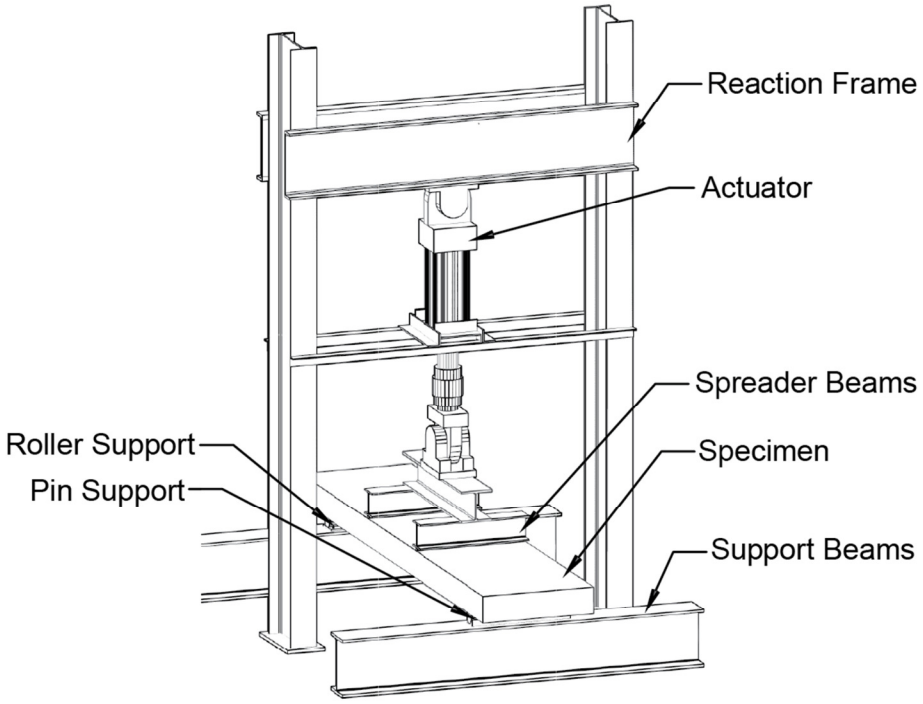


Figure 3: Corrosion Resistant Reinforcing Bar Project Test Setup

Crack width measurements in laboratory experiments are commonly performed using a crack-scope or a crack width card. A crack-scope requires proper lighting; crack-scopes

typically come with an attached one millimeter light bulb, which is fragile and difficult to replace. Crack-scopes have precision of ±0.02 millimeters. Crack width cards require the inspector to judge the crack width relative to specific line thicknesses that are of a known width. Crack cards are common among engineers, however, the precision and accuracy of this technique is inconsistent. These methods are time-consuming if there are a large quantity of cracks and potentially unsafe. For this reason, researchers have been developing methods of crack detection as well as crack property retrieval for damaged structures.

There have been other crack detection systems developed for identifying damaged structures. Abdel-Qader et al. compared four crack detection techniques (fast Haar transform, fast Fourier transform, Sobel, and Canny) on bridge surfaces (Abdel-Qader et al. 2003) finding that the fast Haar transform is the most reliable method out of the four compared methods. Using continuous wavelet transform, Subirats et al. identified cracks for pavement surfaces (Subirats et al. 2006). In addition to crack detection, some researchers have incorporated digital crack measurements into their methods. One such method requires the use of referencing frames to measure the cracks, but is only capable of doing so in a small region (Barazzetti et al. 2009). A methodology has been developed that generates crack maps and distance fields that enables the user to retrieve crack information, such as crack width, crack length, and crack orientation (Zhu et al. 2011a). These automated methods have been refined to incorporate detection of exposed rebar (Zhu et al. 2011b). A different algorithm found in the literature utilizes the concept of depth perception to identify surfaces of various structural elements, location of cracks, and depth of cracks (Jahanshahi et al. 2011). Unfortunately, these approaches are only useful in two-dimensions, which limits the applicability to real-world structures. Most methods only construct the cracks and not the full structure; some methods are capable of retrieving three-dimensional crack information, but not incorporate a three-dimensional structure.

There have been attempts at three-dimensional surface recognition, but available methodologies are not capable of automatically identifying cracks; detection of large-scale

concrete columns for bridge inspection has been developed, which incorporate a concept of precision (how many detected columns are correctly identified) and accuracy (how many actual columns are correctly identified) (Zhu et al. 2010). Jahanshahi et al. expanded the depth perception concept to incorporate three-dimensional scene reconstruction (Jahanshahi et al. 2012). Unfortunately, cracks are still determined on a two-dimensional basis. Photogrammetry has been successfully utilized to capture the behavior of concrete structures as cracks propagate, including tests on flexure/shear, tension and plate tests using photogrammetry to monitor the crack behavior of each specimen (Benning et al. 2004). However, these tests only looked at a single surface. Photogrammetry has the capability to capture structural behavior in three-dimensions, but is mainly only focused on two-dimensions for cracks. It also is not applicable in real life because of the number of reference points that are required to be placed on the structures; photogrammetry represents discrete points rather than a continuous structure. The method discussed in this paper includes three-dimensional capability and crack detection.

The crack detection approach discussed in this paper is based on the work of Torak, Golparvar-Fard, and Kochersberger (Torak et al. 2012). This method of automated crack detection utilizes images from cameras to process into a three-dimensional point cloud, which can be made into a mesh model. In the literature, others have utilized other types of equipment, such as a laser scanner, to generate point clouds (Tang et al. 2010). Laser scanners have high precision with the capability of generating accurate point clouds; however, the equipment is expensive and should not be brought from jobsite to jobsite. For the experiments, a digital single-lens reflex (DSLR) camera was used, but a simple point-and-shoot digital camera could have also been utilized. Once a three-dimensional model is generated, the crack detection algorithm (CDA) identifies the location of cracks through the comparison of normals of adjacent mesh triangles. If the normals of the meshes differ by a certain threshold, then the algorithm recognizes there is a crack.

In this paper, crack measurements are retrieved and compared to manual measurements on reinforced concrete slabs loaded in one-way flexure. Manual crack measurements were made on multiple cracks for each concrete specimen utilizing a crack-scope as load was applied. The new crack detection methodology is evaluated for accuracy in matching with manual crack measurements, and also for precision and recall, which is the ability to properly identify cracks, as discussed in the following sections.

CRACK DETECTION PROCEDURE

In this section, the crack detection procedure is explained in depth. The crack detection procedure entails taking images captured from a camera, running these images through a program that stitches the images together and generates a three-dimensional (3D) point cloud using Structures-from-Motion (SfM), utilizing a program to trim the point cloud and create a 3D mesh model from the point cloud, and finally using code to detect and color-code cracks in the mesh model based on normals of the mesh model. The robustness of the crack detection code will be checked using the concept of precision and accuracy, which will be explained later in this section.

The images required for crack detection can be taken from equipment as simple as a point-and-shoot digital camera or more complex equipment such as a digital single-lens reflex (DSLR) camera. The DSLR camera would produce higher resolution pictures, but the images would be larger in data size, which may require more processing time during the SfM process. An aspect that must be kept in mind is quality versus quantity. Enough images must be captured to obtain a complete reconstruction of the object of interest, but too many images can greatly increase computational time. Furthermore, with high resolution images, the computational time can also be increased. However, with a low resolution camera, more images may be needed to acquire all the information necessary. Fortunately, if more images are

required to generate the full reconstructed 3D model, those pictures can be processed through the SfM program to fill in missing regions, so long as the object is still available (i.e., not demolished or underwent significant changes). Figure 4 represents some of the many images that were processed through the SfM program; these pictures contain both overview shots and zoomed in shots of defects, such as cracks.

Figure 4: Sample of Images to be processed through Structure-from-Motion program

Once the images have been taken, they can be processed through a Structure-from-Motion program to generate the 3D point cloud model; for this project, the open source program VisualSFM (found at http://homes.cs.washington.edu/~ccwu/vsfm/) was used. By identifying a feature point in an image, such as a corner (edges with gradients in multiple directions), and tracking that point through all the images, a 3D point cloud model can be generated. One image is set as a reference with as many feature points that can be identified, then all the images in the sequence are examined with respect to the reference image and the feature points are matched. After all the images have been compared to the reference image, the next image is set as the reference image and all the pictures are compared to it. This is repeated until all the images have been set as the reference image and compared to all the pictures in the set. At the end of the process, a three-dimensional point cloud model is created with the proper color fills.

The next step in the crack detection process is trimming the 3D point cloud. The SfM process will likely generate more than just the structure of interest, so a program is required to open the file with the point cloud and remove the points associated with unnecessary information. For this project, Meshlab was used to trim and perform the mesh generation; Meshlab is a free program (found at http://meshlab.sourceforge.net/). Below in Figure 5, the 3D point cloud is shown without trimming. This process can be quite tedious depending on how many excess points are present. Furthermore, cleaning up noise close to the structure of interest can be time consuming due to the carefulness that is required; the user must be careful not to accidently remove portions of the structure while trimming.

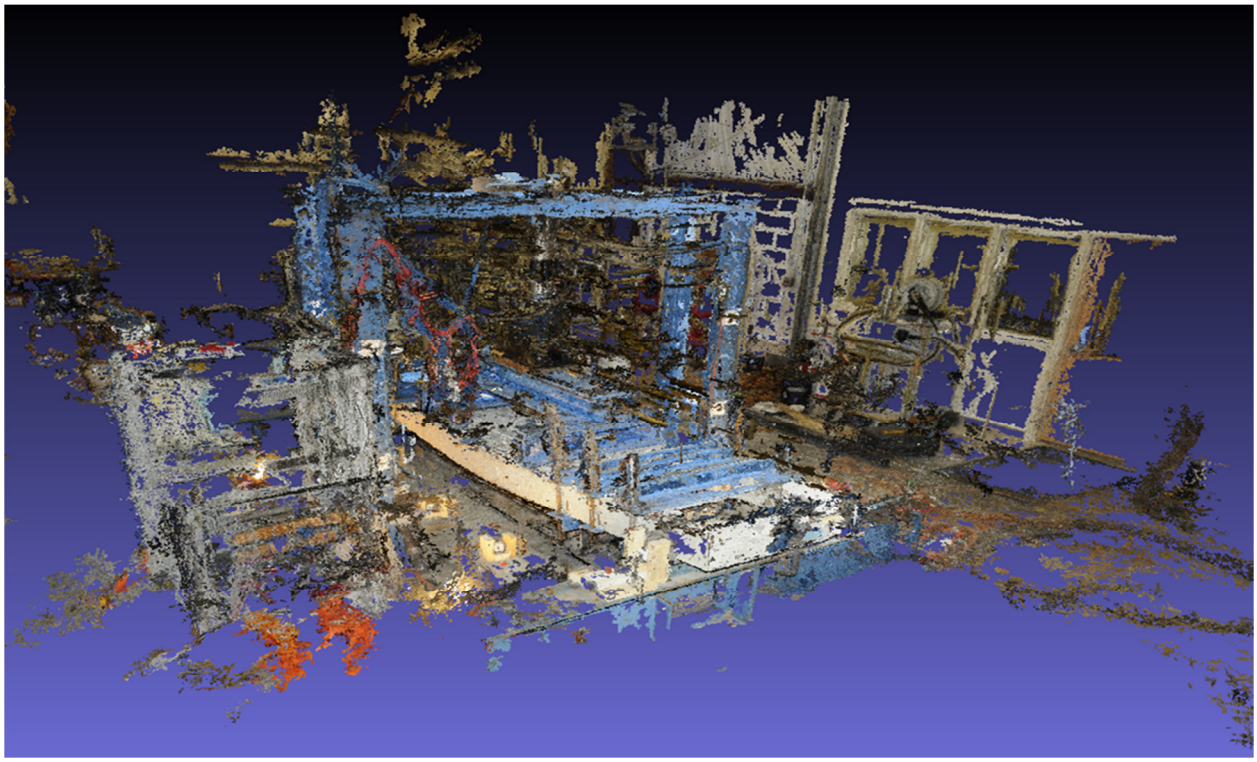
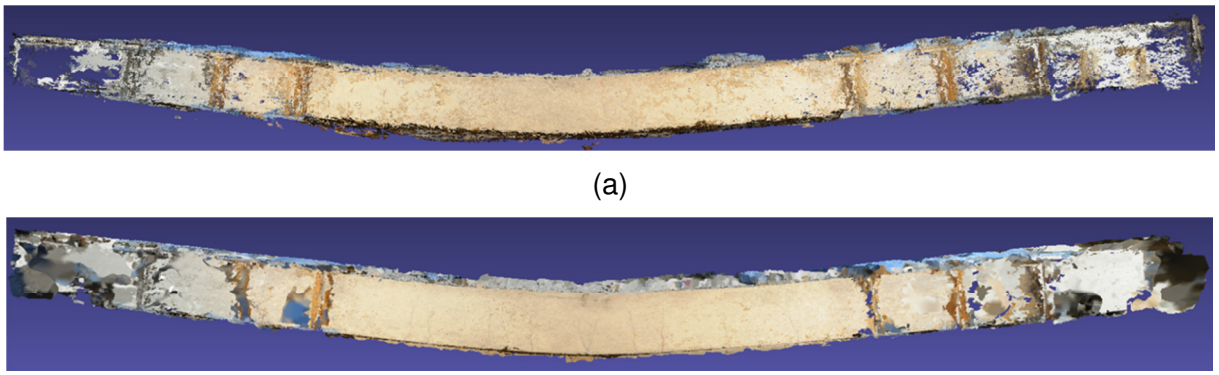


Figure 5: Three-Dimensional Point Cloud Model

A mesh model is generated from a trimmed point cloud model. The software Meshlab can easily create the mesh model, which gives the user control of the mesh density. There are multiple options in Meshlab for the creation of the mesh model (Poisson, VCG, and Ball Pivoting); the Poisson mesh is the most common option used for mesh generation as well as

the option utilized by Torak et al. Therefore, Poisson Reconstruction for mesh creation was used. The Poisson mesh is able to handle curved surfaces since it creates a mesh with a curve mesh, however, this means that there may be more trimming needed after the mesh is made. In Figure 6, a trimmed point cloud of the slab specimen is shown along with the mesh that is generated from the reduced point cloud. Before the mesh can be created, normals for the point cloud needs to be generated, which are used for the crack detection code.



(a)

(b)

Figure 6: Slab Specimen (a) Before and (b) After Meshing

The crack detection algorithm (CDA) examines the face normals that were generated along with the mesh model. Some faces may not be aligned with a surface (horizontal or vertical) due to the cracks in the structure. The presence of cracks will cause points in the point cloud model to be out-of-plane with the vertical or horizontal surface. Generating a mesh model with out-of-plane points will cause faces to not be in alignment with a surface. For this research, only the vertical surfaces with cracks were examined with the CDA. Faces in alignment will have normals that are perpendicular to the vertical axis, but around cracks, the normals will not be perpendicular. This is the premise of how the CDA identifies cracks in a structure. Figure 7 is a representation of how the CDA compares normals; the black lines are the expected vertical axis and the normal of an uncracked surface whereas the blue lines represent a triangular mesh for a face of a cracked region and its normal. The angle β is the angle between the vertical axis and the normal of the face associated with a crack region.

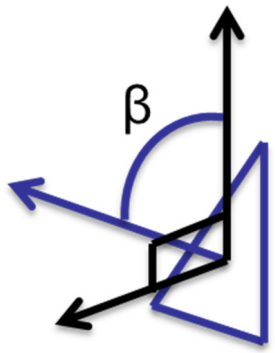


Figure 7: Crack Detection Algorithm Methodology

In the CDA, the angle α, the absolute value of the difference between the perpendicular angle (90 degrees) and the angle β, is compared to a threshold angle, θ, which is set by the user. If α is greater than θ, then the program will color the crack accordingly. The CDA examines all the face normals, color coding cracks then producing a file which can be viewed in Meshlab to see the cracks more clearly.

$$\alpha = |90 - \beta| \tag{1}$$

$$\text{If } \alpha < \theta, \text{ then uncracked; if } \alpha \geq \theta, \text{ then cracked} \tag{2}$$

where θ is the threshold angle set by the user

After the CDA is complete, the algorithm has to be checked on how well it performed at identifying cracks. To do this, a Precision and Accuracy code is implemented. Precision is the measure of how many detected cracks are correctly identified, whereas Accuracy is the measure of how many actual cracks are correctly identified. Precision and Accuracy can be applied to the entire structure, or to specific regions, since the user inputs the values for the calculations of these two parameters. The Precision and Accuracy equations are shown below.

$$P = \frac{TP}{TP+FP} \tag{3}$$

$$A = \frac{TP}{TP+FN} \tag{4}$$

where P = Precision

A = Accuracy

TP = Number of correctly detected cracks (True Positive)

FP = Number of incorrectly detected cracks (False Positive)

FN = Number of actual cracks not detected (False Negative)

CRACK DETECTION AND MEASUREMENTS

Crack Detection Results

The crack detection algorithm was performed on specimen T17 after failure when the crack widths are at their largest, thus the most visible. It was the only model with clear enough cracks to perform the CDA on due to lighting issues which will be explained later. Based on the figures below, it can be seen without running the CDA that the mesh generation produced the same cracks. The arrows in Figure 8 show which cracks from the photo corresponds with the cracks from the mesh generation. Using the CDA, the cracks in the mesh will be colored so that they are easier to identify and compare.

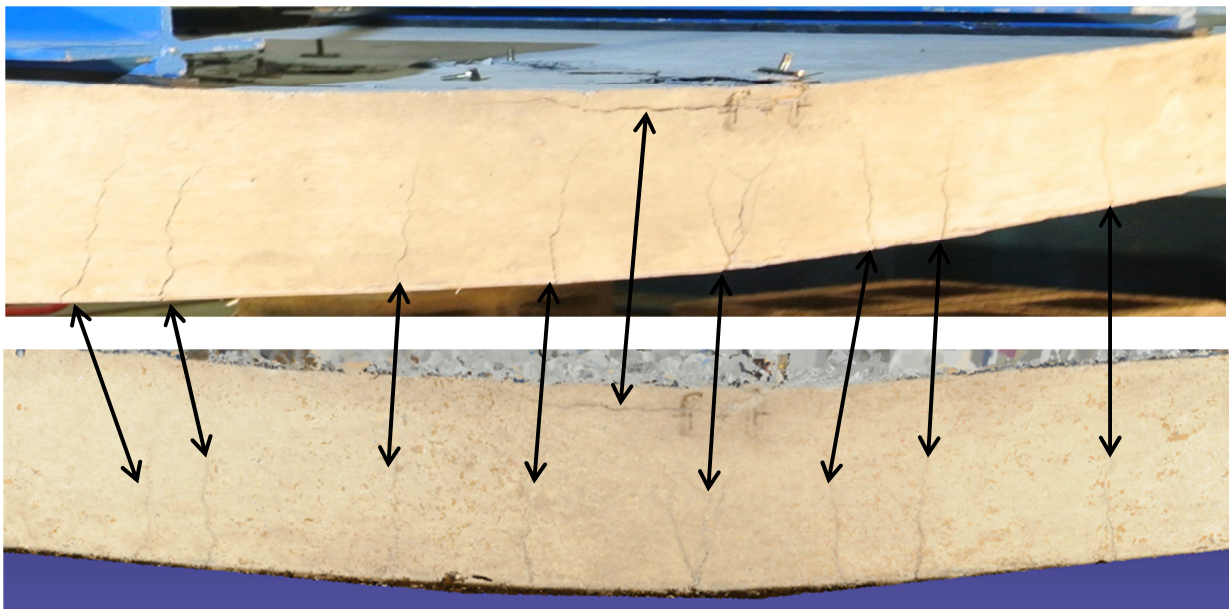


Figure 8: Preliminary Comparison of Cracks between Photo and Generated Mesh

After performing the CDA, the cracks on the specimen were colored red if the algorithm recognized them as cracked, that is the angle $\alpha \geq \theta$. For this project, 15 degrees was used because it was the angle used by Torak et al. for their CDA. The figure below, Figure 9, demonstrates the locations of correctly detected cracks; Figure 10 is just a close-up view of showing the color coding for crack detection. Along with correctly detected cracks, the CDA may incorrectly detect cracks, which is shown in Figure 11. Cracks that were not detected through the CDA are shown in Figure 12.

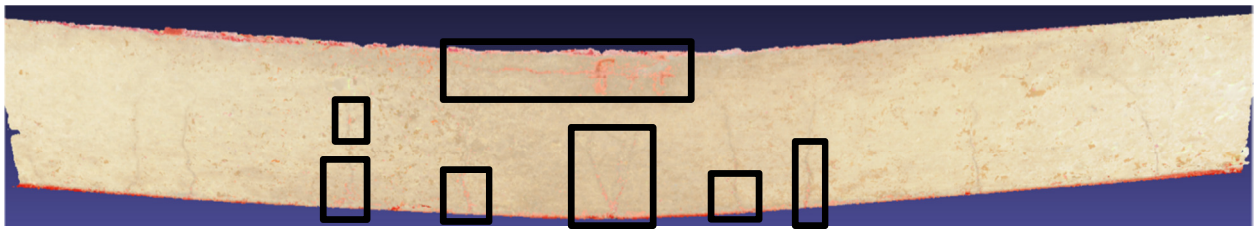


Figure 9: Results from CDA of Correctly Detected Cracks

Figure 10: Close-up of Crack Detection Color Coding

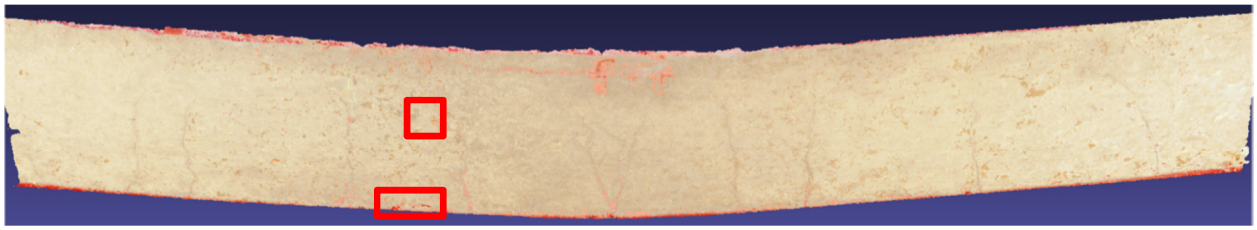


Figure 11: Results from CDA of Incorrectly Detected Cracks

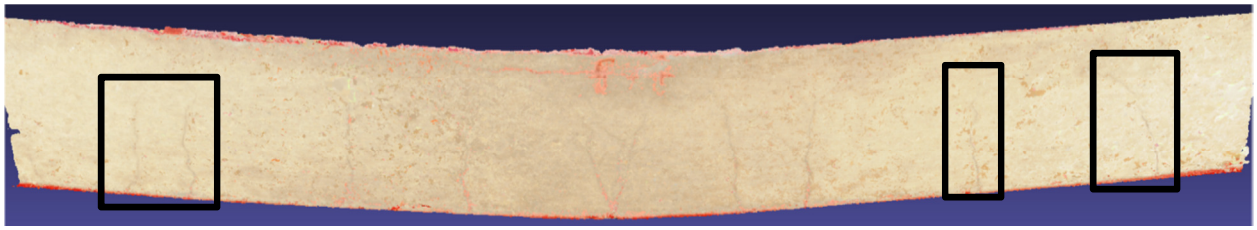


Figure 12: Results from CDA of Non-Detected Cracks

From the crack detection, the Precision and Accuracy code is run to see how well the code did. The results from that code are as follows: Precision = 75%, Accuracy = 60%. Based on these results, the crack detection algorithm performed on a fair level. As stated above, the threshold angle was set as 15 degrees. Using a smaller threshold angle may result with better precision and accuracy results. A sensitivity analysis was performed to determine how the threshold angle affects the precision and accuracy results; Table 1 contains the results from the sensitivity analysis and Figure 13 is a plot of how precision and accuracy varies as the threshold angle changes.

Table 1: Dimension Comparison for Specimen T17 after Failure

Threshold Angle, θ (degree)	Precision (%)	Accuracy (%)
1	77	100
5	83	100
10	82	90
15	75	60
20	75	60
25	75	60

A larger crack sample would give a better indication of how precision and accuracy is affect by the threshold angle. The crack detection code should also be investigated further. However, though the automated crack detection only performed on a fair level, the mesh model can still be examined manually to determine where there are cracks.

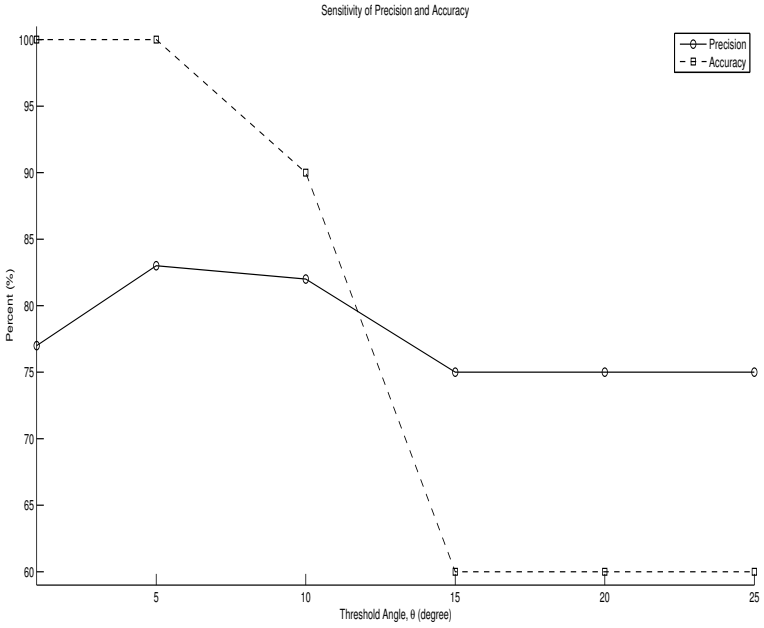


Figure 13: Sensitivity of Precision and Accuracy

Model Scaling

An important feature that can also be done with the three-dimensional point cloud is to scale it. Scaling the point cloud model allows the user to extract information about the specimen for assessment. For instance, during construction, an engineer can check the dimensions of a building element to ensure the contractors are constructing it correctly. In order to scale a point cloud model, reference points are required; these reference points must remain stationary, thus not be placed on elements that will change over time (i.e., slab specimen, vehicles, etc.).

For the Corrosion Resistant Reinforcing Bar project, reference points were placed on the test frame columns. As seen in Figure 14, the white squares on the test frame columns are the reference points that were placed. By having the dimensions between fixed reference points, a scaling code can be implemented into Matlab to transform the original .ply file that is in pixel space into a .ply file that is in millimeter space. The new space can be whatever units the user chooses (millimeters, meters, inches, feet) so long as the physical dimensions entered into the code is consistent.

Figure 14: Column Reference Points

Though there were reference points for the tests, due to computer issues, which will be explained later, the reference point locations were lost. This complication led to the implementation of another method to scale the model. Points within the point cloud were selected based on the correspondence to measured dimensions of the specimen. These points were then used in the scaling code to transform the point cloud model from pixel space into millimeter space. The tables below are the comparisons of various point cloud models that have been scaled using the new method to the known dimensions. Two different test specimens were scaled (specimens T16 and T17). Only the model after failure was generated for specimen T17, whereas all load increment models were available for T16. Based on the Table 2 to Table 8, it can be determined that the scaling method implemented performed well. The precision of the digital measurements was one millimeter because of how the model had to be viewed in order to make the measurement, which caused the point selection to not be as accurate.

Table 2: Dimension Comparison for Specimen T17 after Failure

Actual Slab Dimension (mm)	Digital Slab Dimension (mm)	Percent Error (%)
219	221	0.9
219	218	0.5
222	225	1.2
219	223	1.9
221	219	0.8

Table 3: Dimension Comparison for Specimen T16 before Loading

Actual Slab Dimension (mm)	Digital Slab Dimension (mm)	Percent Error (%)
222	223	0.4
219	222	1.4
221	224	1.4
222	224	0.9
219	216	1.4

Table 4: Dimension Comparison for Specimen T16 at 20 kip-ft

Actual Slab Dimension (mm)	Digital Slab Dimension (mm)	Percent Error (%)
222	223	0.4
219	224	2.3
221	223	0.9
222	224	0.1
219	223	1.8

Table 5: Dimension Comparison for Specimen T16 at 40 kip-ft

Actual Slab Dimension (mm)	Digital Slab Dimension (mm)	Percent Error (%)
219	218	0.5
219	215	1.8
222	223	0.4
219	223	1.8

Table 6: Dimension Comparison for Specimen T16 at 50 kip-ft

Actual Slab Dimension (mm)	Digital Slab Dimension (mm)	Percent Error (%)
222	214	3.6
219	221	0.9
221	220	0.4
222	222	0.0
219	218	0.5
222	220	0.9

Table 7: Dimension Comparison for Specimen T16 at 60 kip-ft

Actual Slab Dimension (mm)	Digital Slab Dimension (mm)	Percent Error (%)
222	223	0.4
918	911	0.8
219	219	0.0
221	218	1.4
219	220	0.5
219	218	0.5
222	219	1.4

Table 8: Dimension Comparison for Specimen T16 after Failure

Actual Slab Dimension (mm)	Digital Slab Dimension (mm)	Percent Error (%)
222	222	0.0
219	219	0.0
918	915	0.3
219	218	0.5
221	220	0.4
222	220	0.9
222	223	0.4
219	219	0.0
918	918	0.0

Equation 5, shown below, is implemented in the scaling code.

$$B(i) = s * R * A(i) + T \tag{5}$$

where *B* is the 3 x N scaled coordinates of the model

A is the 3 x N original coordinates of the model

N is the number of coordinates

i is the row number

s is the scale factor

R is the 3 x 3 rotation matrix

T is the 3 x 1 translation matrix

Crack Measurements

Based on the results of the dimension comparisons, the scaling procedure was successfully implemented. The scaled model can be used to retrieve crack width measurements, which are useful to inspectors, engineers, and researchers; utilizing non-contact digital crack measurements can reduce the risks that are involved with performing manual crack measurements. Figure 15 is an example of a dangerous situation during an experimental test while making crack width measurements on a specimen. Space that is available for a person to make crack width measurements can be limited, which would put the person in an

uncomfortable position. Also, when testing new products or materials, the behavior of the specimen may be unpredictable, especially at higher loads, so the risk of injury is increased.

Figure 15: Example of Potentially Dangerous Situation while Performing Crack Measurements

To obtain crack measurements from the scaled mesh model, a protocol was developed so the digital measurements could be consistent. The crack width measurements were done utilizing the measurement tool found in Meshlab. The steps to the crack measurement protocol are as follows:

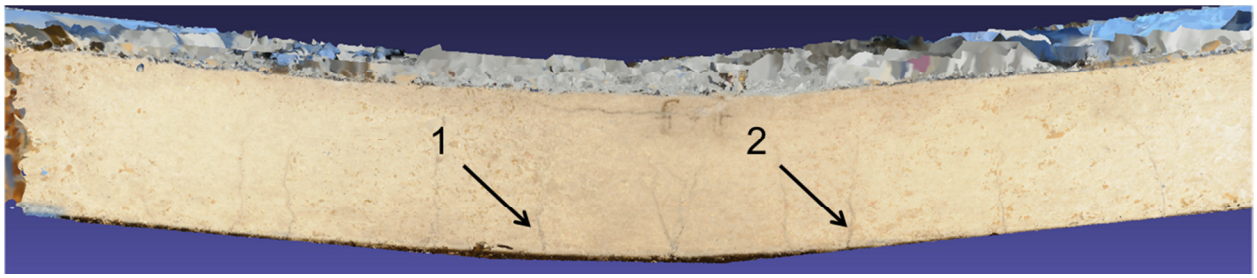
1) Locate crack containing crack width measurement data
2) Zoom in on crack
3) Determine point where crack measurement was made on specimen
4) Measure crack from color change to color change

Typically, some type of mark would indicate where a crack had measurements taken. However, there may be cases where the mark may not appear in the mesh model or a new crack measurement is of interest. If this is the case, the following modified crack measurement protocol should be followed:

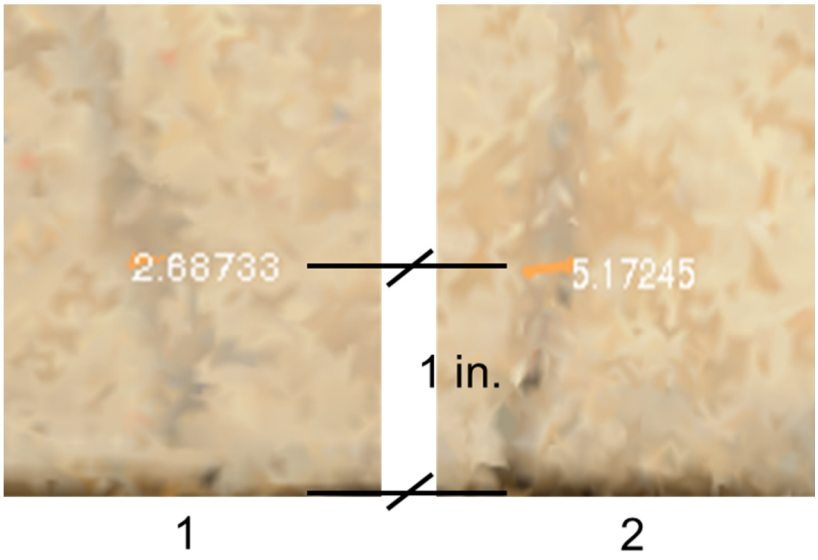
1) Determine crack of interest
2) Zoom in on crack
3) Determine location along crack where crack measurement should be made

4) Measure crack from color change to color change

The crack width measurement protocol is straight forward and easy to follow. The difficulty will be in determining where the edge of a crack is. The color change between the uncracked surface and a crack may not always be prominent. The subsequent figures are a demonstration of following the protocol to make crack measurements; Figure 16(a) shows the determination of cracks of interest and Figure 16(b) represents zooming into the specimen, locating where the crack measurements need to be made, and the results of making the crack measurements from color change to color change.



(a)



(b)

Figure 16: (a) Determining Crack of Interest, (b) Digital Crack Measurement

The results from the crack measurements utilizing the protocol and Meshlab are compared to the actual crack width measurements for the test specimen; these are shown in Table 9. Based

on the percent error of the crack measurement comparison, the digital crack measurements performed well; the precision of the digital crack measurements were 0.1 millimeters since the models were zoomed in so the point selection was more accurate.

Table 9: Crack Width Measurement Comparison

Crack Number	Actual Crack Measurement (mm)	Digital Crack Measurement (mm)	Percent Error (%)
1	2.98	2.7	9.4
2	5.48	5.2	5.1

Tracking Structural Behavior

The application of the crack detection procedure through the mesh generation portion allows for multiple mesh models of the same specimen to be produced at different loads to track the behavior of the structure. The process of examining pictures is simplified since the entire structure can be viewed in Meshlab and the user does not have to sift through numerous images, attempt to patch the information together in his or her mind, and keep track of the load value of the pictures. It can also be applied to structures under construction to maintain a record of the construction process as well as for examination for quality control and quality assurance. The program allows for closer inspection through the use of the zoom function. Furthermore, the storage of pictures can be more troublesome than storing the .ply files that contain the mesh models.

Shown in Figure 17 are the mesh models from one specimen at the different load increments during an experimental test. By examining these models, a person can become familiar with the behavior of the slab specimen without having to have been there for the test. It can be seen from the figures that the specimen does not go through much deformation until after the peak load is reached and the structure fails. The peak load for this specimen was 65 kip-ft, which is just greater than the last load increment before the slab failed. The tests

performed at Virginia Tech with corrosion resistant reinforcing bars used displacement control, so the ultimate load could be reached and the test could still continue until the specimens failed.

(a)

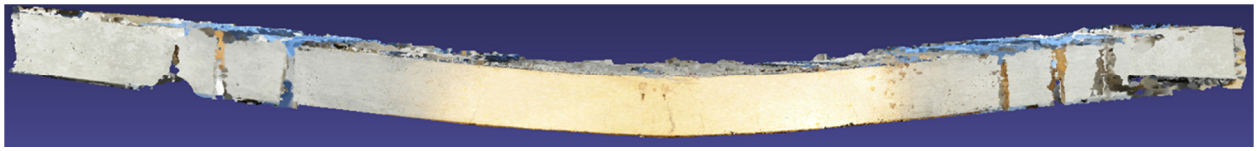
(b)

(c)

(d)

(e)

(f)



(g)

Figure 17: (a) Before Loading, (b) 20 kip-ft, (c) 30 kip-ft, (d) 40 kip-ft, (e) 50 kip-ft, (f) 60 kip-ft, (g) After Failure (Failure Load = 43 kip-ft)

SUMMARY AND CONCLUSIONS

This report presented the results of utilizing image-based reconstruction for crack detection and crack measurements. Along with crack detection and crack measurements, it has been shown that the technology can be applied to other applications such as tracking structural behavior. Also, the crack detection procedure was explained. The crack detection algorithm performed fair, which was checked using the Precision and Accuracy concept. The crack measurements obtained from the digital measurements were compared to the manual crack measurements taken during testing and compared; the percent difference between these crack widths were within 10%. Since the digital models were scaled to obtain crack measurements, the dimensions of the specimens were checked as well. The results of those comparisons were within 5%. Though the implementation of digital crack detection and measurement was successful, there are limitations to the crack detection and measurement process.

Limitations

Throughout the project, there were many limitations encountered, which will be explained in the following section. Some of these limitations include computational time, manual time, computer issues, lighting issues, and modeling issues. These limitations did have an effect on the crack detection process and crack measurement results.

The computational time to generate the point cloud models can be quite extensive. On average the total computational time for each model was approximately ten hours. To generate the 3D point cloud shown earlier in this report, a sparse point cloud has to be created in order to generate the dense point cloud shown. To obtain a refine point clout mode, the largest image size was used (3200 pixels). The size of the images affected the computational time along with the number of images that had to be processed through VisualSfM. The idea of quantity versus quality was introduced earlier in the report. Higher quality images may not be necessary if there is a large quantity of them, however, large quantity of pictures may not be needed if high quality images were taken. Ideally, a balance would be struck with precise and high quality images so

the optimal computational time is utilized and the point clouds reconstructed are complete – no missing portions of the structure. There was also computational time associated with creating the normals of the faces as well as generating a mesh model for each specimen at various load points was about an hour. The density of the mesh to ensure a greater quality model required more time to generate. Comparatively, the time to perform the Matlab codes was minimal.

The manual time required was intensive for the project. Trimming the point cloud models, trimming the mesh models, and scaling all required a large amount of computational time; the computational time associated with scaling was not anticipated. When a point cloud model is generated, a lot of excess points are created because of the congestion in the testing facility. Once the point cloud model is used to generate a mesh model, the mesh generation also produces points that are not part of the structure of interest; this is due to the mesh generation algorithm. As such, the mesh model must be trimmed. For both the point cloud model and the mesh model, the majority of the manual time is put into the precise reduction in the models. The scaling of the models required more time than originally expected. As described in this report, there were reference points placed on the testing frame columns. Unfortunately, the values for the location of these reference points were lost due to computer issues, so a new methodology to scale the models had to be used.

The computer issues involved with this research project include hard drive crashing of the computer containing the majority of the images from the experimental section, the lack of backup for the data on the computer, no available computer for a period of two to three months, and lack of available computers to generate multiple point cloud models at the same time. These computer problems caused setback in the amount of digital crack measurements that could have been made.

Another issue with the digital crack measurements involved the lighting of the slab specimens. Too much light washes out the specimens and made it difficult to see cracks. This can even be seen when examining the pictures taken of the slabs. Fortunately, there were

pictures that were backed up with proper lighting so digital crack measurements were able to be made. Unfortunately, due to model issues, not as many digital crack width measurements were able to be made. The point cloud models were missing portions due to the lack of images of certain regions of the slab, thus VisualSfM could not reconstruct the section. Other modeling issues encountered involve multiple models being constructed of the same specimen at the same load point. This is caused by the program VisualSfM not able to connect some images to others with the feature points. Though Meshlab has the ability to align models, it is difficult to utilize with the shape of the slab and the color of the models to align; Figure 18 below is a representation of this.

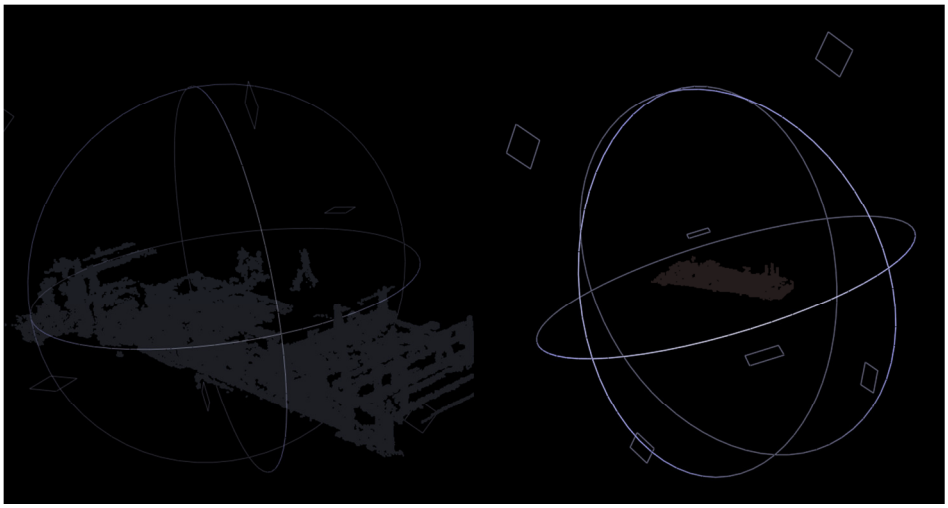


Figure 18: Model Alignment Difficulties in Meshlab

Future Work

To improve on the crack detection process along with the crack measurement process, the limitations must be addressed. Also, the crack detection algorithm should be examined more carefully to ensure it performs well; a larger sample set should be used next time for a better idea of the performance of the CDA. Furthermore, the future work should also involve checking to see if the VisualSfM program can patch missing sections of a specimen or structure from additional images. This was not able to be done during this project because the specimens were no longer available to capture more pictures for processing. Additional future work involves

optimizing the whole process so that the time required is reduced. Finally, incorporating automated image capturing is part of the future work to cut down on cost and to help improve the image capturing process.

Conclusion

The difficulties from the issues discussed above caused some setbacks. However, the crack detection algorithm performed on a fair level. The mesh models can still be examined manually to determine where there are cracks on the specimens. The crack measurements that were made compared well to the actual crack measurements and the scaling performed on models were accurate. The overall process took both computational and manual time. The refinement and incorporation of digital crack detection and digital crack measurement can help in fields such as quality control and quality assurance, field inspections, research and post disaster reconnaissance by increasing safety.

REFERENCES

Abdel-Qader, I., O. Abudayyeh, and M. Kelly, “Analysis of Edge-Detection Techniques for Crack Identification in Bridges,” *Journal of Computing in Civil Engineering*, 17.4: 255-263, 2003.

Barazzetti, L., and M. Scaioni, "Crack Measurement: Development, Testing and Applications of an Automatic Image-Based Algorithm," *ISPRS Journal of Photogrammetry and Remote Sensing*, 64.3: 285-296, 2009.

Benning, W., J. Lange, R. Schwermann, C. Effkemann, and S. Görtz, “Monitoring Crack Origin and Evolution at Concrete Elements using Photogrammetry,” *ISPRS Congress Istanbul Commission*, Vol. 2004, 2004.

Bowen, G., P. Zheng, C. Moen, and S. Sharp, “Service and Ultimate Limit State Flexural Behavior of One-Way Concrete Slabs Reinforced with Corrosion Resistant Reinforcing Bars,” *Transportation Research Board 92nd Annual Meeting*, No. 13-3314, 2013.

Jahanshahi, M. R., S. F. Masri, C. W. Padgett, and G. S. Sukhatme, "An Innovative Methodology for Detection and Quantification of Cracks through Incorporation of Depth Perception," *Machine Vision and Applications*, 24.2: 227-241, 2011.

Jahanshahi, M. R., and S. F. Masri, "Adaptive Vision-Based Crack Detection using 3D Scene Reconstruction for Condition Assessment of Structures," *Automation in Construction* 22: 567-576, 2012.

Subirats, P., J. Dumoulin, V. Legeay, and D. Barba, "Automation of Pavement Surface Crack Detection using the Continuous Wavelet Transform," *Image Processing, 2006 IEEE International Conference,* IEEE, pp. 3037-3040, 2006.

Tang, P., D. Huber, B. Akinci, R. Lipman, and A. Lytle, "Automatic Reconstruction of as-built Building Information Models from Laser-Scanned Point Clouds: A Review of Related Techniques,“ *Automation in Construction* 19.7: 829-843, 2010.

Torok, M., M. Golparvar-Fard, and K. Kochersberger, “Post-Disaster Robotic Building Assessment: Automated 3D Crack Detection from Image-Based Reconstructions,” *Computing in Civil Engineering*, ASCE, 2012.

Zhu, Z., S. German, and I. Brilakis, “Detection of Large-Scale Concrete Columns for Automated Bridge Inspection,” *Automation in Construction* 19.8: 1047-1055, 2010.

Zhu, Z, S. German, and I. Brilakis, "Visual Retrieval of Concrete Crack Properties for Automated Post-Earthquake Structural Safety Evaluation," *Automation in Construction* 20.7: 874-883, 2011a.

Zhu, Z, S. German, S. Roberts, I. Brilakis, and R. DesRoches, "Machine Vision Enhanced Post-earthquake Inspection,“ *Computing in Civil Engineering,* 152, 2011b.

APPENDIX 1

Guide to Crack Detection

1) Image Capturing

    a. The pictures should be taken with a schematic plan, that is there should be a planned sequence of how the images are captured rather than them random taken. A schematic approach to taking the pictures would help the program VisualSfM match features better and not generate multiple models or having missing portions in the specimen.

    b. The size of the image should be set to a maximum of 3200 since that is the maximum size VisualSfM can handle without altering the picture size internally. This helps ensure a good quality image for feature extraction.

    c. If the pictures need to be resized, the program IrfanView can be utilized to readjust the images in bulk.

    d. When taking the pictures, make sure to review the images so that the quality of the images is sustained (i.e., no blurry pictures).

    e. Review the images to make sure the lighting of the specimen is correct (i.e., the cracks are visible and not washed out).

2) VisualSfM – Point Cloud Generation

    (Note: These instructions are given for a Linux operating system. There are two ways to use VisualSfM, one is through the graphical user interface and the other is through the command window – cmd. More help and detail can be found on the following website: http://homes.cs.washington.edu/~ccwu/vsfm/)

    a. To operate VisualSfM with the command window, the location of the program must be known.

b. Once this is determined, the folder in which the images that need to be processed can be located by using the 'cd' command. An example of how to use 'cd' is demonstrated below.

    i. raamacuser@Mehrdad:~$ cd Desktop/Paul_Crack-Detection/

        1. This will change the folder from the home folder to the folder 'Paul_Crack-Detection' which is located on the Desktop.

c. When the proper folder is reached, the command to run a sparse point cloud model can be executed. The command is shown below.

    i. raamacuser@Mehrdad:~/Desktop/Paul_Crack-Detection/7-15-12_originals/7-15-12_Failure$ /home/raamacuser/reconstruction_files/VSFM/bin/VisualSfM sfm ./ [Enter name for file].nvm

        1. The portion in blue shows the location of where the images are, while the portion in green shows the folder in which the VisualSfM is located. The selections in red are the input parameters.

d. To run the dense reconstruction when the sparse is complete, the following command should be implemented.

    i. raamacuser@Mehrdad:~/Desktop/Paul_Crack-Detection/7-15-12_originals/7-15-12_Failure$ /home/raamacuser/reconstruction_files/VSFM/bin/VisualSfM sfm+resume+cmvs+pmvs ./ [Sparse reconstruction file name].nvm [File name for dense reconstruction file].nvm

        1. The portion in blue shows the location of where the images are, while the portion in green shows the folder in which the VisualSfM is located. The selections in red are the input parameters.

e. The output from VisualSfM when the dense reconstruction is complete should be a .ply file. This can be opened in Meshlab.

3) Meshlab – Point Cloud Viewer and Mesh Generation

a. Opening and viewing a .ply file is the same as other programs. When opening a new .ply file, the light bulb button should be turned off.

b. After opening a file, the .ply file will need to be trimmed. To do so, click the 'Select Vertexes' button and highlight the unwanted points. When the points are selected, hit ctrl+delete to delete the points. Repeat until left with the desired specimen. Also, to continuously select points, hold the ctrl button while selecting.

c. When the desired specimen is left after trimming, the normals need to be created. Go to Filters → Normals, Curvatures and Orientation → Compute normals for point sets

d. Now the mesh model can be generated. Go to Filters → Remeshing, simplification and reconstruction → Surface Reconstruction: Poisson

e. After the mesh model is created, trimming will need to be performed again. This process is dealt with the same way as when trimming the point cloud model.

f. Now the colors from the point cloud model can be transferred onto the mesh model. This is done by going to Filters → Sampling → Vertex Attribute Transfer

4) Crack Detection Algorithm

a. The crack detection code, which can be seen in Appendix 2, can be applied to the .ply file that contains the mesh model.

b. The new file that is generated can be viewed in Meshlab to see how the crack detection algorithm did.

5) Precision and Accuracy

a. The Precision and Accuracy code can be seen in Appendix 2. This code requires the user to input the number of correctly identified cracks, the number of incorrectly detected cracks, and the number of actual cracks that were not identified to quantify how the CDA performed.

APPENDIX 2

Scaling Codes

```
% Apply_sRT.m
% Scaling Code
% by Zheng, Paul

clear all
close all
clc

%% Load Point Cloud
fprintf('Loading Points\n')
[all_pts,h] = loadPlyPntCloud('7-15-12_30kips_reduced.ply');
x_pts = all_pts(:,1);
y_pts = all_pts(:,2);
z_pts = all_pts(:,3);
r = all_pts(:,4);
g = all_pts(:,5);
b = all_pts(:,6);
pts = [x_pts y_pts z_pts];
color = [r g b];

%% Obtain s, R, T, and error
fprintf('Finding s,R,T\n')

% Pixel Space
AA = [23.9527 0.389512 10.5789;
    24.1123 1.48595 10.724;
    22.6277 0.0970566 15.1746;
    22.7896 1.19199 15.3419;];

% mm Space
BB = [0 0 0;
    0 -222.25 0;
    917.575 0 0;
    917.575 -219.075 0;];

% Find s,R,T in pixel space
[s R T error] = absoluteOrientationQuaternion( AA', BB', 1);

% Find s,R,T in mm space
[s2 R2 T2 error2] = absoluteOrientationQuaternion( BB', AA', 1);

%% Transform Point Cloud

fprintf('Transforming Points\n')

pts2 = zeros(length(pts),3);
for i = 1:length(pts)
    pts2(i,:) = (s*R*(pts(i,:)') + T)';
end
```

```
fprintf('Compiling Point Set\n')

new_set = horzcat(pts2,color);

%% Write new ply file with transformed point cloud
fprintf('Writing Ply File\n')

fid = fopen('7-15-12_30kips_scaled_S.ply','w');
fprintf(fid,'ply\r\n');
fprintf(fid,'format ascii 1.0\r\n');
fprintf(fid,['element vertex ' num2str(length(new_set)) '\r\n']);
fprintf(fid,'property float x\r\n');
fprintf(fid,'property float y\r\n');
fprintf(fid,'property float z\r\n');
fprintf(fid,'property uchar red\r\n');
fprintf(fid,'property uchar green\r\n');
fprintf(fid,'property uchar blue\r\n');
fprintf(fid,'end_header\r\n');
for i = 1:length(new_set)
    fprintf(fid,'%f %f %f %d %d %d\r\n', new_set(i,:));
end
fclose(fid);

fprintf('error = %f\n',error)
fprintf('error2 = %f\n',error2)


% loadPlyPntCloud.m
% Zheng, Paul

function [d,h]=loadPlyPntCloud(fn)
% [d,h]=loadPlyPntCloud(fn)
%
% load .ply formatted point cloud
% d = data, Nx6 [x y z r g b]
% h = header
if ~exist('fn','var')
    [fn,pn,index] =uigetfile('C:/*.ply', 'Please choose a .ply file');
    fn=[pn fn];
end

% read file
fid = fopen(fn,'r');
inc=1;
h={};
while inc~=0
    h = textscan(fid, '%s',1,'delimiter','\r\n');
    if strcmp(h{1}(1),'end_header')||inc>100
        inc=0;
    else
        inc=inc+1;
    end
    h(end+1)=h{1};
end
d = textscan(fid, '%f %f %f %f %f %f %f');
```

```
fclose(fid);
d = [d{1} d{2} d{3} single(d{4}) single(d{5}) single(d{6})];


% absoluteQrientationQuaternion.m
% Zheng, Paul

% [s R T error] = absoluteOrientationQuaternion( A, B, doScale)
%
% Computes the orientation and position (and optionally the uniform scale
% factor) for the transformation between two corresponding 3D point sets Ai
% and Bi such as they are related by:
%
%     Bi = sR*Ai+T
%
% Implementation is based on the paper by Berthold K.P. Horn:
% "Closed-from solution of absolute orientation using unit quaternions"
% The paper can be downloaded here:
% http://people.csail.mit.edu/bkph/papers/Absolute_Orientation.pdf
%
% Authors:      Dr. Christian Wengert, Dr. Gerald Bianchi
% Copyright:    ETH Zurich, Computer Vision Laboratory, Switzerland
%
% Parameters:   A           3xN matrix representing the N 3D points
%               B           3xN matrix representing the N 3D points
%               doScale     Flag indicating whether to estimate the
%                           uniform scale factor as well [default=0]
%
% Return:       s           The scale factor
%               R           The 3x3 rotation matrix
%               T           The 3x1 translation vector
%               err         Residual error
%
% Notes: Minimum 3D point number is N > 4
function [s R T err] = absoluteOrientationQuaternion( A, B, doScale)

%Argument check
if(nargin<3)
    doScale=1;
end
%Return argument check
if(nargout<1)
    usage()
    error('Specify at least 1 return arguments.');
end
%Test size of point sets
[c1 r1] = size(A);
[c2 r2] = size(B);
if(r1~=r2)
    usage()
    error('Point sets need to have same size.');
end
if(c1~=3 | c2~=3)
    usage()
    error('Need points of dimension 3');
end
```

```
if(r1<4)
    usage()
    error('Need at least 4 point pairs');
end

%Number of points
Na = r1;

%Compute the centroid of each point set
Ca = mean(A,2);
Cb = mean(B,2);

%Remove the centroid
An = A - repmat(Ca,1,Na);
Bn = B - repmat(Cb,1,Na);

%Compute the quaternions
M = zeros(4,4);
for i=1:Na
    %Shortcuts
    a = [0;An(:,i)];
    b = [0;Bn(:,i)];
    %Crossproducts
    Ma = [  a(1) -a(2) -a(3) -a(4) ;
            a(2)  a(1)  a(4) -a(3) ;
            a(3) -a(4)  a(1)  a(2) ;
            a(4)  a(3) -a(2)  a(1)  ];
    Mb = [  b(1) -b(2) -b(3) -b(4) ;
            b(2)  b(1) -b(4)  b(3) ;
            b(3)  b(4)  b(1) -b(2) ;
            b(4) -b(3)  b(2)  b(1)  ];
    %Add up
    M = M + Ma'*Mb;
end

%Compute eigenvalues
[E D] = eig(M);

%Compute the rotation matrix
e = E(:,4);
M1 = [  e(1) -e(2) -e(3) -e(4) ;
        e(2)  e(1)  e(4) -e(3) ;
        e(3) -e(4)  e(1)  e(2) ;
        e(4)  e(3) -e(2)  e(1)  ];
M2 = [  e(1) -e(2) -e(3) -e(4) ;
        e(2)  e(1) -e(4)  e(3) ;
        e(3)  e(4)  e(1) -e(2) ;
        e(4) -e(3)  e(2)  e(1)  ];


R = M1'*M2;

%Retrieve the 3x3 rotation matrix
R = R(2:4,2:4);
%Compute the scale factor if necessary
```

```
if(doScale)
    a =0; b=0;
    for i=1:Na
        a = a + Bn(:,i)'*R*An(:,i);
        b = b + Bn(:,i)'*Bn(:,i);
    end
    s = b/a;
else
    s = 1;
end

%Compute the final translation
T = Cb - s*R*Ca;

%Compute the residual error
if(nargout>3)
    err =0;
    for i=1:Na
        d = (B(:,i) - (s*R*A(:,i) + T));
        err = err + d'*d;
    end
    err = sqrt(err)/Na;
end

%Displayed if an error occurs
function usage()
disp('Usage:')
disp('[s R T error] = absoluteOrientationQuaternion( A, B, doScale)')
disp(' ')
disp('Return values:')
disp('s         The scale factor')
disp('R         The 3x3 rotation matrix')
disp('T         The 3x1 translation vector')
disp('err       Residual error (optional)')
disp(' ')
disp('Input arguments:')
disp('A         3xN matrix representing the N 3D points')
disp('B         3xN matrix representing the N 3D points')
disp('doScale   Optional flag indicating whether to estimate the uniform
scale factor as well [default=0]')
disp(' ')
```

Crack Detection Codes

```
% CrackFinder.m
% Check for cracks with poisson
% Zheng, Paul

clear all
close all
clc
```

```
[all_pts,h,vertices,faces] = loadPlyPoisson('7-18-
12_fail_reduced_scaled2_mesh2.ply');

% read vertices, then faces
% ignore color data
for i = 1:vertices
    vertex(i).x = all_pts(i,1);
    vertex(i).y = all_pts(i,2);
    vertex(i).z = all_pts(i,3);
end
% vertex format is 3 Pt1 Pt2 Pt3
for i = 1:faces
    face(i).pt1 = all_pts(i+vertices,2);
    face(i).pt2 = all_pts(i+vertices,3);
    face(i).pt3 = all_pts(i+vertices,4);
end
fprintf('\nPoints loaded')
%% Now go through and look for tilted mesh triangles
angle = zeros(length(faces));
for i = 1:faces
    A = [vertex(face(i).pt1+1).x, vertex(face(i).pt1+1).y,
vertex(face(i).pt1+1).z];
    B = [vertex(face(i).pt2+1).x, vertex(face(i).pt2+1).y,
vertex(face(i).pt2+1).z];
    C = [vertex(face(i).pt3+1).x, vertex(face(i).pt3+1).y,
vertex(face(i).pt3+1).z];

    AB = B-A;
    AC = C-A;
    normplane = [1 0 0]; % normal vector defining y-z plane
    normvec = cross(AB, AC);
    angle(i) =
asind(dot(normplane,normvec)/(sqrt(normvec(1)^2+normvec(2)^2+normvec(3)^2)));

    % mark as crack
    if abs(angle(i))>15
        % ADDED: COLOR DETECTION... DON'T CHANGE WHITE (ARTIFICAL) VERTICES
        if (all_pts(face(i).pt1+1,7) ~= 255 && all_pts(face(i).pt1+1,8) ~=
255 && all_pts(face(i).pt1+1,9) ~= 255 && ...
                all_pts(face(i).pt2+1,7) ~= 255 && all_pts(face(i).pt2+1,8)
~= 255 && all_pts(face(i).pt2+1,9) ~= 255 && ...
                all_pts(face(i).pt3+1,7) ~= 255 && all_pts(face(i).pt3+1,8)
~= 255 && all_pts(face(i).pt3+1,9) ~= 255)
            % change color to red
            if all_pts(face(i).pt1+1,7) < 200
                all_pts(face(i).pt1+1,7) = all_pts(face(i).pt1+1,7) + 55;
            end
            if all_pts(face(i).pt2+1,7) < 200
                all_pts(face(i).pt2+1,7) = all_pts(face(i).pt2+1,7) + 55;
            end
            if all_pts(face(i).pt3+1,7) < 200
                all_pts(face(i).pt3+1,7) = all_pts(face(i).pt3+1,7) + 55;
            end
        end
    else
        if all_pts(face(i).pt1+1,8) < 250
```

```
            all_pts(face(i).pt1+1,8) = all_pts(face(i).pt1+1,8) + 5;
        end
        if all_pts(face(i).pt2+1,8) < 250
            all_pts(face(i).pt2+1,8) = all_pts(face(i).pt2+1,8) + 5;
        end
        if all_pts(face(i).pt3+1,8) < 250
            all_pts(face(i).pt3+1,8) = all_pts(face(i).pt3+1,8) + 5;
        end
    end
end

fprintf('\nWriting file')
% write output file
writePlyPoisson(all_pts,vertex,face,'7-18-
12_fail_reduced_scaled2_mesh2_crack.ply')

fprintf('\nDone\n')


% loadPlyPoisson.m
% Zheng, Paul

function [d,h,num_vert,num_face]=loadPlyPoisson(fn)
% INCLUDES ALPHA ACCOMIDATION (for MeshLab)
% [d,h]=loadPlyPntCloud(fn)
%
% load .ply formatted point cloud
% d = data, Nx6 [x y z r g b]
% h = headera
if ~exist('fn','var')
    [fn,pn,index] =uigetfile('C:/*.ply', 'Please choose a .ply file');
    fn=[pn fn];
end

% read file
fid = fopen(fn,'r');
inc=1;
h={};
while inc~=0
    h = textscan(fid, '%s',1,'delimiter','\r\n');
    % find number of vertices
    if findstr(char(h{1}(1)),'element vertex')
        linenum = inc;
        vert_str = char(h{1}(1));
        vert_str(16:length(vert_str));
        num_vert = str2num(vert_str(16:length(vert_str)));
    end
    % find number of faces
    if findstr(char(h{1}(1)),'element face')
        linenum = inc;
        face_str = char(h{1}(1));
        face_str(14:length(face_str));
        num_face = str2num(face_str(14:length(face_str)));
    end
    if strcmp(h{1}(1),'end_header')||inc>100
        inc=0;
```

```
    else
        inc=inc+1;
    end
    h(end+1)=h{1};
end
d = textscan(fid, '%f %f %f %f %f %f %f %f %f %f %f');
fclose(fid);
d = [d{1} d{2} d{3} single(d{4}) single(d{5}) single(d{6}) ...
    single(d{7}) single(d{8}) single(d{9}) single(d{10}) single(d{11})];


% writePlyPoisson.m
% Zheng, Paul

function writePlyPoisson(d,vertex, face, fn)
% writePlyPntCloud(d,fn)
% d is a Nx6 point cloud with [x y z r g b]
if ~exist('fn','var')
    fn='newPointCloud';
end
fn=strrep(fn,'.ply','');
fn2=[fn '.ply'];

% Write to ply file
fid2 = fopen(fn2,'w+');
fprintf(fid2,'ply\r\n');
fprintf(fid2,'format ascii 1.0\r\n');
fprintf(fid2,['element vertex ' num2str(length(vertex)) '\r\n']);
fprintf(fid2,'property float x\r\n');
fprintf(fid2,'property float y\r\n');
fprintf(fid2,'property float z\r\n');
fprintf(fid2,'property float nx\r\n');
fprintf(fid2,'property float ny\r\n');
fprintf(fid2,'property float nz\r\n');
fprintf(fid2,'property uchar diffuse_red\r\n');
fprintf(fid2,'property uchar diffuse_green\r\n');
fprintf(fid2,'property uchar diffuse_blue\r\n');
fprintf(fid2,'property uchar alpha\r\n');
fprintf(fid2,['element face ' num2str(length(face)) '\r\n']);
fprintf(fid2,'property list uchar int vertex_indices\r\n');
fprintf(fid2,'end_header\r\n');

for i=1:length(vertex)
    fprintf(fid2,'%f %f %f ',vertex(i).x, vertex(i).y, vertex(i).z);
    fprintf(fid2,'%f %f %f ',d(i,4:6));
    fprintf(fid2,'%d %d %d %d \r\n',d(i,7:10));
end
for i=1:length(face)
    if i ~= length(face)
        fprintf(fid2,'3 %d %d %d \r\n',face(i).pt1, face(i).pt2, face(i).pt3);
    else
        fprintf(fid2,'3 %d %d %d',face(i).pt1, face(i).pt2, face(i).pt3);
    end
end
fclose(fid2);
```

```
% PrecisionRecall.m
% Precision and Recall Code
% Zheng, Paul

clear all
close all
clc

%% Input
TP = 28; %TP = number of correctly detected cracks
FP = 6; %FP = number of incorrectly detected cracks
FN = 3; %FN = number of actual cracks that are not detected

filename = 'Results.txt';

%% Precision - Measure of how many detected cracks are correctly identified
Precision = TP/(TP+FP)*100; %(TP+FP) = total number of detected cracks

%% Recall - Measure of how many actual cracks are correctly identified
Recall = TP/(TP+FN)*100; %(TP+FN) = total number of actual cracks

%% Output
fid = fopen(filename,'w');
fprintf(fid,'Precision is a measure of how many detected cracks are ');
fprintf(fid,'correctly identified\n');
fprintf(fid,'Precision = %0.2f%%\n',Precision);
fprintf(fid,'Recall is a measure of how many actual cracks are correctly ');
fprintf(fid,'identified\n');
fprintf(fid,'Recall = %0.2f%%\n',Recall);
fclose(fid);
```