Visual and automated crack detection in metal samples

J. Betran Menz, A. Brugnera, V.J.A. Houben, S. Joseph, S. Yao

I. Abstract

This study presents an innovative approach to automating the detection and measurement of crack growth in metal specimens under controlled conditions, using an Artificial Intelligence (AI) system. Current methods for crack detection and estimation in steel specimens are time-consuming, subjective, and miss out on certain details of the crack growth. Importantly, these methods primarily quantify the crack length as a single numerical value, neglecting the complex nature of crack propagation, which doesn't always follow a linear path.

Addressing these issues, this paper presents a efficient and reliable system using AI that is designed for this task. The project comprises of five distinct stages: Data Collection, Data Labeling, Neural Network Implementation, Curve Measurement, and Image Acquisition. Custom data set was gathered and labeled in order to train the neural network, for which three different architectures and ideas were explored, from which Unet with VGG backbone showed the best results. The trained network was employed to detect cracks and estimate their growth. Furthermore, for the image acquisition process a physical system had to be designed and developed that was then optimized to capture the instances of maximum crack visibility. The successful implementation of this system serves a indicator for future exploration and innovation in the application of AI within material science and engineering. This study serves as a foundational step towards a more automated and accurate approach to material testing. Relevant code and documentation can be found on: https://github.com/a-brugnera-tue/5ARIP0

II. Introduction

Research is critical for understanding how cracks evolve in metal, and to investigate crack development in metal specimens under controlled conditions. This can be conducted by means of fatigue tests in which cyclic load is applied. To gain information, it is necessary to measure the crack length during the tests. Currently, there are no widely agreed methods to determine the crack length in a metal specimen in an easy-to-use and automated way. The prevailing methods require manual execution or only estimate the crack length. One method, for example, measures the electric current through a sensor that is placed where the crack is expected to develop. The crack growth causes the wires inside the sensor (crack gauge) to break and thus decrease the current. Another method is using a clip gage that measures the crack opening which is related to the crack length. By measuring this value, an estimation of the crack length can be found. This method is explained in more detail in section IV. Although other research has already been conducted in the implementation of AI systems to determine cracks in metal structures [1]–[3], to the best of the authors knowledge none has yet been for controlled crack growth in specimen. The goal of the project described in this report, is to develop an automated and visual approach to determine the crack length in metal specimen. This approach deploys an Artificial Intelligence (AI) network that is trained for this specific task.

This project can be split into five semi-separate elements: the Data collection, the Data labeling, the Neural network, the Curve measurement, and the Image acquisition. All five elements are explained in section III and the system as a whole is validated in section IV. The data collection section explains how custom data for training the AI system was necessary and how the data was collected. For the data labeling, two different techniques for labeling and their results are shown and discussed. In the subsection about the neural network, three different neural networks are explored and the results from one of them are given and discussed. The curve measurement subsection explains two methods to determine the length of the crack as it was predicted by the neural network. For the image acquisition, a method to take images of a specimen has been developed and hardware and accompanying software has been built. This method provides potential users with a way to implement the trained AI system. Afterwards, the performance of the AI system is validated in section IV, using multiple methods to show the performance and robustness of the system. Lastly, the results of the project are summarized and future recommendations are given in section V.

III. Methodology

The process of creating a tool that can be used to automatically detect cracks in steel specimens using visual methods can be divided in five main parts. These are data collection, image acquisition, data labelling, the neural network and curve measurement. The different considered methods are discussed in this section, together with an explanation which methods have been used for the final product.

A. Data collection

Training an AI system to recognize cracks in metal specimen requires many images of such cracks. The first step in this project was collecting these images, as already existing data sets are insufficient. The images that were provided at the start of the project were of a low resolution with much glare, making it difficult to train on. Additionally, the images were not labeled, and labeling is quite a labor intensive task, as is explained in more detail in subsection III-B. To make sure that the images in our own data set were well suitable, they had to be taken such that the crack is clearly visible. Meaning that the cracks are wide enough and that the contrast between the crack and the steel is large. First, the setup that was used to create cracks in the specimen is explained. Next, the different methods to take images of the specimen that were explored are discussed.

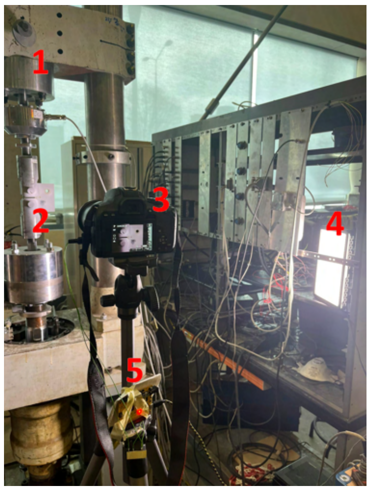


Fig. 1: Image showing data collection setup with the following main components. 1: Testing machine; 2: steel specimen; 3: camera; 4: flood lights; 5: Arduino Uno.

1) Experimental setup: The setup with the testing machine that was used, is shown in Figure 1. The testing machine is connected to a controller that can in turn be controlled by a desktop with MOOG software. The software is used to send a signal to pull or push with a certain amount, in this case only pulling on the specimen was needed. The camera is used to take the images, when triggered by the Arduino Uno that is connected to it. The flood light ensures that high-quality images can be taken under varying light conditions, without varying results.

In Figure 2 the dimensions of the steel specimen that was used to acquire the majority of the data is shown.

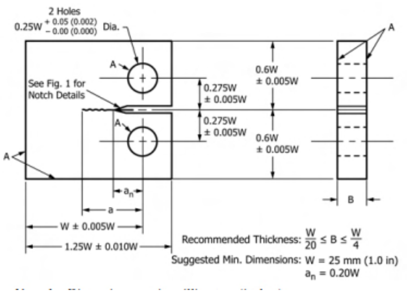

2 Holes
0.25W +0.05 (0.002) −0.00 (0.000) Dia.
A
See Fig. 1 for Notch Details
0.275W ± 0.005W
0.275W ± 0.005W
0.6W ± 0.005W
0.6W ± 0.005W
A
A
A
a_n
a
W ± 0.005W
1.25W ± 0.010W
B
Recommended Thickness: W/20 ≤ B ≤ W/4
Suggested Min. Dimensions: W = 25 mm (1.0 in)
a_n = 0.20W

Fig. 2: Standard dimensions of the test specimen used.

Such a specimen can be used for fatigue crack growth experiments, using cyclic loading. It is notched beforehand to ensure the point from which crack growth starts. The clevis and pin assembly is used to keep the specimen in place during testing. To ensure a large contrast between crack and specimen, the specimen is polished and sprayed with anti-glare spray [4]. This also reduces the risk of reflection in the image.

2) Timing of images: In order to create data that can be used to effectively train the neural network later-on, the moment of taking the images must be regulated. An important aspect is that the crack must be visible on the images. When unloaded, the crack closes (partially), making it very difficult to determine the size. Two ways to overcome this issue have been considered and will now be explained. The first technique is to read the force output from the controller in real-time and take the images at peak values, while cycling with sinusoidal varying loads. Determining the peak value, or a range of peak values, would have to be done in real-time. With the force being applied as a sine wave, a typical output signal can be seen in Figure 3. The voltage is a measure for the applied force, with 10 V corresponding to 250 kN. The output was read with the the aid of a NI-6002 DAQ device, connected to a separate laptop.

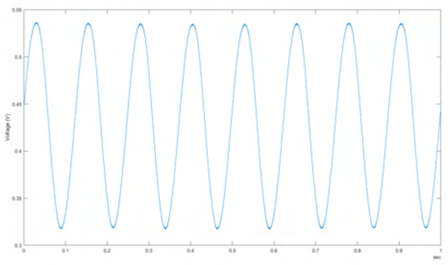


Fig. 3: A graph representing a typical signal output of a frequency of 8 Hz with a discrete data acquisition rate of 50,000 Hz.

This means that the frequency of the cycling depends on the speed of transmitting information in the setup. When implemented correctly, with a reasonably fast setup, this would be ideal as there are no interruptions to the cycling of the load.

The second technique letting the input plateau at certain values, shown in 4. During these plateaus at different values, images can be taken. This would allow for taking more time to take the images, meaning that it is more likely that the image is taken at the right moment. At the time of collecting the data, the logbooks created by the MOOG software were used to send a signal to trigger the cameras. A more sophisticated method has later been implemented and is described in Figure III-E.

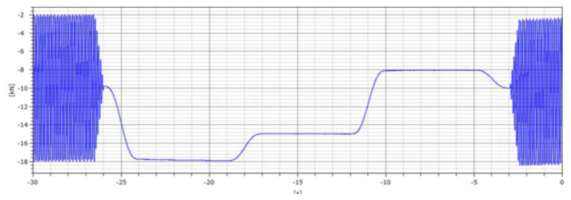


Fig. 4: A graph representing a segment of the force cycle applied to the specimen, using the plateau technique.

A disadvantage of this method is that the cyclic loading of the specimen has to be interrupted to create plateaus. This could potentially disrupt the crack growth pattern, forming an issue especially when a strict protocol must be followed.

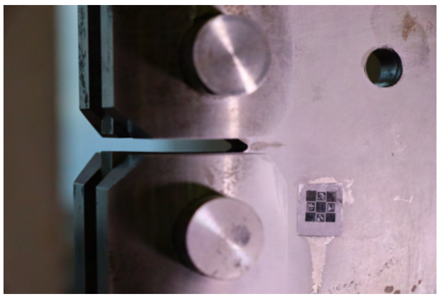


Fig. 5: Image of specimen tested under load showing the placement of the ChArUco diamond marker

3) Triggering the camera: The cameras are triggered to take pictures by the Arduino Uno. The exact code and an explanation for how this is done, can be read in section A. This means that no human interaction is needed for the data collection. What is important to note, however, is that for the AI system to accurately determine the real length of the crack, it requires a reference to relate the pixel ratio to actual distance. Without this reference, the AI would be unable to determine the actual size of the crack. Often, objects with known dimensions are used for this purpose. ARUCO markers are stickers that have been developed for this purpose and contain much detail to ensure their precision. ARUCO is an open-source library for marker detection, based on the use of 2D barcodes. It allows for the identification of objects in an image marker, which is a small, unique code that can be printed and placed on objects. This can be seen in Figure.5 which shows the placement of the ARUCO marker which was printed out to be exactly 10mm x 10mm and then glued to the specimen (away from where the crack is expected to grow). It is important to try and place the Aruco marker as straight as possible, more specifically to be aligned with the specimen to allow for the marker to work best. How they are exactly used to translate pixel to a metric system distance, is explained in subsubsection III-D3.

B. Data labelling

Accurate image labeling is very important for effective neural network training, particularly in tasks like pixel-wise segmentation. During annotation, a segmented mask is created with values of 0 for non-crack pixels and 1 for crack pixels. Manual segmentation is commonly used for this task. However, it is a very time-consuming process, especially considering most images have more than 500k pixels.

1) Segmentation Anything Model: An automated approach can greatly expedite the process and reduce the time and effort required. Meta created a segmentation tool, Segmentation Anything Model [5]. The tool was not trained to segment a certain class, but instead to be able to label any sort of data. It was tested on images containing a crack in a steel specimen, the result of the segmentation is shown in Figure 21.

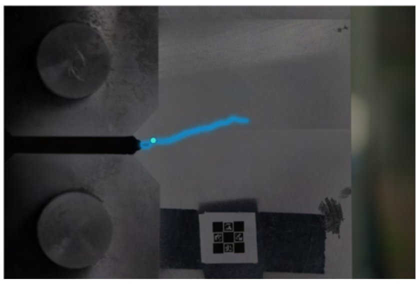


Fig. 6: Image showing the label created for a specimen overlapped on the original image using SAM tool

One significant advantage of this labeling tool is its ease of use. The model supports interactive operations, requiring only to click on some pixels that are part of the crack, which automatically generates labels for all crack pixels in the image. However, it is important to note that this tool may encounter challenges in detecting small items, which is rather crucial for this project. This is limiting its performance and thus for the complete AI system.

2) Crack Segmentation Tool: This is a semi-automatic segmentation tool developed by Andrii [6] that can generate a pixel-wise segmentation of the crack. This tool makes use of an image processing algorithm, which was originally designed for analyzing retinal images of vascular systems. The algorithm utilizes a multi-orientation wavelet transform to generate ”orientation scores”, which are then filtered to establish an optimal path for crack identification. By employing a cutting-edge geometric tracking method, the tool calculates the globally optimal path between manually selected crack endpoints.

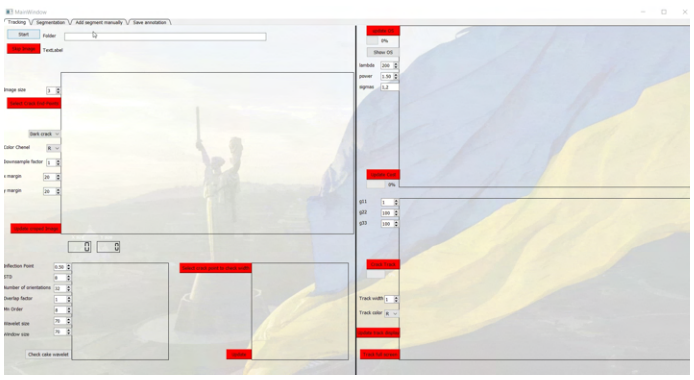


Fig. 7: Image showing the Segmentation Tool UI

Impressively, this tool performs exceptionally well even with small cracks and was hence chosen for this project. However, it requires the manual selection of crack endpoints. The image processing algorithm offers various adjustable parameters that can be modified through a user-friendly graphical interface 7. Unfortunately, this necessitates the laborious task of manually tuning the parameters for each individual image. As an output, the tool generates a JSON file containing the identified crack pixels, and a dedicated program was developed to convert this file into a PNG image file.

C. Neural network

In this project, a crucial objective is the precise segmentation of cracks in images. For this, several neural network architectures were explored in order to select the most appropriate for this task. Several models were investigated, each displaying a unique set of attributes and limitations.

1) Auto encoder: A potential architecture that could be used for this task is an auto-encoder. Auto-encoders are unsupervised learning models that aim to learn a compressed, distributed representation of the dataset, most typically used for the purpose of dimensionality reduction or de-noising. They work by trying to reconstruct their input data from a lower-dimensional code, thereby learning the most salient features of the data, therefore, also making it useful for segmentation tasks. The first step towards implementing this approach was to apply a single-layer auto-encoder to gauge its viability for this task. If the single-layer did not show promising results another solution would need to be explored. In these initial tests, the single-layer auto-encoder was configured to segment cracks in the dataset. However, due to the simplicity of the auto-encoder model the results showed a rather broad detection range. As instead of just isolating the crack, the auto-encoder successfully detected all edges in the images, including those of the specimen and the machine, implying a general focus on high-contrast features rather than just the targeted fissure. An example of the results obtained from the single-layer encoder can be seen in Figure 8.

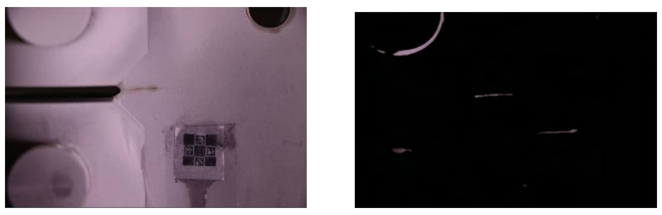

(a) (b)

Fig. 8: Initial test results of auto-encoder model, where image (a) is the original image used for testing and image (b) is the segmented mask produced by the auto-encoder

One may argue that increasing the models complexity by adding more layers would allow it the NN to learn more complex features and segment only the desired crack. However, it was anticipated that further layers would only continue to enhance edge detection, rather than specifically isolate the cracks. The reasoning behind this is that the core function of an auto-encoder, is to recreate its input, which inherently limits its applicability in tasks where the aim is to isolate specific features rather than mimic the overall input. This is particularly evident when the model fails to discern between various edge types, consequently detecting all edges, including the irrelevant ones, while the primary aim is to detect cracks.

In light of these results, it was concluded that the auto-encoder architecture did not present a suitable solution for the specific problem of crack detection. Hence, the decision was made to discard the auto-encoder approach and proceed with alternative, more promising methods.

2) U-net with VGG backbone: The U-Net architecture, a widely recognized approach in image segmentation, has demonstrated exceptional performance in various scenarios [7]. Drawing inspiration from its success, the decision was made to adopt the U-Net architecture as the baseline model. However, it was observed during experimentation that the traditional U-Net model struggles to accurately detect small objects and overlook cracks in their early stages. To address this issue, a backbone layer was introduced to enhance the feature extraction process.

In the context of neural networks, the backbone network refers to a pre-existing architecture utilized as a feature extractor. Typically, it consists of multiple layers of convolutional neural network (CNN) modules designed to learn hierarchical representations from input data. The main objective of the backbone network is to capture and extract low-level, mid-level, and high-level features that characterize the input data. The architecture is shown in

For this project, the VGG (Visual Geometry Group) network was chosen as the backbone network. The VGG network is renowned for its deep architecture and has demonstrated remarkable performance across various computer vision tasks. Prior to its application on the specific dataset, preliminary tests were conducted using the VGG backbone on a bridge crack dataset. The results obtained were highly promising, further affirming the suitability of the VGG backbone for the crack detection test.

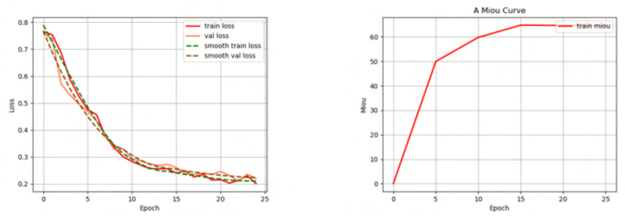
To test the performance of the neural network, three evaluation method are used. The definition and the result of these standard is shown as follows:

$$Precision = \frac{TP}{TP+FP}, Recall = \frac{TP}{TP+FN}, \tag{1}$$

$$IoU = \frac{TP}{TP+FP+FN}, \tag{2}$$

The performance of neural network is given in the following table:

	Recall	Precision	IoU
crack	0.63	0.88	0.58
background	1.00	1.00	1.00



(a) (b)

Fig. 9: Graphs illustrating the training process of the U-net, graph (a) shows the loss per epoch, graph (b) shows the mIoU per epoch

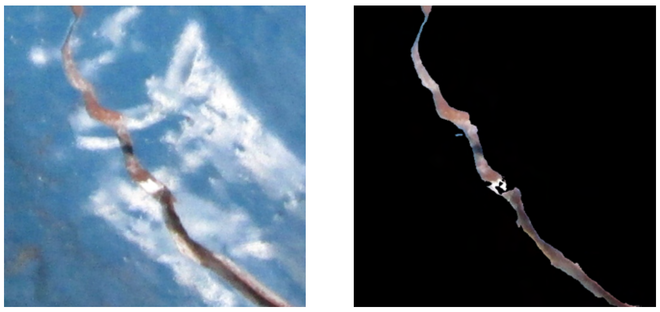


Fig. 10: U-net with 4d input test results, image (a) is the original image used for testing and image (b) is the segmented mask produced by the U-net with 4d input

3) U-net with 4d input image: The performance of a U-net model can be improved by adding future segmentation of the specimen to the input. The cracks become only clearly visible at the later stages of the experiment and are barely visible initially. As real-time segmentation is not required, a model that utilizes later specimen images can be developed. This means, that the segmentation is performed in the reverse order with the fully cracked image or the last image of the experiment at the beginning. The segmentation of the current input image is then added as a fourth channel to next input. This approach enables the model to identify the specific regions it needs to focus on for accurate segmentation. Consequently, the model would become more skilled at disregarding regions that may resemble cracks due to lighting or other image setup factors.

During training, the images of the same specimen are grouped into sequences. The ground truth mask of the previous image is added as the fourth channel to the next image. For the first image in a sequence, the fourth channel will be a non-segmented black image. Additionally, a non-segmented black image can be used as the fourth channel for a few random images in the sequence. This technique would improve the model's robustness, enabling it to perform segmentation effectively even when the provided fourth-channel segmentation is inaccurate. During the testing phase, the output of each image is saved and added as the fourth channel for the subsequent image, creating a sequential flow of information.

An architecture of a 5-layer U-net was implemented. Initially, the model faced difficulties in prioritizing the fourth channel. This issue was addressed by creating feature channels specifically for the fourth channel in the first layer of the encoder and concatenating it with the feature channels derived from the 4D image. This adjustment significantly improved the model's performance.

The model successfully segmented some test images 11, but the overall accuracy remained low. The training was conducted using Google Colab, which had a GPU limit of 15GB. Due to this, a larger model couldn't be trained. Using Colab, the model has to be trained on a single image at a time to avoid quickly reaching the GPU limit and interrupting the training process. Unfortunately, due to this limitation, the model was trained with only a few images and without batching, leading to overfitting. These constraints prevented further exploration of the model's potential.

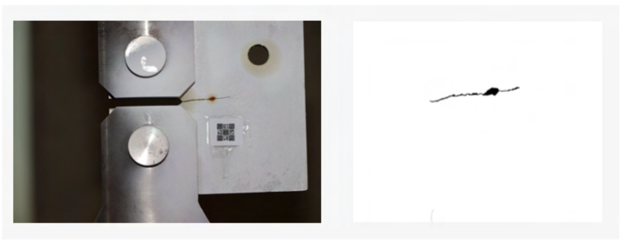


Fig. 11: Image showing the original image and the segmented output of U-net with 4d image for said image

However, it is anticipated that this model would exhibit exceptional performance if the complexity of the model is increased and a larger training dataset with a higher batch size is utilized. By incorporating a more intricate architecture and providing the model with a wider range of training examples, it can learn the task more effectively and improve its ability to accurately segment cracks. Additionally, employing a larger batch size during training can improve the model's generalization. Therefore, by addressing these factors, it is likely that the performance of the model would be significantly enhanced.

D. *Curve measurement*

In subsection III-C, it is explained how a mask of the crack can be created using a neural network. To finalize the AI system to determine the crack length, is to actually determine the length of the crack based on this mask. For this, the number of pixels within the crack must be determined to then be converted to mm. It was attempted to first utilize curve fitting and later an algorithm to determine the shortest path from begin to end. Both methods will be explained and discussed below. The conversion to the metric system is done using an Aruco sticker that was placed on the specimen, this too is discussed in this section.

1) Curve fitting: After obtaining the mask representing the original image, the next objective is to determine the length of the initial crack through a two-stage approach involving curve fitting and length calculation. The algorithmic procedure is as follows:

Initially, the mask from the previous step is considered as a graphical representation of a specific function type, such as a polynomial or exponential function. The goal is to find the optimal coefficients that best fit the mask. In one method, we assume that the crack can be approximated using the following form:

$$f(x) = a + e^{-bx+c} \tag{3}$$

Before applying the curve-fitting algorithm to the mask, an additional preprocessing step is performed. By observing the existing thickness of the mask, a precise definition is introduced, where the masks length corresponds to the shape formed by the lowest pixel in the mask. This shape, denoted as l, has a thickness of one pixel.

Next, a coordinate system is established, and the coordinates of the pixels within l are determined. These coordinates are then utilized in a curve-fitting algorithm known as the least squares algorithm. The objective of this algorithm is to minimize the sum of squared differences between the predicted values of the model and the actual data points.

Once the coefficients of the function are derived, the length of the crack can be calculated using calculus. Specifically, the length of the curve is evaluated using the following formula:

$$L = \int_a^b \sqrt{1 + (f'(x))^2} \tag{4}$$

Here, a represents the starting point of the crack, and b represents its endpoint. Consequently, the length of the crack is determined at the pixel level.

A limitation of this approach arises from our incomplete knowledge regarding the shape of the crack, which makes it challenging to accurately assume a fitting function, especially in cases where the crack exhibits branching patterns.

2) Dijkstra algorithm: One of the most used methods to determine the distance in a network, is the Dijkstra algorithm. It is one of the most famous algorithms in computer science in general [8]. When provided with a graph of nodes and edges between them, the Dijkstra algorithm can find the shortest path from a starting to a final node, the first publication about the algorithm can be found in [9].

To provide the algorithm with a graph, pixels can be used as nodes. The edges between nodes have a certain weight, which can be considered to be the cost for using the edge to move from one to a next node. First of all, the starting and end node must be determined. This can be done by finding the left and right most pixel of the crack as start and end, respectively. It would be negligible whether the top or bottom of the crack is being measured, due to the relatively small width of the cracks. Thus, for simplicity, the top pixels can be chosen as start and end points. Next, the relevant nodes and accompanying edges between must be recognized and created. One option is to only consider the pixels that are predicted to be part of the crack as nodes. The edges can be created between neighboring nodes and all have the same weight. As an example, a small possible mask of a crack was made with the edges between them, shown in Figure 12.

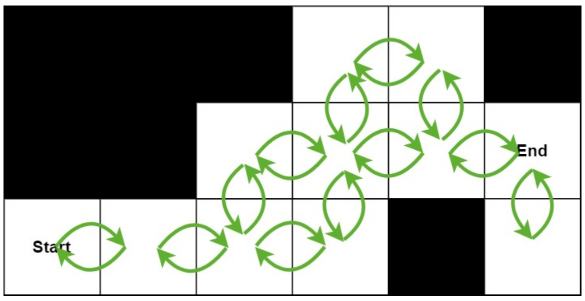


Fig. 12: Graph for the Dijkstra algorithm for a continuous predicted crack. The white pixels together form the predicted crack, functioning as nodes with the green arrows the edges (weight = 1) between them.

However, this method would not allow for any interruptions in the white line as then no path could be formed from start to end. This problem could be addressed by also including neighboring black pixels as nodes. In Figure 13, a crack with an interruption of one pixel is shown, with black pixels also functioning as nodes. The edges (in red) between the white and neighboring nodes now make sure that the interruption can be passed and the shortest path can still be found. These edges, however, should be much more expensive than the green ones between white nodes, to prevent the system from taking shortcuts. By setting the weight, e.g., a hundred times higher, the algorithm is expected to always use the white nodes except when there is an interruption.

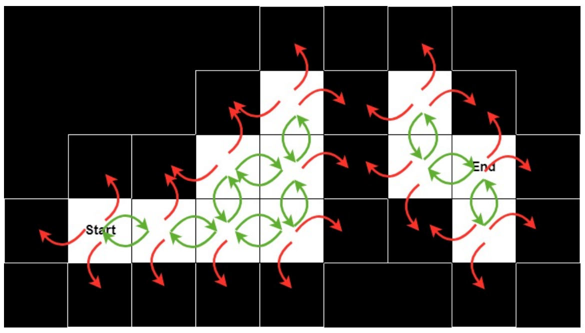


Fig. 13: Graph for the Dijkstra algorithm when there is a gap of one pixel in the crack prediction. The white pixels together form the predicted crack, the black pixels are the pixels directly surrounding them and also function as nodes. The green arrows are the edges (weight = 1) between white nodes, the red arrows the edges (weight = 100) between white and black nodes.

When there are larger interruptions, also black pixels that are indirect neighbors should be considered as nodes, with edges that are even more expensive to include in the shortest path.

As an example, the shortest path that was found by the algorithm in a label, is shown in Figure 14.

Fig. 14: An example of a label of a crack with in red the shortest path that was found with the Dijkstra algorithm.

Using the Dijkstra algorithm is expected to work quite well, as it can handle any shape of the crack and does not require pre-knowledge about it. Also, it is expected to be more exact than the curve fitting method as it is a more 'custom-made' method, assuming an accurate mask. A limitation however, is that if the mask is not accurate, the path can have large deviations from the actual length as shortcuts or detours are taken. Furthermore, if there are interruptions that have a width of multiple pixels, it might be computationally expensive to create the graph, as many (indirect) neighbors should be taken into account too.

3) Aruco sticker: The initial measurement acquired of the crack is in pixel units. However, for practical use, it is critical to convert this pixel length into a real-world unit of measure, such as millimeters. This necessitates the introduction of a scale factor, which can be computed using a ChArUco diamond marker of known dimensions. A ChArUco diamond marker is a variant built off of a Aruco marker, it is a grid of black and white squares (the white squares are aruco markers) as seen in Figure 15a, with a specific checkerboard design allowing it to be easily detected and identified in images. By placing a ChArUco diamond marker with known dimensions (in millimeters) within the field of view of the image, a pixel-to-mm conversion factor can be established.

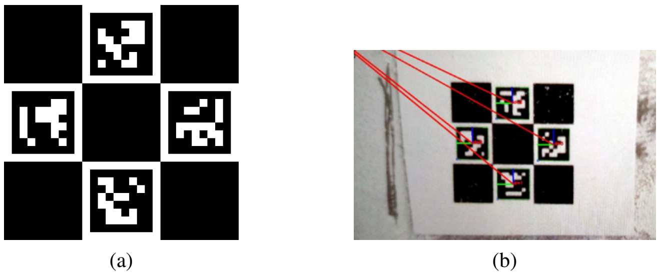


Fig. 15: demonstrating how the ChArUco diamond marker works. (a) example of ChArUco diamond marker, (b) demonstrating how the marker is detected

Let d_{mm} denote the known dimension of the ChArUco diamond marker in millimeters and d_{px} represent the corresponding dimension in pixels as measured in the image. The conversion factor, f, can be computed as the ratio of these two quantities:

$$f = \frac{d_{mm}}{d_{px}} \tag{5}$$

It must be noted that the detection algorithm for the ChArUco diamond marker is resilient to image distortions. Meaning if the marker is imaged at an angle, the detection algorithm uses the geometric properties of the marker to perform a perspective transformation, effectively 'deskewing' the marker and maintaining accurate dimension measurements. An example of how the detection algorithm works can be seen in Figure 15b. Once the conversion factor is determined, the length of the crack, L_{px} in pixels, can be converted into millimeters using the following relation:

$$L_{mm} = L_{px} \times f \tag{6}$$

Where L_{mm} is the crack length in millimeters. The adoption of a ChArUco diamond marker in this process provides a reliable method for obtaining accurate real-world measurements from the digital images. Moreover, as the conversion factor is established automatically from the image itself, this technique allows the program to autonomously compute the length of the crack in millimeters.

E. Image acquisition

The image acquisition process is an essential part of the visual crack detection system. This process is necessary as the cracks in steel specimens develop over thousands of cycles during fatigue tests. Thus, capturing consistent and reliable images over this long period forms a cornerstone of this AI-based crack detection approach. The image acquisition task is inherently repetitive and prone to human error when performed manually, particularly over such an extended period, not to mention extremely tedious. Thus, an automated image acquisition (AIA) system, like the one described in this report, reduces the potential for human error and ensures consistent image capture over time, while facilitating future studies in material testing.

The AIA system needed to be able to work independently from the testing machine controller and command interface, meaning the AIA can be used on any test setup given they have an output force signal. Furthermore the AIA system as depicted in Figure 16 was designed with an emphasis on reliability and adaptability, allowing for future improvements and accommodating different research needs. It must be noted that once the box is sealed off all electronics are sealed off except for the input and output ports, this is not only for aesthetic purposes but also to ensure safety during use and to protect the components from any damage.

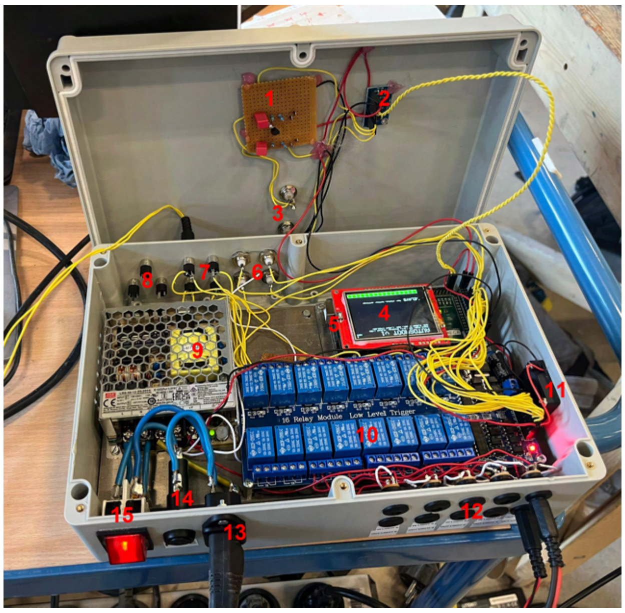


Fig. 16: Interior of Automated Image Acquisition (AIA) system. 1: high pass filter and protective circuit for Arduino, 2: I2C ADS1115 16-bit ADC to capture the input signal, 3: two input ports for differential and single ended signals, 4: 2.4-inch TFT display shield with touchscreen for user interaction, 5: Arduino Mega 2560 R3 is the central controller, 6: general purpose I/O ports, 7: output plugs for 4 cameras (only testes with Canon cameras), 8: output plugs for any output requiring a pure contact to trigger, 9: Mean Well Power Supply - 12V 4.2A used to power the system as converts 230V AC into 12V DC, 10: 12V Relay 16-Channel Low-active - with LM2596 Step-down (5V) Buck Converter it takes several digital inputs and switches the relays, 11: fan for cooling, 12: output plugs rated for up to 36V, 3A, used to control flood lights, 13: input power supply, 14: protection fuse, 15: bi-polar on/off switch

In this design, the Arduino Mega 2560 R3 was selected as the central controller due to its high processing capability and numerous I/O ports, allowing for efficient communication with various components and management of input/output operations. Along with the Arduino a 2.4-inch TFT display shield with touchscreen capability is employed to facilitate user interaction. Its function is to offer real-time updates on the AIA status, enables the user to change the settings if desired and displays errors if they occur.

Furthermore, a 16-channel relay, powered by a 12V power supply, is integral to the system's function. This component is needed for all the switching operations, as it allows for precise control over the connected devices, such as cameras and flood lights. A DC-DC buck converter is incorporated in the relay to manage power supply to the system components. This not only ensures efficient power use, but also regulates the voltage supplied to each component. The inclusion of an ADS1115 16-bit ADC in the system is to accommodate the conversion of incoming analog signals into digital signals. This ensures the Arduino, which operates on digital signals, can effectively process the data received. Lastly the Mean Well Power Supply converts 230V AC into 12V DC, so the system can be powered by a standard power outlet, it can support up to 3A and is protected by a 2A fuse and switch.

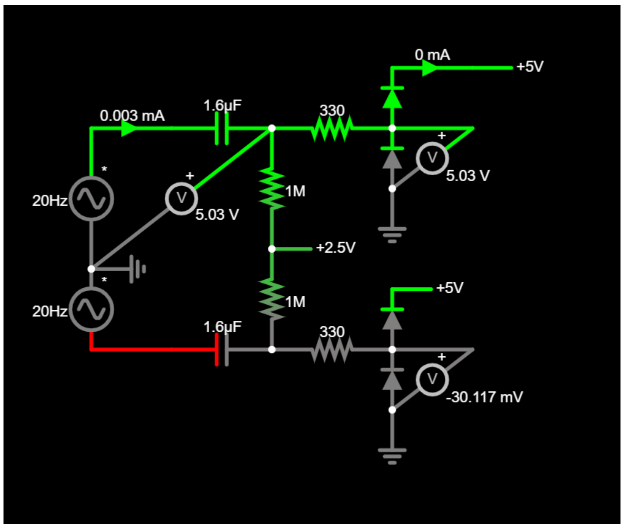


Fig. 17: Diagram of circuit used to protect Arduino from input signal

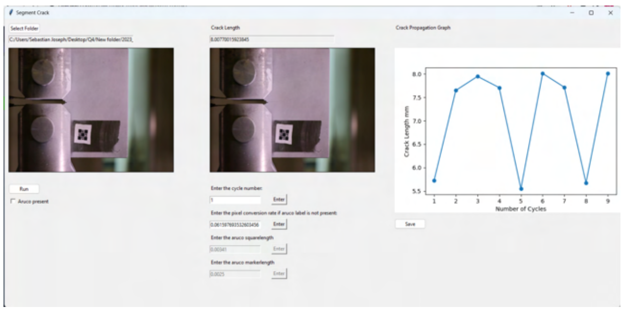
As the AIA needs to work independently from the machine controller a protective circuit had to be designed to safeguard the Arduino from potential power surges or short-circuits. The circuit acts as a voltage clamp and insulator, shielding the Arduino's input pins from voltage spikes and guarantees a maximum input voltage of several volts over 5V. The single ended input port however is protected only by a 5.1V zener diode which clamps the voltage between -0.6 and +5.1 volts. The complete circuit diagram is provided in Figure 17.

The core software of the AIA is developed in C++, utilizing the standard Arduino library and environment. The simplicity and flexibility of C++ allow the software to process multiple operations concurrently. This simultaneous processing capacity is vital as the software must manage a variety of functions in real time, including signal acquisition, signal processing, output management, and display update. The input signal is read 200 times a second, this frequency has been chosen to provide an accurate and representative reading of the input signal, capturing enough data to facilitate comprehensive analysis while maintaining system performance. This does restrict the AIA to only work on tests with a maximum frequency of $15Hz$. It's important to understand that, in differential mode (differential input signal) the protection circuit, which is a high pass filter, deletes the common mode signal and leaves only high frequency voltage changes. These changes are usually quite close to zero, meaning that the input signal needs to be scaled down to provide a meaningful data set for analysis.

After the acquisition of the input signal, the software calculates two essential statistical measures from said signal: the mean and the variance. The variance is particularly crucial for detecting a plateau in the signal. These calculations occur every 500 milliseconds, providing regular and timely updates of the signal's behavior. This detection process uses a single threshold trigger, a relatively simple yet effective mechanism for detecting state changes in a signal. The threshold for variance is set empirically, based on observed system behavior, this allows the system to define two states for the signal: HIGH and LOW. A HIGH state indicates that the variance is above the threshold, signifying that the signal is changing rapidly, indicating cycling. On the other hand, LOW state (close to zero), is when the variance is below the threshold signifying that the system is at rest, this is interpreted as a detected plateau. Upon plateau detection, the software initiates a specialized routine comprising four main stages: lights on, camera shooting, lights off, and reset. These stages are carried out in sequence, with each one initiated only after the previous one is complete to ensure system stability and reliability. The first stage of the routine involves turning on the lights. The lights remain on for several seconds before the camera is activated. This delay is intentional and essential, giving the camera's Auto White Balance system time to adjust to the lighting conditions. Next comes the second stage: camera shooting, where the software activates the camera to capture an image. After the image capture, a delay is implemented to provide sufficient time for the camera to complete its shooting process, as depending on the camera settings (ISO) it may take a bit of time. Subsequent to the camera shooting stage, the lights are turned off in the third stage. Then the final stage the system is reset so the system is ready for the next potential plateau. This also ensures that the system will not initiate multiple plateau routines in quick succession, preventing the overlap of routines that would compromise the data (images) collected.

F. *Graphical User Interface*

The final part of the developed system is the graphical user interface (GUI). This component lays on a layer above all the others and its purpose is to integrate and connect everything. This GUI allows the user to seamlessly use the tools developed to determine the crack length in an image acquired during


Segment Crack

Select Folder: C:/Users/Sebastian Joseph/Desktop/Q4/New folder/2023_

Crack Length: 8.00770015923845

Enter the cycle number: 1

Enter the pixel conversion rate if aruco label is not present: 0.061597603532603456

Enter the aruco squarelength: 0.00341

Enter the aruco markerlength: 0.0025

Fig. 18: The Graphical User Interface (GUI) used to determine the crack size of the collected test images

testing. Upon launching the GUI, which can be seen in Figure 18, the user can select the desired input folder (containing the acquired images) by clicking on the "Select Folder" button. The user can input various parameters like the number of cycles between each image, Aruco input values and the pixel-to-mm ratio using various readable text inputs. Additionally, there is a checkbox to indicate the availability of Aruco markers on the sample. The Aruco input text fields will be enabled only if this checkbox is checked. If Aruco is unavailable the pixel-to-mm ratio given by the user will be used for the calculation of crack length.

The user can initiate the program by clicking on the "Run" button. This button will only be enabled if the input folder and number of cycles are provided. The GUI features two image placeholders for displaying the current input image and highlighted crack path. The crack propagation graph will also be displayed in the GUI. These features will allow users to observe the progress of the analysis. As the program runs, it calls the model to predict the crack segmentation, which is then passed to the Dijkstra module to calculate the crack length in pixels. The Aruco module is also called later, to check if Aruco is present and to calculate the pixel-to-mm ratio. The text fields for the conversion rate and crack length will be updated for each image automatically as the analysis progresses.

Once the program completes the analysis for the entire selected folder, the "Save" button becomes enabled. The user can use this button to save the crack propagation graph and generate a CSV file containing the number of cycles, crack length, starting and ending crack pixels and Aruco pixel values. The GUI also saves all segmentation and crack path generated as images for the user to inspect them later if required.

IV. Validation

In the previous section, the methods and their reasoning for the different parts of the project have been explained. Here, the validation of the whole system is discussed.

For the validation the whole fatigue test of a specimen was measured, with the system developed during this project -the AIA- and conventional techniques. For this process, from now on called the demo, an aluminium specimen was used. This was done because the fatigue test of an aluminium specimen is completed faster than that of a steel specimen, which was preferable due to limited time, while showing a similar cracking as a steel specimen. During the demo, a sinusoidal cyclic load with a frequency of 8 Hz, an average of 5.5 kN and an amplitude of 4.5 kN was used. After cycling for 500 cycles the test is kept at the maximum load of 10 kN for 1 second, creating a plateau. During this plateau the AIA triggers the flood lights, then the camera to take an image and shuts down the lights. Next to that, the MOOG software records the applied force and the clip gage. These acquired clip gage values can be translated into a predictive crack length in the following way.

In the standard for conducting fatigue crack growth tests [4], it is explained how the crack size a [mm] can be estimated, using the equation:

$$\alpha = \frac{a}{W} = \sum_{i=0}^{N} C_i \cdot (u_x)^i \quad \text{with} \quad N = 5. \tag{7}$$

Here, α [-] is a ratio that should have a value between 0.2 and 0.975. The width of the specimens is W [mm] and C_i [-] are constants that depend on measurement location. The dimensions of a specimen could be seen in Figure 2. There, it is also shown how the value for a, the crack size, is measured. The value for u_x can be determined with the following equation:

$$u_x = \left\{ \left[\frac{EvB}{P} \right]^{\frac{1}{2}} + 1 \right\}^{-1}. \tag{8}$$

With E [Pa] the Young's modulus of the material of the specimen; v [mm] the maximum value of the gap as measured with a clip connected to the crack opening; B [mm] the thickness of the specimen and P [N] the maximum force that is being loaded upon the specimen. The value of E can differ from the theoretical value of the material. To calibrate the actual value of E for the used specimen, the crack size should be measured with a caliper or microscope.

The predicted crack length can then be used as a reference to the crack length that can be found with the AI system, using the images taking by the AIA. Such, the accuracy of the system can be indicated. Next to that, one specimen with a crack was photographed from multiple angles and with different zoom settings. Ideally, the system should predict that the crack size in every picture is the same. With this, a measure for the sensitivity is determined. Below, all the results are discussed and compared.

A. Results

1) Crack size accuracy: To obtain a measure for the accuracy of the AI system, the crack sizes that are calculated with the clip gage and the crack sizes as predicted by the system are compared. This requires that the calculated crack size should be adapted to represent only the crack itself and not the notch of the specimen. This adjusted crack size is called Δa [mm]:

$$\Delta a = a_0 - a, \tag{9}$$

with a_0 [mm] the length of the notch, thus the crack size when the specimen is not yet cracked. For the aluminium specimens, this value was measured to be 25.9 ± 0.1 mm.

During the demo a total of 193 times a run of 500 cycles was completed. This means that there are 193 clip gage values and corresponding images. However, not the full cracking of the specimen was usable. Only the first part where the crack was propagating can be used. Therefore, it was decided to only take into account the first 150 runs, resulting in a total of 75,000 cycles. Using the predicted crack size of every eight run, the result in Figure 19 is found. All the values are given in Table I

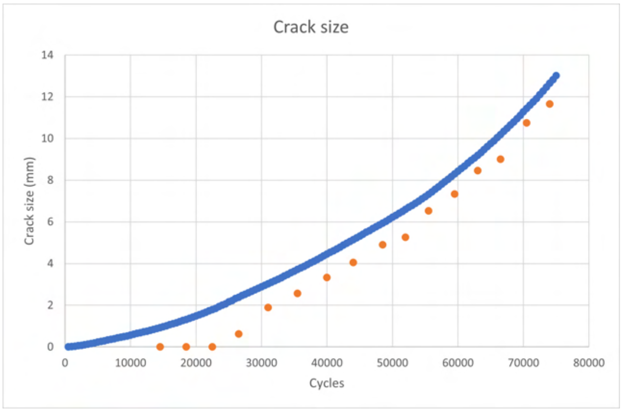


Fig. 19: Comparison of the crack size found with the clip gage (in blue) and the crack size predicted with the AI system (in orange).

As can be seen, there is an offset between the actual crack size as it was calculated with the clip gage and the predicted crack size. The offset slightly decreases for larger crack sizes, but overall is quite stable. The relative error, calculated by dividing the difference in the two values by the actual value, does decrease significantly for larger crack sizes, as is shown in Figure 20.

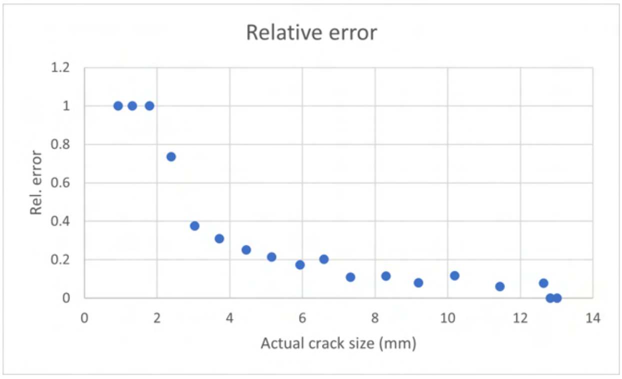


Fig. 20: The relative error of the predictive crack size against the actual crack size, showing a significant decrease for increasing crack sizes.

This means that there is a systematic error in the system, the random errors that occur appear to be of less influence. This systematic error is most likely due to to small cracks and the final part of cracks not being detected. As the average absolute error (the offset) is 1.1 mm with a standard deviation of 0.3 mm, it is expected that cracks of this size are not detected by the system.

2) Crack size sensitivity: One way of giving a measure to the sensitivity of the AI system, is by feeding it different photos of the same crack. Such, an indication of the sensitivity for the zoom and the angle can be calculated. In total, six different zoom settings and four different angles were used. Some of the used images are shown in section A. The image with that was taken with standard zoom settings and without an angle, was used for the determination of the errors of the predictions of the other images.

The model was given images from different angles for the same sample. The model was able to provide nearly identical predictions for these images. The model was able to give predictions with a mean relative error of 0.00819 and a mean absolute error of 0.2127 mm. Similarly, the model's performance for different camera zoom settings was also conducted and a mean relative error of 0.0118 and a mean absolute error of 0.3070 mm was observed. All values can be found in more detail in Table II. This indicates that the model is able to give consistent predictions irrespective of position of the camera with respect to the sample.

The model is performing this efficiently due to the Aruco marker. We are using the Aruco marker to calculate n the pixel-to-mm ratio by comparing the pixel length of the side of the marker and the real length of Aruco.

Overall, it can be seen that the systematic error in the system is of more influence than the random error, even when it is attempted to increase it by changing the zoom or angle settings. The accuracy of the AI system is 1.1 ± 0.3 mm and the sensitivity is 0.259 mm. This means that the system as it is, could already be used to obtain reasonably reliable results, showing that this is a promising method of determining the crack size in metal specimens. It is expected that by further improving the model, the accuracy and sensitivity of the system will only improve.

B. *Future recommendations*

As was shown in the previous subsection, the AI system already gives promising results, but there is still room for improvement. The most progress could be made in the training of the neural network. Now, the training of the neural network was limited by the RAM capacity of the used GPUs. This required the downsampling of the images, hence information that was especially useful for the smaller cracks was not being taken into account. The training could be improved by using GPUs with a larger capacity, but also by improving the learning method. For example, first a downsampled image could be used to determine a region of interest. This region of interest would be a rectangular of the area that contains the crack. This region, then, could be used to train the network on in full resolution.

Furthermore, to ensure that the fatigue testing of a specimen can be done without interruptions, it would be beneficial if the image acquisition box could detect peaks and trigger the camera without the need for creating plateaus. Whether this is possible does depend on the trigger speed of the camera and the frequency with which the load is being applied.

For future research, it might be interesting to look into the possibilities of using the visual aspects of the crack growth more. This could entail determining the angle of the crack and possibly even predicting the trajectory of the crack. Using these features has the potential to make this method even more promising for future applications of the research.

V. Conclusion

The objective of this research was to develop an AI-based system, to automate the process of detecting and quantifying the length of cracks in metal specimens undergoing fatigue tests. Utilizing an AI-driven approach provides an innovative alternative to traditional manual techniques, which are generally time-consuming and prone to human errors, or other estimation-based methodologies that may not provide the same level of accuracy. The project is divided into five primary components: Data Collection, Data Labelling, Neural Network Development, Curve Measurement, and Image Acquisition. Each part plays a critical role in constructing the overall system, which are all needed to create an efficient and accurate crack detection tool.

Firstly, custom data was collected and labelled in order to train the AI system. From here several neural network models were explored, in order to find the best fitting for this task, a UNet with a VGG backbone was chosen in the end. The system was further refined by introducing an advanced curve measurement technique to accurately calculate crack length. To allow for efficient image acquisition, a hardware box with an Arduino Mega 2560 R3 as the basis was designed and created. The system was validated using an aluminium specimen, which replicates the cracking behavior of a steel specimen, albeit at a quicker pace. During the test, the AI and crack length system's accuracy was evaluated by comparing its crack size predictions with those calculated via conventional methods (using a clip gage). The AI system showed remarkable promise, despite the presence of a systematic error. This error, attributable mainly to the non-detection of small cracks, was quantified as an average absolute error of 1.1 mm with a standard deviation of 0.3 mm.

Moreover, the system's sensitivity was assessed by feeding it images of the same crack from different angles and zoom settings. The AI system consistently performed well, indicating its robustness against changes in camera angles or zoom levels. This consistency is largely attributed to the use of an Aruco marker, which helps in calculating the pixel-to-mm ratio and consequently, ensuring the model's performance.

Overall, the project's success lies in creating an automated and reliable crack detection tool for metal structures, which are integral components of civil infrastructure. Its performance validates the efficacy of AI-based solutions for crack detection and measurement, with accuracy and sensitivity measures showing promising results. Even though there are areas of improvement, the current system can already generate reasonably dependable results, rendering it a valuable asset in metal specimen fatigue tests. Recommendations for future development include refining the training process of the neural network to increase its detection capability for smaller cracks. Utilizing a GPU with higher RAM capacity for handling higher resolution images or implementing regions of interest would also improve the performance greatly. Another aspect that could be improved, is by enhancing the AIA code to recognize individual peaks during the sinusoidal cycle, thereby eliminating the need for testing cycles with plateaus.

The findings of this project underscore the considerable potential of AI in advancing the realm of material science and engineering, particularly in specimen testing scenarios. These developments not only optimize the testing procedures but also open the doors for future exploration and innovation in the application of AI-based solutions within the broader sphere of materials testing and examination.

VI. Evaluation Javier

At the start of our group project, we found ourselves in a bit of a fog. We had a general idea of what we wanted to achieve, but the path forward seemed unclear. It took some time for us to gain momentum and start making real progress. Like with any project, getting started was the toughest part. However, as we dove deeper into the work, we realized that collaboration and coordination were key.

I was assigned to the image acquisition component along with Vera and Sebastian. Our initial task was to find a way to capture images at the peak of the cycle. Little did we know, it would be more challenging than we anticipated. System delays and the rapid pace of specimen testing presented obstacles. We had to think on our feet and adapt. We decided that understanding the available equipment was crucial to designing a viable solution. This led us to explore the use of an Arduino Mega, which wasn't initially part of our plan. Additionally, I was heavily involved in the data collection process, working alongside the rest of the team in the lab.

Since the midterm, I've dedicated a significant amount of time to creating the physical hardware for the image acquisition as well as testing the system. Also spent several hours labeling the images to train the AI. I've actively participated in decision-making tasks and brainstorming sessions, ensuring that all the pieces come together seamlessly. The teamwork and collaboration within the group has improved immensely, especially once we gained a clearer understanding of our objectives. We've learned valuable lessons for future projects, focusing on efficiency and better communication. Furthermore, I'm extremely proud of what we've accomplished so far and have thoroughly enjoyed working on this project.

Looking back, one aspect I would have wanted to spend a bit more time on is on the AI and not only focusing on image acquisition components. I also believe that would have benefited from just trying stuff out, as it's better to take a leap of faith and start experimenting, learning from both successes and failures, rather than spending excessive time trying to determine the perfect solution from the outset.

VII. Evaluation Alessandro

In the beginning of the project, we decided to divide everything into two parts: the AI development and the data collection. At first, I started with the AI section of the project. The first thing to do was to understand the problem. Me and Sizhuo split the problem into 3 sub-problems: detection of the crack, measurement of the crack and final analysis of the crack. The detection became pretty much instantly a segmentation problem (decide which pixels are a crack and which are not) and therefore I began by trying know models, such as U-Net, which are known to work really well for this kind of work. We started by training the AI with a dataset that was provided to us. This set of images was not labeled and then we had to label some images by hand in order to get started. This however has not been the final method for labeling as it was too slow and we decided to go for something with an automation component to speed up the process. This was kindly provided as an already made tool by a PhD student. After this we created the measuring program which is able to measure the crack size in pixel. This is then converted into millimeter values with the help of the ChArUco diamond markers that we decided to put onto each specimen. After this, since we understood the need for data, we decided to temporarily switch to the data collection team and help them to speed up the process. I then built a small system with an Arduino mega that triggers the camera when the controller halts the cycles and applies a fixed load to the specimen to better see the crack. This system seems to be more versatile as the Arduino can be programmed to interface with a lot of different cameras autonomously or via a computer and also directly with various types of sensors, for example the load cell that measures the force applied to the specimen. This board has been then aggregated with a set of relays that also manage the lights around to not waste energy and with more than one camera in order to have a multi-camera setup that takes pictures from different angles, with different light settings and possibly with different colors of the lights. Overall I think that this project went really well and I learned a lot from it. The people inside the team I think did everything together and everyone worked on their part in the best way. I also enjoyed the lab work as it is very practical and I personally think that it's the best way to learn. Although I wished that for this time we could have had something that works even better, I still think that we have done a good job, in a project that started all the way back from data collection to then move onto the final AI model, like what we will be supposed to do in a company.

VIII. Evaluation Vera

About two weeks after starting with the project, we decided to spread the group over two separate tasks, acquiring the data and creating the neural network. Together with Javier and Sebastian, I worked on the data collection and image acquisition. Later, Alessandro also helped with the data collection. Our first task was to figure out the exact set-up and how we should connect the different parts of the set-up. This turned out to be quite challenging as it was rather unclear which parts of the set-up we had to decide upon and what equipment we could use. After some back-and-forth between us, our project owner and the technicians at the structure lab in Vertigo, we had a clearer idea of what to use and made a first proposal for the set-up. But only after starting working the actual testing machine, the full set-up became clear to me. After finding a way to acquire data, as was described in subsection III-A and the midterm presentation, we decided to merge the two groups again.

Up to this point, we did meet up quite often and made much progress, but our meetings tended to be somewhat chaotic. As this was mainly my responsibility as project leader, I wanted to change this to ensure that we would finish the project in time. Together with our project owner and track coordinator we decided to implement a more strict structure with clear and written down tasks and future goals. This definitely helped our progress.

After the midterm, we have labeled our data, tested multiple neural networks and trained the most promising one, created the image acquisition box, written a script for the curve measurement and combined all these elements. Next to that, we validated our system as a whole with a demo. My main contributions are to the data collection, data labeling, writing the curve measurement script, keeping an overview and making sure that this report was written.

During this project, I have learned quite a lot about performing experiments and working with electronica devices. This was fairly new to me, making it interesting and sometimes challenging. Furthermore, I learned more about how to lead a project with so many facets and diverse people. But most of all, I learned to work in a culturally very diverse group where everyone has different working and communication styles. It was very nice to work intensively with a group for a longer period, as it allows a group to grow.

Overall, I think that we as a group have learned a lot and are working together well. Everyone is more than willing to explain concepts to the others and help each other out whenever needed. Much time was spend on this project, especially by those designing and assembling the image acquisition box, and I think we managed to do many things and can present a product that we are proud of. Now, at the end of the project, we are on a tight schedule. This means that our report is not as polished as we might have wished for, though still as correct and complete as possible. I have faith that we will be able to give a nice presentation and demonstration next week, showing our work.

IX. Evaluation Sebastian

The project had two different parts- neural network and data collection. The neural network is the machine learning model that will predict the cracks in the image. The data collection includes collecting images to train the model and designing a system to capture images during the testing. We are expected to design an overall system that will take pictures of the test specimen during the testing and send data to the model to perform the prediction. During the first quarter, my main focus was on data collection and image acquisition. As part of the team, we gathered training images to build a robust dataset. It was an iterative process where we experimented with multiple setups until we reached the final setup that met our requirements. Moving into the second quarter, my primary responsibility shifted towards the development of artificial intelligence (AI) components. We trained several models to identify the best-performing one for our project. I took the initiative to implement the U-Net architecture with 4D input, which proved to be effective in our specific context. Additionally, I played a key role in implementing the graphical user interface (GUI). Overall, this project provided me with valuable hands-on experience, fostering teamwork and collaboration among the team members. It allowed me to gain insights into the end-to-end process of AI development and the practical challenges associated with creating a user-friendly UI system.

X. Evaluation Sizhuo

Working on this interdisciplinary project has been an enriching experience, allowing me to collaborate with partners from diverse fields and combine our expertise to address a complex problem.

My primary responsibility in this project was to develop a neural network using the U-Net architecture for crack detection through semantic segmentation. Initially, I implemented a pure U-Net model, but it struggled to accurately detect small cracks. To overcome this limitation, I incorporated a backbone network into the U-Net architecture. By leveraging the capacity and feature extraction capabilities of a pre-trained model, the enhanced U-Net performed better on smaller cracks.

Dealing with variations in image sizes within the dataset posed another challenge. To ensure consistent training, I employed a downsampling technique to resize all images to a uniform size. This not only resolved the issue of differing dimensions but also reduced computational overhead during training.

During the fine-tuning phase, I observed a significant class imbalance between crack and non-crack categories, with the majority of pixels belonging to the non-crack class. To address this imbalance, I applied class weights. Assigning higher weights to the minority class (crack) ensured that the model focused on capturing crack patterns accurately, leading to a better balance between the two categories.

Additionally, I recognized that the original meta model is not for labelling and lack of user friendliness. To enhance the user experience, I took the initiative to develop a user interface that simplified interactions with the model. This interface acted as a bridge, providing a more intuitive and accessible means of utilizing the sophisticated model architecture.

In conclusion, this interdisciplinary project provided a valuable opportunity to collaborate with people from various fields. By incorporating a backbone network, downsampling images, addressing class imbalance, and creating a user-friendly interface, I contributed to building and improving the accuracy and usability of the crack detection system. I am grateful for the opportunity to work alongside my partners, and their contributions have been instrumental in the project's success. Together, we have made significant progress in advancing crack detection through semantic segmentation.

APPENDIX

The code in Listing 1 sets up an output pin to control the camera and receives commands through the serial port which is connected to the computer. The integer variable OUT PIN is assigned the value 13, which is the pin number that will be used for the output. In the setup() function, the pin is configured as an output and initially set to a low state. The Serial.begin(9600) function initializes the serial communication at a baud rate of 9600 bits per second. In the loop() function, the code checks if there is any serial data available using the Serial.available() function. If there is data available, it reads the incoming data as a string using Serial.readString() and stores it in a String variable called cmd. If the received command is ”trigger”, the code prints ”triggered” to the serial port using Serial.print(), this was set up to ensure the Arduino is properly taking the image and the desired moment. It then sets the output pin to a high state using digitalWrite(OUT PIN, HIGH), causing the camera to take a picture. The code then waits for 2 seconds using delay(2000), before setting the output pin back to a low state using digitalWrite(OUT PIN, LOW. Finally, the code waits for another 1 second using delay(1000) before looping back to check for more incoming serial data.

Listing 1: Arduino (C++) code for triggering an output pin

```
int OUT_PIN = 13;

void setup() {
    pinMode(OUT_PIN, OUTPUT);
    digitalWrite(OUT_PIN, LOW);
    Serial.begin(9600);
}

void loop() {
    if(Serial.available()) {
        String cmd = Serial.readString();
        if (cmd == ”trigger”) {
            Serial.print(”triggered”);
            digitalWrite(OUT_PIN, HIGH);
            delay(2000);
            digitalWrite(OUT_PIN, LOW);
            delay(1000);
        }
    }
}
```

In order for the Arduino to set off the camera it needs to receive the trigger message from the computer which is controlling the press. This is done using the python code presented in Listing.3 which is run on this computer after the press has been started with the given cycle as previously mentioned. The code reads the log file of the software used to control the press and sends a command to the Arduino when at the chosen moment(at every plateau). In order to do this the serial, time, and os modules are imported. Then, a serial connection is established using serial.Serial with the port name ’COM5’ (or which ever COM the Arduino is connected to). Next, the code reads the log file and stores the first 19 characters of each line containing ”AppOutput” in a list called lines done. The code then enters a loop where it opens the log file again, reads its lines, and checks if a line contains ”AppOutput”, (a command only given when the press goes to a plateau that was programmed into the cycle) and has not already been processed (i.e., the first 19 characters are not in lines done). If the condition is met, the code prints the first 19 characters of the line and the rest of the line after ”AppOutput” and sends the command ”trigger” to the connected Arduino using ser.write(b’trigger’). The first 19 characters of the line are added to lines done. Finally, the code closes the log file, reads any data from the serial connection using ser.read(128), waits for 1 second using time.sleep(1), and repeats the loop. The serial connection is closed using ser.close() when the code is terminated.

Listing 2: Python code to read a log file and send a trigger to a serial port

```
import serial
import time
import os

ser = serial.Serial('COM5', timeout=1)
print(ser.name)
lines_done = []

file1 = open('C:/Program␣Files␣(x86)/Moog
/Integrated␣Test␣Suite/TestSuite/logs
/MoogIntegratedTestSuite.log', 'r')
Lines = file1.readlines()
for line in Lines:
    if ”AppOutput” in line:
        lines_done.append(line[0:19])

while 1 == 1:
    file1 = open('C:/Program␣Files␣(x86)
␣␣␣␣/Moog/Integrated␣Test␣Suite/TestSuite
␣␣␣␣/logs/MoogIntegratedTestSuite.log',
    'r')
    Lines = file1.readlines()
    for line in Lines:
        if ”AppOutput” in line
        and not line[0:19]
        in lines_done:
            lines_done.append(line[0:19])
            print(line[0:19] + ”␣” + 1
            line[line.index(”AppOutput”)
            + 9:])
            ser.write(b'trigger')
    file1.close()
    ser.read(128)
    time.sleep(1)
ser.close()
```

Listing 3: Python code to read a log file and send a trigger to a serial port

```
    Model(
  (vgg): VGG(
  (features): Sequential(                    (final): Conv2d(64, 19, kernel_size=(1, 1), str
    (0): Conv2d(3, 64, kernel_size=(3, 3),)stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
)
(up_concat4): block(
  (conv1): Conv2d(1024, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (up): UpsamplingBilinear2d(scale_factor=2.0, mode=bilinear)
  (relu): ReLU(inplace=True)
)
(up_concat3): block(
  (conv1): Conv2d(768, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (up): UpsamplingBilinear2d(scale_factor=2.0, mode=bilinear)
  (relu): ReLU(inplace=True)
)
(up_concat2): block(
  (conv1): Conv2d(384, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (up): UpsamplingBilinear2d(scale_factor=2.0, mode=bilinear)
  (relu): ReLU(inplace=True)
)
(up_concat1): block(
  (conv1): Conv2d(192, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (up): UpsamplingBilinear2d(scale_factor=2.0, mode=bilinear)
  (relu): ReLU(inplace=True)
)
```

APPENDIX

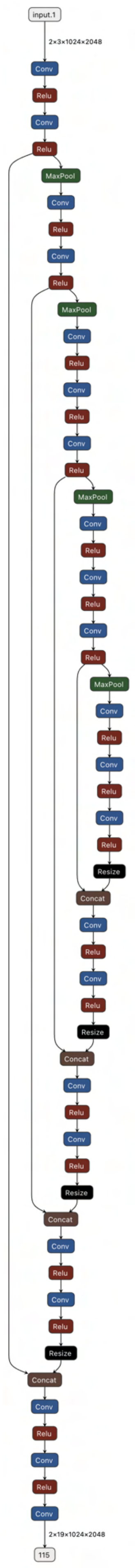


Fig. 21: model architecture

APPENDIX

TABLE I: The absolute and relative error in the prediction of the crack size by the AI system, showing the accuracy.

Cycles	Actual size (mm)	Predicted size (mm)	Absolute error (mm)	Relative error (-)
14500	0.924	0	0.924	1
18500	1.318	0	1.318	1
22500	1.789	0	1.789	1
26500	2.383	0.63	1.753	0.736
31000	3.032	1.89	1.142	0.377
35500	3.715	2.565	1.150	0.309
40000	4.450	3.33	1.120	0.252
44000	5.151	4.05	1.101	0.214
48500	5.934	4.905	1.029	0.173
52000	6.596	5.265	1.331	0.202
55500	7.324	6.525	0.799	0.109
59500	8.298	7.335	0.963	0.116
63000	9.195	8.46	0.735	0.080
66500	10.193	9.00	1.193	0.117
70500	11.441	10.755	0.686	0.060
74000	12.645	11.655	0.990	0.078

TABLE II: The absolute and relative error in the prediction of the same sample at different camera setting. Setting 0 shows the crack size that is predicted for the image that was used as the basis.

Camera setting	Predicted size (mm)	Absolute error (mm)	Relative error (-)
Setting 0	25.96	0	0
Zoom setting 1	25.98	0.01432	0.0005516
Zoom setting 2	26.35	0.3889	0.01498
Zoom setting 3	26.55	0.5910	0.02276
Zoom setting 4	26.19	0.2245	0.008649
Zoom setting 5	26.28	0.3165	0.01219
Angle setting 1	25.52	-0.4395	-0.01693
Angle setting 2	26.08	0.1205	0.004643
Angle setting 3	26.04	0.07817	0.003011

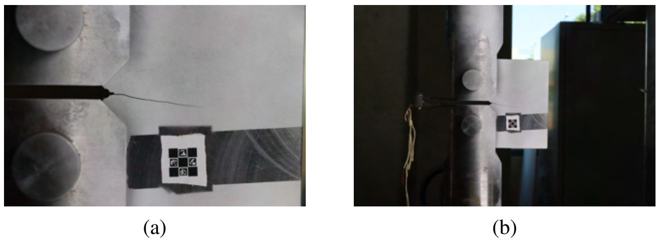

(a) (b)

Fig. 22: Images of the same crack with different zoom settings, image (a) is the ’standard’ zoom setting also used primarily for the training of the network, called Setting 0; image (b) is further zoomed out.

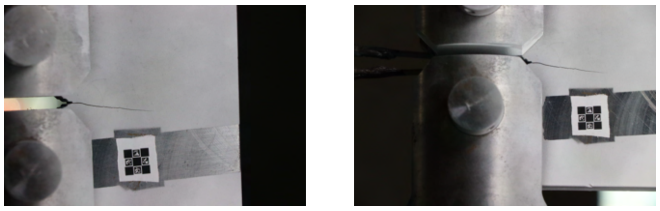

(a) (b)

Fig. 23: Images of the same crack but at different angles, image (a) has an angle from the side; image (b) is taken from below.

References

[1] Y. Z. M. Wang and L. Wang, "An overview of intelligent inspection technology for steel structures," *Automation in Construction*, pp. 125–143, 2020. [Online]. Available: https://doi.org/10.1016/j.autcon.2020.103142

[2] M. A. Zidan and E. Abdel-Rahman, "An intelligent system for steel structures inspection using ai techniques," *Ain Shams Engineering Journal*, no. 3, pp. 717–729, 2020. [Online]. Available: https://doi.org/10.1016/j.asej.2020.05.007

[3] Q. Han, X. Liu, and J. Xu, "Detection and location of steel structure surface cracks based on unmanned aerial vehicle images," *Journal of Building Engineering*, vol. 50, p. 104098, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352710222001115

[4] W. T. O. T. B. to Trade (TBT) Committee, "Standard test method for measurement of fatigue crack growth rates."

[5] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything," *arXiv:2304.02643*, 2023.

[6] D. L. N. v. d. B. Andrii Kompanets, Remco Duits and H. B. Snijder, "Segmentation tool for images of crack."

[7] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[8] M. Sniedovich, "Dijkstra's algorithm revisited: the dynamic programming connexion," *Control and Cybernetics*, vol. 35, no. 3, pp. 599–620, 2006.

[9] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec 1959. [Online]. Available: https://doi.org/10.1007/BF01386390