Project No. R924070

Development of UAS-enabled Bridge Deck Inspection System from Investigation to Implementation

Final Report
Jan 1, 2024 – June 12, 2026

Report Submitted to
Research Bureau
New Mexico Department of Transportation
7500B Pan American Freeway NE
PO Box 94690
Albuquerque, NM 87199-4690

Prepared by

Pouya Almasi
Graduate Research Assistant
Department of Civil and Environmental Engineering
New Mexico State University
&
Qianyun Zhang
Assistant Professor
Department of Civil and Environmental Engineering
New Mexico State University
&
Su Zhang
Assistant Professor
Department of Geography and Environmental Studies
University of New Mexico

June 1st, 2026

SUMMARY PAGE

<table>
<tr><td>1. Report No.</td><td colspan="3">2. Recipient’s Catalog No.</td></tr>
<tr><td>3. Title and Subtitle
Development of UAS-enabled Bridge Deck Inspection System from Investigation to Implementation</td><td colspan="3">4. Report Date
05/31/2026</td></tr>
<tr><td>5. Author(s):
Pouya Almasi, Qianyun Zhang, Su Zhang</td><td colspan="3">6. Performing Organization Report No.</td></tr>
<tr><td rowspan="2">7. Performing Organization Name and Address
New Mexico State University
Civil Engineering Department
3035 S Espina St.
Las Cruces, NM 88003-8001</td><td colspan="3">8. Performing Organization Code</td></tr>
<tr><td colspan="3">9. Contract/Grant No. R924070</td></tr>
<tr><td rowspan="2">10. Sponsoring Agency Name and Address

Research Bureau, NMDOT
7500B Pan American Freeway
PO Box 94690
Albuquerque, NM 87199-4690</td><td colspan="3">11. Type of Report and Period Covered

Final Report
Jan 1, 2024 – June 12, 2026</td></tr>
<tr><td colspan="3">12. Sponsoring Agency Code
NMDOT</td></tr>
<tr><td colspan="4">13. Supplementary Notes
None</td></tr>
<tr><td colspan="4">14. Abstract
We introduce a pilot study that is geared towards inaugurating a UAS-centric bridge inspection program, operating on a component-level approach, with the overarching goal of enhancing the caliber of bridge inspection methodologies within the confines of New Mexico. The envisioned program encompasses the formulation of UAS-based inspection strategies, the establishment of frameworks for data interpretation, and the development of specialized software implementations tailored to individual bridge components as delineated in the National Bridge Inventory (NBI). To achieve automated data acquisition, cutting-edge UAS platforms equipped with high-resolution sensors will be deployed. Moreover, innovative algorithms for image processing and machine learning will be investigated to facilitate the semi-automated or fully automated analysis of the amassed data. Intuitive software applications, constructed upon these novel algorithms, will be devised to facilitate practical implementation. In addition, a dedicated workshop will be orchestrated for the personnel of the New Mexico Department of Transportation (NMDOT), providing a platform to showcase the tangible research outcomes stemming from the proposed project. This strategic initiative serves to broaden the reach and amplify the impact of the research endeavors undertaken.</td></tr>
<tr><td colspan="2">15. Key Words</td><td colspan="2">16. Distribution Statement

Available from NMDOT Research Bureau</td></tr>
<tr><td>17. Security Classification of this Report
None</td><td>18. Security Classification of this page
None</td><td>19. Number of Pages</td><td>20. Price
N/A</td></tr>
</table>

PREFACE

NOTICE

The United States government and the State of New Mexico do not endorse products or manufacturers. Trade or manufactures' names appear herein solely because they are considered essential to the object of this report. This information is available in alternative accessible formats. To obtain an alternative format, contact the NMDOT Research Bureau, 7500B Pan American Freeway NE, PO Box 94690, Albuquerque, NM 87199-4690, (505)-841-9145

DISCLAIMER

This report presents the results of research conducted by the authors and does not necessarily reflect the views of the New Mexico Department of Transportation. This report does not constitute a standard or specification.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TASK 1: Personnel Training and Certification

In this Section, a guideline is developed for either First-time Pilots or Existing Part 61 Certificate Holders. The first part of this guideline is for first-time pilots, and the second part of this guideline is for existing part 61 certificate holders.

1.1 First-time Pilots

1.1.1 Eligibility

1) Be at least 16 years old
2) Be able to read, speak, write, and understand English
3) Be in a physical and mental condition to safely fly a drone
4) Pass the initial aeronautical knowledge exam: "Unmanned Aircraft General –Small (UAG)"

1.1.2 Steps to Become a Drone Pilot

Step 1: Obtain an FAA Tracking Number (FTN) by creating an Integrated Airman Certification and Rating Application (IACRA) profile prior to registering for a knowledge test.10

Step 2: Schedule an appointment with an FAA-approved Knowledge Testing Center. Be sure to register for the ***"Unmanned Aircraft General –Small (UAG)"*** test and bring a government-issued photo ID to your test.

Step 3: Study UAG-related knowledge to pass the test. Knowledge test topic areas include:

1) Applicable regulations relating to small unmanned aircraft system rating privileges, limitations, and flight operation
2) Airspace classification and operating requirements, and flight restrictions affecting small unmanned aircraft operation
3) Aviation weather sources and effects of weather on small unmanned aircraft performance
4) Small unmanned aircraft loading and performance
5) Emergency procedures
6) Crew resource management
7) Radio communication procedures
8) Determining the performance of small unmanned aircraft
9) Physiological effects of drugs and alcohol
10) Aeronautical decision-making and judgment
11) Airport operations
12) Maintenance and preflight inspection procedures
13) Operation at night

Useful downloadable material to be reviewed:

1) Unmanned Aircraft General Sample Questions (download here)

2) Remote Pilot –Small Unmanned Aircraft Systems Study Guide (download here)
3) Title 14 CFR Part 107 (download here)
4) Advisory Circular AC 107-2A (download here)
5) Airman Knowledge Testing Supplement for Sport Pilot, Recreational Pilot, Remote Pilot, and Private Pilot (FAA-CT-8080-2H) (download here)
6) Title 14 CFR Part 47 (download here)
7) Title 14 CRF Part 48 (download here)
8) Remote Pilot Airman Certification Standards (download here)

Step 4: Pass the initial aeronautical knowledge test: "Unmanned Aircraft General –Small (UAG)".

Step 5: Complete FAA Form 8710-13 for a remote pilot certificate (FAA Airman Certificate and/or Rating Application) using the electronic FAA Integrated Airman Certificate and/or Rating Application system (IACRA)*

1) Login with username and password
2) Click on "Start New Application": Application Type "Pilot", Certifications "Remote Pilot", Other Path Information, Start Application
3) Follow application prompts
4) When prompted, enter the 17-digit Knowledge Test Exam ID (Note: it may take up to 48 hours from the test date for the knowledge test to appear in IACRA)
5) Sign the application electronically and submit it for processing.

Step 6: A confirmation email will be sent when an applicant has completed theTSAsecurity background check. This email will provide instructions for printing a copy of the temporary remote pilot certificate from IACRA as shown in **Figure 1**.

i. UNITED STATES OF AMERICA
DEPARTMENT OF TRANSPORTATION – FEDERAL AVIATION ADMINISTRATION
ii. TEMPORARY AIRMAN CERTIFICATE
iii. CERTIFICATE NO. PENDING

THIS CERTIFIES THAT iv. YANGJIAN XIAO
v. 201 HERNANDEZ HALL
3035 S ESPINA STREET
LAS CRUCES NM 88003

DATE OF BIRTH	HEIGHT	WEIGHT	HAIR	EYES	SEX	NATIONALITY vi.
9/26/1989	66 IN.	140	BLACK	BROWN	M	CHINA

ix. has been found to be properly qualified and is hereby authorized in accordance with the conditions of issuance on the reverse of this certificate to exercise the privileges of
REMOTE PILOT

RATINGS AND LIMITATIONS
xii. SMALL UNMANNED AIRCRAFT SYSTEM

xiii.

THIS IS ☒ AN ORIGINAL ISSUANCE ☐ A REISSUANCE OF THIS GRADE OF CERTIFICATE
DATE OF SUPERSEDED AIRMAN CERTIFICATE

BY DIRECTION OF THE ADMINISTRATOR
EXAMINER'S DESIGNATION NO. OR INSPECTOR'S REG. NO.

x. DATE OF ISSUANCE 03/07/2023 01:58:30 PM
xi. SIGNATURE OF EXAMINER OR INSPECTOR
MANAGER, AIRMEN CERTIFICATION BR
IACRA E-SIGNED APPLICATION
DATE DESIGNATION EXPIRES

vii. AIRMAN'S SIGNATURE

FAA Form 8060-4 (8-79) USE PREVIOUS EDITION Application Number: 3694081 IACRA Equivalent

XIV. CONDITIONS OF ISSUANCE

This is an interim certificate issued subject to the approval of the Federal Aviation Administration pending the issuance of a certificate of greater duration. It becomes void –

1. Upon the receipt of a certificate of greater duration to replace it;
2. Upon a finding by the FAA that an error has been made in its issuance;
3. Upon a finding by the FAA that it was issued illegally or as the result of fraud or mis-representation;
4. Upon the refusal or failure by the holder to accomplish a flight check by a Flight Standards Inspector if so requested; and
5. In any case, at the expiration of 120 days from date of issuance.

Figure 1. Temporary remote pilot certificate

Step 7: A permanent remote pilot certificate (as shown in **Figure 2**) will be sent via mail once all other FAA-internal processing is complete.

Figure 2. Permanent remote pilot certificate

Step 8: Have your Remote Pilot Certificate available whenever you fly your UAS.

1.2 Existing Part 61 Certificate Holders

1.2.1 Eligibility

1) Must hold a pilot certificate issued under14CFRpart 61

2) Must have completed a flight review within the previous 24 months

1.2.2 Steps to Become a Drone Pilot

Step 1: Create an account, or log into your existing account, on the FAA Safety Team (FAASTeam) website.

Step 2: Complete the Part 107 Small UAS Initial (ALC-451) online training course. The course will cover these topic areas:

1) Applicable regulations relating to small unmanned aircraft system rating privileges, limitations, and flight operation
2) Effects of weather on small unmanned aircraft performance
3) Small unmanned aircraft loading and performance
4) Emergency procedures
5) Crew resource management
6) Determining the performance of small unmanned aircraft
7) Maintenance and preflight inspection procedures
8) Operation at night

Step 3: Create an account, or log into your existing account, in IACRA.

Step 4: Complete Form 8710-13 for a remote pilot certificate (FAA Airman Certificate and/or Rating Application) in IACRA.

1) Login with username and password
2) Click on "Start New Application" and 1. Application Type "Pilot", 2. Certifications "Remote Pilot", 3. Other Path Information, 4. Start Application
3) Follow application prompts
4) Sign the application electronically and submit for processing.

Step 5: Make an appointment with one of the following entities to validate your identity. Bring your completed Form 8710-13, proof of your current flight review, photo ID, and your online course completion certificate.

1) At an FAA Flight Standards District Office (FSDO)
2) With an FAA-designated pilot examiner (DPE)
3) An airman certification representative (ACR)
4) An FAA-certificated flight instructor (CFI)*

Please note: *CFIs cannot issue temporary certificates. They can process applications for applicants who do not want a temporary certificate.13

Step 6: The representative will sign your application and issue you a temporary airman certificate.

1) You'll receive your permanent certificate via U.S. mail within several weeks.

Step 7: Have your Remote Pilot Certificate available whenever you fly your UAS.

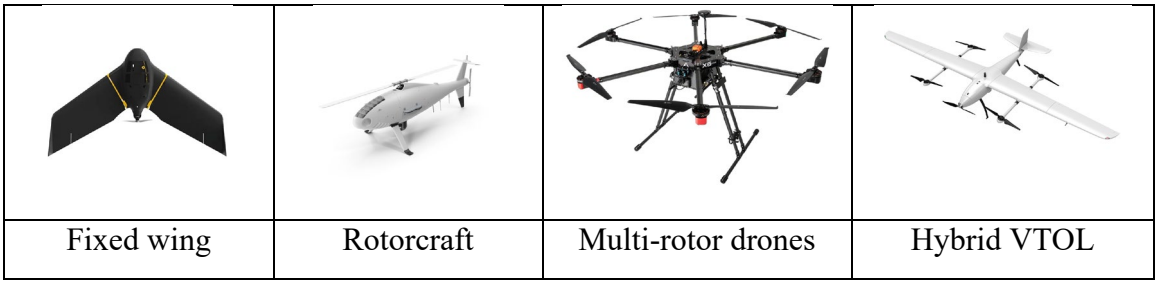
1.2.3 Requirements for Remote Pilot Certificate

1) Must be easily accessible by the remote pilot during all UAS operations
2) Certificate holders must complete an online recurrent training every 24 calendar months to maintain aeronautical knowledge recency

TASK 2: Hardware Selection

2.1 UAS Platform Selection

Before integrating UAVs into the airspace, a few aspects, including equipment features, pilot protocols, object qualities, surroundings, and safety rules, must be considered. It is crucial to have a thorough awareness of these aspects to allow safe and effective operations [1]. Some equipment-related aspects influence UAV performance including aerial system size and design, payload capacity and compatibility, battery capacity, control range distance, and duration for safe flight [2]. The typical UAV is composed of a frame, motors, control unit, onboard sensors, communication system, and power supply. Many UAVs display a dual tube substructure to make it easier to mount various payloads [3]. Fixed-wing, rotorcraft, multi-rotor drones, and hybrid vertical take-off and landing (VTOL) vehicles are the four common UAV designs [4]. **Figure 3** shows examples of these different types of UAVs.



Fixed wing	Rotorcraft	Multi-rotor drones	Hybrid VTOL

Figure 3. Common UAV types

When delving into operational attributes and various airframes, fixed-wing drones mirror traditional aircraft, typically boasting larger sizes that efficiently cover extended distances. In comparison, rotorcraft UAVs employ rotating propellers integrated into their framework, akin to helicopters. Single-rotor drones exhibit enhanced efficiency over their multi-rotor counterparts. Functioning as compact helicopters powered by either gas or electricity, these single-rotor models represent a specialized niche. Among these, the multirotor UAV emerges, an evolution featuring multiple propellers extending from the core to amplify flight capabilities. This category finds common utility in applications such as aerial photography and surveillance. Variants include the Tricopter (with three rotors), Quadcopter (with four rotors), Hexacopter (with six rotors), and Octocopter (with eight rotors). Multirotor UAVs excel in intricate 3D mapping due to their agility and stability. Conversely, rotorcrafts demand more finesse in manual piloting compared to multirotors, thanks to their enhanced controllability, increased lift capacity, and built-in redundancy in case of motor malfunction. Measurement errors may arise from the UAV's instability or heightened maneuverability. Lastly, hybrid VTOL UAVs amalgamate fixed-wing and multirotor configurations, initially ascending vertically before transitioning into horizontal flight [2].

Several additional factors come into play when choosing the right platform for inspections. For close-range assessments, the chosen platform must possess the ability to maintain stability, even in the face of strong gusts. Typically, optimizing flight duration is paramount for various applications, often hinging on payload capacity and battery performance. A larger payload capacity allows for the accommodation of more sensors, but this can sometimes compromise overall flight time. Conversely, camera resolution and the desired quality of surveys can also influence flight duration. This is due to the correlation between higher-resolution cameras and the potential for shorter flight paths [2]. The choice of the appropriate UAV platform and sensors has proven to be a difficult problem due to UAV performance requirements related to flights close to the bridge structure (e.g., turbulent flow characteristics around the bridge) and terrain characteristics (e.g., surface roughness, temperature, and humidity) [5]. This includes positioning and maneuvering the UAV around or under bridges (operations prohibited by GPS) [6] and the stability of the platform in windy situations, where turbulence and other aerodynamic phenomena cause unpredictable wind effects [7]. The main difficulties in maximizing the UAV for bridge inspections are striking a balance between payload capacity, endurance, vehicle stability, and navigational capabilities. **Figure 4** indicates the parameters for UAV platform selection. A vehicle with a stabilizing gimbal that can change the camera pointing angle to any vertical angle, a camera with optical zoom for capturing high-resolution imagery while at a safe standoff distance, a vertical takeoff and landing capability, and the capacity to hover in place during the flight are examples of parameters and vehicle characteristics appropriate for bridge inspections. In addition, **Figure 5** shows some of the UAV platforms used in former studies. **Figure 6** indicates the cost-endurance of these different UAV platforms.

Figure 4. Considerable parameters for UAV platform selection

				
UAV	DJI Mavic	Aurelia X6 Standard LE	DJI Phantom 4	senseFly Albris
Price	$2700	$5700	$3000	$2000
Maximum Endurance	31 minutes	45 minutes	30 minutes	22 minutes
Payload Capacity	1 kg	5 kg	1 kg	N/A
				
UAV	3DR Solo	3DR Iris	DJI Inspire 1 Pro	Bergen hexacopter
Price	$1000	$750	$3900	$6000
Maximum Endurance	15 min	22 min	18 min	30 min
Payload Capacity	1.5 kg	0.4 kg	3.4 kg	5 kg

Figure 5. Examples of UAVs with some specs

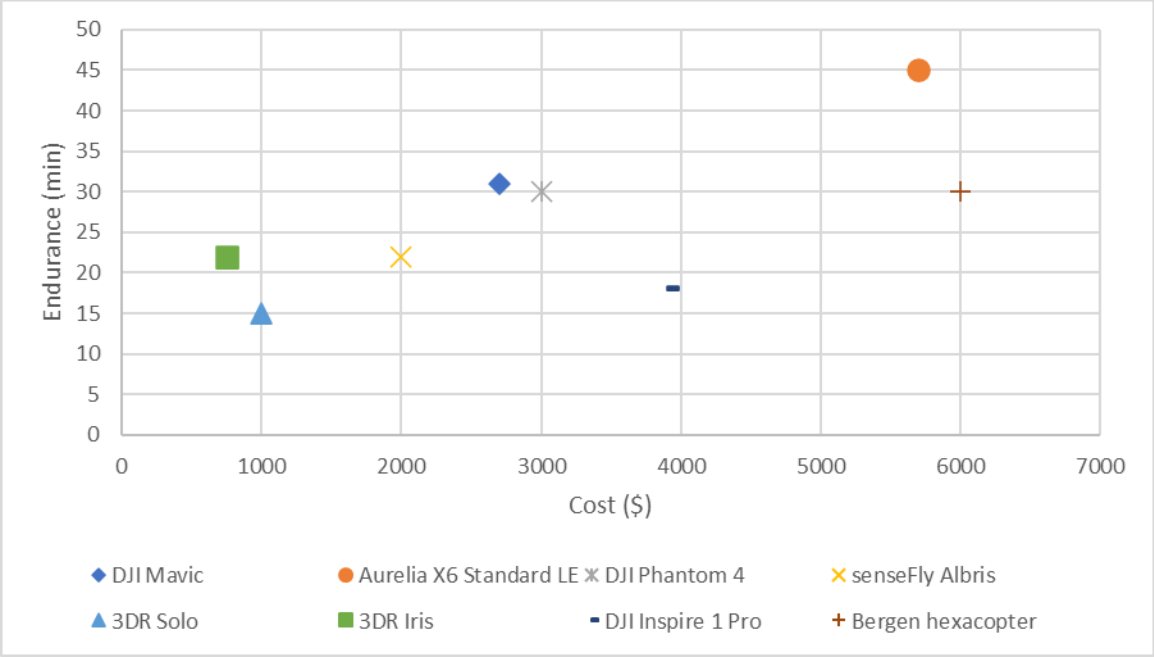


Figure 6. Cost-endurance relation of different UAV platforms

2.2 Payload Selection

The choice of the best sensors for bridge inspection depends on several factors, including cost, flight time, mission objectives, the UAV's payload capacity, and navigational needs [6]. These factors are shown in **Figure 7**.

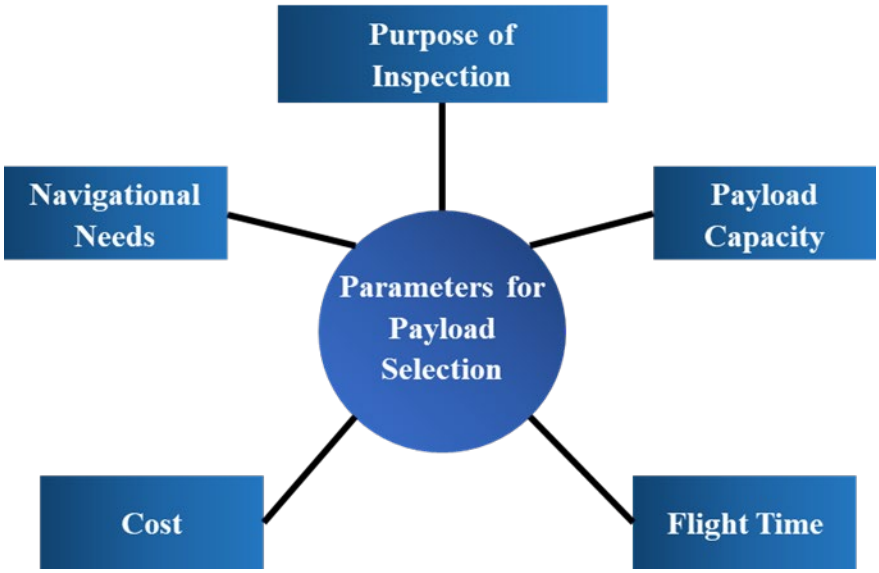


Figure 7. Considerable parameters for payload selection

The most important factor is the purpose of the inspection. Without considering the purpose of inspection, a suitable payload may not be chosen and consequently, it will lead to some problems in other steps such as low data quality or inappropriate data collection. Five common purposes are

defined in this study: crack detection, delamination, fatigue, 3D modeling, and corrosion. These purposes of inspection are discussed briefly below with the proper sensors.

- Crack: Most of the studies mentioned crack detection as the basic and major application of bridge inspection using UAVs [3]. There are two primary steps in the image-based surface crack assessment method. Crack detection comes first, with the goal of removing noise and extracting cracked objects from the images. The extraction of crack edges and the calculation of crack characteristics, such as crack width and length, make up the second stage of crack assessment [2]. For this purpose, mostly RGB cameras are used to find cracks in the surface of bridges. Using optical cameras, the UAVs can take high-quality pictures from the bridge's difficult-to-reach regions. It is worth mentioning that due to the wide use of this purpose, the general framework of this study is generated for crack detection, but it can be applied for other purposes too with some adjustments.
- Delamination: Deck delamination, also known as horizontal debonding in the deck's subsurface, is frequently a sign that the deck reinforcement has deteriorated due to corrosion. The shape and depth of delamination, environmental factors like air temperature and solar intensity, which introduce feature variation of the same delamination, and surface textures like cracks, color differences, patching, and road painting, which add external noise, are the current challenges for the purpose of delamination profiling through thermography [8].
- Fatigue: Fatigue cracks can have lengths less than 7 mm and diameters as small as 0.1 mm, and they are exceedingly difficult to discern. Inaccessible areas such as huge cross frames, welded stiffeners, or other complex geometries are typically where fatigue cracks develop in the superstructure. Commonly, RGB and IRT cameras are utilized to identify fatigue cracks. The effectiveness of UAV-based fatigue crack detection is greatly influenced by the platform that is used, the environment, and the lighting [9].
- 3D Modeling: Bridge managers can view geometric data, such as damage location, and surface condition, such as damage kind and amount, by using 3D models of the structures, which provide a base from which damage information can be compared. To create 3D models, RGB cameras and LiDAR sensors can be used [10]. Photogrammetry creates 3D points from a set of 2D photos collected from various angles and positions all around the structure, as opposed to LiDAR, which often contains more 3D points. Photogrammetry has a higher processing cost and lower accuracy than LiDAR because it compares image attributes to build the 3D points. However, UAV-based LiDAR systems need expensive LiDAR sensors and GPS systems, which reduce battery life by adding more payload to the system, whereas photogrammetry merely needs an optical sensor.
- Corrosion: A positive charge is released during the electrochemical process of corrosion, which results in the formation of a stable compound. Despite some corrosion on the underlying metal components, such as the steel reinforcement used in bridge concrete, the surface of steel bridges experiences a great deal of corrosion. The most popular cameras for detecting corrosion are RGB and IRT cameras. Although infrared thermography is a promising technique for measuring, mapping, and detecting corrosion, more study is required before it can be used perfectly in the field [11].

Table 1 gives a summary of 3 commonly used payloads and some other information that is effective for the suitable payload selection. It is worth mentioning that if needed, using multi-sensors is possible if they are compatible with the UAV platform.

Payload Type	Weight Range (kg)	Mission	Limitations
Visual Camera	0.1 - 1	-Crack Detection -Fatigue -Delamination -Corrosion -3D Modeling	- Vibration and wind effect - Lightning condition - GPS deprived navigation
IRT Camera	0.2 – 1.5	-Fatigue -Delamination	- Low pixel resolution - Inspection time affects the results
LiDAR Sensor	1.3 – 2.8	-3D Modeling	- High weight - High price

Table 1. Specifics of different payloads and their limitations

2.3 Recommendations

UAV platform selection:

- Type of Drone: Select the drone based on the application. For bridge inspection, a multi-rotor UAV is recommended.
- Stability: Prioritize drones with advanced stabilization systems to handle varying wind conditions and ensure precise data collection.
- Payload Capacity and Compatibility: Choose a drone with sufficient payload capacity to accommodate necessary sensors and cameras without compromising flight time.
- Flight Duration: Balance payload capacity and battery performance to achieve a suitable flight duration that covers the inspection area effectively. An extra battery is recommended.
- Cost Efficiency: Evaluate the costs associated with the drone, additional equipment, and maintenance to align with your budget.

Payload Selection:

- Camera or Sensor Type: Select cameras or sensors that align with the inspection objectives. Consider RGB cameras for visual inspections, thermal cameras for detecting structural anomalies, or LiDAR for generating detailed 3D models.
- Resolution and Accuracy: Select cameras or sensors with appropriate resolution and accuracy levels to capture the required level of detail for the inspection.

- Integration: Ensure that the chosen payload can be seamlessly integrated with your drone's existing systems and can be easily attached and removed when needed.

- Cost-Effectiveness: Balance the capabilities of the payload with its cost to ensure it meets your inspection requirements within your budget.

TASK 3: Pre-flight Preparation for Data Collection

3.1 Flight Path Planning

A well-planned flight path is of paramount importance when using UAVs for inspection operations. To unlock the full potential of the UAV, the mission must be carefully designed to encompass all inspection targets [12], [13]. However, path planning for UAV-based bridge inspection presents challenges in finding the optimal or near-optimal path. The flight path comprises a set of camera positions from which images will be captured. These camera positions are determined by their horizontal and vertical distance from the object, as well as the angle of the camera. In the following two subparts of this section, we will delve into discussions on camera positions and flight path planning, exploring the crucial aspects of these elements in UAV-based bridge inspection [14].

In the process of selecting camera positions, one of the key considerations is the Ground Sampling Distance (GSD). The Ground Sampling Distance refers to the distance between two consecutive pixel centers measured on the ground. It plays a crucial role in determining the spatial resolution of the image and the level of visible details. A larger GSD value corresponds to lower spatial resolution, resulting in fewer visible details in the captured images. **Figure 8** provides a visual representation of GSD and its associated parameters, illustrating its significance in the context of UAV-based bridge inspection.

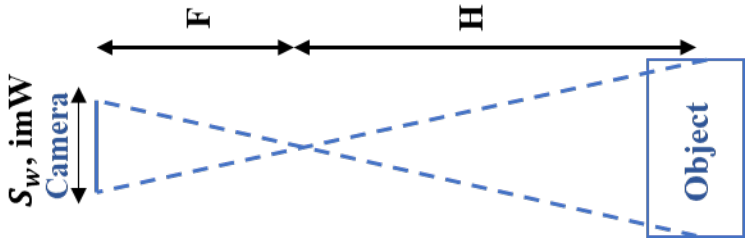


Figure 8. GSD example with related parameters

As shown in **Figure 8**, *H* is the flight height in meters, *imW* is the image width in pixels, *F* is the real focal length of the camera in millimeters, *SW* is the sensor width in millimeters, and GSD is the ground sampling distance in centimeters/pixels.

The equation for calculating the GSD is:

$$GSD = \frac{S_W \times H}{F_R \times imW} \times 100 \qquad (1)$$

It is important to decide on the GSD value before starting the image acquisition in order to adjust the flight height and the camera specifications to the project requirements. From equation (1), the required flight height can be calculated if the GSD is defined for an inspection as below:

$$H = \frac{GSD \times F_R \times imW}{S_w \times 100} \quad (2)$$

Also, by knowing the image height width and height of the single image footprint on the ground (distance covered on the ground by one image in width and height directions) can be calculated as below:

$$D_w = GSD \times imW \quad (3)$$

$$D_H = GSD \times imH \quad (4)$$

where imH is the image height in pixels, and D_w, D_H are the width and height of a single image footprint on the ground in meters respectively.

The selection of GSD and camera positions depends on the specific purpose of the inspection and the characteristics of the payload. Debus et al. propose three distinct levels of interest for conducting the inspection, [12]. 2.0 mm/pixels, 1.0 mm/pixels, and 0.1 mm/pixels are defined as level 1(for rough geometry), level 2 (for detailed geometry), and level 3 (for crack detection) of interest respectively, which provides valuable guidance for tailoring the GSD and camera positions to effectively meet the inspection objectives. Also, according to the Specifications for the National Bridge Inventory 2022 (SNBI), the following quantitative standards are considered to categorize the cracks by their width:

- Insignificant - crack width less than 0.004 inches (prestressed) or 0.012 inches (reinforced), or medium width cracks that have been sealed.
- Medium - crack width ranging from 0.004 – 0.009 inches (prestressed) or 0.012 to 0.05 inches (reinforced).
- Wide - crack width wider than 0.009 inches (prestressed) or 0.05 inches (reinforced).

While the defined levels of interest are valuable as a starting point, it is crucial to recognize that they may need to be adjusted based on the specific requirements of each inspection task. Different cases may demand varying levels of interest to effectively address the inspection objectives. Additionally, even when flying at a constant height, the images captured during the project may exhibit variations in GSD. These discrepancies arise due to differences in terrain elevation and changes in the camera angle during image capture. To ensure comprehensive coverage and accurate data collection, it is generally recommended to have at least a 50% overlap of images between consecutive camera positions, as suggested by various studies [15]. This overlapping ensures that critical details are captured redundantly, minimizing the risk of missing essential information.

Once a set of points has been chosen, taking into account the previously mentioned parameters, the next crucial step is to determine the most optimum or near-optimum flight path. This task poses a challenging optimization problem, as it involves minimizing flight time (energy) or path while considering all relevant parameters.

3.2 Camera Calibration

Camera calibration is an essential step to extract metric data from 2D photos in 3D computer vision. Over the years, numerous studies have explored camera calibration, initially in the field of photogrammetry and more recently in the computer vision community [16], [17]. In aerial images, pre-calibration or on-the-job calibration is frequently used to handle camera parameters, such as intrinsic parameters and lens distortion coefficients. The goal of camera calibration is to establish the relationship between the 3D world coordinates of the object and their corresponding 2D image coordinates, forming the projection matrix.

We assume a point on the object such as X_w and the projection of this point in the captured image such as U. Coordinates of these points are as shown below:

$$X_w = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} \quad (5)$$

$$U = \begin{bmatrix} u \\ v \end{bmatrix} \quad (6)$$

Where x_w, y_w, z_w are the coordinates of a known object in millimeters or inches, and u, v are the coordinates of the projection of that known point in the captured image in pixels. This can be done for every corresponding point of object and captured image. For each corresponding point i in the scene and image, we get a mapping from the point in 3D coordinates to the image coordinates in 2D using a projection matrix:

$$\begin{bmatrix} u^{(i)} \\ v^{(i)} \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{134} \end{bmatrix} \begin{bmatrix} x_w^{(i)} \\ y_w^{(i)} \\ z_w^{(i)} \\ 1 \end{bmatrix} \quad (7)$$

As shown in equation (7), the only unknown matrix is the projection matrix which should be estimated.

With the fundamental steps for camera calibration, specifically for reference object-based calibration, various patterns and benchmarks can be utilized to perform the calibration. In this study, the commonly used and straightforward checkboard pattern is employed. **Figure 9** displays the two checkboard patterns utilized for camera calibration in this study. Checkboard patterns are selected because of their simplicity and almost all the calibration tools are compatible with this type of benchmark. The control points for this pattern are the corners that lie inside the checkerboard. Because corners are extremely small, they are often invariant to perspective and lens distortion. The calibrator apps can also detect partial checkerboards, which can be useful when calibrating cameras with wide-angle lenses. A checkerboard should contain an even number of squares along one edge and an odd number of squares along the other edge, with two black corner squares along one side and two white corner squares on the opposite side. This enables the app to determine the orientation of the pattern and the origin. The calibrator assigns the longer side as the x-direction. A square checkerboard pattern can produce unexpected results for camera extrinsics.

In general, the checkboard size will not affect the camera calibration process or intrinsic and extrinsic camera parameters very much in mathematical representation. However, it is important

practically but, if their size is within the recommended ranges for calibration tools (such as 15, 20, 30, 40, 50, or 60 millimeters) considering the distance from the target, it will not affect the crack detection. If the check squares are very small, probably the corners of the squares will not have appropriate quality and that will make the data "noisier". It is worth noting that more squares (more corners between squares) will give better results since there will be a more overdetermined system of equations to be solved [18].

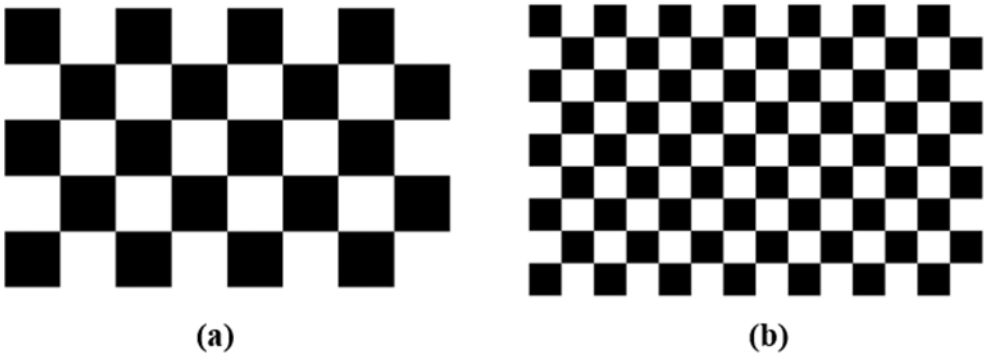


Figure 9. Checkboard patterns for camera calibration. (a) 297 mm×400 mm with check square size of 50mm (b) 400 mm×600 mm with check square size of 40mm

For the purpose of camera calibration in this study, the MATLAB camera calibration toolbox will be used which uses a robust feature detection algorithm based on Zhang's method [19], [20], and this approach will help to reduce potential errors related to feature point detection. By using this method and the checkboard benchmarks shown above, the internal and external parameters of the camera will be determined. The first important parameter is the reprojection error which is the distance between a pattern key point detected in a calibration image, and a corresponding world point projected into the same image. The acceptable and recommended mean reprojection error is less than 1 pixel but an error less than 0.5 pixel is better for a good alignment [21]. Another parameter to consider is the focal length in the x and y directions (f_x, f_y) which is in pixels, and the relation between this focal length and real focal length (F) is shown below:

$$f_x = F \times s_x \quad (8)$$

$$f_y = F \times s_y \quad (9)$$

Where F is the focal length in millimeters, and s_x, s_y are the number of pixels per millimeter in the x- and y-direction respectively.

In the methodology proposed in this study, there are two kinds of camera calibrations. In-lab camera calibration is a one-time calibration, and it is beneficial for flight path planning based on the intrinsic and extrinsic parameters of the drone's camera where this calibration gives a better understanding of the required flight plan considering different camera positions (different heights and angles) and their corresponding reprojection errors and distortion. On the other hand, the results of the on-site calibration will give the parameters considering the real-world conditions for the bridge inspection and due to the stability issues with the drones, these parameters from on-site calibration will result in calculating the exact GSD for the collected data which is crucial for the data processing and crack detection. While the required flight height was determined by using GSD from the previous section, considering the stability issue with UAVs, there would be a

tolerance in the on-site flight. But, using the focal length in pixels from calibration and by knowing the real focal length of the camera in millimeters, the exact GSD for each image can be easily found by using equations 8 and 9 which would be beneficial for more accurate crack detection. Other than stability, another source of error could be lighting conditions which have been considered and explained in the experiments. Also, some other potential sources of error could be lens distortion, human error, and other environmental factors. To mitigate these errors, as will be presented in the experiments, multiple images have been collected.

Finally, the other important parameter is the radial distortion in the x (RD_x) and y (RD_y) directions. Radial distortion is the displacement of image points along radial lines extending from the principal point and it occurs when light rays bend more near the edges of a lens than they do at its optical center.

3.3 Case Studies and Results

3.3.1 Inspection Purpose and Bridges

To evaluate the feasibility of the proposed framework, two case studies have been carried out. The purpose of the inspection is generally to detect cracks on the top surface and side surface of the bridge decks.

Two bridges have been selected for the case study. The first bridge is located at 4.8 Mi N of Sierra C/L, New Mexico, United States (NBI bridge number: 01791). This bridge is in fair condition and consists of 5 simple spans at 39’ each, 6 steel girders per span, full-height concrete abutments with concrete wingwalls, concrete pier caps on concrete pier walls, and a CIP concrete deck, as shown in **Figure 10**. The second bridge is in satisfactory condition according to the reports of the New Mexico Department of Transportation and is located at 1.9 Mi W of NM-28/NM-359, New Mexico, United States (NBI bridge number: 06255). The bridge consists of 8 spans, 2 units of 4 continuous spans at 54ft, 69ft-5in, 69ft-5in, and 54 ft, 5 rolled steel girders per span, concrete stub abutments, concrete pier caps on steel piles, and CIP concrete deck, as shown in **Figure 11**.

Figure 10. Profile view of the first bridge facing north (NBI number: 01791)

Figure 11. Profile view of the second bridge facing south (NBI number: 06255)

3.3.2 In-lab Calibration

In-lab calibration using various camera positions would be helpful to understand the camera parameters which would be beneficial for flight path planning. Considering the flight height range based on inspection purposes and GSD, various images are captured from different heights, angles,

and directions. The results of the camera calibration for these sets of images will lead to better flight path planning and a better selection of camera positions for on-site bridge inspection.

The in-lab calibrations take place in a parking lot at New Mexico State University. During this calibration process, three distinct sets of images are captured from various heights, angles, and directions. Each set contains 36 photos with different heights and angles (shown in **Table 2**), resulting in a total of 108 different images used for camera calibration. The calibration results are then categorized into three parts, corresponding to the camera's direction, namely south, east, and north. In **Figure 12**, an example of captured images is depicted, along with their detected and projected points used for in-lab calibration. This approach of using different groups of images in three separate sets makes it more convenient for an inspector or pilot to create a flight plan, taking into account the calibration results from these different image groups. This division aids in tailoring the flight plan according to specific camera orientations and ensures accurate imaging during the inspection process.

Heights (m)	Angles (°)
0.5, 1, 1.5, 2, 2.5, 3	15, 30, 45, 60, 75, 90

Table 2. Heights and angles for in-lab calibration for each direction

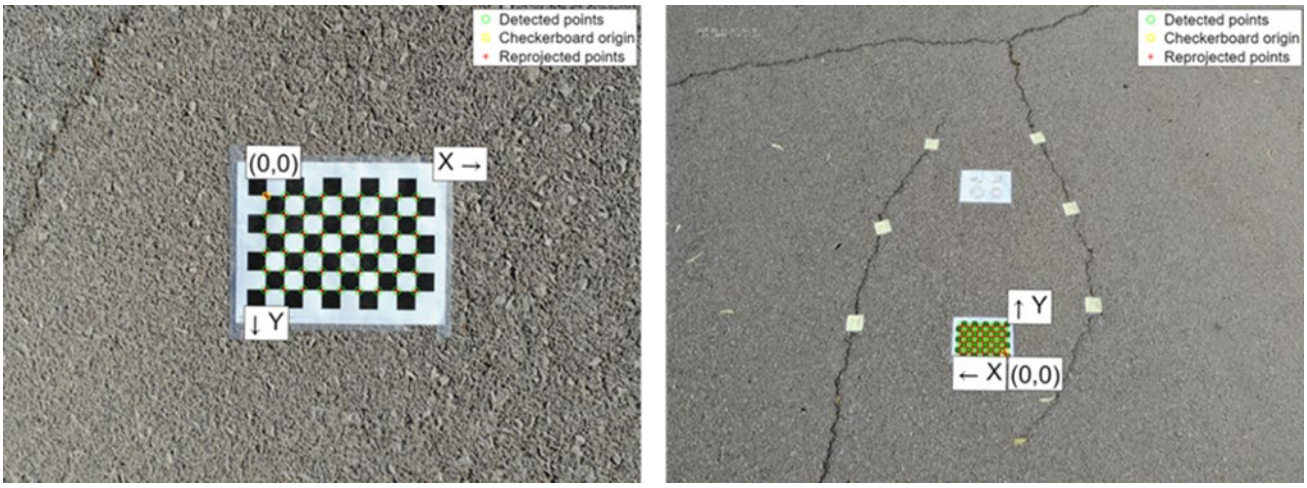


Figure 12. Different pictures from in-lab calibration with shown detected and reprojected points

Figures **13**, **14**, and **15** display the reprojection errors, overall mean error, and trendline obtained from data captured by the camera in the eastward, northward, and southward directions respectively. The overall mean reprojection error, as well as the focal length in the x and y directions, and radial distortion in the x and y directions are calculated based on equation 7, and by using MATLAB camera calibration, and presented in **Table 3** for all three groups of images. When comparing the overall mean reprojection error with the individual error of each image within each group, it is observed that 41.7%,33.3%, and 36.1% of the images have a higher error than the mean error for eastward, northward, and southward groups respectively. Also, the images captured from the south direction have a noticeably lower overall mean error. It is worth mentioning that 2

images from the east side were rejected during calibration data processing due to the high reflection of sunlight on the benchmark. This analysis provides valuable insights into the accuracy and consistency of the calibration results for this set of images. By considering these results, the southward which is in the direction of the back to the sun has less mean reprojection error than the other sides and in this set of images, pictures with a height between 2m to 3m have fewer reprojection errors. Finally, the images which have been captured from angles 30 to 45 have less reprojection error. This information will be used in the next step for better flight path planning.

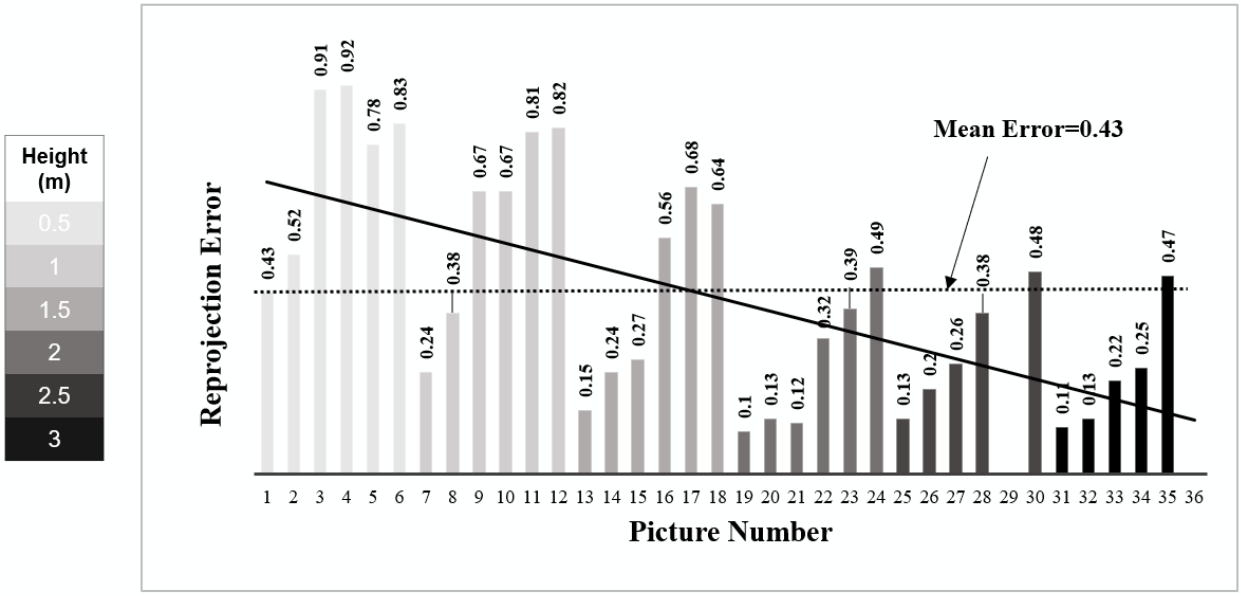


Figure 13. The reprojection errors of the pictures from the east side and the overall mean error

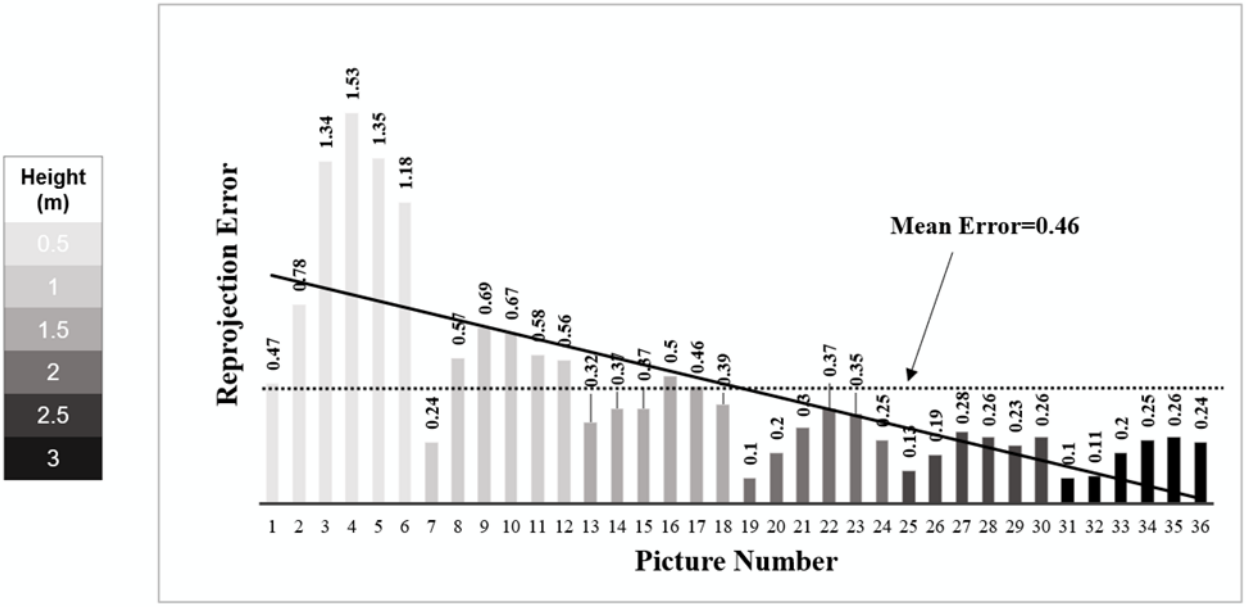


Figure 14. The reprojection errors of the pictures from the north side and the overall mean error

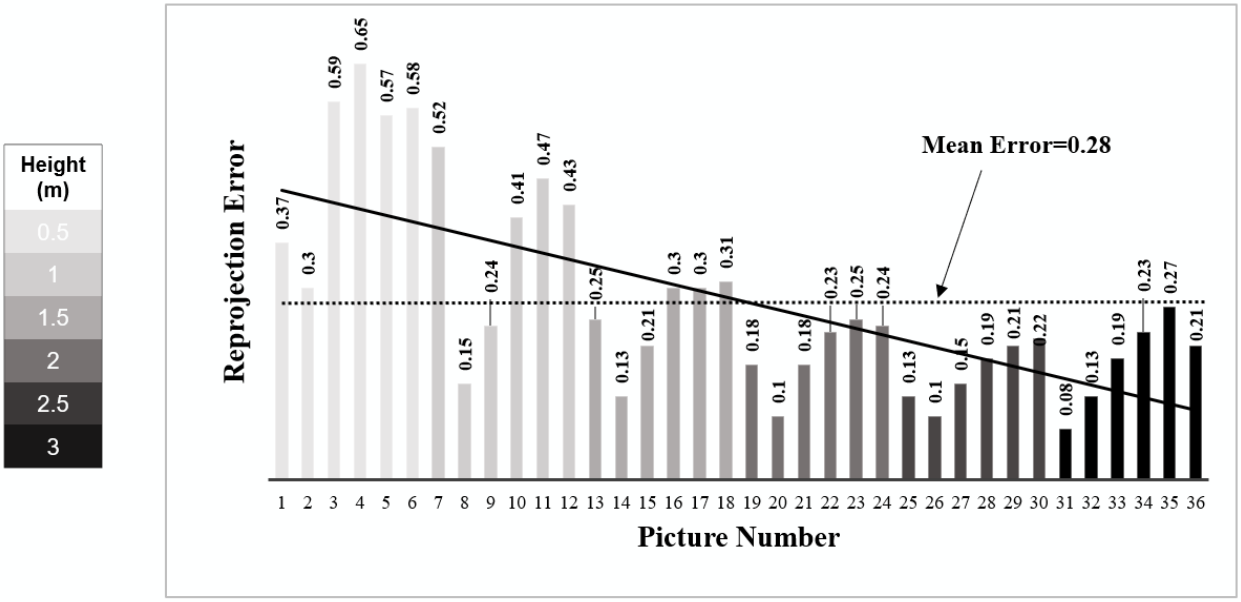


Figure 15. The reprojection errors of the pictures from the south side and the overall mean error

East Direction				
Overall Mean Error (pixels)	x Focal Length (pixels)	y Focal Length (pixels)	x Radial Distortion (pixels)	y Radial Distortion (pixels)
0.43	$f_x = 3199$	$f_y = 3193$	$RD_x = 0.0548$	$RD_y = -0.2710$
North Direction				
Overall Mean Error (pixels)	x Focal Length (pixels)	y Focal Length (pixels)	x Radial Distortion (pixels)	y Radial Distortion (pixels)
0.46	$f_x = 2993$	$f_y = 2995$	$RD_x = -0.0184$	$RD_y = -0.2410$
South Direction				
Overall Mean Error (pixels)	x Focal Length (pixels)	y Focal Length (pixels)	x Radial Distortion (pixels)	y Radial Distortion (pixels)
0.28	$f_x = 3110$	$f_y = 3121$	$RD_x = -0.0265$	$RD_y = -0.0198$

Table 3. Results of the camera calibration for the images from the east side

3.3.3 Flight Path Planning

It's important to note that the real focal length of the visual camera provided by the manufacturer is 4.3 millimeters, which will be utilized for flight planning, particularly in determining camera positions and the image size is 4056×3040 pixels. In order to facilitate flight planning for this camera, various heights have been generated using the GSD equations and the camera's given specifications.

For the purpose of this study and experiment, which focuses on the top and side surface crack detection on bridge decks (level 3 interest), two different flight heights have been chosen for the

inspection of the surface of the deck (2 and 3 meters) and three horizontal distances have been chosen for the inspection of the side of the deck (2, 3, and 4 meters) considering GSD calculations from flight path planning section and results of the in-lab camera calibration to achieve less image distortion and reprojection error in order to get more accurate results after data processing for crack detection.

Regarding the bridge inspection, the focus is on the deck of the bridge, and the FAA regulations require the flight to be conducted away from the traffic. Therefore, the flights should be performed off the road near the edge of the deck. Also, it is worth mentioning that the UAV is equipped with LiDAR sensors to avoid obstacles or unexpected objects.

The first bridge is oriented along the southeast to northwest direction, and for optimal lighting conditions, the flight will take place in the morning. It is preferred to capture photos from the east part of the deck to position the camera with the sunlight behind it. This arrangement will improve the image quality by minimizing unwanted reflections. Also, the second bridge is oriented along the east-to-west direction, and for the conditions mentioned above, the photos are captured from the south part of the bridge. Considering the length of the bridges, images are captured at intervals of 10 ft, ensuring a minimum overlap of 50% between consecutive images. A visual representation of the flight plan is depicted in **Figure 16** for the first bridge. As shown in this figure, the flight paths for the top and side of the deck are straight lines parallel to the bridge orientation and with the transverse distances from the deck of the bridge as shown in **Table 4** and **Table 5**. The positions of the UAV on the east part of the deck for photo capture are indicated by orange signs. Additionally, black rectangular signs mark the locations of benchmarks for camera calibration, located on the shoulder of the road, which will be discussed later. It is worth mentioning that the flight plan for the second bridge is similar to the first bridge and only the length of the bridge is different. Besides, for the second bridge only one unit of the bridge is inspected to have the same number of images for both bridges to ensure the comparison of the results will be more realistic.

For the data collection, 4 different flight plans are generated for the top surface of the deck for each bridge and 3 flight plans are generated for the side surface of the deck for each bridge. These flight plans maintain the same locations for the UAV positioned along a straight line, but they differ in terms of flight height, camera angles, and transverse distances from the edge of the deck. **Table 4** shows the flight plans for the top surface of the bridges and **Table 5** shows the flight plans for the side surface of the bridges.

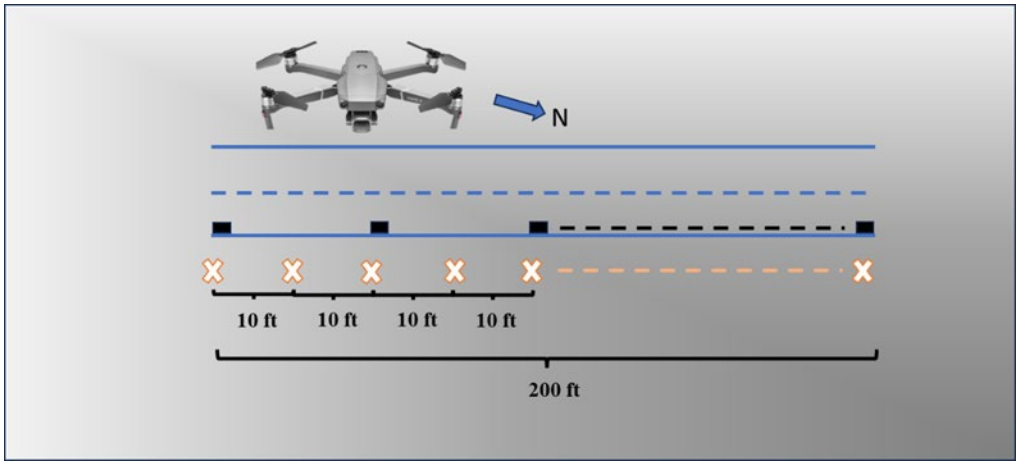


Figure 16. Schematic view of the flight plan for the first bridge (NBI number: 01791)

Flight Number	Flight Height (m)	Transverse Distance (m)	Camera Angles (°)
1	2	1	30, 35
2	2	2	30, 35
3	3	1	30, 35
4	3	2	30, 35

Table 4. Flight plans for the top surface of the bridges

Flight Number	Flight Height (m)	Transverse Distance (m)	Camera Angles (°)
1	0	2	0
2	0	3	0
3	0	4	0

Table 5. Flight plans for the side surface of the bridges

3.3.4 On-site Camera Calibration

On-site camera calibration offers the advantage of using the same camera parameters and distortion values for the collected bridge inspection data. To achieve on-site camera calibration, benchmarks are attached to specific parts of the bridge that will be covered in the inspection images. Then, the camera calibration can be performed directly during the bridge inspection simply by capturing the photos. Also, the study is not only limited to the top of the deck, and a calibration process is tested for the side of the deck for both bridges. Although the results of camera calibration for the top of the deck can be used for the side of the deck, a separate calibration for the side surface would give better and more accurate results to use for the inspection of the side of the deck. The reason for this difference in the calibration of the top and side of the deck is generated from the different photographic situations and parameters. As an example, the camera angle during the data collection for the top of the deck is different from the angle of the camera for the side of the deck which is zero. Another example for this reason is the lighting and reflection of the light which is different for the top of the deck and side of the deck.

In **Figure 17**, a collection of images is displayed from the on-site camera calibration process conducted on the first bridge for the top surface and side surface of the deck for this study's experiment. As indicated in the flight path schematic, a total of 13 benchmarks are strategically placed on the bridge deck, positioned near the edge of the bridge at regular intervals of approximately 20 feet.

Figure 17. On-site camera calibration examples for the first bridge (NBI bridge number: 01791)

A total of 104 images from the bridge inspection dataset are utilized from each bridge's top surface for camera calibration and the estimation of camera parameters and reprojection errors. The specific details of these images, such as flight height, transverse distance from the edge of the deck, and camera angles have been presented in **Table 4**. For the side of the deck, 6 benchmarks have been used and a total of 18 images (from 3 flights) are captured for each bridge. The details of the flight are shown in **Table 5**.

In **Table 6**, the overall mean reprojection error, as well as the focal length in the x and y directions, and radial distortion in the x and y directions for the top surface of the first bridge are displayed for these 104 images undergoing on-site calibration. Also, in **Table 7**, the mentioned results are displayed for the side surface of the first bridge.

Overall Mean Error (pixels)	x Focal Length (pixels)	y Focal Length (pixels)	x Radial Distortion (pixels)	y Radial Distortion (pixels)
0.25	$f_x = 3141$	$f_y = 2965$	$RD_x = 0.0161$	$RD_y = -0.0298$

Table 6. Results of the camera calibration for the images from on-site calibration for the top surface of the first bridge (NBI bridge number: 01791)

Overall Mean Error (pixels)	x Focal Length (pixels)	y Focal Length (pixels)	x Radial Distortion (pixels)	y Radial Distortion (pixels)
0.20	$f_x = 4513$	$f_y = 4367$	$RD_x = 0.0746$	$RD_y = -1.3192$

Table 7. Results of the camera calibration for the images from on-site calibration for the side surface of the first bridge (NBI bridge number: 01791)

Figure 18 visually represents the histogram of the reprojection mean errors of the images for the top of the first bridge and **Figure 19** represents the same histogram for the side surface of the bridge. The first bin in each histogram shows the number of images with a mean error lower than the overall mean error. This graphical representation aids in understanding the distribution and dispersion of the reprojection errors within the dataset. Comparing the mean error of each image with the overall mean error reveals that 55% of the images have a lower mean error than the overall

mean error for the top surface of the first bridge and 56% of the images for the side surface have a lower mean error than the overall mean error. This analysis provides insights into the variations in calibration accuracy among the different images. Also, as mentioned before, these low reprojection errors are achieved by considering the results of the in-lab calibration, and the main advantage of this approach will be discussed in validation.

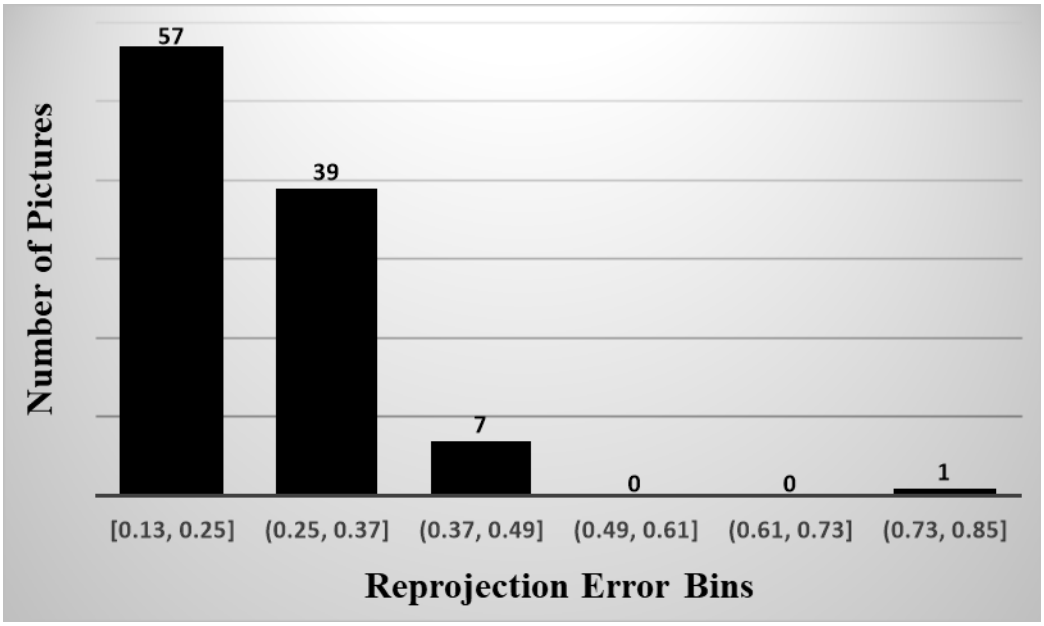


Figure 18. Histogram for the distribution of the images according to their mean reprojection errors for the top surface of the first bridge (NBI bridge number: 01791)

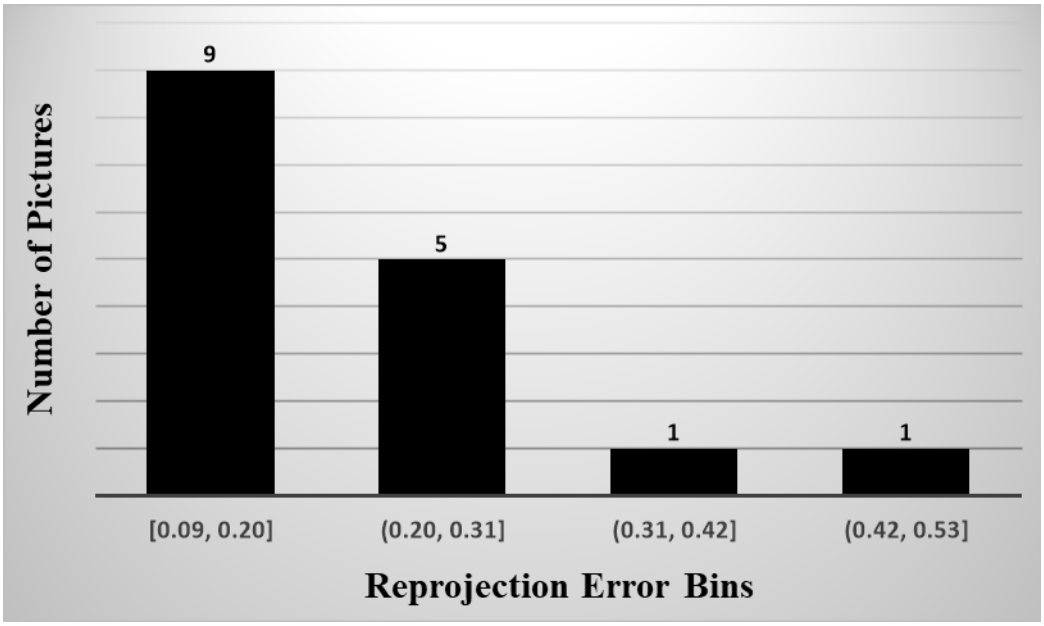


Figure 19. Histogram for the distribution of the images according to their mean reprojection errors for the side surface of the first bridge (NBI bridge number: 01791)

In **Figure 20**, a collection of images is displayed from the on-site camera calibration process conducted on the second bridge for the top and side surfaces of the deck for this study's experiment.

Figure 20. On-site camera calibration examples for the second bridge (NBI bridge number: 06255)

In Tables 8 and 9, the overall mean reprojection error, as well as the focal length in the x and y directions, and radial distortion in the x and y directions for the top and side surfaces of the second bridge are displayed for undergoing on-site calibration.

Overall Mean Error (pixels)	x Focal Length (pixels)	y Focal Length (pixels)	x Radial Distortion (pixels)	y Radial Distortion (pixels)
0.29	$f_x = 3165$	$f_y = 3002$	$RD_x = 0.0277$	$RD_y = -0.0519$

Table 8. Results of the camera calibration for the images from on-site calibration for the top surface of the second bridge (NBI bridge number: 06255)

Overall Mean Error (pixels)	x Focal Length (pixels)	y Focal Length (pixels)	x Radial Distortion (pixels)	y Radial Distortion (pixels)
0.14	$f_x = 6627$	$f_y = 6585$	$RD_x = 0.1661$	$RD_y = -6.9088$

Table 9. Results of the camera calibration for the images from on-site calibration for the side surface of the second bridge (NBI bridge number: 06255)

Figure 21 visually represents the histogram of the reprojection mean errors of the images for the top of the second bridge and **Figure 22** represents the same histogram for the side surface of the bridge. The first bin in each histogram shows the number of images with a mean error lower than the overall mean error. This graphical representation aids in understanding the distribution and dispersion of the projection errors within the dataset. Comparing the mean error of each image with the overall mean error reveals that 68% of the images have a lower mean error than the overall mean error for the top surface of the bridge and 69% of the images for the side surface of the bridge have a lower mean error than the overall mean error. Comparing the results for the side and top of the deck for both bridges indicates that the percentage of images with lower mean error than the

overall mean error is almost the same for the side and top of each bridge (only a 1% difference for the side and top of each bridge).

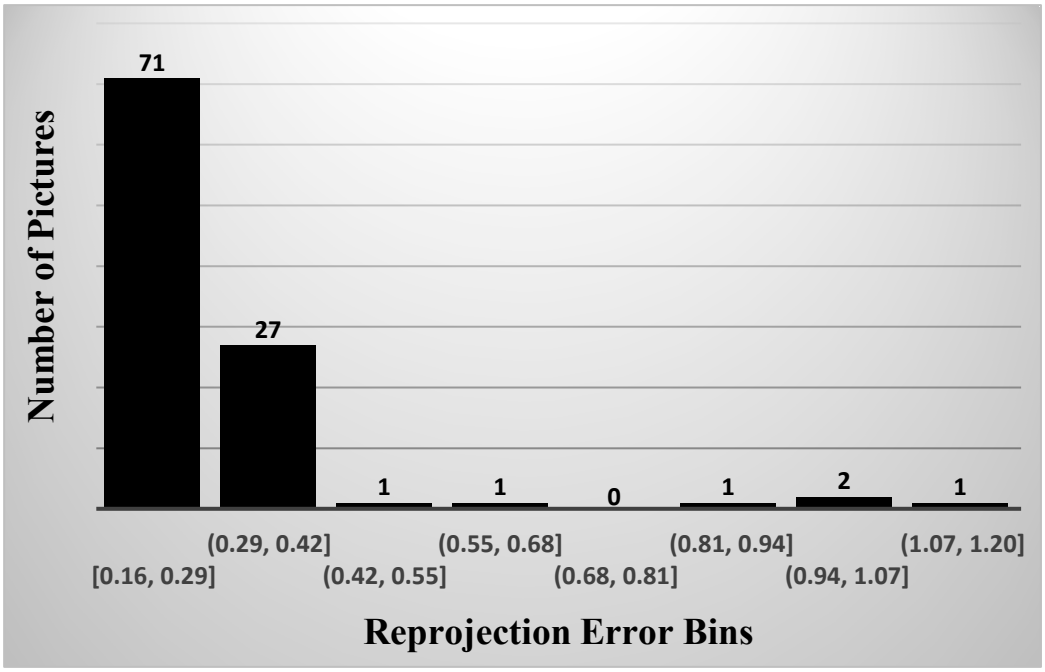


Figure 21. Histogram for the distribution of the images according to their mean reprojection errors for the top surface of the second bridge (NBI bridge number: 06255).

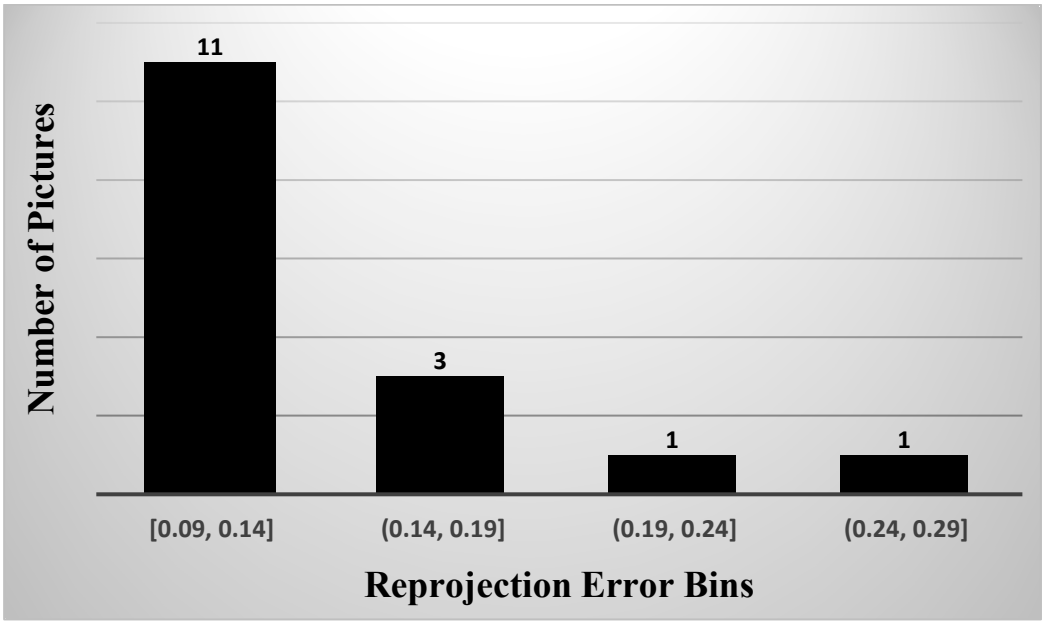


Figure 22. Histogram for the distribution of the images according to their mean reprojection errors for the side surface of the second bridge (NBI bridge number: 06255).

It is worth mentioning that among all the flight plans mentioned before, the minimum reprojection mean error for both bridges is achieved in the flight with 3m height, 1m transversal distance, and a camera angle of 30 for the top surface and transverse distance of 3m for the side surface, which aligns with the results of in-lab calibration. This result would be beneficial for the future inspection and flight plans.

3.3.5 Validation

To evaluate the feasibility of the proposed method, validations have been conducted for case studies. Two cracks from the bridge deck side surface were chosen for validation investigations. The widths of the cracks were measured from both the raw and calibrated images and subsequently compared with the ground truth measured by the inspector.

The width of the first crack is 0.51 mm, and the width of the second crack is 1.53 mm. The cracks are shown in **Figure 23** and **Figure 24** respectively. For crack detection in this study, the GSD is known from flight height and moreover, the accurate GSD is known from the calibration results which is another benefit of on-site camera calibration. By knowing the GSD, each pixel represents a known number of millimeters. The crack width is detected from raw data by image processing techniques, then the crack width is detected from the images after correction by using the calibration and reprojection results. For this purpose, first, the original image is converted to grayscale and the contrast of the grayscale image is enhanced. Then, adaptive thresholding is applied to create a binary image and finally, Gaussian smoothing is applied to the binary image to reduce noise and create smoother edges, facilitating more accurate crack detection.

The crack width detection results considering the flight plans for the side of the deck, are shown in **Table 10** for the first crack and in **Table 11** for the second crack. Finally, the percentage of the detection accuracy is compared between these two results. For both cracks, the results indicate that the result accuracy improvement is higher for the higher flight heights (more distance from the object). Also, the detected crack width is closer to the measured width in flight 2 which has a 3m distance from the object and it completely aligns with the results of camera calibration where the 3m flight had the minimum reprojection error. Moreover, it is worth mentioning that for small dimension cracks the human measurement has more error, and using this kind of accurate crack detection techniques leads to better assessments of the bridge condition.

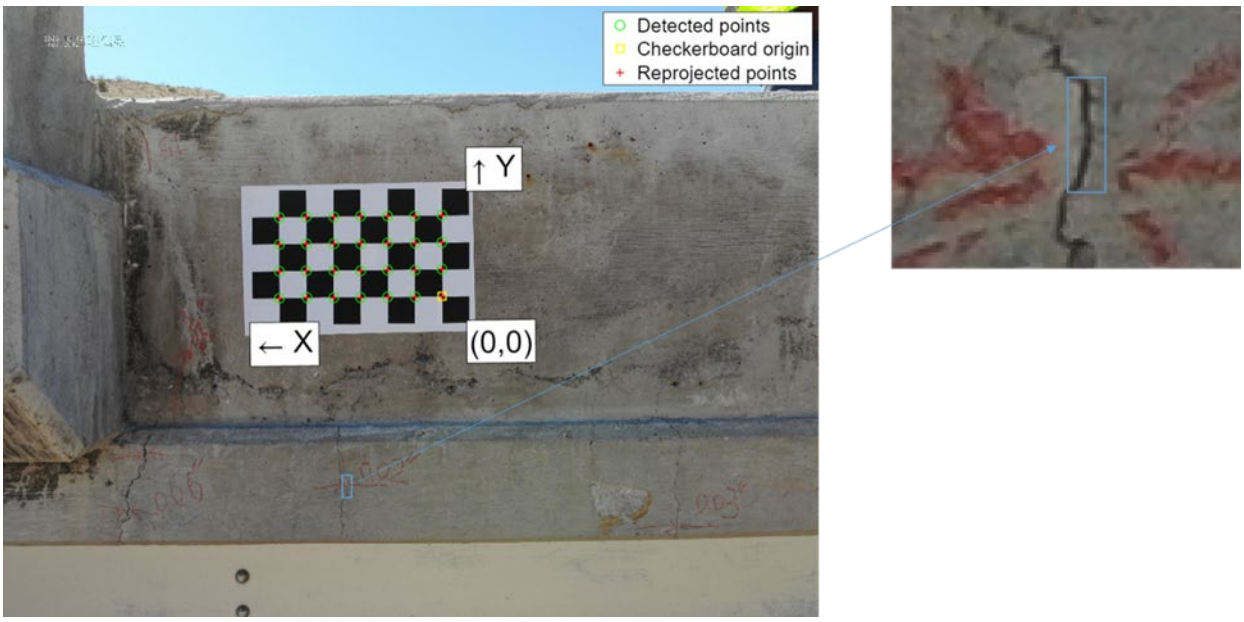


Figure 23. The first crack with a width of 0.51 mm on the side of the first bridge

Flight	Distance (m)	Measured Width (mm)	Detected Width from Raw Images (mm)	Detected Width After Correction (mm)	Result Accuracy Improvement (%)
1	2	0.51	0.71	0.66	9.80
2	3	0.51	0.71	0.62	17.65
3	4	0.51	1	0.9	19.61

Table 10. Results of the crack detection for the first crack from raw and corrected images

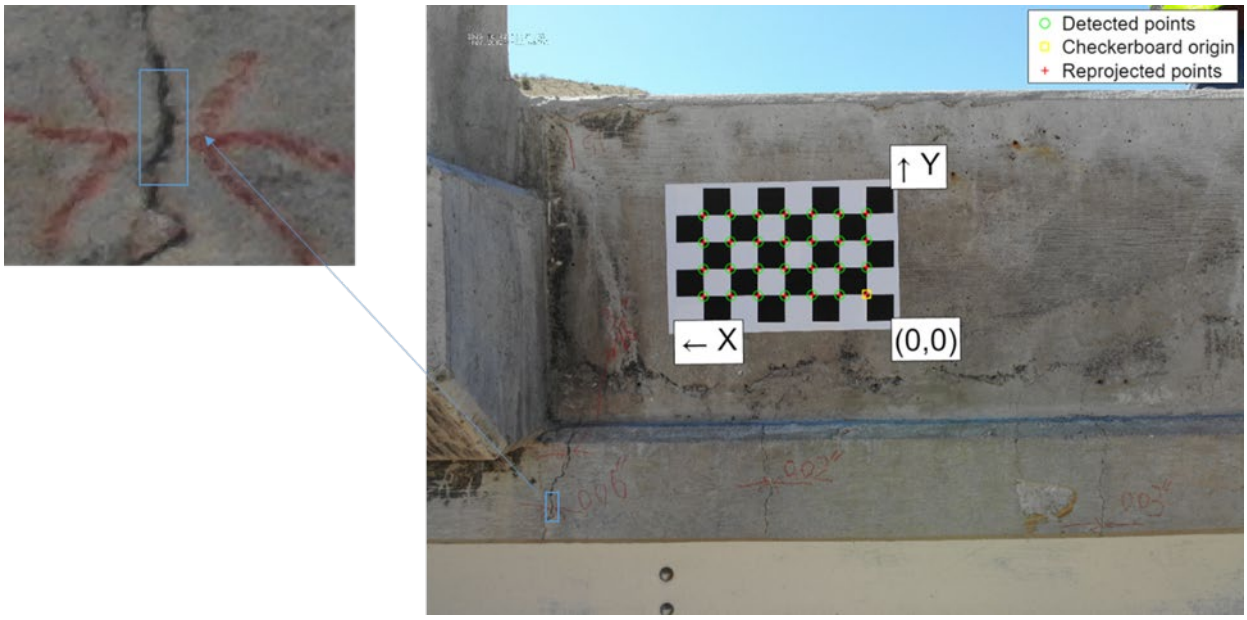


Figure 24. The second crack with a width of 1.53 mm on the side of the first bridge

Flight	Distance (m)	Measured Width (mm)	Detected Width from Raw Images (mm)	Detected Width After Correction (mm)	Result Accuracy Improvement (%)
1	2	1.53	1.78	1.67	7.19
2	3	1.53	1.77	1.55	14.38
3	4	1.53	2.13	1.80	21.57

Table 11. Results of the crack detection for the second crack from raw and corrected images

The validation for the case studies indicates an improvement in result accuracy from 7.19% to 21.57% and it shows that the improvement is higher in longer distances, which means that inspection can be done even in longer distances where more area is covered by each image and consequently less camera points are needed which leads to a shorter flight time.

TASK 4: Software Development for Flight Path Optimization

Among different UAV types, multi-rotor UAVs are flexible and can hover and vertically lift which results in higher maneuverability and makes them a suitable candidate to collect in-detailed data but on the other hand, they suffer from short endurance time [22]. To fully harness the time efficiency potential of the UAVs for inspection purposes, and to overcome the above-mentioned challenge of the short endurance time for multi-rotor UAVs, a well-planned flight path is crucial. UAV path planning involves addressing the optimal path planning for UAVs. The main goal of path optimization is to minimize energy consumption while ensuring the successful completion of the UAV's mission.

Recently, some studies have been conducted for UAV flight path optimization by using meta-heuristic algorithms. In contrast to typical optimization techniques, meta-heuristic algorithms' primary concept is based on simulating natural processes and does not require extensive mathematical computations during the optimization process [23], [24]. Over the past two decades, a multitude of meta-heuristic algorithms have been developed to aid in solving different optimization problems that were previously difficult or inconceivable to solve using mathematical programming algorithms. Genetic Algorithm (GA), Ant Colony Optimization (ACO), Particle Swarm Optimizer (PSO), Charged System Search (CSS), Colliding Bodies Optimization (CBO), and Shuffled Shepherd Optimization Algorithm (SSOA) are some of the meta-heuristic algorithms [25], [26], [27], [28], [29], [30]. The majority of these algorithms are easy to implement and provide optimal/near-optimal solutions in acceptable computation times even in complex search spaces [31].

A heuristic evolutionary algorithm was presented by Fu et al. for the purpose of optimizing the flight path for a single path connecting two points. For 2D areas, they took into account the costs associated with path smoothness, length, and safety [32]. Yu et al. proposed a novel hybrid PSO algorithm for 3D complex environments [33]. In another 3D flight path planning in urban environments, Rienecker et al. used the extended A-Star-Algorithm for path optimization considering the wind field [34]. Phung et al. conducted research for flight path planning for surface inspection by using a discrete PSO optimization algorithm and tested the proposed method on an office building and a bridge [35]. For flight route generation between two points with obstacles between them, Zhang et al. proposed a new path-planning algorithm for UAVs based on a version of the White Shark Optimization (WSO) and defined the flight length, height, and smoothness as the constraints of the problem [36]. In another study for flight path optimization between two points, Bai et al. proposed an algorithm based on the enhanced Dynamic Window Algorithms (DWA) to achieve global path optimization by taking safety and motion constraints into consideration, and according to their findings, they achieved a better flight length even when the obstacles are unknown [37]. Also, in a recent study, Souto et al. conducted a method to lower the energy usage of drones by using the Q-Learning algorithm to analyze potential courses of action that the UAVs could follow in response to random urban obstacles distributed in the scenario and in response to wind speed [38]. The improvements, challenges, and future trends of UAV flight path planning and optimization have been studied and discussed in several studies [39], [40], [41].

One of the main challenges of coverage path planning is that the search space is infinite. Although this challenge was addressed in our previous study, the computation time was still high. Considering this challenge for software development, the optimization engine has been modified so that the area will be divided into a large set of points, and the optimum waypoints will be chosen from this set of points. For this purpose, several meta-heuristic algorithms are selected and used to find the optimal path on a continuous area, among these algorithms, PSO achieved better results and has been used for the software's optimization engine where the area is not considered continuous and is divided into a set of points. By finding the optimum flight path, the challenge of the optimum flight path planning for area coverage and the challenge of the short battery lifetime of the UAVs will be addressed simultaneously due to the energy-saving resulting from the optimal path.

Key features of the software are mentioned below and will be discussed in the next section:

1. Optimization Engine
2. Constraint Handling
3. Flight Path Conversion
4. User Interface

4.1 Optimization Engine

The flight path comprises a set of camera positions from which images will be captured. These camera positions are determined by their horizontal and vertical distance from the object, as well as the angle of the camera. In this section, the crucial aspects of the optimization engine for the software will be explored shortly.

The main component of the optimization engine is the meta-heuristic algorithm. The main idea of meta-heuristic algorithms is based on the simulation of natural phenomena and, unlike the traditional optimization methods, do not need to perform heavy mathematical calculations in the optimization process. Numerous meta-heuristic algorithms have been created within the last two decades to assist in fathoming optimization problems that were already troublesome or inconceivable to solve utilizing mathematical programming algorithms. Most of these algorithms are simple to implement and present (near) optimal solutions in acceptable computation times even in complex search spaces [15].

In this study, several meta-heuristic algorithms such as Particle Swarm Optimizer (PSO), Genetic Algorithm (GA), and Shuffled Shepherd Optimization Algorithm (SSOA) have been used to not only find the best flight path among these algorithms but also, to compare the results of the algorithms and their performance. After comparing the results for different missions with different constraints, PSO achieved better results compared to other algorithms. Therefore, for the optimization engine of the software, PSO will be used.

Particle Swarm Optimization (PSO) is an optimization technique inspired by the social behavior of birds flocking or fish schooling. It operates by having a population (or swarm) of candidate solutions (particles) that move through the solution space to find the optimum. Each particle adjusts its position based on its own experience and the experience of neighboring particles, converging towards the best solution over iterations. Key parameters of PSO include the number of particles, the self-learning coefficient (which influences how much a particle is influenced by its own best-known position), the global-learning coefficient (which influences how much a particle is influenced by the best-known positions of its neighbors), and the inertia weight (which controls the impact of the previous velocity on the current velocity) [27].

By tuning these above-mentioned parameters appropriately, this study aims to leverage the strengths of PSO to effectively solve flight path optimization problems. For this purpose, Clerc and Kennedy's constriction coefficient method [42] has been used in this study to tune the parameters to control the velocity update of particles for PSO. The selected coefficients and other parameters for PSO are shown in **Figure 25**.

Parameters	Value
$MaxIt$ (Maximum Number of Iterations)	200-500
n_{pop} (Population/Swarm Size)	200-2000000
φ_1 (Cognitive Acceleration Coefficient)	2.05
φ_1 (Social Acceleration Coefficient)	2.05
φ	$\varphi_1 + \varphi_2$
χ (Constriction Coefficient)	$\frac{2}{(\varphi - 2 + \sqrt{\varphi^2 - 4 \times \varphi})}$
ω (Inertia Weight)	χ
ω_{damp} (Inertia Weight Damping Ratio)	0.999
c_1 (Personal Learning Coefficient)	$\chi \times \varphi_1$
c_2 (Global Learning Coefficient)	$\chi \times \varphi_1$

Figure 25. Selected parameters for PSO

Similar to most of the classical path optimization problems, the energy consumption of UAV flying is represented by the flight path length/time. The length of the path refers to the path from the starting point to the end point of the flight. The flight time has a direct relation to the path's length at constant speed, which in turn influences the UAV platform's energy consumption. Considering the length of the path as the objective function to minimize, the cost function is defined as below:

$$Cost_l(X) = L = \sum_{i=1}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \quad (10)$$

Where $Cost_l$ represents the length cost of the path in meters, L is the total length of the path in meters, and n is the number of points traversed by the path (waypoints) where the images are captured. x_i, y_i are the coordinates of the ith traversal point in meters.

By having the $Cost_l$, the consumed energy during a flight can be found by the below equation:

$$Cost_{El} = E_l' \times Cost_l \quad (11)$$

Where $Cost_{El}$ is the consumed energy (J) based on the flight length and E_l' is the energy consumption rate based on the flight length (J/m) and it can be found either from the drone's specifications or by experiments.

Under constant speed conditions, flight time is directly proportional to flight length, thus the consumed energy can be calculated by the following equation:

$$Cost_{Et} = E_t{}' \times t \qquad (12)$$

Where $Cost_{Et}$ is the consumed energy (J) based on the flight time, t is the flight time (s), V_{avg} is the average flight speed (m/s), and $E_t{}'$ is the energy consumption rate based on the time (J/s).

Search space for the variables will consist of points that are generated specifically for the mission area and the number of the points is based on the dimensions of the area. The required number of waypoints for the flight path will be chosen from the initial points of the search space.

4.2 Constraint Handling

For flight path optimization there are three constraints that should be checked: area coverage, minimum overlap, and obstacle avoidance. To ensure that the whole inspection area is covered by the images taken from the traversal points, the area will be meshed and divided into small triangle areas that contain a set of m points, and each point has x and y coordinates ($M = \{x_1, x_2, \dots, x_m, y_1, y_2, \dots, y_m\}$). To satisfy the coverage constraints each of these m points should be covered at least in one of the images. Image width and height (D_W, D_H) are known from equations 3 and 4, and by having this width and height, each point of the M will be checked to be covered at least in one of these image footprints. **Figure 26** shows an example of meshing for an area of 10×10 $(100m^2)$.

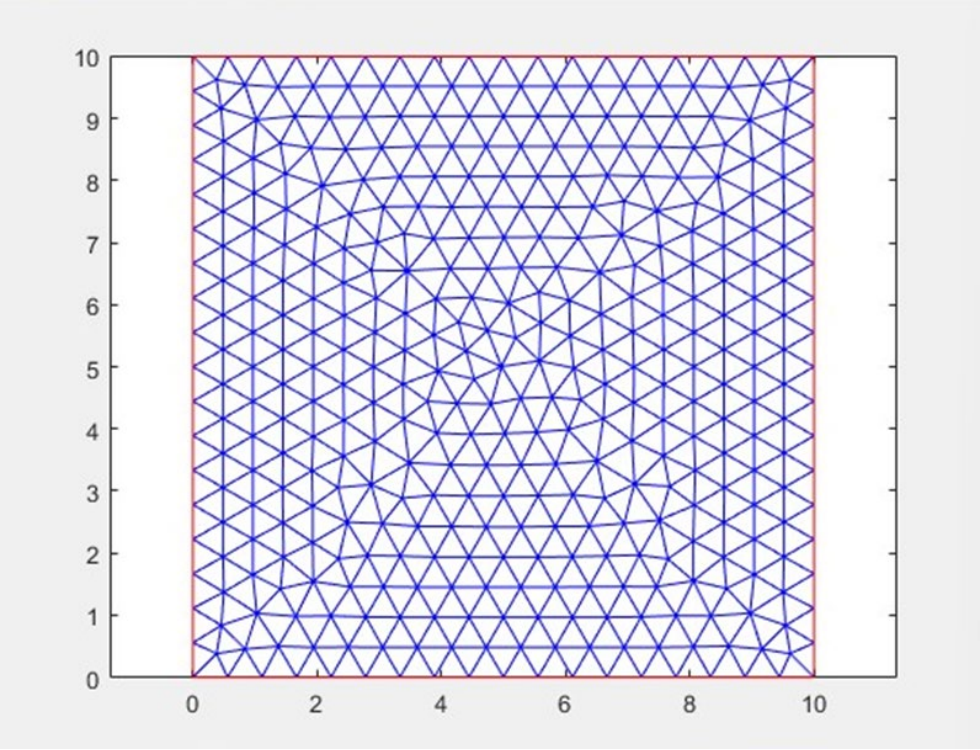


Figure 26. A meshing example for an area of $10 * 10$

To ensure at least 50% overlap between the consecutive images, an adjacency radius (r) is defined. This radius is the maximum allowable distance between two adjacent points which means that the allowable location for the traversal point $i + 1$ is inside or on a circle where the center of the circle is the point i and the radius of the circle is r. **Figure 27** shows this overlap constraint between two consecutive points. As illustrated in the figure, the blue point is the ith point which has an image footprint equal to the blue rectangle and the possible area to select the $i + 1$th point is inside or on the circle.

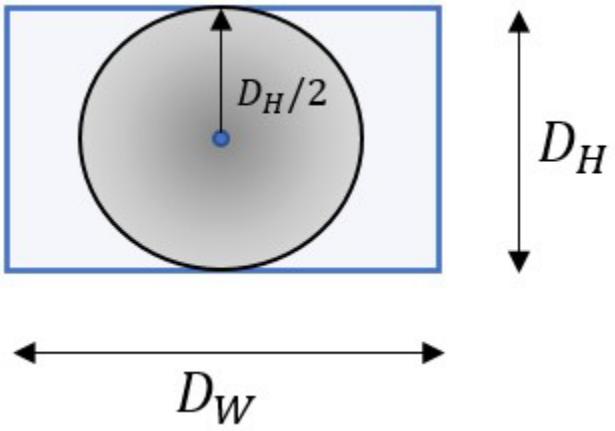


Figure 27. Overlap constraint between two traversal points

In this study, to satisfy the overlap constraint, the mentioned radius is set to $D_H/2$. This adjustment ensures at least 50% overlap between consecutive points. The minimum overlap occurs if the point $i + 1$ is located exactly at a vertical distance of $D_H/2$ form point i as it is shown in **Figure 28**. Consequently, if the point $i + 1$ is located in another position to the point i or the distance is lower than $D_H/2$, the overlap would be more than 50%. Some examples of this condition are shown in **Figure 28**. It is worth noting that this 50% overlap is the minimum consideration but in the software, users will be able to adjust the overlap between 50% to 90%.

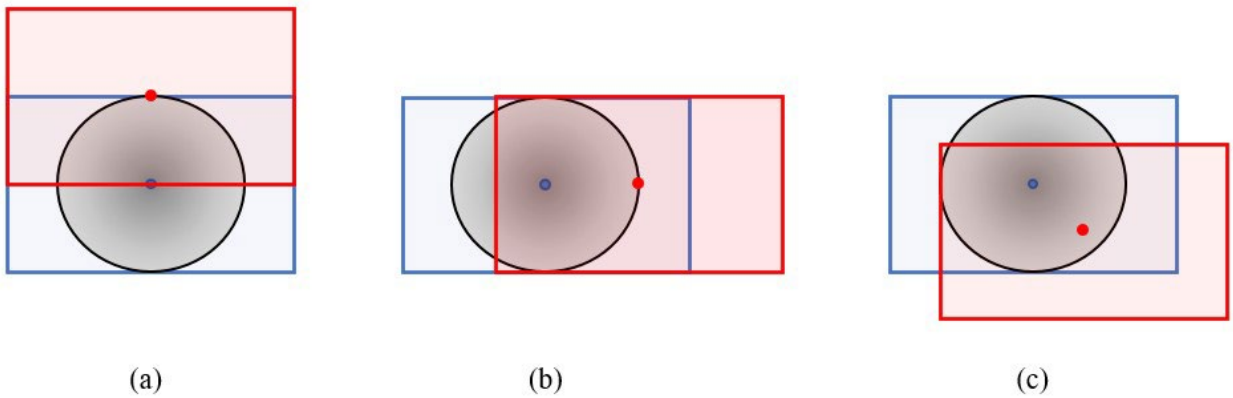


Figure 28. Overlap conditions: (a) overlap = 50% (b), (c) overlap > 50%

The last constraint is to make sure that obstacles have been avoided in the flight path to ensure the safety of the flight. For this purpose, obstacles are represented as rectangles and the number of obstacles is assumed to be *K*. By knowing the coordinates of the bottom-left corner of the obstacles, which are x_k, y_k respectively, and by knowing the length and width of the rectangles, which are l_k, w_k respectively, the intersection between each rectangle and the flight path between two waypoints could be checked. By finding the potential intersection points and their coordinates $(I_{1x}, I_{1y}, I_{2x}, I_{2y})$, the length of the flight path inside the obstacles can be determined using the following equation:

$$L_O = \sqrt{(I_{2x} - I_{1x})^2 + \left(I_{2y} - I_{1y}\right)^2} \qquad (13)$$

Where L_O is the length of the flight path inside the obstacle in meters.

Finally, the optimum flight path problem is formulated by the following expression:

Find an integer design vector $\{X\} = \{x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n\}$, where x_i, y_i are the x and y coordinates of the ith traversal point of the flight path. Hence, the design problem can be expressed as:

Minimize Equation 10

Subjected to

$\boldsymbol{g_1}$: $For\ j = 1\ to\ m$:

$For\ i = 1\ to\ n$:

$$\left(x_i - x_j\right) - D_W/2 \leq 0 \quad \& \quad \left(y_i - y_j\right) - D_H/2 \leq 0 \qquad (14)$$

$\boldsymbol{g_2}$: $For\ i = 1\ to\ n-1$:

$$\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} - D_H/2 \leq 0 \qquad (15)$$

$\boldsymbol{g_3}$: $For\ i = 1\ to\ n-1$:

$For\ k = 1\ to\ K$:

$$L_O = 0 \qquad (16)$$

Where m is the number of the generated nodes for meshing to check the coverage, n is the number of the nodes of the flight path (traversal points), x_i, y_i are the coordinates of the ith traversal point,

and x_j, y_j are the coordinates of the jth generated points of the coverage meshing. Also, g_1, g_2, g_3 ae the constraints of the problem.

The steps and the flowchart of the process are shown in **Figure 29**. It is worth mentioning that termination criteria is the number of iterations for meta-heuristic algorithms.

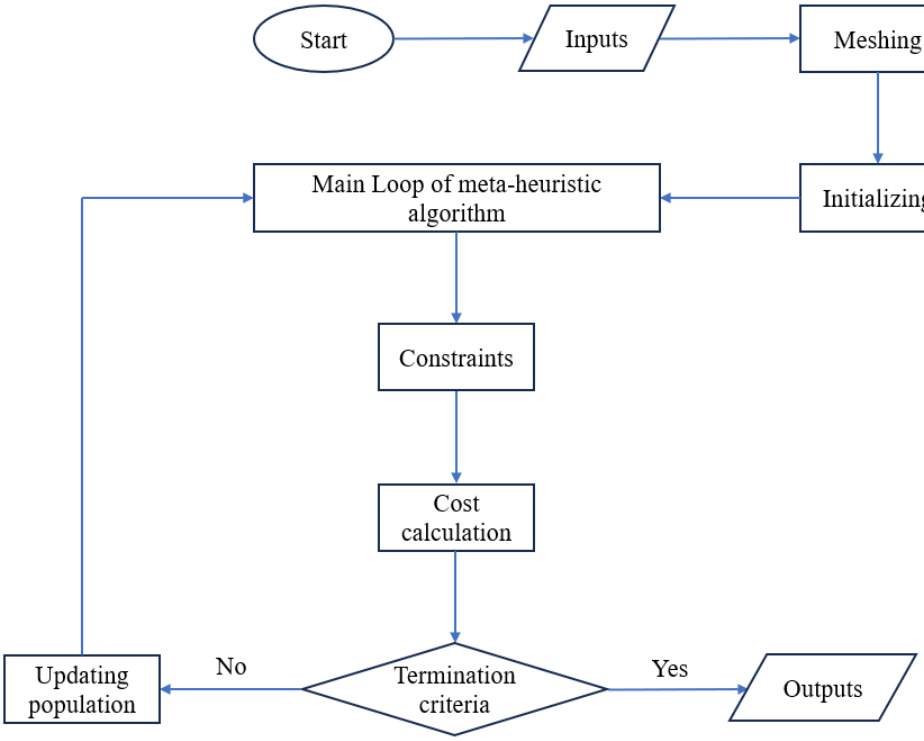


Figure 29. Flowchart of the flight path optimization process

4.3 Flight Path Conversion

With the adjustments that have been made to the code, users would be able to define any polygon shapes for the mission area. Once the optimum flight path is generated, it should be projected on the real mission area defined. For this purpose, the generated flight path, which is in decimal units (m) in the x and y directions, will be converted to the Geographic Coordinate System (GCS) so that the user can export the flight path of the mission area and import it into UAV.

4.4 Flight Path Planning Web Application

This section outlines the key features and progress of the flight planning software being developed in collaboration with NMSU researchers. The software program, which will be developed as a web application, aims to optimize drone flight paths by addressing key constraints and delivering a streamlined, intuitive, and user-friendly experience. It achieves this through the implementation of a particle swarm optimization algorithm. Additionally, the software program will be developed using Python, MATLAB, HTML, CSS, and JavaScript within the Flask framework.

Below is a comprehensive list of the key constraints that the web application will address, along with a detailed explanation of its features. This overview outlines how each constraint is managed and the corresponding functionalities that enhance the software's capabilities in optimizing drone flight paths.

Mission Area or Area of Interest (AOI): The web-based interface allows users to select an AOI. Based on this selection and the following parameters, the software automatically generates a flight path tailored to the AOI. Instead of a traditional grid system, which can be inefficient, the software employs a single, continuous flight line strategy. This method simplifies flight planning and optimizes both time and battery usage.

Battery Life: The web application considers battery life as a fundamental constraint in flight, aiming to maximize image collection within the battery's operational limits. The software calculates the maximum allowable flight time based on the drone's battery capacity, while reserving sufficient battery power to prevent depletion. It enhances the drone's practical usability and ensures reliable operation throughout a mission.

Flight Speed: It influences the stability of the drone, the quality of captured data, and the efficiency of the flight path. Properly planning and adjusting flight speed ensures smoother operation, optimal image or data collection, and helps in avoiding collisions or interference with other objects. Users will be able input the appropriate values for flight speed.

Camera Specifications: Users will be able to input the camera's focal length in millimeters, the sensor height and width in millimeters, and the image height and width in pixels.

Flight Height: This height information is crucial in drone flight path planning. It affects factors such as the quality of the captured images, the coverage area, the ground sampling distance (GSD), and the safety of the flight. Users will be able to input the appropriate values for flight height.

Overlap Considerations: To ensure comprehensive coverage and high-quality data collection, the software manages both forward and side overlap of collected aerial imagery. Forward overlap pertains to the extent of overlap between consecutive flight paths, while side overlap addresses the overlap between parallel flight lines or parallel imagery frames. By assigning values (50% to 90%) to the desired overlaps, the software ensures complete area coverage and enhances data reliability. This feature supports the creation of co-registered orthophotos and digital surface models (DSMs), allowing for the detection of finer-scale bridge deck distresses.

Waypoint Optimization: The software is designed to minimize the amount of waypoints required for a given flight mission. Advanced algorithms, including metaheuristic algorithms, are employed to generate efficient flight paths, reducing waypoint management complexity. This optimization simplifies user interaction and reduces the computational load on the drone's navigation system.

Ground Sampling Distance (GSD): The software ensures that the spatial resolution of captured data meets user-specified ground sampling distance (GSD) requirements by calculating and adjusting flight paths accordingly. This guarantees that the imaging mission fulfills end-user specifications and results in high-quality data collection. Additionally, this tailored approach not only fulfills user requirements but also enhances the accuracy and usefulness of the collected data for various applications.

Obstacle Avoidance: To enhance flight safety, the software integrates pre-defined obstacle avoidance algorithms within the user interface. This feature allows users to input and define specific obstacles, which the algorithm then uses to dynamically adjust flight paths and prevent collisions. By incorporating these user-defined obstacles, the software ensures safe and reliable operations throughout the mission.

For the outputs of the web application, users will have access to the following items:

- Visualized Flight Path: an example of the visualization is shown in **Figure 30**.
- Waypoints: displayed as individual points on a background image, including their coordinates in either a geographic or projected coordinate system.
- GSD: shown in mm/pixel, based on the flight height and overlap specified by the user.
- Flight Path Length: indicated in meters (m).
- Approximate flight time: provided in seconds (s).

Please note that as a web application, the software will be accessible only to authorized users. After the optimized flight path is generated, users will be notified and provided with a link to download the flight path file, which they can then upload to the drone.

The sprint planning for the software development is as follows:

- Sprint 1: Requirements gathering and initial design.
- Sprint 2: Basic optimization engine implementation.
- Sprint 3: UI design and integration with optimization engine.
- Sprint 4: Testing and validation.
- Sprint 5: Improvements.

The first two sprints are achieved, and the project team is working on the third sprint. More specifically, the UNM Team has begun software development in close collaboration with the NMSU team. This collaborative effort highlights the co-development approach, involving joint planning, design, and execution phases to ensure that the development aligns with our shared objectives and requirements. As of the reporting period, we have undertaken several key activities, including:

Algorithm Development: We have participated in the development of algorithms that support the constraints of battery life, overlap, GSD, collision avoidance, and waypoint optimization. This collaborative effort ensures that the software meets the specified requirements effectively. Since the project's inception, the UNM Team has conducted four meetings with the NMSU team to maintain active engagement in the algorithm development process. These meetings have been essential for coordinating our efforts, discussing progress, addressing challenges, and refining our approach.

Bounding Box System: In our approach to algorithm development of the AOI selection, we have decided to assign no value (null) to the cell centers of a minimum grid bounding box system that encloses the entire AOI. This decision aligns with our goal of simplifying the flight path planning process and avoiding unnecessary complexity. The bounding box system will also ensure that either a projected coordinate system or a geographic coordinate system is assigned to the

waypoints within the AOI. Additionally, it ensures that the AOI properly overlaps with the bridge object on the ground.

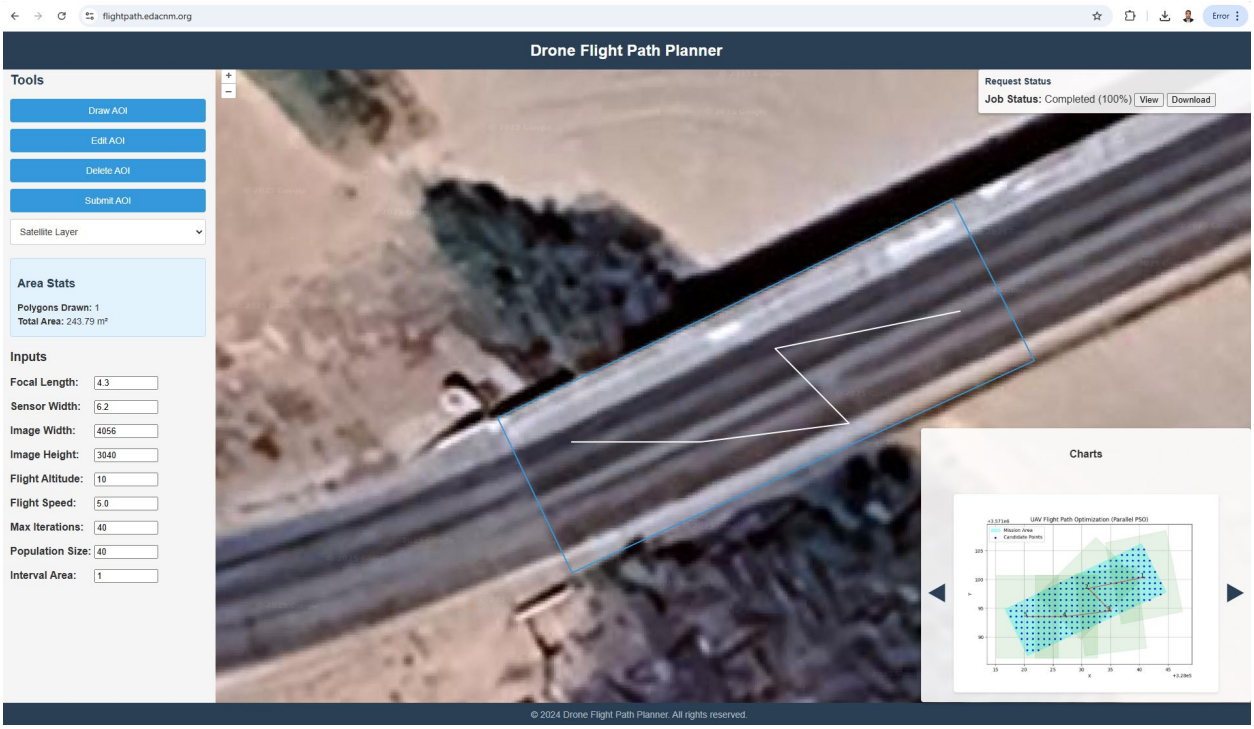


Figure 30. Flight path visualization example

4.5 Result Analysis

The PSO and GA algorithms have been tested on different mission areas, and the results have been compared to the results from DJI GS Pro (DJI GSP) mission planning app for the same areas. **Figure 31** represents the energy consumption based on the flight time for each of the algorithms in each inspection flight path planning for different areas. As a result of this chart, PSO achieved a better result (less energy consumption and flight time) compared to GA in all the experiments. The energy consumption results of PSO are optimized for an average of 57% compared to the DJI GSP, whereas the results of GA indicate an average of 50% optimization compared to the DJI GSP. Since these results are based on the flight time, the optimization results for the flight time have the same percentages.

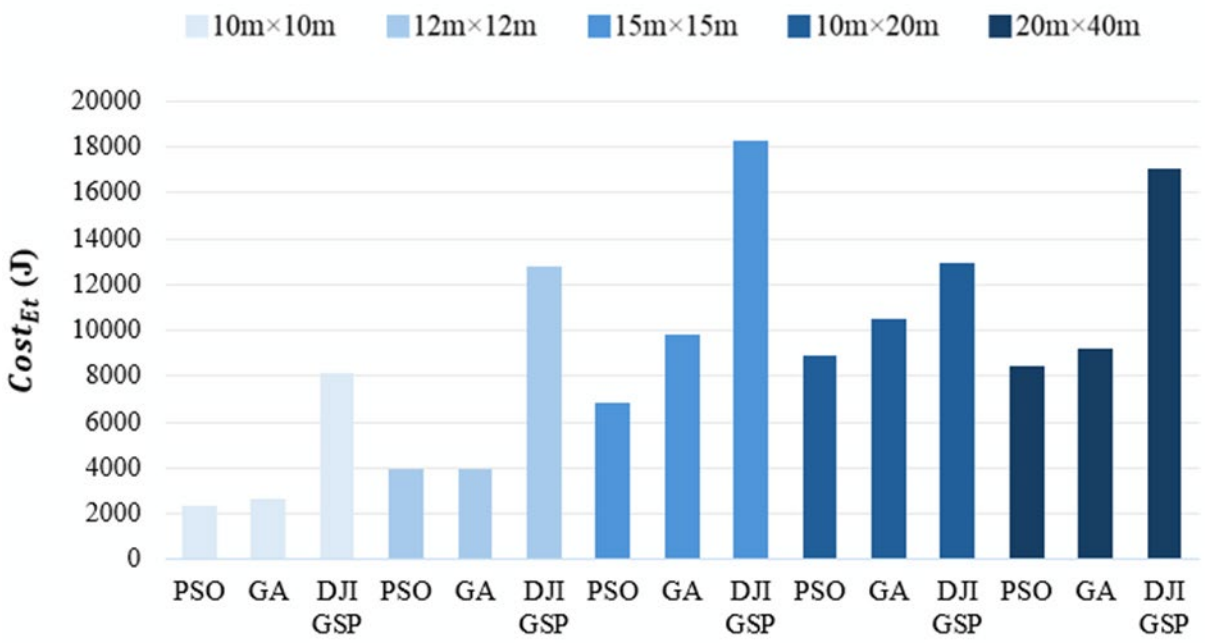


Figure 31. Energy consumption of algorithms based on the flight time for the experiments

TASK 5: Dataset Collection

The objective of this task was to collect comprehensive datasets from bridges across New Mexico, which will be utilized for image processing and model training for the purpose of damage detection and condition assessment. The dataset will be divided into three categories including the training dataset, the validation dataset, and the testing dataset. The collected data include high-resolution images and infrared thermography images capturing various aspects of bridge conditions, such as surface cracks, deformations, and material degradations. Also, different parts of the bridge such as piers, girders, and substructure are included in the captured images. These images are acquired and will be acquired using drones ensuring thorough coverage and detail.

The collected datasets play a crucial role in the field of image processing and model training. High-quality images from RGB cameras provide clear visual data on visible damages like cracks and surface wear, while IRT cameras capture thermal anomalies that indicate underlying structural issues. These datasets will undergo a rigorous preprocessing phase to enhance image quality, remove noise, correct distortions, and ensure consistency across all collected data. This preprocessing step is essential for creating a reliable dataset that can be used to train machine learning models effectively. Annotating these images with detailed information about the observed structural conditions will serve as ground truth for model training, ensuring that the models learn to accurately identify and classify different types of damages.

For this purpose and for now, datasets are collected from 11 bridges with these bridge numbers: 5825, 5629, 5734, 5801, 1791, 8900, 6255, 7264, 7265, 6803, and 5964. The location and NBI number of the bridges are shown in **Figure 32**.

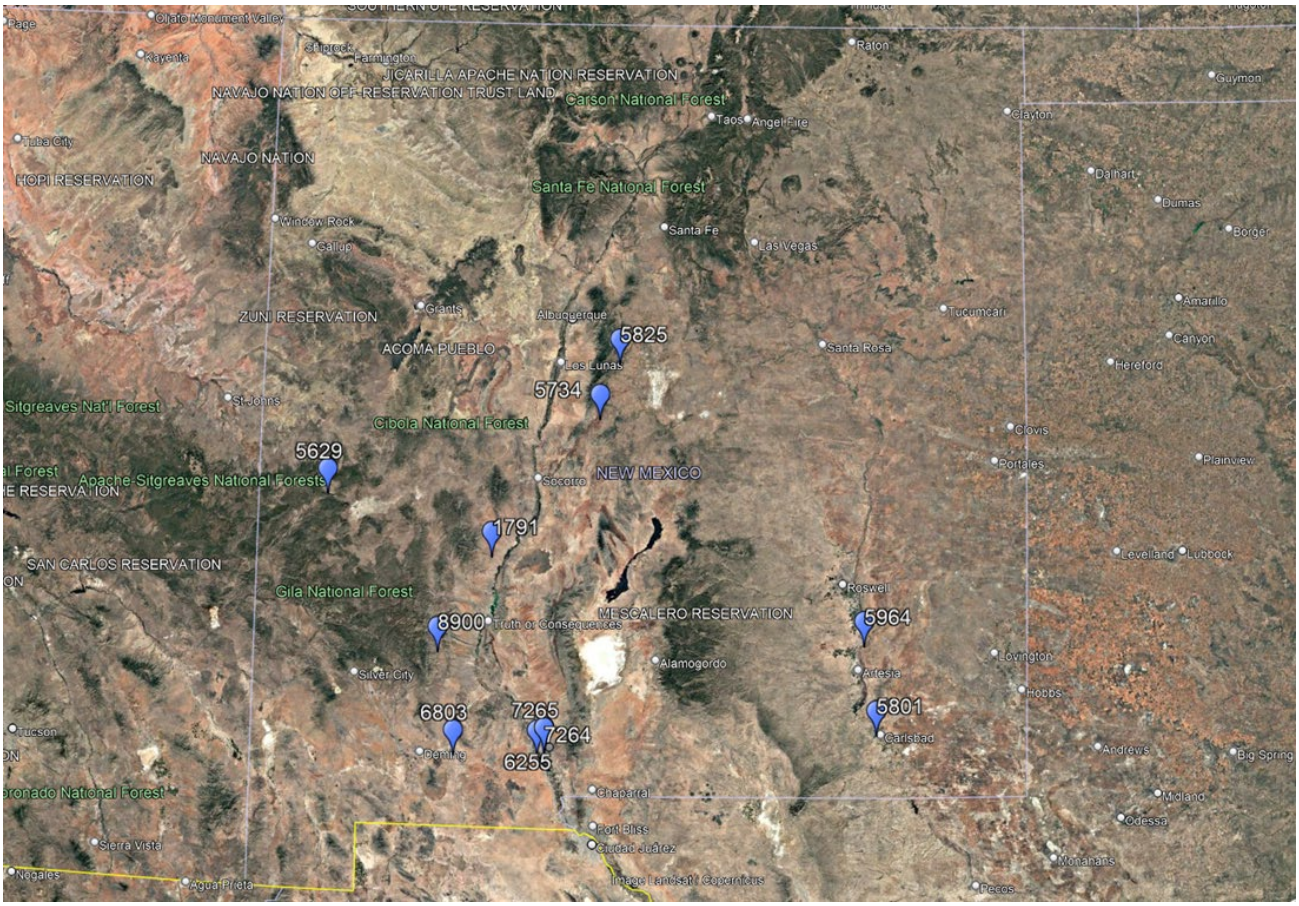


Figure 32. Bridges that are data have been collected from

The task is still ongoing, and data collection continues. The ultimate goal of this task is to develop robust image-processing algorithms and machine-learning models capable of autonomously detecting and classifying structural issues in bridges. Some of the collected RGB images are shown in **Figure 33**, and thermal image samples are shown in **Figure 34**:

Figure 33. RGB samples from the collected datasets

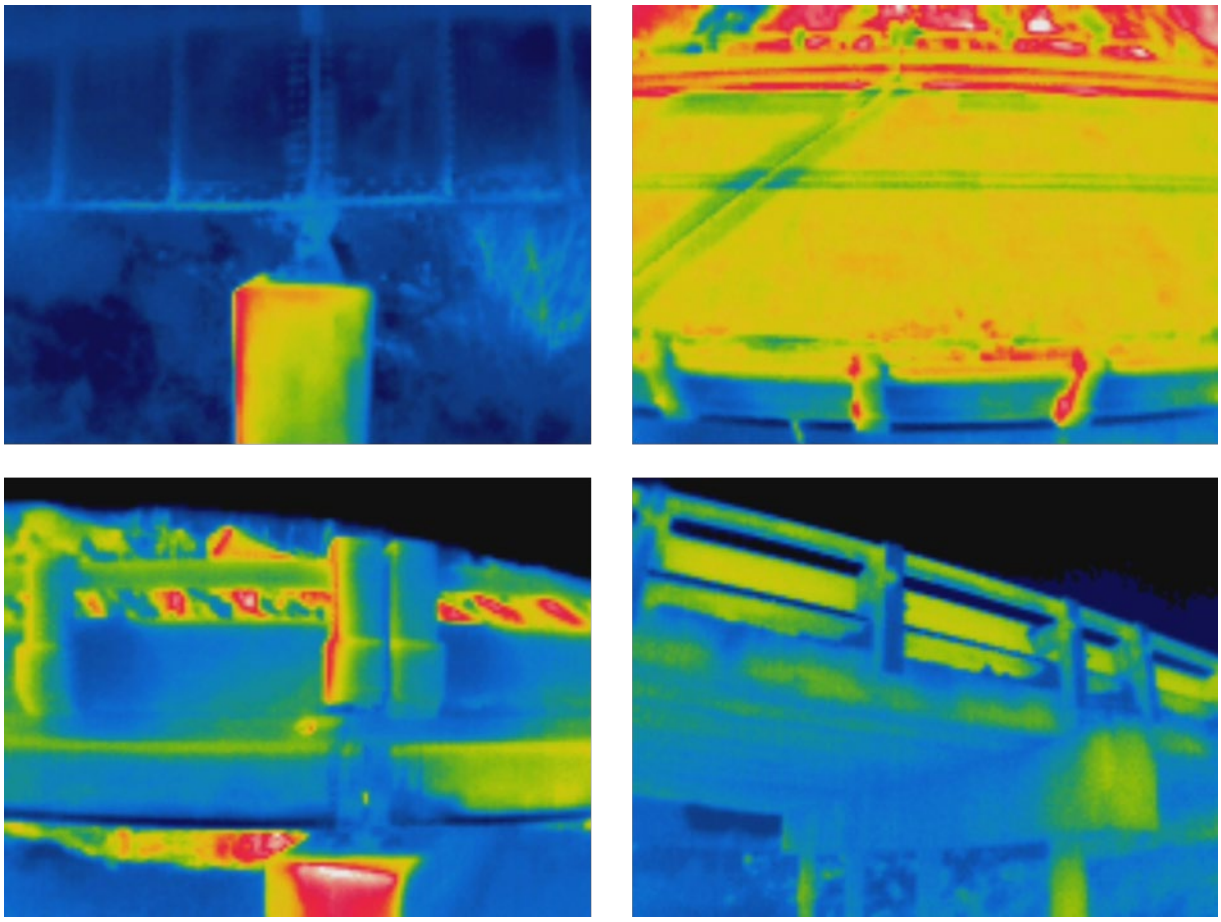


Figure 34. Thermal image samples from the collected datasets

TASK 6: Bridge Deck Identification

6.1 Dataset Preparation

The dataset utilized in this study comprises 113 high-resolution images of bridge decks, each with dimensions of 3040×4056 pixels. These images were collected using a DJI Mavic 2 Enterprise to capture detailed visual data of bridge surfaces from a top-down perspective. The primary objective of this dataset was to support the development of a semantic segmentation model capable of distinguishing bridge deck surfaces from their surrounding background environments, such as pavements, barriers, and structural supports.

Each image in the dataset was manually annotated to create a pixel-wise ground truth mask, where the two classes of interest were:

- Deck: representing the concrete or asphalt surface of the bridge deck (foreground).
- Background: encompassing all other elements not part of the deck, including sky, shadows, guardrails, curbs, and adjacent infrastructure.

The annotation process was performed using MATLAB's Image Labeler application [19]. Annotators manually outlined the deck boundaries, and the label masks were exported as

categorical images and then converted into binary masks, with the deck class represented as pixel value 255 and the background as 0. This binary format was chosen to simplify the segmentation task and ensure compatibility with pixel-level loss functions during training.

To ensure efficient learning while preserving the aspect ratio of the original images, all input images and masks were uniformly resized to a resolution of 384×512 pixels. This resizing was done using bicubic interpolation for RGB images and nearest-neighbor interpolation for binary masks to avoid label smearing.

The entire dataset was randomly partitioned into three subsets:

- Training set: 80 images (~70.8%)
- Validation set: 17 images (~15.0%)
- Test set: 16 images (~14.2%)

This split was designed to provide sufficient samples for model optimization while ensuring reliable validation and testing. It is important to ensure that the test set contains diverse examples with varying lighting conditions, surface textures, and obstructions such as debris or markings, to robustly assess the model's generalization capability. **Figure 35** shows some examples of images of the labeling process, and **Figure 36** illustrates the dataset structure. This curated dataset served as the foundation for training and evaluating the DeepLab v3+ semantic segmentation model described in the subsequent sections.

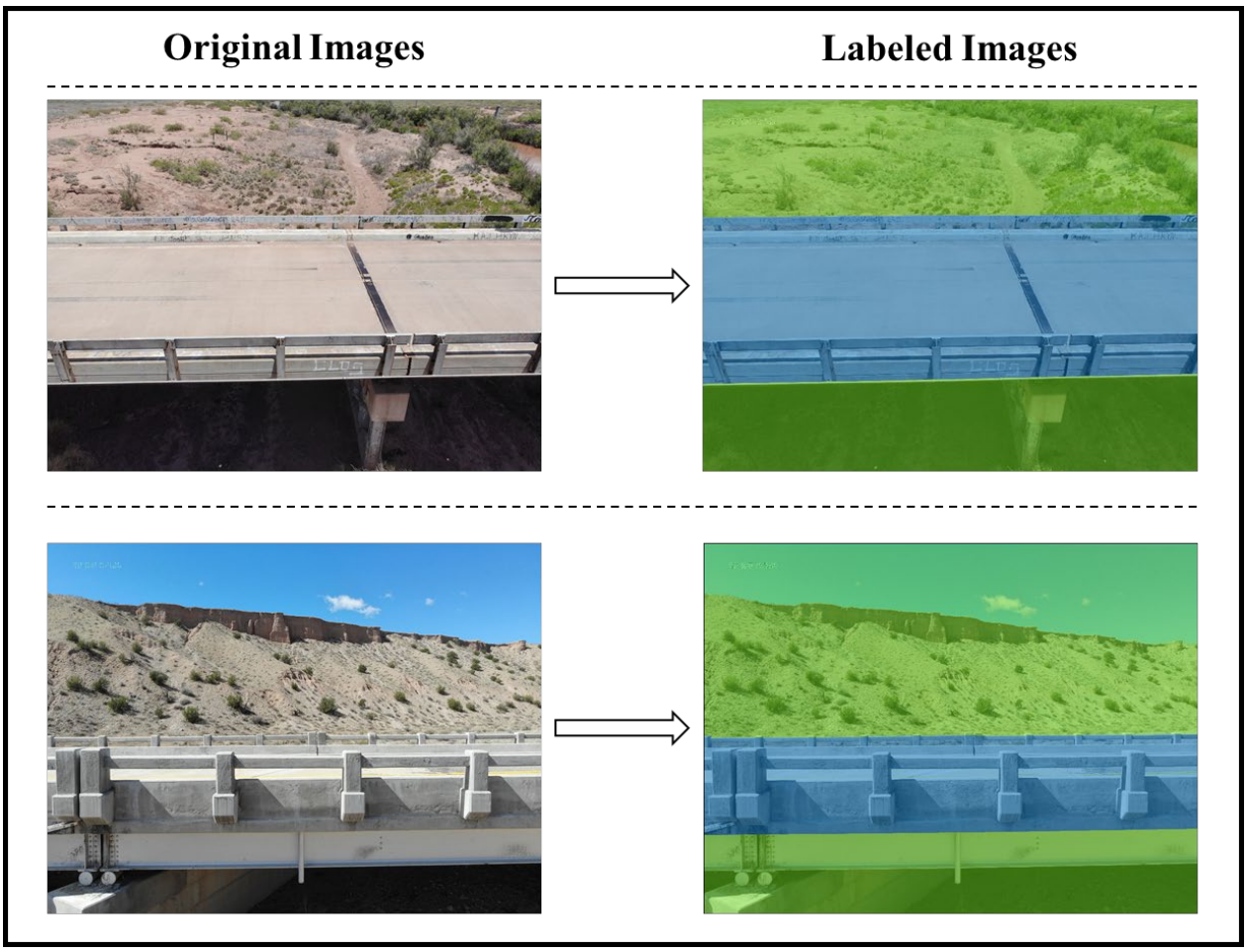


Figure 35. Examples of deck identification image labeling

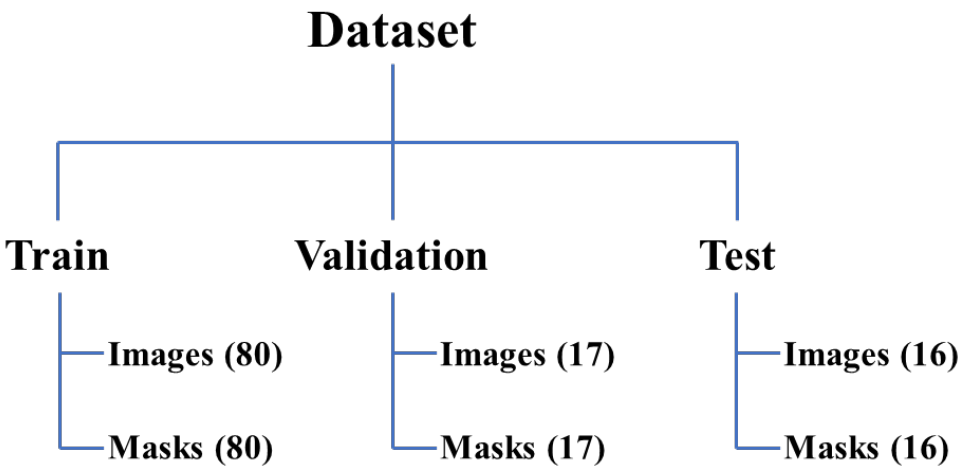


Figure 36. Dataset structure for deck identification model training

6.2 Data Augmentation

To enhance the generalization capability of the semantic segmentation model and mitigate overfitting due to limited data, a suite of data augmentation techniques was applied during training. The original dataset consisted of only 80 training images, which is relatively small for training a deep convolutional neural network. Therefore, online augmentation was used to artificially increase the variability of the training data.

All augmentations were performed on-the-fly using MATLAB's "imageDataAugmenter" class, ensuring that each training epoch introduced slight variations in image appearance and geometry. The goal was to simulate realistic conditions that the model might encounter during actual UAV-based inspections, such as different flight angles, lighting conditions, and bridge deck appearances.

The following transformations were applied randomly and independently to each image-mask pair during training:

Transformation	Range / Parameters	Purpose
Horizontal Flipping	50% probability	Simulates bidirectional UAV flight over the same bridge deck
Random Rotation	±10 degrees	Simulates variations in UAV camera orientation during capture
Random Scaling	90%–110% (X and Y independently)	Mimics changes in UAV altitude and image zoom

Table 12. Applied augmentations to the training dataset

These augmentations were applied only to the training set, ensuring that validation and test results remained unbiased and representative of real-world performance. Data augmentation is especially critical in pixel-wise segmentation tasks because it not only increases the diversity of image appearance but also expands the spatial configurations that the network is exposed to. For example, horizontal flips help the model learn symmetry in deck structures, while rotation and scaling aid in learning translation-invariant features.

6.3 Model Architecture

The model architecture employed for the semantic segmentation of bridge decks is based on DeepLab v3+, a state-of-the-art convolutional neural network known for its accuracy and efficiency in pixel-wise classification tasks [43]. DeepLab v3+ combines spatial pyramid pooling, atrous convolutions, and a decoder module to produce high-resolution segmentation outputs while maintaining deep contextual understanding. The ASPP module allows the network to perceive objects at multiple receptive field sizes, which is particularly useful for bridge decks that may appear at various scales due to changes in UAV altitude or perspective.

In this study, DeepLab v3+ was implemented using MATLAB's Deep Learning Toolbox, with the encoder initialized using a ResNet-18 backbone pretrained on ImageNet. ResNet-18 is a convolutional neural network that is 18 layers deep. We can load a pretrained version of the network trained on more than a million images from the ImageNet database [44], [45]. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse,

pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. This choice strikes a balance between model complexity and computational efficiency, making it suitable for deployment on resource-constrained platforms such as UAVs.

Key architectural components of the model are briefly described in **Table 13**.

Component	Description
Backbone	ResNet-18 (18-layer residual network) pretrained on ImageNet
ASPP Module	Atrous Spatial Pyramid Pooling with multiple dilation rates for context
Decoder Module	Refines segmentation results by combining low- and high-level features
Upsampling Strategy	Bilinear upsampling to restore final output to full spatial resolution
Output Layer	Softmax classification layer for binary segmentation (deck vs. background)

Table 13. Key architectural components of the DeepLab v3+ model

The choice of input size [384×512] preserved the aspect ratio of the original 3040×4056 images and allowed for efficient GPU training. The number of output classes was set to 2, corresponding to the binary nature of the segmentation task. The model was trained using a single GPU, and the use of ResNet-18 (instead of deeper alternatives like ResNet-50) ensured that training and inference were computationally tractable. This configuration also supports potential onboard deployment on embedded GPUs (e.g., NVIDIA Jetson platforms), aligning with real-time UAV inspection applications. The overall architecture follows a typical encoder-ASPP-decoder structure, enabling robust segmentation of bridge decks even in small datasets. In our trained version, its network consists of 100 layers with 20.6 million learnables.

6.4 Training Setup

The core training configurations used to optimize the model performance, including network initialization, hyperparameter tuning, learning rate strategy, and training execution environment, are empirically tuned and listed below in **Table 14**.

Parameter	Value
Optimizer	Adam
Initial Learning Rate	1e-4
L2 Regularization Factor	0.0001
Mini-Batch Size	4
Max Epochs	30
Iterations per Epoch	20
Max Iterations	600

Shuffle	Every Epoch
Validation Frequency	Every 50 Iterations
Verbose Output	Enabled
Execution Environment	Single GPU
Data Normalization	Automatic per-channel

Table 14. Training Hyperparameters for DeepLab v3+

The learning rate was fixed throughout training, and no custom learning rate schedule or early stopping was applied due to consistent improvements in training and validation accuracy.

The training was performed on a single GPU using MATLAB's trainNetwork function. Automatic environment detection ensured efficient use of GPU memory and enabled real-time display of training metrics. The model reached convergence in under two minutes of training time.

A live training plot, shown in the **Figure 37**, displayed metrics including:

- Training accuracy and loss
- Validation accuracy and loss
- Iteration-wise updates per mini-batch

The final model achieved high training and validation performance, suggesting the selected setup was effective for the binary segmentation task with a relatively small dataset.

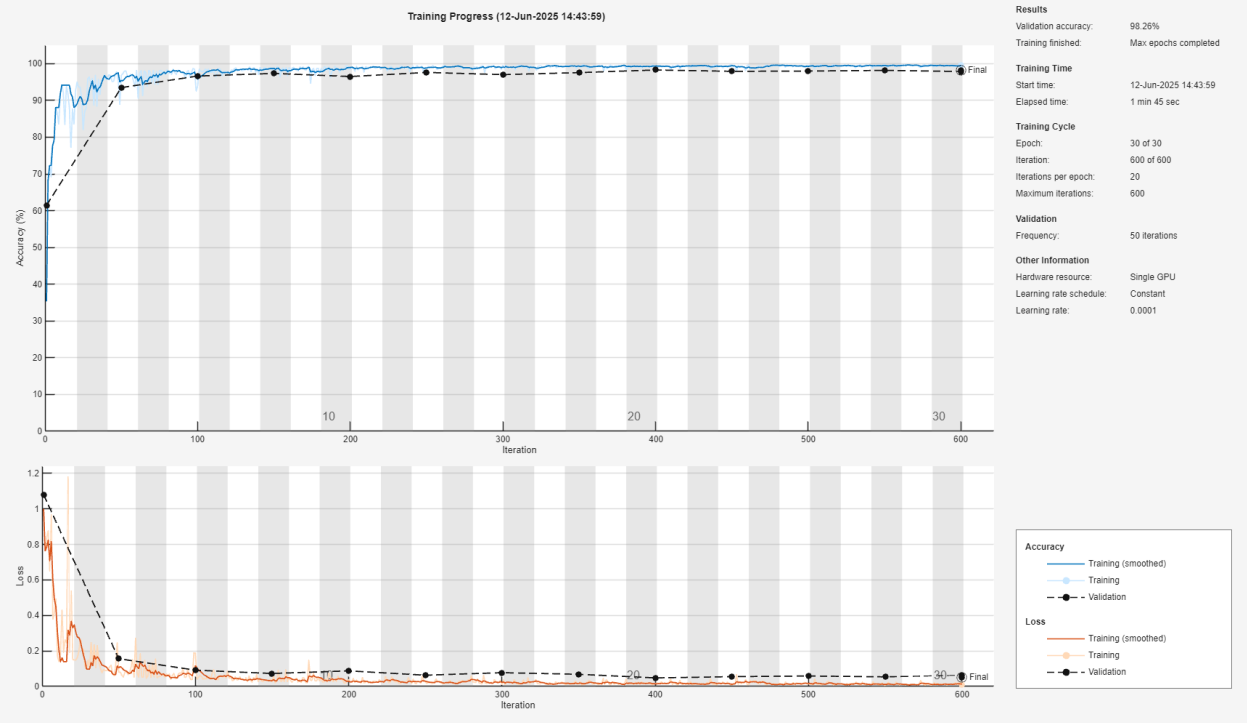


Figure 37. Training progress of DeepLab v3+ for deck identification

At the end of training, the final mini-batch achieved an accuracy of 99.59%, while the validation accuracy reached 98.26%, indicating strong generalization. These results confirm that the selected configuration enabled highly effective learning, even with a relatively small dataset. These results are summarized in **Table 15**.

Metric	**Value**
Mini-batch Accuracy	99.59%
Final Validation Accuracy	98.26%
Elapsed Time	1 min and 45 sec

Table 15. Results of the training progress for deck identification

6.5 Results and Evaluation

To evaluate the performance of the model, the following metrics have been used:

- Global Accuracy: Ratio of correctly classified pixels to the total number of pixels.
- Mean Accuracy: Average of per-class pixel accuracies.
- Mean IoU (Intersection over Union): Average IoU across classes.
- Weighted IoU: IoU weighted by pixel count per class.
- Mean Boundary F1 Score (BF Score): Boundary matching score assessing segmentation contour alignment.

The model was evaluated on a test set comprising 16 high-resolution images that were not seen during training or validation. The evaluation metrics obtained are summarized in **Table 16**.

Metric	**Score**
Global Accuracy	98.63%
Mean Accuracy	98.70%
Mean IoU	97.18%
Weighted IoU	97.30%
Mean BF Score	92.61%

Table 16. Metrics for the overall performance of the DeepLab v3+ model

These results are interpreted below:

- Global Accuracy (98.63%): Indicates that nearly all pixels were correctly classified across the entire test set. This is a strong indicator of generalization to unseen data.
- Mean Accuracy (98.70%): Reflects high consistency in performance across both deck and background classes, without bias.

- Mean IoU (97.18%): IoU is one of the most rigorous segmentation metrics, requiring a high overlap between predicted and actual areas. A score over 97% confirms precise shape and region prediction.
- Weighted IoU (97.30%): Slightly higher than Mean IoU due to the dominance of background pixels, showing that the model performs well even when accounting for class imbalance.
- Mean Boundary F1 Score (92.61%): A strong BF score indicates that object boundaries (e.g., deck edges) are sharply predicted.

This level of segmentation performance confirms that the DeepLab v3+ model is highly effective at identifying bridge deck surfaces.

To further analyze the segmentation performance of the trained model, we examine per-class metrics that offer insight into how well each class (deck, background) was identified across the test set. **Table 17** presents the results for accuracy, IoU, and boundary precision (BF Score) for each semantic class:

Class	Pixel Accuracy	Intersection over Union (IoU)	Mean Boundary F1 Score (BF Score)
Background	98.35%	97.75%	94.54%
Deck	99.05%	96.61%	90.68%

Table 17. Class-wise evaluation results

Despite deck areas often featuring variable textures, lighting changes, or surface defects, the model maintained exceptional accuracy and shape alignment. The deck class achieved a remarkably high accuracy of 99.05%, meaning nearly all deck pixels were correctly identified.

Finally, **Figure 38** presents sample visual comparisons of the model's predictions against the ground truth. These visualizations affirm that the model preserves the structural integrity of deck boundaries even under varying textures and lighting conditions.

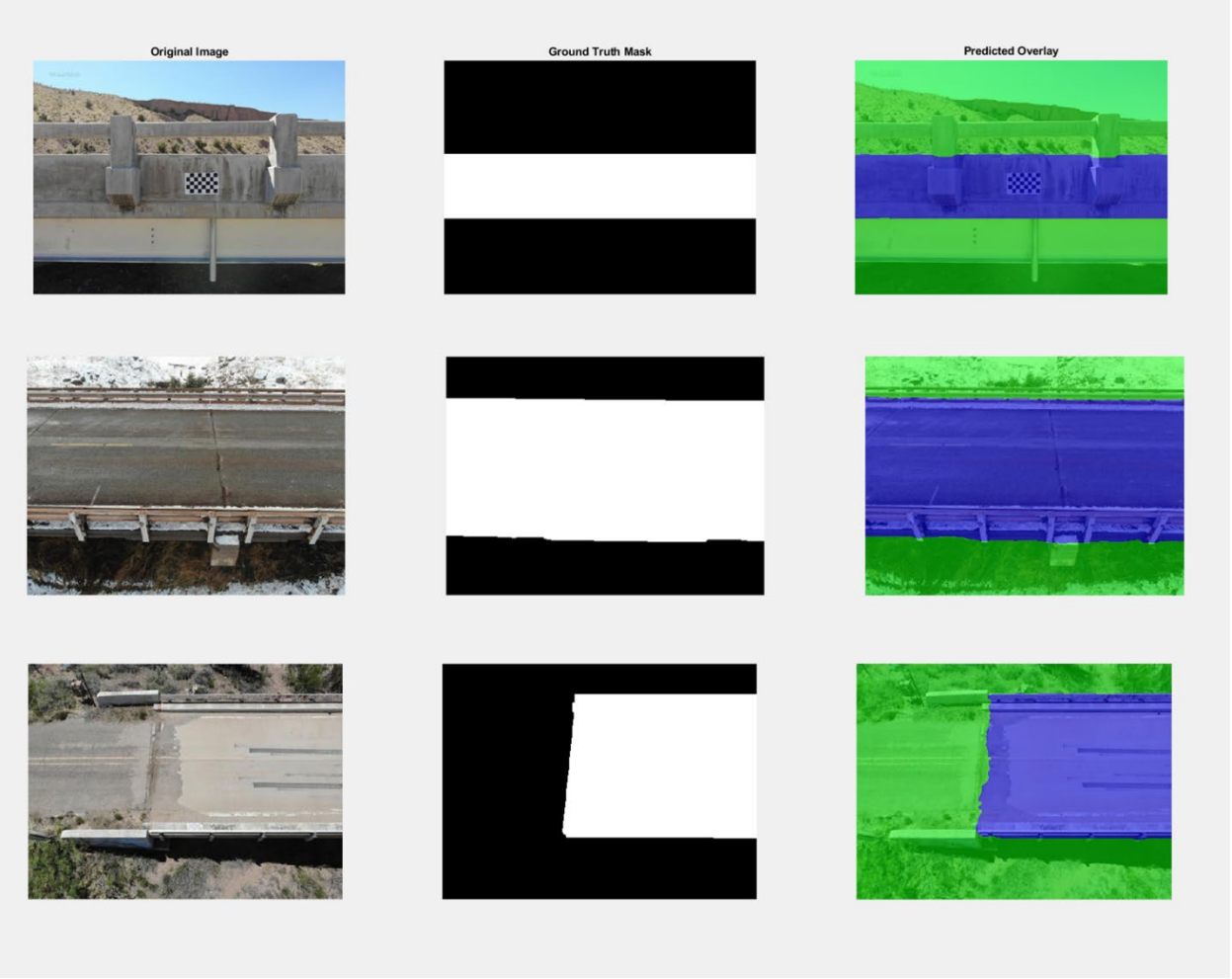


Figure 38. Samples of the DeepLab v3+ model's predictions

TASK 7: Crack Detection and Quantification by Using Deep Learning

The integration of UAVs and AI, especially deep learning, has emerged as a transformative approach to automate, expedite, and enhance the reliability of damage detection processes [46]. Among various structural deficiencies, cracks are particularly significant because they often serve as early indicators of progressive deterioration. Early and accurate crack detection and quantification can prevent catastrophic failures and inform timely maintenance decisions. However, automated crack detection remains challenging due to the variability of crack shapes, sizes, background textures, lighting conditions, and noise present in real-world images [47], [48].

Deep learning, particularly convolutional neural networks (CNNs), has demonstrated remarkable potential in addressing these challenges by learning hierarchical features directly from raw image data without relying on handcrafted features [49]. In this study, a U-Net model, an encoder-decoder convolutional architecture originally developed for biomedical image segmentation [50], was adapted and trained for the pixel-wise segmentation of cracks in bridge structures.

Despite the advancements, several limitations persist in existing studies:

- Dataset Size and Diversity: Many models are trained on limited datasets with relatively homogeneous backgrounds, making them less robust to diverse real-world conditions [18].
- Resolution Challenges: Crack detection requires high-resolution imagery to capture thin and fine cracks, but training on high-resolution datasets increases computational demands.
- Quantitative Evaluation: While qualitative results are promising, detailed quantitative analyses, particularly at the pixel level (e.g., Mean Intersection over Union (IoU), Dice coefficient), are often missing.
- UAV Deployment: Few studies rigorously integrate their methods into UAV workflows, limiting real-world applicability.

This study contributes to the state-of-the-art by employing an extensive dataset, applying tailored preprocessing steps, optimizing the U-Net architecture for crack detection, and performing detailed quantification through pixel-level outputs. This combination offers a novel, high-fidelity system suitable for UAV-based bridge inspection applications, a domain where achieving high precision in uncontrolled environments remains an ongoing challenge.

For pixel-wise labeling, 20,000 positive and 20,000 negative images have been collected from the available dataset. In order to annotate all the images, which can be a time-consuming process, a method has been developed to automate labeling. For this purpose, first, a small set of images has been labeled to train a light model so the rest of the images can be labeled automatically. Finally, after auto-labeling the rest of the dataset images and refining the labels, the main model will be trained. The steps of this dual-phase pixel-wise crack detection are summarized below:

Step 1: Small Dataset Pixel-wise labeling

240 images have been chosen for the first set of labeling, which has been performed by using MATLAB labeling toolbox. To improve the accuracy of the labeling, several considerations have been included during the labeling process as follows:

- Noise Handling: To avoid including non-crack areas (e.g., background textures) in the segmentation to improve model accuracy.
- Edge Refinement: To ensure that the labeling follows the crack edges and captures the full extent of the crack, including thinner sections.
- Point Density: To ensure sufficient density for a smooth boundary and add more points if finer details need to be captured.

The labeled crack masks have been converted into binary segmentation masks (background = 0, crack = 1) to be used as the inputs of the model. **Figure 39** shows two examples of labeled images and binary masks.

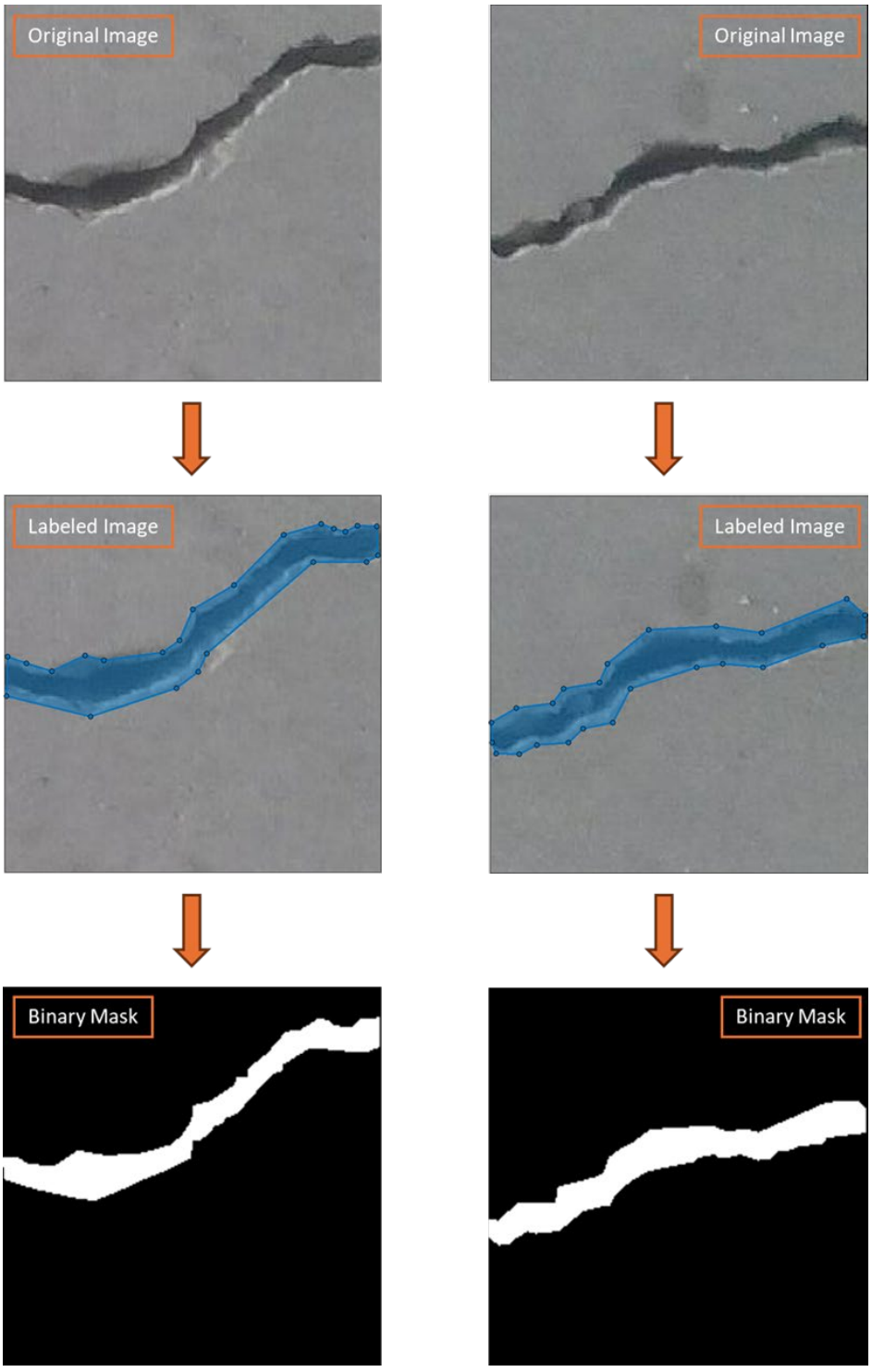


Figure 39. Pixel-wise labeled images and their binary masks

Step 2: Pre-training Data Preparation

In order to prepare data for model training, several adjustments have been applied as follows:

1. Image and Mask Resizing:

To ensure compatibility with the U-Net model, all images and their corresponding masks were resized from 227×227 pixels to 224×224 pixels. This standardization facilitates efficient processing while maintaining the integrity of structural details in the dataset.

2. Data Splitting Strategy:

   The dataset was divided into three subsets to train, validate, and evaluate the model effectively:

   - Training Set: 70% of the data
   - Validation Set: 15% of the data
   - Testing Set: 15% of the data

   This distribution ensures that the model learns effectively from a large portion of the data while being validated and tested on separate, unseen samples.

3. Data Augmentation Techniques:

   To enhance the model's generalization ability and robustness, a series of augmentation techniques were applied to the training set:

   - Random Rotation: Images were randomly rotated within the range of -45° to 45°.
   - Random Reflections: Horizontal (X-axis) and vertical (Y-axis) reflections were applied.
   - Random Scaling: Images were scaled randomly within a factor range of 0.8 to 1.5.
   - Random Translation: Horizontal and vertical translations were applied within the range of -30 to 30 pixels.

   These augmentations help improve the model's ability to recognize patterns under different transformations, ultimately enhancing its robustness in real-world applications.

Step 3: Light Model Training for Autolabeling

As mentioned above, this model training phase is only to automate labeling, and for this purpose, the required model should be able to handle the requirements mentioned below:

- Handles pixel-wise segmentation with high accuracy
- Works well with grayscale images
- Deals with class imbalance effectively
- Generalizes well across different input variations

U-Net fulfills all these requirements by providing a strong balance between accuracy, efficiency, and adaptability.

U-Net is a deep learning model designed for image segmentation, where the goal is to classify each pixel in an image. It was originally developed for biomedical image segmentation but has

since been widely used in various fields, including remote sensing, medical imaging, and structural damage detection [50].

U-Net follows a fully convolutional neural network (FCNN) architecture, meaning it does not use fully connected layers like traditional CNNs. Instead, it is structured in a U-shaped design, consisting of two main parts:

1. Encoder (Contracting Path) – Extracts important features from the input image by progressively reducing its size through convolutional and pooling layers.
2. Decoder (Expanding Path) – Reconstructs the image by gradually increasing its resolution and using skip connections to retain fine details.

Figure 40 shows the layers of the U-Net model used in this study for 224×224-pixel images for binary segmentation. As shown in this figure, the model consists of 58 layers.

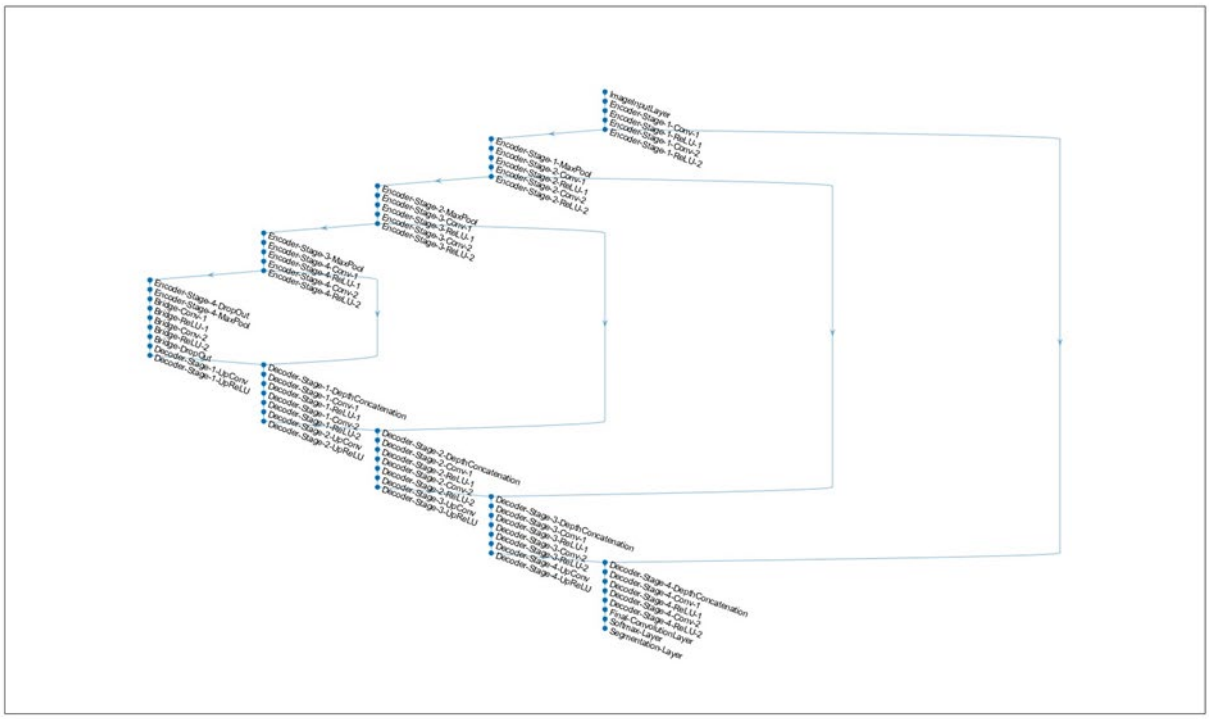


Figure 40. Layers of the U-Net model for this study

To define the U-Net architecture for the purpose of this study, several adjustments have been applied as follows:

- Adding dropout layers: Dropout is a technique used to prevent overfitting, which happens when a model performs well on training data but poorly on new data. It randomly turns off some neurons during training, forcing the model to learn robust features. Three dropout layers have been inserted after activation layers in different encoding stages. These dropout layers introduce randomness, which makes the model more generalizable and resistant to overfitting.

- Handling class imbalance using class weights: In many segmentation tasks, the foreground class (objects of interest) appears much less frequently than the background class. If the model is trained without adjustments, it might ignore the foreground class since predicting the background everywhere would still give high accuracy [51]. To solve this problem, a higher importance (weight) of 50 has been assigned to the foreground class (cracks). This means that the model will be penalized 50 times more for misclassifying a foreground pixel compared to a background pixel. By doing this, the model will focus more on learning foreground pixels, even though they appear less frequently.
- Optimization algorithm: Adam, an advanced optimization algorithm that adjusts the learning rate automatically to speed up training and improve convergence, has been chosen for this study [52].
- Controlling learning rate: The learning rate determines how much the model updates its parameters after each training step. By applying a piecewise learning rate schedule, after a certain number of epochs, the learning rate is reduced to help the model refine its learning. For this study, after every 50 epochs, the learning rate is multiplied by 0.5, allowing the model to fine-tune its predictions.
- Epochs and batch size: The model has been trained for 200 epochs, which is enough to learn useful features without excessive training. Also, the mini-batch size was set to 8, meaning that 8 images are processed simultaneously.

The above-mentioned considerations ensure that the model is optimized for segmentation tasks, robust to class imbalance, and resistant to overfitting. After setting the training options, the model has been trained, and the training progress is shown in **Figure 41**.

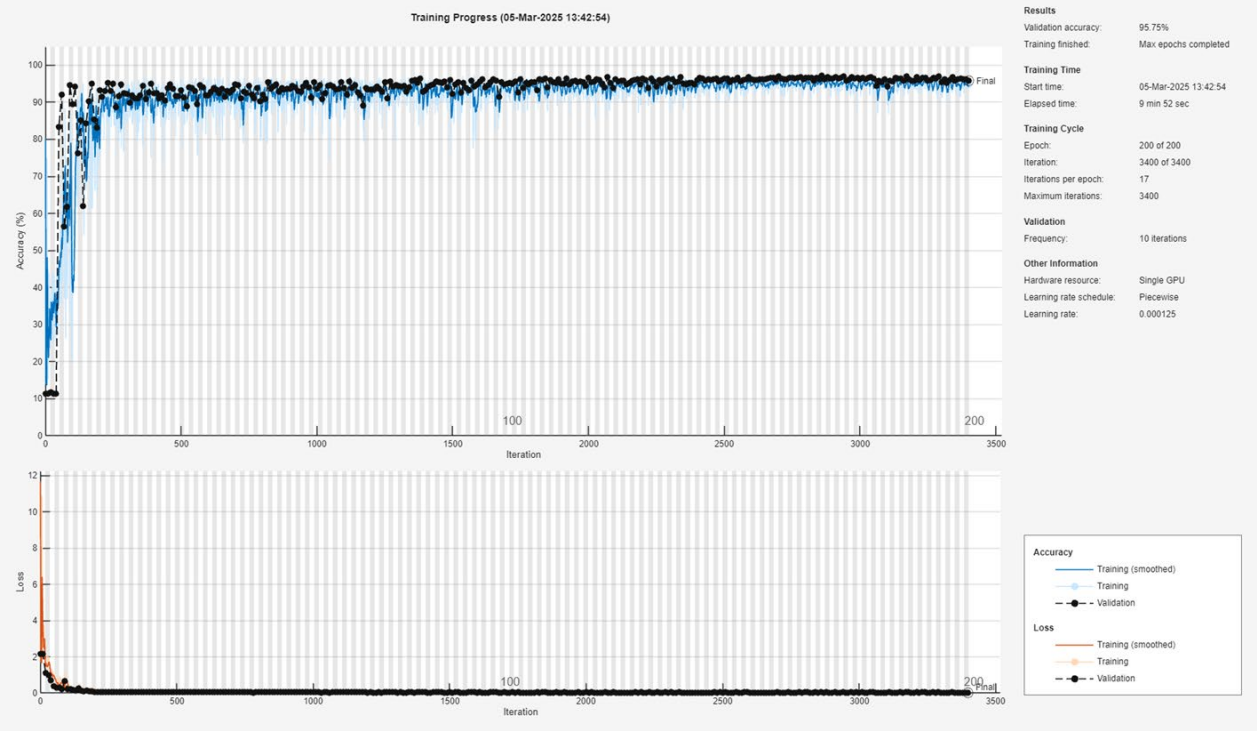


Figure 41. Training progress of the light U-Net model

As shown in **Figure 41**, the training progress consists of two subplots. The top plot shows the accuracy of the model during training (blue line) and validation (black dots). The bottom plot shows the loss function values over iterations for training (orange line) and validation (black dots). By analyzing the training progress and subplots, the following conclusions have been made:

- Training accuracy starts low (~10%) but improves rapidly within the first few epochs.
- By the end of training (epoch 200), training accuracy stabilizes close to 97%.
- Validation accuracy starts lower but quickly improves, suggesting the model learns meaningful features.
- Final validation accuracy reaches 95.75%, indicating strong generalization to unseen data.
- The validation accuracy curve remains relatively stable, implying no significant overfitting.
- Validation loss closely follows training loss, confirming that the model generalizes well.
- Training completed in 9 minutes 52 seconds, demonstrating fast convergence.

The training results indicate that the U-Net model was successfully trained for the auto-labeling task. **Table 18** shows the summary of the training progress results.

Metric	Value
Final Training Accuracy	96.41%

Final Validation Accuracy	95.75%
Total Epochs	200
Iteration per Epoch	17
Total Iterations	3400
Elapsed Time	9 min and 52 sec

Table 18. Training progress results of the light U-Net model

Step 4: Auto-Labeling of the Dataset and Noise Reduction

After training the light U-Net model, the whole dataset (20,000 positive + 20,000 negative) has been auto-labeled. Moreover, to refine the auto-labeled mask, morphological operations have been applied to the masks, so the noises are automatically corrected. **Figure 42** shows some examples of the auto-labeled images and their corresponding masks after automatic corrections.

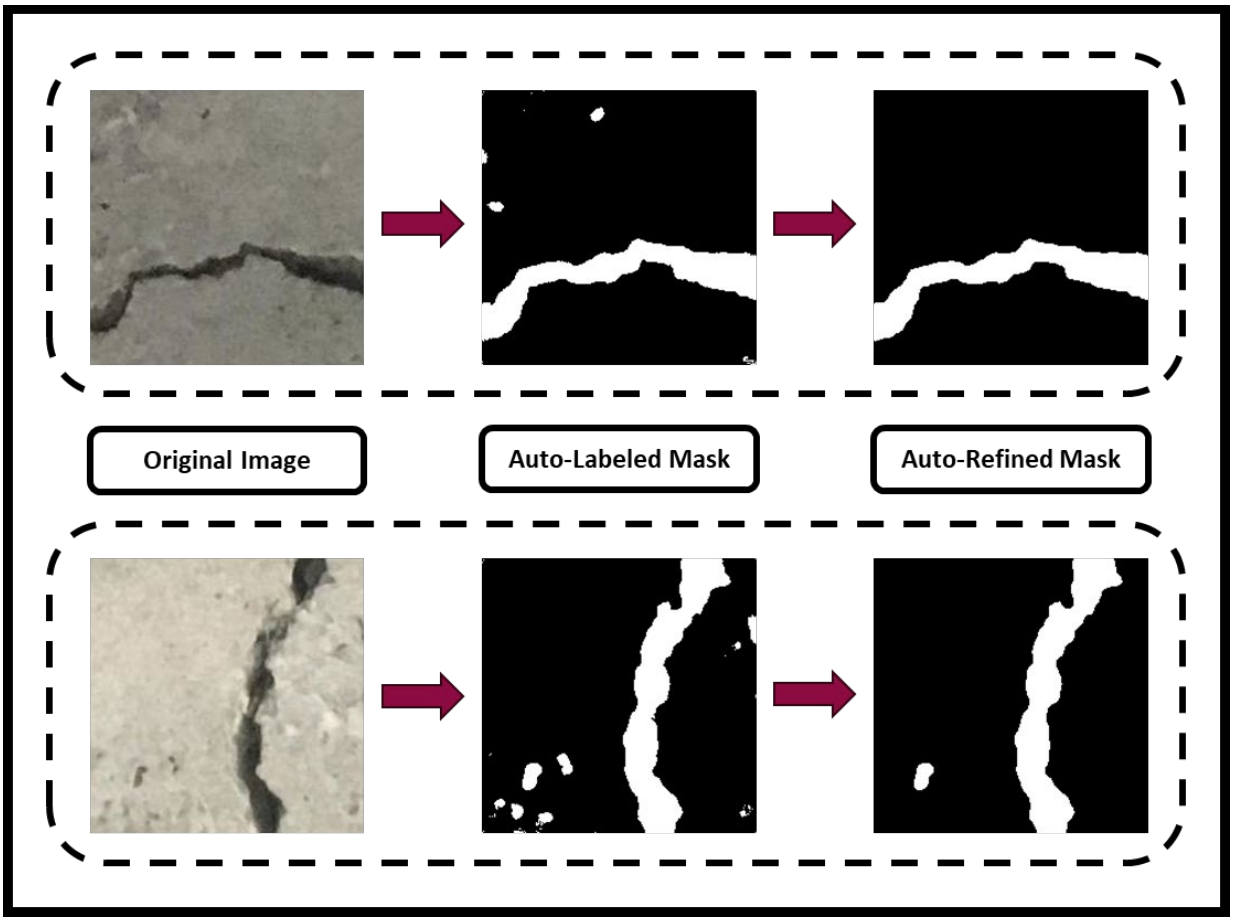


Figure 42. Auto-labeled masks and auto-refined masks of the original images

Step 5: Main U-Net Model Training

For the main U-Net model training by using the auto-labeled dataset, the data is split as follows:

- Training: 80% (32,000 images)
- Validation: 10% (4,000 images)
- Testing: 10% (4,000 images)

Figure 43 shows the training progress of the main U-Net model, and its results are provided in **Table 19**.

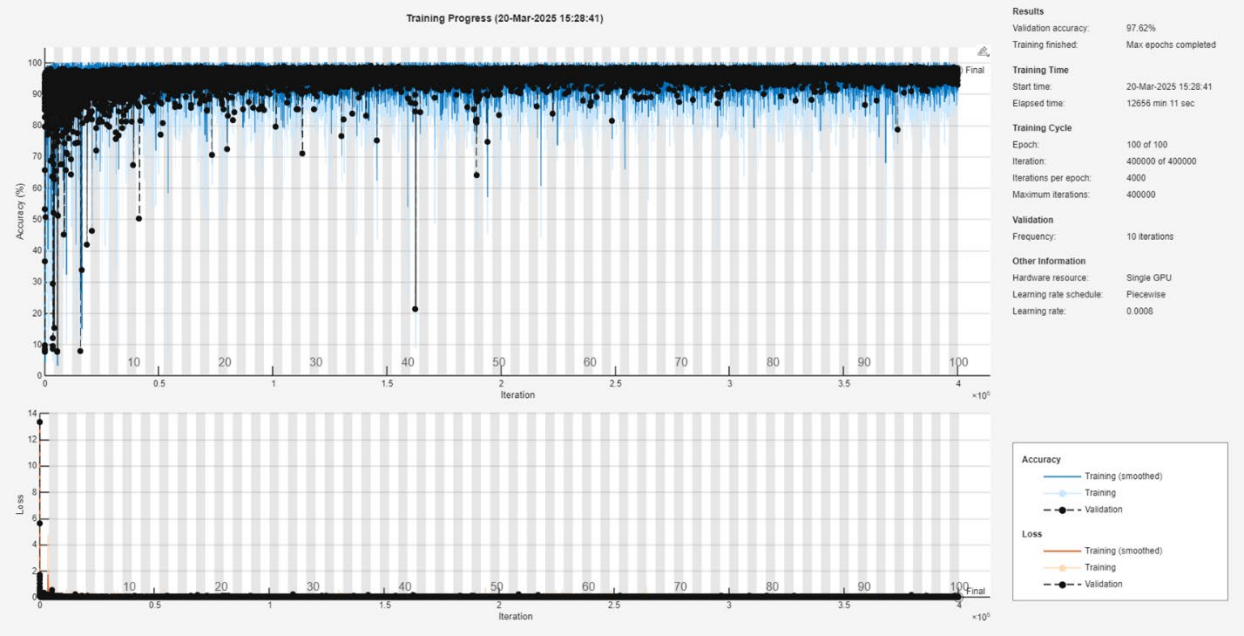


Figure 43. Training progress of the main U-Net model for crack detection

Metric	Value
Final Training Accuracy	95.58%
Final Validation Accuracy	97.62%
Total Epochs	100
Iteration per Epoch	4000
Total Iterations	400,000
Elapsed Time	12,656 min

Learning Rate	1e-3
Learning Rate Drop Factor	0.8

Table 19. Results of the training progress of the main U-Net model

As it is shown in **Table 19**, the model was trained in 12,656 minutes, and it reached a validation accuracy of 97.62%.

Step 6: Evaluation of the U-Net Segmentation Performance

After training the U-Net model, it was evaluated on test images using three standard segmentation metrics: Intersection over Union (IoU), Dice Similarity Coefficient (DSC), and Pixel Accuracy. These metrics provide insights into the model's effectiveness in segmenting cracks from images.

Intersection over Union (IoU):

IoU, also known as the Jaccard Index, measures the overlap between the predicted segmentation mask and the ground truth mask [53]. It is defined as:

$$IoU = \frac{|Prediction \cap GroundTruth|}{|Prediction \cup GroundTruth|} \quad (1)$$

Where, intersection represents correctly predicted pixels, and the Union includes all pixels present in either the prediction or ground truth. From the results, the mean IoU obtained for test images is 0.7538, indicating a very good overlap between predicted crack regions and ground truth masks.

Dice Similarity Coefficient (DSC):

The Dice Similarity Coefficient (DSC), also called the F1-score for segmentation, is a commonly used metric for evaluating the similarity between two sets. It is computed as:

$$DSC = \frac{2\times|Prediction \cap GroundTruth|}{|Prediction|+|GroundTruth|} \quad (2)$$

The mean DSC for test images is 0.8532, which suggests that the model performs well in segmenting crack regions.

Pixel Accuracy:

Pixel Accuracy is a simple but effective metric that calculates the proportion of correctly classified pixels over the entire image and is computed as [54]:

$$Pixel\ Accuracy = \frac{Correctly\ Classified\ Pixels}{Total\ Pixels} \quad (3)$$

The mean pixel accuracy obtained is 0.9504, meaning that 95.04% of all pixels were classified correctly. Higher pixel accuracy (>90%) indicates strong overall performance.

Table 20 summarizes the evaluation results. These values confirm that the model is effective for crack detection, with potential for further refinement.

Metric	Result	Interpretation

IoU	75.38%	Excellent - very good overlap with ground truth
DSC	85.32%	Very strong match - above 0.85 is highly desirable
Pixel Accuracy	95.04%	The majority of pixels were correctly classified.

Table 20. Evaluation results of the U-Net model

Finally, **Figure 44** shows some examples of the predictions.

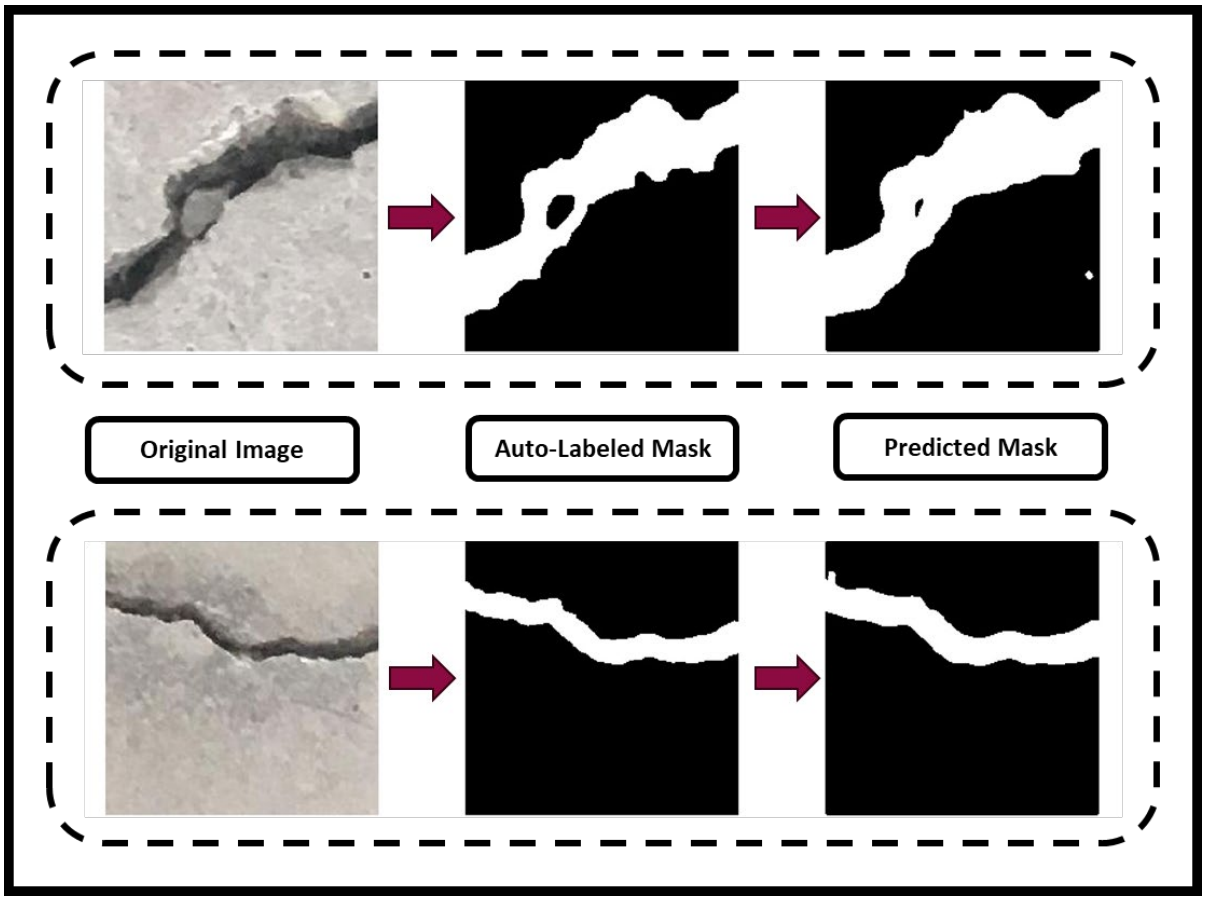


Figure 44. Crack prediction samples by using the U-Net model

TASK 8: Automated Bridge Deck Health Evaluation Aligned with NBI Ratings

8.1. Methodology

The methodology of this study proposes a fully automated, unsupervised anomaly detection pipeline for bridge deck inspection using UAV-acquired imagery and a sparse autoencoder. The pipeline is designed to process multiple high-resolution images from various angles of a bridge

deck, detect local surface anomalies through reconstruction errors, and aggregate the findings into a global bridge condition score that aligns with the National Bridge Inspection (NBI) standards. **Figure 45** shows the general flowchart and steps developed for the proposed methodology. These steps are described in detail in the following subsection.

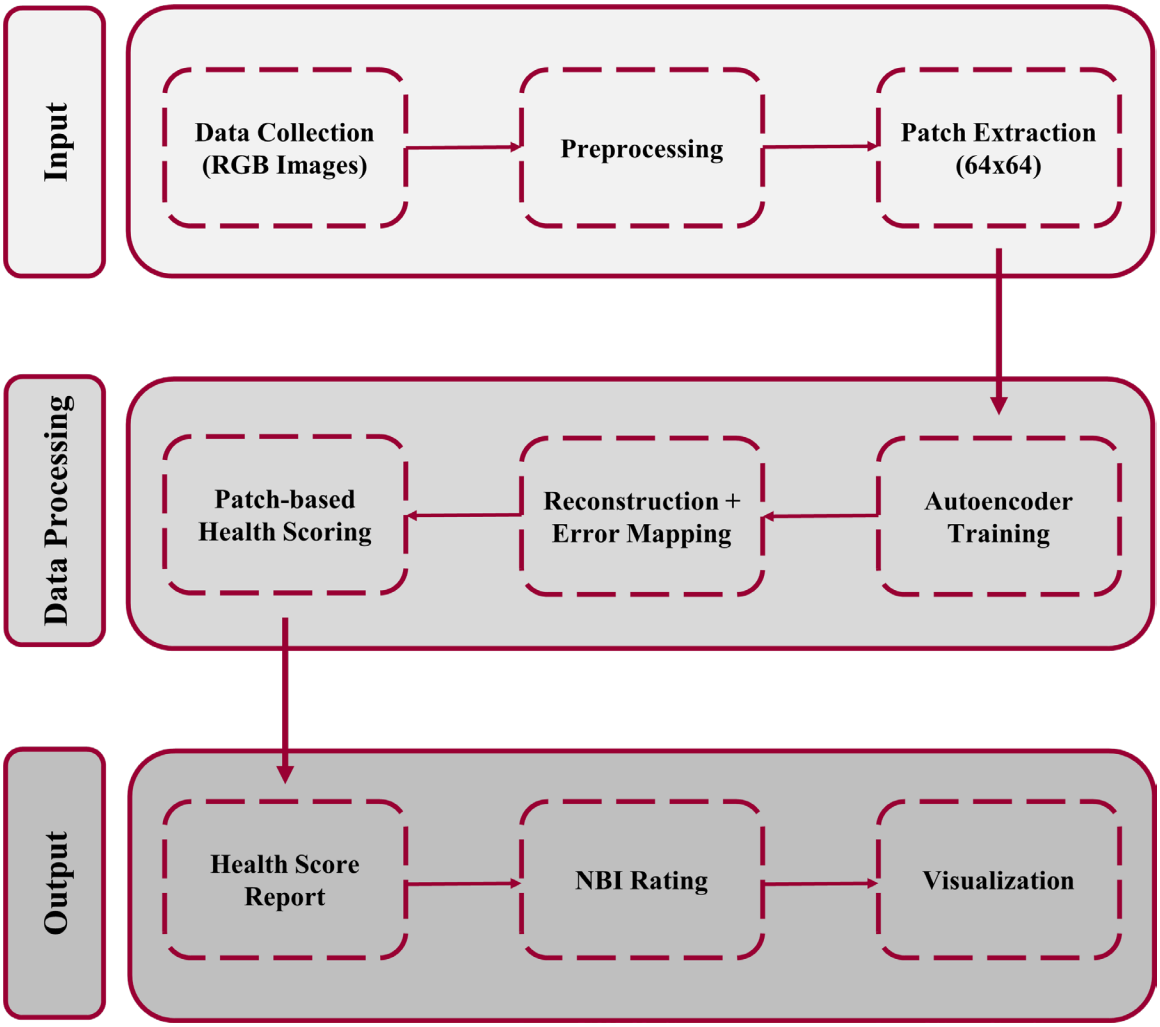


Figure 45. Flowchart of the methodology for automated bridge deck health evaluation aligned with NBI ratings

8.1.1 Data Collection, Preprocessing, and Patch Extraction

In the first step, high-resolution bridge deck images were collected using a DJI Mavic 2 Enterprise UAV equipped with a standard RGB camera. To maintain image clarity and geometric consistency, the UAV flights were performed at a controlled altitude (typically 3–5 meters above the deck surface), which corresponds to a ground sampling distance (GSD) of 1-1.5 mm/pixel. Each image captured covers a portion of the bridge deck with visible concrete texture, cracks, shadows, and surface features to enhance the model's learning even under various lighting or texture conditions. The typical image resolution was 4056 × 3040 pixels, allowing for fine-grained

patch extraction. Images were cropped to remove irrelevant backgrounds (e.g., sky, trees, traffic barriers) and focus exclusively on the concrete surfaces of the deck, including the top surface, underneath, and sides.

Once the images are acquired, each captured image undergoes a standardized preprocessing pipeline to ensure consistency and readiness for the anomaly detection stage. The first step involves grayscale conversion, where each RGB image is transformed into a single-channel grayscale image. This is based on the observation that most structural defects on concrete surfaces manifest through intensity variations (e.g., cracks, delaminations) rather than chromaticity. Grayscale conversion reduces data dimensionality and computational load without sacrificing relevant features. Subsequently, if the input images contain high-frequency sensor noise or compression artifacts, a mild Gaussian smoothing filter is applied.

The preprocessed grayscale images are then divided into non-overlapping square patches of size 64×64 pixels, forming the fundamental input unit for the unsupervised learning model and balancing between resolution and coverage. Each patch is flattened into a one-dimensional vector of 4096 elements and normalized to the range [0,1]. This normalization step ensures consistent intensity scaling across all patches and avoids numerical instability during model training. Formally, let $I_k \in \mathbb{R}^{H \times W}$ represent the k-th grayscale bridge deck image, where $k = 1, 2, \ldots, K$. Also, H and W are the image height and image width in pixels, respectively. Each image is partitioned into N_k patches $\{x_{k,1}, x_{k,2}, \ldots, x_{k,N}\}$, where each patch $x_{k,i} \in \mathbb{R}^{64 \times 64}$, and flattened to $x_{k,i} \in \mathbb{R}^{4096}$ for compatibility with the autoencoder. This preprocessing pipeline ensures that all patches used for analysis are spatially consistent, resolution-normalized, and intensity-standardized, enabling robust feature learning and reconstruction-based anomaly detection in subsequent stages. **Figure 46** shows an example of the process for data collection, preprocessing, and patch extraction.

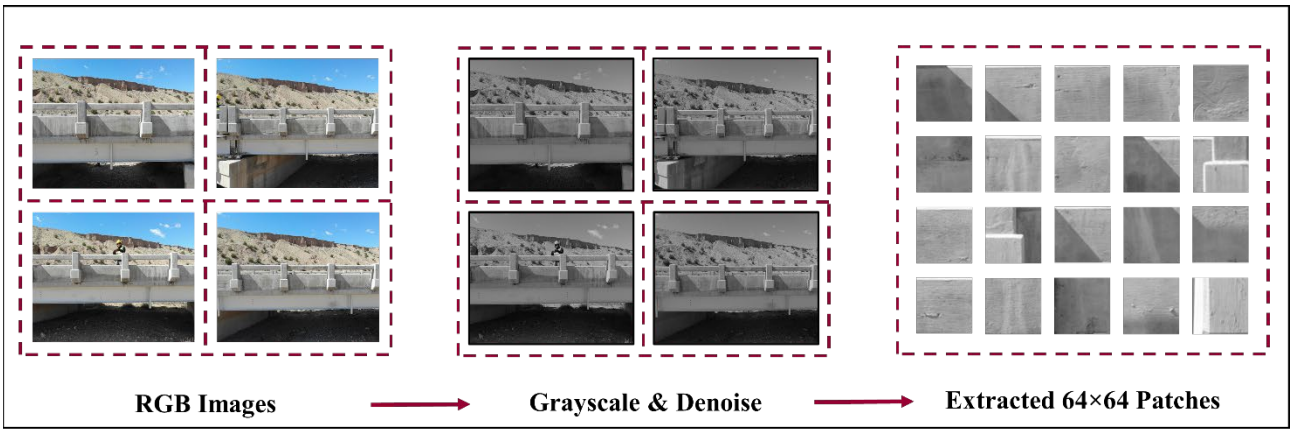


Figure 46. An example of the process of data preprocessing and patch extraction

It is worth noting that filtering was performed to remove patches containing sky, vegetation, or heavily obstructed views, ensuring that only surface-representative patches were used for training. In order to enhance the robustness and accuracy of the autoencoder, additional negative (without

any defects) images of the concrete surface have been added to UAV-collected data. Therefore, the total number of the extracted 64×64 patches for autoencoder training is 36,000.

8.1.2 Sparse Autoencoder Training

The training set consisted exclusively of normal (non-anomalous) data, ensuring the model learned a compact latent representation of healthy patterns only. The core of the proposed framework is an unsupervised sparse autoencoder neural network, which has been trained in MATLAB using the trainAutoencoder function [19]. The architecture comprised a single hidden layer with 200 neurons. Sparsity constraints were enforced using Kullback–Leibler divergence to encourage the network to activate only a subset of neurons for any given input, thereby learning discriminative features of healthy surfaces. Additionally, the input and output layers contain 4096 neurons due to the flattened 64×64 patches. **Figure 47** represents the architecture of the sparse autoencoder neural network.

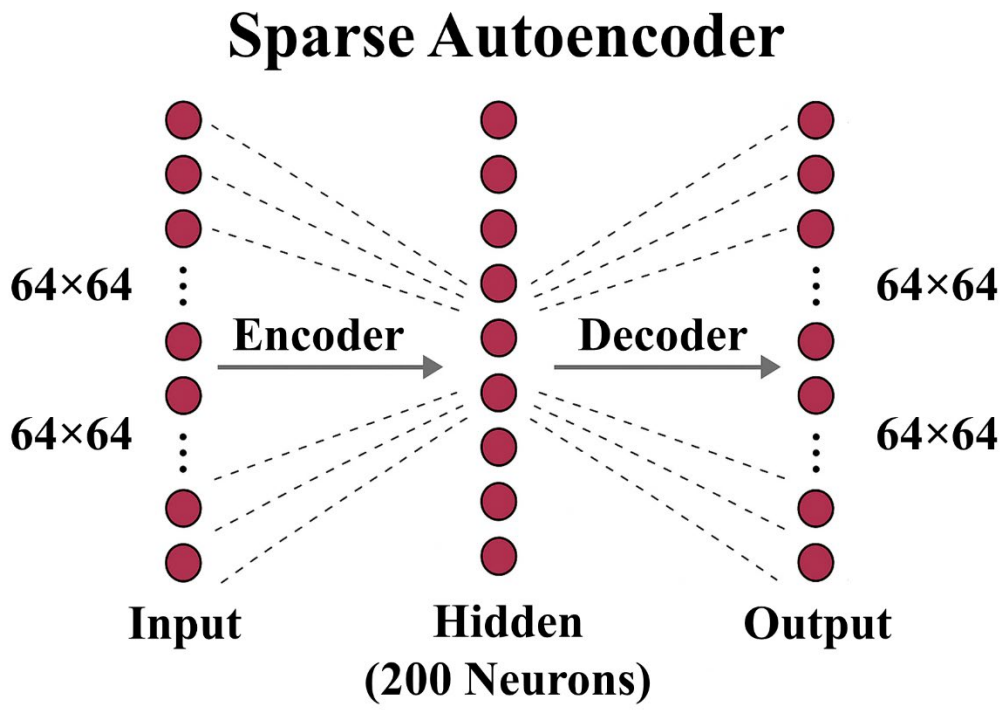


Figure 47. Architecture of the sparse autoencoder neural network

The training is performed using the Scaled Conjugate Gradient (SCG) algorithm, which is a second-order optimization method combining the speed of conjugate gradient descent with efficient step-size scaling. Unlike traditional backpropagation with basic stochastic gradient descent, SCG avoids line searches, accelerates convergence, and is well-suited for smooth error surfaces like reconstruction loss. Also, L2 weight regularization is used to prevent overfitting by penalizing large weights that could memorize training data. Moreover, sparsity regularization is used to enforce that each hidden neuron remains inactive for most inputs, promoting part-based representations. Finally, the sparsity proportion is used to set the desired average activation per hidden unit. A value of 0.05 encourages most neurons to remain off for any given patch, ensuring

that only meaningful features are encoded. The configuration of these hyperparameters is summarized in **Table 21**.

Hyperparameter	**Value**
Epochs	200
L2 Regularization	0.001
Sparsity Regularization	4
Sparsity Proportion	0.05
Optimizer	Scaled Conjugate Gradient (SCG)
Loss Function	Mean Squared Error (MSE)

Table 21. Configuration of the model's hyperparameters

The reconstruction error for each patch was computed using Mean Squared Error (MSE) by the following equation:

$$MSE = \frac{1}{4096}\sum_{i=1}^{4096}(x_i - \hat{x}_i)^2 \qquad (1)$$

where x_i and $\hat{x}_i$ are the original and reconstructed pixel intensities, respectively.

The performance plot of the sparse autoencoder neural network training is visualized in **Figure 48**. The model exhibited strong convergence behavior over the course of 200 epochs with an elapsed time of 31 minutes, reaching a final MSE of approximately 0.00299. This steady decline in error validates the model's ability to accurately reconstruct non-defective inputs and learn a meaningful latent representation of healthy bridge textures. The gradient magnitude decreased from 0.579 to 0.000147, confirming that the optimizer approached a local minimum with smooth convergence.

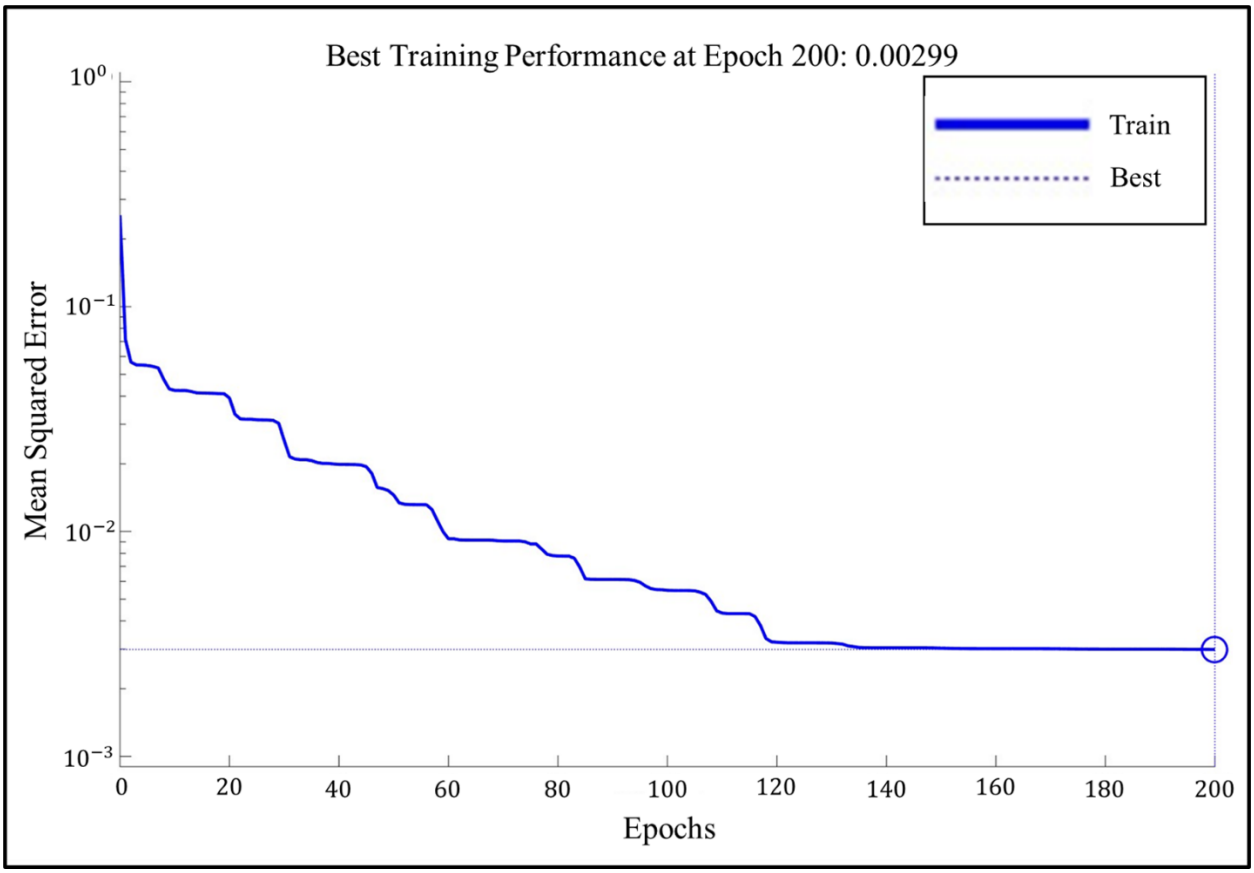


Figure 48. Training performance of the sparse autoencoder

All MSE values were aggregated into a 2D error map, with each value representing the anomaly score for a corresponding spatial region in the image. The error map was resized (via bicubic interpolation) to match the original image resolution and normalized to [0,1]. To enhance interpretability, the error map was converted to a heatmap using a perceptually intuitive hot colormap. The heatmap was overlaid on the original image to highlight potential anomalies, where darker areas mean fewer anomalies and lighter areas represent higher anomalies.

Given the unsupervised design and lack of pixel-level anomaly ground truth, evaluation relied on visual and statistical measures. To quantify the anomaly heatmap into actionable metrics, the following are computed:

- Average Error: Mean of reconstruction error across all patches, which indicates the global health.

$$AE = \frac{1}{N}\sum_{i=1}^{N} MSE_i \qquad (2)$$

Where N is the total number of patches.

- Anomalous Area (%): Percentage of patches with error above a threshold $\theta = 0.015$, which indicates spread.

$$AA = \frac{1}{N}\sum_{i=1}^{N} 1(MSE_i > \theta) \times 100 \quad (3)$$

- Severity Score: Sum of excess error beyond the threshold $\theta = 0.015$, which indicates damage intensity.

$$Severity = \frac{1}{N}\sum_{i=1}^{N} max(0, MSE_i - \theta) \quad (4)$$

The threshold mentioned above is chosen based on the sensitivity analysis, which is discussed in the next subsection.

To provide a standardized interpretation of the computed anomaly metrics, the results were mapped to National Bridge Inventory (NBI) condition rating guidelines for decks. Based on the average error, anomalous area, and normalized severity score from multiple images of a single bridge deck, a rule-based classifier was used to assign an NBI-aligned rating between 1 (critical) and 9 (excellent). The correlation between these metrics and the NBI rating, resulting from trial and error, is summarized in **Table 22**.

Condition (Rating)	Average Error	Anomalous Area	Severity Score
Excellent (9)	AE ≤ 0.0020	AA ≤ 5%	SS ≤ 0.0005
Very Good (8)	0.0020 < AE ≤ 0.0035	5% < AA ≤ 10%	0.0005 < SS ≤ 0.0010
Good (7)	0.0035 < AE ≤ 0.0050	10% < AA ≤ 15%	0.0010 < SS ≤ 0.0020
Satisfactory (6)	0.0050 < AE ≤ 0.0100	15% < AA ≤ 30%	0.0020 < SS ≤ 0.0035
Fair (5)	0.0100 < AE ≤ 0.0150	30% < AA ≤ 45%	0.0035 < SS ≤ 0.0055
Poor (3-4)	0.0150 < AE ≤ 0.0200	45% < AA ≤ 60%	SS > 0.0055
Critical (1-2)	AE > 0.0200	AA > 60%	SS > 0.0055

Table 22. Correlation of metrics with NBI rating

Finally, based on the above-mentioned heatmap overlay and these correlations with NBI rating, a full report of the bridge's deck and its anomaly visualization is provided. The next section discusses the model training results and these reports and visualizations.

8.2. Sensitivity Analysis and Experimental Validation

This section first discusses the selection of the above-mentioned threshold, which is based on the sensitivity analysis. Then it discusses the experimental validations from real-world scenarios based on the proposed methodology and the selected threshold.

8.2.1 Threshold Sensitivity Analysis

To evaluate the robustness and reliability of the anomaly detection process, a sensitivity analysis was performed over a range of reconstruction error thresholds from 0.010 to 0.025, with increments of 0.001. For each threshold value, two key metrics are computed: (i) the percentage of anomalous patches, defined as the fraction of image patches exceeding the threshold, and (ii)

the normalized severity score, calculated as the mean amount by which the anomalous patches exceeded the threshold.

The sensitivity analysis was performed independently on images from 10 bridges located in New Mexico to observe how the system's response varied under different surface conditions. In all cases, increasing the threshold led to a monotonic decline in both anomalous percentage and severity score. However, the rate of decline varied between bridges. These variations highlight the importance of careful threshold selection. A lower threshold may be overly sensitive, capturing minor surface variations and increasing false positives, while a higher threshold may miss early-stage damage. Based on a balance between detection sensitivity and false anomaly suppression, and consistent with NBI rating transitions observed, the final operating threshold, θ, is selected to be 0.015. All bridge condition classifications reported in Section 3 use this calibrated threshold.

As an example of the sensitivity analysis, results for two representative bridges are shown in **Figure 49**. Based on the DOT reports, both bridges are classified as Fair (NBI rating: 5) condition. As shown in this figure, for Bridge 6357, the correct threshold (green dashed line) to fall within Fair condition is 0.015, whereas this value for Bridge 6358 was 0.012. Although both bridges have the same condition, their sensitivity trends exhibit distinguishable differences. As shown in **Figure 49**, Bridge 6357 maintained a relatively high anomalous area (≈38%) and normalized severity ($\approx 5.4\times10^{-3}$) at the threshold of 0.015, transitioning from Poor (NBI = 4) to Fair (NBI = 5). In contrast, Bridge 6358 transitioned to Fair earlier, around 0.012, with anomalous percentage and severity dropping more sharply, reaching ≈36% and $\approx 3.6\times10^{-3}$ at the same threshold (0.015). This behavior suggests that Bridge 6357 has more localized or intense anomalies that quickly fall below threshold detection as the value increases, whereas Bridge 6358 exhibits more widespread but shallower anomalies that persist over a broader threshold range.

Such analysis supports two key insights:

1. Consistency of NBI estimation: Both bridges converge to the same condition rating despite their different sensitivity slopes, supporting the robustness of the selected threshold.
2. Potential for prioritization: Bridges that transition later (e.g., after 0.015) may warrant earlier inspection or maintenance, even within the same NBI rating, based on the persistence of high-severity patches.

This type of sensitivity-driven comparative profiling can thus be used to differentiate bridges within the same rating class and potentially inform more nuanced maintenance prioritization strategies.

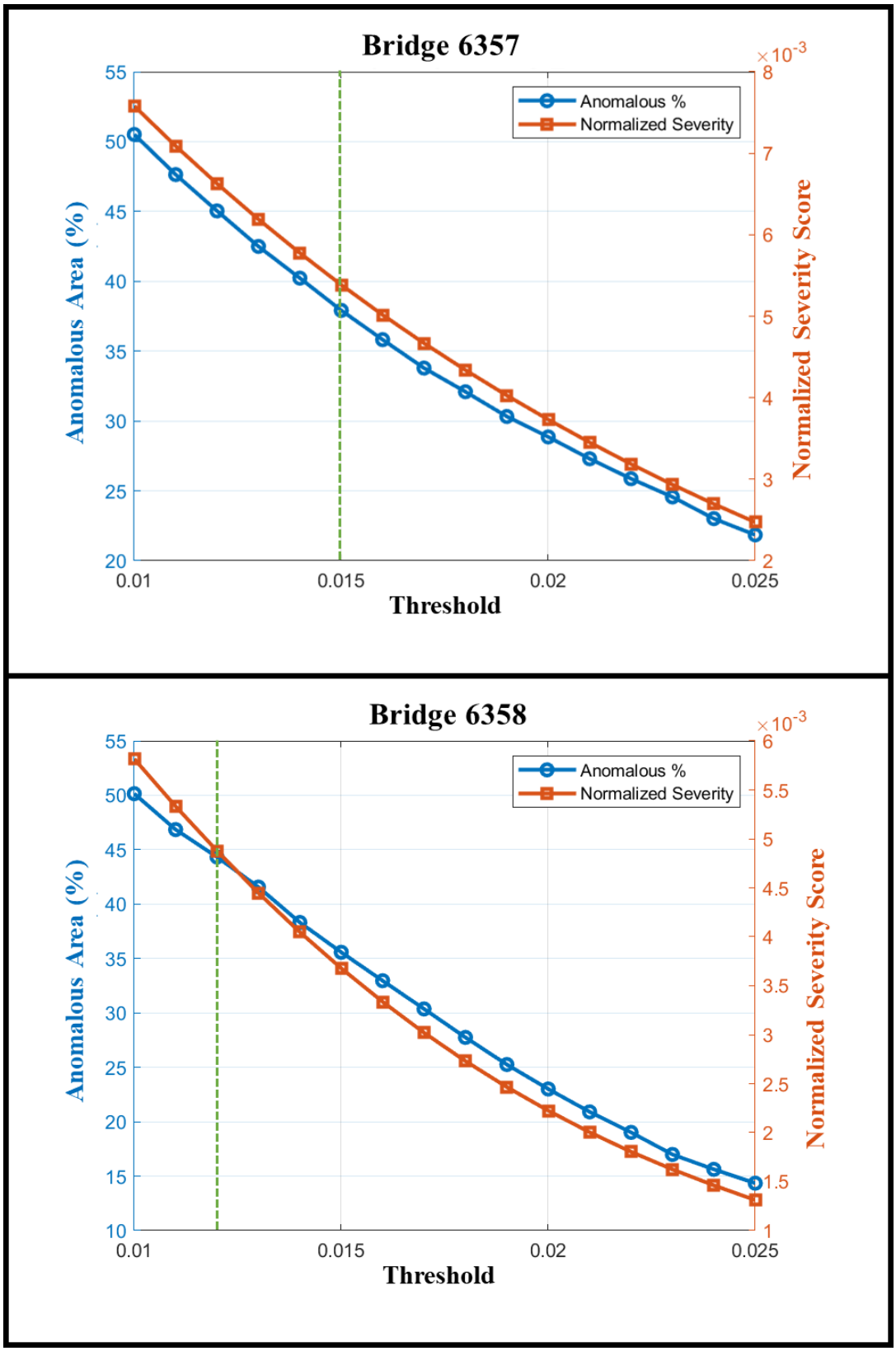


Figure 49. Examples of threshold sensitivity analysis for two bridges

8.2.2 Experimental Validation

To evaluate the performance of the proposed methodology in real-world scenarios, 7 bridges have been selected across New Mexico to predict their deck condition and NBI rating. Images from different sides of the bridges' decks are collected by a UAV to be fed to the model for predictions. The model shows full agreement with the actual NBI ratings (DOT reports) in 6 out of 7 cases, achieving an 85.7% accuracy in predicting correct deck conditions. This includes precise classification of bridges in Fair, Satisfactory, and Poor condition categories. **Table 23** summarizes the computed metrics for these bridges by considering the above-mentioned 0.015 threshold, and **Table 24** compares the predictions with the real conditions and ratings.

Bridge Number	Average Reconstruction Error	Anomalous Area (%)	Severity Score
1791	0.0102	21.60	0.0035
5838	0.0051	4.20	0.0003
5839	0.0094	16.00	0.0021
5840	0.0148	37.90	0.0054
5841	0.0169	45.20	0.0069
6455	0.0132	35.60	0.0036
6456	0.0273	53	0.0073

Table 23. Computed metrics for the test bridges' decks

Bridge Number	Predicted Condition	Real Condition	Predicted NBI Rating	Real NBI Rating
1791	Fair	Fair	5	5
5838	Satisfactory	Satisfactory	6	6
5839	Satisfactory	Satisfactory	6	6
5840	Fair	Fair	5	5
5841	Poor	Poor	4	4
6455	Fair	Fair	5	5

6456	Poor	Fair	4	5

Table 24. Comparison of the predicted vs. real bridge deck conditions and NBI ratings

As shown in **Table 23**, bridges with lower NBI ratings (e.g., Bridge_5841 and Bridge_6456) consistently exhibit higher average reconstruction errors, more widespread anomalous regions, and greater severity scores. For instance, Bridge 5841, which is rated as Poor (NBI = 4), shows an anomalous area of 45.2% and a high severity score. In contrast, Bridge 5338, rated as Satisfactory (NBI = 6), demonstrates only 4.2% anomalous area and a minimal severity score of 0.0003, validating the model's sensitivity to true damage signatures. Also, in **Table 24**, the only mismatch occurred with Bridge 6456, which was predicted as Poor (NBI = 4) while being rated Fair (NBI = 5) in the official record. However, this bridge exhibited both the highest reconstruction error (0.0273) and the largest anomalous area (53%), suggesting that the prediction may reflect emerging deterioration not yet documented in the manual inspection. This discrepancy highlights the proactive potential of the proposed method in detecting structural degradation earlier than conventional inspections, making it especially valuable for large-scale monitoring and prioritization.

For practical applications, it is crucial to provide a visualization of the deck's anomalies, as inspectors and decision-makers require spatially coherent visualizations of potential defects, in addition to condition ratings. To generate image-level anomaly maps, each test image was first divided into non-overlapping 64 × 64 grayscale patches. For each patch, the trained sparse autoencoder reconstructed the original content, and the MSE between the input and output vectors was computed as the patch anomaly score. These scalar errors were then spatially reassembled to form a 2D anomaly heatmap, preserving the patch-wise location of each score. Also, since the images and patches are independent, parallel computing is used to reduce the computation time. It is worth noting that since the model is light and parallelizable, it makes the proposed system suitable for deployment on the drone for real-time processing.

Figure 50 shows examples of UAV-captured bridge deck images from underneath and the side of the deck of the bridge 5839, and compares them with the predicted overlay of the anomaly heatmap based on the proposed method of this study. Areas shown in yellow and white indicate high reconstruction error and thus likely surface anomalies such as cracks and spallings, while dark red and black areas represent well-reconstructed regions typical of healthy concrete. The top panel showcases an image captured from the side of the bridge deck, where prominent horizontal cracking is visually evident. The predicted heatmap highlights these regions with light red and yellow hues, correlating with the actual damage zones. The model accurately localizes distributed anomalies, including minor spalls and crack propagation zones that are typically hard to detect using classical threshold-based image processing. The bottom panel displays an under-deck view, where multiple intersecting cracks are present. The reconstruction-error-based heatmap successfully intensifies around both longitudinal and transverse cracks, reinforcing the model's robustness across varying crack morphologies, orientations, and lighting conditions. The

localization precision is notable even in heavily textured regions where manual inspection becomes challenging.

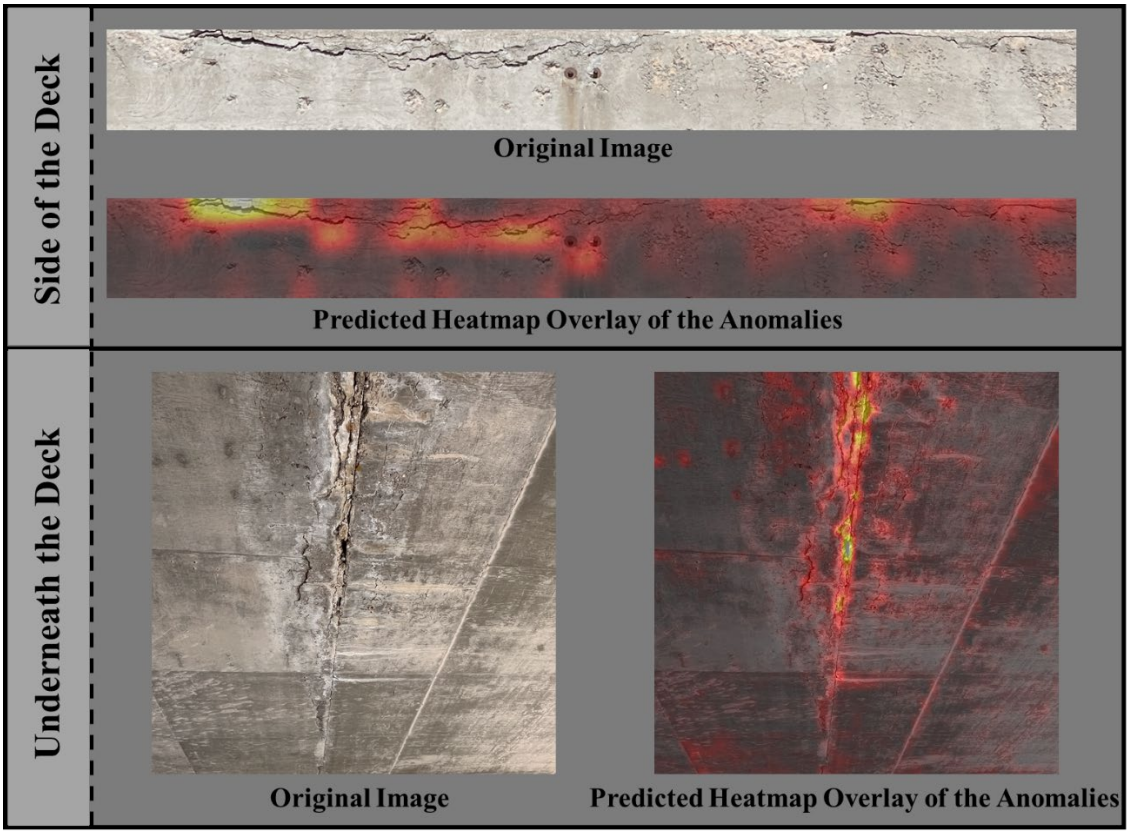


Figure 50. Sample images and predicted heatmap overlays from bridge 5839 deck

These visual results emphasize the model's utility not only for automated assessment and scoring but also for human-in-the-loop decision-making, where inspectors can use the heatmaps to focus their attention on the most critical structural regions. Combined with the metric reports and NBI-based rating mentioned before, the proposed system is not only able to predict the anomalies and condition rating, but also can be used for deeper analysis by inspectors to prioritize the bridges' maintenance based on the detailed predictions and visualizations.

TASK 9: Bridge Element Identification

9.1 Dataset Preparation

The dataset utilized in this study comprises 144 high-resolution images of bridges, each with dimensions of 3040×4056 pixels. These images were collected using a DJI Mavic 2 Enterprise to capture detailed visual data of bridge surfaces from a top-down perspective. The primary objective of this dataset was to support the development of a semantic segmentation model capable of distinguishing bridge elements from each other and from their surrounding background environments.

Each image in the dataset was manually annotated to create a pixel-wise ground truth mask, where the six classes of interest were deck, girder, cross-girder, pier, pier-cap, and background.

The annotation process was performed using MATLAB's Image Labeler application [19]. Annotators manually outlined the element boundaries, and the label masks were exported as categorical images. To ensure efficient learning while preserving the aspect ratio of the original images, all input images and masks were uniformly resized to a resolution of 768×1024 pixels. This resizing was done using bicubic interpolation for RGB images and nearest-neighbor interpolation for binary masks to avoid label smearing.

The entire dataset was randomly partitioned into three subsets:

- Training set: 110 images (~76%)
- Validation set: 22 images (~15.0%)
- Test set: 12 images (~9%)

This split was designed to provide sufficient samples for model optimization while ensuring reliable validation and testing. It is important to ensure that the test set contains diverse examples with varying lighting conditions, surface textures, and obstructions such as debris or markings, to robustly assess the model's generalization capability. **Figure 51** shows some examples of images of the labeling process, and **Figure 52** illustrates the dataset structure. This curated dataset served as the foundation for training and evaluating the DeepLab v3+ semantic segmentation model described in the subsequent sections.

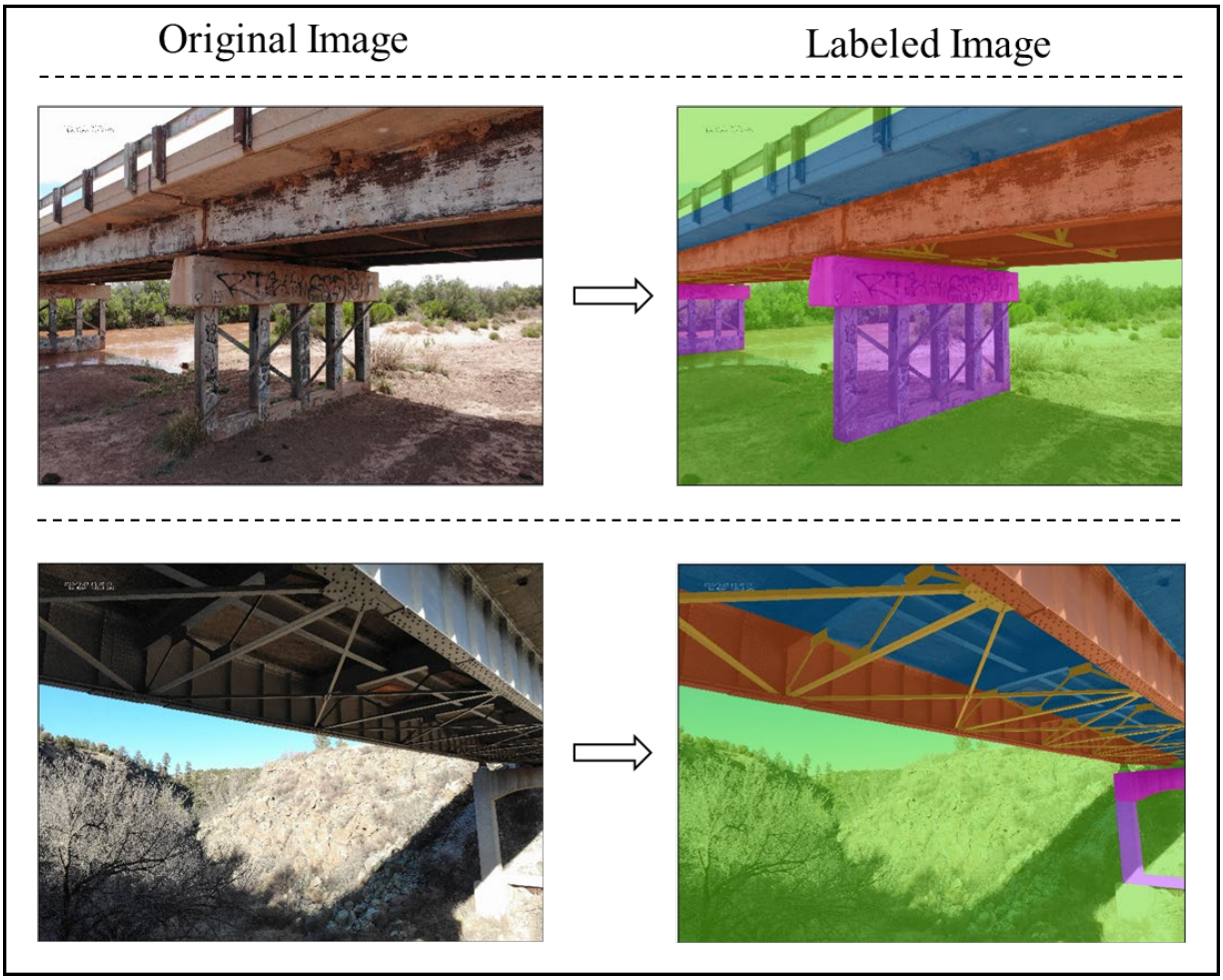


Figure 51. Examples of element identification image labeling

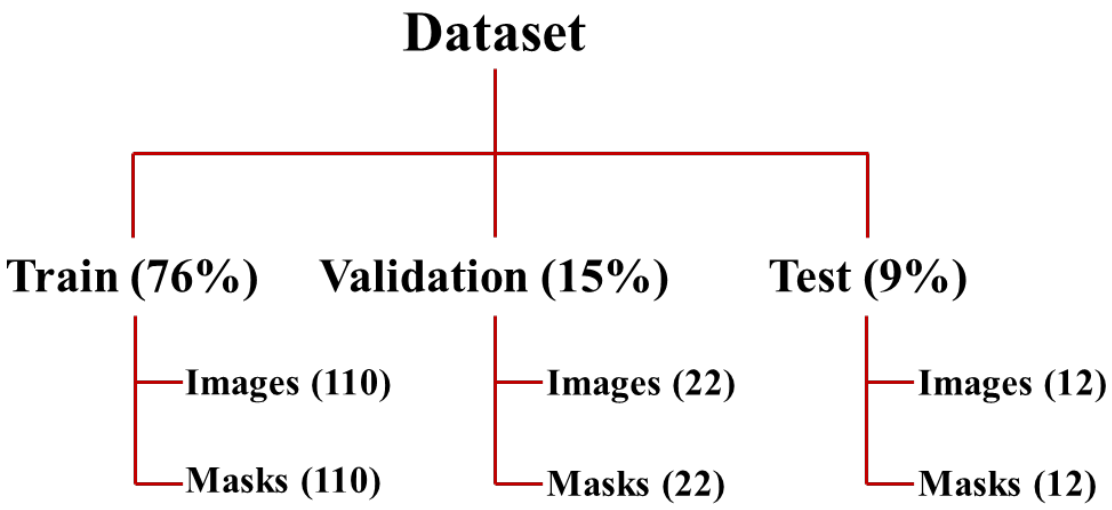


Figure 52. Dataset structure for element identification model training

9.2 Data Augmentation

To enhance the generalization capability of the semantic segmentation model and mitigate overfitting due to limited data, a suite of data augmentation techniques was applied during training. The original dataset consisted of only 110 training images, which is relatively small for training a deep convolutional neural network. Therefore, online augmentation was used to artificially increase the variability of the training data.

All augmentations were performed on-the-fly using MATLAB's "imageDataAugmenter" class, ensuring that each training epoch introduced slight variations in image appearance and geometry. The goal was to simulate realistic conditions that the model might encounter during actual UAV-based inspections, such as different flight angles, lighting conditions, and bridge deck appearances.

The following transformations were applied randomly and independently to each image-mask pair during training:

Transformation	Range / Parameters	Purpose
Random Reflection	50% probability	Simulates bidirectional UAV flight over the same bridge deck
Random Rotation	±30 degrees	Simulates variations in UAV camera orientation during capture
Random Scaling	80%–120% (X and Y independently)	Mimics changes in UAV altitude and image zoom

Table 25. Applied augmentations to the training dataset

These augmentations were applied only to the training set, ensuring that validation and test results remained unbiased and representative of real-world performance. Data augmentation is especially critical in pixel-wise segmentation tasks because it not only increases the diversity of image appearance but also expands the spatial configurations that the network is exposed to. For example, random reflection helps the model learn symmetry in deck structures, while rotation and scaling aid in learning translation-invariant features.

9.3 Model Architecture

The model architecture employed for the semantic segmentation of bridge elements is based on DeepLab v3+, a state-of-the-art convolutional neural network known for its accuracy and efficiency in pixel-wise classification tasks [43]. DeepLab v3+ combines spatial pyramid pooling, atrous convolutions, and a decoder module to produce high-resolution segmentation outputs while maintaining deep contextual understanding. The ASPP module allows the network to perceive objects at multiple receptive field sizes, which is particularly useful for bridge decks that may appear at various scales due to changes in UAV altitude or perspective.

In this study, DeepLab v3+ was implemented using MATLAB's Deep Learning Toolbox, with the encoder initialized using a ResNet-50 backbone developed by Microsoft. ResNet-50 is a convolutional neural network that is 50 layers deep. We can load a pretrained version of the

network trained on more than a million images from the ImageNet database [44], [45]. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. This choice strikes a balance between model complexity and computational efficiency, making it suitable for deployment on resource-constrained platforms such as UAVs.

Key architectural components of the model are briefly described in **Table 26**.

Component	Description
Backbone	ResNet-50 (50-layer residual network) pretrained on ImageNet
ASPP Module	Atrous Spatial Pyramid Pooling with multiple dilation rates for context
Decoder Module	Refines segmentation results by combining low- and high-level features
Upsampling Strategy	Bilinear upsampling to restore final output to full spatial resolution
Output Layer	Softmax classification layer for binary segmentation (deck vs. background)

Table 26. Key architectural components of the DeepLab v3+ model

The choice of input size [768×1024] preserved the aspect ratio of the original 3040×4056 images and allowed for efficient GPU training. The number of output classes was set to 6, corresponding. The model was trained using a single GPU, and the use of ResNet-50 ensured that training and inference were computationally tractable. This configuration also supports potential onboard deployment on embedded GPUs (e.g., NVIDIA Jetson platforms), aligning with real-time UAV inspection applications. The overall architecture follows a typical encoder-ASPP-decoder structure, enabling robust segmentation of bridge decks even in small datasets.

9.4 Training Setup

The core training configurations used to optimize the model performance, including network initialization, hyperparameter tuning, learning rate strategy, and training execution environment, are empirically tuned and listed below in **Table 27**.

Parameter	Value
Optimizer	Adam
Initial Learning Rate	1e-4
L2 Regularization Factor	0.0001
Mini-Batch Size	4
Max Epochs	50

Iterations per Epoch	27
Max Iterations	1350
Shuffle	Every Epoch
Validation Frequency	Every 50 Iterations
Verbose Output	Enabled
Execution Environment	Single GPU
Data Normalization	Automatic per-channel

Table 27. Training Hyperparameters for DeepLab v3+

The learning rate was fixed throughout training, and no custom learning rate schedule or early stopping was applied due to consistent improvements in training and validation accuracy.

The training was performed on a single GPU using MATLAB's trainNetwork function. Automatic environment detection ensured efficient use of GPU memory and enabled real-time display of training metrics. The model reached convergence in under two minutes of training time.

A live training plot, shown in **Figure 53**, displayed metrics including:

- Training accuracy and loss
- Validation accuracy and loss
- Iteration-wise updates per mini-batch

The final model achieved high training and validation performance, suggesting the selected setup was effective for the binary segmentation task with a relatively small dataset.

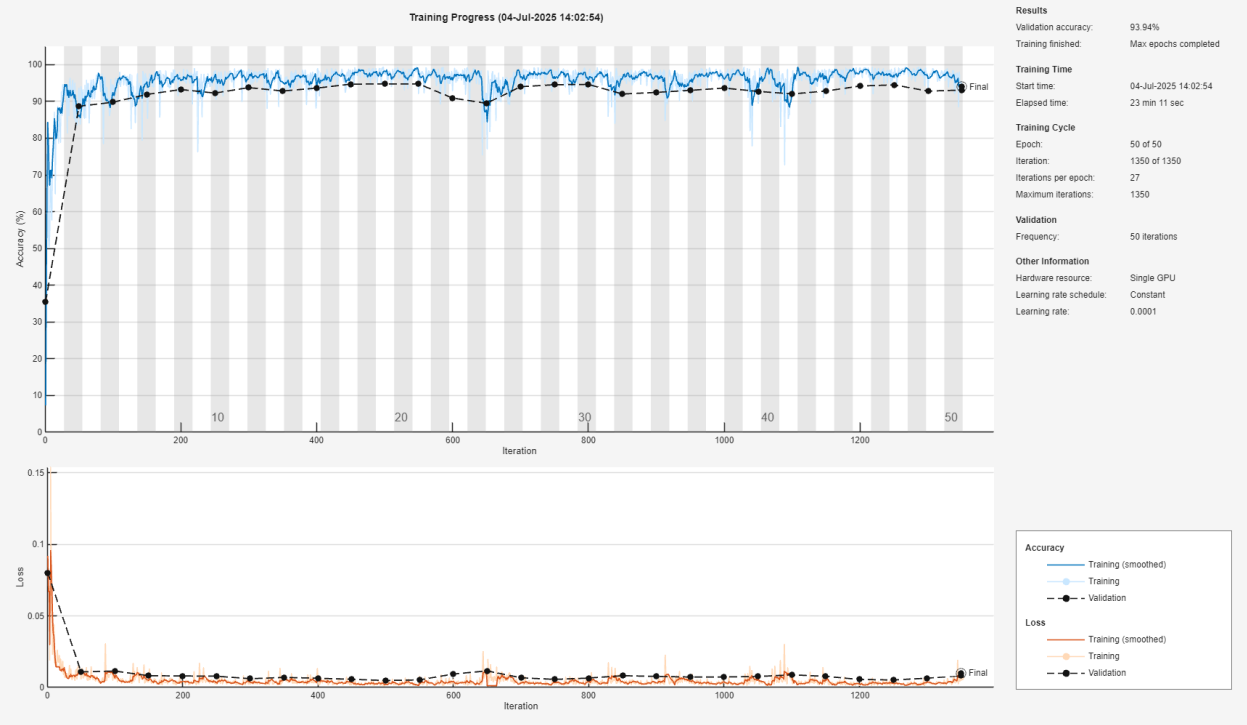


Figure 53. Training progress of DeepLab v3+ for element identification

At the end of the model training process, which took 23 minutes, the validation accuracy was 93.94%, indicating strong generalization.

9.5 Results and Evaluation

To evaluate the performance of the model, the following metrics have been used:

- Global Accuracy: Ratio of correctly classified pixels to the total number of pixels.
- Mean Accuracy: Average of per-class pixel accuracies.
- Mean IoU (Intersection over Union): Average IoU across classes.
- Weighted IoU: IoU weighted by pixel count per class.
- Mean Boundary F1 Score (BF Score): Boundary matching score assessing segmentation contour alignment.

The model was evaluated on a test set comprising 12 high-resolution images that were not seen during training or validation. The evaluation metrics obtained are summarized in **Table 28**.

Metric	Score
Global Accuracy	95.06%
Mean Accuracy	88.42%

Mean IoU	77.39%
Weighted IoU	90.84%
Mean BF Score	85.13%

Table 28. Metrics for the overall performance of the DeepLab v3+ model

These results are interpreted below:

- Global Accuracy (95.06%): Indicates that nearly all pixels were correctly classified across the entire test set. This is a strong indicator of generalization to unseen data.
- Mean Accuracy (88.42%): Reflects high consistency in performance across five element and background classes, without bias.
- Mean IoU (77.39%): IoU is one of the most rigorous segmentation metrics, requiring a high overlap between predicted and actual areas. A score over 77% confirms precise shape and region prediction. But it is worth noting that it is improvable by refining the training and adding more images with cross girders.
- Weighted IoU (90.84%): Higher than Mean IoU due to the dominance of background pixels, showing that the model performs well even when accounting for class imbalance.
- Mean Boundary F1 Score (85.13%): A strong BF score indicates that object boundaries (e.g., deck edges) are sharply predicted.

This level of segmentation performance confirms that the DeepLab v3+ model is highly effective at identifying bridge elements.

To further analyze the segmentation performance of the trained model, we examine per-class metrics that offer insight into how well each class (deck, girder, background, etc.) was identified across the test set. **Table 29** presents the results for accuracy, IoU, and boundary precision (BF Score) for each semantic class:

Class	Pixel Accuracy	Intersection over Union (IoU)	Mean Boundary F1 Score (BF Score)
Background	96%	93%	88%
Deck	94%	90%	80%
Girder	98%	90%	90%
Cross Girder	70%	42%	67%
Pier	84%	70%	78%
Pier Cap	88%	78%	79%

Table 29. Class-wise evaluation results

Despite different elements often featuring variable textures, lighting changes, or surface defects, the model maintained exceptional accuracy and shape alignment. The deck and girder classes achieved a remarkably high accuracy of 94% and 98%, meaning nearly all their pixels were correctly identified.

Finally, **Figure 54** presents sample visual comparisons of the model's predictions against the ground truth. These visualizations affirm that the model preserves the structural integrity of element boundaries even under varying textures and lighting conditions.

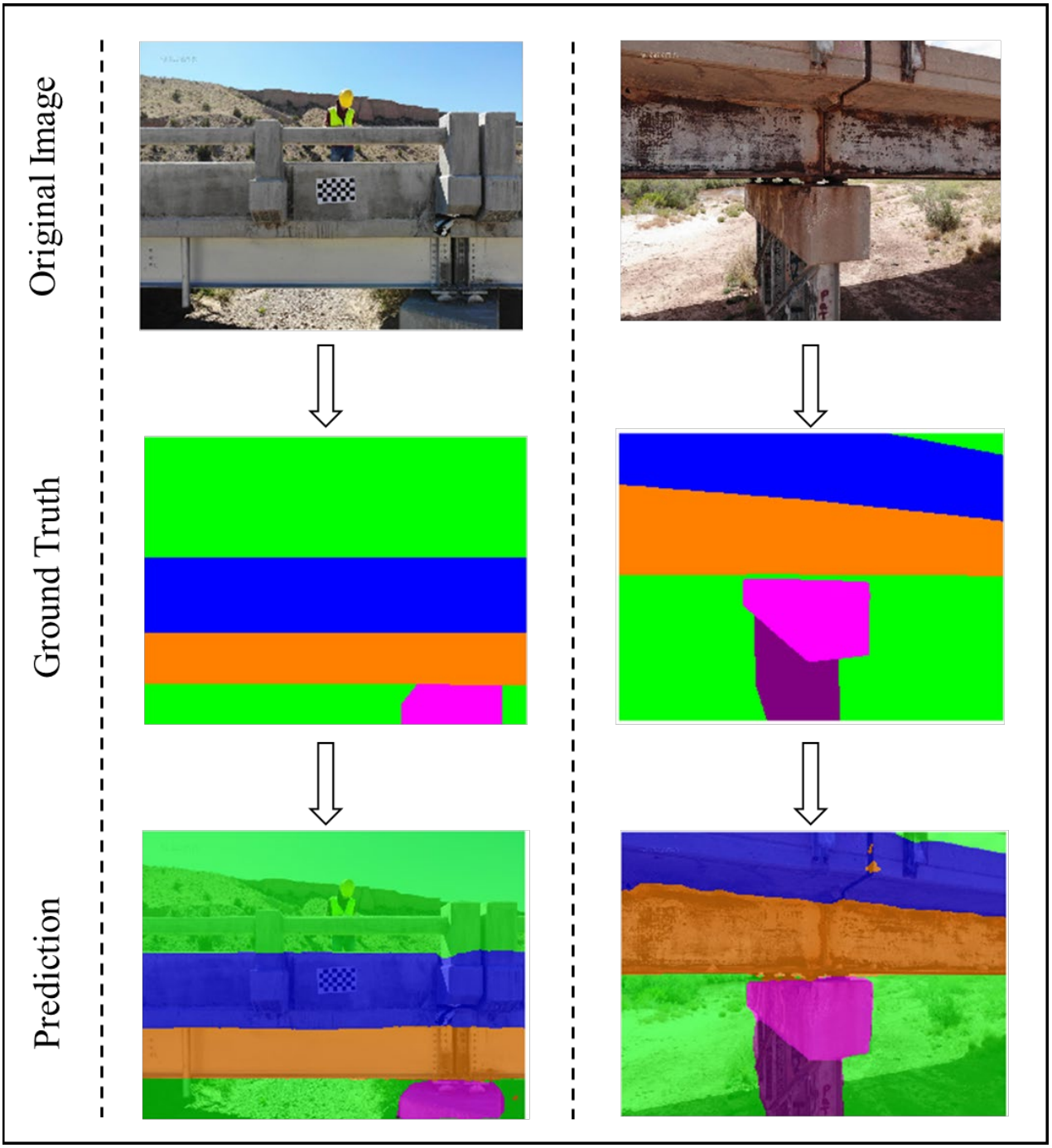


Figure 54. Samples of the DeepLab v3+ model's predictions

TASK 10: Spalling Damage Segmentation in Concrete Infrastructure

Reinforced concrete (RC) structures are widely used in infrastructure, including buildings, bridges, and dams, due to its exceptional strength, durability, flexibility, and stability. However, structural and material defects in RC structures, such as concrete spalling, exposed steel bars, and steel bar corrosion, are inevitably caused by natural disasters, environmental changes, and extended periods of use. The timely and accurate detection of concrete spalling and exposed steel bars after concrete spalling in reinforced concrete structures has become a task of far-reaching significance. In RC structures, a prevalent structural ailment is the delamination of the concrete protective layer, often triggered by factors such as material aging and environmental corrosion. In fact, the issue of concrete cover spalling can lead to prolonged exposure of the steel bars to the air. The strength of steel bars is significantly reduced by corrosion caused by oxygen, water, and other airborne agents, posing a significant threat to the integrity of the structure. Structural health detection is dedicated to ensuring that buildings and infrastructure can be used safely and reliably within their designed lifespan [55], [56].

A meta-optimized deep learning framework was developed for pixel-level segmentation of concrete spalling in inspection images. The overall workflow was designed to address two major challenges that commonly degrade segmentation performance in infrastructure damage detection: (i) severe foreground–background imbalance, because spalling typically occupies a relatively small portion of the image area; and (ii) heterogeneous failure modes, because spalling regions may appear as thin boundaries, small isolated defects, or visually ambiguous patterns that resemble shadows or textured background. To address these issues, the proposed framework was composed of four tightly coupled components: (1) structured image and mask preparation; (2) generation of a patch pool with explicit patch-type categorization; (3) meta-heuristic optimization of the training curriculum and loss formulation; and (4) final U-Net training, threshold calibration, and quantitative and qualitative evaluation. The workflow of the proposed framework is illustrated in **Figure 55**.

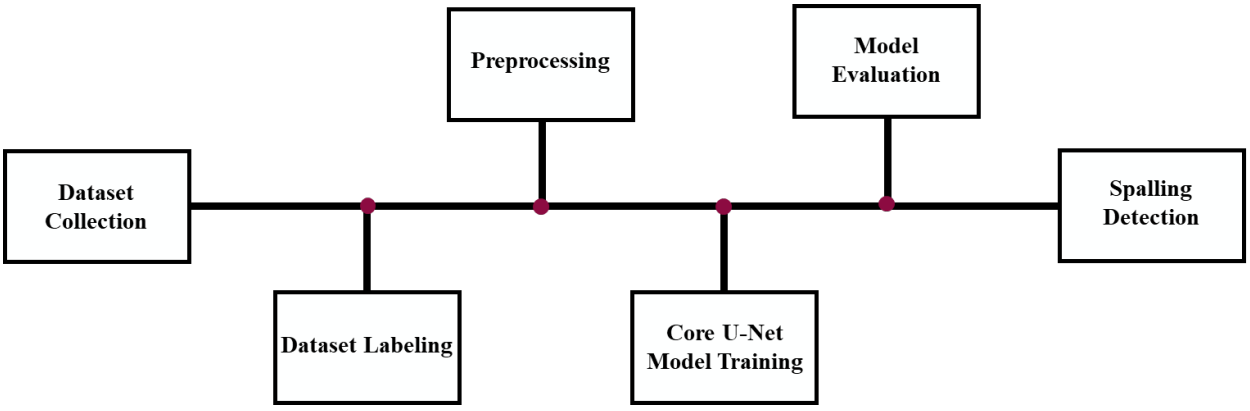


Figure 55. Workflow of the proposed framework for spalling segmentation

In the proposed framework, image patches were not treated as uniformly informative training samples. Instead, candidate patches were first categorized into interpretable groups representing common segmentation difficulties, including edge-dominant spalling, tiny spalling regions, textured negative regions, and shadow-like negative regions. A particle swarm optimization (PSO) strategy was then employed to search for an optimal sampling distribution over these patch categories while simultaneously optimizing the weighting between binary cross-entropy and Dice

losses. The objective of this search was not merely to maximize validation overlap, but to identify a training policy that yielded strong validation performance while maintaining high recall and reducing overfitting. After the meta-optimization stage, the best sampling policy and loss-mixing parameter were fixed and used for full training of the segmentation network. Finally, validation-based threshold selection, test-set evaluation, and full-image qualitative overlays were performed.

10.1 Dataset Organization and Data Representation

The dataset used in this study consists of 1040 RGB concrete surface images (1024 pixels by 1024 pixels) and their corresponding binary spalling masks. The pixel-wise labeling of the dataset was performed using MATLAB's ImageLabeler toolbox [1]. Each image was paired with a single-channel mask in which spalling pixels were represented by nonzero values and background pixels were represented by zero values. To ensure a fully supervised segmentation setting, each mask was converted to a binary representation according to:

$$Y_{(i,j)} = \begin{cases} 1, & if\ M_{(i,j)} > 0 \\ 0, & otherwise \end{cases} \quad (1)$$

where $M_{(i,j)}$ denotes the grayscale mask value at pixel (i, j), and $Y_{(i,j)}$ denotes the binary ground-truth label. Therefore, $Y_{(i,j)}$=1 corresponds to spalling, and $Y_{(i,j)}$=0 corresponds to background. **Figure 56** shows several examples of the labeling process for some images and their corresponding binary masks.

The complete dataset was divided into three non-overlapping subsets: training, validation, and test sets. The training set was used to optimize model parameters, the validation set was used for model selection and threshold calibration, and the test set was reserved exclusively for final performance evaluation. In the implementation used in this study, 728 image-mask pairs were assigned to the training set, 156 pairs to the validation set, and 156 pairs to the test set. This partitioning ensured that final performance metrics were computed on previously unseen images.

Because the final objective was dense semantic segmentation, each image was represented as an RGB tensor $I \in \mathbb{R}^{H\times W\times 3}$, and each corresponding mask was represented as a binary tensor $Y \in \{0,1\}^{H\times W}$. During model training, images were normalized to the range [0,1], and masks were converted to floating-point binary arrays to match the expected input of the loss functions.

Figure 56. Examples of original images and their corresponding labeled images for spalling segmentation

Direct training on full-resolution inspection images often leads to inefficient learning in damage segmentation tasks because the proportion of spalling pixels is usually small relative to the background area. Under such conditions, a network can achieve deceptively high pixel accuracy by over-predicting the dominant background class while failing to segment spalling regions accurately. In addition, full-image training may under-expose the model to localized damage details, especially when defects are spatially small or visually subtle.

To mitigate these limitations, a patch-based training strategy was adopted. Instead of training exclusively on full images, training patches of fixed size 512×512 were extracted from the original images.

Patch-based training was adopted for three reasons. First, it increased the effective number of training samples because multiple patches could be extracted from each image. Second, it allowed the model to focus on local spalling morphology and boundary characteristics that may be diluted in a full-image formulation. Third, it enabled explicit control over the sampling of difficult patch types, which was essential for the proposed curriculum optimization.

However, patch-based learning was not implemented using simple random cropping alone. Instead, a candidate patch pool was first created and then sampled according to either baseline or optimized policies. This distinction is important because the key novelty of the proposed framework lies not in patch extraction by itself, but in the optimization of which patches should be emphasized during training.

A candidate patch pool was built separately for the training, validation, and test sets. For each image in a given split, a predefined number of random candidate patches was extracted. Specifically, 35 candidate patches per image were used for the training set, whereas 12 candidate patches per image were used for the validation and test sets. These values were selected to provide sufficient diversity in the training pool while maintaining a manageable computational cost for the validation and test pools. Each candidate patch was stored as a structured entry containing the image index, patch location, patch type, and spalling fraction.

Very weak positive patches, i.e., patches containing extremely small amounts of spalling, may contribute disproportionate noise during optimization if retained indiscriminately. Therefore, for the training patch pool, a minimum positive fraction threshold was defined as $\tau_{min} = 0.001$. For any patch with $0 < \rho < \tau_{\min}$, the patch was excluded from the training pool. This filtering retained genuinely informative positive patches while removing extremely sparse positive samples that could destabilize patch categorization and training.

10.2 Patch-Type Categorization and Failure-Mode Modeling

A central idea of the proposed framework was that not all patches contribute equally to segmentation learning. In infrastructure damage detection, several failure modes are repeatedly encountered: some patches contain thin damage boundaries, some contain very small defects, and others contain visually misleading but non-damaged regions such as rough texture or shadows. If these cases are sampled uniformly, the network may not allocate sufficient learning capacity to the most difficult and practically relevant patterns.

To address this, each candidate patch was assigned to one of several interpretable categories:

1. EDGE_POS: positive patch with significant spalling boundary content,
2. TINY_POS: positive patch with a very small damaged area,
3. NEG_TEXTURE: negative patch with strong texture likely to induce false positives,
4. NEG_SHADOW: negative patch with dark intensity likely to resemble damage,
5. OTHER: patches not captured by the above categories.

This patch taxonomy served as the basis for curriculum optimization.

A patch-pool-based dataset class was constructed so that training patches could be sampled according to a specified probability distribution over patch types. During one data retrieval step, a patch type t was first sampled, and then a candidate patch belonging to that type was selected uniformly from the corresponding subset of the patch pool. In this way, the network did not sample image locations directly, but rather sampled from a structured pool through a category-aware policy.

For the validation and test patch datasets, no data augmentation was applied, and uniform patch-type sampling was used to ensure consistency in evaluation. For the training patch dataset, augmentation was applied and the patch-type probabilities were either fixed (for baseline experiments) or optimized (for the proposed method). The training patch dataset size was defined using a target number of samples per epoch rather than by enumerating every patch in the pool. Specifically, 8000 patch samples per training epoch were used, whereas 1200 validation patch samples per epoch were used. This approach provided a stable and reproducible training schedule while maintaining flexibility in how patches were sampled.

Data augmentation was employed during training to improve generalization and reduce sensitivity to geometric orientation and illumination variation. Augmentation was applied only to training data and not to validation or test data. The following transformations were used:

- horizontal flip with probability 0.5,
- vertical flip with probability 0.5,
- random 90-degree rotation with probability 0.5,
- shift-scale-rotate transformation with translation limit 0.05, scale limit 0.15, rotation limit 20°, and probability 0.7,
- random brightness and contrast adjustment with probability 0.5.

These augmentations were selected because they preserve the structural identity of concrete spalling while increasing appearance diversity. In particular, geometric transformations help the network become less sensitive to acquisition angle, while brightness/contrast transformations improve robustness to changing illumination.

10.3 Segmentation Network Architecture

A U-Net architecture was adopted as the base segmentation model. U-Net was selected because of its strong performance in binary semantic segmentation and its widespread use as a reference model in infrastructure damage detection. The use of a well-established architecture was intentional so that improvements could be attributed to the proposed training strategy rather than to a more complex network design.

U-Net is a deep learning model designed for image segmentation, where the goal is to classify each pixel in an image. It was originally developed for biomedical image segmentation but has since been widely used in various fields, including remote sensing, medical imaging, and structural damage detection [50]. U-Net follows a fully convolutional neural network (FCNN) architecture, meaning it does not use fully connected layers like traditional CNNs. Instead, it is structured in a U-shaped design, consisting of two main parts [8]:

1. Encoder (Contracting Path): Extracts important features from the input image by progressively reducing its size through convolutional and pooling layers.

2. Decoder (Expanding Path): Reconstructs the image by gradually increasing its resolution and using skip connections to retain fine details.

Figure 57 shows the layers of the U-Net model used in this study for 512 pixels by 512 pixels images for binary spalling segmentation.

Figure 57. U-Net architecture of this study for spalling detection

After completion of the meta-optimization stage, the best sampling distribution and best α were fixed, and a final U-Net model was trained from scratch. The training process was performed using the full training patch dataset and the same augmentation strategy described previously. The optimizer used was AdamW with a learning rate of 10^{-3}. Training was performed for 35 epochs. At the end of each epoch, performance was evaluated on the fixed validation patch loader, and the model with the highest validation IoU was saved. This model selection strategy was adopted to avoid choosing a model based on test performance.

To maintain a consistent evaluation format with the training and validation protocol, test performance was first computed using a patch-based global pixel-wise evaluation. A test patch pool was constructed in the same manner as the validation patch pool, and a fixed number of test patch samples per epoch was used. For the selected threshold, the following metrics were computed:

- Intersection over Union (IoU): $\frac{TP}{TP + FP + FN}$
- Dice coefficient: $\frac{2TP}{2TP + FP + FN}$
- pixel accuracy: $\frac{TP + TN}{TP + TN + FP + FN}$

- precision: $\frac{TP}{TP + FP}$
- recall: $\frac{TP}{TP + FN}$
- F1-score: $\frac{2\,.\,Precision\,.\,Recall}{Precision + Recall}$

Where TP is true positives, TN is true negatives, FP is false positives, and FN is false negatives.

10.4 Training Results

Before training the model, as mentioned above, PSO has been used to optimize the training curriculum and the loss-mixing coefficient. **Figure 58** presents the convergence behavior of the PSO procedure used to optimize. The vertical axis represents the global best fitness, where lower values indicate better solutions, and the horizontal axis represents the PSO iteration number. The best fitness improves from approximately −0.515 at the first iteration to about −0.528 by the fourth iteration, after which the curve remains unchanged through iteration 10.

The figure indicates that the meta-optimization process converged rapidly and stably. A substantial improvement is observed during the first few iterations, showing that the swarm was able to quickly identify a more favorable region of the search space. This early improvement suggests that the optimization problem was sufficiently structured for PSO to exploit informative gradients in the fitness landscape, even though the objective itself was evaluated through short-burst neural network training rather than through an explicit analytical function.

After iteration 4, the global best fitness remains nearly constant. This plateau indicates that the swarm had reached a stable optimum, or at least a near-optimal region, and that additional iterations did not produce further meaningful gains. From a methodological standpoint, this behavior is important because it shows that the proposed meta-search did not oscillate excessively and did not exhibit unstable or erratic optimization dynamics. In other words, the search space defined by the curriculum weights and the loss-mixture coefficient appears to be well posed and sufficiently smooth for PSO.

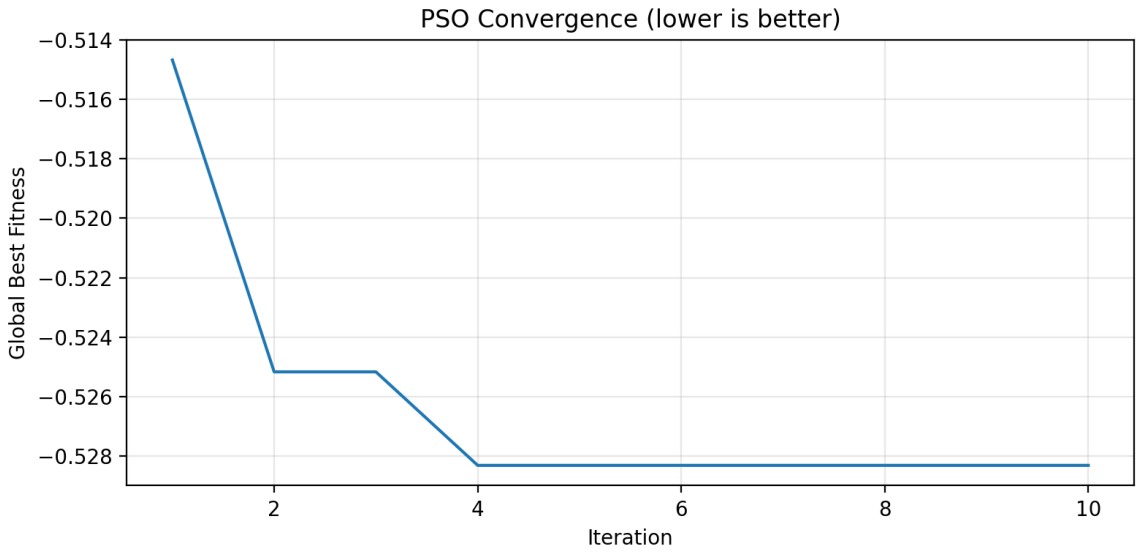


Figure 58. PSO convergence behavior

For the training progress, **Figure 59** presents the learning dynamics of the proposed framework over 35 training epochs. The training loss decreases monotonically, while the validation loss remains within a stable range after an initial decline, indicating effective optimization without severe overfitting. Training and validation IoU both improve substantially, with the validation IoU stabilizing at a relatively high level despite moderate epoch-to-epoch fluctuations that are expected in patch-based damage segmentation. Pixel accuracy remains high throughout training, reflecting robust background classification, whereas recall remains consistently strong on the validation set, confirming the ability of the proposed model to preserve damage sensitivity under inspection-oriented learning objectives. Overall, the curves suggest that the meta-optimized curriculum and loss formulation lead to stable convergence and balanced segmentation performance.

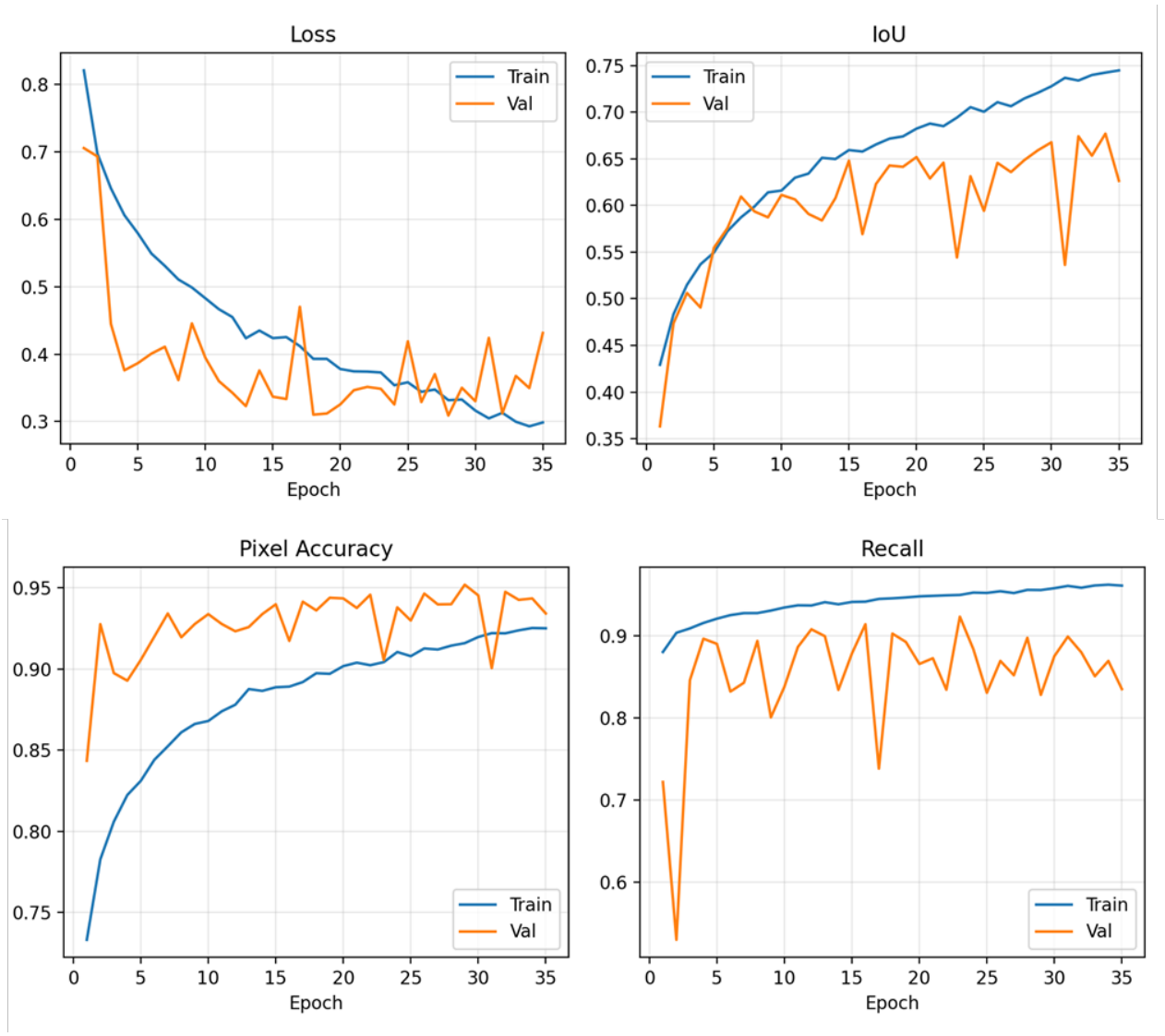


Figure 59. Results of the optimized model training progress

By the end of training, the proposed framework achieved strong performance on both the training and validation sets, indicating that the network learned a highly effective representation of spalling morphology while maintaining satisfactory generalization. At epoch 35, the training metrics reached a loss of 0.2987, IoU of 0.745, pixel accuracy of 0.9250, precision of 0.768, and recall of 0.961. These values show that the model fit the training data well and was particularly successful in identifying damaged pixels, as evidenced by the very high recall. This behavior is consistent with the recall-oriented design of the proposed framework, in which the curriculum optimization and loss formulation were intended to reduce false negatives and improve sensitivity to spalling regions.

On the validation set, the model achieved a loss of 0.4315, IoU of 0.626, pixel accuracy of 0.9342, precision of 0.715, and recall of 0.835 at the same epoch. Although a moderate gap is observed between the training and validation IoU and recall values, the validation performance remained strong and stable, suggesting that the model generalized well despite the difficulty of the

segmentation task and the heterogeneous nature of the validation patches. The validation pixel accuracy slightly exceeded the training accuracy, which is not unusual in highly imbalanced segmentation problems where background pixels dominate. Overall, these results, which are represented in **Table 30**, indicate that the proposed method produced a robust segmentation model with high overlap quality and strong damage sensitivity, while maintaining acceptable precision on previously unseen data.

Set/Metric	Loss	IoU	Pixel Accuracy	Precision	Recall
Training	0.2987	0.745	0.9250	0.768	0.961
Validation	0.4315	0.626	0.9342	0.715	0.835

Table 30. Training and validation performance at the end of the model training

10.5 Testing Results

To evaluate the performance of the proposed methodology and trained model, a separate test dataset, which is unseen to the model, has been analyzed to report the confusion matrix, performance metrics, and prediction overlays.

Figure 60 presents the pixel-level confusion matrix of the proposed method on the test set. A large number of true negatives (238,094,641) is observed, which is expected because background pixels dominate concrete surface images. More importantly, the number of true positives (43,712,448) is substantially larger than the number of false negatives (4,274,534) indicating that the model successfully identified the majority of spalling pixels. This behavior is consistent with the high recall achieved by the proposed framework and confirms that the model remained sensitive to damaged regions rather than collapsing toward conservative background prediction.

At the same time, the number of false positives (28,491,177) is higher than the number of false negatives, which reflects the operating preference of the proposed system. In safety-critical infrastructure inspection, over-detection is generally more acceptable than missed damage, since false positives can be reviewed during subsequent assessment, while false negatives may lead to undetected deterioration. Therefore, the confusion structure shown in **Figure 60** supports the intended design objective of the proposed framework: to prioritize reliable damage capture while maintaining reasonable overall segmentation precision. When interpreted together with the reported IoU, Dice, precision, and recall values, the confusion matrix confirms that the method achieved a practically meaningful balance between overlap quality and inspection-oriented sensitivity.

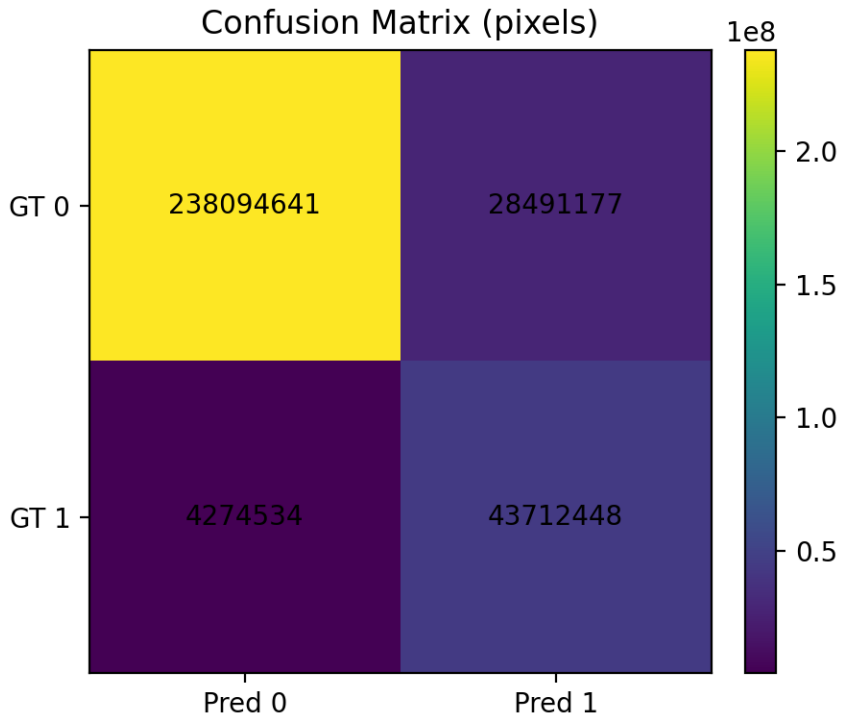


Figure 60. Pixel-level confusion matrix of the proposed method on the test set

On the test set, the proposed framework achieved an IoU of 0.5716, Dice coefficient of 0.7274, pixel accuracy of 0.8958, precision of 0.6054, recall of 0.9109, and F1-score of 0.7274. These results indicate that the model maintained strong generalization performance on previously unseen data and, most importantly, preserved a high level of sensitivity to spalling regions. The recall of 0.9109 shows that the vast majority of damaged pixels were successfully identified, which is particularly valuable in infrastructure inspection where missed damage is generally more critical than moderate over-segmentation. The Dice and F1 values, both equal to 0.7274, further confirm that the predicted masks retained a substantial level of agreement with the ground-truth damage regions.

At the same time, the precision value of 0.6054 indicates that the proposed method favored a recall-oriented operating point, producing some false positives in exchange for a relatively low number of false negatives. This behavior is consistent with the design philosophy of the proposed framework, which explicitly prioritized recall during meta-optimization and threshold selection in order to support safety-critical spalling detection. Although the pixel accuracy remained high at 0.8958, this metric should be interpreted together with IoU and recall because background pixels dominate the test images. Overall, the test results, summarized in **Table 31**, demonstrate that the proposed framework achieved a practically meaningful balance between segmentation overlap and defect sensitivity, making it suitable for automated damage screening applications in concrete infrastructure inspection.

Set/Metric	IoU	Dice	Pixel Accuracy	Precision	Recall	F1
Test	0.5716	0.7274	0.8958	0.6054	0.9109	0.7274

Table 31. Test-set performance of the proposed framework

Finally, in **Figure 61**, some of the predictions of the trained model based on the proposed framework are shown. The figure shows the original images and the prediction overlays, where yellow areas are true positives (correctly predicted spalling), green areas are false negatives (not correctly predicted spalling), red areas are false positives (not a spalling, but predicted as spalling), and the remaining areas with no color are true negatives (correctly predicted no spalling).

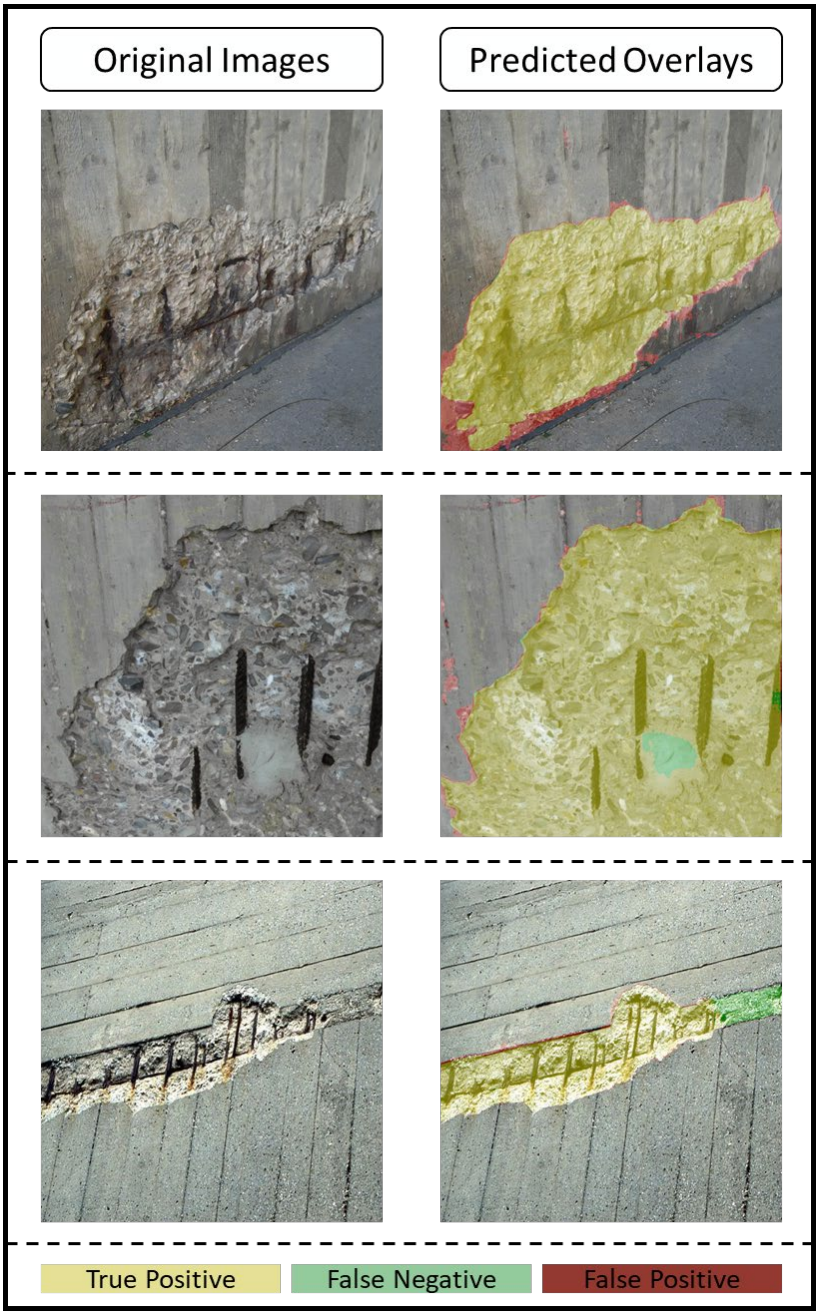


Figure 61. Prediction overlays of the trained model

TASK 11: Software Development for Crack Detection and Quantification

11.1 Model Adaptation

The purpose of this task is to develop a web-based application that supports automated crack detection and quantification for bridge inspections. The web-based application processes aerial imagery collected by drones, enabling bridge inspectors to identify, localize, and measure cracks on bridge deck surfaces in a rapid and efficient manner. The web application features an intuitive user interface, replicates the functionality demonstrated in Task 7 by the research team at New Mexico State University (NMSU), and delivers accurate, data-driven results generated from the deep learning model developed in Task 7. In addition, the web application integrates visualization and reporting tools to support decision-making, allowing inspectors to analyze, document, and manage bridge deck condition data quickly and efficiently.

The research team at NMSU provided a trained segmentation model (i.e., trained_unet.mat), the original MATLAB training script (i.e., main_model.m), and a dataset of training and testing drone images to the research team at the University of New Mexico (UNM). However, the model was stored in MATLAB format, which is not natively compatible with Python-based frameworks, the primary framework used for the web application under development.

Initial attempts at directly converting the .mat model into a Python-compatible format were unsuccessful. To address this, the development team at UNM analyzed the provided MATLAB script to understand the model architecture and training methodology. Leveraging the drone imagery dataset provided by NMSU, the training pipeline was then faithfully replicated in Python with TensorFlow, effectively recreating and modernizing the model in a framework suitable for deployment within a web-based environment. This ensured continuity with NMSU's methodology while enabling integration into a scalable, production-ready architecture.

11.2 Web Application Integration

The web application was built using Flask and designed to provide bridge inspectors with a clear and straightforward workflow. Although NMSU's example executable (MyAppInstaller_web.exe) was not used directly, it served as a reference for the intended user interface behavior. Building on this design guidance, the application enables bridge inspectors to:

- Upload drone-acquired aerial images of bridge deck surfaces for analysis;
- Define and select a specific area of interest (AOI) within an image for detailed analysis;
- Submit analysis requests to the backend for automated processing;
- Have each request logged in the backend database as a job, ensuring full traceability and enabling asynchronous processing.

11.3 Processing Service

To manage computationally intensive model inference without compromising the responsiveness of the web application, a standalone Gunicorn-based Python service was developed. This service continuously polls the database for new jobs, upon detecting one, performs the following steps:

- Retrieves the associated drone imagery and defined AOI;
- Executes the TensorFlow-based crack detection model;

- Generates segmentation masks and calculates detailed quantitative crack metrics such as crack length and crack density, and spatial distribution across the AOI;
- Updates the job status in the database and stores the processed results;

This architecture decouples the user interface from backend processing, ensuring that inspectors interact with a responsive, user-friendly web application even while computationally intensive tasks are executed in the background. By supporting asynchronous job management and scalable deployment, the system improves operational reliability and enables inspectors to analyze large datasets across multiple projects simultaneously. Inspectors can upload drone imagery, define an AOI, and receive automated crack detection and quantification results through the web-based interface, streamlining the workflow of bridge deck inspection.

By retraining the model in TensorFlow and integrating it into a modular Flask/Gunicorn architecture, the web application meets the functional requirements specified by NMSU while addressing the practical operational needs of bridge inspectors. This solution demonstrates the potential of combining drone-based imaging with deep learning models to accelerate structural health monitoring, providing both visual segmentation and quantitative crack metrics for informed decision-making. The modular design further establishes a foundation for future improvements, including support for additional bridge defect types, real-time drone-to-web integration, and expanded visualization capabilities for inspection reporting.

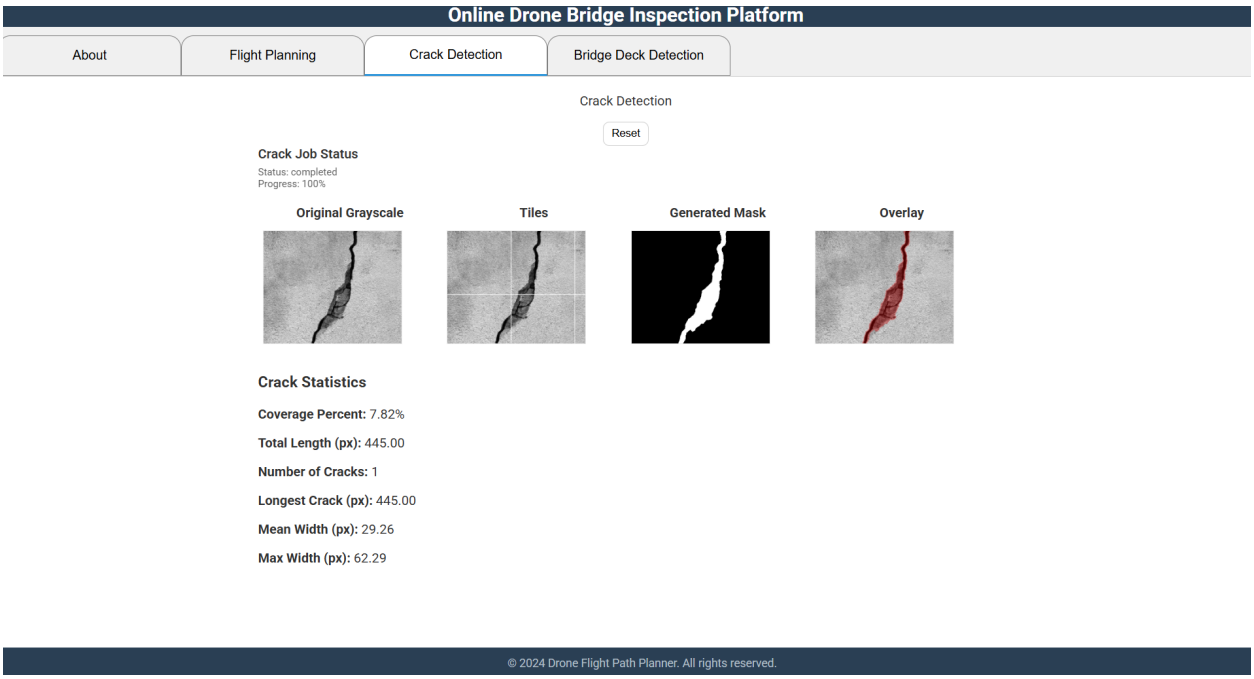


Figure 62. The user interface of the online crack detection and quantification tool

TASK 12: Software Development for Deck Identification

The objective of this task is to develop a web-based application to support the automatic identification of bridge decks from imagery collected via various remote sensing platforms, including but not limited to drones. The web application will leverage a deep learning model provided by the research team at NMSU and deliver results through an intuitive and user-friendly interface, consistent with the framework established in Task 6. This approach ensures that bridge inspectors can efficiently process and analyze imagery while maintaining continuity with the methodologies and workflows developed in previous tasks.

12.1 Model Adaptation

For this task, the research team at NMSU provided a trained semantic segmentation model, deeplabv3plus_bridge_deck.mat. Unlike the crack detection model described in Task 6, the development team at UNM was able to convert it from MATLAB format into ONNX successfully, a widely supported open standard for representing machine learning models.

This direct conversion eliminated the need to replicate the training process in TensorFlow, allowing the model to be used natively within Python via the ONNX format. By leveraging ONNX, integration into the web application was significantly streamlined, reducing development time and minimizing potential errors associated with retraining or reimplementation. Importantly, this approach preserved the accuracy, functionality, and behavior of the original NMSU model, ensuring fidelity to the validated methodology while enabling seamless deployment within the Python-based Flask/Gunicorn architecture.

12.2 Web Application Integration

The web application was developed using Flask, following the same architectural principles established in Task 6 to maintain consistency across tools. The interface enables inspectors to:

- Upload imagery of bridges from drones, handled cameras, or other sources.
- Submit the imagery for backend processing using the ONNX bridge deck model.
- Retrieve and visualize the segmentation results identifying bridge decks.

This architecture builds upon the database-driven job management system implemented in Task 6, enabling scalable processing and reliable tracking of all submitted analysis requests. By centralizing job management, the system ensures that each request is accurately logged, monitored, and processed, supporting asynchronous execution, improving operational efficiency, and providing a robust foundation for handling large volumes of bridge inspection data.

12.3 Processing Service

A dedicated Gunicorn-based Python service was deployed to manage model execution efficiently and reliably. This service continuously monitors the backend database for new bridge deck identification jobs. Once a job is detected, the service performs the following step:

- Retrieves the associated imagery from the database.
- Executes the ONNX-based bridge deck model for semantic segmentation.
- Generating segmentation results that clearly highlights bridge deck areas.
- Updates the job record in the database with both the processed outputs and the current status, including detailed metadata, ensuring full traceability and transparency.

By leveraging and adapting the processing pipeline developed in Task 6, development effort was significantly reduced, while ensuring a robust, modular, and scalable architecture. This approach not only supports efficient integration of the bridge deck identification model but also enhances maintainability, reliability, and performance, providing a solid foundation for future extensions, such as additional defect detection, real-time processing, and expanded visualization capabilities.

12.4 System Workflow

The system workflow closely mirrors the crack detection pipeline described in Task 6, with the key distinction being the integration of the ONNX-based bridge deck identification model. While the overall architecture and process logic remain consistent with the previously established pipeline, this adaptation enables the system to specifically detect and segment bridge decks from diverse imagery sources. The steps in the workflow are as follows:

- Inspectors submit bridge imagery via the web application;
- A new job record is created in the backend database;
- The Gunicorn service detects the job and applies the ONNX deck identification model;
- Segmentation outputs are stored in the database;
- Inspectors access and visualize the deck identification results through the web interface.

The completed system enables inspectors to efficiently, accurately, and consistently identify bridge decks from various remote sensing platforms, including drone-acquired high-spatial resolution aerial imagery. By leveraging the ONNX format, the development team avoided the need to retrain the model, significantly accelerating deployment while maintaining full fidelity to the original methodology. This approach preserves the modular and scalable architecture introduced in Task 6, ensuring that both the frontend and backend components can handle large volumes of data and multiple simultaneous users without compromising performance.

The solution demonstrates the versatility and adaptability of the architecture to accommodate a range of inspection tasks beyond bridge deck identification, highlighting its potential for broader applications in structural health monitoring. The reuse of the Flask frontend, database-driven job management system, and Gunicorn backend service not only ensures a consistent, intuitive, and responsive user experience but also provides a robust foundation for integrating additional models, supporting real-time analysis, and expanding visualization and reporting capabilities in future bridge inspection workflows. By combining automated model execution with a streamlined web interface, the system empowers inspectors to make informed, data-driven decisions more efficiently, ultimately enhancing maintenance planning and safety management.

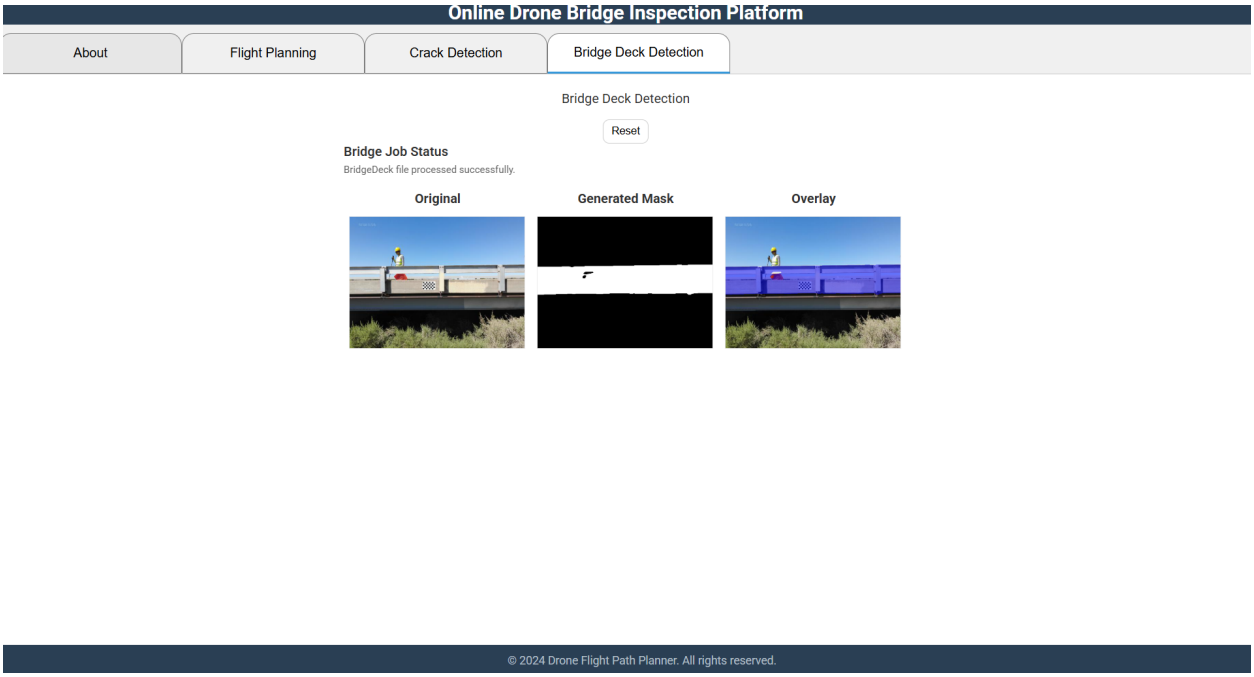


Figure 63. The user interface of the online bridge deck detection tool

TASK 13: Software Development for Bridge Element Identification

The objective of this task was to extend the capabilities of the previously developed web-based bridge inspection portal by integrating automated detection of multiple bridge components from aerial imagery acquired through UAS. In earlier phases of the project, portal functionality was focused primarily on the identification of bridge decks. While valuable, this narrow scope constrained the portal's utility for comprehensive structural assessments. To address this limitation, the current effort incorporated an enhanced deep learning model capable of detecting a wider array of critical structural elements. The enhanced model was trained and optimized to recognize not only bridge decks but also cross girders, main girders, pier caps, and piers, thereby a more holistic understanding of bridge condition and structural performance.

This expanded detection capability required not only improvements to the underlying model but also careful integration into the established portal framework. Consistency with prior design principles, particularly regarding usability, workflow efficiency, and interface clarity, was essential to ensure that the upgraded portal remained intuitive for inspection personnel. Consequently, the user interface and data processing pipeline were refined to support seamless submission of UAS imagery and timely retrieval of analysis outputs.

Furthermore, the integration of multi-element detection into the inspection workflow has important implications for the broader bridge asset management. By automating the identification of several key structural components, the system reduces the reliance on manual review and minimizes the potential for human oversight during initial screening. As the volume of UAS-based inspection data continues to grow, such automated tools will play an increasingly critical role in enabling transportation management agencies to process, interpret, and act upon structural health

information with greater accuracy and efficiency, while simultaneously supporting cost-effective inspection and maintenance operations.

To achieve this enhanced functionality, the development effort incorporated a newly developed semantic segmentation model provided by NMSU. This model served as the core analytical engine for identifying multiple bridge components on UAS-captured imagery. The existing web application, implemented using the Flask framework, was systematically extended to support multi-element detection workflows. These enhancements allowed inspectors to upload imagery (**Figure 64**), initiate automated analyses through the backend processing pipeline, and subsequently review the outputs in the form of color-coded segmentation masks corresponding to each detected structural element (**Figure 65**).

The updated capability was integrated seamlessly into the job management system established in earlier project phases, ensuring consistency in task handling, status reporting, and results retrieval. A dedicated system service manages the execution of the deep learning model by continuously monitoring the database for pending element detection jobs. When a job is identified, the service retrieves the corresponding imagery, applies preprocessing steps such as resizing, normalization, and optional enhancement, and runs the model to generate detailed semantic segmentation outputs. Each predicted class is mapped to a predefined color palette, producing both a standalone segmentation mask and an overlay image for intuitive and informative visualization in the web interface for each predicted class.

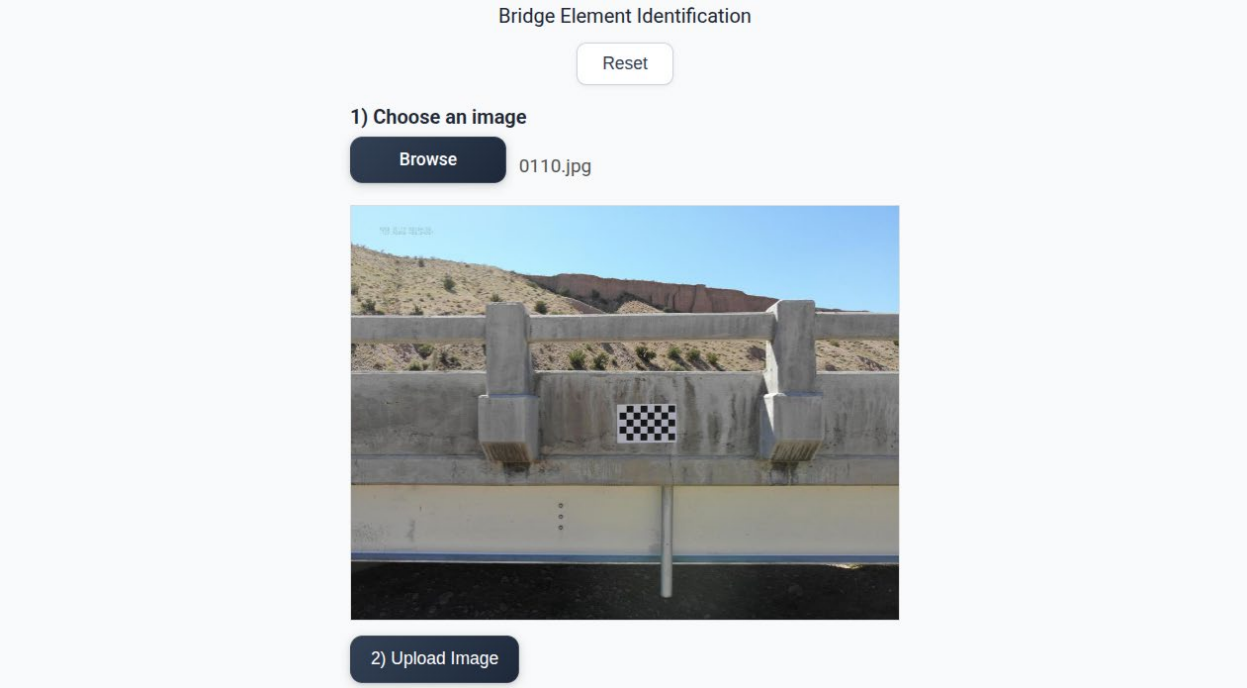


Figure 64. The user interface designed to facilitate the submission of images for automated detection and classification of bridge structural components

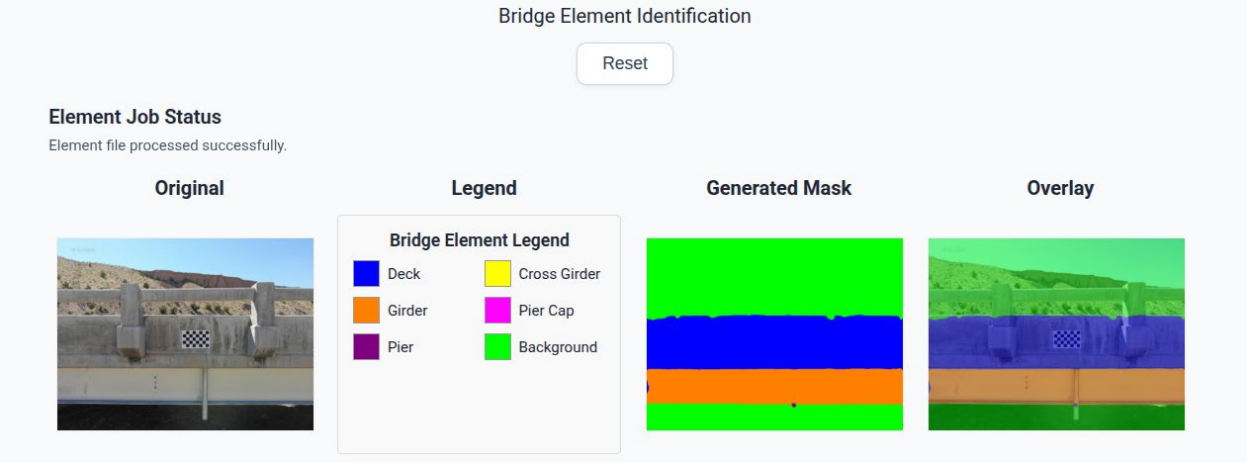


Figure 65. The output interface for identified bridge elements

Collectively, these enhancements significantly strengthened the portal's end-to-end functionality, creating a more comprehensive, efficient, and user-friendly inspection workflow. By automating key processes, the portal increases scalability, reduces manual effort, and accelerates the processing of large volumes of UAS imagery, while producing standardized and reliable outputs. These capabilities support better-informed, data-driven decision-making for bridge maintenance and management, facilitate consistent reporting, and enhance the overall accuracy, reliability, and effectiveness of structural assessments.

TASK 14: Software Development for Anomaly Detection and NBI-Correlated Bridge Deck Ratings

The objective of this task was to extend the existing web-based bridge inspection system by implementing an automated anomaly detection and NBI-correlated deck rating service. This enhancement enables quantitative assessment of bridge deck condition by leveraging imagery acquired by inspectors or UAS. The portal is designed to identify anomalous regions indicative of structural deterioration, aggregate reconstruction-error statistics, and generate global condition assessments that correspond to the standardized NBI rating scale. By incorporating this capability, the portal establishes a standardized, data-driven framework for bridge inspection that augments traditional visual assessments. It enables inspectors and engineers to identify structural anomalies in a systematic way, quantify deterioration in an accurate manner, and evaluate overall deck condition using objective metrics. This approach not only improves the consistency and repeatability of inspections but also supports evidence-based decision-making for maintenance prioritization, resource allocation, and long-term asset management, ultimately enhancing the safety, reliability, and efficiency of bridge infrastructure operations.

Upon image upload (**Figure 66**), the portal supports multiple files in a single session and accepts a wide variety of common image formats. Each image is converted to grayscale to standardize input and reduce computational complexity. The image is then divided into non-overlapping 64×64 pixel patches, which are normalized to prepare for model inference. This patch-based approach allows for localized detection of anomalies, improving sensitivity to subtle structural defects that may be missed in full-image analysis. The system computes reconstruction error for each patch

using mean squared error (MSE), with patches exceeding a threshold of 0.015 designated as anomalous. This threshold was empirically determined to balance sensitivity and specificity for detecting areas of potential deterioration.

The preprocessed patches are passed through a pre-trained anomaly detection model. For each image, the patch-level errors are aggregated into an error map, which is up-sampled to full resolution and overlaid onto the original grayscale image to produce a heat map. The heat map provides a visual representation of anomalous regions, enabling inspectors to interpret areas of concern in a rapid fashion. This visualization supports both single-image and multi-image workflows, facilitating rapid assessment of multiple decks within a single inspection session.

After processing all uploaded imagery, the system computes global metrics across the dataset, including the average reconstruction error, the proportion of anomalous patches, and a normalized severity score. These metrics provide a quantitative summary of deck condition and form the basis for assigning a bridge-level assessment. A custom rule-based classifier maps the aggregated metrics to both a descriptive condition label (ranging from "Excellent" to "Critical") and an NBI-equivalent deck rating (ranging from nine [9] to one [1]). The classification logic applies empirically determined thresholds on average error, anomalous area percentage, and severity score to ensure consistency with the NBI condition scale.

All computed outputs, including patch-level error maps, heat maps, global metrics, descriptive condition labels, and NBI-equivalent ratings, are stored within the system and presented through the web interface for inspector review (**Figure 67**). This automated workflow seamlessly integrates with the existing inspection platform, preserving established job management, file handling, and model execution frameworks. By automating anomaly detection and condition rating, the system reduces reliance on manual interpretation, enhances scalability for processing large volumes of UAS imagery, standardizes reporting, and improves the reliability and repeatability of bridge condition assessments. Overall, this enhancement provides inspectors and maintenance planners with a robust, data-driven tool for evaluating structural health, prioritizing interventions, and supporting long-term infrastructure management.

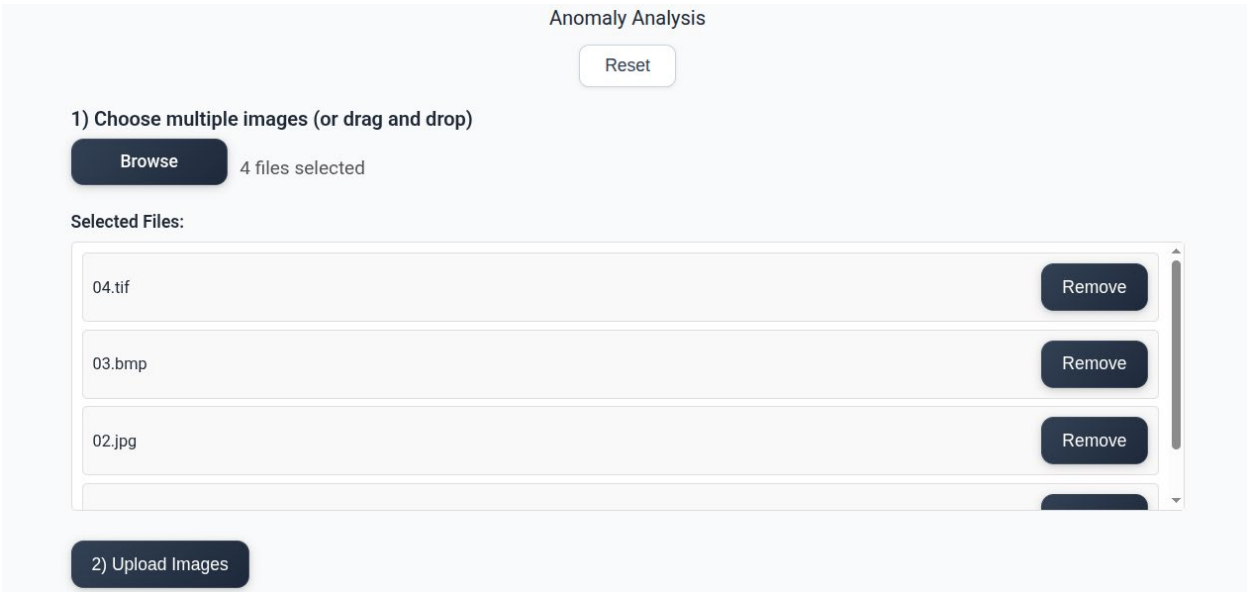


Figure 66. The user interface designed to facilitate the submission of images for automated anomaly detection and NBI-correlated bridge deck ratings.

Figure 67. The outputs of the anomaly analysis comprising heat maps of detected regions, patch-level error metrics, and aggregated quantitative indicators of bridge deck condition.

TASK 15: Software Development for Concrete Spalling Detection

15.1 Objective

The primary objective of this task is to integrate a deep learning-based model for the automated detection of concrete spalling, developed by NMSU, into a pre-existing web-based bridge inspection platform created by the University of New Mexico (UNM) team. As mentioned in the previous section, concrete spalling, characterized by the progressive detachment, flaking, or fracturing of the surface layer of concrete, represents a significant deterioration mechanism that can compromise structural integrity, reduce load-carrying capacity, and accelerate further degradation if left undetected. Early detection of spalling is therefore critical for the maintenance, preservation, and safety assessment of bridge infrastructure. Traditional inspection methods rely heavily on manual visual inspection, which is labor-intensive, subject to human error, and often limited in scope, particularly for large or difficult-to-access structures. The integration of automated spalling detection offers the potential to enhance inspection efficiency, reduce subjective bias, and enable timely maintenance interventions.

The online inspection platform has been designed and developed as a comprehensive platform for drone-based bridge inspection, with capabilities including the planning of drone flight paths, the organization and storage of inspection data, and the processing and analysis of imagery acquired from both aerial and ground-based cameras. In the present task, the platform is being extended to incorporate automated analytical capabilities for the identification of regions exhibiting structural degradation within captured images. The integration of the NMSU deep learning model enhances the platform by enabling automated, pixel-level detection of spalled regions in concrete surfaces, thereby generating both quantitative metrics and qualitative visual outputs that are immediately interpretable by human inspectors. A primary emphasis of the integration effort was to ensure seamless interoperability between the model and all platform components, including the frontend visualization interface, the backend computational pipeline, and the database-driven job management system. The overarching objective of this integration is to deliver a robust, fully operational system capable of providing accurate and interpretable spalling detection results, while requiring minimal specialized technical expertise from end users.

15.2 Model Adaption, Web Application Integration, and Processing Service

The model provided by NMSU was based on the U-Net architecture, a convolutional neural network (CNN) framework that has become widely adopted for image segmentation tasks requiring precise localization of features within complex visual scenes. The U-Net model was trained to identify spalled regions at the pixel level, enabling highly granular detection that is critical for structural health monitoring applications. Semantic segmentation using U-Net is particularly advantageous in this context because it enables the identification and localization of subtle textual and morphological variations in concrete surfaces associated with early-stage deterioration. Unlike conventional image classification approaches, such as standard CNNs or patch-based classifiers, which typically analyze images at a coarse regional level, U-Net performs pixel-level prediction. This capability preserves fine spatial details and structural context, allowing the model to capture small-scale defects that might otherwise be overlooked. Consequently, U-Net provides greater sensitivity to fine-grained structural anomalies, improving the accuracy and reliability of early defect detection in concrete infrastructure such as bridges.

The model was provided by the NMSU team in the Open Neural Network Exchange (ONNX) format, a platform-independent standard designed to facilitate interoperability across diverse machine learning frameworks and runtime environments. The availability of the model in ONNX format eliminated the need for additional retraining or model conversion, thereby preserving the original learned parameters, architectural configuration, and performance characteristics. By maintaining the integrity of the pre-trained model, this approach reduces the risk of performance degradation and minimizes potential sources of error that may arise from retraining on limited datasets or from incompatibilities introduced during cross-framework conversion. Furthermore, the ONNX format enables seamless deployment across different computational environments and hardware platforms without modification. This portability enhances reproducibility and ensures consistent model performance throughout the research workflow.

For deployment, ONNX Runtime was incorporated into the existing Python-based backend environment of the online inspection platform. ONNX Runtime provides a highly optimized inference engine designed to execute ONNX models efficiently, particularly on CPU-based servers. This capability makes it particularly suitable for web-based applications, where access to specialized hardware such as graphics processing units (GPUs) may be limited or unavailable. By

enabling efficient model execution on standard CPU-based computing infrastructure, ONNX Runtime supports scalable and accessible deployment within web-integrated analytical platforms. Moreover, the runtime's optimization features, including graph-level and kernel-level performance enhancements, reduce inference latency and improve throughput, enabling the system to handle multiple concurrent inspection requests without compromising responsiveness. This approach also ensures that the model operates consistently and reliably across different user environments, preserving its performance characteristics during real-world use.

The integration process followed the preprocessing and inference protocols specified by the NMSU team. Adhering to these established procedures ensured that the operational conditions within the UNM platform remained consistent with those used during the model's original training and validation. Maintaining this alignment is essential for preserving the model's predictive reliability and ensuring that inference results generated within the deployed environment accurately reflect the performance characteristics demonstrated during model development. Moreover, following the original protocols substantially reduces the chance of errors or inconsistencies that could affect the model's results. This approach also makes it easier for other researchers or practitioners to reproduce the results under similar conditions, increasing confidence in the model's reliability and usefulness for real-world applications.

Image preprocessing is a critical step in maintaining model fidelity and ensuring reliable predictions. Raw images captured by drones during inspections were first processed using standard image processing libraries to guarantee consistent handling and uniform compatibility across the dataset. Because camera sensors typically store images in BGR (Blue-Green-Red) color order, while the model had been trained on RGB (Red-Green-Blue) inputs, each image was converted to RGB to preserve color consistency, which is essential for accurate feature representation. Subsequently, pixel values, originally represented as unsigned 8-bit integers ranging from 0 to 255, were normalized to a floating-point range between 0 and 1. This scaling not only standardizes the input data but also ensures that the numerical range aligns with the conditions under which the model was trained. By standardizing intensity values, normalization reduces distortions and potential biases in the model's predictions caused by variations in illumination or sensor calibration, while also improving numerical stability during inference.

In addition to color conversion and normalization, particular emphasis was placed on preserving the spatial resolution and structural fidelity of the images, as fine-grained textural and morphological features are critical for the accurate identification of early-stage deterioration in concrete surfaces. Preprocessing procedures were designed to ensure that subtle surface variations, including small-cracks and minor structural irregularities, were retained without distortion or blurring, thereby maintaining the level of detail required for effective feature extraction by the model. By standardizing these preprocessing steps and rigorously adhering to the protocols established during model training, the input data preserves the characteristics essential for accurate and robust inference across diverse real-world inspection conditions.

Following normalization, the images were systematically reshaped to conform to the input tensor format required by the U-Net model. Specifically, images were transformed from the conventional height × width × channel (H × W × C) layout to channel × height × width (C × H × W), reflecting the ordering expected by the network architecture. To accommodate batch processing, an additional dimension was introduced, resulting in a four-dimensional (4D) tensor compatible with ONNX Runtime inference sessions. This preparation ensures that each image precisely aligns with

the structural and dimensional specifications of the neural network, which is critical for preserving the spatial and contextual relationships necessary for accurate segmentation. Furthermore, standardizing the tensor shape across all input images facilitates efficient parallel computation during inference and mitigates the risk of shape-related errors that could otherwise compromise the model's predictive reliability. By rigorously adhering to these data formatting procedures, the preprocessing pipeline guarantees that the U-Net model receives inputs that are fully consistent with the conditions under which it was trained, thereby optimizing segmentation performance and robustness in real-world inspection scenarios.

During inference, the preprocessed tensors were submitted to ONNX Runtime, which executed the U-Net model to produce raw output logits for each pixel. These logits represent the model's degree of confidence regarding whether each pixel belongs to a spalled region. A sigmoid activation function was subsequently applied to convert the logits into probability values bounded between 0 and 1. A probability threshold of 0.5 was employed to generate a binary segmentation mask, with pixels exceeding this threshold classified as spalled and those below classified as intact concrete. This binary output provides a quantitative, machine-readable representation of detected spalling regions, which can be used for automated assessment, statistical analysis, or integration into broader structural health monitoring workflows.

In addition to the binary segmentation mask, the platform generates a visual overlay in which the detected spalled regions are superimposed onto the original inspection image with partial transparency. This visualization preserves the spatial and contextual details of the underlying concrete surface while emphasizing areas of deterioration, thereby facilitating intuitive interpretation by human inspectors. The overlay allows rapid evaluation of the spatial distribution, extent, and relative severity of spalling, enabling inspectors to prioritize maintenance actions and make informed decisions in real time. By providing both the quantitative binary mask and the visually interpretable overlay, the platform supports automated analysis for downstream processing, such as statistical quantification, defect trend monitoring, and integration with structural health monitoring databases, while simultaneously enhancing human interpretability and operational usability. This dual output approach ensures that the system is not only analytically robust but also practically applicable in field inspection scenarios, bridging the gap between machine-generated predictions and actionable insights for infrastructure management.

15.3 Web Application User Interface and Output

The developed web application offers an intuitive and user-friendly interface for uploading and analyzing images containing potential spalling, effectively streamlining the inspection workflow for both automated and human-guided assessment. As illustrated in **Figure 68**, users initiate the process by selecting one or more images through the "Choose Image(s)" button, which opens a standard file selection dialog compatible with common image formats. Once the images are selected, the interface displays the names of the uploaded files and provides options to remove any undesired images, enabling users to verify and refine their selection prior to processing, as shown in **Figure 69**. Following confirmation, the application executes the complete inference pipeline, automatically processing the images through the deployed U-Net model and ONNX Runtime framework. The final output, presented in **Figure 70**, is delivered in two complementary formats to support diverse analytical, operational, and reporting needs. First, a binary segmentation mask is produced, providing a precise, machine-readable representation of the detected spalling regions suitable for quantitative analysis, trend monitoring, or integration into broader structural health

assessment workflows. Second, a visual overlay is generated by superimposing the detected spalled areas onto the original images, preserving critical contextual details and enabling inspectors to intuitively interpret the spatial distribution, extent, and relative severity of the defects. This dual-output framework not only ensures analytical rigor and reproducibility but also enhances practical usability, bridging the gap between automated computational analysis and human inspection to support informed decision-making in real-world infrastructure monitoring and maintenance applications. As a result, the web application in the online inspection platform supports informed, evidence-based decision-making in real-world infrastructure monitoring, facilitating timely maintenance interventions, prioritization of critical defects, and integration with broader structural health assessment workflows.

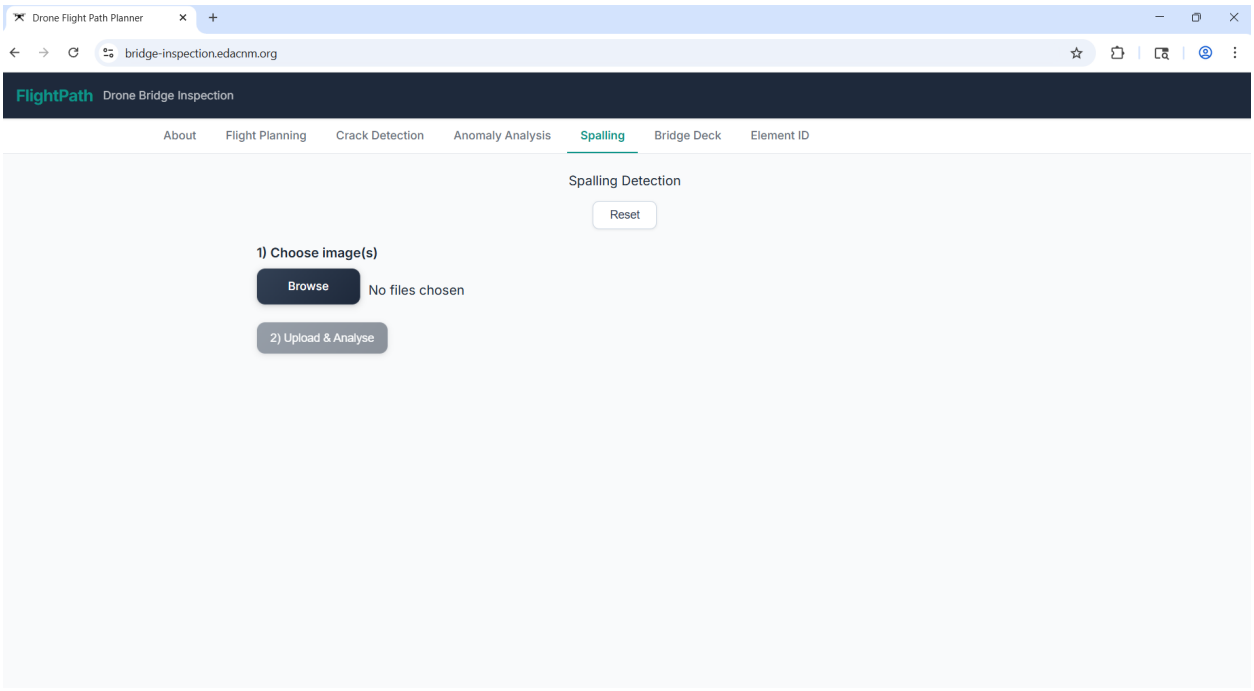


Figure 68. User interface for uploading inspection images, featuring the "Choose Image(s)" button for selecting files

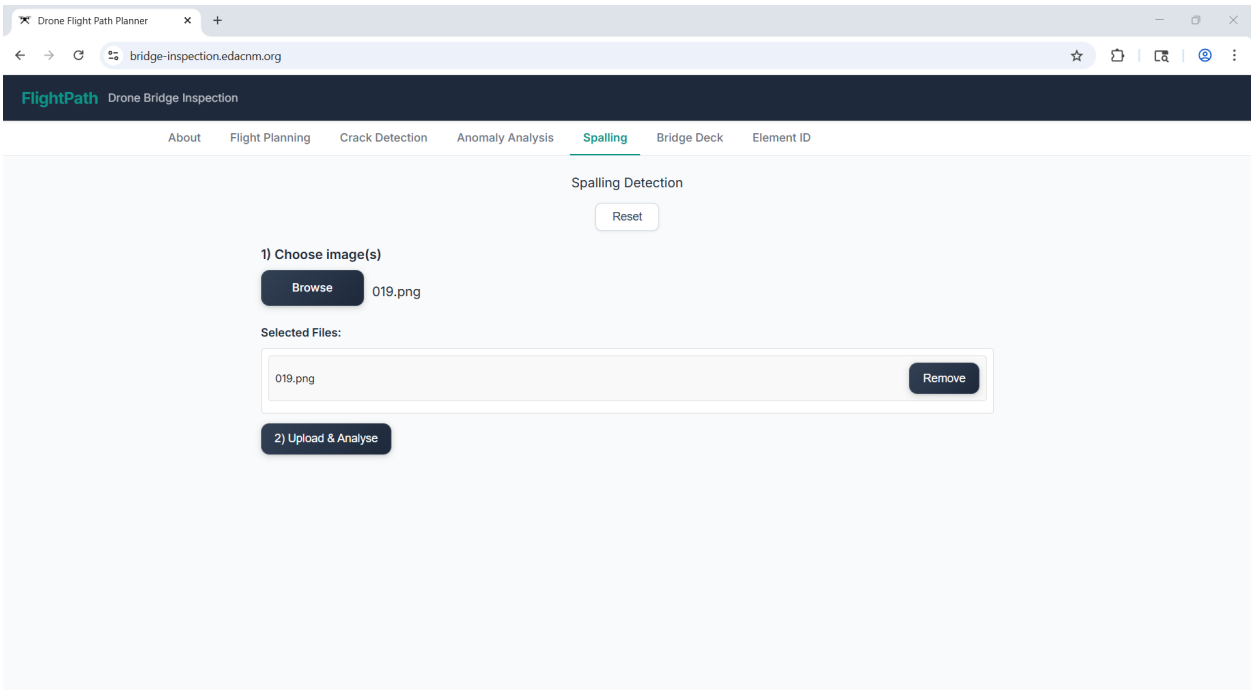


Figure 69. Preview of uploaded images with the option to remove any undesired files before processing

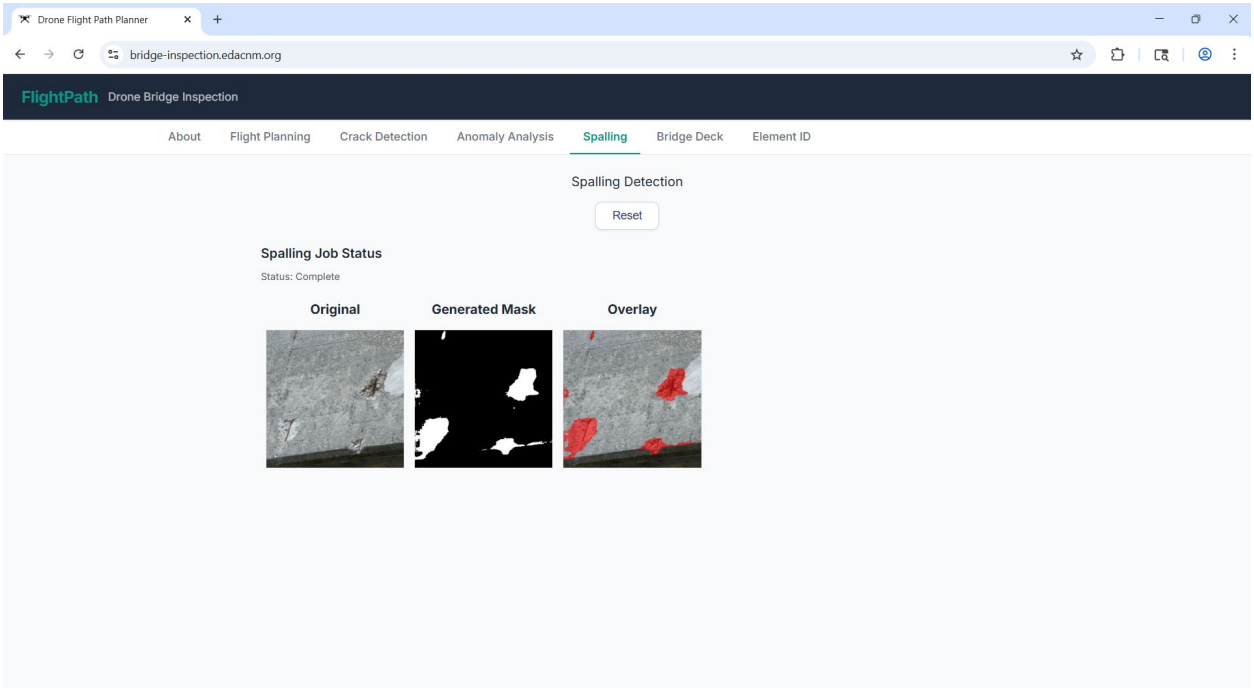


Figure 70. Final output of the inference pipeline, including both a binary segmentation mask and a visual overlay highlighting detected spalled regions on the original image

TASK 16: UAS-Enabled Bridge Inspection Workshop

In order to transfer the technology and knowledge required for UAS-enabled bridge inspection to NMDOT bridge inspectors through a workshop, the following agenda has been developed by NMSU and UNM teams, which includes both hands-on and mind-on panels:

March 19th

9:30-10:00 Coffee and check-in

10:00-10:15 Welcome remarks

10:15-11:00

Module 1: UAS Fundamentals and Use Cases

- Multi-rotor vs. fixed-wing, payloads (RGB, oblique, thermal), GNSS/RTK
- Bridge-specific constraints: occlusions, GPS-denied areas, wind and ground effect

11:15 -12:00

Module 2: FAA Regulations and Pilot Certification

- Part 107 overview, waivers (for example night operations), LAANC, airspace classes
- Crew roles (PIC and VO), preflight documentation, and incident reporting

12:00-1:00 Lunch

1:15-2:00

Module 3: Pilot Training Pathway

- Part 107 exam prep resources, recurrent training, record-keeping
- Agency SOPs and risk management (SMS principles)

2:15-3:15

Module 4: Data Collection Strategies for Bridge Inspection

- Flight envelopes for decks, girders, diaphragms, and barriers
- Nadir and oblique capture, baseline overlap, GSD targets, exposure control
- Under-bridge tactics: standoff, lighting, and prop-wash considerations
- QC checklists (coverage heatmaps and redundancy for shadowed areas)

3:30-4:00

Module5: Data Processing Software Showcase

- AI workflow overview: semantic segmentation and crack masks
- Tooling landscape: commercial and open-source options, hardware sizing

Closing for the day

March 20th:

9:30-10:00 Coffee and check in

10:00-11:30

Module 6: Simulator Lab (Hands On)

- Basic flight controls and emergency procedures
- Automated mission design: waypoint, corridor, and structure-scan templates
- Telemetry interpretation and fail-safe drills

11:30 -1:00 Lunch and Preparation for Site Demonstration

1:30-After

Module 7: Field Demonstration (Site Dependent)

- On-site hazard brief, cordon setup, and stakeholder communication
- Mission execution with live telemetry screen

Rapid QA on captured imagery (coverage and sharpness)

Concluding Remarks

Based on this report, the following concluding remarks can be stated:

1) Task 1 established the foundational requirements for safe, compliant, and effective UAS operations for bridge inspection. By summarizing the pathways for both first-time pilots and existing Part 61 certificate holders, the task provides a practical guide for obtaining and maintaining FAA Part 107 remote pilot certification. The outlined procedures support the development of qualified personnel capable of conducting UAS-based bridge inspections in accordance with federal aviation requirements and agency safety expectations.

2) Task 2 identified the key factors that should guide UAS platform and payload selection for bridge inspection applications. The comparison of UAV types, endurance, payload capacity, stability, cost, and sensor compatibility demonstrates that multi-rotor UAVs equipped with appropriate RGB, thermal, or LiDAR sensors can effectively support different inspection objectives. The recommendations developed in this task provide a practical basis for selecting hardware that balances data quality, operational safety, cost efficiency, and mission-specific requirements.

3) Task 3 demonstrated the importance of systematic pre-flight planning, camera calibration, and mission design for accurate UAS-based bridge inspection. The integration of GSD calculations, flight height selection, image overlap considerations, and both in-lab and on-site camera calibration improved the reliability of image-based measurements. The validation results confirmed that calibrated imagery can enhance crack-width estimation accuracy, supporting more reliable and repeatable bridge condition assessment.

4) Task 4 developed and evaluated a flight path optimization framework to improve the efficiency of UAS-based inspection missions. By incorporating Particle Swarm Optimization, area coverage constraints, image overlap requirements, obstacle avoidance, and geographic coordinate conversion, the proposed approach enables more efficient waypoint generation for bridge inspection. The results showed that optimized flight paths can reduce flight time and energy consumption, helping address the limited battery endurance of multi-rotor UAVs while maintaining adequate inspection coverage.

5) Task 5 established a comprehensive bridge inspection dataset collected from multiple bridges across New Mexico using UAS-based RGB and infrared imaging. The collected data include diverse structural components, surface conditions, and damage patterns, creating a valuable foundation for image processing, machine learning model training, validation, and testing. Continued dataset expansion and refinement will further improve the robustness, generalizability, and practical applicability of automated bridge inspection models.

6) Task 6 developed a deep learning-based semantic segmentation framework for automatic bridge deck identification from UAS imagery. The trained DeepLab v3+ model demonstrated strong performance in distinguishing deck surfaces from surrounding background elements, even under varying textures, lighting conditions, and image perspectives. This capability provides an essential preprocessing step for downstream inspection tasks, including crack detection, deck condition assessment, and automated damage quantification.

7) Task 7 developed a deep learning-based crack detection and quantification framework using UAS-acquired bridge deck imagery. The U-Net-based segmentation model enabled pixel-level

identification of cracks and supported quantitative assessment through crack coverage and related damage metrics. The results demonstrate the potential of combining UAV imagery and deep learning to support faster, more objective, and more scalable bridge deck condition evaluation.

8) Task 8 introduced an automated bridge deck health evaluation framework that connects UAS imagery, unsupervised anomaly detection, and NBI-aligned condition assessment. By using sparse autoencoder-based reconstruction error maps and aggregated deck-level metrics, the framework enables label-free identification of anomalous regions and supports condition rating interpretation. This approach provides a promising pathway for transforming large volumes of UAV imagery into standardized, decision-support information for bridge asset management.

9) Task 9 expanded the UAS-based inspection framework by developing a semantic segmentation model for identifying multiple bridge elements, including decks, girders, piers, pier caps, and other structural components. Automated bridge element identification improves the organization and interpretation of inspection imagery by linking visual data to specific structural components. This capability supports more detailed condition assessment, improved reporting, and future integration with element-level bridge management systems.

10) Task 10 developed a deep learning-based framework for concrete spalling segmentation using UAS imagery and optimized model training. The proposed method produced pixel-level spalling masks and visual overlays that allow both quantitative analysis and intuitive interpretation of damaged regions. The results demonstrate that automated spalling segmentation can support more consistent, efficient, and objective evaluation of concrete surface deterioration in bridge inspection workflows.

11) Task 11 translated the crack detection and quantification model into a web-based inspection tool. By recreating the model in a Python-compatible framework and integrating it into a Flask/Gunicorn-based architecture, the developed software enables users to upload images, define areas of interest, process inspection data, and obtain automated crack detection outputs. This task represents an important step toward practical deployment of AI-assisted crack assessment tools for bridge inspectors.

12) Task 12 implemented the bridge deck identification model within a web-based application environment. The developed interface and backend processing workflow allow users to upload inspection imagery and automatically identify bridge deck regions. This software capability improves usability, supports efficient preprocessing of UAS imagery, and provides a scalable foundation for integrating deck-level analysis into broader bridge inspection and condition assessment workflows.

13) Task 13 extended the web-based inspection platform to support automated identification of multiple bridge structural elements. By integrating a multi-class semantic segmentation model into the existing software architecture, the system enables users to process UAS imagery and obtain color-coded outputs for different bridge components. This expanded functionality improves the platform's usefulness for comprehensive bridge inspection and supports more organized, component-specific structural assessment.

14) Task 14 integrated the anomaly detection and NBI-correlated bridge deck rating framework into the web-based inspection platform. The software enables automated generation of heat maps, reconstruction-error metrics, and aggregated condition indicators from uploaded bridge deck

imagery. This capability helps convert image-based inspection data into interpretable condition information, supporting more efficient screening, prioritization, and data-driven bridge maintenance decision-making.

15) Task 15 implemented the concrete spalling detection model into the online inspection platform, enabling users to upload inspection images and receive automated segmentation masks and visual overlays of spalled regions. The developed tool bridges the gap between deep learning model development and practical field application by providing outputs that are both quantitatively useful and visually interpretable. This functionality supports more efficient defect documentation, maintenance planning, and structural health monitoring.

16) Task 16 supported technology transfer by developing and delivering a UAS-enabled bridge inspection workshop for NMDOT personnel. The workshop combined technical instruction, regulatory guidance, data collection strategies, software demonstrations, and field-based learning to improve practical understanding of UAS inspection workflows. This effort strengthened the connection between research outcomes and implementation, helping prepare transportation personnel to adopt UAS and AI-assisted tools for bridge inspection and asset management.

References

[1] S. Feroz and S. Abu Dabous, "UAV-Based Remote Sensing Applications for Bridge Condition Assessment," *Remote Sensing*, vol. 13, no. 9, p. 1809, May 2021, doi: 10.3390/rs13091809.

[2] Y. Xu and Y. Turkan, "BrIM and UAS for bridge inspections and management," *ECAM*, vol. 27, no. 3, pp. 785–807, Nov. 2019, doi: 10.1108/ECAM-12-2018-0556.

[3] S. Sreenath, H. Malik, N. Husnu, and K. Kalaichelavan, "Assessment and Use of Unmanned Aerial Vehicle for Civil Structural Health Monitoring," *Procedia Computer Science*, vol. 170, pp. 656–663, 2020, doi: 10.1016/j.procs.2020.03.174.

[4] W. W. Greenwood, J. P. Lynch, and D. Zekkos, "Applications of UAVs in Civil Infrastructure," *J. Infrastruct. Syst.*, vol. 25, no. 2, p. 04019002, Jun. 2019, doi: 10.1061/(ASCE)IS.1943-555X.0000464.

[5] S. HekmatiAthar, N. Goudarzi, A. Karimoddini, A. Homaifar, and D. Divakaran, "A systematic evaluation and selection of UAS-enabled solutions for bridge inspection practices," in *2020 IEEE Aerospace Conference*, Big Sky, MT, USA: IEEE, Mar. 2020, pp. 1–11. doi: 10.1109/AERO47225.2020.9172795.

[6] Z. Ameli, Y. Aremanda, W. A. Friess, and E. N. Landis, "Impact of UAV Hardware Options on Bridge Inspection Mission Capabilities," *Drones-Basel*, vol. 6, no. 3, p. 64, Mar. 2022, doi: 10.3390/drones6030064.

[7] B. Hubbard and S. Hubbard, "Unmanned Aircraft Systems (UAS) for Bridge Inspection Safety," *Drones*, vol. 4, no. 3, p. 40, Aug. 2020, doi: 10.3390/drones4030040.

[8] C. Cheng, Z. Shang, and Z. Shen, "Automatic delamination segmentation for bridge deck based on encoder-decoder deep learning through UAV-based thermography," *NDT & E International*, vol. 116, p. 102341, Dec. 2020, doi: 10.1016/j.ndteint.2020.102341.

[9] S. Dorafshan, L. E. Campbell, M. Maguire, and R. J. Connor, "Benchmarking Unmanned Aerial Systems-Assisted Inspection of Steel Bridges for Fatigue Cracks," *Transportation Research Record*, vol. 2675, no. 9, pp. 154–166, Sep. 2021, doi: 10.1177/03611981211001073.

[10] S. Chen, D. F. Laefer, E. Mangina, S. M. I. Zolanvari, and J. Byrne, "UAV Bridge Inspection through Evaluated 3D Reconstructions," *Journal of bridge engineering*, vol. 24, no. 4, 2019, doi: 10.1061/(ASCE)BE.1943-5592.0001343.

[11] Q. Chen, X. Wen, S. Lu, and D. Sun, "Corrosion Detection for Large Steel Structure base on UAV Integrated with Image Processing System," *IOP Conf. Ser.: Mater. Sci. Eng.*, vol. 608, no. 1, p. 012020, Aug. 2019, doi: 10.1088/1757-899X/608/1/012020.

[12] P. Debus and V. Rodehorst, "Multi-scale Flight Path Planning for UAS Building Inspection," in *Proceedings of the 18th International Conference on Computing in Civil and Building Engineering*, vol. 98, E. Toledo Santos and S. Scheer, Eds., in Lecture Notes in Civil Engineering, vol. 98. , Cham: Springer International Publishing, 2021, pp. 1069–1085. doi: 10.1007/978-3-030-51295-8_74.

[13] P. Almasi, R. Premadasa, S. Rouhbakhsh, Y. Xiao, Z. Wan, and Q. Zhang, "A Review of Developments and Challenges of Preflight Preparation for Data Collection of UAV-based Infrastructure Inspection," *CTCSE*, vol. 10, no. 2, Jan. 2024, doi: 10.33552/CTCSE.2024.10.000734.

[14] P. Almasi *et al.*, "A General Method for Pre-Flight Preparation in Data Collection for Unmanned Aerial Vehicle-Based Bridge Inspection," *Drones*, vol. 8, no. 8, p. 21, Aug. 2024, doi: 10.3390/drones8080386.

[15] Y. Liu, X. Nie, J. Fan, and X. Liu, “Image-based crack assessment of bridge piers using unmanned aerial vehicles and three-dimensional scene reconstruction,” *Computer-Aided Civil and Infrastructure Engineering*, vol. 35, no. 5, pp. 511–529, May 2020, doi: 10.1111/mice.12501.
[16] H. Yanagi and H. Chikatsu, “CAMERA CALIBRATION IN 3D MODELLING FOR UAV APPLICATION,” *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, vol. XL-4/W5, pp. 223–226, May 2015, doi: 10.5194/isprsarchives-XL-4-W5-223-2015.
[17] M. Cramer, H.-J. Przybilla, and A. Zurhorst, “UAV CAMERAS: OVERVIEW AND GEOMETRIC CALIBRATION BENCHMARK,” *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, vol. XLII-2/W6, pp. 85–92, Aug. 2017, doi: 10.5194/isprs-archives-XLII-2-W6-85-2017.
[18] S. Yu, R. Zhu, L. Yu, and W. Ai, “Effect of Checkerboard on the Accuracy of Camera Calibration,” in *Advances in Multimedia Information Processing – PCM 2018*, vol. 11166, R. Hong, W.-H. Cheng, T. Yamasaki, M. Wang, and C.-W. Ngo, Eds., in Lecture Notes in Computer Science, vol. 11166. , Cham: Springer International Publishing, 2018, pp. 619–629. doi: 10.1007/978-3-030-00764-5_57.
[19] *MATLAB*. ((R2024a)). The MathWorks, Inc. [Online]. Available: https://www.mathworks.com
[20] Z. Zhang, “A flexible new technique for camera calibration,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, no. 11, pp. 1330–1334, Nov. 2000, doi: 10.1109/34.888718.
[21] “Reprojection error.” [Online]. Available: https://support.pix4d.com/hc/en-us/articles/202559369-Reprojection-error
[22] J. Yuan *et al.*, “Global Optimization of UAV Area Coverage Path Planning Based on Good Point Set and Genetic Algorithm,” *Aerospace*, vol. 9, no. 2, p. 86, Feb. 2022, doi: 10.3390/aerospace9020086.
[23] A. Kaveh, P. Almasi, and A. Khodagholi, “Optimum Design of Castellated Beams Using Four Recently Developed Meta-heuristic Algorithms,” *Iran J Sci Technol Trans Civ Eng*, vol. 47, no. 2, pp. 713–725, Apr. 2023, doi: 10.1007/s40996-022-00884-z.
[24] Ramin Babazadeh Dizaj and Nastaran Sabahi, “Optimizing LSM-LSF composite cathodes for enhanced solid oxide fuel cell performance: Material engineering and electrochemical insights,” *World J. Adv. Res. Rev.*, vol. 20, no. 1, pp. 1284–1291, Oct. 2023, doi: 10.30574/wjarr.2023.20.1.2183.
[25] D. E. Goldberg and J. H. Holland, “Genetic Algorithms and Machine Learning,” *Machine Learning*, vol. 3, no. 2/3, pp. 95–99, 1988, doi: 10.1023/A:1022602019183.
[26] M. Dorigo, V. Maniezzo, and A. Colorni, “Ant system: optimization by a colony of cooperating agents,” *IEEE Trans. Syst., Man, Cybern. B*, vol. 26, no. 1, pp. 29–41, Feb. 1996, doi: 10.1109/3477.484436.
[27] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of ICNN’95 - International Conference on Neural Networks*, Perth, WA, Australia: IEEE, 1995, pp. 1942–1948. doi: 10.1109/ICNN.1995.488968.
[28] A. Kaveh and S. Talatahari, “A novel heuristic optimization method: charged system search,” *Acta Mech*, vol. 213, no. 3–4, pp. 267–289, Sep. 2010, doi: 10.1007/s00707-009-0270-4.
[29] A. Kaveh and V. R. Mahdavi, “Colliding bodies optimization: A novel meta-heuristic method,” *Computers & Structures*, vol. 139, pp. 18–27, Jul. 2014, doi: 10.1016/j.compstruc.2014.04.005.

[30] A. Kaveh and A. Zaerreza, “Shuffled Shepherd Optimization Method: A New Meta-Heuristic Algorithm,” in *Structural Optimization Using Shuffled Shepherd Meta-Heuristic Algorithm*, vol. 463, in Studies in Systems, Decision and Control, vol. 463. , Cham: Springer Nature Switzerland, 2023, pp. 11–52. doi: 10.1007/978-3-031-25573-1_2.
[31] A. Kaveh and F. Shokohi, “APPLICATION OF GREY WOLF OPTIMIZER IN DESIGN OF CASTELLATED BEAMS”.
[32] Z. Fu, J. Yu, G. Xie, Y. Chen, and Y. Mao, “A Heuristic Evolutionary Algorithm of UAV Path Planning,” *Wireless Communications and Mobile Computing*, vol. 2018, pp. 1–11, Sep. 2018, doi: 10.1155/2018/2851964.
[33] Z. Yu, Z. Si, X. Li, D. Wang, and H. Song, “A Novel Hybrid Particle Swarm Optimization Algorithm for Path Planning of UAVs,” *IEEE Internet Things J.*, vol. 9, no. 22, pp. 22547–22558, Nov. 2022, doi: 10.1109/JIOT.2022.3182798.
[34] H. Rienecker, V. Hildebrand, and H. Pfifer, “Energy optimal 3D flight path planning for unmanned aerial vehicle in urban environments,” *CEAS Aeronaut J*, vol. 14, no. 3, pp. 621–636, Jul. 2023, doi: 10.1007/s13272-023-00666-x.
[35] M. D. Phung, C. H. Quach, T. H. Dinh, and Q. Ha, “Enhanced discrete particle swarm optimization path planning for UAV vision-based surface inspection,” *Autom. Constr.*, vol. 81, pp. 25–33, Sep. 2017, doi: 10.1016/j.autcon.2017.04.013.
[36] R. Zhang, X. Li, H. Ren, Y. Ding, Y. Meng, and Q. Xia, “UAV Flight Path Planning Based on Multi-Strategy Improved White Sharks Optimization,” *IEEE Access*, vol. 11, pp. 88462–88475, 2023, doi: 10.1109/ACCESS.2023.3304708.
[37] X. Bai, H. Jiang, J. Cui, K. Lu, P. Chen, and M. Zhang, “UAV Path Planning Based on Improved A $*$ and DWA Algorithms,” *International Journal of Aerospace Engineering*, vol. 2021, pp. 1–12, Sep. 2021, doi: 10.1155/2021/4511252.
[38] A. Souto, R. Alfaia, E. Cardoso, J. Araújo, and C. Francês, “UAV Path Planning Optimization Strategy: Considerations of Urban Morphology, Microclimate, and Energy Efficiency Using Q-Learning Algorithm,” *Drones*, vol. 7, no. 2, p. 123, Feb. 2023, doi: 10.3390/drones7020123.
[39] A. Israr, Z. A. Ali, E. H. Alkhammash, and J. J. Jussila, “Optimization Methods Applied to Motion Planning of Unmanned Aerial Vehicles: A Review,” *Drones*, vol. 6, no. 5, p. 126, May 2022, doi: 10.3390/drones6050126.
[40] C. Zhang, Y. Zou, F. Wang, E. Del Rey Castillo, J. Dimyadi, and L. Chen, “Towards fully automated unmanned aerial vehicle-enabled bridge inspection: Where are we at?,” *Construction and Building Materials*, vol. 347, p. 128543, Sep. 2022, doi: 10.1016/j.conbuildmat.2022.128543.
[41] G. Gugan and A. Haque, “Path Planning for Autonomous Drones: Challenges and Future Directions,” *Drones*, vol. 7, no. 3, p. 169, Feb. 2023, doi: 10.3390/drones7030169.
[42] M. Clerc and J. Kennedy, “The particle swarm - explosion, stability, and convergence in a multidimensional complex space,” *IEEE Trans. Evol. Computat.*, vol. 6, no. 1, pp. 58–73, Feb. 2002, doi: 10.1109/4235.985692.
[43] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, “Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation,” in *Computer Vision – ECCV 2018*, vol. 11211, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., in Lecture Notes in Computer Science, vol. 11211. , Cham: Springer International Publishing, 2018, pp. 833–851. doi: 10.1007/978-3-030-01234-2_49.
[44] “ImageNet.” [Online]. Available: http://www.image-net.org

[45] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 770–778. doi: 10.1109/CVPR.2016.90.
[46] A. Ellenberg, A. Kontsos, F. Moon, and I. Bartoli, “Bridge related damage quantification using unmanned aerial vehicle imagery,” *Struct. Control. Health Monit.*, vol. 23, no. 9, pp. 1168–1179, Sep. 2016, doi: 10.1002/stc.1831.
[47] Y. Cha, W. Choi, and O. Büyüköztürk, “Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks,” *Computer aided Civil Eng*, vol. 32, no. 5, pp. 361–378, May 2017, doi: 10.1111/mice.12263.
[48] A. Mohan and S. Poobal, “Crack detection using image processing: A critical review and analysis,” *Alexandria Engineering Journal*, vol. 57, no. 2, pp. 787–798, Jun. 2018, doi: 10.1016/j.aej.2017.01.020.
[49] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.
[50] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, vol. 9351, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds., in Lecture Notes in Computer Science, vol. 9351. , Cham: Springer International Publishing, 2015, pp. 234–241. doi: 10.1007/978-3-319-24574-4_28.
[51] V. Badrinarayanan, A. Kendall, and R. Cipolla, “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, Dec. 2017, doi: 10.1109/TPAMI.2016.2644615.
[52] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” 2014, *arXiv*. doi: 10.48550/ARXIV.1412.6980.
[53] P. Jaccard, “THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE.[1],” *New Phytologist*, vol. 11, no. 2, pp. 37–50, Feb. 1912, doi: 10.1111/j.1469-8137.1912.tb05611.x.
[54] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA: IEEE, Jun. 2015, pp. 3431–3440. doi: 10.1109/CVPR.2015.7298965.
[55] S. Wang, J. Wan, S. Zhang, and Y. Du, “Automatic Detection Method for Concrete Spalling and Exposed Steel Bars in Reinforced Concrete Structures Based on Machine Vision,” *Buildings*, vol. 14, no. 6, p. 1580, May 2024, doi: 10.3390/buildings14061580.
[56] M. Amran, G. Murali, N. Makul, M. Kurpińska, and M. L. Nehdi, “Fire-induced spalling of ultra-high performance concrete: A systematic critical review,” *Construction and Building Materials*, vol. 373, p. 130869, Apr. 2023, doi: 10.1016/j.conbuildmat.2023.130869.