RESEARCH REPORT DOCUMENTATION PAGE

1. Report No.	2. Report Date	3. Contract No.	4. Project No.
UND 21-01	May 2024	91210512	

5. Title and Subtitle	6. Report Type	7. Project No.
Smart Inspection of Ancillary Structure in North Dakota Using Unmanned Aerial Systems.	Work Plan ☐ Construction ☐ Evaluation ☐ Final ☒	8. Project No. 9. Project No. 10. Project No.

11. Author(s)/Principal Investigator(s)
Dr. Sattar Dorafshan

12. Performing Organization Name and Address	13. Sponsoring Agency Name and Address
NDDOT M+R ☐ NDDOT OTHER* ☐ NDSU ☐ UND ☒ UGPTI ☐ OTHER* ☐ *See supplementary notes North Dakota DOT Materials and Research Division 300 Airport Road Bismarck ND 58504-6005	North Dakota DOT Materials and Research Division 300 Airport Road Bismarck ND 58504-6005

14. Supplementary Notes

15. Abstract

Objective

The research team proposes a UAS payload designed for effective and accurate inspection of ancillary structures. The proposed prototype will be equipped with several sensors; and will allow inspectors to perform both visual and NDE inspections. AI models will be used for defect detection, and data fusion in real-time. The prototype will have novel features that do not currently exist for highway structures inspections and asset management.

Scope

This project aims to develop a UAS-based system for autonomous defect detection in highway ancillary structures using AI and sensor technology. The work will be carried out in six tasks over a span of two years. Initially, a literature review will inform the selection of AI models, sensors, and equipment. The team will then integrate miniature sensors, design a lightweight payload, and develop software for data collection. AI models for defect detection will be trained using a comprehensive database of sensor data and images. The prototype system will be tested in controlled and real-world environments, with field inspections validating its performance. Finally, a comprehensive report will summarize the project findings, including the design and validation of the system.

Summary

This report details the development of an Unmanned Aerial System (UAS) equipped with AI for real-time, autonomous defect detection in highway ancillary structures. The system uses deep learning models to detect common defects such as corrosion, cracks, and missing bolts. The UAS payload includes visual and thermal cameras, a microcomputer for processing, and a ground station laptop with a Graphical User Interface (GUI) for inspectors to collect and analyze data. Models trained on annotated datasets achieved over 90% accuracy. While the system performed well in tests, a limitation was identified in slow processing time during real-time inspections.

16. Key Words	17. Distribution Statement	18. No. of Pages
Ancillary Structure, Structural Inspection, Automated Inspection, Crack Detection, Corrosion Detection, Artificial Intelligences, Unmanned Aerial Systems, Deep Learning, Machine Learning	No restrictions. This document is available to the public from: North Dakota Department of Transportation Materials and Research Division: 300 Airport Road Bismarck ND 58504-6005 Office: (701) 328-6900 Fax: (701) 328-0310	91 19. File type/Size Pdf/ 6.1 mb

SMART INSPECTION OF ANCILLARY STRUCTURES IN NORTH DAKOTA USING UNMANNED AERIAL SYSTEMS

North Dakota Department of Transportation

Final Report

Prepared for:

North Dakota Department of Transportation

Prepared by:

Amrita Das (Graduate Research Assistant)

Faezeh Jafari (Graduate Student)

Dalton Reitz (Research Engineer)

Dr. Sattar Dorafshan (Principal Investigator)

Department of Civil Engineering

University of North Dakota,

Grand Forks, North Dakota, USA.

Jack Heichel (Graduate Student)

Rajrup Mitra (Graduate Research Assistant)

Dr. Naima Kaabouch (Co Investigator)

School of Electrical Engineering and Computer Science

University of North Dakota,

Grand Forks, North Dakota, USA.

May 2024

Acknowledgement

The research project is chiefly funded by the North Dakota Department of Transportation. The authors thank the North Dakota Department of Transportation Materials and Research, especially T. J Murphy and Nancy Huether for their invaluable assistance and co ntinuous flow of communication and information.

The authors would like to extend their gratitude to Bruce Dockter, Senior Lecturer at the department of Civil Engineering of University of North Dakota for helping in experimental procedures carried out in this project. The authors are grateful to Zachary Stangl, former aviation student, Scott Keane, UAS chief Pilot, and Zachary Reeder, Project Manager, Research Institute for Autonomous System at University of North Dakota for data collection at different phases of the project

Disclaimer

The contents of this report reflect the work of the authors, who are responsible for the results and the accuracy of the information presented.

Table of Contents

List of Tables

List of Figures

Executive Summary

Periodic inspection of highway ancillary structures plays a vital role in maintaining uninterrupted highway operation. Utilizing small Unmanned Aerial Systems (UAS) technology allows for ancillary structure inspections to become faster and cheaper, providing a benefit to state agencies and the public in the state of North Dakota. However, the existing available UAS technology and UAS payload does not offer real-time autonomous defect detection using artificial intelligence (AI) updated with inspector input. This report describes the design and functionality of a payload equipped UAS that can provide real-time inspection of ancillary structures with a developed built-in AI model interface. The models are developed using deep learning models to autonomously detect common defects in ancillary structures (corrosion, missing bolts, and cracks), to assist inspectors for more robust condition assessment. The developed payload system includes a microcomputer capable of running multiple Convolutional Neural Network (CNN) models during flight. The research team developed a set of annotated datasets for each type of defect investigated in this project. AlexNet-integrated models for corrosion and crack detection were trained on 9257 and 1500 images, respectively. The models label tiles of each image if corrosion or crack is detected. Faster RCNN was trained on 1000 images for defective bolted connection that are common in ancillary structures. The trained R-CNN automatically puts a bonding box around the defected area in the bolted connection images. All models reached over 90% accuracy in training and validation. A Graphical User Interface (GUI) is developed to interact with the payload through a laptop. The inspector can run the GUI to collect visual or thermal images, classify defects, accept or reject the defects, re-train the models based on new annotated data, and store final defect detection results. The payload consist of both bota and thermal sensing to capture images and live stream, relaying the data to the ground station laptop through a shared Wi-Fi network. A live stream of the visual and thermal sensors allows the operator to quickly assess the structure and determine which regions need further evaluation. The payload consists of Ground Station Laptop with GUI and repository, Portable Wi-Fi router, Microcomputer Board, Visual camera, Thermal camera, Housing and mounting equipment, Gimbal. The payload functions were tested and verified in realistic environments. The payload performed well during the test but was found to have a limitation of slow processing time.

1 Introduction

Non-bridge structures such as overhead sign structures, high mast light poles, and traffic signal mast arms are referred to as ancillary structures on highways. Regular inspection of these structures is important. To ensure structural integrity, all responsible authorities require annual inspections of anchor bolts, joints, and base plates. Negligence to do so over the time can reduce the service life and, in many cases, cause the structure to fail. Among these, corrosion, which reduces the structure's service life, is the destructive attack of a metal by chemical or electrochemical reaction with its environment. Though some ancillary steel structures may be painted, protection is most often provided using galvanizing or fabrication using weathering steel. Unfortunately, environmental corrosion cannot be generalized in terms of sources. Several factors such as exposure time to the corrosive environment, atmospheric pollution level might control corrosion at micro level. The authors of [1-2] revealed that starting with the rough texture at the surface corrosion can propagate inside the structure. This not only increases the maintenance cost due to the continuous reactivity with the surrounding electrolyte but also responsible for 42% of failure condition of infrastructures [3]. The other defects that need to be taken care of are the fatigue crack and defective bolts. High ancillary structures are subjected to dynamic loads such as wind gusts, truck induced gust etc. This cyclic loading can introduce fatigue stress which may be started earlier than the yield stress at static loading. In addition to this, defective welding can initiate fatigue cracks by acting as a weak joint. However, joint damage can happen due to loosening or missing bolts too. The fastener can be loosened and started to contribute to failure of the joint. Though the initiation of failure of an anchor rod or bolt in a structural connection may seem apparent, even secondary fasteners that fail can lead to sign breakage and small items falling into traffic.

The cause of a joint failure is not only cyclic loads, but bolted joint failure can also occur due to environmental causes [4]. For instance, higher temperature may reduce the load carrying capacity of the bolt and thus lead the structure to be unstable. Because of losing structural integrity, catastrophic accidents may take place. Thus, effective monitoring and diagnosis of the bolt connections are necessary to ensure that structures are safe and reliable.

In the United States, periodic inspections of in-service structures such as bridges, dams etc. cost significant amount of money to be conducted by human labor [5]. Satisfying the maintenance and protection of traffic safety requirements while controlling costs within the acceptable limits could be challenging in the current practice. There is a need to establish a safe, repeatable, and cost-effective methodology to inspect ancillary structures in North Dakota. Autonomous defect detection mechanism integrated with unmanned aerial vehicles (UAVs) can be used as a safe and inexpensive measure to introduce revolutionary refinement in this arena. The MDOT (2019) has shown 60% cost savings associated with drone-based inspections; moreover, a report from the American Association of State Highway and Transportation Officials (AASHTO 2018) has proclaimed that 35 of 44 reporting state DOTs with previous experience are deploying aerial platforms in some capacity.

2 Inspection Methodology

2.1 Current Practice for Inspection

There are approximately 1000 different types of state-owned ancillary structures (Table 1) on the different highways of North Dakota. Visual Inspection is the currently practiced method for defect assessment, which is time-consuming for vast areas, impossible for inaccessible areas and subjective to the inspector. In addition to this, gaining access to the structure for inspection personnel is one of the most difficult challenges for the inspection and evaluation of overhead sign structures. Inspection challenges arise from the need to satisfy Maintenance and Protection of Traffic (MOT) safety requirements while controlling costs within acceptable limits. Such access strategies include night work, mobile lane closures, and other innovative methods for short-term lane closures. Moreover, FHWA recommends to be equipped with enough auxiliary equipment to perform any kind of structural inspection [6].

Table 1: Number of ancillary structures at North Dakota

Type	**Number (approximate)**
Traffic Signals	450
Overhead Signs	175
High Mast Lights	375

Sign structure inspection can be a hazardous structural inspection. Per guidelines, the sign structures are often located at the 'gore' or exit areas of high-speed roads where work zone safety setups could be extremely difficult to set up [6]. In consequence, it is routine that the inspector 'climbs' the structure, which is complicated due to angled diagonals and slippery structural members. Vehicle mounted bucket (Figure 1) is the most typical way to access the sign structure. A 30 ft boom is sufficient for inspection. But most of the time these vehicles should be rented from cable and telephone companies. Because of traffic in the morning, some work needs to be postponed to nighttime. If inspections are planned to be done at night, adequate lighting must be provided to avoid hazards.

Figure 1: Inspection with (a) Bucket truck and (b) Climbing [2]

For external corrosion and bolt missing/loosening detection, visual inspection is the most reliable method. But for crack and internal corrosion detection, non-Destructive Testing (NDT) is an important tool used for inspection of ancillary structures. Examples include small fatigue cracks in welds, corrosion occurring on the interior of the structural element, and cracked anchor rods. Usually, a dye penetrant test and magnetic particle test are performed to detect surface cracks. For the internally propagated cracks, eddy current is used. Ultrasonic thickness 'D meter' measurements can help to confirm the internal corrosion which may be externally invisible.

Figure 2: Thickness Measurement at Critical Location of Structure Post [2]

Current inspection techniques can be challenging to the transportation agencies due to the requirement of preplanning for the lane closure [7]. Moreover, onsite documentation is another challenge as lack of coordination may lead the inspector to revisit the site once again. In addition to this, manual inspection result may subject to the inspector's perception which might be inconsistent.

2.2 Autonomous defect detection method

Different image-based algorithms have been used by the researchers to identify the corrosion in steel structures. Conventional image processing methods such as image registration by the binary information, k-mean clustering, color space changing etc. were used by the researchers [8-10, 1, 11] in detecting the corroded pixels from the images. On the other hand, a model has been developed by Lee et al. [12] to identify the defective pixel from the variation of statistical parameters such as mean, mode, median etc. Some researchers stepped forward by using different deep learning models such as Faster RCNN [13], ResNet 50 [13], VGG16 [14] for detecting corrosion in steel structures. Fatigue cracks in steel structure is a challenging problem to mitigate [15]. Unfortunately, fatigue cracks in steel did not get the attention of many researchers as concrete cracks due to absence of realistic data for AI model development. In the past, nondestructive methodology such as attaching piezoelectric material [16] with drone used to determine the crack in concrete bridge.

On the other hand, a crack detection algorithm using canny edge detector has been developed by the authors of [17, 18] which detected cracks with less than 0.15 mm error to ground truth. The problem with the conventional image-based algorithm is that it needs

user input threshold value which may change depending on the quality of the dataset. Dorafshan et al. [19] also revealed that the deep learning models such as AlexNet outperformed different types of edge detector for detecting concrete cracks. A deep convolution neural network has been developed by the authors of [20] to detect and localize crack in concrete bridges from images. Again, the authors of [12] developed an automated process using deep learning models for detecting, localizing, and mapping of five different types of defects in concrete bridges and reported overall 85.3% accuracy. The application of different artificial intelligence models was not limited to corrosion and crack detection in the infrastructures. The authors of [19] used image based deep learning algorithm such as Hough transformation to estimate the looseness of the bolted joints. On the other hand, the authors [20] developed a deep learning model with images collected from the real steel infrastructure showing different modes of rotation of loosened bolts. However, the authors of [21] generated a deep learning convolution neural network from the signal collected from the existing nondestructive method such as ultrasonic wave propagation.

2.3 Real time defect detection using UAS

The use of UAS for detection of structural defects is not novel, but very few solutions utilize an approach that provides AI defect detection support in real time. Experimental research using UAS for detecting delamination, corrosion, cracks on photovoltaics (PV) modules used in power plants was carried out by the authors of [22]. A thermal camera mounted on UAS (PLP-610) was used to collect the images and processed them on the ground using different image-based algorithms.

Eschmann et al. [23] implemented an eight-rotor unmanned aerial vehicle (UAV) with a payload equipped with different sensors such as gyroscopes, accelerometers, and a barometric altitude sensor to do the aerial survey as a part of regular inspections of the buildings. In addition, Chen et al. [13] used a high-definition camera mounted on six-axis UAV platform with some intelligent features such as obstacle avoidance, positioning, and stable hover to detect corrosion in large steel structures.

A novel approach is introduced in [24], wherein a UAS solution is proposed that can mount onto a wall, climb along its flat surface, and identify cracks. While this method would not be practical on ancillary structures which are usually not flat, the paper discusses real-time

use of deep learning models to support the identification of cracks. The crack detection models were proven successful at the short range, as lighting was not a significant factor.

An method, that is entirely autonomous, to inspect bridges for common defects is proposed and partially demonstrated in [25]. The autonomous flight was scaled to a limited, controlled environment. AI Models for the defect types of steel corrosion, steel crack, and loosened bolt were demonstrated to run simultaneously with high accuracy.

2.4 Choice of UAV

The choice of a UAV to use in an imaging application depends on several factors. The payload weight should be estimated based on the intended application. The UAV to carry the payload should have a payload capacity greater than the expected payload weight, with adequate margin (at least 20% margin). The margin is to account for any other weight that may get added to the payload as the design process progresses. Another factor that may limit the choice of UAV is the adaptability of the flight control system. Some drone manufacturers may protect the Intellectual Property (IP) of the UAV and flight control system, which requires the control of the payload system to be separate from that of the UAV. This adds to the weight and complexity of the entire system. The UAV's stability system should be evaluated to determine if the sensor payload can be equipped onto it. Many UAVs include a gimbal system that maintains a stable camera angle, even as the drone changes its positioning. Utilizing the UAV's built-in gimbal system for the payload provides a stable mounting to ensure no motion blur in images. Table 2 provides a comparison of different UAVs in terms of the number of rotors, payload weight, diameter, and maximum flight time.

Table 2: UAV Comparison

Name	Rotors	Payload	Diameter	Max Flight Time*
DJI Mavic Air 2	4	300g	30.2 cm	34 min
DJI Phantom 4 Pro	4	500g	35 cm	30 min
Yuneec Typhoon H	6	1800g	52 cm	25 min
DJI Matrice 600 Pro	6	6000g	113.3 cm	32 min
Freefly Alta 6	6	6800g	112.6 cm	45 min

2.5 Wireless Communication

The sensor data must be sent in real-time to the ground control station (GCS) at a high speed due to the high number and size of frames. To achieve this, a long-range and high-speed communication system is needed for the UAV to communicate with the GCS.

\In many UAV systems, real-time data is transmitted from the UAV to the GCS with a live feed of the camera equipped on the UAV. The payload designer for such project should consider using the existing communication system between the GCS and the UAV, which can reduce the weight and power consumption. Also, the range of the communication network will remain consistent.
In the review of communication systems used by similar UAV imaging applications, two techniques are primarily used. The first technique is to have the sensor data sent on the same network as used by the Flight Control System. The second technique is to implement a communication network separate from the Flight Control System. This has no risk of slowing the flight controls. However, this technique likely adds to the weight and power consumption of the UAV payload. Both disadvantages reduce the flight time of the UAV. As previously mentioned, reduced flight time may result in longer mission time as the UAV may need to be charged.

2.6 Smart Graphical User Interface

The mission control and data displaying system form the user-interface level in the UAV system, wherein the mission commands and instructions are conveyed to the UAV and data captured using the payload sensors are processed, analyzed, and displayed at the user end in form of a GUI based application. The display system serves as a visual interface between the UAV and the GCS staff. In the realm of autonomous systems, smart GUI-based user interfaces have long played an important role.

There are some research efforts aiming to provide ways to connect UAVs and the cloud infrastructure forming a smarter way of interfacing between the UAV and the ground control. Lin et al. [25] put forward a solution of integrating the cloud service of Google Earth with the UAV. This was done using transmission of data to a MySQL database using

an android phone. The user used a web browser to access the database's UAV information. UAVs were controled using a specific flight plan defined in the database via a waypoint. Similarly, a multirobot control and communication architecture with a user interface were developed by the authors of [26].

2.7 Cameras

The cameras deployed on the payload were selected to satisfy three key requirements. First, they must be lightweight so they could be used on a variety of UAV platforms. Second, there must be visual and thermal cameras in the payload to augment detection. Live streaming with both cameras allow for real-time imaging to assist the inspector in defect detection. The third requirement is that the sensors must provide high resolution images to the AI models. Any unwanted blur can affect the ability of the AI models to identify defects. Also, the live streaming is best with high quality images to ensure the inspector knows the areas of interest to run the defect detection models.

The selected visual camera is an Arducam HQ, with a resolution of 12.3 MP. The resolution was dropped to 1980 * 1080 pixels to improve the processing time of the neural networks running in real time. The sensor alone has a low focal point with no lens, meaning the sensor alone would only focus on a close object. Also, the aperture is wide, meaning that the depth of field it can focus on would be very limited. To improve focal length and depth of field, an external lens is added to the Arducam to ensure the camera can focus on many objects simultaneously at a moderate distance. This lens adds weight and complexity to the design but the resulting camera is lighter and less expensive than other drone-mounted cameras, with similar image quality.

The selected thermal camera is a longwave infrared (LWIR) thermal camera. This camera uses a 12 μm pitch Vanadium Oxide (VOx) uncooled detector capable of capturing 640 * 512 pixels. For the live streaming of the sensors, GStreamer was used. It is an open-source pipeline-based multimedia framework.

2.8 Data Transfer

To transmit a large amount of data with a high throughput, an IEEE 802.11ac Wi-Fi network is established to connect the Jetson and ground control station. The Wi-Fi is established by a router transmitting at the 5 GHz (433 Mbps) band. Secure Shell (SSH)

protocol is used for cryptography of the data to prevent spoofing. The payload uses a static IP address, so Wi-Fi password protection should be implemented as needed for securing the system. The Transmission Control Protocol (TCP) Server-Client architecture was implemented for the robust transfer of data from payload to the GCS. The payload's Jetson microcomputer acts as the server, reserving a port number to listen for commands. The GCS acts as the client, sending various commands to Jetson. When the computer sends a command to Jetson, the Jetson runs the appropriate processing, which is often a still image capture, stream request, or AI model(s) to be run on the latest image.

3 Graphical User Interface

The control of the payload is handled by a Graphical User Interface (GUI) run on a Ground Station laptop. This GUI is written in C# language, developed in Microsoft Visual Studio 2022. The GUI is compiled as an executable so that most computers can run it easily. A user guide is provided as part of the GUI to assist the inspector in using each function of the GUI with the payload, as well as technical details on the back-end processing.
The GUI has several functions:

1. Command the payload to do the following:
    a. Capture still images from visual or thermal cameras
    b. Start live stream from visual or thermal cameras
    c. Run AI models on the most recent image
2. Crop images for faster processing of areas of interest
3. Show the location of defects with bounding boxes (as determined by AI models)
4. Allow inspector to modify bounding boxes
5. Store images with embedded bounding boxes in a repository
6. Provide the details of background processes to the inspector

Below image shows the main screen of the GUI and the sections that comprise it.

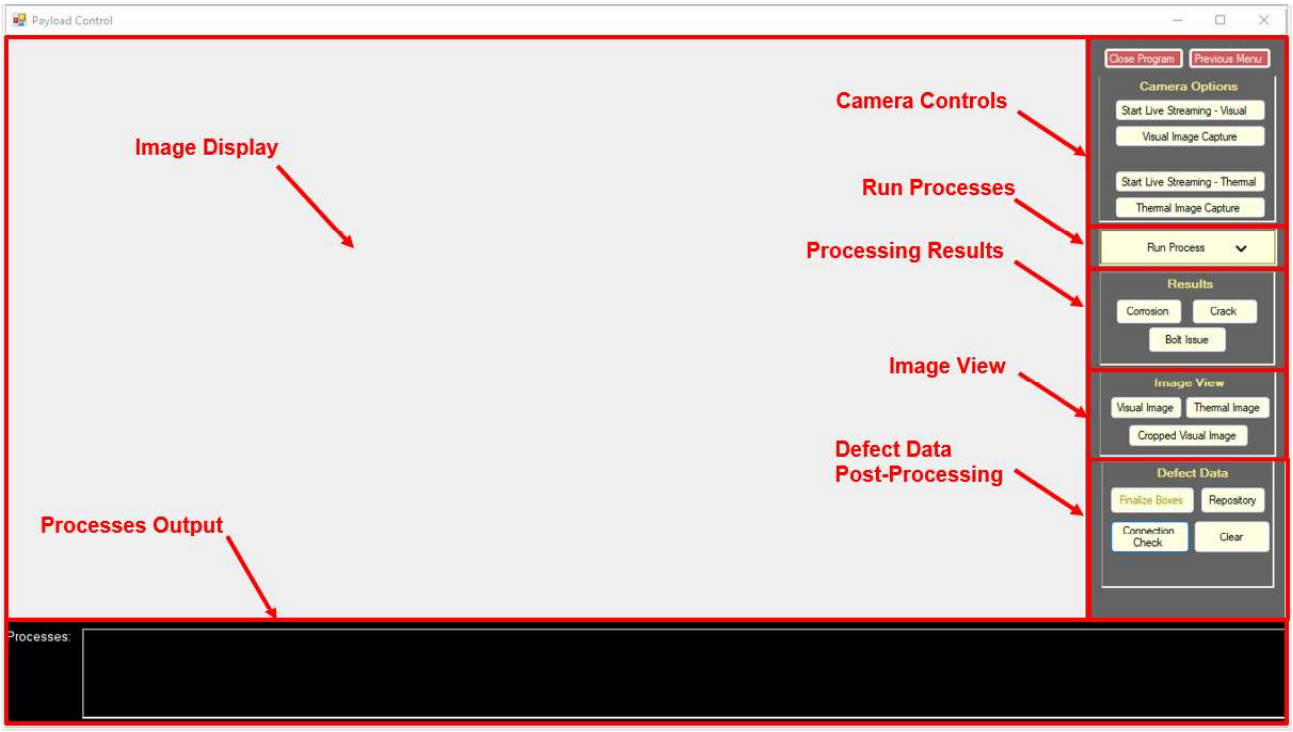


Figure 3: GUI Main Screen

GUI Sections and Descriptions:

- Image Display
  - Displays still images
  - Allows for modification of defect locations after defect detection processes are run
- Processes Output
  - Describes the steps and output of back-end processing to assist inspector
- Camera Controls
  - Starts visual or thermal live streaming
  - Captures visual or thermal still images
- Run Processes
  - Initiates defect detection processes
  - The number of defect types can be selected
- Processing Results
  - Switches between results of defect processes (if multiple are run)
- Image View
  - Switches between different image types (often used before processes are run). The image types include Visual, Thermal, and Cropped Visual.
- Defect Data Post-Processing
  - Finalizes boxes from defect processes before storage to repository
  - Stores image(s) to repository
  - Trains defect processing models to improve accuracy

The cropping function allows for a smaller area of interest within an image to go through the selected AI model(s). This will run through the models faster than the whole image, while also providing a result in the repository focused on the defect alone.

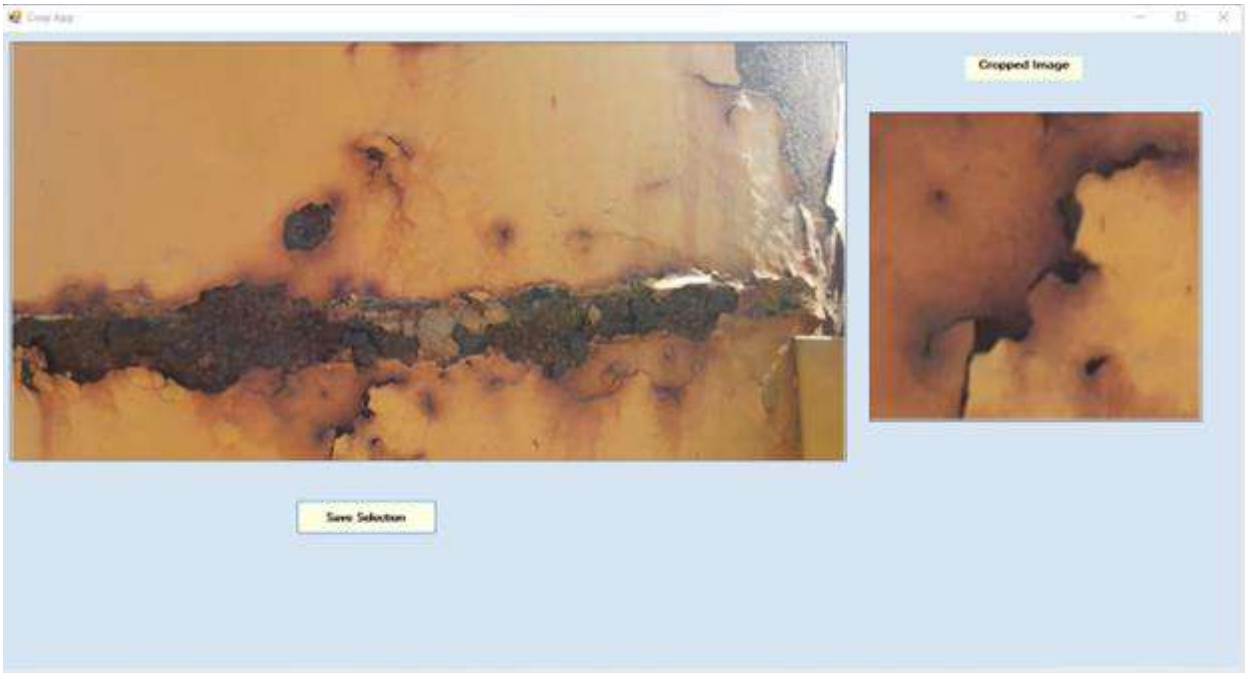


Figure 4: GUI Crop Function

Figure 5 shows the repository interface of the GUI. The bounding boxes that were selected by the inspector are embedded in the image. The repository functionality allows for quick and easy storage of the captured images during an imaging mission. This is crucial so that the inspector has an archive of all the areas that were deemed as potentially containing a defect. During a mission, the inspector can quickly capture an image of a target area with potential defect locations, store it in a repository, then move to the next location. After the mission, the inspector can review the repository to determine the next maintenance steps to take.

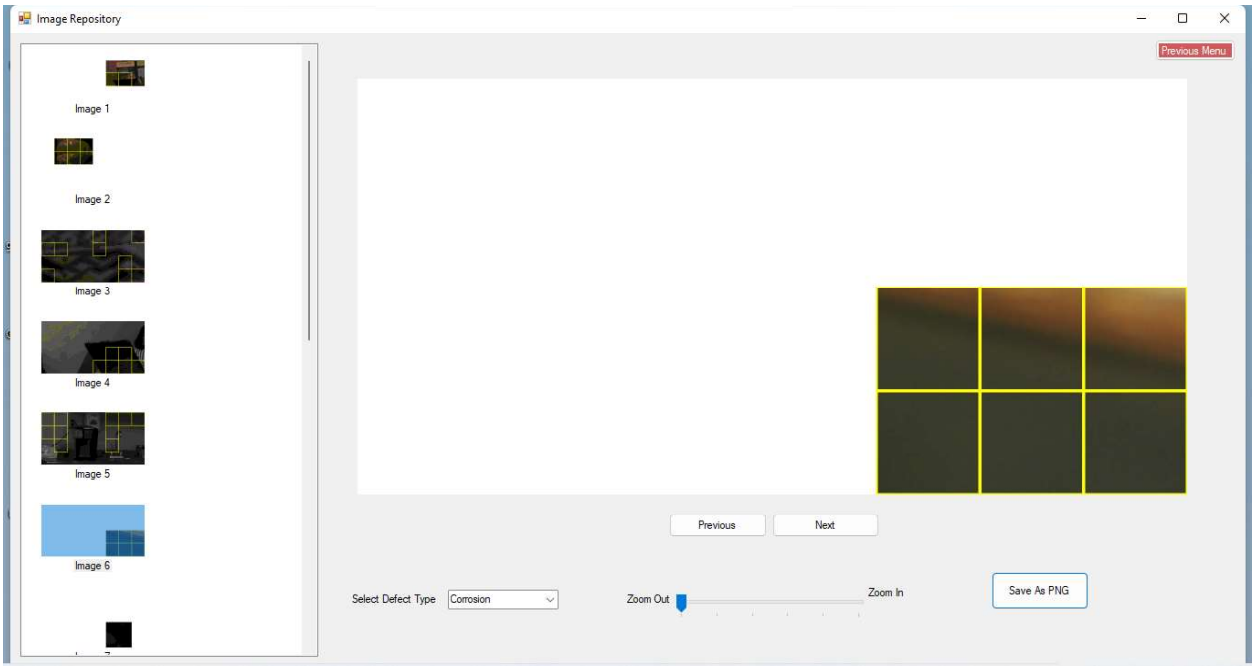


Figure 5: GUI Repository

3.1.1 Interactive Defect Identification on GUI

The Interactive GUI functionality works the same for both Corrosion and Crack defects. The only difference is that the Corrosion defects are identified by blue boxes, while Crack defects are identified by red boxes. When the Defect Detection process for Corrosion and/or Crack are run on an image, the Interactive GUI divides the image into an 8 x 4 grid of sub-images to prepare for processing. When the processing is complete, the Interactive GUI places a box around each sub-image that contains the defect(s). The box is not centered around the defect, meaning the defect could be anywhere within the box. If the inspector disagrees with the identified defect locations, they can change the selected boxes before storing the image in the repository. They can de-select a sub-image that they believe does not contain a defect, and the box will be removed. Also, they can select a sub-image that they believe does contain a defect that was not identified by the process. This process of de-selecting or selecting is as simple as clicking on the location within the GUI.

The Interactive GUI for Bolt Issues works differently than Crack and Corrosion. When the Defect Detection process for Bolt Issues is run on the current image, the Interactive GUI identifies the exact location of each Bolt Issue, without having to divide the image into sub-images. Yellow bounding boxes are drawn around each identified defect, with numbers listed on the top left corner of each box. The size of the boxes varies, depending on the

sizes of the defective areas. When the processing is complete, the bounding boxes appear on the screen, superimposed on the image. The inspector has the option to remove any of the bounding boxes placed by the model. Then, the inspector can click and drag anywhere within the displayed image to place a new bounding box, if they determine a bolt defect is in that location. The boxes can be any size and location, as long as it is within the image. The bounding boxes changed by the operator are saved when toggling between different image views. This allows the operator to evaluate multiple defect types within one image and modify multiple sets of defect locations before saving to the repository.

3.1.2 Back-end GUI Processes

There were many challenges in designing the GUI to include user-friendly features. Many of these were due to limitations with C# as a programing language. For one, the main C# form needs the images from the payload to be displayed with bounding boxes overlaid on top to identify defects. To accomplish this with the most optimal processing speed, the image is redrawn into the Image Display every time a bounding box is removed. To elaborate further, the bounding boxes shown in the Image Display area are not separate elements of the GUI, but rather static shapes drawn onto the image. This method was deemed to be less intensive on the PC's memory and processing power than alternative methods of either resaving the image every time or creating new GUI elements that are placed in front of the image.

There are many files within the folder structure of the GUI that are python or batch files to support the GUI. The GUI calls these files multiple times during normal operation. These files are mainly used for interaction with the payload or for image modification. The reason that these are external to the GUI is due to limitations of the C# language for inter-agent communication and image processing.

To ensure the inspector interaction with the boxes is saved when switching from one defect type to another, different approaches are taken for different defects. For Crack and Corrosion, an array named "boxarray" defines the box locations. This array is 3 dimensional, with the first two dimensions corresponding to the x and y coordinates of the box, respectively. The 3rd dimension is simply used to switch between Crack and Corrosion. To preserve the bounding boxes used for Bolt defects, the box locations and size is saved in a text file in the project folder.

3.2 AI model Architecture

3.2.1 Corrosion detection model

The development of an Artificial Intelligence model for detecting corrosion required several steps. Overall data generation, data augmentation, data annotation, model training, and testing are the basic phases of the model development. AlexNet, the robust image classification model, was used for detecting corrosion. The architecture of AlexNet has been proposed by Krizhevsky [29] which has 8 main layers. There are 25 sublayers in these 8 main layers which are trainable. This is a pre-trained deep convolution neural network on imageNet dataset. Figure 6 depicts the backbone of the AlexNet in terms of five convolution layers(C1-C5), seven layers (ReLU1-ReLU7) with rectified linear unit(ReLU) to solve the issue with non-linearity, two normalization layers (Norm1-Norm2), three fully connectedlayers ((FC1-FC3). Among these three fully connected layers the last layer has been modified so that the model can work as binary classifier with the help of sigmoid function at the last layer.

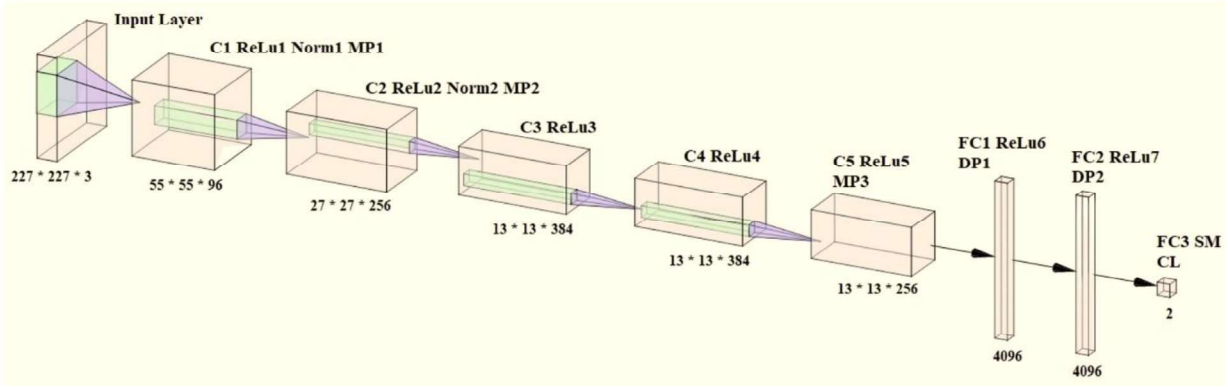


Figure 6: The architecture of AlexNet [29]

Two types of sensors have been used for data generation. The Specification of both the sensors are mentioned in Table 3. The data generation process is completed in two phases; mobile phone has been used in the first phase and UAS camera later.

Table 3: Camera Comparison

Type of the device	**Samsung Galaxy M30**	**UAS camera**
Resolution	13 Megapixels	7860 x 4320 Megapixels
Aperture size	f-stop; f/1.9.	f-stop; f/2.8 - f/11

Using images from different sensors in model development helped to test and rectify the dependence of the model on the sensor's quality. Approximately 300 images from the existing signal posts with an average height of 6.7m(22ft) and a cantilever arm length of 6.1-6.7m(20-22ft) have been collected by the research team by using the mobile phone from 9.53 am to 11.53 am on May 17, 2021. The sizes of the images were 2311 pixels X 4128 pixels [30]. In the second phase, approximately 200 images are collected by UAS camera from the in-service ancillary structures.

Data augmentation library such as albumentation from python has been applied on the most representative images with corrosion to augment the data set. The model development starts with 4000 images. At present, it is augmented to 9257 images after testing model performances for different images (Figure 7).

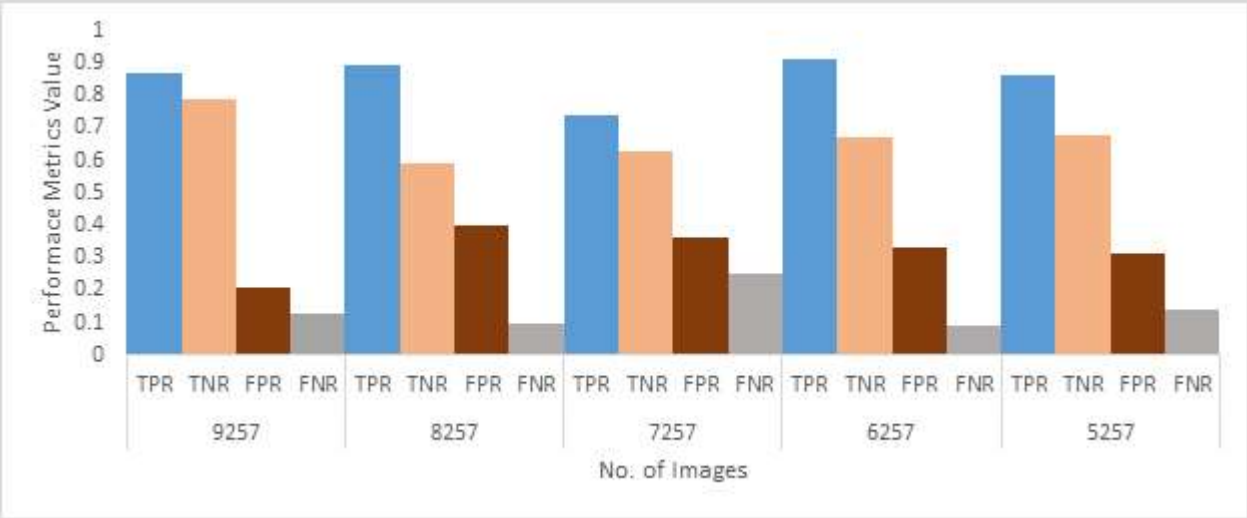


Figure 7: Performance evaluation of model by varying the number of images

TPR (true positive rate) represents the percentage of the correctly detected corrosion images. Similarly, TNR (true negative rate) depicts how many non-corroded images have been detected accurately. Higher value of both TPR and TNR value work as indication of less false detection. All these parameters were determined considering the training labels as ground truth.

After data augmentation all the images have been labelled with corrosion (WC) and without corrosion (WOC) by the research team. The total number of images with corrosion is 4600

and 4657 for without corrosion. To confirm the robustness of the model the without corrosion images consists of diverse background such as cars, sky, trees including the images from the sound part of the steel structures.
The model has been trained with 9254 images and tested with the images collected by the research team from the internet and the combined dataset comprises images of different sensors.

3.2.2 Crack detection model

The dataset utilized in this study for AlexNet consisted of 250 images sourced from prior research studies [31,32], in addition to an extra 30 images obtained from real-world ancillary structures. This dataset was assembled to encompass images from various sections of the structures for a comprehensive examination of cracks. It was split into two annotated sets based on the types of detection performed by deep convolutional neural networks (DCNN). The AlexNet annotated dataset comprised 200 images displaying fatigue cracks and 250 sub-images without cracks, all with dimensions of 256 by 256 pixels. To augment the training dataset, various data enhancement methods were applied, including adjustments to color, brightness, and crack orientation, resulting in an expanded dataset of 1400 sub-images. Furthermore, realistic images of fatigue cracks were overlaid onto images of undamaged and in-service ancillary structures, boosting the dataset's size to 1500 sub-images. To enhance subsets related to cracks, a combination of random under-sampling and data augmentation was used.
To create a more diverse training dataset, some images of ancillary structures, often coated in silver or blue anticorrosion paint alongside red, had their colors modified to silver or blue. Additionally, since corrosion is a frequent occurrence in steel structures, the colors of select images were altered to mimic corroded plates or galvanized steel plates. This color transformation was carried out using methods outlined in Reference [33], wherein the color of galvanized steel plates, both corroded and intact, served as the target color. Images from in-field structures were considered input objects, and the process described in Reference [34] was employed to adjust the color of raw images to match the color of images in the target dataset, thus enlarging the training set. Figure. 8 illustrates the original images, target images, and the outcomes of the color transformation algorithm.

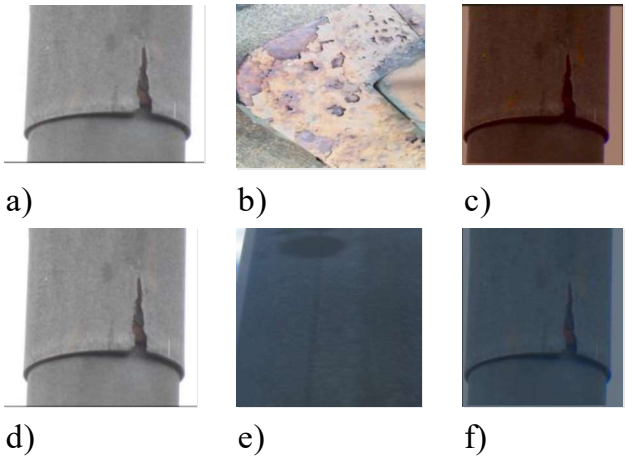


Figure 8: Data augmentation.

a) raw images, b) raw image with corrosion c) fused image, d) raw images, e) galvanized steel, f) fused images

Acquiring images of structures with cracks can be challenging as they are often promptly repaired to prevent structural risks. To address this, a limited number of images depicting ancillary structures with and without fatigue cracks were blended into a color algorithm. This multi-faceted data augmentation approach was employed to generate lifelike images of ancillary structures with fatigue cracks, further expanding the dataset to 1500 sub-images. Figure 9 illustrates the overlaying of images with and without cracks to create authentic images of ancillary structures with fatigue cracks [33].

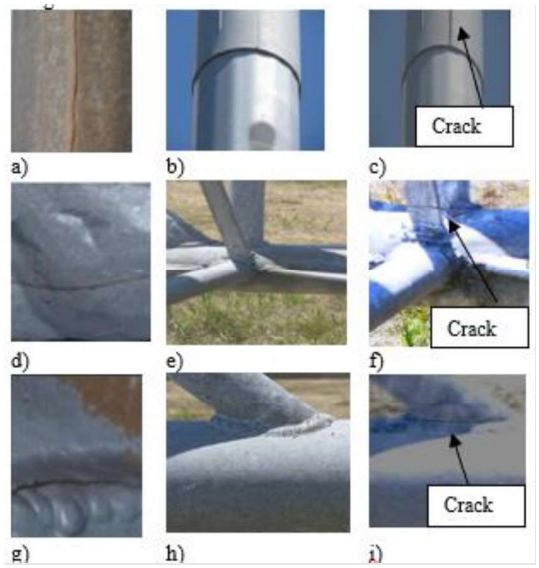


*Figure 9: Examples of u*sing *a superimposed approach to augment data, a) Original image with a crack, b) Original image without crack, c) a superimposed crack, d) raw image from Reference*

[32], e) Original image without crack, f) superimposed crack g) Original image from Reference [34], h) Original image without crack, i) a superimposed crack at structure's [31].

Rotation was employed as a data augmentation technique to boost the size of the crack dataset, a method previously demonstrated to be effective in research. Images were uniformly rotated at 45° and 135° angles, as depicted in Figure 10, to augment the training set.

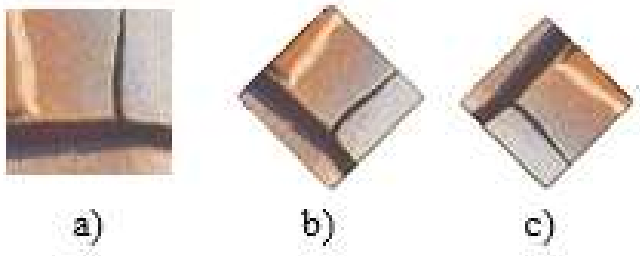


Figure 10: Data augmentation, a) raw image, b) rotated by 45 compared to original image, c) rotated by 90 compared to original image [1].

It is the most challenging part of crack detection to create a dataset based on limited images from steel structures and run AlexNet as a deep learning algorithm in real time to detect cracks as defects. The accuracy rate of the model can be improved by adding more data from real structures since only two ancillary structures with fatigue crack (HM 0029-138.442 and HM 0029-137.911) exhibited during inspection of this project. We have already taken images of these two structures and have added them to the main datasets. However, the number of images from real structures should increase to have better defect detections. The testing images captured by the drone 's camera was used to test algorithms. These images were divided into 1,100 smaller segments. Subsequently, trained Alex Net, also referred to as the Crack Model, was applied to categorize these images into two distinct categories (sub images with cracks and sub images without cracks).

The results generated by AlexNet models were recorded in separate Excel files for crack and corrosion, organized according to image names and their corresponding labels.

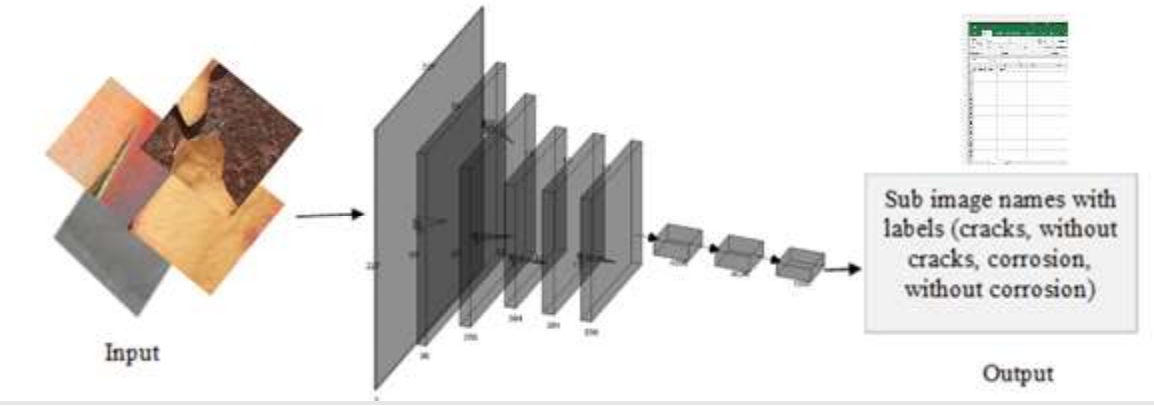


Figure 11: AlexNet architecture in paper [5].

3.2.3 Bolt Detection Method

The dataset comprises approximately 1,000 images depicting absent bolts or nuts in ancillary structures within laboratory settings across diverse locations, including North Dakota and Baltimore in the USA, as well as Istanbul in Turkey.

Faster R-CNN (FRCNN) is a variant of Region-Based Convolutional Neural Networks (RCNN), belonging to the family of machine learning models designed for object detection in images. The RCNN approach involved producing a series of bounding boxes as output, each encapsulating instances of missing bolts as objects. The RCNN model was specifically trained to identify and localize defective bolts within these bounding boxes during testing. Implementation of the model was carried out using Python, where the defected bolts were placed within the bounding boxes, and predictions for bolt or nut defects in the test set were made.

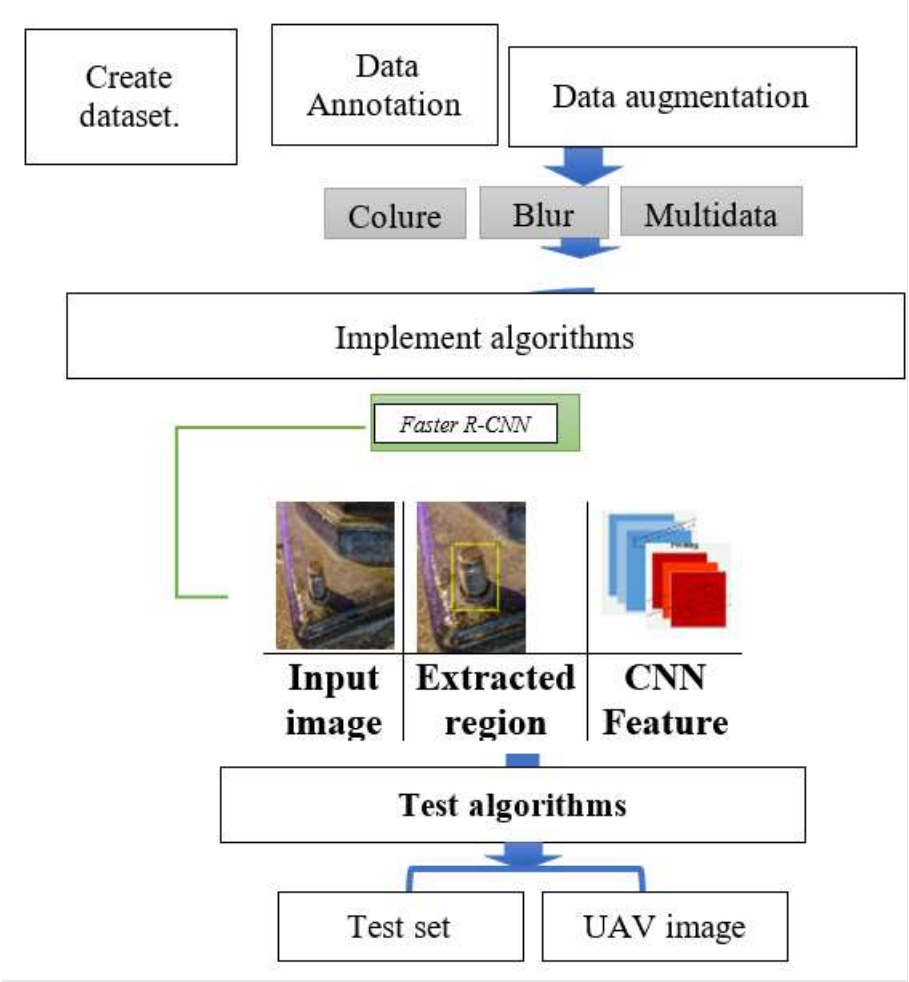


Figure 12: Bolt Detection Method

3.3 Data Annotation

Bounding boxes were used to annotate missing bolt area. Figure 13 displays an example of annotated missing bolts with bounding box (shown a rectangular shape)

Figure 13: Structure with a missing bolt, shown by bonding box annotation

3.4 Data Augmentation

Our objective was to enhance the resilience of object detection algorithms through the implementation of data augmentation techniques. Various augmentation methods, including noise addition, color adjustments, blurring, and a diverse set of augmentations were employed to augment the dataset's image sizes. This approach aimed not only to improve the model's ability to detect objects but also to diversify the dataset by introducing variations through augmentation techniques.

Figure 14 shows the performance for some images in the testing set. This image is trickier compared to other image since the color of bolt is black and different from trainset (Figure14c), blur image (Figure14b), hidden by structure (Figure14f), and not visible enough (Figure 14b).

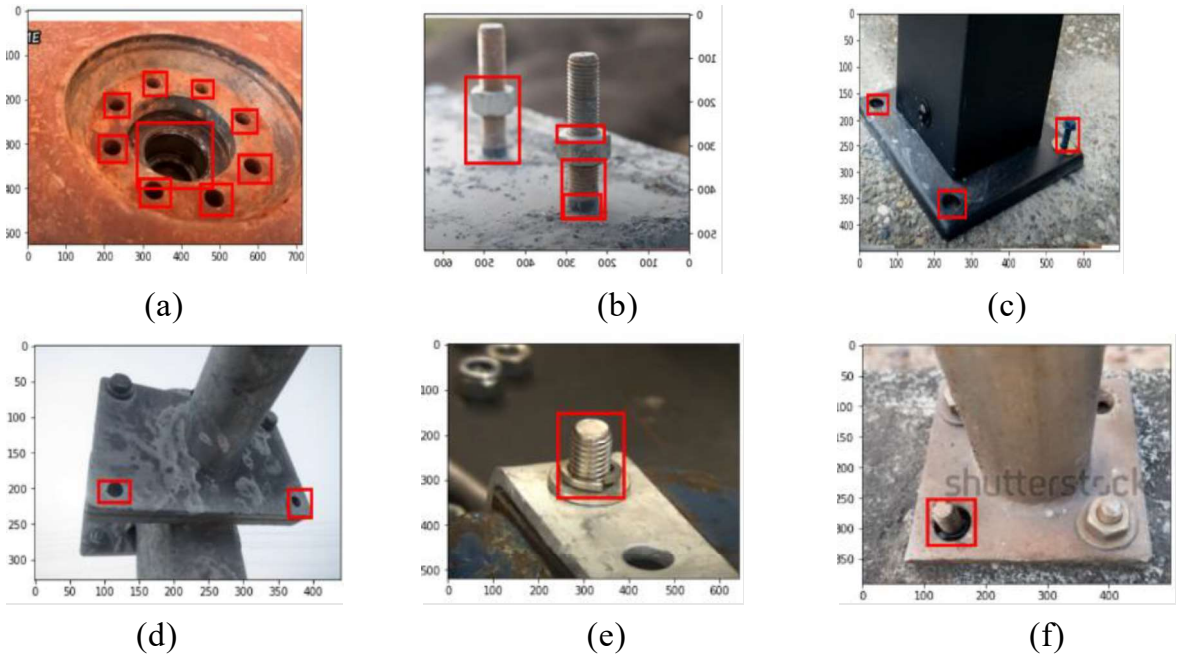


Figure 14: FRCNN result. a) multiple missing bolt output, b) multiple loosen bolt output, c) black loosen bolt, d) missing bolt, e) loosen nut, f) loosened nut.

4 Payload System Design

The System Block Diagram is shown in Figure 15. The left side of the diagram is for the hardware/software located on the ground, while the right side is for the hardware/software located in the air, on the UAV.

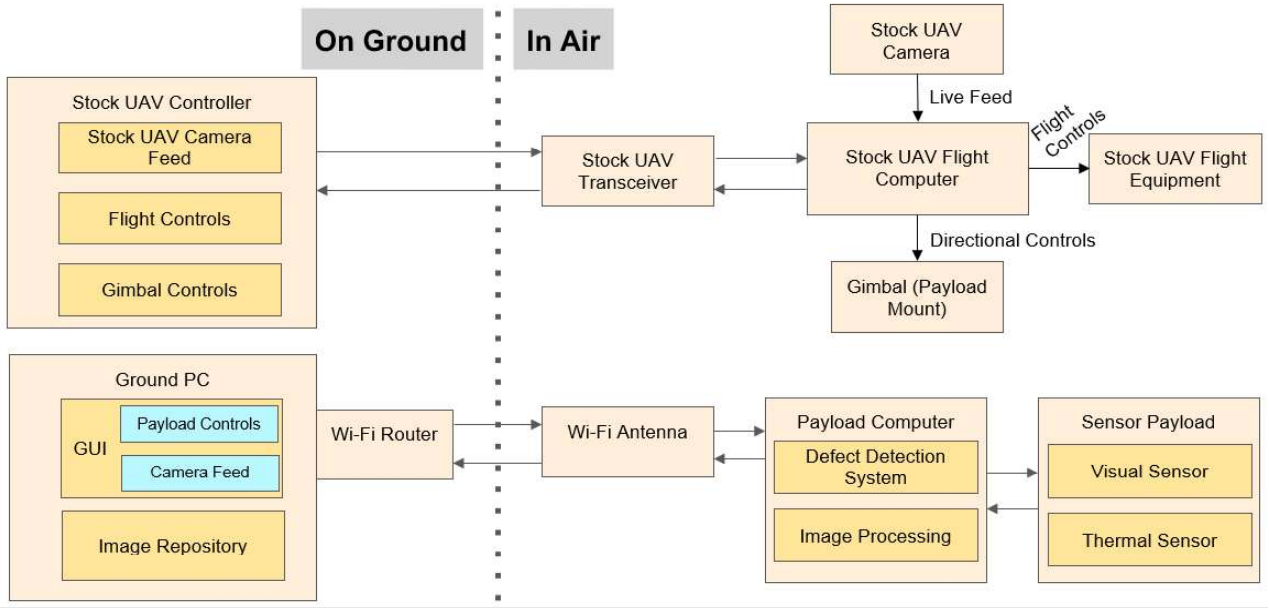


Figure 15: System Block Diagram

The payload described in this document works separately from the stock UAV hardware and software. The only interaction is the control of the gimbal that holds the payload should be commanded by the UAV flight controller. The method for accomplishing this varies, depending on the UAV controller, but usually is a simple setting in the software provided by the flight controller. Alternatively, the gimbal can be configured to be controlled by the Ground PC, using QGroundControl software on the laptop establishing a link to the gimbal through the Jetson running MavProxy software. In this case, the Jetson receives gimbal commands from the laptop and transfers the commands to the gimbal.

The payload's computer (Jetson Xavier NX) runs the AI models to detect defects within the captured images on command. The data and commands between the payload computer and Ground PC are transmitted over a Wi-Fi network established by a travel router (IEEE 802.11ac).

A Ground PC is used to run the GUI and store images in a repository. The Dell Latitude 5430 was selected, as it is rugged to survive any moisture or physical stress from operating outdoors. The chosen PC's 16GB RAM and i5 processor has been able to effectively run the GUI with minimal latency or crashing.

4.1 Payload Equipment

Table 4 below lists the equipment used in the payload and the Ground Station. The features of the equipment as well as the purpose that each piece fulfills is included.

Table 4: Payload Equipment

Type	Name	Features	Purpose
Laptop	Dell Latitude 5430	• Semi-rugged with enhanced battery life • Intel Core i5 1145G7 /2.6 GHz • 1 TB SSD NVMe Class 40 • 16GB, 2x8GB, 3200 MHz DDR4 RAM	• It runs the Smart Defection Detection Interface • The router is powered using an USB connected to the laptop • Its hosts the AI training processes for defect detections
Microcomputer	Nvidia Jetson NX	• CUDA-enabled parallel computing capability • 384 NVIDIA CUDA® Cores, 48 Tensor Cores, 6 Carmel ARM CPUs • Delivers up to 14 TOPs for AI applications in 10W power utilization	• Its host the AI test processes for defect detections • The two sensors are controlled using this system-on-module • Establishes a connectivity with the laptop when both in same network
Visual Camera with Lens	Arducam 477P HQ Camera Board Lens: Arducam C-Mount Lens	• Optical Format: 1/2.3″ • Maximum still resolution: 4056 × 3040 • 30fps@Full 12.3MP • Supports NVIDIA Argus Camera plugin for H264 encoding, JPEG snapshots • C-Mount Lens: • 16mm Focal Length • Manual Focus and Aperture Adjustment, F1.4 to F16	• Provides live visual camera stream • Captures visual image for AI processing • Lens wide aperture allows for very deep depth of field
Thermal Sensor	Teledyne FLIR Boson	• Resolution: 640x512 • 12 µm pixel pitch VOx microbolometer • Temperature rating: -40 °C to +80 °C • Low power consumption around 500 mW	• Provides live thermal camera stream • Captures thermal image

		• For rugged construction	
Gimbal	Gremsy Mio	• Carries payload up to 400g • Lightweight	Allows stabilization of the sensors to prevent motion blur
Router	GL.iNet GL-AR750	• AC VPN Travel Router • 300Mbps(2.4GHz) + 433Mbps(5GHz) Wi-Fi	Establishes a cryptographic Wi-Fi network connection between the microcomputer and the laptop
Battery	HRB 4S Lipo Battery		It is needed for powering up the microcomputer and the gimbal during flight
Battery Charger	Hobby Fans B6 Balance Charger		For charging the batteries in balanced charged mode for short-circuit, overcharge, overcurrent and overheat protection.

4.2 Power Consumption

Table 5 lists the power consumption of the electronic devices located on the payload. Jetson power consumption was captured as the worst-case during operation.

Table 5: Power Consumption

Component	Max Power Consumption (W)	Voltage Range
Jetson	12	9-19
Visual Camera	1	(from USB)
Thermal Camera	0.5	(from USB)
Gimbal	8.4	14-52
Total	21.9	

The payload voltage ranges allowed for either a 14.8V or 18.5V LiPo battery. For reference, each LiPo battery cell operates at 3.7V, so adding LiPo cells in series allows for voltage values at multiples of 3.7V. Several batteries were assessed based on weight, capacity, and voltage. Table 6 lists these parameters, and a ranking was provided based on the most ideal parameters. Ultimately, a low weight battery was selected that will still allow for 121 minutes of operation. This time is most often beyond the amount of time that most UAVs can fly, so this operating time is acceptable.

Table 6: Battery Comparison

Voltage	Capacity (mAh)	Minutes of Operation	Weight (g)	Price	Final Rank
14.8	3000	121.6	297	$32	1
18.5	4000	202.7	495	$54	2
14.8	4000	162.2	403	$45	3
14.8	5200	210.8	470	$42	4
14.8	3300	133.8	318	$36	5
14.8	5000	202.7	492	$54	6
18.5	5000	253.4	608	$72	7

4.3 Custom Modeled Payload Fastening System

To carry the required payload on the DJI Inspire 2 Drone, attachments needed to be fabricated to safely fasten the payload to the drone and not hinder operation. To do this, prototypes were designed and modeled on AutoCAD, imported into GrabCAD Print for formatting, then transferred via USB drive to the StrataSys F370 composite 3D printer to be printed using ABS M30 material. Details on the F370 and ABS M30 material properties are included in the appendix.

One of the prototypes housed the cameras. It was designed to be fully enclosed and to minimize camera movement and foreign material contamination while the drone is operating. A classic half-hinged clipping design was employed to be able to accomplish this. Additional attributes of the design include cord accessibility for both cameras while the box is still closed and multiple mounting locations to mount the box to the gimbal. Below are images of this design. Annotated drawings are included in Appendix A, Section 2.

Figure 16: Prototype Housing

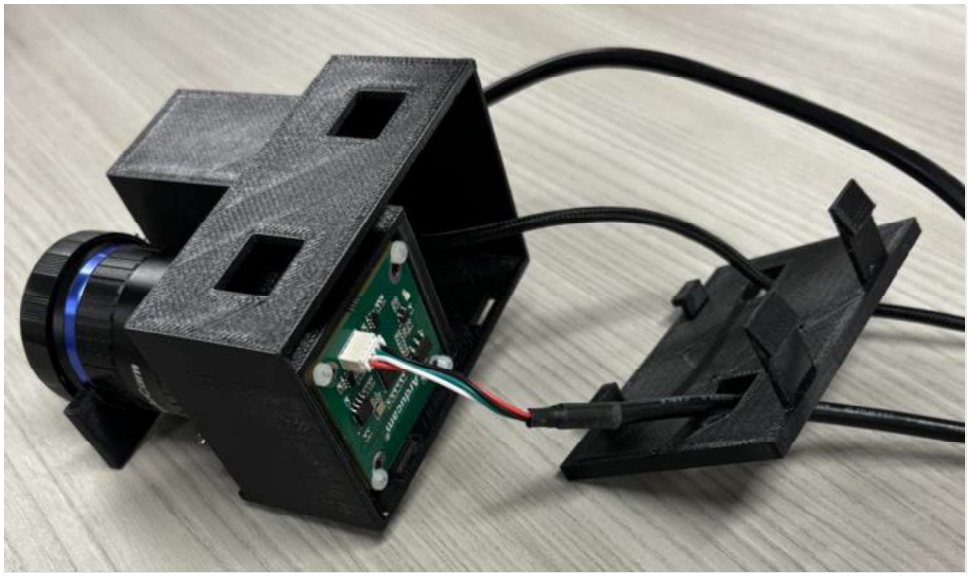


Figure 17: Prototype Housing Inside

Another prototype helped fasten the power supply and onboard computer. The goal of this design was to allow plenty of air flow so the power supply and computer would not overheat. Additionally, it was important to make sure this design did not interfere with any of the drone's collision detection and monitoring systems. Unfortunately, the detection system for under the drone is partially impeded by the design due to limitations in cable reach, overall drone stability (or affecting the Center of Gravity too severely), and the

awkward size and geometry of the Jetson and power supply. A three-piece mounting bracket was designed to be able to fasten the Jetson/Power supply housing to the drone. They were designed to maximize stability with the limited cross section area available due to the drone's design. Figure 18 is an image of the designs. Annotated drawings are included in Appendix A, Section 2.

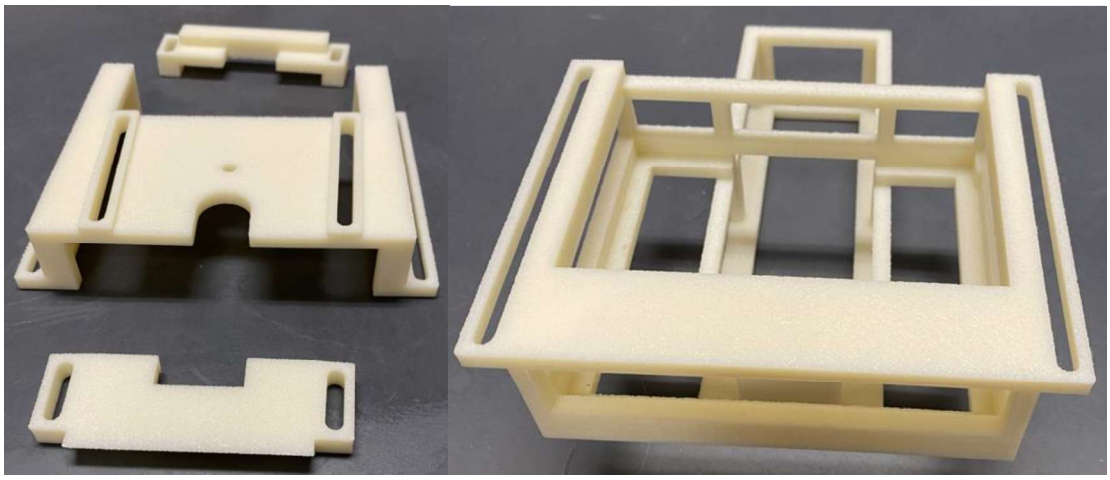


Figure 18: Gimbal Adapter 1

Finally, an adapter was also needed to be able to attach the gimbal to the drone since the gimbal used was not designed by DJI, thus preventing the proper attachment interface required. To accomplish this, 5 pieces were required. Two of them were the main gimbal housing that is meant to hold the gimbal in place on the drone while allowing free movement of the gimbal and access to each of the required ports on the gimbal. The other three pieces were designed to mount the gimbal and gimbal housing to the damper that is on the drone. The goal was to make this as stable as possible while still allowing functionality of the damper to provide further stabilization to the cameras while the drone is in flight. Below is an image of the pieces for the design. Annotated drawings are included in Appendix A, Section 2.

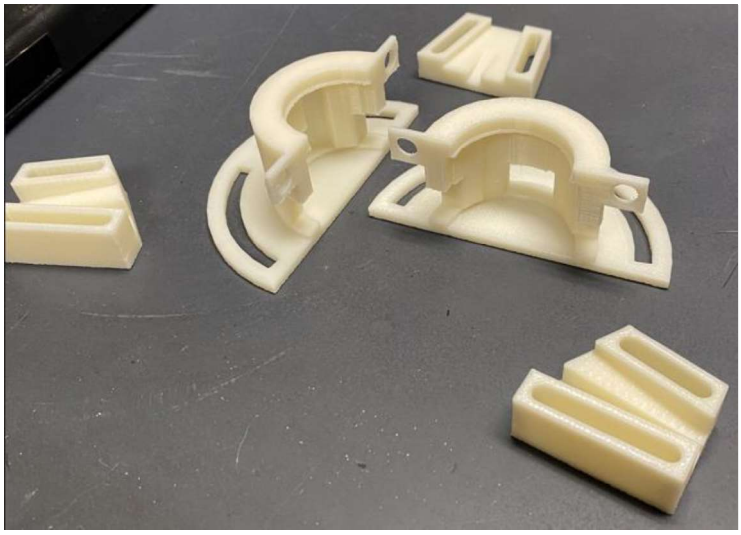


Figure 19: Gimbal Adapter 2

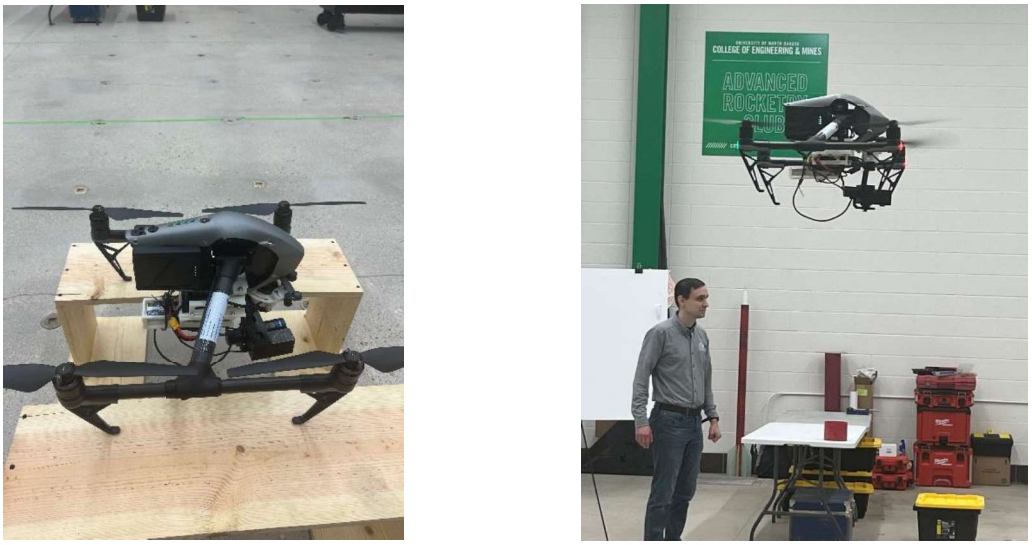
After inserting all components into their respective housing and attaching them to the drone with their respective brackets, the final full design is assembled (Figure 20).

Figure 20: UAV Integration

4.4 Validation

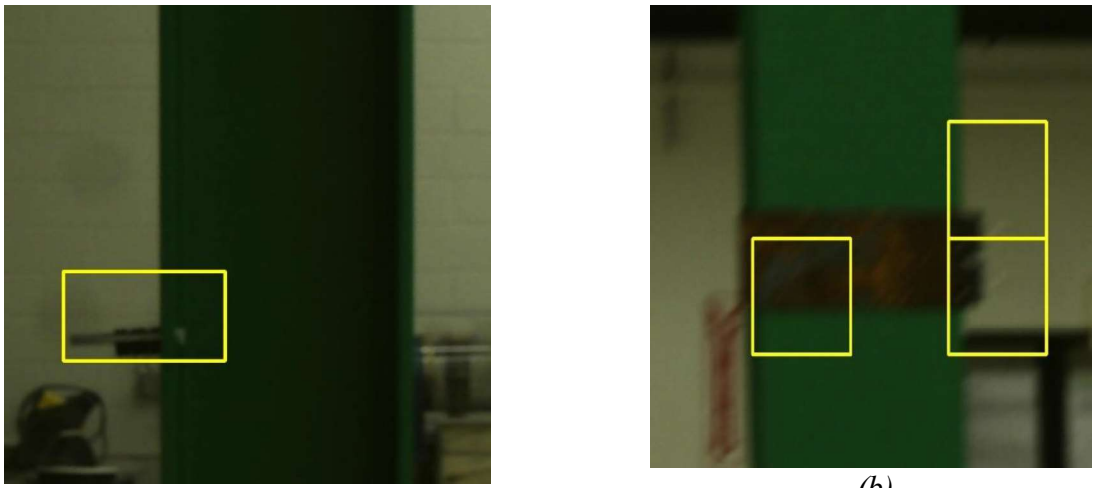
4.4.1 System Validation in Controlled Environment

A demo filed test has been simulated at the Civil Engineering High Bay facility, University of North Dakota. A corroded steel plate has been clamped on a column to replicate the scenario in the field. In addition to this, bolts were also attached to the column without nuts to simulate the defective bolts in the real steel structures. The lighting condition was not as natural as the sunlight, and this might lead the model to misdetection in case of corrosion. The inspection demo with the results is reported here in Figure 21-22. One the other hand, the low lighting condition made missing bolt detection challenging without use of AI model (Figure 22b).



(a) *(b)*

Figure 21 (a)& (b): Pre-flight condition checking at Lab



(a) *(b)*

Figure 22 (a)& (b): Loosen Bolt and Corrosion detection by the AI models

4.4.2 System Validation with Field Test

The outdoor defect detection mission with the payload mounted on UAS has been completed. In this section, an example of model performance in defect detection was reported for the corrosion model only, since the inspected pole did not have any fatigue crack or bolt issues. The detection results are reported in this section (Figures 23-27). Out of 32 split sub-images, 8 truly have corrosion. The corrosion model correctly detected 5 sub images as corroded but falsely indicated 3 images as uncorroded. The reason for this misdetection could be the ratio of corroded (approximately 20%) and background (approximately 80%). The inspector was able to correct the miss detections using GUI. This demonstrated the successful interactive functionality of the GUI, which allowed the inspector to correct the model detection output.

The time required for the processing functions of the system is mentioned in Table 7. The battery voltages were taken before and after the test. Because the voltage of a LiPo battery needs to be above 3.2V per cell, the operator must ensure that the voltage does not drop below that value. The payload ran at approximately full load for about 15 minutes and depleted 0.22v. Therefore, the payload depleted the battery at about 14.7mV per minute. Assuming a starting charge of 4.1v and an acceptable margin of 0.1v about 3.2v, the battery should be limited to only running the payload long enough to drop 0.8v. This equates to about 54 minutes, but due to battery depletion over time and adding more margin, this limit should be reduced to **45 minutes**. Because it takes setup time before and after the test, the time that the payload runs should be tracked carefully to ensure it doesn't run longer than 45 minutes.

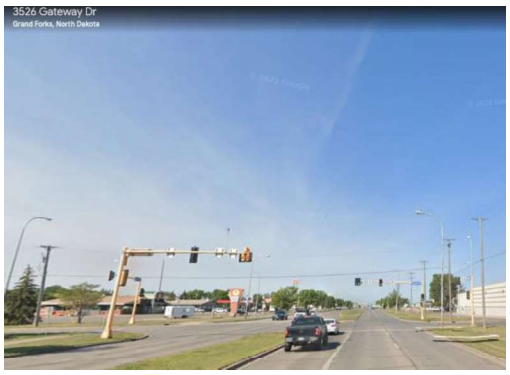


Figure 23:Inspection Location, 3526 Gateway Drive, Grand Forks, ND

(a)

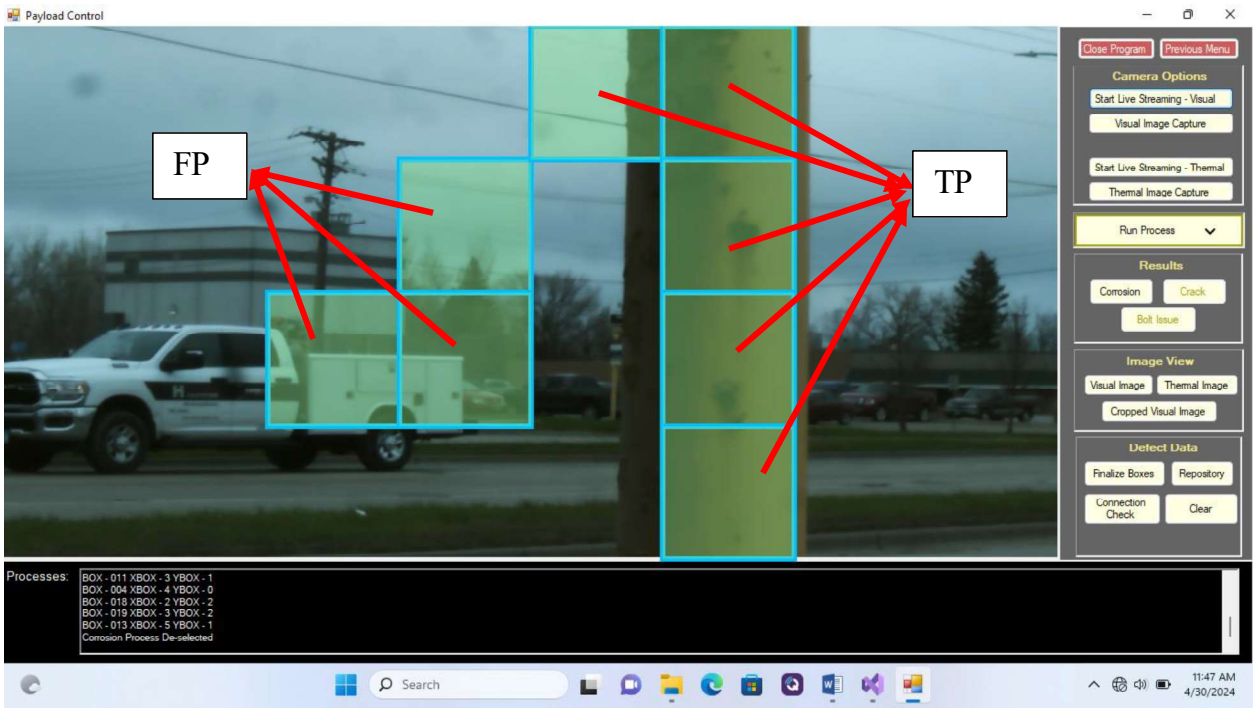
(b)

Figure 24: (a) &(b) Pre-flight condition checking

(c)

(d)

Figure 25: (c) &(d) Inspection team during the outdoor flight



(a)

		Predicted	
		TRUE	FALSE
Actual	TRUE	5	3
	FALSE	3	21

(b)

Figure 26: (a) Corrosion detection results (b) confusion matrix

(a) (b)

Figure 27: Saved image in repository after completing the inspection (a) Corrosion (b) Crack & defective bolt

Table 7: Flight Datasheet

Mission Phase	Parameter	Measurement
Pre-Flight	Payload Battery Voltage	4.09 v per cell
Flight	Visual Image Capture Process Time	9 second (Approximate)
Flight	Corrosion Process Time	23.846 seconds
Flight	Crack Process Time	26.185 seconds
Flight	Bolt Process Time	31.749 seconds
Flight	All process	83 seconds
Post-Flight	Payload Battery Voltage	3.87 v per cell

5. Limitation and future work

Although the developed payload-equipped UAS improves on the current inspection system, it has some limitations. The UAS can be operated only when the environmental conditions such as position of cloud, presence of wind condition etc. satisfy the FAA recommendation. This requirement may hinder the inspection in North Dakota where the environmental conditions change abruptly.

All the AI models are developed with enough datasets. However, the datasets were not diversified as many defective poles were already replaced or over-coated. For example, the corrosion model was developed with the images collected from Grand Forks and Fargo. Most of the inspected poles painted yellow in color. So, 90% of images with corrosion are with yellow color structure. Training of the model on the specific color may be one of causes of misdetection. However, the provision of retraining will give the opportunity to update the model to be more robust with new annotated data from the field.

The processing times reported for the AI models were longer than expected. This is mostly due to the remote processing of these models occurring at the microcomputer. This could be mitigated by the operator limiting the number of areas of interest on which models are run for inspection. Also, the operator may consider moving to a new area of interest while the previous area is being evaluated by the AI models. The live streaming shown on the GUI can occur while the models are running. Another limitation of the payload is that the housing takes time to assemble and disassemble. A more robust design may be considered

if these times are an issue for the inspection. Future work could improve the processing time by re-architecting the system to run the models on the laptop. If this would occur, a laptop with greater processing power would be more optimal.

References

1 Khayatazad, M., De Pue, L., & De Waele, W. (2020). Detection of corrosion on steel structures using automated image processing. *Developments in the Built Environment*, *3*, 100022

2 Garlich, M. J., & Thorkildsen, E. T. (2005). *Guidelines for the installation, inspection, maintenance and repair of structural supports for highway signs, luminaires, and traffic signals* (No. FHWA-NHI-05-036). United States. Federal Highway Administration.

3 Hoang, N. D. (2020). Image processing-based pitting corrosion detection using metaheuristic optimized multilevel image thresholding and machine-learning approaches. *Mathematical Problems in Engineering*, *2020*.

4 Tran, D. Q., Kim, J. W., Tola, K. D., Kim, W., & Park, S. (2020). Artificial intelligence-based bolt loosening diagnosis using deep learning algorithms for laser ultrasonic wave propagation data. *Sensors*, *20*(18), 5329.

5 Hoskere, V., Narazaki, Y., Hoang, T., & Spencer Jr, B. (2018). Vision-based structural inspection using multiscale deep convolutional neural networks. *arXiv preprint arXiv:1805.01055*.

6 Wells, J., & Lovelace, B. (2018). *Improving the quality of bridge inspections using unmanned aircraft systems (UAS)* (No. MN/RC 2018-26).

7 Li, Y., Kontsos, A., & Bartoli, I. (2019). Automated rust-defect detection of a steel bridge using aerial multispectral imagery. *Journal of Infrastructure Systems*, *25*(2), 04019014.

8 Bondada, V., Pratihar, D. K., & Kumar, C. S. (2018). Detection and quantitative assessment of corrosion on pipelines through image analysis. *Procedia Computer Science*, *133*, 804-811.

9 Hoang, N. D. (2020). Image processing-based pitting corrosion detection using metaheuristic optimized multilevel image thresholding and machine-learning approaches. *Mathematical Problems in Engineering*, *2020*.

10 Naik, D. L., Sajid, H. U., Kiran, R., & Chen, G. (2020). Detection of Corrosion-Indicating Oxidation Product Colors in Steel Bridges under Varying Illuminations, Shadows, and Wetting Conditions. *Metals*, *10*(11), 1439.

11 Lee, S., Chang, L. M., & Skibniewski, M. (2006). Automated recognition of surface defects using digital color image processing. *Automation in Construction*, *15*(4), 540-549.

12 Lin, J. J., Ibrahim, A., Sarwade, S., & Golparvar-Fard, M. (2021). Bridge Inspection with Aerial Robots: Automating the Entire Pipeline of Visual Data Capture, 3D Mapping, Defect Detection, Analysis, and Reporting. *Journal of Computing in Civil Engineering*, *35*(2), 04020064.

13 Chen, Q., Wen, X., Lu, S., & Sun, D. (2019, August). Corrosion Detection for Large Steel Structure base on UAV Integrated with Image Processing System. In *IOP Conference Series: Materials Science and Engineering* (Vol. 608, No. 1, p. 012020). IOP Publishing.

14 Na, W. S., & Baek, J. (2017). Impedance-based non-destructive testing method combined with unmanned aerial vehicle for structural health monitoring of civil infrastructures. *Applied Sciences*, *7*(1), 15.

15 Jahanshahi, M. R., Masri, S. F., Padgett, C. W., & Sukhatme, G. S. (2013). An innovative methodology for detection and quantification of cracks through incorporation of depth perception. *Machine vision and applications*, *24*(2), 227-241.

16 Lim, R. S., La, H. M., & Sheng, W. (2014). A robotic crack inspection and mapping system for bridge deck maintenance. *IEEE Transactions on Automation Science and Engineering*, *11*(2), 367-378.

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

18 Sakagami, T. (2015). Remote nondestructive evaluation technique using infrared thermography for fatigue cracks in steel bridges. *Fatigue & Fracture of Engineering Materials & Structures*, *38*(7), 755-779.
19 Cha, Young-Jin, Wooram Choi, and Oral Büyüköztürk. "Deep learning-based crack damage detection using convolutional neural networks." *Computer-Aided Civil and Infrastructure Engineering* 32, no. 5 (2017): 361-378.
20 Huynh, T. C., Nguyen, B. P., Pradhan, A. M. S., Pham, Q. Q., Nguyen, N. T., Nguyen, M. N., ... & Bui, T. Q. (2021). Vision-based inspection of bolted joints: Field evaluation on a historical truss bridge in Vietnam. *Evaluation*, *55*, 77.
21 Tran, D. Q., Kim, J. W., Tola, K. D., Kim, W., & Park, S. (2020). Artificial intelligence-based bolt loosening diagnosis using deep learning algorithms for laser ultrasonic wave propagation data. *Sensors*, *20*(18), 5329.
22 Aghaei, M., Grimaccia, F., Gonano, C. A., & Leva, S. (2015). Innovative automated control system for PV fields inspection and remote control. *IEEE Transactions on Industrial Electronics*, *62*(11), 7287-7296.
23 Eschmann, C., Kuo, C. M., Kuo, C. H., & Boller, C. (2012). Unmanned aircraft systems for remote building inspection and monitoring.
24 Jiang, S., Zhang, J (2019). Real-time crack assessment using deep neural networks with wall-climbing unmanned aerial system. *Computer-Aided Civil and Infrastructure Engineering*.
25 Ali, R., Kang, D., Sug, G., Cha, Y (2021). Real-time multiple damage mapping using autonomous UAV and deep faster region-based neural networks for GPS-denied structures. *Automation in Construction Volume 130.*
26 Horstrand, P., Guerra, R., Rodríguez, A., Díaz, M., López, S., & López, J. F. (2019). A UAV platform based on a hyperspectral sensor for image capturing and on-board processing. *IEEE Access*, *7*, 66919-66938.
27 Lin, C. E., Li, C. R., & Lai, Y. H. (2012, September). UAS cloud surveillance system. In *2012 41st International Conference on Parallel Processing Workshops* (pp. 173-178). IEEE.
28 Balakirsky, S., Carpin, S., Kleiner, A., Lewis, M., Visser, A., Wang, J., & Ziparo, V. A. (2007). Towards heterogeneous robot teams for disaster mitigation: Results and performance metrics from robocup rescue. *Journal of Field Robotics*, *24*(11-12), 943-967.
29 Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems* 25 (2012).
30 Das, Amrita, Eberechi Ichi, and Sattar Dorafshan. "Image-Based Corrosion Detection in Ancillary Structures." *Infrastructures* 8, no. 4 (2023): 66.
31 Jafari, F., Dorafshan, S., & Kaabouch, N. (2023, June). Segmentation of fatigue cracks in ancillary steel structures using deep learning convolutional neural networks. In 2023 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM) (pp. 872-877). IEEE.
32 IPC-SHM The 1st International Project Competition for Structural Health Monitoring (IPC-SHM 2020). 2020. Available online: http://www.schm.org.cn/#/IPC-SHM (accessed on 30 August 2020).
33 Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. Neural networks, 1(2), 119-130.
34 Reinhard, Erik, et al. "Color transfer between images." IEEE Computer graphics and applications 21.5 (2001): 34-41.
35 Heichel, J., Mitra, R., Jafari, F., Das, A., Dorafshan, S., & Kaabouch, N. (2023, June). A System for Real-Time Display and Interactive Training of Predictive Structural Defect

Models Deployed on UAV. In *2023 International Conference on Unmanned Aircraft Systems (ICUAS)* (pp. 1221-1225). IEEE.

5 User Guide

5.1 Introduction

Before beginning this User Guide, it is recommended to understand the hardware and software components and their integration. This section also will guide through the initial steps to get started with the smart inspection process.

5.2 Hardware Components

Table 8: Hardware Components

Type	Name	Features	Purpose
Laptop	Dell Latitude 5430	Semi-rugged with enhanced battery life Intel Core i5 1145G7 /2.6 GHz 1 TB SSD NVMe Class 40 16GB, 2x8GB, 3200 MHz DDR4 RAM	-Runs the GUI for Smart Defect Detection -Powers the router -Hosts the AI training processes for defect detections
Microcomputer	Nvidia Jetson NX	CUDA-enabled parallel computing capability 384 NVIDIA CUDA® Cores, 48 Tensor Cores, 6 Carmel ARM CPUs Delivers up to 14 TOPs for AI applications in 10W power utilization	-Hosts AI processes for defect detection -Controls the payload cameras -Establishes connectivity with the laptop when both in same network
Visual Camera with Lens	Arducam 477P HQ Camera Board Lens: Arducam C-Mount Lens	Optical Format: 1/2.3″ Maximum still resolution: 4056 × 3040 30fps@Full 12.3MP Supports NVIDIA Argus Camera plugin for H264 encoding, JPEG snapshots C-Mount Lens: 16mm Focal Length Manual Focus and Aperture Adjustment, F1.4 to F16	-Provides live visual camera stream -Captures visual image for AI processing -Lens' wide aperture allows for very deep depth of field
Thermal Camera	Teledyne FLIR Boson	Resolution: 640x512 12 µm pixel pitch VOx microbolometer Temperature rating: -40 °C to +80 °C	-Provides live thermal camera stream -Captures thermal image

		Low power consumption around 500 mW For rugged construction	
Gimbal	Gremsy Mio	Payload upto 400g Lightweight	-Allows stabilization of the sensors to prevent motion blur
Router	GL.iNet GL-AR750	AC VPN Travel Router 300Mbps(2.4GHz) + 433Mbps(5GHz) Wi-Fi	-Establishes a cryptographic Wi-Fi network connection between the microcomputer and the laptop
Battery	HRB 4S Lipo Battery		-Powers microcomputer and the gimbal during flight
Battery Charger	Hobby Fans B6 Balance Charger		-Charges the batteries in balanced charged mode for short-circuit, overcharge, overcurrent and overheat protection.

Figure 29 shows the hardware components and how they are interconnected. Please refer to the respective hardware documentation for detailed system requirements and compatibility information.

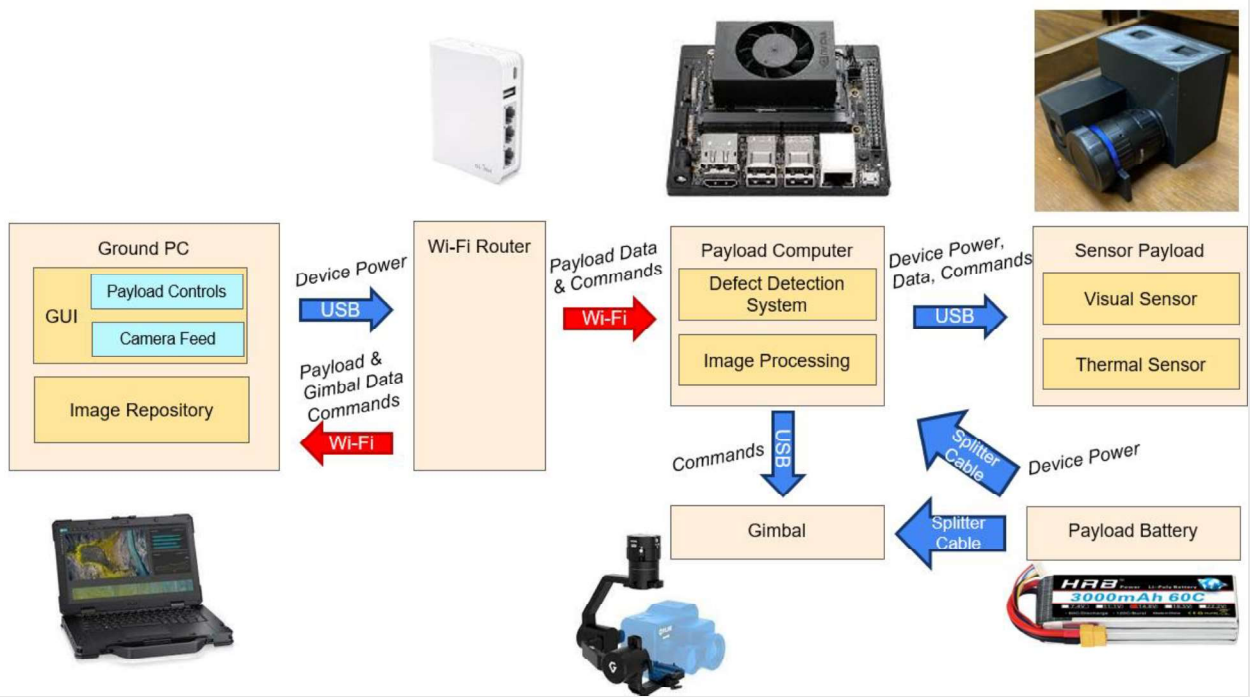


Figure 28: System Connections

5.3 Software Components

Table 9 lists the software components used on the laptop and microcomputer. Please refer to the respective software documentation for detailed system requirements and compatibility information.

Table 9: Software Components

Segment	Name	Version	Usage
Laptop	GStreamer 1.0 GStreamer 1.0 (Development Files)	1.20.2	-Pipeline-based multimedia framework -Used for live video streaming and image transfer
	gTuneDesktop	1.4.9.1	-Configures the gimbal
	Python	3.9.7	-Controls peripheral features of the GUI
	Visual Studio Community 2022	17.3.5	-Platform on which the GUI is developed
	C#	10.0	-Core code of the GUI
	.Net Framework	4.7.2	-Framework on which the GUI runs
Microcomputer	Python	n/a	-Core code of the Jetson

5.4 System Architecture

The System Block Diagram is shown below. The left side of the diagram is for the hardware/software located on the ground, while the right side is for the hardware/software located in the air, on the UAV.

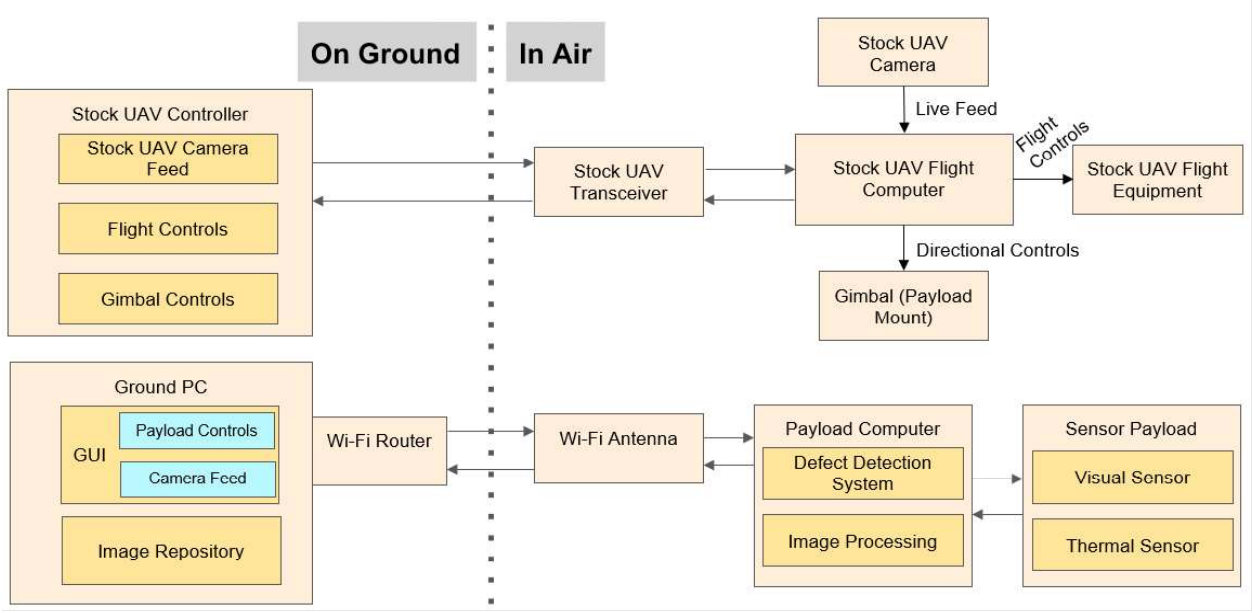


Table 10: System Block Diagram

The payload described in this document works separately from the stock UAV hardware and software. The only interaction is the control of the gimbal that holds the payload should be commanded by the UAV flight controller. The method for accomplishing this varies, depending on the UAV controller, but usually is a simple setting in the software provided by the flight controller.

The payload's computer (Jetson Xavier NX) contains the AI models to detect defects within the captured images on command. The data and commands between the payload computer and Ground PC are transmitted over a Wi-Fi network established by a travel router (IEEE 802.11ac).

A Ground PC is used to run the GUI and store images in a repository. The PC was selected to be rugged to survive any moisture or physical stress from operating outdoors.

5.5 Startup Instructions

Before initiating the inspection, it is crucial to follow a series of steps to ensure a seamless and successful operation. These steps will help set up the necessary connections and configurations for controlling and monitoring the drone's payload, which is equipped with advanced defect detection capabilities.

- Power on the laptop and make sure it remains on a stable surface to ensure smooth operation.
- Establish a physical connection between the laptop and the router to enable data transfer.
- Allow the router to activate the 5G WIFI network (green light should be on), which is essential for seamless communication between devices.

Connect the laptop to the 'NDDOT_ROUTER_5G' network, providing internet access for further actions.

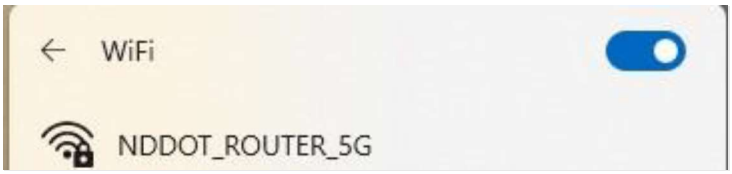


Figure 29: Wi-Fi network

- Ensure the UAS has met all the pre- and post-flight requirements (Follow general operation and safety guidelines recommended by FAA).
- Attach the gimbal, which is the mechanism responsible for stabilizing and controlling the payload, to the drone securely.
- Power up both the microcomputer and the gimbal using the designated payload batteries to activate their functionalities.
- Once the drone is airborne and stable, launch the *gTuneDesktop* app on the laptop.
- Verify gimbal connectivity and controls using the app controls.
- Open the *Smart Defect Detection Interface* app (GUI) on the laptop, specifically designed for controlling the payload's defect detection capabilities.

Figure 30: GUI Launch Page

- Navigate to the *Payload Control* page within the app's interface to access the relevant controls

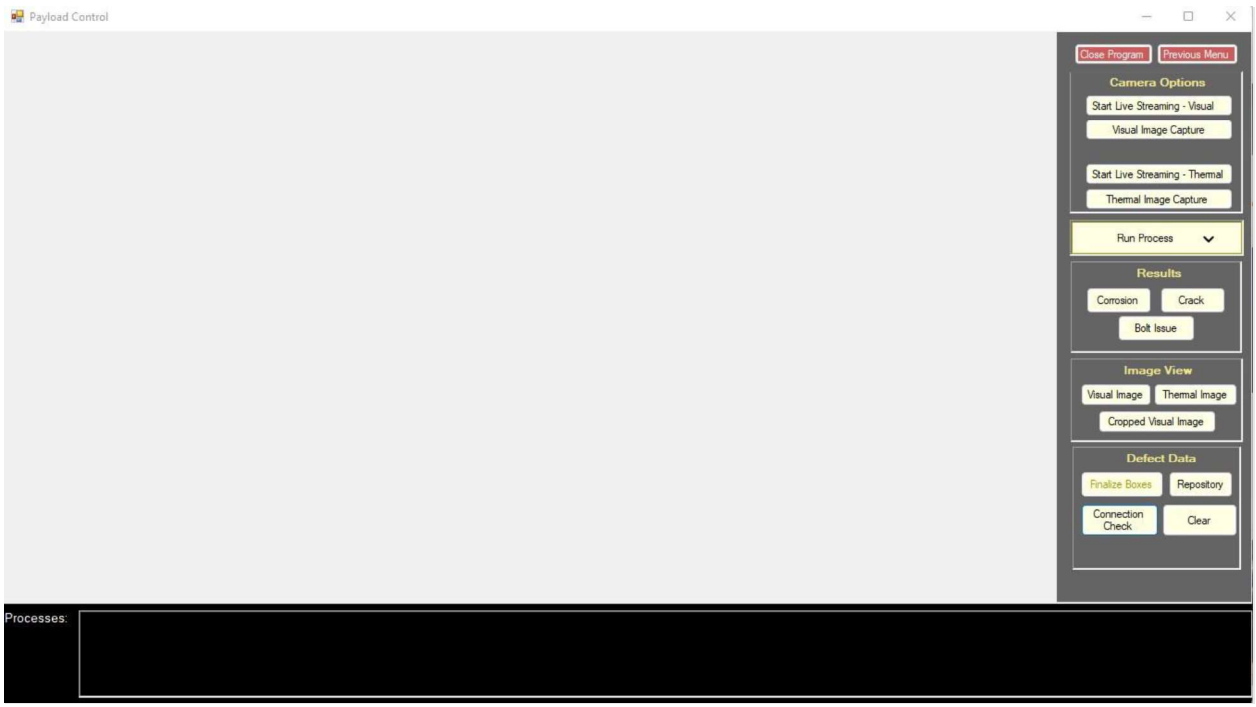


Figure 31: Payload Control Page

Click on the '*Connection Check*' button within the *Payload Control* page. A successful ping would indicate that the connection between the laptop and the payload is established and ready for inspection.

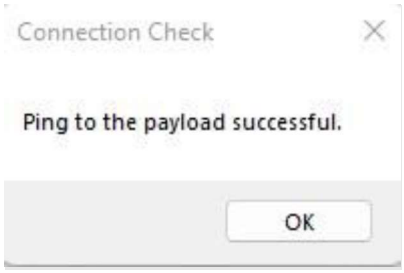


Figure 32: Payload Ping

5.6 Operation Instruction

5.6.1 Functionalities of the GUI

5.6.1.1 Initial Launch page

This page is the initial launch page to the GUI. The initial launch page gives access to the payload control application and desktop application.

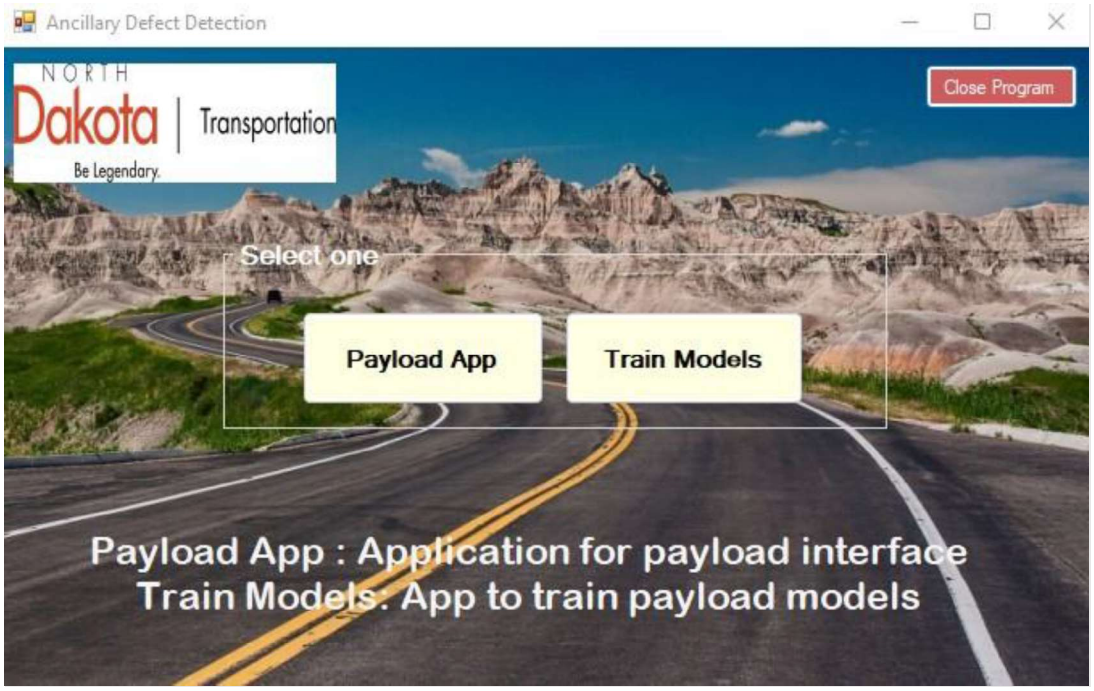


Figure 33: GUI Launch Page

Table 11: GUI Launch Page Description

Button Name	Button Functionality
Payload App	-Launches the Payload Control application for defect detection
Train Models	-Launches the Train Models application
Close Program	-Closes the GUI application

5.6.1.2 Payload Control Page

The Payload Control page functions as a primary hub, giving access to all the tools and needes to successfully control the payload for detecting defects in the target structure. The purpose of this section of the user guide is to give an understanding of the Payload Control

capabilities. Section 5.6.1.6 will guide through the step-by-step process for using the Payload Control features effectively during the field inspections.

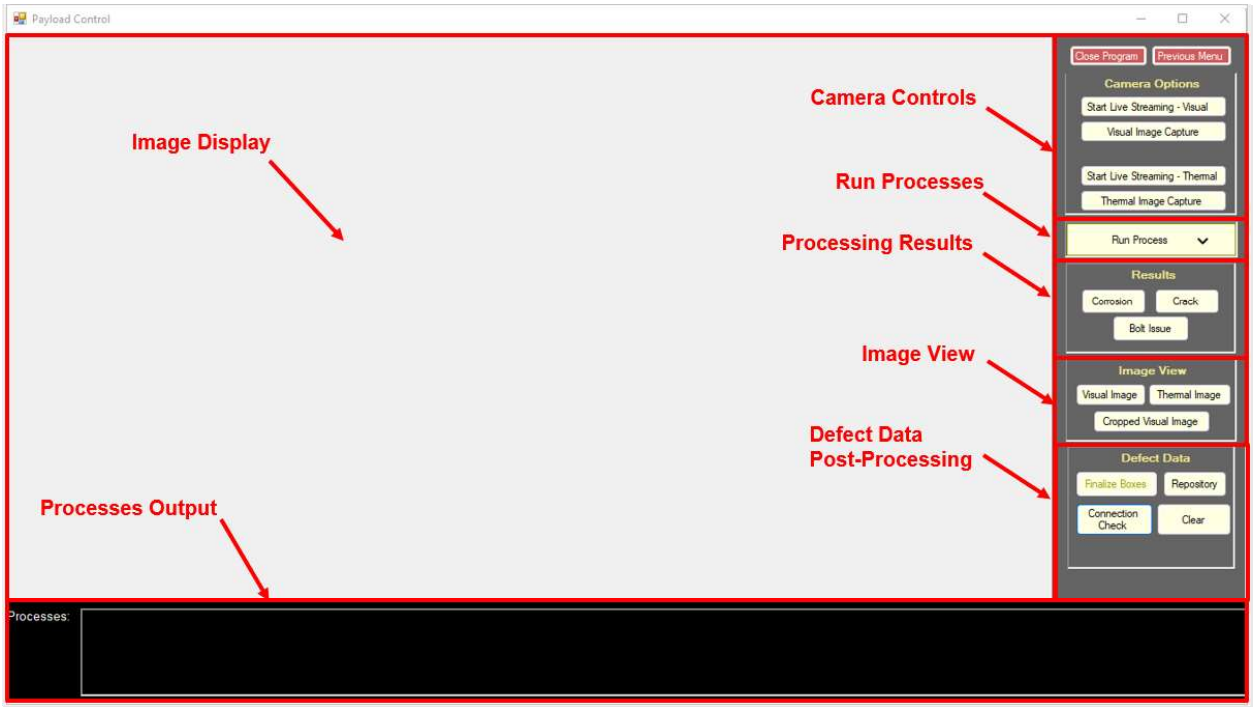


Figure 34: Payload Control Page Details

The Payload Control page consists of the following sections:

- Camera Controls:

This section gives access to use the live streaming and image capture feature using the visual and thermal camera.

- Run Processes:

This section can run the preferred AI model or models for detecting various defects. With a focus on enhancing safety and reliability, the available AI models cater specifically to the identification of corrosion, cracks, and bolt issues. The user has the flexibility to select any one AI process or a combination of processes. This gives the ability to modify the fault detection strategy to fit the particular traits of the area of ancillary structure under study. Utilize *Run Process* button drop-down flexible option to tailor the defect detection procedure using the AI process or processes that work best for the mission. The *Run Process* button has the following drop-down options:

Table 12: Run Process Options

Tick Button Name	**Tick Button Functionality**
All Defects	-Runs all the AI processes
Corrosion	-Runs the Al process for detecting corrosion defects
Crack	-Runs the Al process for detecting crack defects
Bolt	-Runs the Al process for detecting bolt issue defects

The user can choose any one or combination of AI processes they wish to run.

- Processing Results

This section gives access to the output of all the AI processes. The user can also toggle between the output screen for multiple AI process runs. This section provides the real-time and interactive capability to examine the predictions made by the AI processes. The output of the AI processes will be imaged with the defected regions marked on the image. Additionally, the user can add missed regions of defects and edit incorrectly categorized regions of defects. This serves two purposes: it allows the user to use their expertise to store accurate images to the repository and it also provides accurate retraining feedback for the AI processes. The output image generated by each AI process is color coded based on the defect type. The color codes are as following:

Corrosion: Blue Crack: Red Bolt Issue: Yellow

- Corrosion and Crack AI Processes

The AI processes for Corrosion and Crack use a similar method of defect inspection and annotation. These AI processes mark the entire image into small rectangular tiles of fixed dimensions. The tiles with defected region of interests are slightly color-shifted. This is for better and easier identification of the defected regions. The color shift is necessary in the event that a large region of tiles is flagged for defect and the tile outline alone becomes ambiguous between adjacent boxes. These tiles are interactive, as the user can check and uncheck them depending on their expertise that the AI process falsely classified or missed classifying the defect.

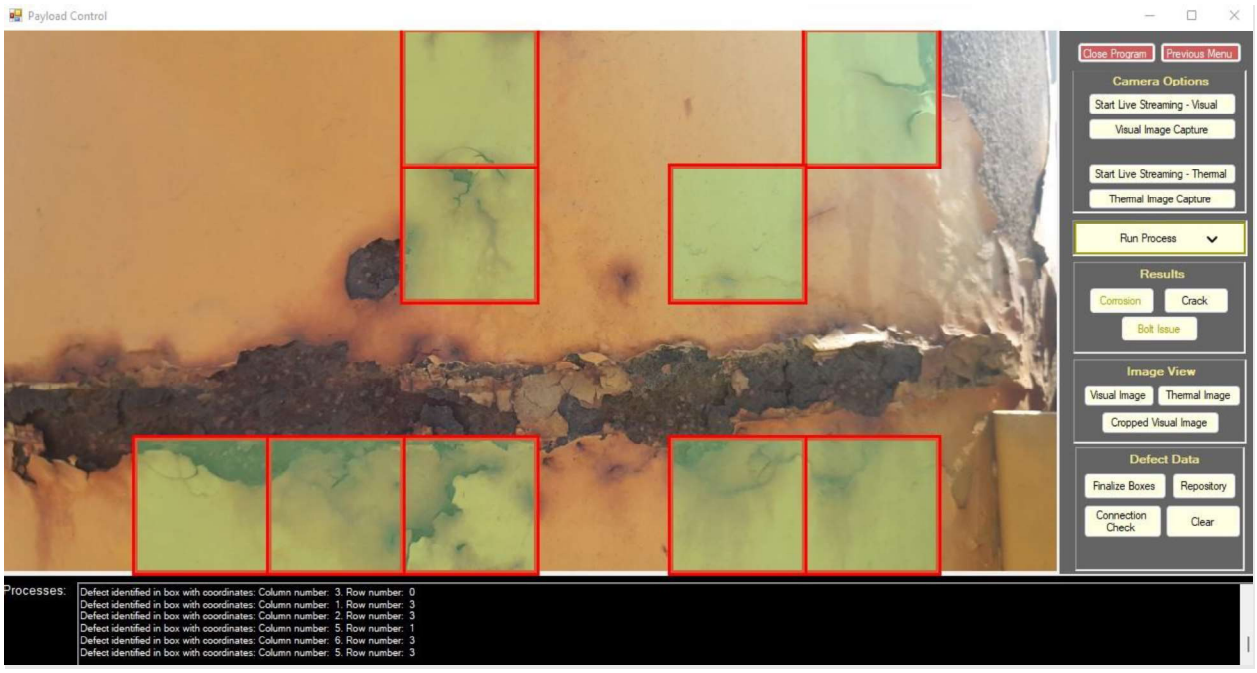


Figure 35: Crack Processing Feedback

- Bolt Issue AI Process

The AI process for bolt issues is used to identify loosening and missing bolt faults. By using AI, a rectangular region of interest is drawn around any missing or loosened bolts. The bolt or bolt hole being inspected determines the size of these rectangular regions of interest. The user can draw, remove and redraw the areas of interest depending on their expertise that the AI process falsely classified or missed classifying the fault.

Figure 36: Bolt Processing Feedback

- Image View

This section grants access to view the latest image captured by the visual and thermal cameras, along with the ability to view the latest masked image. The buttons provided under this section offer adaptability, allowing the user to set the visual image or re-crop it for rerunning AI processes on the existing visual image. With just a click, the user can examine the visual and thermal images, analyze the masked image for better marking of defect region of interests, and effortlessly manipulate the visual image for optimized AI analysis. Utilizing the strength of these capabilities on the Payload Control page will simplify and improve the ancillary structure defect detection process.

- Miscellaneous

The Miscellaneous section offers a collection of graphical user interface centric control options, providing convenient functionalities for enhanced user experience. Within this section, the user can find options such as saving output images, accessing the repository, performing connection checks with the payload, and clearing all selections. These will not only provide easy preservation of the generated images for future reference or documentation purposes but also will provide quick access to a centralized repository. The *Connection Check* button is a crucial functionality and must be done below every flight. This will allow the user to verify the connectivity status with the payload microcomputer and the laptop, ensuring a stable and reliable connection. Lastly, the *Clear All Selections*

button conveniently resets any selected settings or parameters, enabling a fresh start or facilitating a streamlined workflow. These graphical user interface-centric controls enhance the overall usability and efficiency within the application.

Table 13: Miscellaneous GUI Buttons

Button Name	**Button Functionality**
Start Live Streaming - Visual	-Starts live streaming from visual camera
Visual Image Capture	-Takes a picture from the visual camera
Start Live Streaming - Thermal	-Starts live streaming from thermal camera
Thermal Image Capture	-Takes a picture from the thermal camera
Run Process	-Chooses the AI process/processes to run
Corrosion	-Visualizes the output of the AI model for corrosion defect
Crack	-Visualizes the output of the AI model for crack defect
Bolt Issue	-Visualizes the output of the AI model for bolt issue defect
Visual Image	-Visualizes the last captured image from the visual camera
Thermal Image	-Visualizes the last captured image from the thermal camera
Cropped Visual Image	-Visualizes the last cropped visual image
Finalize Boxes	-Saves the final inspector annotated and approved selection of the defect output shown on the screen
Repository	-Saves the final annotated output shown on the screen to a repository
Connection Check	-Checks whether a connection between the laptop and the microcomputer on the payload has been established
Clear	-Clears the current process run and set all the values to default
Close Program	-Closes the GUI application and turn off the microcomputer on the payload
Previous Menu	-Goes back to the Initial Launch page

5.6.1.3 Image Masking page

The Image Masking page provides the functionality to crop a specific area of interest within the visual image offering a high level of control and customization. This subsequently becomes the focus for applying the AI process/processes of choice. This feature offers flexibility in running AI processes, as it enables users to focus on a particular region within the image for analysis. By creating the mask, the user can define and isolate the desired

area, fine-tune, and optimize the analysis allowing for targeted and precise application of AI algorithms. This capability enhances the efficiency and accuracy of AI processing by concentrating computational resources on the specific region of interest.

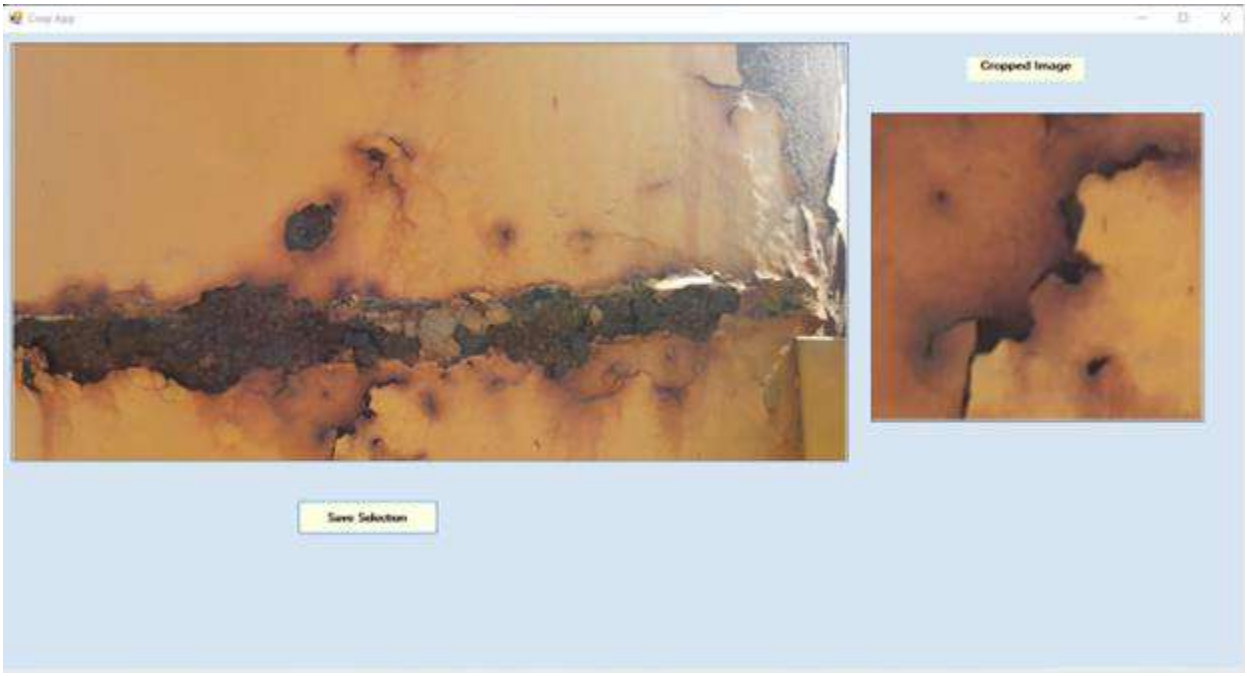


Figure 37: Image Masking Page

Button Name	Button Functionality
Save Selection	-Saves the cropped selection of the visual image

5.6.1.4 Repository Page

The Repository Page is specifically designed to provide inspectors with a visual overview of past inspections, showcasing images of three types of detected defects: corrosion, cracks, and bolt issues. This page serves as a centralized platform to catalog and organize the visual evidence of these defects detected after the AI processing and appropriate annotations given by the inspectors. The user may quickly access and analyze the detected defects for additional investigation and documentation using this page. The repository allows for efficient retrieval and comparison of defect images, enabling the user to track the progression of corrosion, monitor crack growth, and assess the severity of bolt issues over time. In addition, the page also features a save button that allows the user to save the

displayed image. This will allow the user to easily attain and utilize the defect images for documentation, reporting, or further analysis outside the software.

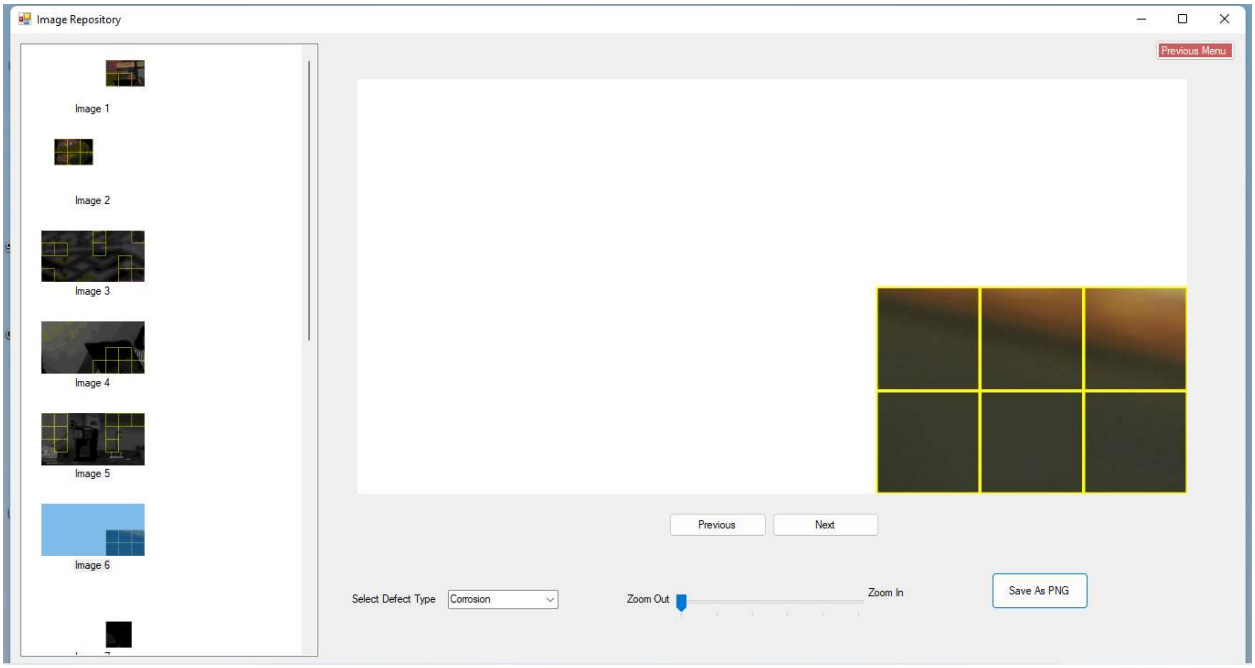


Figure 38: Repository Page

The user can choose the type of defect repository they wish to view using the *Select Defect Type* drop down options. The *Select Defect Type* has the following drop-down options:

Figure 39: Select Defect Type

Option Name	**Option Functionality**
Corrosion	To view the defects from corrosion repository
Crack	To view the defects from crack repository

Button Name	**Button Functionality**

Previous	-Views the previous image in the selected defect repository
Next	-Views the next image in the selected defect repository
Save As PNG	-Saves the image as PNG in the download folder
Previous Menu	-Goes back to the Payload Control page

5.6.1.5 AI Process Retraining Control page

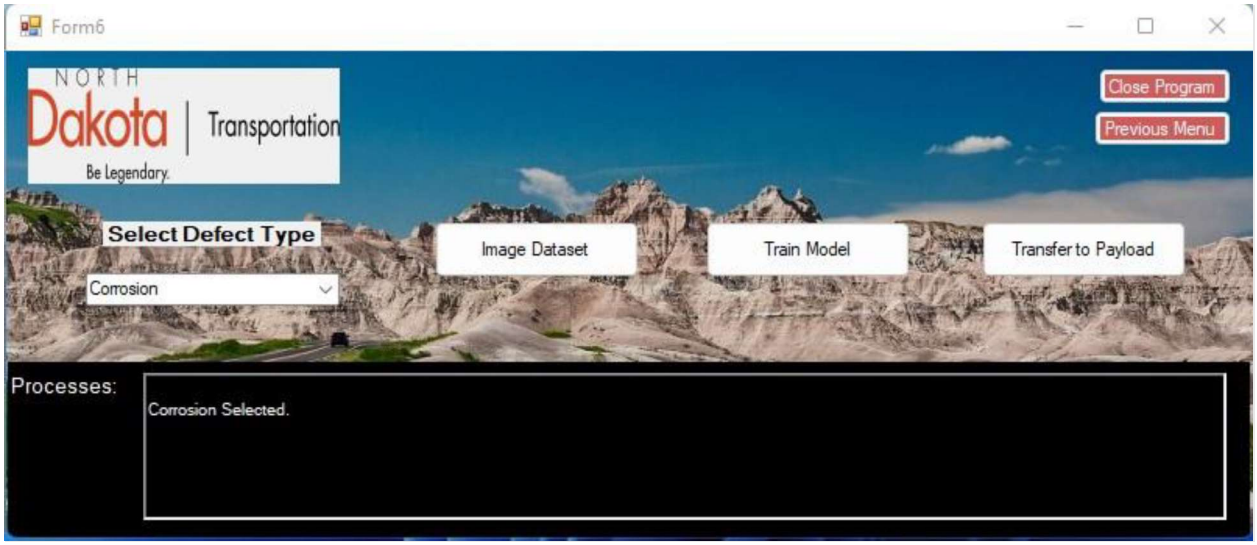


Figure 40: Retraining Page

Button Name	**Button Functionality**
Image Dataset	-Optimizes the image dataset for the selected AI process for retraining
Train Model	-Trains the selected AI process
Transfer to Payload	-Transfers the trained AI process model file to the microcomputer on the payload
Connection Check	-Checks whether a connection between the laptop and the microcomputer on the payload has been established
Previous Menu	-Goes back to the Initial Launch page
Close Program	-Closes the GUI application and turn off the microcomputer on the payload

The user can choose the type of AI process they wish to train using the *Select Defect Type* drop down options. The *Select Defect Type* has the following drop-down options:

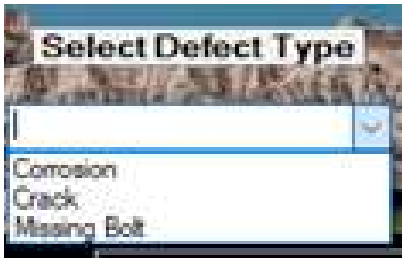
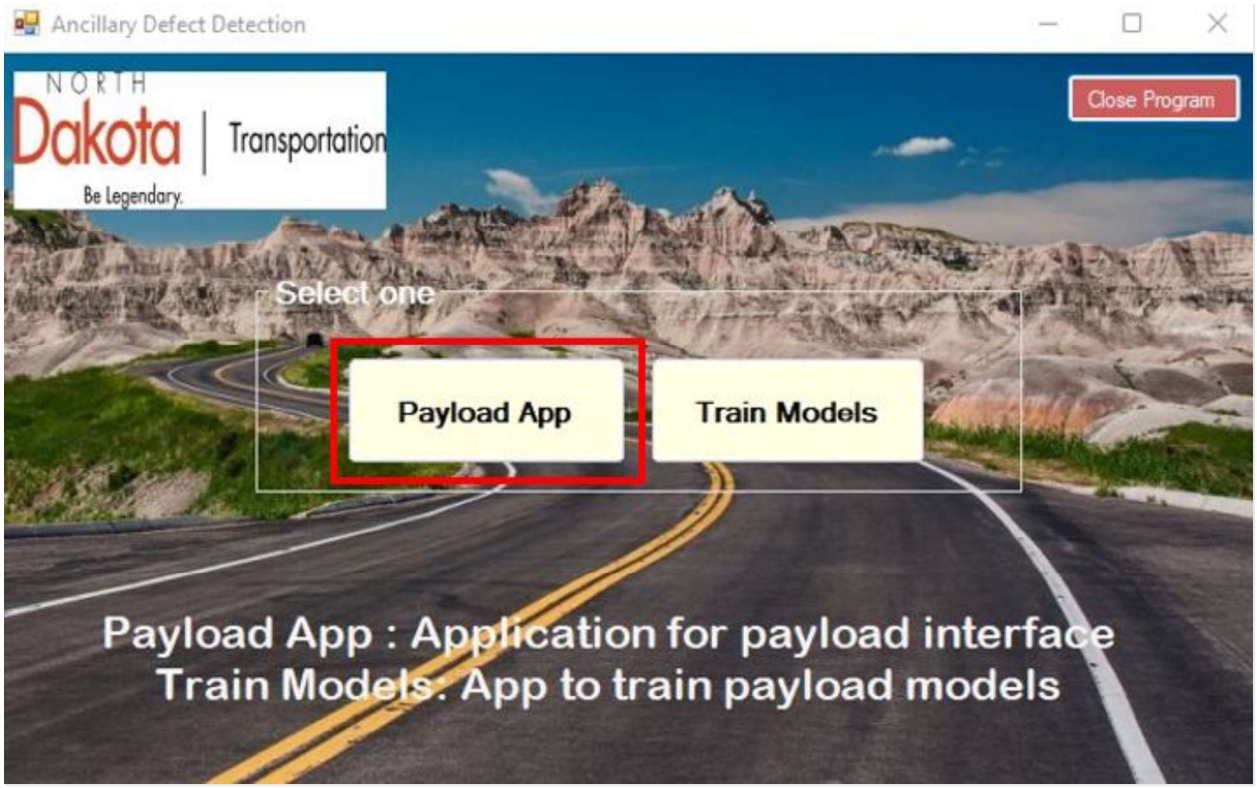


Figure 41: Select Defect Type

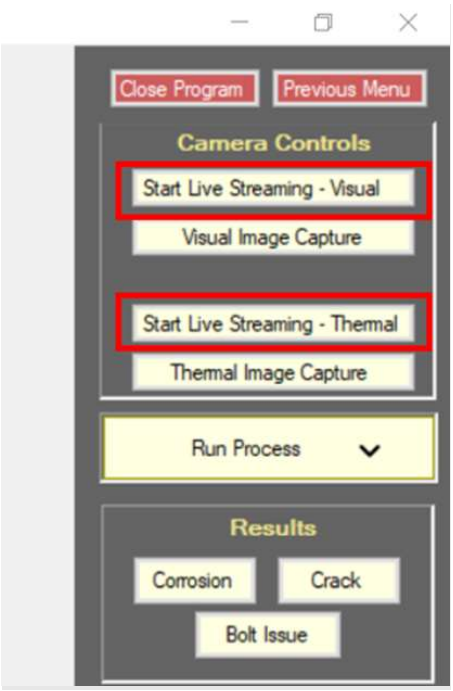
Option Name	Option Functionality
Corrosion	-Runs the train Al process for detecting corrosion defects
Crack	-Runs the train Al process for detecting crack defects
Bolt	-Runs the train Al process for detecting bolt issue defects

5.6.1.6 Step-by-step Instructions for Inspection Mission

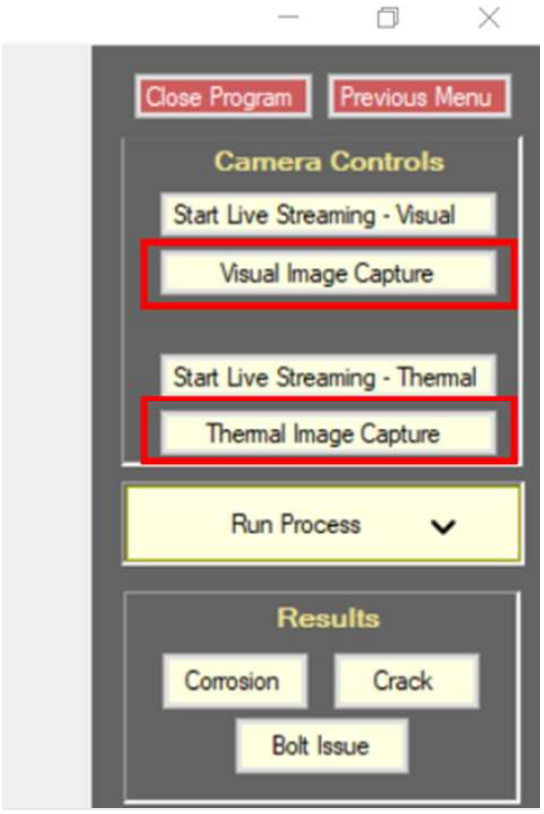
1. On the launch screen of the GUI, click Payload App to start.



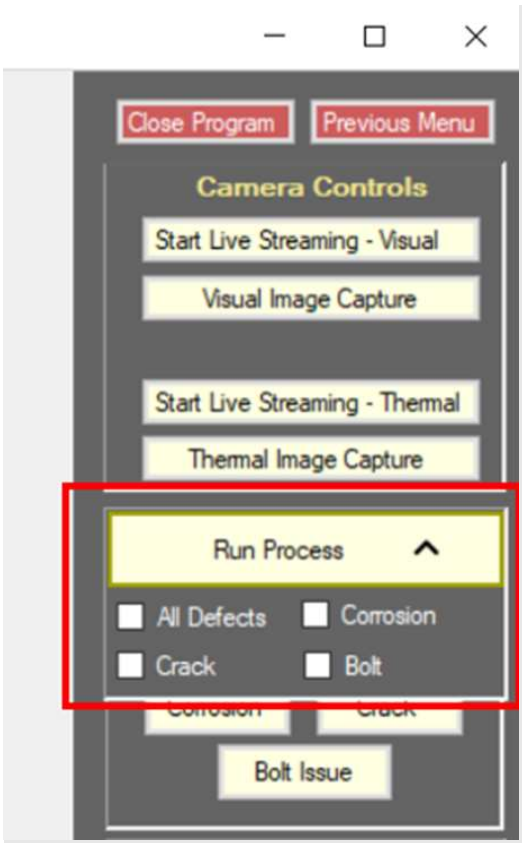
2. Start live streaming from visual and/or thermal cameras by clicking the corresponding buttons shown below.
    i. Place the new popup window(s) containing live streams to a preferred location on the screen.



3. Start the flight.
4. Any area of interest on the target structure can have a still image captured from either the visual or thermal camera. Click the corresponding button shown below.



5. If the inspector suspects one or more of the three defect categories (Corrosion, Crack, or Bolt Issue) may be present, they can run a defect detection process to assist. Dropping down the 'Run Process' box results in the box shown below, where one or more of the defect detection processes can be checked to run. After selecting the processes to run, click 'Run Process'.



6. The processing may take several minutes to run. The steps and output of the processing can be shown in the Processes box.
7. When the processing is complete, the GUI will display the areas of the image that contain the selected defects. The GUI provides interactive controls to add or remove defects, if the inspector determines the processes are incorrect. These controls are explained in Section 5.6.1.7 for Corrosion and Crack defects and Section 5.6.1.8 for Bolt defects.
8. Once the defects are finalized, the images can be saved to a repository for future review. Section 5.6.1.4 explains saving the images to a repository.
9. These steps can be repeated for all other areas of interest in the target structure.

5.6.1.7 Interactive functionality for Corrosion and Crack defects

The Interactive GUI functionality works the same for both Corrosion and Crack defects. The only difference is Corrosion defects are identified by blue boxes, while Crack defects are identified by red boxes.

When the Defect Detection process for Corrosion and/or Crack are run on the current image, the Interactive GUI will divide the image into an 8 x 4 grid of sub-images to prepare for processing. When the processing is complete, the Interactive GUI will place a box around each sub-image that contains the defect(s). If the inspector disagrees with the

identified defect locations, they can change the selected boxes before storing the image in the repository. They can de-select a sub-image that they believe doesn't contain a defect, and the box will be removed. Also, they can select a sub-image that they believe does contain a defect that wasn't identified by the process.

An example is provided in Figures 35 and 36 below. In Figure 35, the Corrosion process identified defects in the following sub-image locations (column, row coordinates from top left): (5,1), (6,1), (7,1), (8,1), (4,2), (1,3), (3,3), (4,3), (5,3), (2,4), (6,4), (7,4), and (8,4). However, Figure 36 shows the result of the inspector disagreeing with the process. They de-selected sub-images (5,1), (6,1), and (8,4) meaning the model falsely selected these sub-images as defective. The inspector also selected sub-images (4,1), (3,2), (5,2), etc. meaning the process falsely did not identify these sub-images as defective.

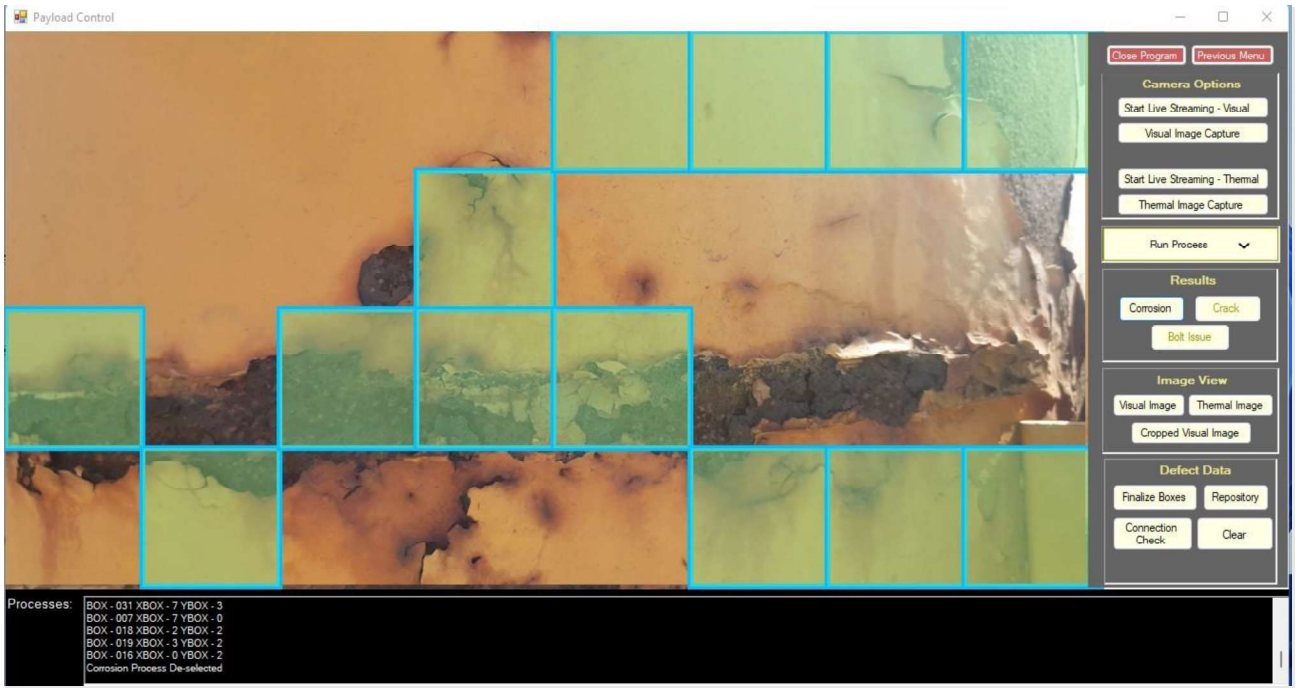


Figure 42: Image after Corrosion Process

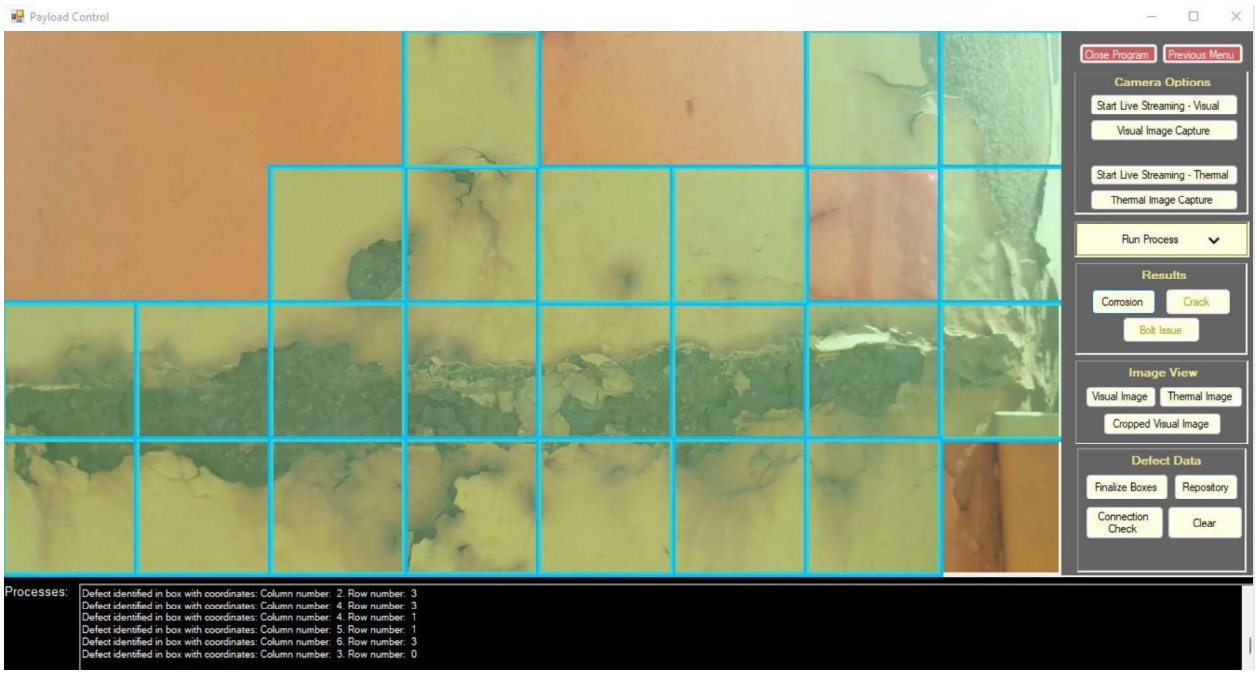
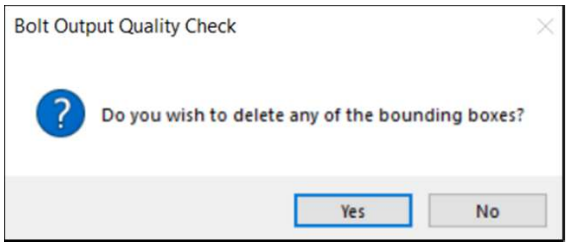


Figure 43: Image After Corrosion Process and Inspector Interaction

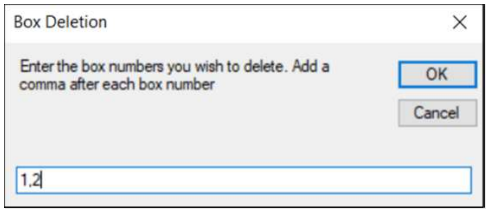
When the inspector is finished with modifying the defect boxes on the image, they can save the image to the repository, which will include any changes made by the inspector. The inspector can repeat the process of modifying the boxes of the current image and saving to repository.

5.6.1.8 Interactive functionality for Bolt defects

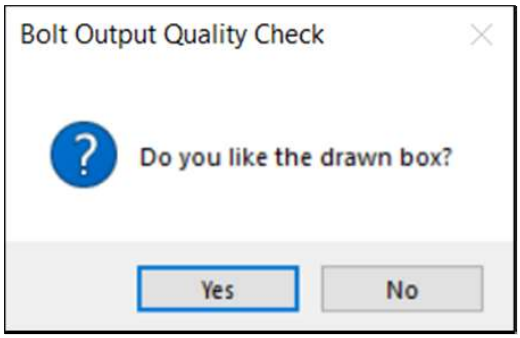
The Interactive GUI for Bolt Issues works differently than Crack and Corrosion. When the Defect Detection process for Bolt Issues is run on the current image, the Interactive GUI will identify the exact location of each Bolt Issue, without having to divide the image into sub-images. Yellow bounding boxes will be drawn around each identified defect, with numbers listed on the top left corner of each box. The size of the boxes vary, depending on the sizes of the defective areas. When the processing is complete, the pop-up box shown below appears.



If any defects appear to be incorrect, the inspector can click yes, otherwise they can click no. If yes, is clicked, the pop-up box below appears. The inspector enters in the box numbers to be deleted, separated by a comma (example 1,2).



The bounding boxes for Bolt Issues can't be deleted after the pop-up is closed. The Bolt Issues process would need to be re-run to get to this pop-up again to delete the boxes. After these boxes are closed. The inspector can draw boxes around any areas that they determine to be a Bolt Issue. The boxes can be any size and can be drawn in any direction. The inspector just needs to click and hold in one corner of the desired box, then drag the mouse to the opposite corner of the box. After each new box is drawn, the pop-up box below is shown. The inspector can click Yes to save the box, or no to delete the box.



When the inspector is finished with modifying the defect boxes on the image, they can save the image to the repository, which will include any changes made by the inspector. The

inspector can repeat the process of modifying the boxes of the current image and saving to repository. An example image with identified Bolt defects is shown below.

Figure 44: Bolt Defect Box Feedback

5.6.2 Step-by-step Instructions for Model Retraining

- On the launch screen of the GUI, click 'Train Models' button to start.
- On the Train Model page, choose the type of defect training model to train using the *Select Defect Type* drop down options.
- Choose Corrosion or Crack.
- Click on the 'Image Dataset' button to view the message for choosing the training dataset. Inspector needs to choose the images from wc/woc folder (c:/Project_Parent/Model_training/corrosion_dataset or c:/Project_Parent/Model_training/crack_dataset) for retraining. The selected images should be stored in the respective training dataset. For example for corrosion : C:\Project_Parent\Model_Training\corrosion\Corrosion\Submission_nov_2\Dataset\train\ WC or WOC
  For crack: C:\Project_Parent\Model_Training\crack\Pythoncode\caltec256subset\train\a(WC) or b(WOC)
- Click on the 'Train model' button to start the training. A black consol window will appear which will highlight the progress of the training. Please do not turn off the consol window or lock the laptop screen. It will take nearly 2 hours to complete the training for corrosion and around 45 mins for crack.
- Once the black console window automatically disappears, it means the model has been trained successfully. Corrosion model will be training for 50 epochs and Crack model will be training for 30 epochs.

- Now click on the ‘Previous Menu’ button to return to the launch page of the GUI.
- Click on the ‘Payload App’ button within the app's interface.
- Click on the 'Connection Check' button on the Payload Control page. A successful ping would indicate that the connection between the laptop and the payload is established and ready for file transfer.
- Now click on the ‘Previous Menu’ button to return to the launch page of the GUI.
- Click on the ‘Train Models’ button within the app's interface.
- Reselect the defect model type from the *Select Defect Type* drop down options
- Click on Transfer to Payload button. This button takes a backup of the existing model prediction file in the Jetson board and transfers the newly generated model file to its necessary folder in the Jetson board.

6 Technical Section

The following sections in the User Guide describe the design features of the Payload and GUI. These are not crucial to operating the system. If troubleshooting is necessary, these sections are a good resource to understand the system design.

6.1 Microcomputer wireless connectivity

To ensure a seamless connection, the WiFi network named ‘NDDOT_ROUTER_5G’ is given the highest priority level of 999. This prioritization means that when multiple networks are available, the device will prioritize connecting to ‘NDDOT_ROUTER_5G’ above others. Consequently, as soon as the microcomputer boots up, it will promptly connect to the ‘NDDOT_ROUTER_5G’ WiFi network without requiring any manual intervention. This streamlined setup ensures a convenient and reliable connection between the microcomputer and the laptop, facilitating efficient operation and maximizing productivity during the inspection process. If the router changed to factory setting, it can be reset again to NDDOT_ROUTER_5G and the key is “password”. The password for the laptop and the jetson are “nddot” and “password” respectively.

6.2 Auto connectivity between laptop and the microcomputer

SSH password less login is used here to establish a password less and synchronous mode of communication between the laptop and the microcomputer. It is an effective authentication method for tasks like file synchronization, and server access. This method relies on a pair of public and private keys. It is set up by firstly generating a key pair using

the *ssh-keygen* command. Then, after creating an SSH directory on the server, public keys are uploaded to the server using *ssh-copy-id* command on Linux client or *scp* command on Windows client. Next, connections are tested after configuring the SSH agent permissions for the *.ssh* directory and the *authorized_keys* file. Once completed, a passphrase-free passwordless connection to the server is established, enhancing security and convenience. Here both the Windows OS on the laptop and the Linux OS on the microcomputer act are set with the passwordless SSH login. In the Windows system the *.ssh* directory is located at "*C:Users\NDDOT*" and in the microcomputer system the.*ssh* directory is located under the root directory.
** If the router goes back to the factory setting, please check docs.gk-inet.com for resetting the password. The security key is "password".

6.3 Form 1 Overview

Form 1 is the main form of the GUI that the inspector will use. It commands the payload, shows the images from the payload, and includes interactive controls that the inspector can use to modify and store defect data. The back-end code files are Form1.cs and Form1.Designer.cs.

Form1.Designer.cs provides the settings for all the GUI elements (buttons and text boxes for example). Most of the lines in this file were automatically generated by Visual Studio, so not much annotation is present.

Form1.cs has all the other code for handling inputs and outputs of the GUI. All the methods that end with "bat" in their name are used for running the .bat files, which are most commonly used for interfacing with the payload's sensors or defect detection AI processes. A large section of Form1.cs code is used for handling the interaction between the inspector and the GUI for modification of defect data. The three-dimensional array "boxarray" is modified frequently and is used for saving the crack and corrosion defect location to ensure the information is retained when the GUI is changing the currently displayed image. When the inspector changes the image to be displayed on the GUI, the new image is re-printed on the GUI screen and any defect boxes are re-drawn based on the saved data within the "boxarray" array. The integer "defectenum" is for enumerating between different defect types as follows:

- 0 = Corrosion
- 1 = Crack
- 2 = Bolt

The methods that end with "Repo" are used for running the .bat files to prepare different images for storage in the repository (as explained in the section on Form 5). If the image contains defect boxes, the images saved in the repository will include those boxes.

6.4 Form 4 Overview

Form 4 is the form that allows the inspector to crop a captured image to run a reduced subset through the AI processes. The Form takes the current image and allows the inspector to click and drag their mouse over the regions that they would like to crop. The form will prompt them upon creating the crop if they like it or would like to redo the cropping. To allow cohesion with the Crack and Corrosion models, the crop extends the crop region to cover the 8 x 4 grid of boxes used for Crack and Corrosion defect classification. This form does not require many lines of code, and thus is able to run and close quickly when called.

6.5 GStreamer

GStreamer is an open-source multimedia framework used to create versatile multimedia applications. By constructing pipelines that connect different plugins, developers can process and transmit audio and video data in a flexible and modular manner. With support for a wide range of multimedia formats and protocols, GStreamer is highly customizable and extensible, making it a popular choice in the Linux ecosystem. Its extensive documentation, tutorials, and community support empower developers to leverage its capabilities for creating diverse multimedia applications and solutions.
GStreamer can be used to create both the server and client components of a streaming system. The server-side component typically involves designing a GStreamer pipeline that captures audio and video from a source (e.g., camera), encodes the data, and streams it over a network protocol such as RTP (Real-time Transport Protocol) or RTSP (Real-Time Streaming Protocol). This pipeline can be set up on a server machine. On the client-side, another GStreamer pipeline is designed to receive the streamed audio and video data from the server. This pipeline decodes the received data and can be configured to play it back, display it on a screen, or process it further.

Please refer to the Gstreamer official documentation for detailed system requirements and compatibility information at https://gstreamer.freedesktop.org/.

6.6 Visual image capture

The visual image capture using the Arducam 477P HQ camera is executed using the python file '*Visual_Capture.py*'. The code captures an image using the camera device and saves it with a timestamp in a designated folder. It then performs some operations on the captured image, such as copying it to another folder, renaming it, resizing it, and saving the resized version. The shell script '*Image_Save_Visual.sh*' is used to execute the '*Visual_Capture.py*'.

The breakdown of the code executed on the microcomputer is as follows:

1. The *cmd* variable contains the command to capture an image using the Arducam 477P HQ camera.
2. It initially stores the image with the size, width = 1920 and height = 1080, and format of jpeg.
3. The *os.system()* function is used to execute the entire command.
4. The image is saved in the path 'home/nddot/IMAGE_CAPTURES/E_CON_CAM_PIC/Pic_1.jpg'
5. The current timestamp is obtained using *datetime.datetime.now()* and formatted as a string with microsecond precision.
6. The captured image is renamed using *os.rename()* to include the timestamp in the filename.
7. The *glob* module is used to find all the JPEG files in the folder specified by *folder_path*. The *max()* function is then used to find the most recently created file based on the creation time *os.path.getctime()*.
8. The most recently created file is copied to the '*/home/nddot/IMAGE_CAPTURES/RUN_FOLDER*' directory using *shutil.copy()*.
9. Similar to step 4, the most recently created file in the '*/home/nddot/IMAGE_CAPTURES/RUN_FOLDER*' directory is found using *glob* and *max()*.
10. The file is renamed to *1.jpg* in the '*/home/nddot/IMAGE_CAPTURES/RUN_FOLDER*' directory using os.rename().
11. The image is loaded using *cv2.imread()*.
12. The *height1* and *width1* variables are calculated to obtain dimensions that are multiples of 227.
13. The image is resized using *cv2.resize()* with the calculated dimensions.
14. The resized image is saved as *C.jpg* in the '*/home/nddot/IMAGE_CAPTURES/RUN_FOLDER*' directory using *cv2.imwrite()*.

The images are numbered according to the date and time to consider the latest image for processing. The images are also rescaled to the multiple of 227 * 227 as the AI models

used for detecting corrosion and crack have 227 * 227 as the input shape of the image. The image file named *C.jpg* located in the '*/home/nddot/IMAGE_CAPTURES/RUN_FOLDER'* directory serves as the input image for running various AI models. It acts as a base image that undergoes processing either in its entirety or after being cropped depending on the inspector specific requirements of the AI models. As the AI models require input images to work with, *C.jpg* must be regularly updated with the latest content to ensure accurate and up-to-date results.

6.7 Thermal image capture

The thermal image capture using the Teledyne FLIR Boson camera is executed using the python file '*thermal_capture.py*'. The code captures an image using the camera device and saves it with a timestamp in a designated folder. It then performs some operations on the captured image, such as copying it to another folder, renaming it, resizing it, and saving the resized version. The shell script '*Image_Save_Thermal.sh*' is used to execute the '*thermal_Capture.py*'.

The breakdown of the code executed on the microcomputer is as follows:

1. The *cmd* variable contains the command to capture an image using the Teledyne FLIR Boson camera.
2. It initially stores the image with the size, width = 640 and height = 512, and format of I420.
3. '*jpengc*' gstreamer command is used to command the thermal image to jpg format.
4. The *os.system()* function is used to execute the entire command.
5. The image is saved in the path 'home/nddot/IMAGE_CAPTURES/THERMAL/Pic_1.jpg'
6. The current timestamp is obtained using *datetime.datetime.now()* and formatted as a string with microsecond precision.
7. The captured image is renamed using *os.rename()* to include the timestamp in the filename.
8. The *glob* module is used to find all the JPEG files in the folder specified by *folder_path*. The *max()* function is then used to find the most recently created file based on the creation time *os.path.getctime()*.
9. The most recently created file is copied to the '*/home/nddot/IMAGE_CAPTURES/RUN_FOLDER*' directory using *shutil.copy()*.
10. Similar to step 4, the most recently created file in the '*/home/nddot/IMAGE_CAPTURES/RUN_FOLDER*' directory is found using *glob* and *max()*.
11. The file is renamed to 2.*jpg* in the '*/home/nddot/IMAGE_CAPTURES/RUN_FOLDER*' directory using os.rename().

The images are numbered according to the date and time to consider the latest image for processing.

6.8 Visual image live streaming

The smart GUI interface on the laptop displays the live stream captured by the Arducam 477P HQ camera, which is connected to a microcomputer. The camera captures visual data, and this real-time video feed is transmitted to the laptop for viewing. For availing the live streaming capabilities, GStreamer has been used.

The breakdown of the server end bash script – '*live_test.sh*' executed on the microcomputer is as follows:

1. *kill* command is used to kill any process that is using port 5010.
2. The *gst* command is used to launch the *gst-launch-1.0* utility with a pipeline for video streaming.
3. It uses the *v4l2src* element to capture video from the specified camera device.
4. The *image/jpeg* format is selected with a width of 1920 and height of 1080, and a framerate of 15 frames per second.
5. The *tcpserversink* element is used to stream the video over TCP, specifying the microcomputer (host) IP address as 192.168.8.139 and the port as 5010.

The breakdown of the client end Windows batch script – '*gstreamer_client.bat*' executed on the laptop is as follows:

1. *timeout* command is used to pause the script execution for 4 seconds without displaying any output on the console.
2. As the GStreamer binaries are installed in the location '*C:\gstreamer\1.0\msvc_x86_64\bin*', the *cd* command is used to change the current directory to that directory.
3. The *gst* command is used to launch the *gst-launch-1.0* utility with a GStreamer pipeline.
4. It uses the *tcpclientsrc* element to receive video from the microcomputer (TCP server) running at 192.168.8.139 and port 5010.
5. The received video is then decoded using *decodebin* and displayed using *d3dvideosink*, which renders the video using *Direct3D* on Windows.

The script prints the current time, waits for 4 seconds, and then prints the time again. It then changes the directory to the GStreamer installation directory and launches a GStreamer pipeline to receive and display video from a remote TCP server.

Note: Ensuring that the IP addresses and ports specified in the scripts are appropriate under the 'NDDOT_ROUTER_5G' static IP network setup.

6.9 Thermal image live streaming

The smart GUI interface on the laptop displays the live stream captured by the Teledyne FLIR Boson camera, which is connected to a microcomputer. The camera captures thermal data, and this real-time video feed is transmitted to the laptop for viewing. For availing the live streaming capabilities, GStreamer has been used.

The breakdown of the server end bash script – '*live_test_thermal.sh*' executed on the microcomputer is as follows:

1. *kill* command is used to kill any process that is using ports 5123 and 5021.
2. The *gst* command is used to launch the *gst-launch-1.0* utility with a pipeline for video streaming.
3. It uses the *v4l2src* element to capture video from the specified camera device.
4. The *I420* format is selected with a width of 640 and height of 512.
5. The *tcpserversink* element is used to stream the video over TCP, specifying the microcomputer (host) IP address as 192.168.8.139 and the port as 5123.

The breakdown of the client end Windows batch script – '*gstreamer_client_thermal.bat*' executed on the laptop is as follows:

1. *timeout* command is used to pause the script execution for 4 seconds without displaying any output on the console.
2. As the GStreamer binaries are installed in the location '*C:\gstreamer\1.0\msvc_x86_64\bin*', the *cd* command is used to change the current directory to that directory.
3. The *gst* command is used to launch the *gst-launch-1.0* utility with a GStreamer pipeline.
4. It uses the *tcpclientsrc* element to receive video from the microcomputer (TCP server) running at 192.168.8.139 and port 5123.
5. The received video is then decoded using *decodebin* and displayed using *d3dvideosink*, which renders the video using *Direct3D* on Windows.

The script prints the current time, waits for 4 seconds, and then prints the time again. It then changes the directory to the GStreamer installation directory and launches a GStreamer pipeline to receive and display video from a remote TCP server.

Note: Ensuring that the IP addresses and ports specified in the scripts are appropriate under the 'NDDOT_ROUTER_5G' static IP network setup.

6.10 Data transfer from payload and laptop

For transferring images, text and other necessary files in between the Linux system running on the microcomputer and Windows system running on the laptop, socket programming based on python is used. In socket programming for server and client data transfer, two separate programs are created. One acts as the server, and the other as the client. The server

establishes a connection to a particular port on a local IP address and waits for client connections. The server initiates a connection with the client once it connects. On the other hand, the client starts a connection to the server by giving it the IP address and port number. The client sends data to the server after the connection is established, and the server receives and processes that data. Data is often sent as streams of bytes during this client-server conversation. Data is converted from the client into bytes and transmitted over the network to the server, which interprets and utilizes the data in accordance. The similar procedure is used by the server to deliver data back to the client.

Python's socket module offers the tools needed to construct and maintain sockets for network communication, making it simple to design client-server data transfer. The breakdown of the server-end visual image transfer script – '*Server.py*' executed on the microcomputer is as follows:

1. The host variable is defined to specify the IP address of the server. In this case, it is set to 192.168.8.139.
2. The port variable is defined to reserve a specific port number on which the server will listen for incoming connections. It is set to 5020.
3. The socket is bound to the specified IP address and port number using the *bind()* method: *s.bind((host, port))*.
4. The socket starts listening for incoming client connections with a maximum backlog of 5 pending connections using the *listen()* method: *s.listen(5)*.
5. The script enters an infinite loop (while True) to continuously handle incoming connections.
6. When a client connects, the *accept()* method is called, which returns a new socket representing the connection and the client's address.
7. The server receives data from the client using *conn.recv(1024)* command, where 1024 specifies the maximum number of bytes to receive at once.
8. The server opens the file *'C.jpg'* in binary read mode (*'rb'*).
9. The server reads the file in chunks of 1024 bytes using *f.read(1024)*.
10. The server sends each chunk to the client using *conn.send(l)*.
11. The loop continues until the entire file is sent.
12. The file is closed after sending the entire content.
13. The connection with the client is closed using *conn.close()*.
14. The script exits the loop and finishes execution.

The breakdown of the client-end visual image transfer script – '*Client_Visual_Camera.py*' executed on the laptop is as follows:

1. The script creates a client-side socket object using *socket.socket()*, which is used for establishing connections with the server.
2. The host variable is defined to specify the IP address of the server to which the client wants to connect. In this case, it is set to 192.168.8.139.

3. The port variable is defined to reserve a specific port number on which the client will attempt to connect to the server. It is set to 5020.
4. The client initiates a connection to the server using *s.connect((host, port))*, where the IP address and port are provided as a tuple to the *connect()* method. This step establishes the connection with the server.
5. The client opens a file *'Visual_Img.jpg'* in binary write mode (*'wb'*) on the client-side to save the data received from the server.
6. The client enters a loop to receive data from the server in chunks.
7. The client receives data from the server in chunks of 1024 bytes using *s.recv(1024)*. This method call blocks until data is received.
8. If the received data is empty (i.e., the end of the file is reached), the loop breaks, indicating the end of data transmission.
9. The client writes the received data to the file using *f.write(data)*.
10. The client closes the file after receiving the entire content.
11. The client closes the connection with the server using *s.close()*, terminating the communication between the client and server.

Below is the list of python files that indicate a server-client socket connection between the laptop and the microcomputer.

Table 14: Python Socket Files

Executable File on Laptop	**Server/Client**	**Jetson: J Laptop: L**	**Server/Client Executable File**	**Server/Client Specific Python File Name**	**Port**	**Purpose**
Visual_Img.bat	Server	J	Image_Save_Visual.sh	Server.py	5020	To transfer the captured visual image.
	Client	L	Visual_Image_Client.bat	Client_Visual_Camera.py		
Thermal_Img.bat	Server	J	Image_Save_Thermal.sh	Server_Thermal.py	5021	To transfer the captured thermal image.
	Client	L	Thermal_Image_Client.bat	Client_Thermal_Camera.py		
Corrosion.bat	Server	J	Corrosion_Processing.sh Corrosion_Server.bat (L)	Server_Corrosion_After_Run.py	5024	-Transfers the excel file generated after running the AI model for corrosion detection
	Client	L	Corrosion_Client.bat	Client_Corrosion.py		
Crack.bat	Server	J	Crack_Processing.sh Crack_Server.bat (L)	Server_Crack_After_Run.py	5023	-Transfers the excel file generated after running

	Client	L	Crack_Client.bat (L)	Client_Crack.py		the AI model for crack detection
Bolt.bat	Server	J	Bolt_Processing.sh Bolt_Server.bat (L)	Server_Bolt_After_Run.py	5022	-Transfers the excel file generated after running the AI model for bolt issue detection
	Client	L	Bolt_Client.bat	Client_Bolt.py		
Crop_Visual_Img.bat	Server	L	Crop_Visual_Server.bat	Crop_Visual_Server.py	5050	-Transfers the cropped image coordinate text file
	Client	J	Crop_Visual_Transfer.sh Crop_Visual_Client.bat (L)	Client_Crop_Visual.py		

6.11 Image Splitting at Microcomputer End

The image *C.jpg* is used for running the AI models. The corrosion and crack AI models are based on the AlexNet architecture. The AlexNet model requires input images of size 227x227 pixels. To achieve this, python scripts are used to split the original *C.jpg* image into multiple smaller sub-images, each of size 227x227 pixels, which are then used as input for the AI models. This approach allows the AI models to analyze different sections of the original image individually, making it suitable for classification.
Below are the python files and the respective image path being the split images are placed.

Table 15: Python Splitting Files

AI Model	File Name	Sub Image Path
Corrosion	split_jetson_2.py	/home/nddot/Run/Submission_nov_2/Dataset/test/
Crack	split_jetson.py	/home/nddot/Run/crack/datanewc/test/

A brief overview of the two split scripts is given below:

1. The script introduces the infile variable to represent the path of the input image file *C.jpg* located at *'/home/nddot/IMAGE_CAPTURES/RUN_FOLDER/C.jpg'*.
2. The variable *savedir* is defined, pointing to the directory where the split sub-images will be stored.

3. The *start_pos* variable is initialized with the starting position (top-left corner) for cropping the image. It is set to (0, 0).
4. To define the size of the sub-images, the *cropped_image_size* variable is used, with a width and height of 227 pixels each.
5. The script opens the image using the *Image.open()* method from the PIL library and assigns it to the variable *img1*.
6. The width and height of the original image are obtained using the size attribute of the *img1* object.
7. A *frame_num* variable is initialized to keep track of the cropped sub-images, starting from 0.
8. Utilizing nested loops, the script iterates through the original image, cropping it into smaller sections measuring 227x227 pixels using the *img1.crop()* method.
9. For each split sub-image, a filename is created based on the frame number and the original image filename without the extension.
10. Each split sub-image is saved in the specified directory using the *crop.save()* method, with filenames like *C001.png*, *C002.png*, and so on, according to the frame number.
11. After saving each sub-image, the frame number is incremented to ensure unique filenames for subsequent sub-images.

6.12 Image Cropping at Microcomputer End

In the smart interface, inspectors are provided with the tool to crop the captured images from the visual sensor. Image cropping allows the inspector to define a specific area of interest within the captured image. This area could contain the targeted area of defect that would require classification.

The advantage of this cropping feature is two-fold:

Faster Processing: Instead of analyzing the entire image, the system only needs to process the cropped area. This saves time and speeds up the inspection process.

Efficient Resource Use: When running AI models or performing analyses, the system allocates computing resources more effectively since it's working with a smaller, focused image area.

Following the inspector's image cropping, the precise cropping coordinates, encompassing the top left and bottom right points, are transferred from the laptop to the microcomputer to initiate processing. These coordinates are stored in the /home/nddot/IMAGE_CAPTURES/VISUAL_CROP/1.tx" file.

The breakdown of the crop script – '*Crop_Cor_to_img.py*' executed on the microcomputer is as follows:

1. The script starts by importing necessary libraries: *glob*, *os*, *re* (for regular expressions), *PIL* (Python Imaging Library for image processing), *shutil*, and *cv2* (OpenCV).
2. It defines the *folder_path* and *file_type* to locate JPEG image files in a specific directory.

3. The script uses *glob.glob()* and *max()* to find the most recently created JPEG image file in the specified folder.
4. The found file is copied to another directory named '*home/nddot/IMAGE_CAPTURES/VISUAL_CROP'*.
5. It locates the copied image in the *VISUAL_CROP* directory, then opens and reads a text file named 1.txt associated with the image.
6. The script attempts to parse the content of '*1.txt'* using a regular expression to extract numerical values that represents cropping coordinates.
7. It resizes the image to dimensions that are multiples of 227.
8. It crops the image based on the specified coordinates *(x, width, y, height)*.
9. It saves the cropped image in the *'/home/nddot/IMAGE_CAPTURES/RUN_FOLDER/C.jpg'* directory.
10. If no valid coordinates are found in the text file, or if there is an error reading the text file, appropriate messages are printed.

6.13 AI Processes – Corrosion

The image classification-based Alex Net model has been used for corrosion detection. A total of 9254 images have been used for training the model. It is a pretrained model in Pytorch framework in Python. All the images are annotated into two classes; with corrosion (WC) and without corrosion (WOC).

The breakdown of the corrosion script – '*Corrosion.py*' executed on the microcomputer is as follows:

1. Image augmentation has been done by using transform. RandomsizedCrop/RandomRotation/RandomHorizontalFlip/CenterCrop
2. Then the augmented image converted to torch tensor of values 0 and 1 by the function transforms.ToTensor().
3. All the train and validation dataset have been saved into subfolders named "train" and "valid" respectively. These subfolders are saved into a main folder with the name "Dataset".
4. os.path.join() has been used to concatenate the paths of main folder to connect the training and validation folders.
5. The listdir() function provided by the os module has been used to get the number of classes.
6. To Load data from folders ImageFolder() has been used.
7. The label of the images has been obtained by using class_to_idx.items().
8. Dataloader module has been used to iterate the loaded data.
9. The final layer of Alex Net has been modified to work as a binary classifier.

6.14 AI Processes – Crack

Similar to corrosion, the image classification-based Alex Net model has been used for crack detection. About 200 images with 1500 sub images have been used for training the model. It is a pretrained model in Pytorch framework in Python. All the images are annotated into two classes; with crack (WCrack) and without crack (Without Crack). In the following, all steps were summarized.

- Image augmentation and superimposed images has been done by transferring images colors and crack, as well as crack direction.
- Data was divided into two main datasets (train and validation) with two subfolders.
- We generated a balanced data set (791 crack images and 791 uncrack images) for crack with lab data since there is no available data set for crack.
- We also added about 300 new images based on recent crack images and our inspection. This folder (caltec256subset) now contains:

- 845 crack images (folder (a))
- 845 uncrack images (folder (b))

- os.path.join() has been used to concatenate the paths of main folder to connect the training and validation folders.
- The listdir() function provided by the os module has been used to get the number of classes.
- To Load data from folders Image Folder () has been used.
- The label of the images has been obtained by using class_to_idx.items().
- Data loader module has been used to iterate the loaded data.
- The final layer of Alex Net has been modified to work as a binary classifier.
- Test algorithm in real time: this part contains a script file in Python which usually works to call images in test folder and order it based on subimages names, as well as calling trained model to predict class labels of each sub images. This part was designed in such a way that the inspectors can call model and sub images in real time and summarize all results with classes in excel file.
- This excel was used as an input file to get main information to inspectors about the location of crack.
- Finally, the test sub images were used the performance algorithms. To do this, the train model was used to predict the class of each sub image.
- The result was saved in excel files (crack.xls) in two subclasses with image's names in real time.
- In total, the crack folder contains all images related to the train dataset, trained model, test script to show algorithm's performance in real time, and all sub images related to the test dataset.

6.15 AI Processes – Bolt Issue

Just to reduce system complexity in real time as well as make sure that all code is compatible with Jetson, the same liberty (Py torch) was used again for bolt detection. Based

on the problem, bolt considered as small object and using object detection algorithm is one of the best options based on previous studies.

- The following model builders can be used to instantiate a Faster R-CNN model, with pre-trained weights.
- All the model builders internally rely on the torchvision.models.detection.faster _ rcnn.FasterRCNN _ Resnet base class.
- All images were annotated based on bounding box location.
- The folder contains three excel files for all annotated boxes for training and testing, and all images.
- After training, the model was saved in the same folder.
- In test mode, the inspectors call train models with test images just to predict the defected bolt location.
- After prediction, the carinated related to defected bolt was saved in text file.

Appendix A Payload Interface Design

1.Materials and Manufacturing

1.1 Stratasys F370 Composite 3D Printer

F370 Highlights

- Maximum build area of 14 x 10 x 14 inch (356 x 254 x 356 mm)
- Material Bays: 2 model, 2 support
- Insight Software Package
- Touchscreen Graphical User Interface
- Auto changeover capabilities
- Wi-Fi capabilities
- Three USB ports (2 in front, 1 in back)
- Camera for remote monitoring

1.2 ABS M30 Black/Ivory Material Properties

stratasys

Physical Properties

Values are measured as printed. XY, XZ, and ZX orientations were tested. For full details refer to the Stratasys Materials Test Report (immediate download upon clicking the link). DSC and TMA curves can be found in the Appendix.

Table 4. ABS-M30 Physical Properties

Property	Test Method	Typical Values	
		XY	XZ/ZX
HDT @ 66 psi	ASTM D648 Method B	103.8 C (218.9 F)	
HDT @ 264 psi	ASTM D648 Method B	99.9 C (211.7 F)	
Tg	ASTM D7426 Inflection Point	105.2 C (221.4 F)	
Mean CTE	ASTM E831 (40 °C to 140 °C)	60.77 µm/[m*°C] (33.76 µin./[in.*°F])	
Mean CTE	ASTM E831 (40 °C to 80 °C)		58.65 µm/[m*°C] 32.58 µin/[in*°F]
Volume Resistivity	ASTM D257	> 6.75*10^14 Ω*cm	
Dielectric Constant	ASTM D150 1 kHz test condition	2.64	2.78
Dielectric Constant	ASTM D150 2 MHz test condition	2.49	2.61
Dissipation Factor	ASTM D150 1 kHz test condition	0.003	0.005
Dissipation Factor	ASTM D150 2 MHz test condition	0.004	0.007
Specific Gravity	ASTM D257 @23 °C	1.05	

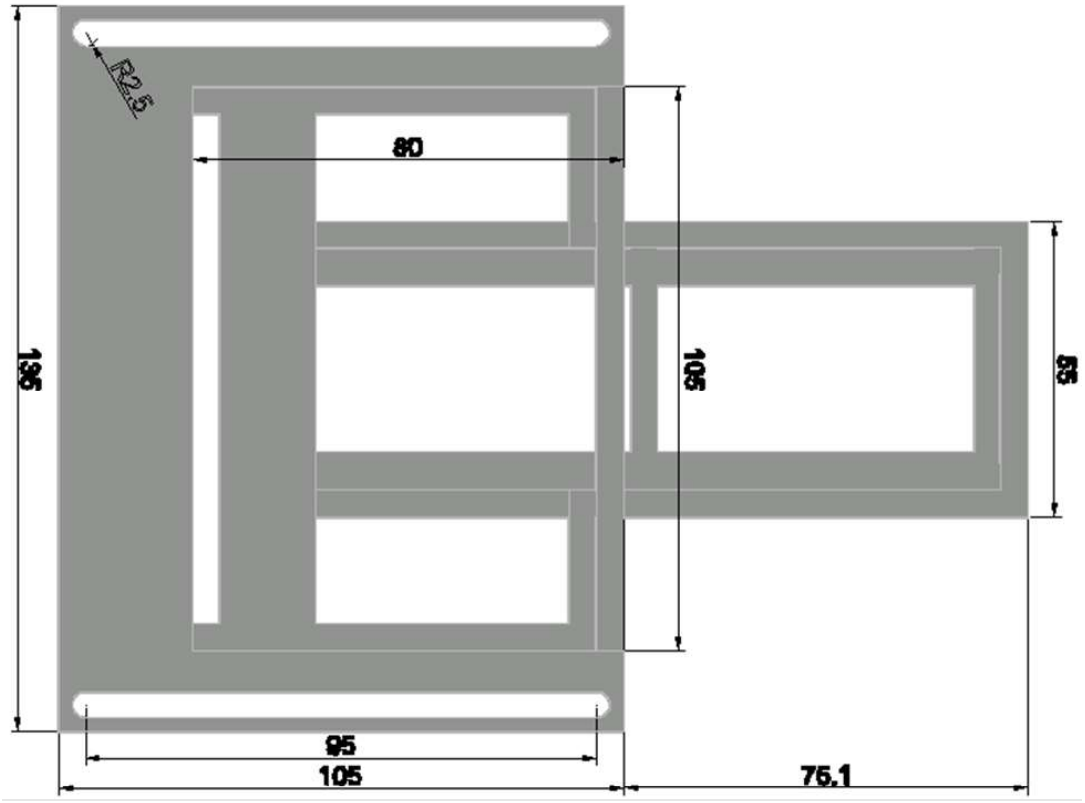
Table 6. ABS-M30 Mechanical Properties (F770)

		XZ Orientation[1]	ZX Orientation[1]
Tensile Properties: ASTM D638			
Yield Strength	MPa	32.5 (1.7)	23.1 (1.3)
	psi	4720 (250)	3350 (190)
Elongation @ Yield	%	2.1 (0.1)	1.6 (0.2)
Strength @ Break	MPa	27.6 (2.4)	22.9 (1.2)
	psi	4000 (350)	3310 (170)
Elongation @ Break	%	4.5 (1.2)	1.6 (0.2)
Modulus (Elastic)	GPa	2.00 (27)	1.78 (29)
	ksi	290 (3.9)	258 (4.1)
Flexural Properties: ASTM D790, Procedure A			
Strength @ Break	MPa	No Break	37.8 (4.1)
	psi	No Break	5480 (590)
Strength @ 5% Strain	MPa	58.1 (2.2)	-
	psi	8430 (320)	-
Strain @ Break	%	No Break	2.2 (0.3)
Modulus	GPa	2.17 (0.03)	1.84 (0.06)
	ksi	315 (4.9)	267 (8.1)
Impact Properties: ASTM D256, ASTM D4812			
Notched	J/m	91.0 (17)	21.7 (3.7)
	ft*lb/in	1.71 (0.31)	0.406 (0.07)
Unnotched	J/m	423 (96)	62.9 (134)
	ft*lb/in	7.92 (1.8)	1.18 (0.3)

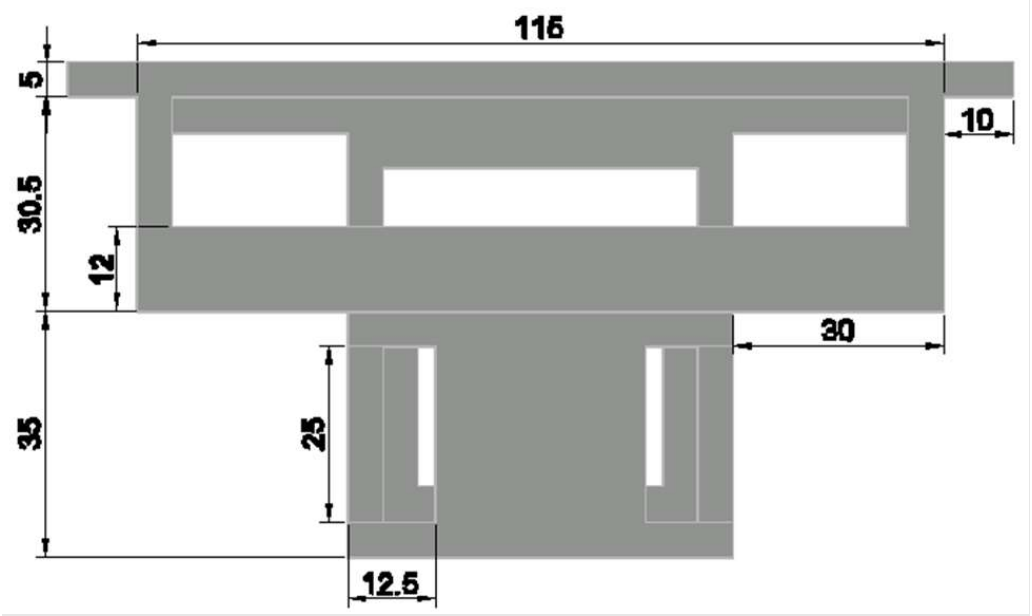
(1) Values in parentheses are standard deviations.

2 Component Design

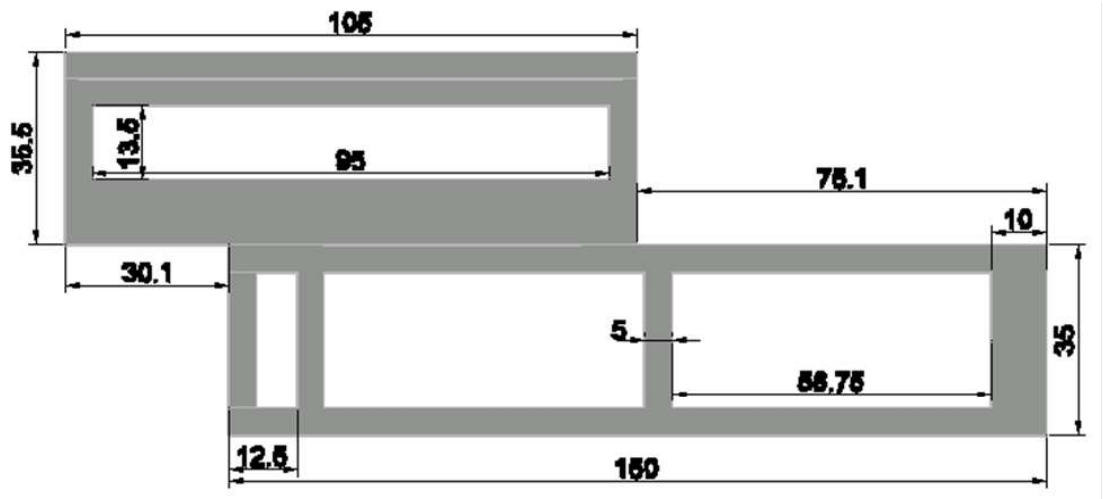
2.1 Battery and Jetson Housing



Top-Down View



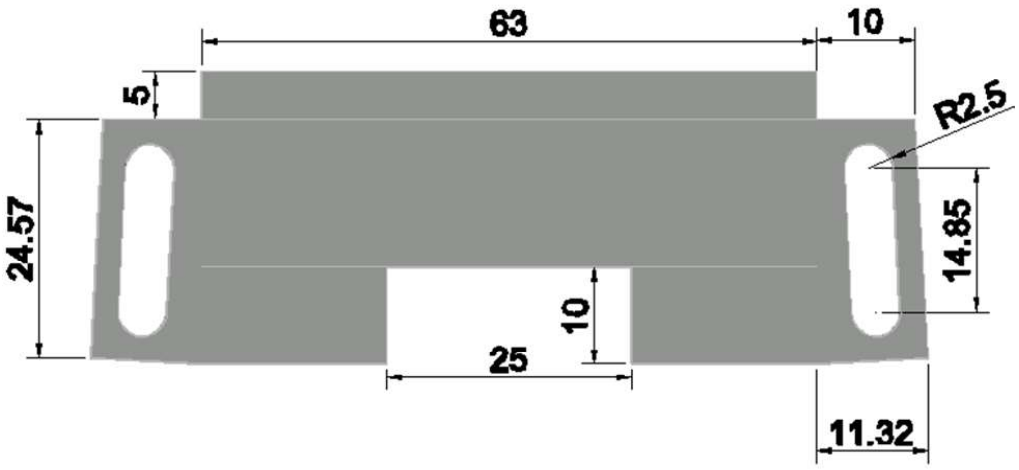
Front View



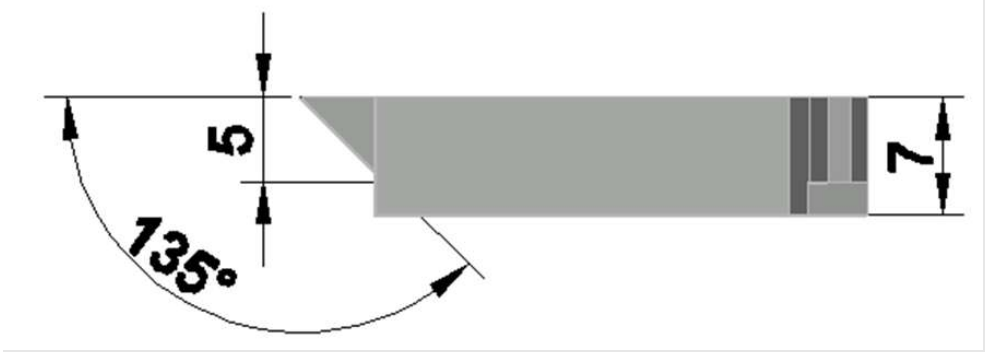
Side View

2.2 Battery and Jetson Housing Brackets

Front Upper Bracket

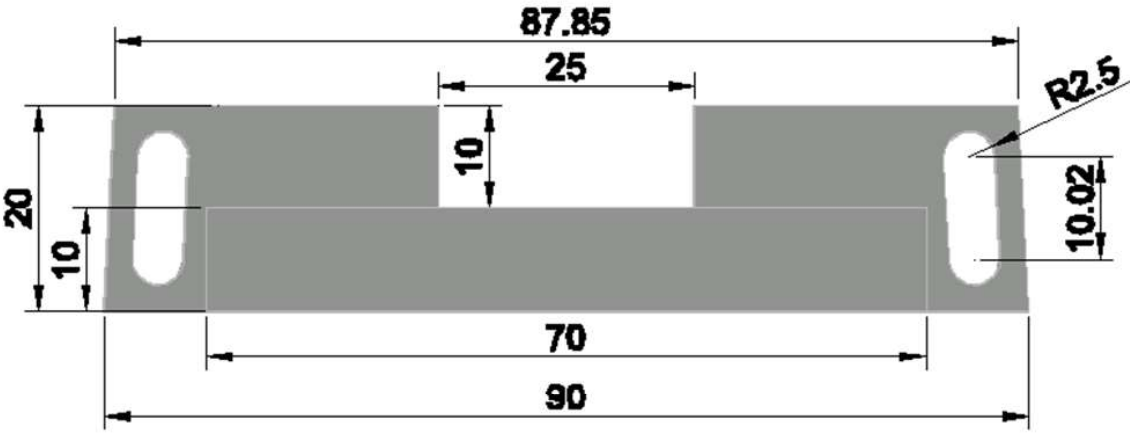


Top-Down View



Side View

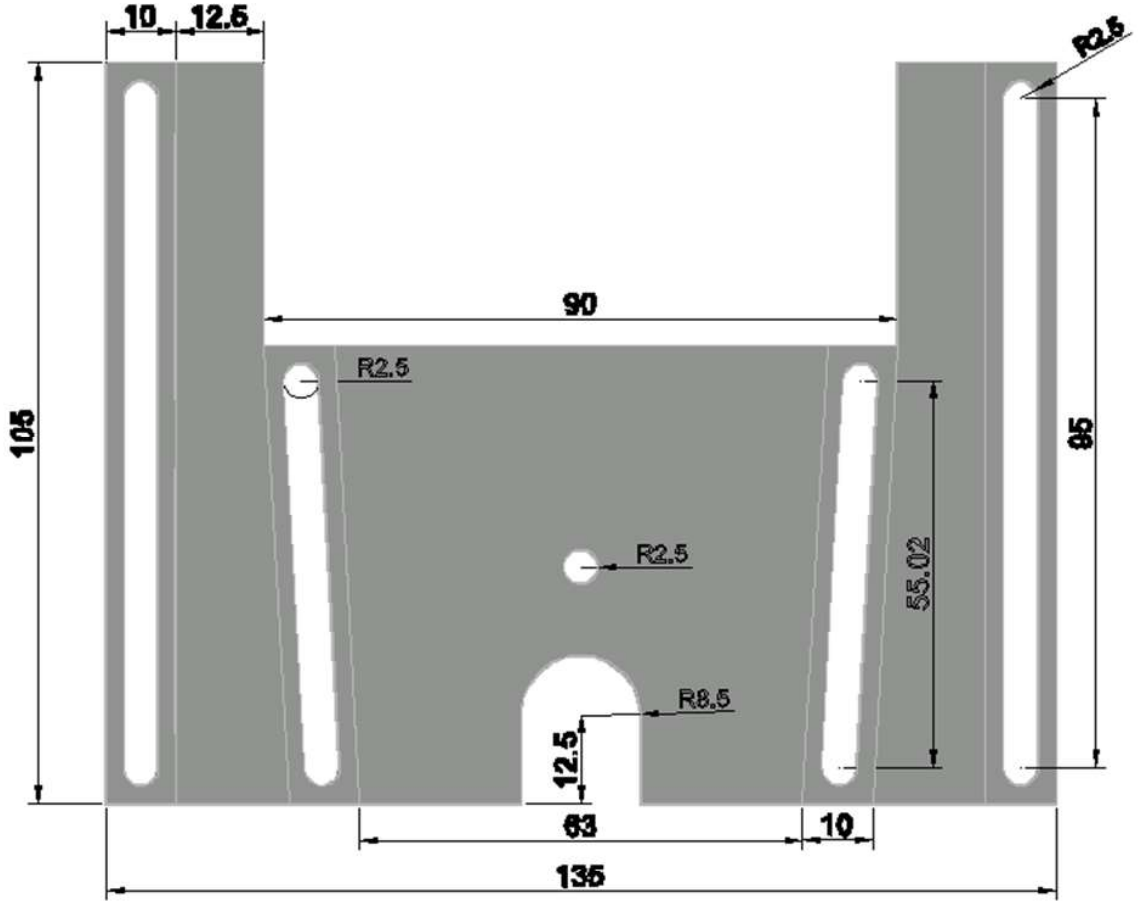
Rear Upper Bracket



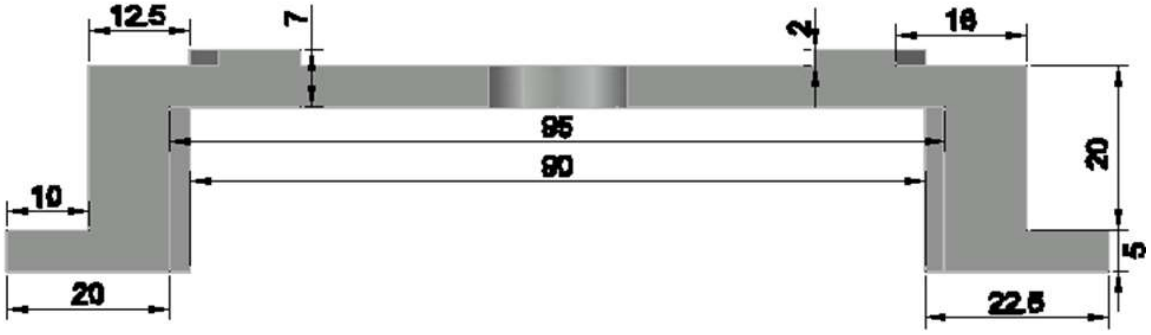
Top-Down View

Back View

Lower Bracket

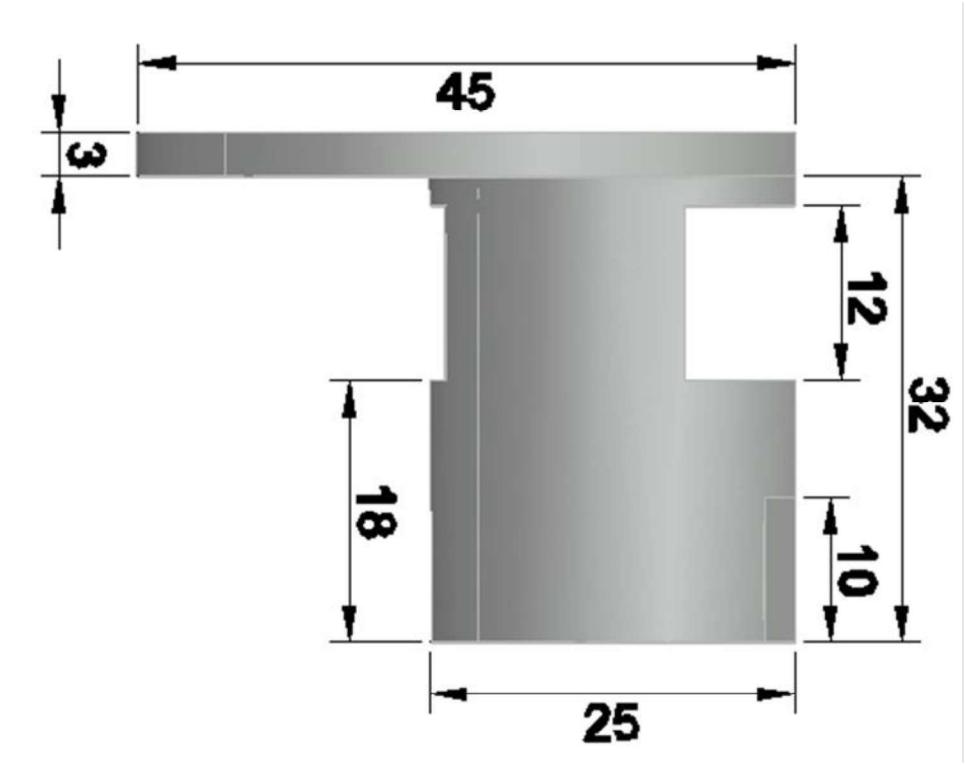


Top-Down View

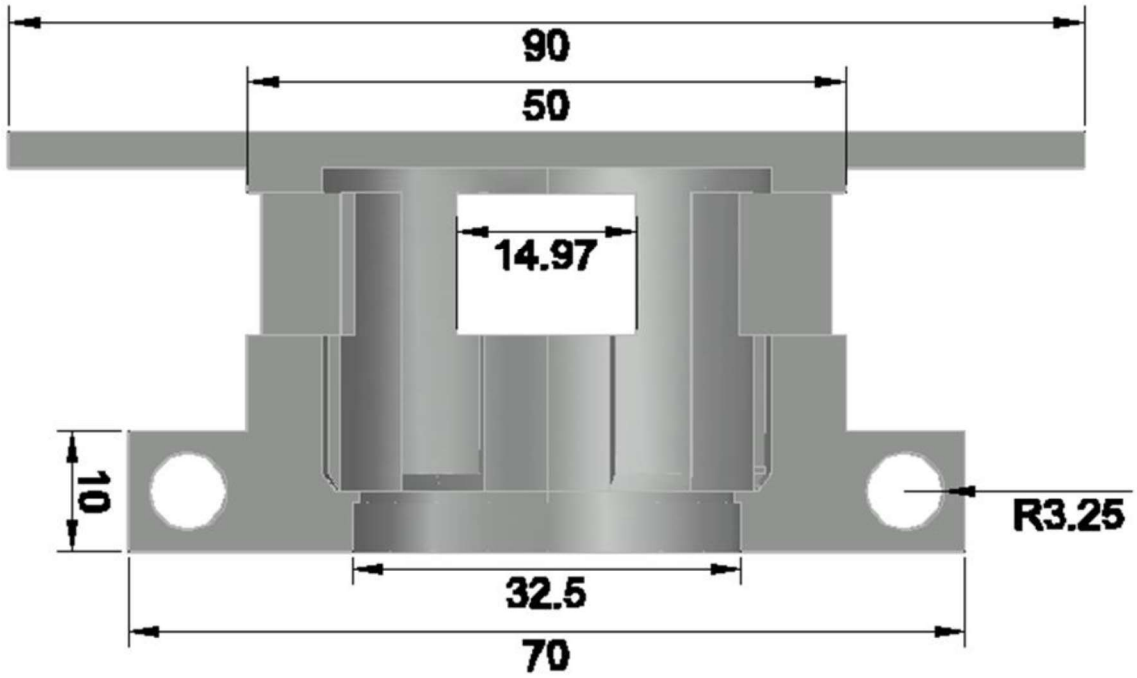


Front View

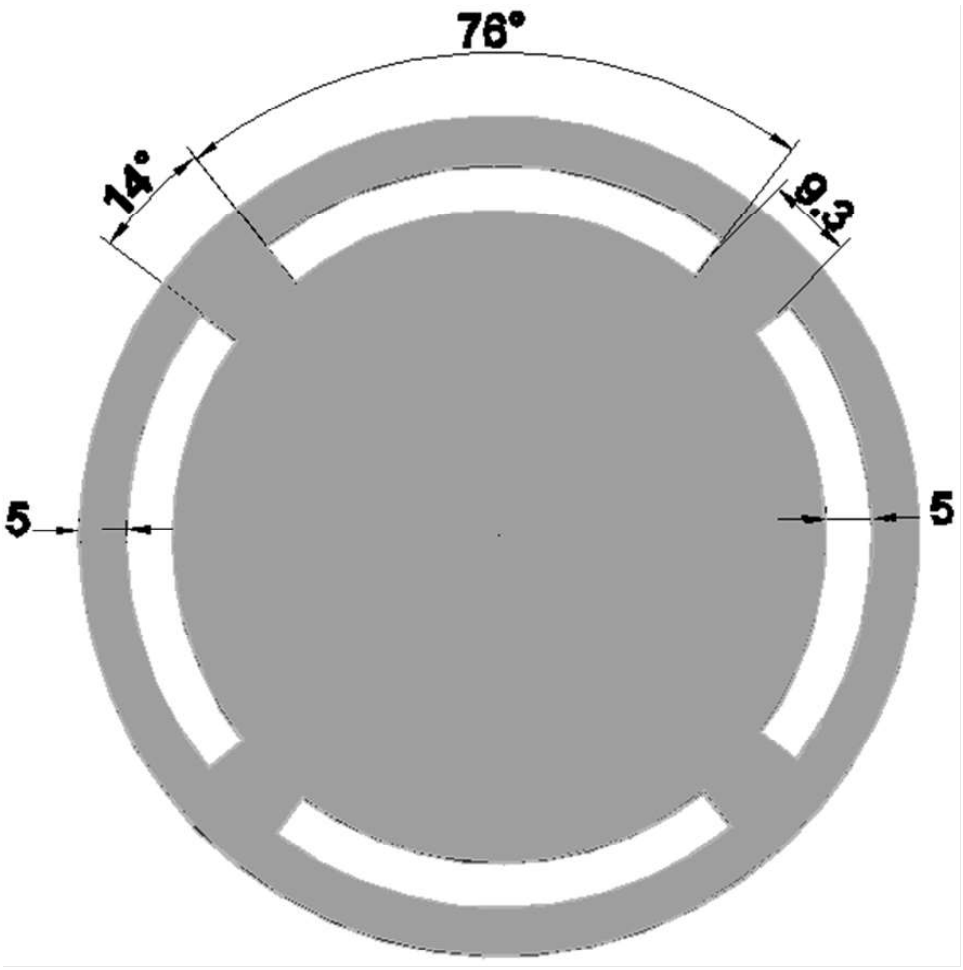
2.3 Gimbal Housing



Side Half-Section View



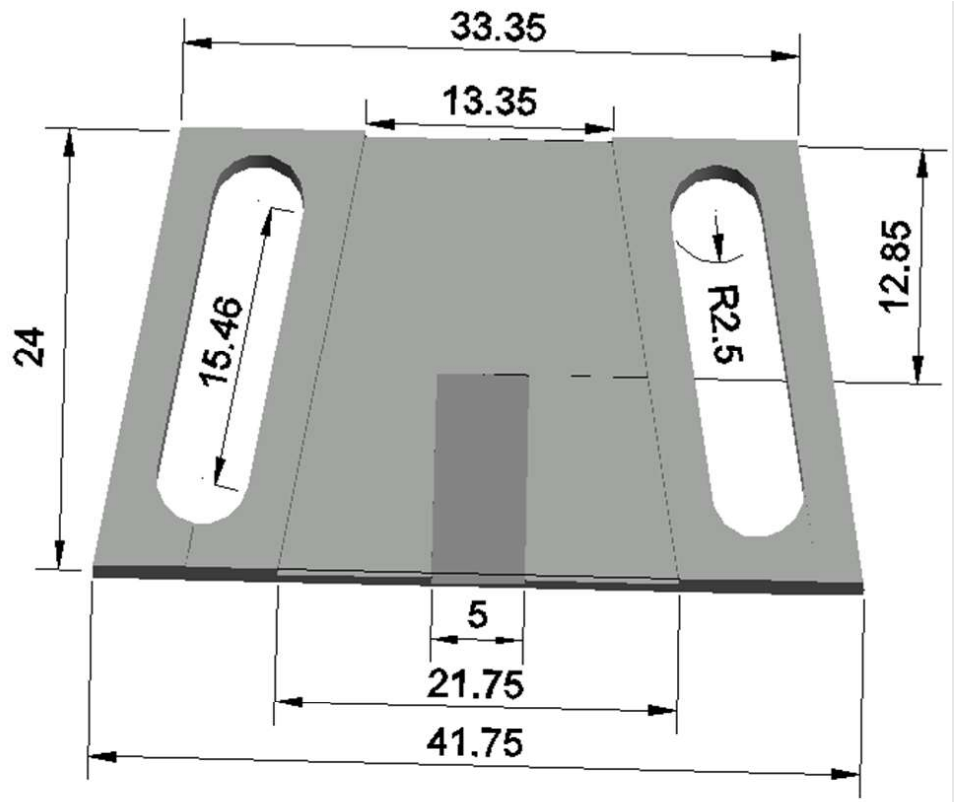
Front Half-Section View



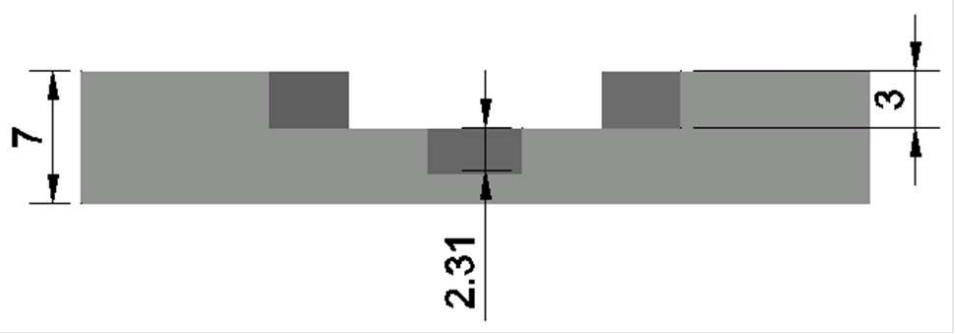
Top-Down View

2.4 Gimbal Housing Brackets

Rear Gimbal Bracket

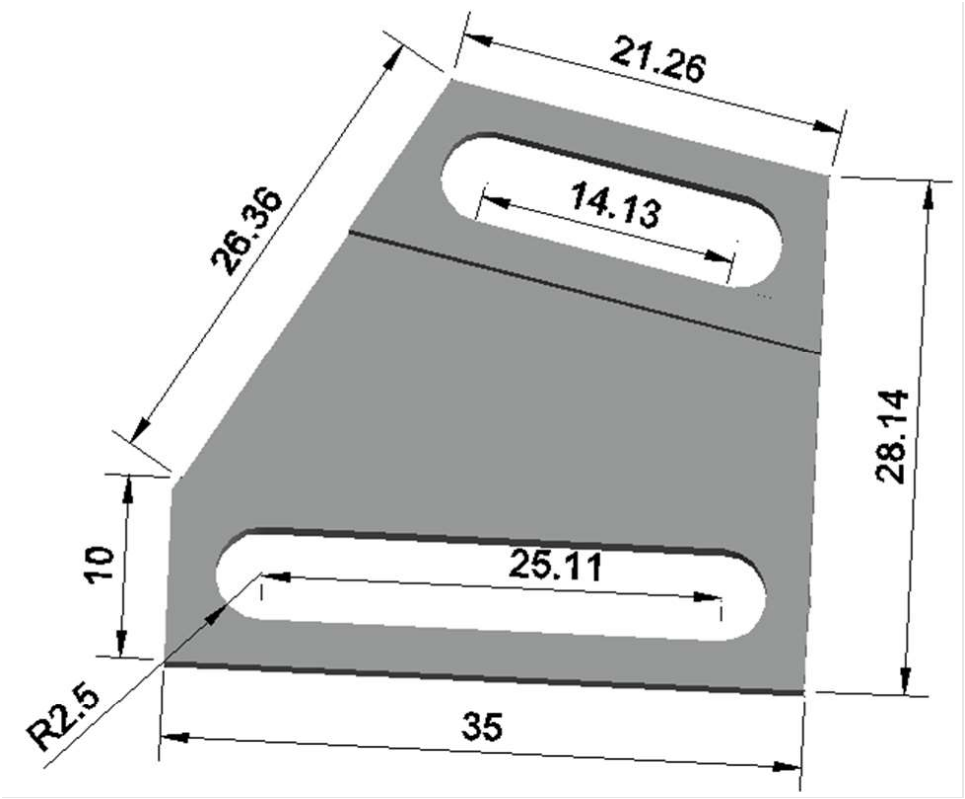


Top-Down View

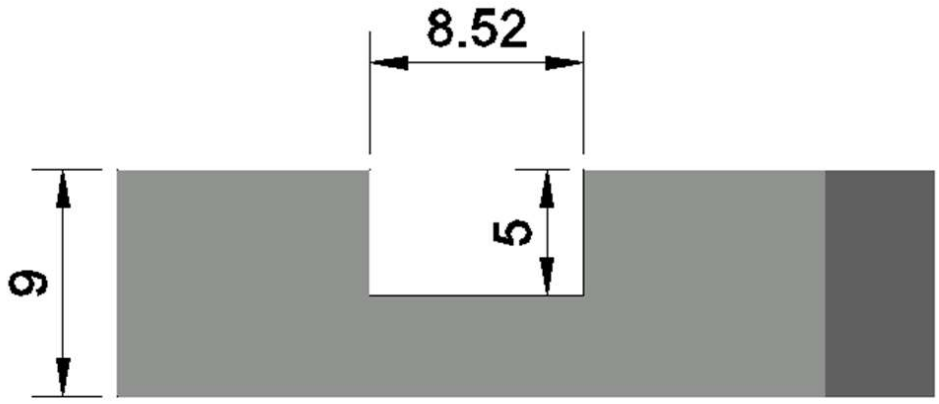


Back View

Front Gimbal Brackets (same model, just mirrored)

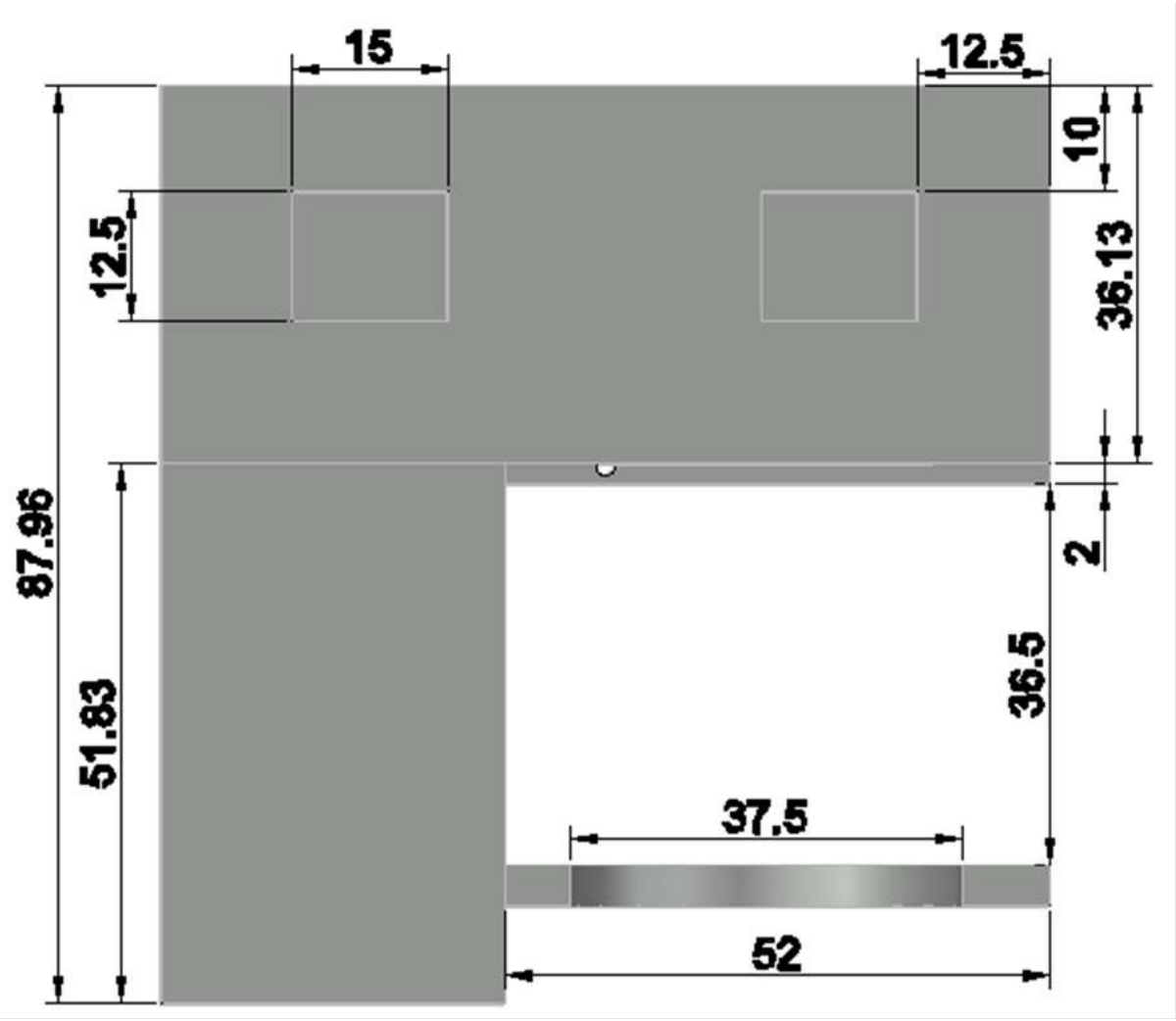


Top-Down View

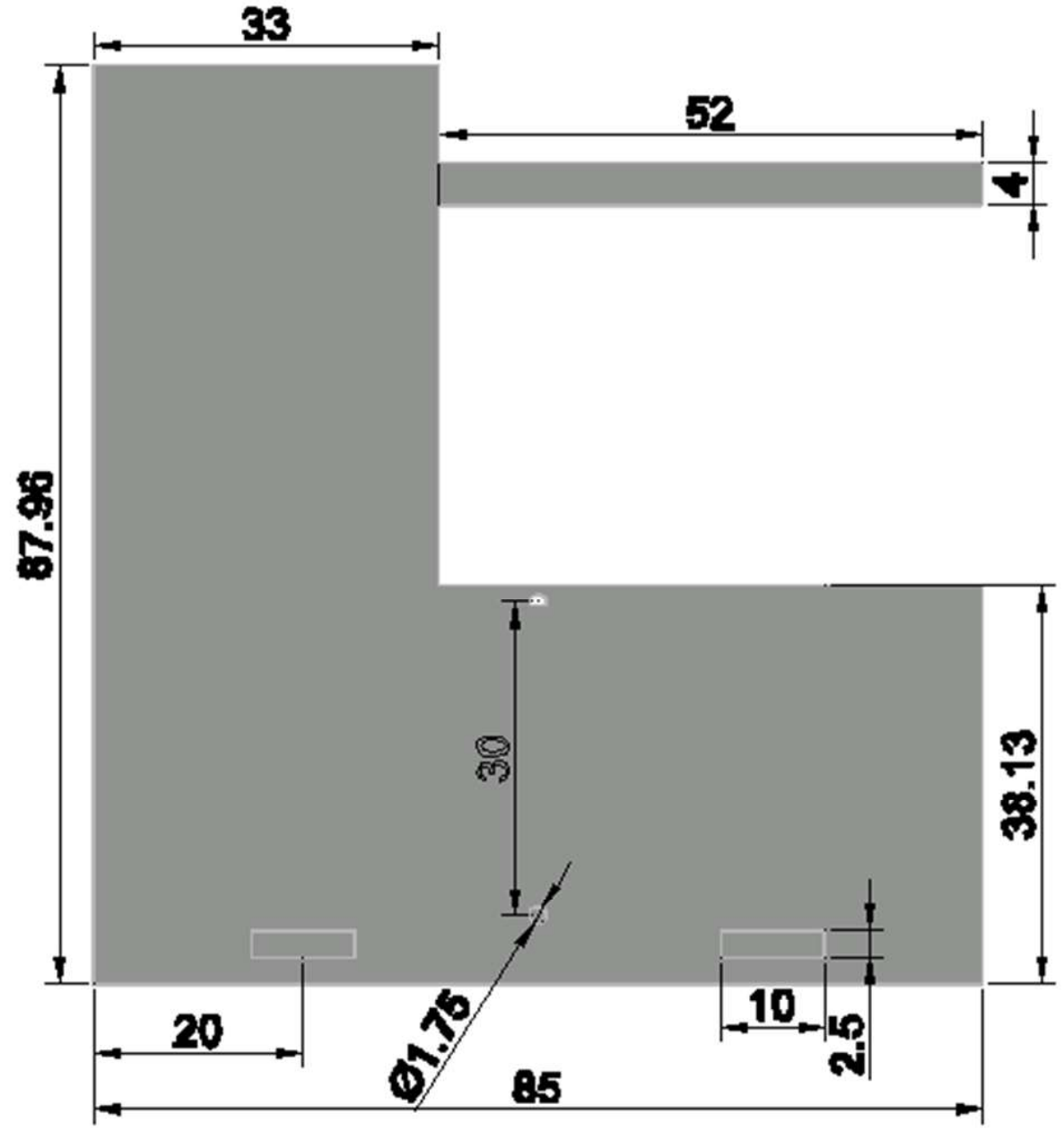


Side View (Smaller Side)

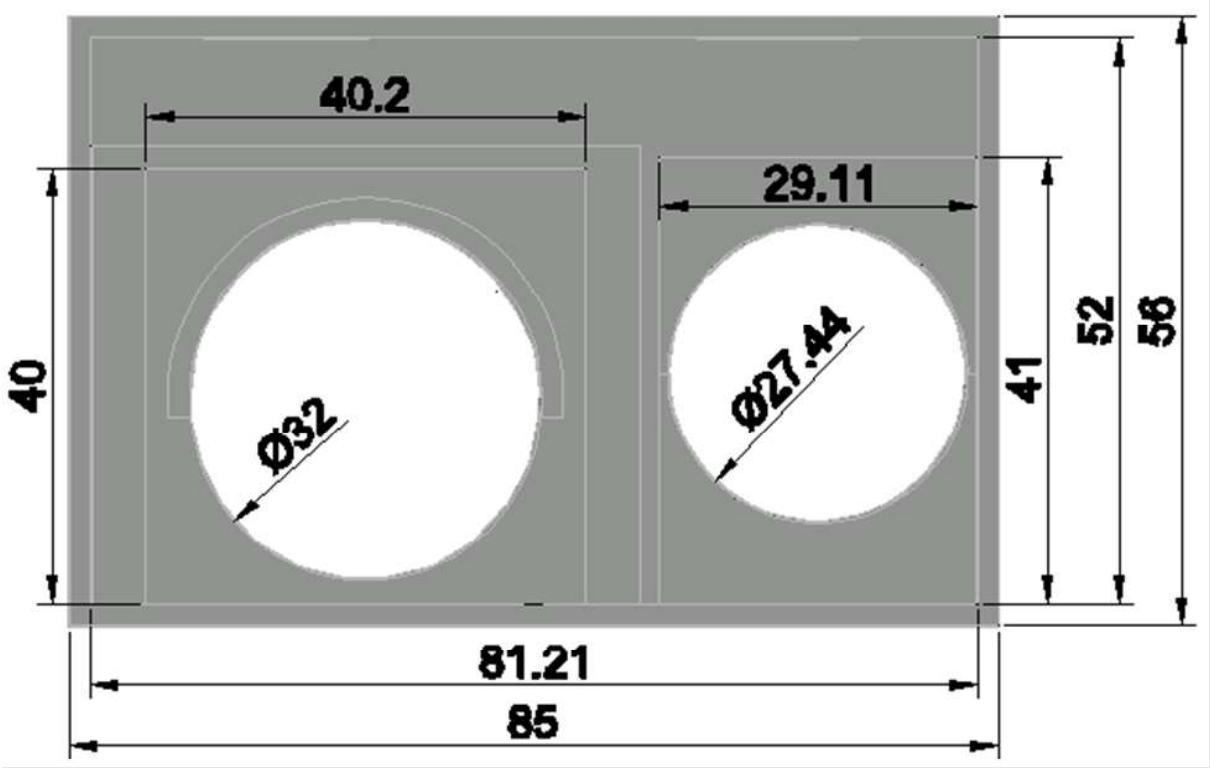
2.5 Camera Housing



Top-Down View

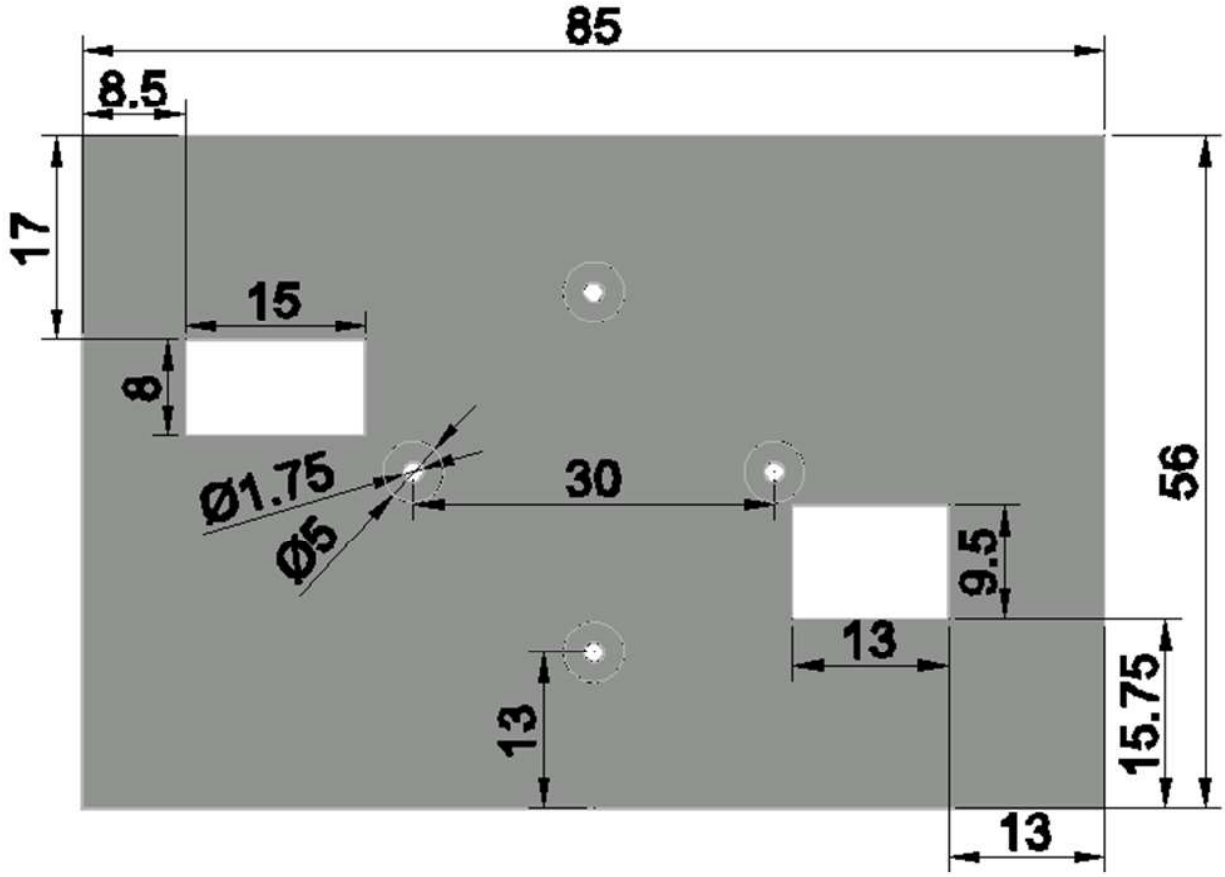


Bottom View

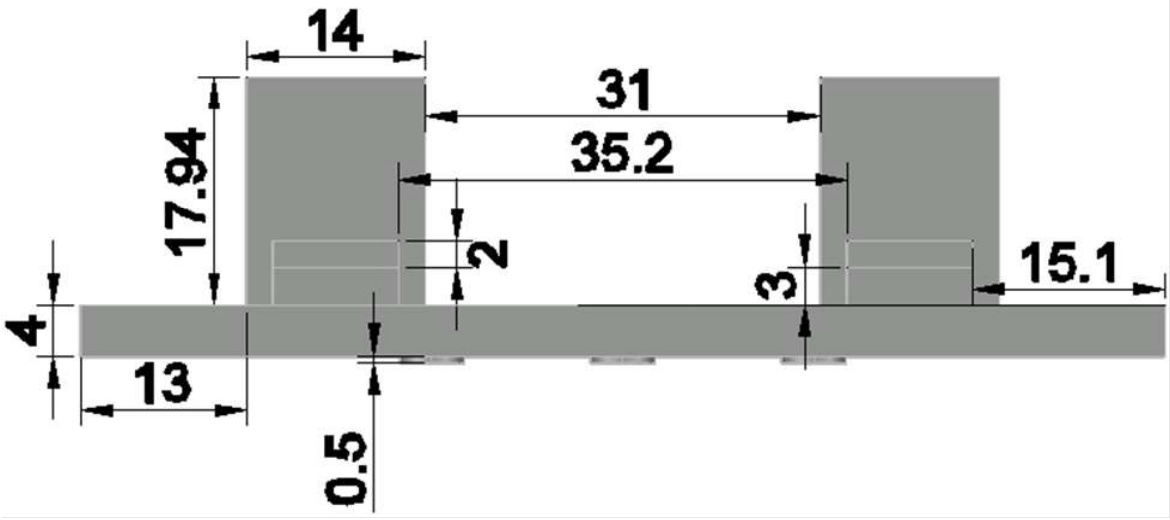


Rear/Interior View

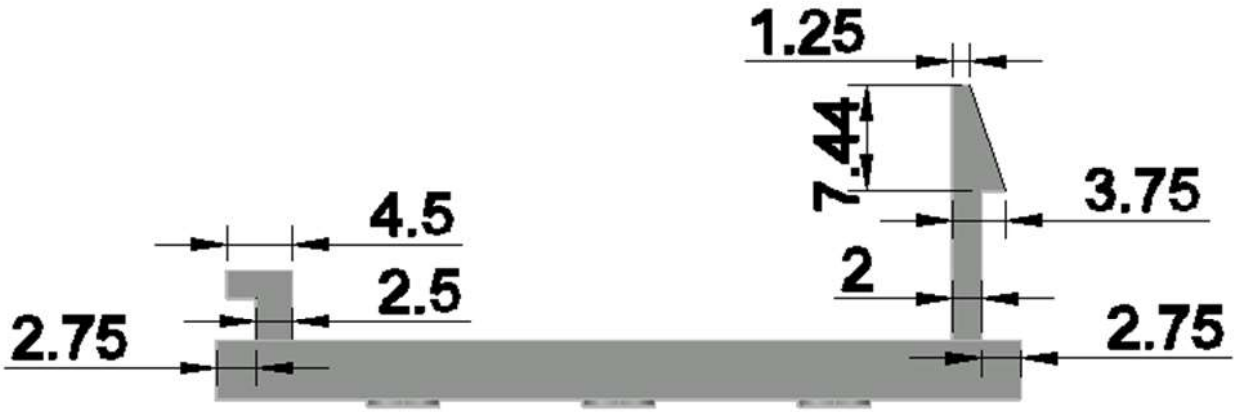
2.6 Camera Housing Clip-on Cover



Exterior-Facing Side View



Bottom View



Side View