Autonomous Railway Defect Detection

by

David Xia

A thesis

presented to the University of Waterloo

in fulfillment of the

thesis requirement for the degree of

Master of Applied Science

in

Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2023

Author’s Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Railway defects can lead to enormous economic and human losses. Among all types of the defects, surface defects are the most common and prominent. Although various non-destructive methods can be used to detect railway defects, optical inspection is considered the most suitable for surface defects. The aim of this project is to utilize the commercial drones as a platform for conducting visual inspections. The project is divided into two parts: flight control of the drone and the development of an algorithm to identify defects.

For the flight control of the drone, two methods are proposed: pre-set flight path and the real-time track tracing. The pre-set flight path is the simplest approach because it is a built-in function of the drone. However, it is heavily dependent on the GPS coordinate system, which can be problematic in areas with weak GPS signals. The real-time track tracing, on the other hand, requires the development of an algorithm to locate the railway track. This can be achieved through either pattern matching or edge detection. Of the two methods, edge detection is more promising since it produces more consistent result and can be achieved by incorporating various image filters.

Numerous attempts have been made to use optical-based non-destructive testing (NDT) methods for detecting railway defects. However, reliable and accurate interpretation of the test data is crucial in the NDT practice. There are many sources of errors that can negatively impact the results of NDT, with human errors being the most unpredictable and frequent. The introduction of artificial intelligence (AI) has the potential to address this problem, but the lack of sufficient railway images with various types of defects is the major obstacle to training the AI models through supervised learning. To address the issue of data scarcity, RailGAN model is proposed in this research. The RailGAN model can improve upon the basic CycleGAN model by incorporating an additional pre-sampling stage for railway tracks in the algorithm. Two pre-sampling techniques are tested for the RailGAN model: image-filtration and U-Net. By applying both techniques to 20 real-time railway images, it is demonstrated that U-Net can produce more consistent results in image segmentation across all images while being less affected by pixel intensity values of the railway track. The application of the RailGAN model implemented with U-Net and the original CycleGAN model to the same real-time railway images shows that the original CycleGAN model generates defects in the irrelevant background, whereas the RailGAN model adds synthetic defect patterns only on the railway surface. The defective images generated by RailGAN

model closely resemble real cracks on the railways and can be used for training neural network-based defect identification algorithms.

The proposed solution for defect identification involves the use of the YOLO network, which can detect and localize defects in the images. It is trained using the generated defective railway images and their corresponding labels. Testing the algorithm on 10 images available online, the YOLO-v5 algorithm can identify 90% of all defects in the images.

However, there are still future works that need to be accomplished. For the drone flight control, the algorithm will need to be written in the JAVA language using the mobile SDK. Regarding defect detection, RailGAN will generate a larger amount of images, and the YOLO network will be trained with a larger dataset. The model will also be saved as a JSON file and implemented in the mobile SDK to enable real-time defect detection in the future.

Acknowledgements

The completion of this project could not have been possible without the assistance of so many people. Honestly, these two years have been very difficult for me. I would first thank Professor HJ Kwon for giving me this opportunity to learn about artificial intelligence. He has been giving me endless support and constant guidance to me during my two-year study. Second, I would like to thank Dr. Yanjun Qian for his timely guidance and effort to teach me the fundamentals of this project especially the knowledge of CNN network. I would also like to thank my family and friends for giving me encouragement and motivation whenever I need them.

Here, I would also like to mention Mohamed Traore and the Roboflow team for giving me additional access for YOLO network training on their website, and also provided me with additional information about the project I am working on.

My thanks and gratefulness also go to Professor. Soo Jeon and Professor Kamyar Ghavam for reading and providing feedback to my thesis as well as the progress of project.

I am very grateful for all the help I have been provided with.

Thank you.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Motivation

According to an annual report by Via Rail Canada Inc. [1], over 5 million passengers chose to travel by rail in 2019. Even during the pandemic year, the ridership across Canada was still over 1 million [2]. In addition to the passenger trains, cargo services through railways are also essential in day-to-day life, making travelling by rail an irreplaceable way of transportation. Safety plays a vital role in maintaining the working condition of a railway system. Due to the capacity of railway transportation, railway accidents can result in significant financial losses, pose lethal threats to passengers, endanger nearby residents, and wreak havoc on the environment. Among all the serious accidents in railway transportation, derailment is the most common [3], and defective railways are the most frequent cause of derailments. According to a recent report by the Transportation Safety Board of Canada, between 2011 and 2020, there were 959 main-track derailments, 37% of which were caused by track-related issues. There were also over 6000 non main-track derailments, and almost 1/3 of these ascribed to problems with railway tracks [3].

One of the most infamous railway derailment accidents caused by the track defect was the Hither Green rail crash happened on November 5th, 1967, [4]. In that accident, a broken rail caused the leading pair of wheels of the third coach to derail, leading to a derailment of 11 coaches in total. The cause of the broken rail was traced back to a minor fatigue crack located at a bolt hole. The developed crack triggered a triangular piece on the rail to break, eventually leading to the fracture of the rail [5]. The accident resulted in the deaths of 49 passengers and injuries to 79 others on the train. Despite modern inspection technologies, defects on the railways can still be overlooked, resulting in trouble. In 2015, a westbound Canadian National Railway (CN) freight train derailed near Saint-Basile, New Brunswick. The main contributor to the accident was the significant gauge face wear on the high rail in the derailment curve. Unclear criteria in assessing rail conditions are one of the critical factors in overlooking the defect [6].

Therefore, to avoid similar accidents in the future, impartial preventive inspections should be conducted regularly. It is essential to identify problems on the rail surface before they get any worse. Also, the inspections should adhere to a definite standard and minimize human error as much as possible. However, most current inspection techniques require the presence of human inspectors in

the field, disrupting the normal operations of the train. There has been some research progress so far by different labs around the world, which are further reviewed in section 2.1, yet very few of them can achieve reliable autonomous examination in the field.

1.2 Railway Defect Background

Depending on the location of the defects on the rail, rail defects can be categorized as follows: transverse defects, longitudinal defects, web defects, base defects, surface defects and defective weld [7,8]. Among all these categories, the surface defects are of most importance, because they can develop over time and initiate other types of defects. They are also more common and approachable than others. In Table 1, commonly seen surface defects, their locations and the corresponding level of importance are summarized.

Table 1: List of Surface Defects [7, 8]

Name	Definition	Level of importance
Nicked rail	A nicked on the head, web, or base, which is usually caused by broken wheel or dragging equipment	High
Shelling	A progress horizontal separation on the gage side.	High
Flaking	A progressive horizontal separation of the running surface around the gage corner	Low
Slivers	A separation of a metal from the surface of head, web and base	Low
Flowed Rail	A rolling out of trend metal beyond field corner	Low
Burned Rail	A marked by left the intense friction from slipping wheel.	Moderate
Mill Defect	Deformation, cavities, seams, or foreign material found in the head, web or base of the rail	Extreme High
Flattened Rail	A length of railway flattened out across the width of the rail to a depth of less than 1 inch.	Low
Crushed Head	Flattening of railhead by more than 1 inch.	Low
Corrosion	It is the decaying of the material of head, web or base of the rail.	Moderate
Corrugation	A wavelike pattern on the running surface.	Low

Damaged Rail	Broken rails caused by broken, flat or slipping wheels. It can also be cause by derailment	High
Head Checks	Slight separation of metal on gauge side of the rail.	Moderate
Spalling	Separation from parent metal on the rail head.	Low

1.3 Project Objectives

Since surface defects are of utmost importance, the primary objective of this project aims is to achieve autonomous detection of all surface defects. This objective is further broken down into the following sub-objectives:

1. Find the suitable NDT method for detecting all types of surface defects introduced in Section 1.2. This includes identifying the methodology and the platform that can carry out the NDT.
2. Develop a track-finding algorithm that can either guide the platform or support the NDT method for surface defect detection.
3. Generate/find surface defect dataset used for training the defect detection model.
4. Train and improve the model for defect detection to enable recognition of surface defects with a high confidence rate.
5. Test the defect detection algorithm using real surface defect images available online.

1.4 Thesis Outline

The thesis is organized into the following chapters:

Chapter 1 provides an overview of the project motivation, types of surface defects that needs to be tackled, and the objective and sub-objective of this project.

Chapter 2 reviews the background knowledge of current NDT methods that can be used for defect detection of metal products and provides a literature review about application of these NDT methods. Based on the information provided, the most appropriate NDT method to detect surface defect will be selected, and the hardware and platform that carry out the NDT will be proposed.

Chapter 3 introduces railway track finding and proposes different ways to accomplish this task. The results of the test of each proposed method are demonstrated in section 3.6.

Chapter 4 presents a novel way of generating surface defect images for use as the training dataset.

Chapter 5 proposes two different defect identification methods followed by a discussion.

Chapter 6 examines training and validation process of the different CNN networks proposed in chapter 4 and chapter 5, as well as the experimental results of the defect detection algorithm, and a detailed conclusion on the implementation of such defect detection algorithm. This chapter is partly based on the journal published:

Chapter 7 presents the main conclusion of this project, as well as the recommendation for future work.

Chapter 2
Non-destructive testing and Carrying Platform

2.1 Non-Destructive Testing Methods (NDT)

The prevailing non-destructive testing (NDT) methods used in railway inspection include thermography, magnetic flux leakage, visual inspection, ultrasonic testing, and acoustic emission. Each of these methods approaches inspection tasks by using different physics, which makes them excel at detecting different types of defects. To determine the most appropriate method for this project, each method is reviewed in the following sections.

2.1.1 Thermography

The thermographic testing is a type of non-destructive testing that maps the temperature pattern of an object's surface or body, and the output image is used to analyze the health of the structure.[9] To carry out such testing, an infrared (IR) camera is required to measure and display the temperature, then the data is transmitted to a computer for recording or processing.[10] Thermography testing can be further broken down into active and passive testing.[11] Passive thermography directly measures the temperature of the specimen, while active thermography measures the temperature change after applying thermal excitation. Both techniques can offer non-contact, full-field, and fast inspection with a thermogram, but they are limited by the high equipment costs and uncertain conductivity. Some researchers have applied active thermography to railway defect identification. Tuschl et al. used inductive thermography to find the railway defects [12]. The equipment they used included an induction generator (consisting of induction coils) and an infrared camera. They used two types of setups: a static and a scanning setup. In the static setup, both equipment and specimen were held in place, while in the scanning setup, both equipment and specimen were moving at a constant speed. The power required in both setups was between 5 to 10 KW. In both settings, the team could identify the crack by differentiating the temperature increase between the intact area and the crack in the specimen. Another research using thermography for railway defect detection was done by Peng et al. [13]. They developed the lock-in thermography to model the rail squat defect. In their experimental setup, they used thermal lamps with a sinusoidal stimulus to excite the rail sample. An infrared camera was implemented to record the phase and amplitude of the response. As a result, the team proposed that a 0.2 Hz excitation rate was a suitable frequency for defect detection, and a higher

excitation (1Hz) could be used to locate the squat. Netzelmann et al. applied induction (eddy current) thermography testing method to detect surface crack of the railway [14]. The process of the experiment involved generating the eddy current pulse with 50 to 500 ms length to the railway, and the infrared camera recorded the radiation change after the heating pulse was applied. The team used this technique and applied such device on a moving test car, and the railway surface crack could be successfully detected when test car was travelling between 2 and 15 km/h. This principle was also applied to detect wheel defect In this research, a robot held the infrared camera and the inductor close to the rotating wheel. While wheel was rotating, the camera scanned the wheel surface, and provided information such as the crack position and crack depth.

2.1.2 Ultrasonic Testing (UT)

UT typically utilizes acoustic waves with frequency range between 400 kHz to 25 MHz to conduct NDT measurements [9,15]. By converting high frequency electrical energy into mechanical waves using transducers, acoustic waves are emitted into the object.

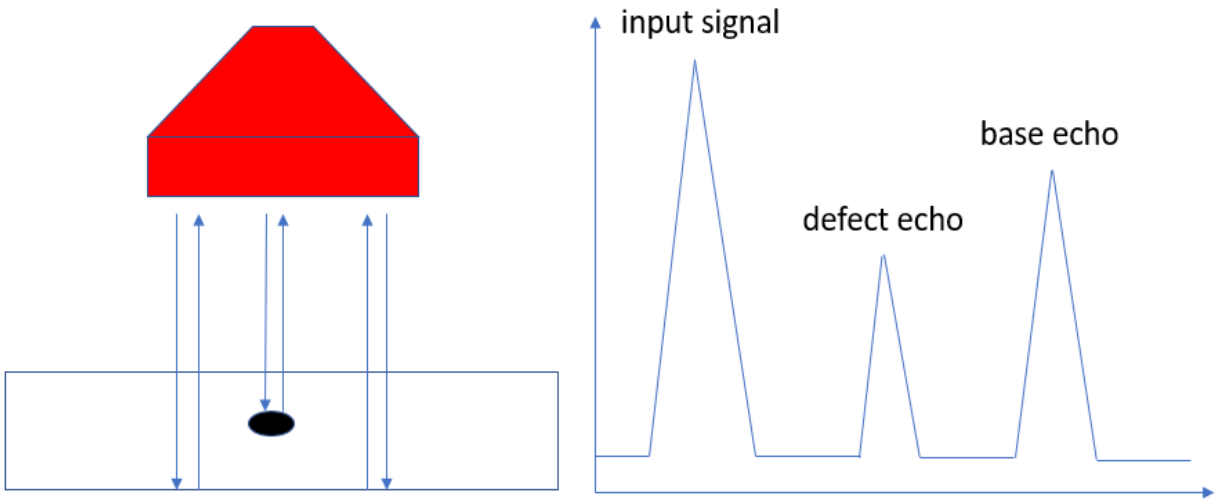

Figure 1: Diagram of ultrasonic Testing

As shown in figure 1, the input signal of the ultrasonic probes is displayed along with the base echo of the signal. If the signal encounters a defect in the object, the defect echo is also displayed on the screen [16].

By examining the time of flight between the pulse and echo and the speed of sound traveling in the material, the location of the defect can be calculated. UT can thus be used for the detection of internal defects, and it is a popular technique for generic NDT due to its low cost and availability [17].

For railway defects detection, mobile stations equipped with UT setups are widely used. For example, in [18], Yilmaz et al. used a vehicle equipped with twelve 25MHz ultrasound probes to investigate defects on the rail, operating at an average speed of 5 km/h to investigate defects on the rail. Mounting all the sensors on the car and operating the car at a low speed were due to the fact that coupling medium such as ultrasonic gel should be applied between the probes and the rail to ensure a reliable testing result. This is because the air gap between the transducer and the rail can cause significant loss in energy transmission, leading to an extremely low signal to noise ratio (SNR). Despite efforts to perform UT in a non-contact manner, such as using air-coupled ultrasonic probes to detect rail defect [19], onerous signal processing was required to achieve a good accuracy and the low SNR for air coupling remained a fundamental problem.

2.1.3 Magnetic Flux Leakage

Magnetic flux leakage testing is another NDT method with high efficiency and accuracy. To conduct magnetic flux leakage (MFL) testing, the surface of the specimen is magnetized to the saturation point using either a yoke-type magnetizer (usually with ferromagnetic yoke) or a magnetizing coil. With the appearance of a defect, the magnetic field in the material leaks into the air. Then the MFL sensors are employed to take measurements on the volumetric changes of the charges to locate the defect spot [20]. MFL testing is very popular in oil and gas industry [21] and can also be used for rail defect detection. In a study to examine the detectability of the MFL on railway defects, Antipov et al. [22] implemented two different methods. They first used three-dimensional computer simulation to relate the defect size and depth to the sensor signal. The result showed that the transverse cracks with the heigh less than 14 mm could be detected when the edge of the defect is less than 20 mm from the rolling surface. In the second method, they recorded and analyzed 600 signals from actual rail defects and used both magnetic and ultrasonic methods to determine the detectability of various types of flows. They were able to identify over 94% of the transverse crack and 91% of the longitudinal cracks by MFL, compared to only 60% by ultrasonic testing. They concluded that MFL method is more promising and efficient than ultrasonic testing in terms of railway NDT. Jia et al. [23] enhanced the signal for railway surface defect detection by adding a ferrite magnetism gatherer (FMG). The team simulated the enhancement effect

by adding FMG with four different cross section shape (rectangle, square, diamond and circular), and the result shows that rectangular FMG achieved a better focusing ability. By further implementing FMG in the real test case, the signal was significantly amplified. The team also utilized a differential detection method to get rid of the interference of the signal.

2.1.4 Visual Inspection

Visual inspection using camera to detect defects is the easiest and most intuitive way among all NDT methods. However, it is not easy to differentiate defects from other part of the surface, especially when the surface takes an irregular shape. Therefore, image enhancement during signal processing is required.

One of the image enhancement methods, which has been successfully implemented on the railway defect detection, is spectral image differencing. In [24], Deutshi et al. introduced the spectral image differencing method by placing two light sources before and after the color-line scan camera. With a flat surface, the scan camera will read the same amount of light from both light sources. However, if on the uneven surface such as flakes or cracks, the amount of light from both lights received by the scan cam will be different. They used convolutional filter and morphological image analysis to process the images. They found that the cavity in the rail could be identified as a light spot in the image. The team installed these systems on the train and conducted the test, and found that the camera could identify the defect with the camera resolution of 0.5 mm * 0.35 mm, when the vehicle traveled at a speed lower than 60 km/h.

2.1.5 Acoustic Emission

Acoustic emission (AE) is a relatively recent detection method used for railway applications. When stress is applied to the object, transient waves are produced when cracks are initiated or propagated due to deformation, and they can be collected by the sensors applied on the objects. By inverse calculations, the locations of the newly formed crack can be found. Therefore, instead of applying external excitations during inspections for signal generation, AE makes use of the stress waves generated during normal operations. So, AE can only be used on structures undergoing dynamic changes [9].

When railway tracks are in service, the load from the train can cause the dynamic change required in AE. Zhang et al. developed a model to perform AE on the railway track [25]. Because it is hard to generate a growing defect on the test track, they simulated the process by dropping pencil leads onto

the contact point between the wheel and the track. They then tried to identify the AE caused by the crushing of the pencil leads to verify the concept. However, this was an experimental study which has not been implemented in real-world scenarios. One challenge with AE is that the measurement takes place when dynamic changes happen in the material, so continuous monitoring must be performed frequently. This can easily overload the remote signal collectors on site with numerous data, making it difficult to analyze the data effectively [26].

2.1.6 Discussion

All of the NDT methods discussed in the previous section have shown their potential and applications in defect identification and recognition for the railways. However, they have limitations that need to be considered when selecting a suitable method for specific applications. Advantages and limitations of each method are summarized in Table 2 below.

Table 2: NDT methods advantages and limitations

NDT Methods	Advantages	Limitations
Thermography	Reliable Portable Can know both surface and sub-surface defects	Active: Need to heat up the whole railway, which requires high power source Passive: Can only perform inspection during certain time range (when temperature changes the most)
Ultrasonic Testing	Portable Relatively expensive Can know the existence of the crack as well their location	Equipment is relatively expensive. Need to be close to the surface Requires couplant gel to enhance the image quality and ensure the energy wave transmission
Magnetic Flux Leakage	Portable Can identify small surface and sub surface cracks	Not suitable for detection axial cracks Need to magnetize the railway
Visual Inspection	Portable Does not require external excitation Does not require other devices	Reliability and accuracy are highly dependent on the resolution of the camera. Can only discover the surface defect
Acoustic Emission	Able to identify inner defect in the railway structure. Does not require external excitation	Can only pick up the defect when it starts to initiate (when inner structure changes)

As shown in Table 2, all NDT methods except acoustic emission can detect surface defects. However, thermography and magnetic flux leakage require external excitation, such as heating up or magnetizing the railway, which requires a large power source on the carrier to achieve defect recognition. Ultrasonic testing also requires the application of a coupling gel on the railway surface to avoid the loss of energy transmission for a more reliable result, which is challenging to achieve in real life situations. Thus, visual inspection is the most suitable method for surface defect detection.

2.2 Carrying Platform

In order to develop the system for autonomous visual inspection within the given time frame, a commercial programmable drone was chosen as the hardware platform due to its agility, flexibility, and availability. A typical configuration of such drones includes a high-resolution camera, a multi-axis gimbal system, and a software development kit (SDK). The SDK allows developers to program different software for the drone in various applications [27]. Additional sensors can also be added to the drone as extra payloads to expand its capability, if necessary.

To select a suitable drone for the project, our team conducted a thorough comparison of the specifications of the drones from different manufacturers. Since the drone should be programmable to achieve our goal, SDK support is a crucial factor for the project. After considering several options, we identified four major drone manufacturers that provide SDK support: DJI, Parrot, Skydio and Yuneec [28]. Based on our analysis, DJI SDK stands out due to its user-friendliness and strong community support. DJI also offers SDK support on multiple platforms, including Linux, Windows, and mobile platforms (Android and IOS), which further enhances the versatility of the system. Moreover, DJI updates its SDK more frequently than other competitors, ensuring that users have access to the state-of-art features, while experiencing fewer programming issues.

Three drones from DJI have SDK support, so they were considered as the hardware platform for this project. Table 3 provides detailed information on each drone.

Table 3: Specs for DJI drones with SDK support [29] [30]

	Flight Time	Sensor	Resolution	Gimbal
Mavic Air 2	34 mins	1/2" CMOS Effective Pixels: 12 MP and 48 MP	4K Ultra HD HDR: 3840×2160 24/25/30 fps 2.7K HDR: 2688×1512 24/25/30 fps	3-axis (tilt, roll, pan) ±0.01°

			FHD HDR: 1920×1080 24/25/30 fps	
Mavic 2 Zoom	31 mins	1/2.3" CMOS Effective Pixels: 12 MP	4K: 3840×2160 24/25/30p 2.7K: 2688x1512 24/25/30/48/50/60p FHD: 1920×1080 24/25/30/48/50/60/120p	3-axis (tilt, roll, pan) ±0.005° (Mavic 2 Zoom)
Mavic 2 Pro	31 mins	1” CMOS Effective Pixels: 20 million	4K: 3840×2160 24/25/30p 2.7K: 2688×1512 24/25/30/48/50/60p FHD: 1920×1080 24/25/30/48/50/60/120p	3-axis (tilt, roll, pan) ±0.01° (Mavic 2 Pro)

As shown in Table 2, the Mavic 2 Zoom (Figure 2) stands out as the most suitable drone for this project due to its stable gimbal system and superior lens quality. Therefore, it has been selected as the hardware platform for the project.

Figure 2: Mavic 2 Zoom

The flight control of the selected drone, the Mavic 2 Zoom, will be programmed to autonomously fly along the railway using an application on a mobile device. Our team is currently developing this software, which will utilize the image processing algorithm outlined in Section 5.2.

2.3 Technical Roadmap

Since visual inspection by the drone has been selected as the main inspection method, the overall roadmap can be summarized as shown in Figure 3.

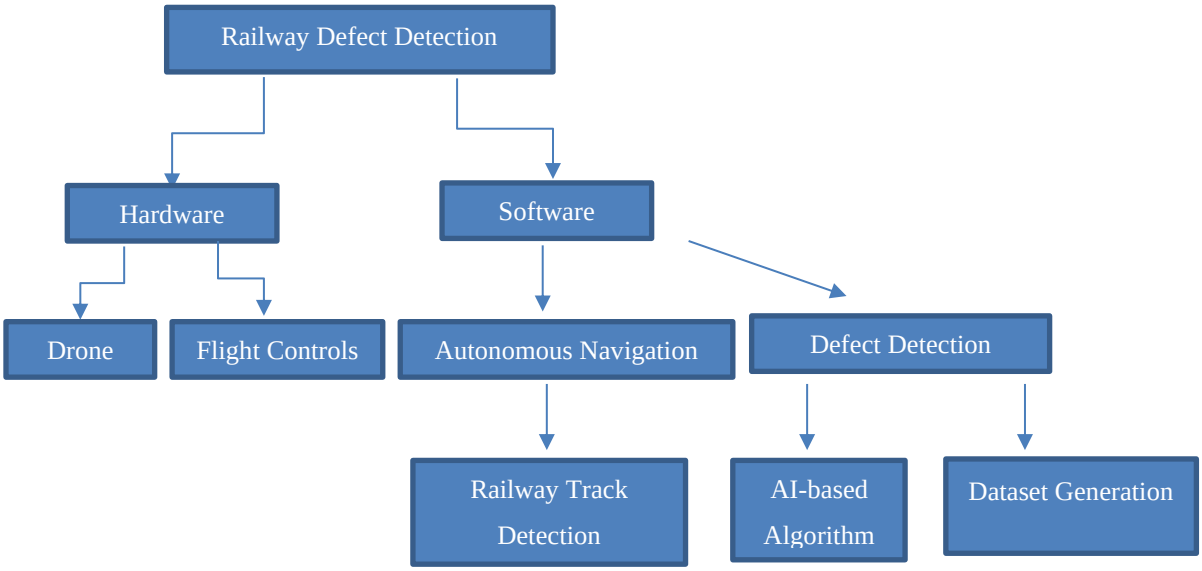

Figure 3: Technical roadmap for the entire project

The project comprises two major parts: hardware development and software design. For hardware, a drone has been selected as the hardware platform, but the issue of flight controls of the drone must be addressed to ensure autonomous flight. On the software side, a railway track detection algorithm is required to achieve autonomous flight. Once the drone can fly reliably along the railway, the defect detection should be in place to perform the inspection, and an AI-based algorithm will be developed to fulfill this task. To achieve this, a database with training and validation data for AI model should be established, and a data generation algorithm will be developed to increase the amount of data that can be used for the process.

Chapter 3
Drone Navigation

3.1 Introduction

The autonomous navigation is required to allow the drone to conduct the inspection without human intervention. It is also necessary in the development phase for swift data collection. To achieve this, two different methods have been identified in the project: pre-set flight path and real-time track tracing.

3.2 Pre-set flight path

The pre-set flight path method utilizes the drone's built-in function of doing waypoint flights, where the drone follows a flight path that is predefined in the software interface provided by DJI [31][32][33]. Waypoint function by DJI drone can retrace the route that is manually defined by the user prior to the flight. To adopt this method to railway inspection, the drone can initially fly manually by controller along the rail to record the flight path, which can be later retraced automatically by the drone. This approach is relatively straightforward and does not require additional coding since it is a built-in function of the drone. However, it relies on GPS coordinate system for path recording, thus weak GPS signals or incorrect readings due to signal jamming can lead to significant deviation from the track. If the drone is offset from the railway track, the camera can no longer record information about the railway surface. Furthermore, the pre-set flight path is fixed after recording and cannot be actively adjusted during the flight, which limits the flexibility and robustness of this method.

3.3 Real-time track tracing on flight

Since the images of a track should be recorded for the inspection, they can also be used to guide the drone for the flight by real-time image processing. The real-time image processing during the flight can also resolve the limitations of pre-set flight path method and allow a more flexible flight control to adapt to different conditions in each flight mission. In order to achieve autonomous navigation for the drone during railway inspection, two different methods have been proposed and experimented. One method involves using template matching [34] to locate the track in the image. To implement this method, a template should be made out of a section of the track. By comparing the template to various track images, the algorithm will identify and highlight the location of the railway in the image. The other method involves using edge detection algorithms to recognize the track as a line feature in the

image. Once the track is located by either method, a flight control algorithm can be implemented to guide the drone to follow the detected track.

3.4 Pattern Matching

To test the feasibility of pattern matching for railway track identification, several real-time track images are acquired by drone. A small section of the railway track is snipped and set as a template for the template matching (from OpenCV package) algorithm to search and highlight all similar structures in the other images. In this process, the template matching algorithm slides the template along the images, compares the overlapped regions using different operations and store the comparison result in a result image, as illustrated in Figure 4.

OpenCV package offers 6 different template matching operations, including squared difference (cv.TM_SQDIFF), normalized squared difference (cv.TM_SQDIFF_NORMED), cross correlation (cv.TM_CCORR), normalized cross correlation (cv.TM_CCORR_NORMED), cosine coefficient (cv.TM_CCOEFF) and normalized cosine coefficient (cv.TM_CCOEFF_NORMED) [35]. The equation for each method is introduced below. For all of these equations, R denotes the result, T denotes the template and I denotes the image [36][37].

1. Squared Difference:

$$\boldsymbol{R}(\boldsymbol{x},\boldsymbol{y}) = \sum_{\boldsymbol{x}',\boldsymbol{y}'}(\boldsymbol{T}(\boldsymbol{x}',\boldsymbol{y}') - \boldsymbol{I}(\boldsymbol{x}+\boldsymbol{x}',\boldsymbol{y}+\boldsymbol{y}'))^{\mathbf{2}} \quad \textbf{(1)}$$

Based on the equation, the best result (the pattern matches the section in the image) is achieved when the function value is zero.

2. Normalized squared difference:

$$\boldsymbol{R}(\boldsymbol{x},\boldsymbol{y}) = \frac{\sum_{\boldsymbol{x}',\boldsymbol{y}'}(\boldsymbol{T}(\boldsymbol{x}',\boldsymbol{y}')-\boldsymbol{I}(\boldsymbol{x}+\boldsymbol{x}',\boldsymbol{y}+\boldsymbol{y}'))^{\mathbf{2}}}{\sqrt{\sum_{\boldsymbol{x}',\boldsymbol{y}'}(\boldsymbol{T}(\boldsymbol{x}',\boldsymbol{y}')\cdot\sum_{\boldsymbol{x}',\boldsymbol{y}'}\boldsymbol{I}(\boldsymbol{x}+\boldsymbol{x}',\boldsymbol{y}+\boldsymbol{y}'))^{\mathbf{2}}}} \quad \textbf{(2)}$$

This is similar to the squared difference, but a normalization step is included. So the best result is also reached when the function value is zero.

3. Cross correlation

$$\boldsymbol{R}(\boldsymbol{x},\boldsymbol{y}) = \sum_{\boldsymbol{x}',\boldsymbol{y}'}(\boldsymbol{T}(\boldsymbol{x}',\boldsymbol{y}') \cdot \boldsymbol{I}(\boldsymbol{x}+\boldsymbol{x}',\boldsymbol{y}+\boldsymbol{y}')) \quad \textbf{(3)}$$

Based on the equation, the most similar section in the image to the pattern is at the maximum value.

4. Normalized cross correlation

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y')\cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'}(T(x',y')\cdot\sum_{x',y'} I(x+x',y+y'))^2}} \quad \textbf{(4)}$$

Similar to the squared difference, the best result is also reached when the result is at the maximum function value.

5. Cosine Coefficient

$$R(x,y) = \sum_{x',y'}(T'(x',y') \cdot I'(x+x',y+y')) \quad \textbf{(5)}$$

where T' and I' can be further expressed as:

$$T'(x',y') = T(x',y') - \frac{1}{w\cdot h} \cdot \sum_{x'',y''} T(x,y) \quad \textbf{(6)}$$

$$I'(x+x',y+y') = I(x+x',y+y') - \frac{1}{w\cdot h} \cdot \sum_{x'',y''} I(x+x'',y+y'') \quad \textbf{(7)}$$

For the cosine coefficient equation, the best match between the template and the image is indicated by a value of 1 in the resultant image, while the worst possible value is -1. The value 0 in the resultant image means that the template and the section of the image being compared are not related.

6. Normalized cosine coefficient

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y')\cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'}(T'(x',y')\cdot\sum_{x',y'} I'(x+x',y+y'))^2}} \quad \textbf{(8)}$$

The characteristics of the resultant image from the normalized cosine coefficient are similar to those of the cosine coefficient. A positive value indicates a good match, and a negative value means a poor matching result. However, unlike the cosine coefficient which falls within the range [-1,1], the result from the normalized cosine coefficient can range from negative infinity to positive infinity.

For all these equations introduced above, the symbol prime ($'$) and double prime (") denote different coordinate system. Let's consider the following example with the first template option (squared difference) in Figure 4:

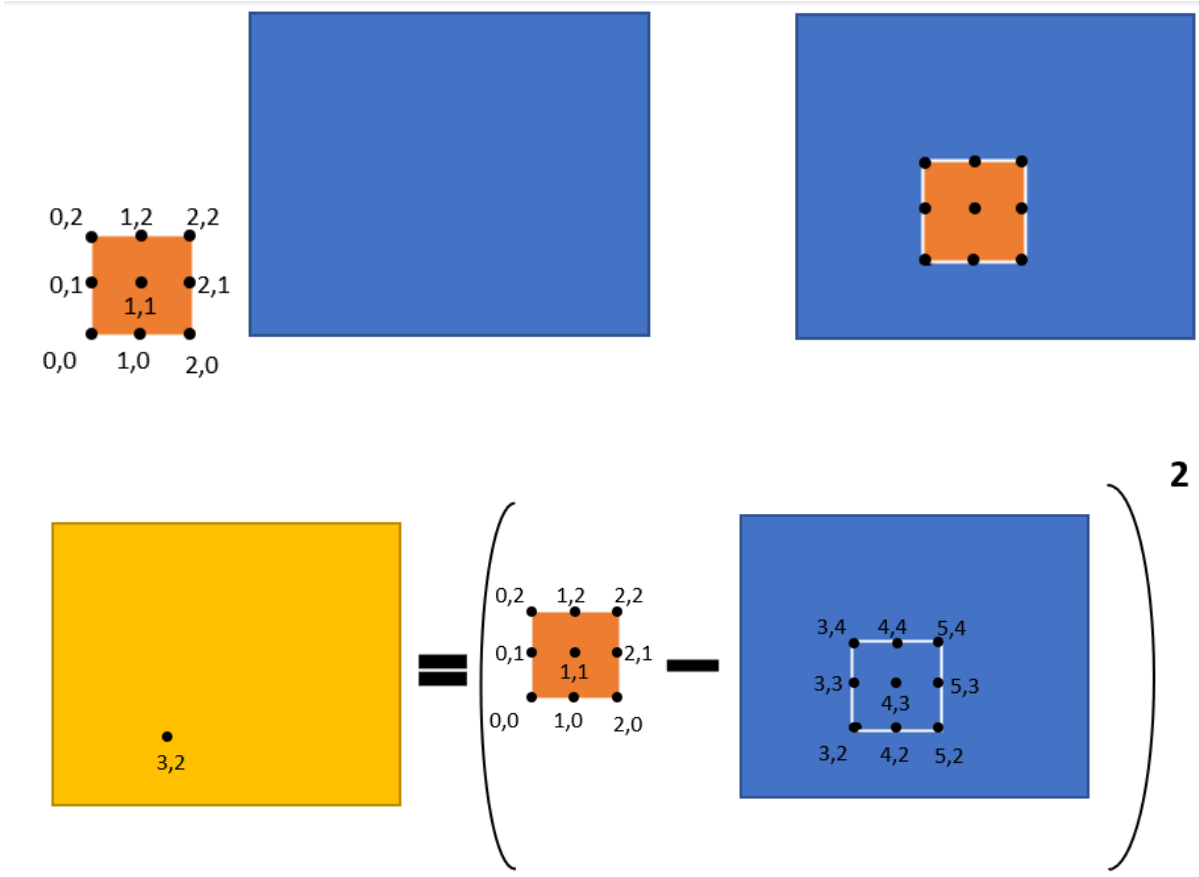

Figure 4: Example of template matching

Suppose the template (orange square) is a 2*2 image with a local coordinate system of (x', y'). As the template is sliding through the image (blue square) and its left corner overlaps with the point (3,2) (the coordinate system of the image), the result (x, y) (the value of the result at point (3,2)) is calculate by the summation of the squared difference between the template $T(x', y')$, and the image $I(x + x', y + y')$, which begins with the point (3,2).

Among all the methods, TM_SQDIFF and TM_SQDIFF_NORMED are widely used due to its fast computational speed when the template is an exact match of the section in the image. However, TM_CCOEFF_NORMED is recommended for cases where there is more contrast or exposure [38]. As a result, TM_CCOEFF_NORMED is selected as a template matching operation for the project.

Different threshold values are applied to the test, and the results acquired by implementing template matching are shown in Figure 5. The algorithm demonstrated the ability to find the patterns with

different confidence levels depending on the chosen threshold value. However, further investigation revealed that the track recognition results were too sensitive to the chosen template. The lighting conditions at the time the pictures were taken played an important role in setting up the comparison, resulting in poor performance when template taken from images acquired in different lighting conditions were used for detection. For example, the template image taken on a sunny day with a higher intensity value could not be used for railway images taken on an overcast day with a lower intensity value.

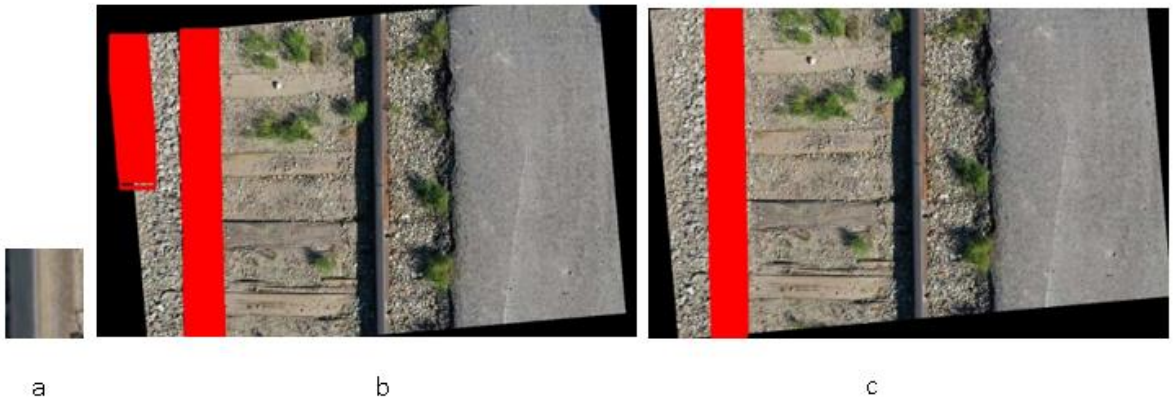

Figure 5: An example of same railway with different threshold for template matching. a) Template; b) Threshold with 0.5; c) Threshold with 0.7

In some cases, even with templates taken from the same lighting conditions, the railway tracks appeared too similar in color to their surrounding features, which could easily confuse the algorithm. Furthermore, since the drone was used in this project, it was also important for the drone to maintain a certain altitude when using this method. Because of the scaling effect caused by the distance, the size of the track in the template can be very different from that of the actual image if the altitude settings were different for the template and the image being processed. Therefore, while this method had the potential to be implemented, but it was limited by several factors that were difficult to control in real-world situations.

3.5 Edge Detection Algorithm

In order to implement the edge detection algorithm, the OpenCV [39] package was utilized. The complete code for both image and video processing can be found in Appendix A and B. To reduce the processing time, the images are first resized to a smaller number of pixels. The original image size

acquired from DJI drone is 5472*3648, but in the testing, it is found that robust results when could be obtained with a target size of 640*480. Next, the images are converted to greyscale to simplify the operations required for the processing [40]. The original image and the greyscale image after this step are shown in Figure 6.

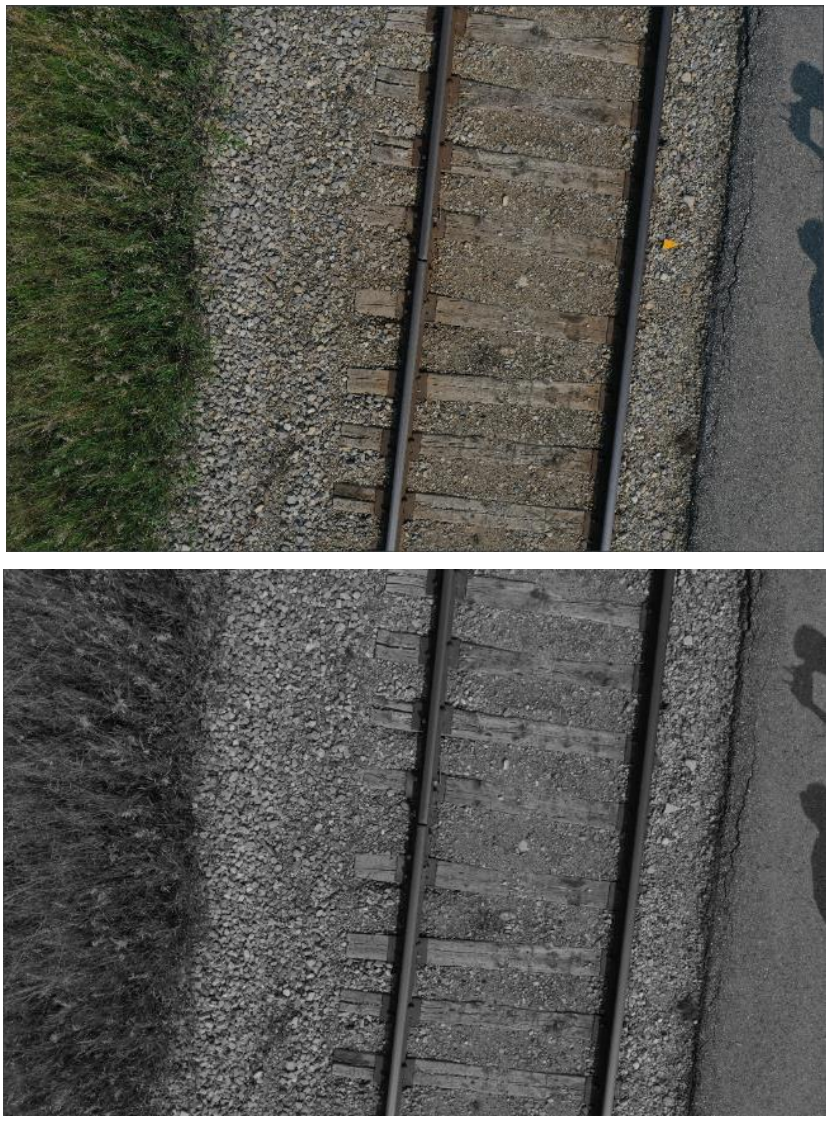

Figure 6: Railway track image (top: original image; bottom: greyscale image)

After the greyscale image is obtained, the gaussian blur is applied to smooth the image. The gaussian filter is a low-pass filter that helps to blur the image and remove the unnecessary features to rule out noises on the images [41]. This spatial filter uses a kernel to convolve with input image, and the

resultant images consists of the new pixel values with the weighted average operation of the original pixel value and its neighboring pixels. The one-dimensional filter can be expressed by equation 9 below [42]:

$$\boldsymbol{g(x) = e^{-\frac{x}{2\sigma^2}}} \quad \textbf{(9)}$$

where x is the x coordinate value, and σ is the standard deviation. For image processing, the gaussian filter is usually in two-dimensional, and the equation is shown as below:

$$\boldsymbol{g(x) = e^{-\frac{x^2+y^2}{2\sigma^2}}} \quad \textbf{(10)}$$

where x and y are the distance between the gradient and the origin in both x and y coordinates. The smoothed image is shown in Figure 7 below:

Figure 7: Image after gaussian filter (top: greyscale image bottom: smoothed image)

Once the image is smoothed, the actual edge detection operation is implemented to enhance contrast between the railway tracks and its surroundings. In this project, two different types of edge detection methods are tested: Canny Edge detection and Sobel Detection. For Canny edge detection, the algorithm first removes the noise in the image with a 5*5 Gaussian filter, and then it finds the intensity gradient and its direction using horizontal and vertical Sobel kernels. The equation for finding the gradient and the angle is shown in equation 11 and equation 12 [43].

$$\boldsymbol{G(x) = \sqrt{\left({G_x}^2 + {G_y}^2\right)}} \quad \textbf{(11)}$$

$$\boldsymbol{\theta = \arctan\left(\frac{G_y}{G_x}\right)} \quad \textbf{(12)}$$

where Gx and Gy are the first derivative of smoothed images by Sobel kernel in both x and y directions, respectively. After obtaining the gradient, the non-maximum suppression (NMS) are performed to enhance the edge detection. This involved scanning the entire image for local maximum point in both horizontal and vertical directions. Pixels that quailed as local maxima were retained, while others are suppressed by setting their intensity value to zero. In this way, the thick edges could be converted to sharp and thin edge, improving edge identification accuracy. The final stage to complete the process is thresholding, where the upper and lower threshold values are set, and only the edge section that falls within the range are retained for further processing.

For the Sobel edge detection method, on the other hand, only two convolutional kernels are applied in both x and y directions. Such kernels are shown in Figure 8 [44]:

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

X-direction

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

Y-direction

Figure 8: The x-direction Sobel kernel and y-direction Sobel kernel

Subsequently, these two kernels are convoluted with the original images to obtain the gradient magnitude Gx and Gy in both x and y directions. The total gradient magnitude is computed using the same equation 11 as Canny edge detection. However, in some cases, the approximate magnitude is calculated using equation 13 [45]:

$$\boldsymbol{|G| = |G_x| + |G_y|} \quad \textbf{(13)}$$

Upon applying both line detection methods, it was discovered that the Sobel detection method proved to be the better option. The principle of the Sobel filter involved applying directional masks to enhance the edge feature in the images [41]. To implement the Sobel filter, we included both horizontal and vertical Sobel filters after the gaussian blur. The result from the Sobel line detection is displayed in Figure 9 below:

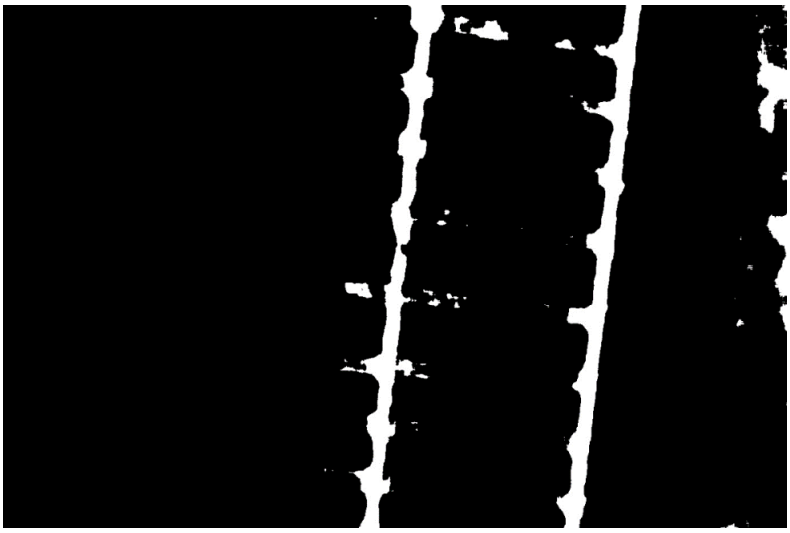

Figure 9: Image obtained after the line detection

The next step is to find the straight lines that were enhanced by the Sobel filters in the images. This is achieved by applying Hough Transform to the image, as detailed in reference [46]. To identify a desired line, the line representation is initially expressed using equation 14:

$$\boldsymbol{y = ax + b} \tag{14}$$

This line expression is then altered to the following form:

$$b = ax + y$$

where *a* is the slope and *b* is the intercept of the line. The purpose of line detection is to identify the appropriate pair of *a* and *b* values, which is visualized in Figure 10.

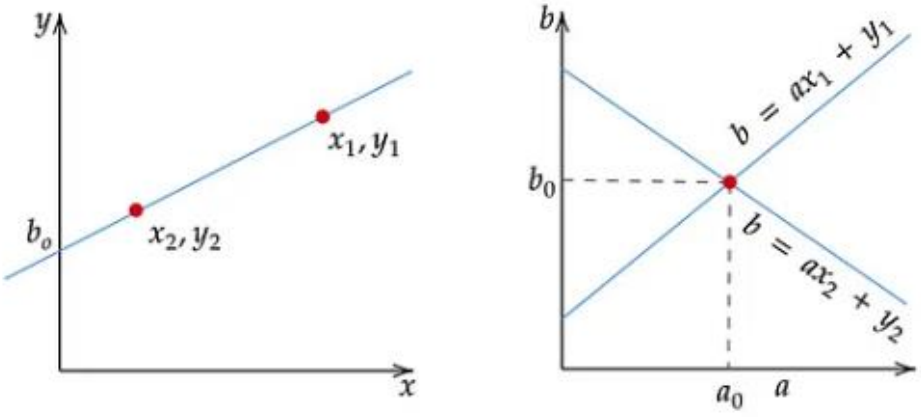

Figure 10: Line detection algorithm

However, this approach has the drawback that vertical line cannot be detected as the slope value becomes undefined. To overcome this problem, the Houghline transform is introduced. The Houghline transform operates on the principle that when two points lie on the same line, their corresponding cosine curves intersect at a specific θ and ρ values, as illustrated in Figure 11 [47][48]. Here, ρ is the length of the normal line and θ is the angle between the normal line and the x-axis. By selecting a threshold for certain θ and ρ pairs, the Houghline transform can identify lines with a desired orientation, enabling the detection of both horizontal and vertical lines.

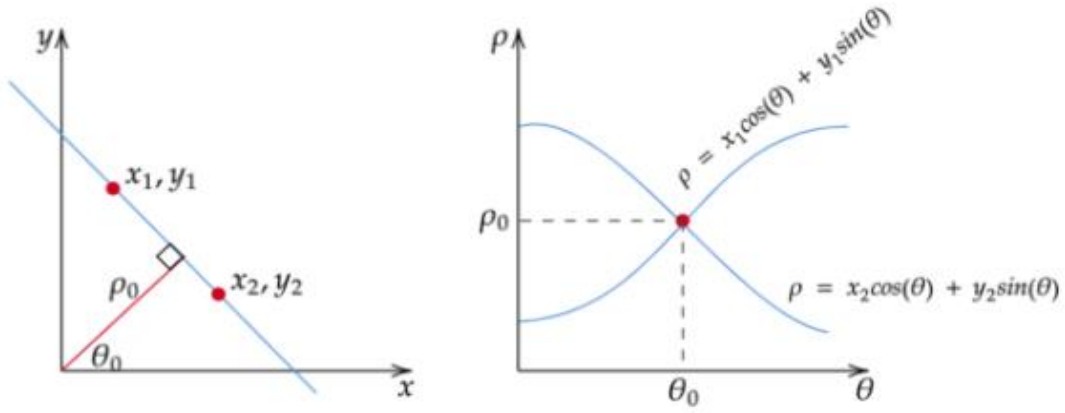

Figure 11: Representation of the θ and ρ pair [45]

Such algorithms output the x and y coordinates of the starting point and ending points of all the straight lines detected in the images. The mid-point of the two tracks can be calculated by taking the average of the top and bottom 50 percentile x and y values of the detected lines which helps finding the mid-point of the two tracks.

The algorithm developed so far can detect the tracks consistently. The sample result is demonstrated in Figure 12, where the detected tracks are highlighted in green, indicating the accuracy of the algorithm in identifying the location of the railway tracks within the images.

Figure 12: Edge Detection Algorithm to Detect Railway Track

3.6 Autopilot

In practice, the centerline of the drone's camera is aligned with the centerline of the drone itself. To ensure accurate tracking of the detected railway tracks, a simple control algorithm is developed that adjusts the drone's position during flight. The algorithm aligns the centerline of the drone's camera with the average position between the two railway tracks, as detected using the method described in the previous section. Based on the alignment, the control algorithm calculates the necessary drone maneuvers using the control model we developed. These commands are then transmitted to the drone, enabling automatic course correction for the drone during the flight.

Figure 13 provides an example of the result from control algorithm when applied to a single frame of the recorded video. The movement requirement is displayed in pixels within the image, which is later translated into the duration of the drone's sideways flight. This information enables precise control of the drone's movements, ensuring accurate tracking of the detected railway tracks.

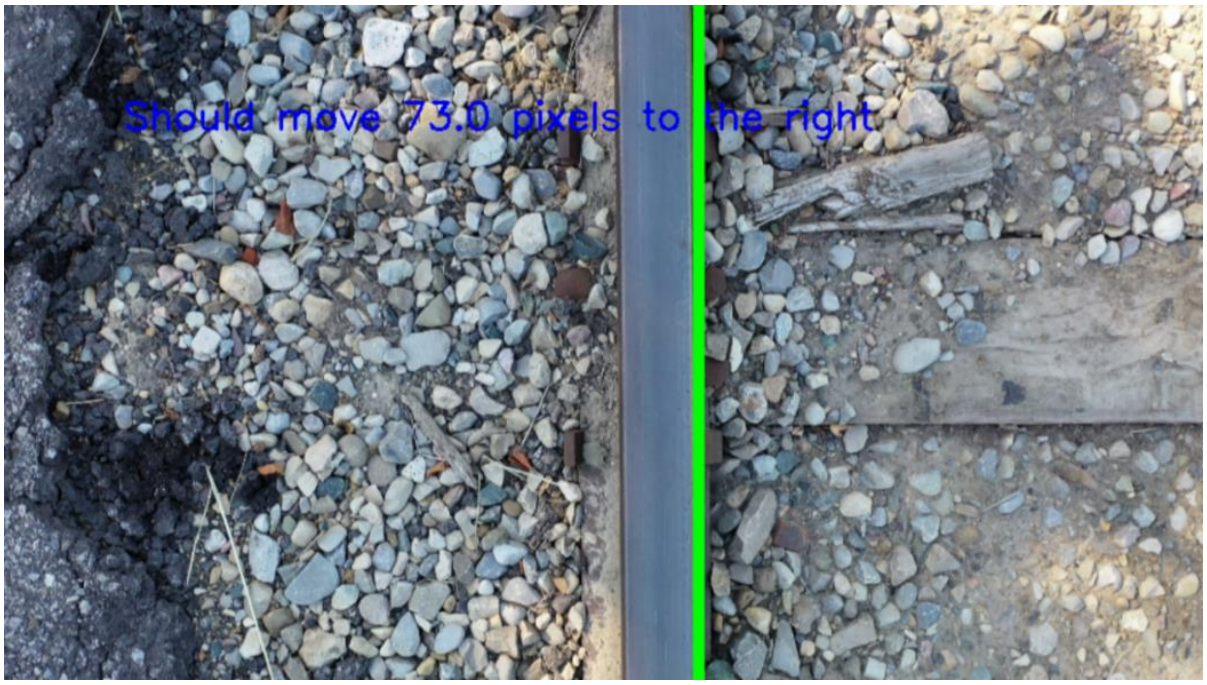

Figure 13: Image snipped by the video processed with control algorithm

Chapter 4
Defect dataset generation

The primary focus of this chapter is to develop a methodology for generating a sufficient number of defective railway images to train a detect identification training model. By systematically introducing a range of defects into high-quality railway images, we aim to create a comprehensive and diverse dataset that can be used to train machine learning algorithms to accurately detect and classify defects in railway infrastructure. This approach has the potential to significantly improve the efficiency and accuracy of railway inspection and maintenance, ultimately contributing to improved safety and reliability in rail transportation.

4.1 Introduction

Since surface defects can be visually inspected, computer vision-based (AI) methods have recently become a popular tool for identifying and classifying surface defects. For example, algorithms such as You Only Look Once (YOLO) and Region-Based Convolutional Neural Network (RCNN) have been implemented for real-time defect detection and segmentation on low-shot walls, steel plates, and even transmission lines [49][50][51]. However, since these algorithms rely on supervised learning, the problem of generating a sufficiently large pool of image data, especially for defective samples, has emerged. While data can be collected from railways in service, this method is time-consuming and may not provide a sufficient quantity of data. Alternative methods, such as engraving defects onto real railway samples or manually creating artificial defects in images of railways, are both inefficient and costly, because they require extensive human intervention and supervision. However, artificial imaging methods seem feasible if the process can be automated. To this end, there have been attempts to create defective images using network-based image processing techniques, such as generative adversarial network (GAN) by Goodfellow et al. [52]. This chapter aims to develop an efficient and automated methodology for generating a diverse dataset of defective railway images, improving the accuracy and efficiency of defect identification and classification using machine learning algorithms.

4.2 Generative adversarial network (GAN)

In machine learning, a typical GAN is a type of neural network model consisting of two networks: a generator network and a discriminator network [53]. The main purpose of the generator network is to

generate synthetic images based on normal images, whereas the discriminator network tries to differentiate the defective and normal images. The two networks are trained simultaneously in a competitive process, where the generator network tries to generate images that can fool the discriminator network, and the discriminator network tries to accurately classify the images as either normal or synthetic. The overall goal of a GAN model is to learn the underlying distribution of normal images and generate new images that are indistinguishable from the normal images. The flowchart of a GAN model is shown in Figure 14.

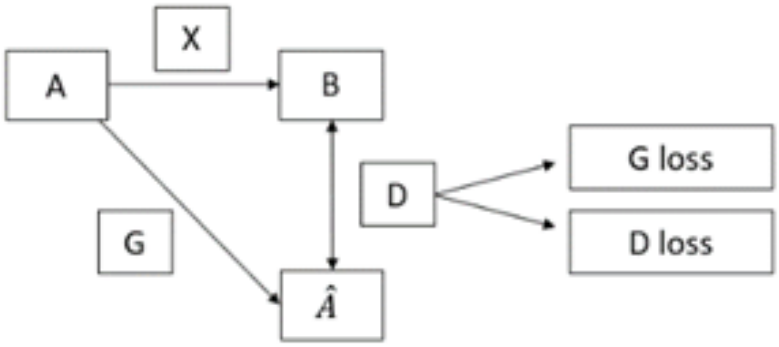

Figure 14:Flow chart of the GAN model

The mapping (X) measures the difference between the non-defective dataset (A) and the defective dataset (B). The non-defective dataset (A) is sent to the generator (G) and a defective image set ($\hat{A}$) is generated according to mapping (X). Then, the generated defective images are sent to the discriminator (D) with the provided defective dataset (B). The comparison result is produced by the discriminator, and according to the return value, differences can be found between the generated dataset ($\hat{A}$) and the defective dataset (B). Both the generator and discriminator need to be updated during the training. The generator loss measures a reward for the generator if the generated images "fool" the discriminator, while the discriminator penalizes itself if it misclassifies the image. When the loss function reaches a balance (discriminator and generator can no longer be improved), the discriminator cannot tell the difference between the real and artificial defects [54][55]. This approach allows the GAN model to generate high-quality synthetic defective images, which can be used to train a classifier to recognize and classify defects in future inspections.

4.3 GAN extensions

The classic GAN model has been evolved into different forms and architectures: DCGAN [56-59], Pix2Pix [60-62], CycleGAN [63-65] and StyleGAN [66-71], and several attempts have been made to generate surface defects using these models.

4.3.1 DCGAN

The DCGAN, developed by Radford et al. [56], is a direct extension of the vanilla GAN model. DCGAN uses convolutional layers with LeakyReLu activation function in the discriminator network; it also uses convolutional-transpose layers with ReLU activation function in the generator network. The generator takes a vector of input images, and returns an image defined by its RGB color matrix, while the discriminator uses the information from the RGB images and outputs a scalar probability [56][57]. Liu et al. [58] applied DCGAN to generate defective images of micro-precision glass encapsulated electrical connectors. In their study, both the generator and discriminator had 5 convolutional layers. With their DCGAN model, Liu et al. successfully generated 1,500 defective images from 125 real defective and 75 non-defective samples. Another example of using DCGAN for defective image generation can be found in [59] where the image enhancement algorithm was combined with the DCGAN to generate defects on magnetic rings images. The image enhancement was conducted with a frequency-based bandpass filter. It was reported that their method can produce images with better details compared to the original GAN model.

4.3.2 Pix2Pix

The Pix2Pix model was first introduced by Isola et al. [60]. The input to the model is a tuple consisting of a non-defective image and an exact corresponding target image. Note that both images are identical other than the appearance of the object of interest or the defects on the objects. The generator is updated according to the L1 losses measured by the discriminator until convergence [61]. Mertes et al. used a Pix2Pix model to generate synthetic defects on carbon fiber surfaces in images [62]. It was shown that Pix2Pix model can generate realistic-looking synthetic defects and can improve the pixel-based defect classification, while the performance of the model is similar to the conventional data augmentation method using normal images.

4.3.3 CycleGAN

Zhu et al. first proposed the CycleGAN model in [63], which has two generators and discriminators, respectively. Whereas other GAN models have only forward mapping, the CycleGAN model introduces one more generator to learn the inverse mapping. By including this additional mapping, the CycleGAN model can introduce cycle consistency losses to improve the model accuracy. Shuanlong et al. [64] used the core concept of cycle consistency loss to build a surface defect generation adversarial network (SDGAN) to generate images with a commutator surface defect. The team combined the D2 loss with the original cycle consistency loss, where D2 loss measured the performance between the pair, the discriminator and the generator. They compared SDGAN with other GAN models in terms of accuracy and quality and found that the SDGAN model could achieve a lower error rate (1.77% in defect generation and 0.74% for defect classification) than the original CycleGAN model. Du-Ming Tasi et al. [65] proposed a two-stage CycleGAN model to generate and identify surface defects on a texture surface. They used images with and without defects to produce a synthetic image in the first stage. Then, they place one defect-free image and the corresponding synthetic image into the second CycleGAN. During the training process, a binary image containing the shape of the defect was returned as the result of the accumulated differences between the defect-free and synthetic defect images. After that, a U-Net architecture was used to perform defect segmentation. Their model could successfully annotate and segment different patterns of defects with an accuracy of over 90%.

4.3.4 StyleGAN

StyleGAN was first developed by Karras et al. [66]. During the process of this model, images go through several modifications [67]. First, the image sizes of both generator and discriminator are doubled in width and height during training. Second, the generator structure uses the adaptive instance normalization, which converts the output into Gaussian distribution with a style vector as bias. Third, Gaussian noise is added to each layer to allow slight variation in the style that the model can generate [66-68]. Lastly, StyleGAN adopts mixing regularization where two random latent codes are used to generate a given percentage of images, respectively. This allows the model to localize the style to specific parts of the images. Situ et al. [69] combined the StyleGAN model with adaptive discriminator augmentation (ADA) to generate surface defects on water sewers. They were able to generate high-resolution synthetic images with the model. They further combined the model with the freeze discriminator (Freeze-D) and found that the performance was better than both StyleGAN and

StyleGAN-ADA. Saiz et al. [70] combined the StyleGAN with a differentiable augmentation (DiffAugment) method to generate images of defects on steel surfaces. The idea of the DiffAugment method is that the data augmentation for GAN models should be differentiable [71]. The dataset can improve segmentation compared to the conventional augmentation methods.

4.3.5 Model Recommendation

The aforementioned models proposed by various literature have demonstrated high accuracy and success rate in generating defect patterns in some specific applications. However, these models have shown their ability only when there is no background. In the training of defect detection algorithms, target object, such as railways in this study, can contain similar features to the background. Therefore, using CycleGAN without any enhancement or pre-processing of the images can generate defects in the background, as there is no control over the locations where the defects are created. Hence, RailGAN is proposed and developed in this research to accurately produce synthetic defects only on the specific object, i.e., railway.

4.4 RailGAN Model

The RailGAN process is composed of two parts: image segmentation and defect generation. The flowchart of the overall process is shown in Figure 15.

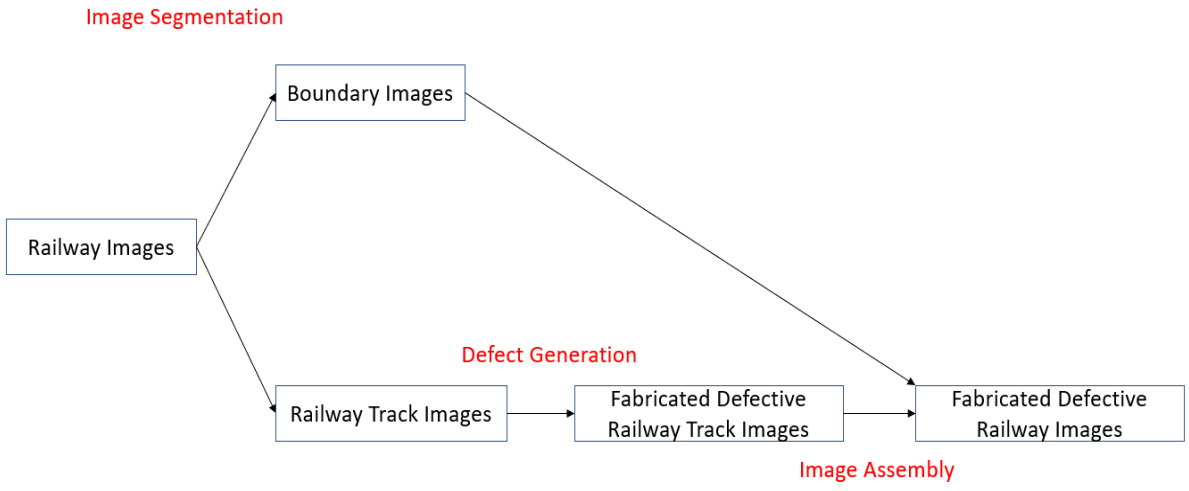

Figure 15: Process of RailGAN

Firstly, the railway images go through the image segmentation process to differentiate the background and the railway track surface sections. Then the defect generation algorithm is applied to the track surface section. Finally, the original background section is added back to fabricate the defective railway images.

Image segmentation techniques are employed to classify each pixel in the image into two different classes: background and the railway track surface. There are several techniques to achieve this, and in this project, two approaches are considered: image filtering and convolutional neural network (CNN).

4.5 Methodology

4.5.1 Image Filtering

For the image segmentation, the railway images are first imported as greyscale images to reduce the computation time. Then, two layers of 2D convolutional filters are applied to the images.

The first filter applies to the image is the Gaussian filter. The Gaussian filter blurs the image by performing a downsampling, removing details in the background and emphasizing key features in the image. By applying Gaussian filter to the railway images, the sketchy edges of rocks, sand or bushes in the background are softened, leaving only the track area as the focus for the next steps. After the Gaussian filtering, a Sobel filter is applied to the image to emphasize the changes between the adjacent pixels. This operation effectively enhances edges, with a threshold value selected based on the railway edge pixel intensity values. However, the light reflection from the uneven surface in the background can also be enhanced, so an additional step of Hough transform is applied to find line features. Hough transform detects the points that can form lines in the image based on the criteria of minimum points and maximum gap width between the points. This function returns coordinates that connect left and right boundaries of both tracks. These coordinates can also provide information on the orientation of the railway presented in the image.

4.5.2 CNN Approach

U-Net is chosen as the CNN solution for image segmentation in this project, because fewer sample images are required to train an accurate model. U-Net was originally developed by Ronneberger et al. [72] for biomedical segmentation. It was named for its U-shape structure, as shown in Figure 16 [72]:

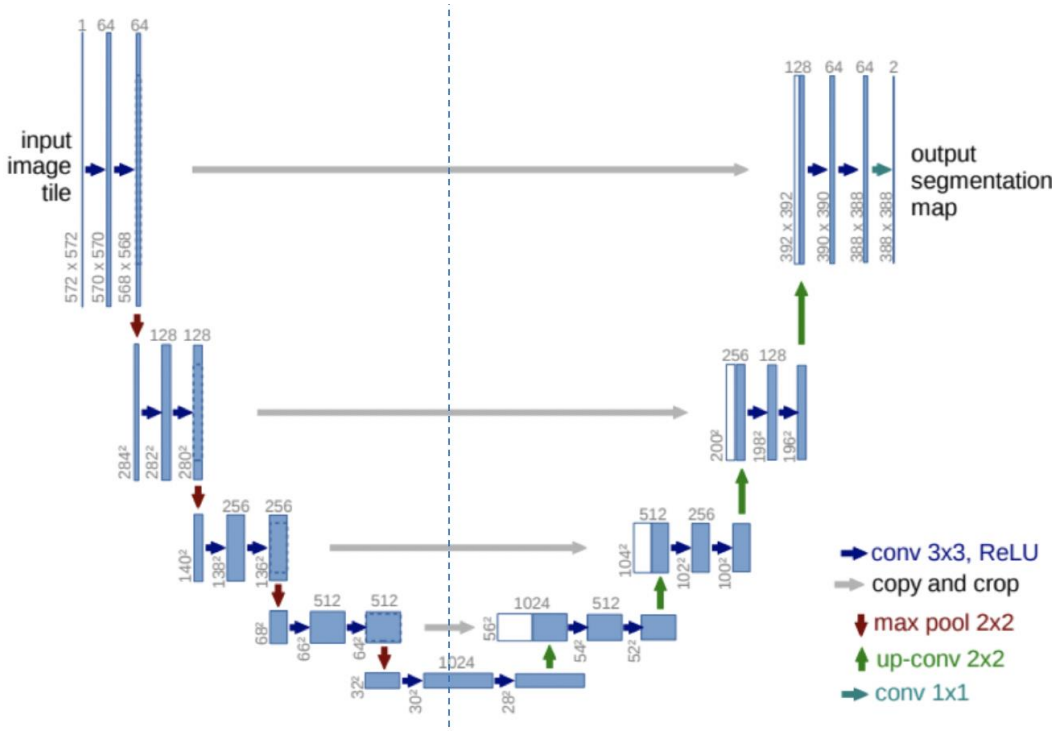

Figure 16: U-Net Structure [72]

U-Net contains two sections: contracting path and expansive path. During the contracting path, which is shown in the left half of Figure 16, images go through convolution operations followed by downsampling. This process is similar to other convolutional neural networks, which convert each input image to its own feature representation. The expansive path is the opposite of the contracting path. For each stage, the feature map is upsampled by up-pooling convolution, followed by unpadded convolution process. With the symmetric shape of the U-Net, the model allows for skipping connections between an encoder layer and a decoder layer, which enables fine-grained details to be constructed in the output segmentation map. For the training process, input images with their corresponding segmentation map are sent to the network, and the neural network learns how to segment the input railway images.

4.6 Defect Generation

The defect generation algorithm in RailGAN is based on CycleGAN. As introduced in the previous section, CycleGAN has shown the potential to outperform the vanilla GAN model in creating

artificial defective images, thanks to its delicate architecture. The detailed process is shown in Figure 17.

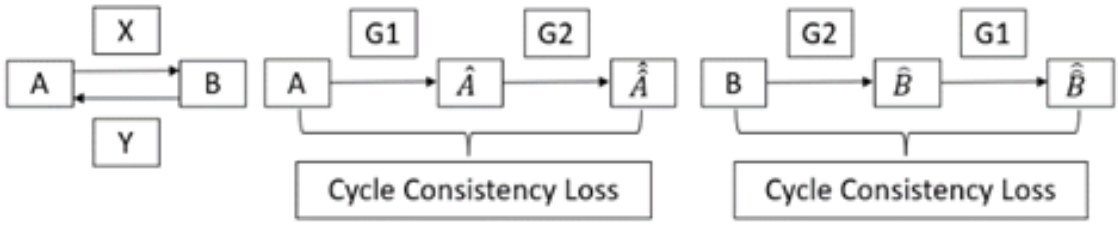

Figure 17: CycleGAN model architecture

Typical GAN model has only the forward mapping X from A (non-defect) to B (defect), while the CycleGAN model introduces one more generator, Y, to learn the inverse mapping from B to A. The two discriminators measure the similarity between the original non-defect dataset and the generated defect images, as well as the original defect dataset with the generated defect images. CycleGAN can further improve its model accuracy by using both forward and inverse mappings and exploiting both cycle consistency losses. After dataset $\hat{A}$ is produced by the first generator, $\hat{\hat{A}}$ is created through the inverse mapping F, which allows one cycle consistency loss to be measured for the similarity between the original sample A and generated $\hat{\hat{A}}$. Similarly, there is one more loss being measured between $\hat{\hat{B}}$ and B. These losses ensure the quality of the generators, providing further benefits to the overall performance of the model. For a better interpretation, when we translate an article from English to Chinese, and then translate it back to English, the difference between the original English article and the translated English article is considered as the cycle consistency loss.

When applying Cyclegan to our railway track project, the generators, discriminators, and cycle consistency loss perform the following tasks:

Generator 1 (G1) takes the photo of a non-defective railway track from the dataset 1 as input and generates a defective railway track image. Then, the discriminator 1 (D1) compares the original defective railway track image from the dataset 2 with the generated defective railway track image from G1.

Generator 2 (G2) inputs an image from the defective railway track dataset and generates a non-defective railway track image with the mapping. Then, the discriminator 2 (D2) compares the original

non-defective railway track image from the dataset 1 with the generated non-defective railway track images by G2.

The process for cycle consistency loss 1 (forward) is as follows: the model first uses G1 to generate a defective image from a non-defective railway track image in dataset 1, and then it further implements G2 to generate the non-defective railway track image from the generated image. The cycle consistency loss will measure the similarity between the original non-defective railway track image with the non-defective railway track image generated by G2. Cycle consistency loss 2 (backwards) has the same process, but it compares the original defective railway track image with the defective railway track images generated by G1.

4.7 Implementation

4.7.1 Image Filter

When following the process from section adding image filters to achieve the image segmentation, the results of each step can be demonstrated as shown in Figure 18.

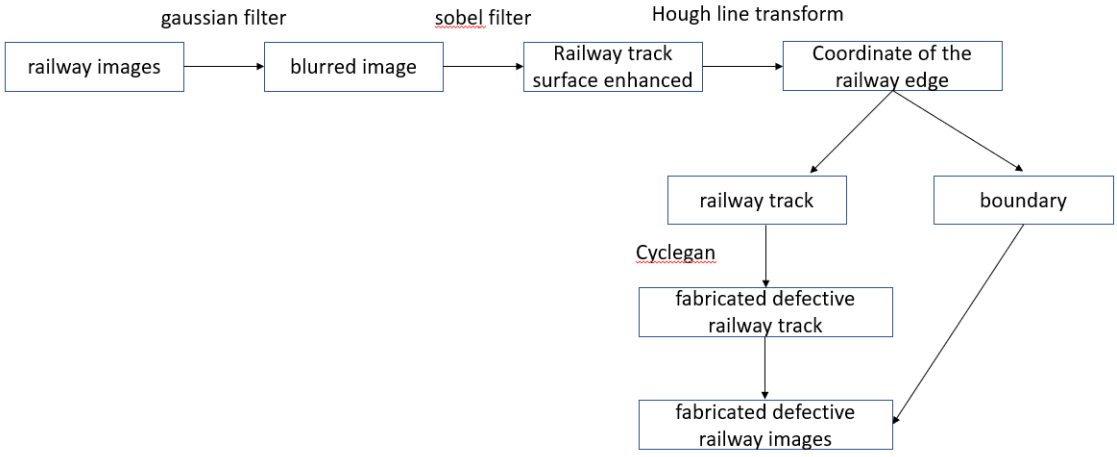

Figure 18: RailGan process with image filter

The output after each convolutional filter is shown in Figure 19.

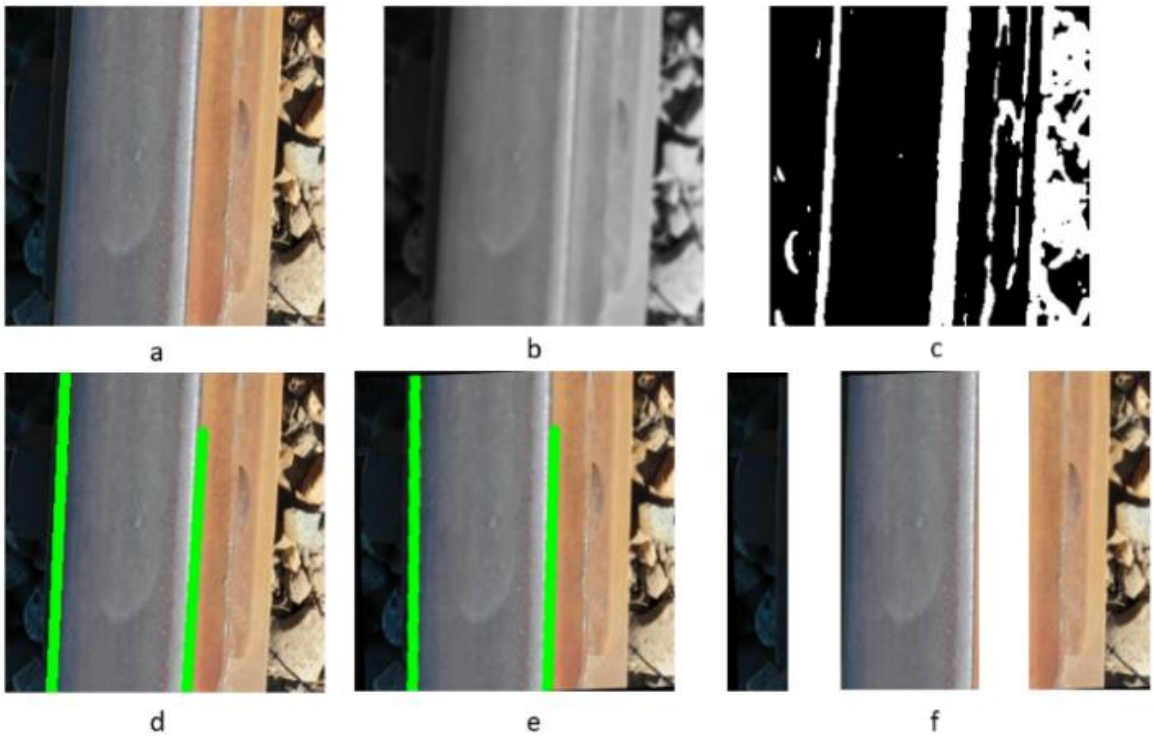

Figure 19: (a) Live image, (b) Greyscale image from Gaussian filter, (c) Binary image from Sobel filter, (d) Lines Detected by Hough transformation, (e) Straightened image after rotation, and (f) Image segmentation after track and background identification

The railway images are first resized to an image size of 256*256 to reduce the required computational time and power. The Gaussian filter uses the kernel of dimension of 5 and had a standard deviation of 5 in the x-direction and 0 in the y-direction. The Sobel filter has a kernel size of 3, and it contains values representing the first order x-derivative, which enhances all vertical edges. A threshold of pixel value is applied, which created binary image as shown in Figure 19(c).

The next step in the image segmentation process it to apply Hough transformation, which detects lines consisting of a minimum of 200 points with a maximum gap of 1. These parameters are adopted empirically to highlight the edges of the railway. By using the coordinates of the end point, the algorithm is able to determine the angle between the found line and the vertical axis. The angle is then used to rotate the image so that the railway was aligned in the vertical direction. Once the rotation is complete, the final step is performed to separate the railway from the background.

4.7.2 U-Net

With the U-Net algorithm, the overall structure is implemented as show in Figure 20:

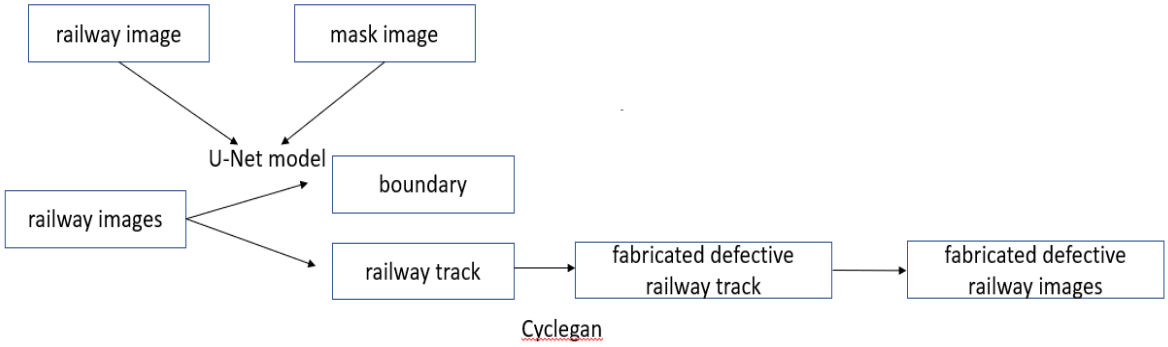

Figure 20: RailGAN with U-Net

The contracting path consists of two consecutive layers of 3*3 unpadded convolutions, which reduces the size of the image by 2 in both width and length. Max pooling with a 2*2 kernel is then applied to discretize the spatial size of the layer by retaining only the maximum value within each 2*2 area, effectively reducing the computational load [73]. An example of a 2*2 max pooling process is shown in Figure 21.

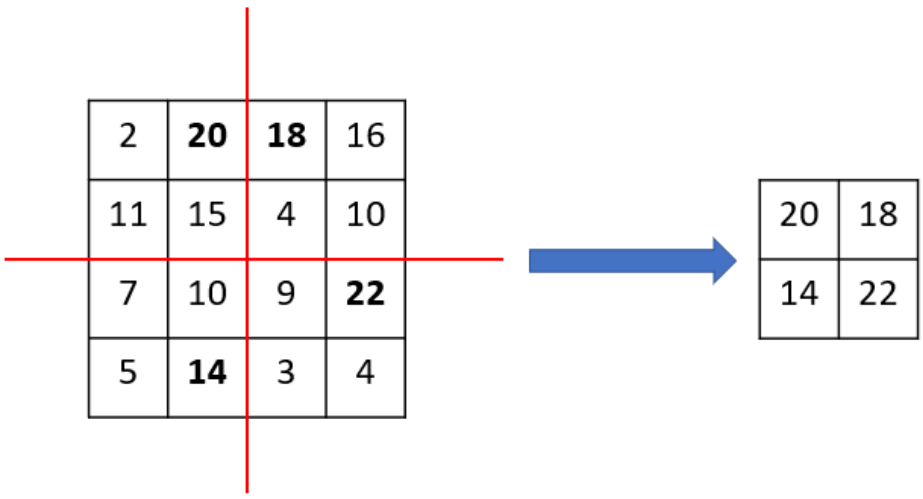

Figure 21: Example of 2*2 maxpooling

The two 3*3 unpadded convolution and one 2*2 max pooling calculation are repeated three times during the contracting path to downsample the image. The ‘ReLu’ activation function i used in each step. The ReLu activation function sets the function value to 0 when x is smaller than 0 and equals to x when x is greater than or equal to 0 [74]. The ReLu activation function can be expressed as equation 15 and can be visually represented in Figure 22. ReLu activation function provides computation efficiency and introduces the non-linearity property of U-Net, which helps to generalize training data.

$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \quad \textbf{(15)}$$

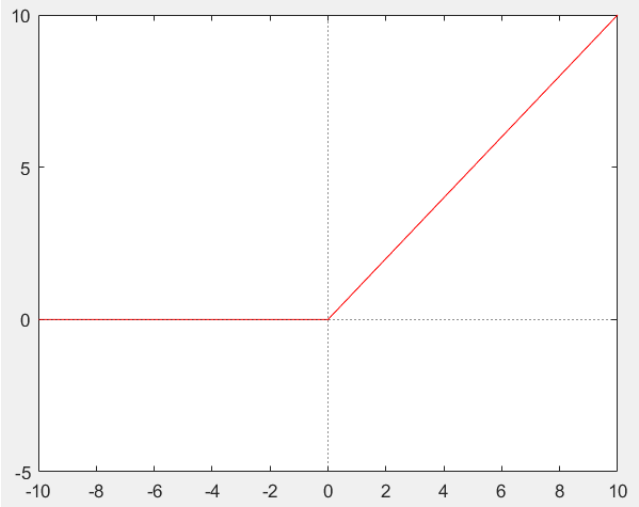

Figure 22: ReLU activation function

During the expansive path, a 2D transposed convolution is used to upsample the feature map, followed by concatenation with the corresponding feature map in the contracting path to retain a more accurate prediction [75]. This process is repeated three times, and each step was finished with two 3*3 unpadded convolution with ‘ReLu’ as an activation function. At the end of the expansion path, a final 1*1 convolution layer is used to map the feature vectors to classes. The loss of the network is originally measured by a multi-class cross-entropy loss function, but for our case, where we only want to segment the images into two classes (railway track and the background), the loss function is changed to binary cross entropy (BCE) for the training. The binary cross entropy function is derived from the Bernoulli distribution, as shown in equation 16.

$$f(k; p) = p^k + (1 - p)^{(1-k)} \quad \textbf{(16)}$$

where k is the possible outcomes, p is the probability of the case 1, and 1-p is the probability of case 2. Binary cross entropy is similar to the Bernoulli distribution with only two cases. However, the formal way of Binary cross entropy loss function is taking the log of both sides of the Bernoulli distribution, and it is expressed as:

$$\log\left(f(k:p) = klog(p) + (1-k)\log\left(1-p\right)\right. \quad \textbf{(17)}$$

The network is trained with 'Adam' optimizer with a learning rate of 0.0001. Adam optimizer is chosen in this case for its fast computation time and less parameter to tune than other optimizer. [76] Adam implements the idea of the momentum stochastic gradient descent and root mean square propagation (RMSP), which are recognized as the two moving averages in the algorithm. Instead of regular stochastic gradient descent (SGD) which maintains the same learning rate during the weight updates and gradient descent, the concept "momentum" incorporates the information from the previous steps to accelerate the gradient descent process, and it is considered as the "exponentially weighted average" [77]. This step can be expressed equation 18 below to calculate the change of parameter (weights or position).

$$\boldsymbol{\theta_t = \theta_{t-1} - \alpha \cdot m_t} \quad \textbf{(18)}$$

where θ_t is the parameter at time t, and θ_{t-1} is the parameter at previous step. $\alpha \cdot m_t$ is the change of parameter where α is the learning rate and m_t is the aggregate of gradients at time t. m_t can be calculated as equation 19:

$$\boldsymbol{m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) * grad(\theta_{t-1})} \quad \textbf{(19)}$$

where β is average parameter, and $grad(\theta_{t-1})$ is the gradient of the parameter in the previous step, and it is calculated as the division of the derivative of loss function over the derivative of the parameter at time t.

$$\boldsymbol{grad(\theta_{t-1}) = \frac{\partial L}{\partial \theta_t}} \quad \textbf{(20)}$$

On the other hand, RMSP updates the parameter based on the moving average of the squared gradients.

$$\boldsymbol{\theta_t = \theta_{t-1} - \alpha \cdot \frac{grad(\theta_{t-1})}{\sqrt{V_t}+e}} \quad \textbf{(21)}$$

where e is a numerical constant to avoid denominator to be zero, and V_t is the moving average of squared gradients, and it can be calculated as equation 22:

$$\boldsymbol{V_t = \beta_2 \cdot V_{t-1} + (1 - \beta_2) \cdot grad(\theta_{t-1})^2} \quad \textbf{(22)}$$

The whole optimization process can be summarized as steps below [78]:

1. All the parameters are initialized at time=0.

$$\begin{cases} \boldsymbol{m_0 = 0} \\ \boldsymbol{V_0 = 0} \\ \boldsymbol{t = 0} \end{cases} \quad \textbf{(23)}$$

2. Then while parameter does not converge, the time will go to the next step. And the gradients are first updated.

$$\boldsymbol{t = t + 1} \quad \textbf{(24)}$$

$$\boldsymbol{g_t = grad(\theta_{t-1}) = \frac{\partial L}{\partial \theta_t}} \quad \textbf{(25)}$$

3. Then we calculate the first and second moment m_t and V_t

$$\boldsymbol{m_t = \beta \cdot m_{t-1} + (1 - \beta_1) * g_t} \quad \textbf{(26)}$$

$$\boldsymbol{V_t = \beta \cdot V_{t-1} + (1 - \beta_2) \cdot g_t{}^2} \quad \textbf{(27)}$$

4. During the next stage, we compute the bias-corrected first and second moments.

$$\boldsymbol{\widehat{m_t} = \frac{m_t}{(1-\beta_1)}} \quad \textbf{(28)}$$

$$\boldsymbol{\widehat{V_t} = \frac{V_t}{(1-\beta_2)}} \quad \textbf{(29)}$$

5. And the last step is updating the parameter.

$$\boldsymbol{\theta_t = \theta_{t-1} - \alpha \cdot \frac{\widehat{m_t}}{\sqrt{V_t}+e}} \quad \textbf{(30)}$$

4.7.3 Cyclegan Implementation

Table 4 shows all the parameters used for building the discriminator network.

Table 4: Discriminator network parameters

	Input Size	Output Size
Input Layer	N/A	256,256,3
First Layer (C64)	256,256,3	128,128,64
Second Layer (C128)	128,128,64	64,64,128
Third Layer (C256)	64,64,126	32,32,256
Fourth Layer (C512)	32,32,256	16,16,512
Second Last Output Layer	16,16,512	16,16,1
Output Layer	16,16,1	N/A

The activation function used in the network is LeakyReLu with a negative slope coefficient of 0.2. LeakyReLu activation function is a variation of the ReLU activation function introduced above. Unlike ReLu, which has a value of 0 when x is smaller than 0, LeakyReLu is a function that has a small slope coefficient (in our case equals 0,2). The function equals x when x is greater than 0, which is the same as the ReLU. LeakyRelu activation function is shown as equation 31, and can be visually interpreted in Figure 23 [79]:

$$f(x) = \begin{cases} 0.2x & x < 0 \\ x & x \geq 0 \end{cases} \quad \textbf{(31)}$$

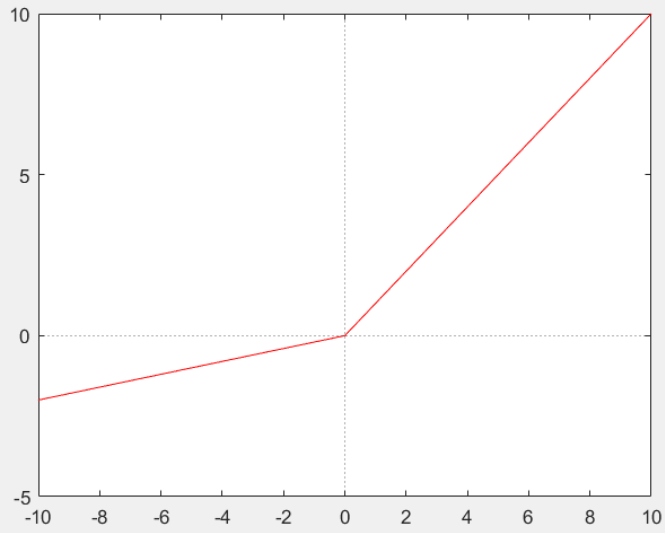

Figure 23: LeakyReLU activation function

The discriminator losses are measured by mean square error (MSE), with the loss function as equation 32 [80]:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \quad (32)$$

where Y_i is the real value, $\hat{Y}_i$ is the predicted value.

A discount factor of 0.5 was used on the loss to make the learning rate of the discriminator slower than the generators. Specifically, the learning rate of discriminator is set to be 0.5 times slower than the learning rate of the generator. The network is trained with the Adam optimizer at a learning rate of 0.0001. Table 5 presents the generator network configuration.

Table 5: Generator network parameters

	Input Size	Output Size
Input Layer	N/A	256,256,3
First Layer ((c7s1-64)	256,256,3	256,256,64
Second Layer (d128)	256,256,64	128,128,128
Third Layer (d256)	128,128,128	64,64,256
Residual Layers (R256)		
Fourth Layer (u128)	64,64,256	128,128,128
Fifth Layer (u64)	128,128,128	256,256,64
Sixth Layer (c7s1-3)	256,256,64	256,256,3
Output Layer	256,256,3	N/A

c7s1-64 is a 7*7 normalized layer with instance normalization and ReLU activation function (instanceNormReLU layer). This layer consists of 64 filters and stride 1. The second and third layers, with names starting with d followed by a number k (abbreviated as dk, similar to uk in the following descriptions), are 3*3 instanceNormReLU layers with k filters and stride 2. The residual layers (R256) consist of two 3*3 convolutional layers with 9 residual blocks concatenated to the output channel-wise. The fourth and fifth layers (denoted as uk) are 3*3 fractional-stride convolutional instanceNormReLU layers with k filters and stride 0.5. It is worth noting that, from the fourth layer, the convolution layers are transposed to convert the generated image back to original size (256*256*3). The final layer, denoted as c7s1-3, uses Tanh activation function with 3 filters and stride 1. Tanh activation can be written as the form in equation 33 [81]:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \quad (33)$$

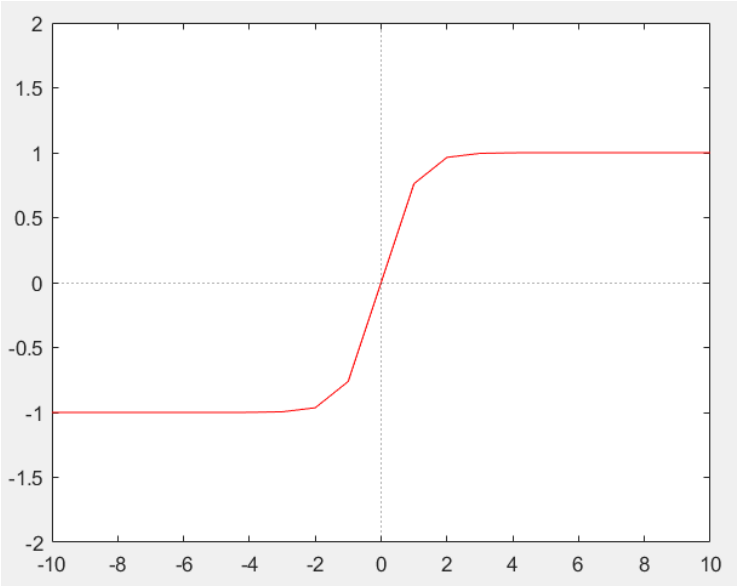

Figure 24: Tanh activation function

As shown in Figure 24, the main characteristic of the tanh activation function is that the output is within the range [-1,1]. For the CycleGAN generator, the pixel values of the images are normalized to [-1,1], so the Tanh activation function is the most suitable for the CycleGAN network to generate images within the same range.

Chapter 5
Defect Detection Algorithm

The next step is to choose an appropriate defect detection algorithm, which serves two purposes in the project:

1. The defect detection algorithm will be used to test and validate the effectiveness of the RailGAN algorithm developed in Chapter 4.
2. It will be implemented on the drone for the final product.

For the defect detection algorithm, there are two potential candidates have been proposed: VGG-16 and YOLO.

5.1 VGG-16

The first candidate for defect detection is VGG-16 [82]. Originally named "Very Deep Convolutional Networks for Large-Scale Image Recognition," it was developed by Visual Geometry Group in Oxford University and has a depth of 16 convolutional layers, including 13 convolutional layers and 3 fully connected layers. The main architecture of VGG-16 is shown in Figure 25.

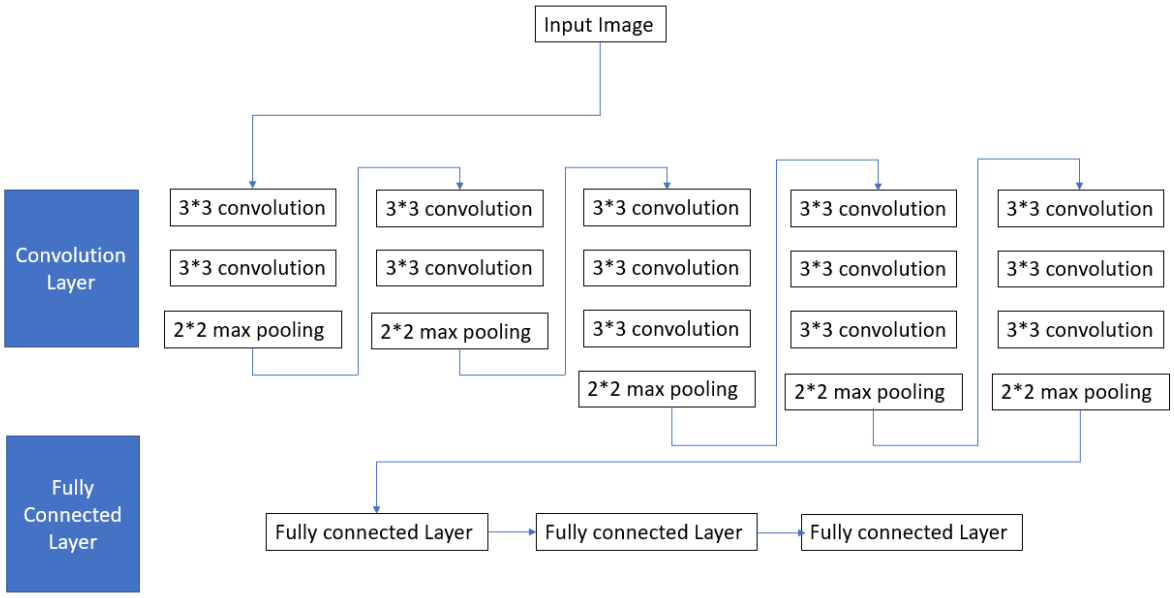

Figure 25: VGG-16 structure

As shown in Figure 25, the input image first goes through convolution layers, which has 5 different sections. During each section, multiple 3*3 convolutions are applied to the input images, followed by a 2*2 max pooling. In this structure, a stack of two 3*3 convolution filter serves as an effective receptive field of 5*5, and a stack of three 3*3 convolution filter serves as a 7*7 filter. The reason for breaking up a larger filter into a stack of 3*3 convolutions is that higher accuracy in distinguishing the decision function, and fewer parameters are required to tune to imporve the performance of smaller filters. These different sections reduce the size of the images.

Following these 5 sections, the input images are flattened, and then go through three fully connected layers. The first two layers have 4096 neurons, and the last layer has 1000 neurons, which relate to 1000 possible classes in the original VGG-16 architecture. However, for our railway track detection task, we only need to differentiate between defective and non-defective railways, so the last layer is modified to have only 2 neurons, one for each class. VGG-16 uses the Softmax activation function for the last layer, which is appropriate for multi-class classification tasks. In the future, the project will identify different types of surface defects, so Softmax activation is considered the most appropriate for this purpose. Softmax activation function converts the output values to a probability distribution over the possible classes, with the equation written as [83]:

$$\boldsymbol{f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}} \quad \textbf{(34)}$$

Suppose there are different classes in x, and the weight for these classes can form an arbitrary vector. In Equation 34, x_i is the ith index in this vector, N is the total number of classes [84]. For better understanding of this equation, Softmax function can be considered as taking a vector with the weight of different classes as inputs and applying an exponential function to each element in the vector, followed by a normalization process by dividing by the summation of all the exponential values in the vector. Softmax ensures that the total of all values in the vector adds up to 1, making it suitable for predicting probability distribution, particularly for multi-class classification tasks such as the defect detection in this project.

To train the network, the images are divided into two categories: defective railways and non-defective railways. During the training, the network learns to map each defect image to its corresponding category. For actual implementation, VGG-16 outputs the possible category that the image belongs to along with its corresponding probability.

5.2 YOLO-Network

YOLO (You Only Look Once) is selected as the second algorithm to validate the fabricated images due to its high overall performance in terms of speed and accuracy compared to other object detection algorithms [85]. Unlike other object detection methods such as RCNN, Faster RCNN or FPN, which use two-stage detection methods [86], YOLO uses a one-stage detection method with a single neural network that combines the two processes by direct regression.

5.2.1 YOLO-Architecture

YOLO was originally modified from the GoogLenet, and its architecture includes a total of 24 convolutional layers with 2 fully connected layers at the end, as shown in Figure 26 [87]:

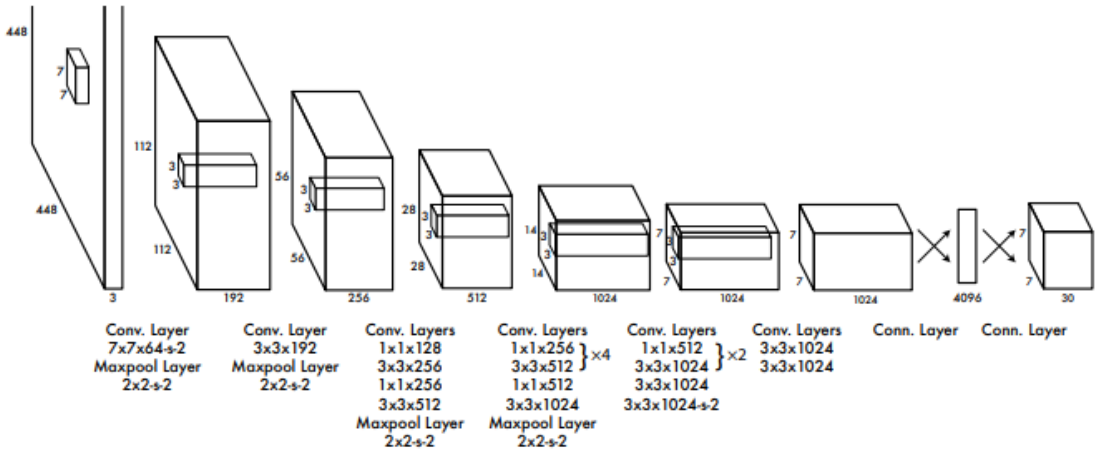

Figure 26: Architecture of YOLO

The first 24 convolutional layers are used for image classification by extracting the image features. This architecture is similar to the GoogLenet, however, YOLO simplifies the GoogLenet by replacing the idea of the inception module with a combination of a 1*1 reduction layer with a 3*3 convolutional layer, except for the first layer that has a (7*7) filter [88]. The last two fully connected layers are used for prediction. YOLO-v5 uses the Leaky ReLU activation function in the hidden layers and sigmoid for the detection layer.

As for the optimizer, YOLO originally suggests stochastic gradient descent (SDG). However, as discussed in section 4.7.2, Adam performs better than the traditional SDG, so for the training our defect detection algorithm, we have decided to use the Adam optimizer instead.

5.2.2 Principle for Defect Detection

First, YOLO splits the input image into an S*S grid. Instead of having large grids that contain the whole object that needs to be detected, these small grids are created to only contain the center of these objects. In order to achieve object detection, each grid is responsible for predicting B bounding boxes, and these bounding boxes have 5 values: the coordinates of the center of the box (x,y), the size of the bounding box (width and height), and the confidence score. The confidence score represents the probability that an object is contained within the bounding box. For example, in Figure 27, the images are divided into 7*7 grids. Each grid has 2 bounding boxes, resulting in a total of 98 bounding boxes. There are also different weights of lines for bounding boxes, which symbolize different confidence score (the thicker line means higher confidence, and thinner line means lower confidence).

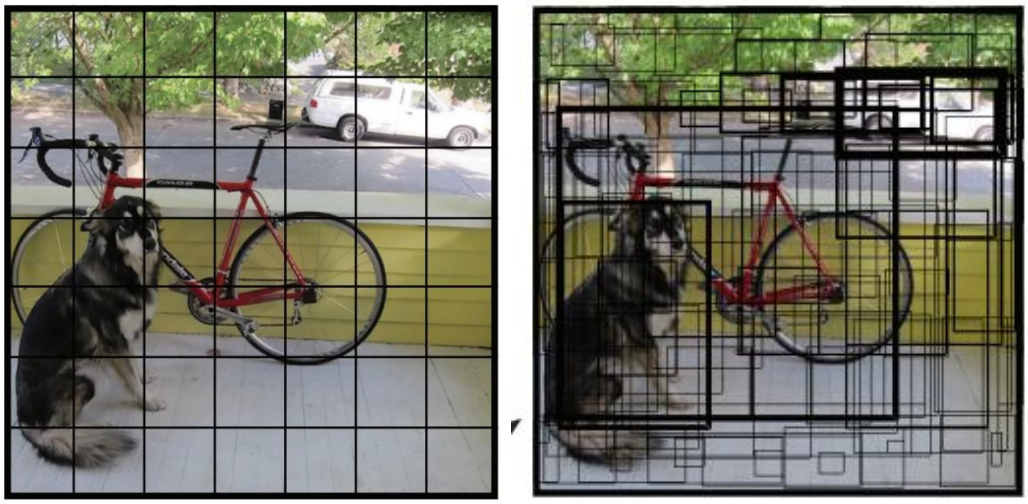

Figure 27: Example of YOLO

The first four values enable the bounding box to locate the object, but they need to be normalized into the range between 0 to 1 to be fed into the CNN. For x and y, these two values only need to be divided by the size of the grid. The width and height of the bounding boxes can be normalized by dividing them by the width and height of the whole image. The confidence score is calculated as

$$\boldsymbol{confidence} = \mathbf{Pr}(\boldsymbol{obj}) * \boldsymbol{IOU}_{\boldsymbol{truth}}^{\boldsymbol{pred}} \quad \textbf{(35)}$$

where $\Pr(obj)$ is the probability of a grid containing an object, and IOU is the intersection over union between predicted bounding box and the ground truth. The principle of IOU is represented in Figure 28 and IoU is defined as:

$$IoU = \frac{truth \cap pred}{truth \cup pred} \quad (36)$$

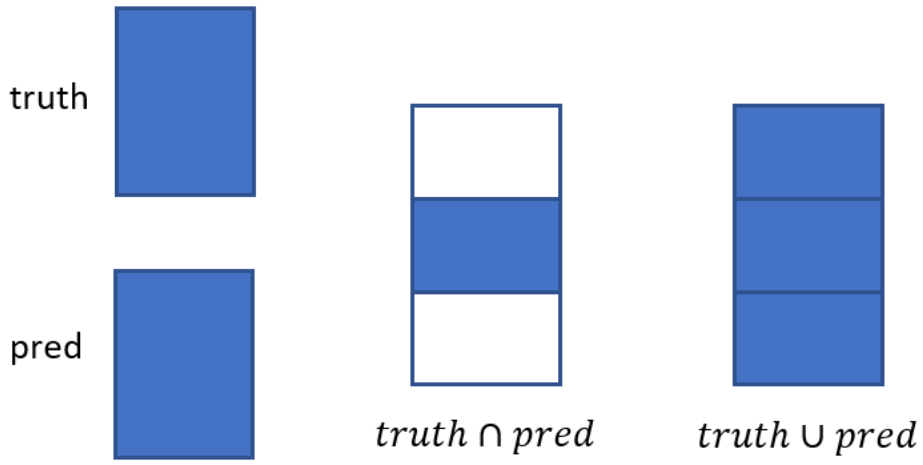

Figure 28: Principle of IoU

The YOLO network can also predict the probability of a grid cell containing an object and its class-specific confidence score. The object can be located by multiplying the confidence score with the class probability. This is represented by equation 37, where the class probability is conditioned on the object being present in the cell.

$$\Pr(Class|Object) * \Pr(obj) * IOU_{truth}^{pred} = \Pr(Class) * IOU_{truth}^{pred} \quad (37)$$

There can be different grid cells that predict the same object. By setting the threshold of IOU, YOLO can discard one that is lower than the threshold (these grids is considered irrelevant to the class). However, this does not solve the problem of having multiple bounding boxes that contains the same object that are all over the threshold. This can cause noise during the training. To tackle this issue, YOLO uses NMS to keep the boxes that has the largest confidence score and remove the one with smaller confidence.

5.3 Discussion

For the defect detection, the first step is to take the video and use the defect detection algorithm to locate any defects from the video, and the next step is to implement such defect detection algorithm on the drone to achieve real-time defect detection.

When comparing VGG-16 with YOLO-v5, VGG-16 can only perform image classification (defect and non-defect), while YOLO-v5 can not only perform classification, but can also localize the object such as defects in the image. Since VGG-16 has a less complex task to perform, the processing and training time are faster compared to YOLO-v5.

If VGG-16 is used as the solution, the video or the live stream will be processed frame by frame, and the algorithm can only identify the frames that contain the defect, i.e. defective images. On the other hand, if we use YOLO-v5, although the processing speed would be slower, the algorithm can detect and locate the defects on the live stream. This would allow the algorithm to output the image of the defect in a bounding box, as well as information such as its shape, width, depth and location on the railway track, which can be analyzed by mechanics to facilitate the repair process.

Considering all these aspects, we have selected YOLO-v5 as the defect detection algorithm for this project.

Chapter 6
Experiment and Results

This chapter will provide an overview of the training, testing and validation process for the different CNN architectures described in Chapter 5 and 6. Additionally, the results of the generated images by the RailGAN proposed in Chapter 5 and the ability to detect defects using YOLO-v5, as introduced in Chapter 6, will be presented.

6.1 Image Segmentation

6.1.1 U-Net Training

To train the U-Net, two sets of images were required: railway images and their corresponding segmentation map output. The segmentation map for each image was manually labeled using online software (labelstudio.io) by highlighting the track area. The dataset used to train the U-Net consisted of 120 real-time railway images taken from the drone and their corresponding masked images labelled on the railway track surface. Examples of the created dataset are presented in Figure 29.

Figure 29: Real-time railway images (top row) and masked images (bottom)

The effectiveness of the image segmentation model is usually evaluated by two criteria: accuracy and dice score. Accuracy is defined as the number of correctly labeled pixel values over the total number of pixels in all images, and it can be represented by equation 38.

$$\boldsymbol{accuracy} = \frac{\boldsymbol{TP+TN}}{\boldsymbol{TP+TN+FP+FN}} \tag{38}$$

where TP is true positive, TN is true negative, FP is false positive, and FN is false negative. The Dice score, on the other hand, is defined as Equation 39 [8]:

$$\boldsymbol{dice\ score}(\boldsymbol{A},\boldsymbol{B}) = \frac{\boldsymbol{2TP}}{\boldsymbol{2TP+FP+FN}} = \frac{\boldsymbol{2|A\cap B|}}{\boldsymbol{|A|+|B|}} \tag{39}$$

where A is the predicted result from U-net and B is the true masked image from the dataset.

The U-net was trained with two sets of datasets using the learning rate of 0.00001 and was initially set to train for 200 epochs. The accuracy and dice score vs number of epochs are shown in Figure 30.

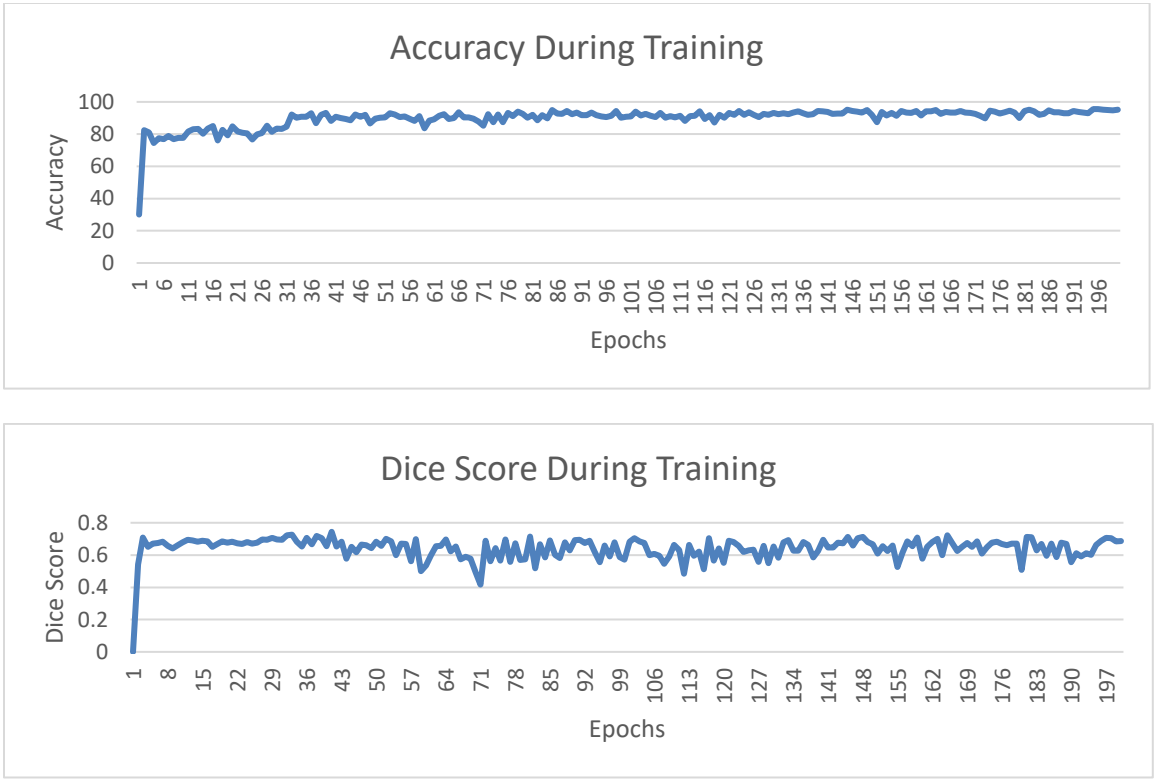

Figure 30:Accuracy and dice score during training

Figure 30 shows that the accuracy of the U-net model started at 30% and quickly reached 90% at around 35 epochs. This was due to the small amount of training dataset for the network. At 200 epochs, the U-net model achieved a high accuracy of 96%, which was considered sufficient to be adopted for the RailGAN.

On the hand, the dice score of the U-net model, which is shown in Figure 30, started at 0 and rose to about 0.7 at the fourth epochs, which was an acceptable dice score for image segmentation. Throughout the training process, the dice score fluctuated between 0.4 to 0.7. This will be further discussed in the Chapter 7.

The trained U-Net model was tested and evaluated with 20 different railway images found online, along with their corresponding masked images. The results of some test images are shown in Figure 31, indicating that the trained U-Net model can successfully segment the railway track from the background regardless of light condition and pixel values.

Figure 31: Result of U-Net after 100 epochs

In order to determine the most effective method for image segmentation, the performance of U-Net was compared with the image filter techniques. Both methods were applied to same set of real-time images by following the flowchart described in sections 4.7.1 and 4.7.2, as illustrated in Figure 32. The original

railway images are shown in the first column, and the results of segmentation using image filters and U-Net are shown in the second and third columns, respectively.

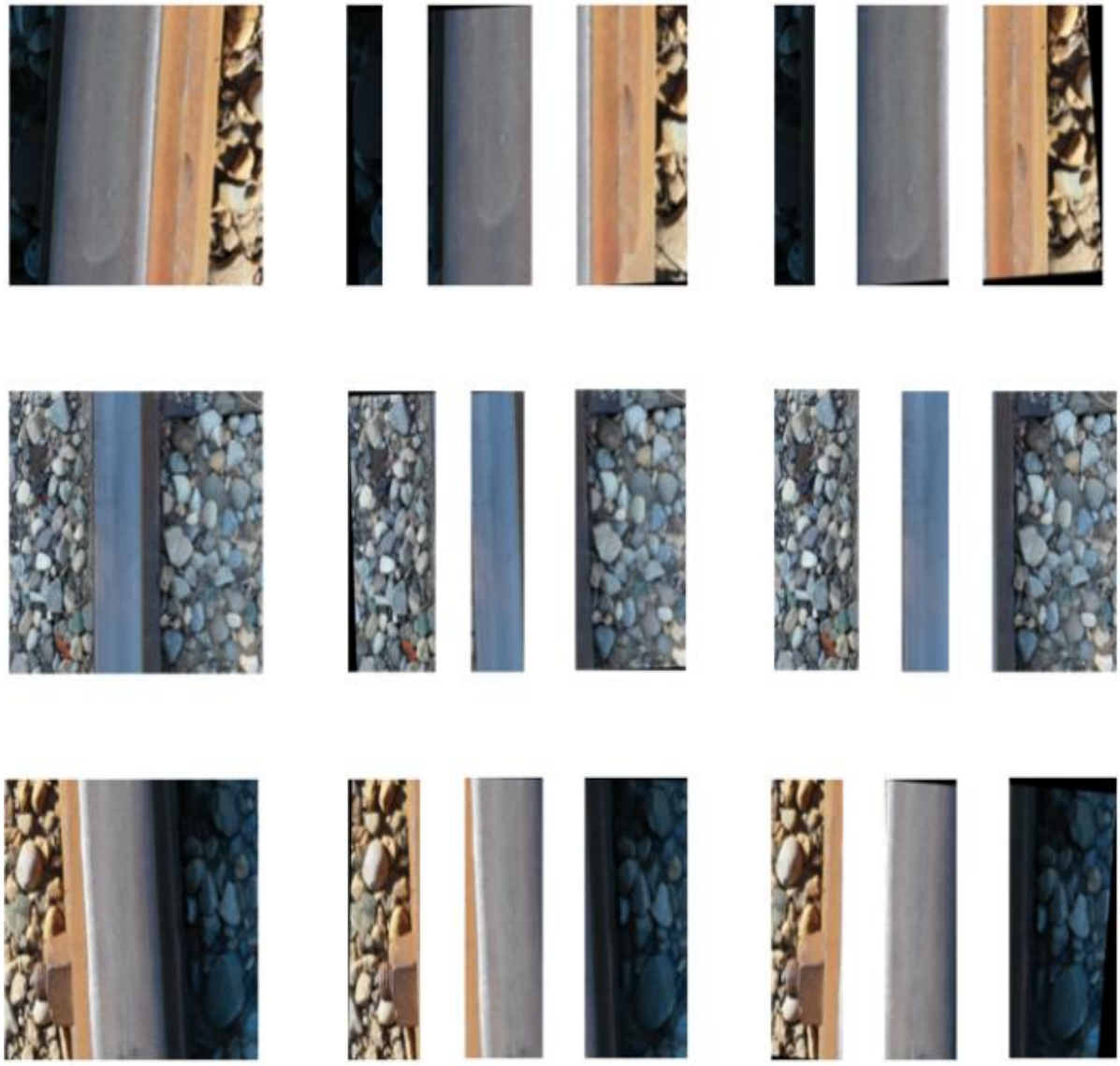

Figure 32: Left column: original images; center column: image segmentation by image filters; right column: image segmentation by U-Net.

When image filter techniques were used to perform railway track segmentation, the threshold value had to be adjusted according to pixel intensity values of the railway track, which were affected by lighting conditions in the images. In the first row of Figure 32, the threshold had to be set between 130 to 220;

in the second row, the threshold was set between 170 and 220. In the third row, since the pixel value of the railway track was higher than the background, an inverse binary thresholding between 160 to 250 had to be used. However, the U-Net approach produced more consistent results regardless of the pixel intensity values of the track surfaces in all three images. Besides, some pixels were mis-categorized in all of the images by image filter technique. The railway track region segmented in the second column included part of the background, and the background also contained some part of railway track surfaces, whereas the U-Net was able to perform a more reliable separation on the track surface from the background, as shown in the third column. Therefore, U-Net was selected as the image segmentation solution for the RailGAN.

6.2 RailGAN Implementation

After selecting U-net as the image segmentation method, by following the flowchart in Figure 20, the CycleGAN model was trained to generate defects on the railway track area only. However, there were only 15 defective railway surface images available for the development of the model from online resources. To overcome the lack of dataset, some image manipulation methods, such as image rotation, resizing, cropping, and contract adjustment, were applied to the existing datasets to expand it to 100 images. These images, along with 100 non-defect images, were first segmented by U-net, and the railway track region of these images were sent to the CycleGAN for training. During training process, the model was saved after every 120 epochs. By observing each model output, the network learned the mapping and improved its performance during training. However, after 1800 epochs, the model began to show signs of overfitting and became unstable, producing images with dramatical changes in color tones. The best performing model in the validation process was the one trained for 1800 epochs. Example images generated by both generator 1 and generator 2 of the model can be shown in Figure 33 and Figure 34, respectively.

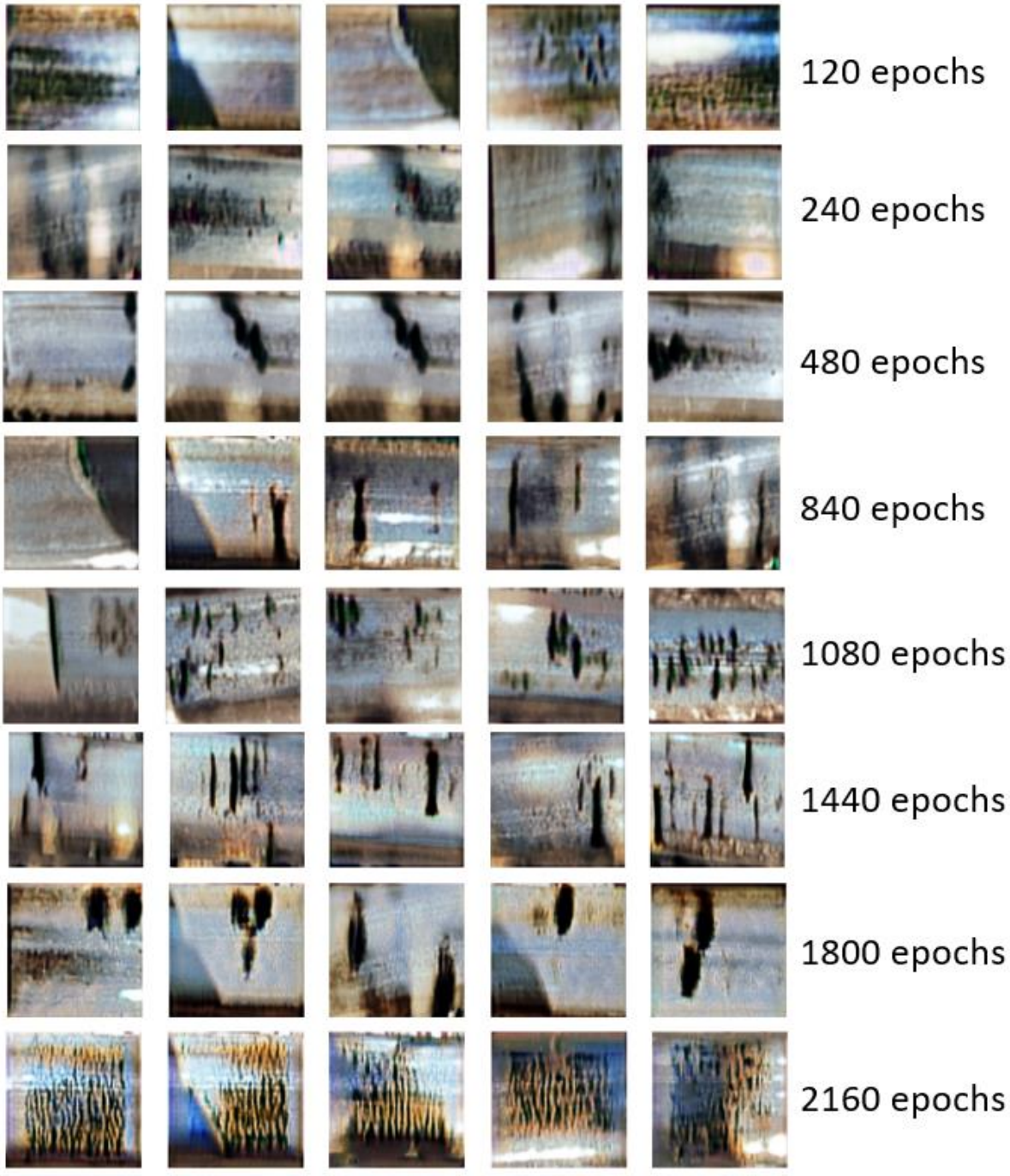

Figure 33: Training progress of generator 1 (non-defect to defect)

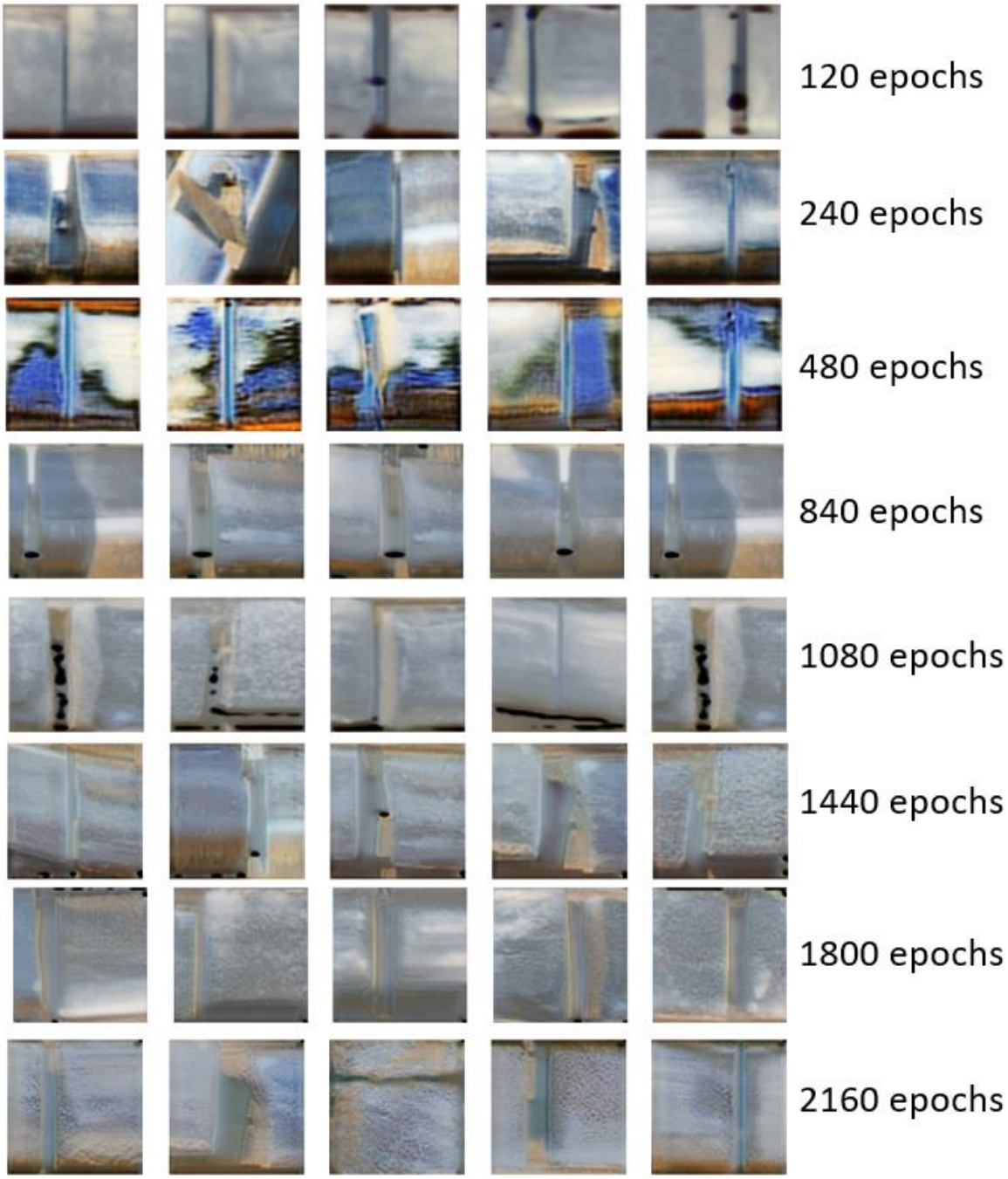

Figure 34: Training progress from defect to non-defect

The RailGAN model was developed by combining the trained U-net segmentation model from section 6.1.1 with the CycleGN model trained. To further evaluate the performance of RailGAN, this model

was applied to 20 images captured from real-time video taken with a drone. These images contained railway tracks taken from different angles and under different light conditions, and some examples are shown in Figure 35.

Figure 35: Images generated from RainGAN

However, there still requires the confirmation of the effectiveness of the RailGAN compared to the original CycleGAN model in terms of generating railway track defects, so a comparison test between two models was conducted. First, the Cyclegan model was re-trained using the same 200-image dataset (100 with defect and 100 non-defect), but this time without the image segmentation by U-Net. Then, both RailGAN model and CycleGAN were applied to the same test images, and the results are shown in Figure 36. Figure 36(b) shows the outputs of the CycleGAN model, which has no control over where the defects are generated. In contrast, Figure 36(c) demonstrates that the RailGAN model performed significantly better by generating defects only on the track surface. Moreover, all of the defect images generated by the RailGAN resemble real cracks on the railway surface, indicating that these images can be effectively utilized for defect identification and classification algorithm in the next stage.

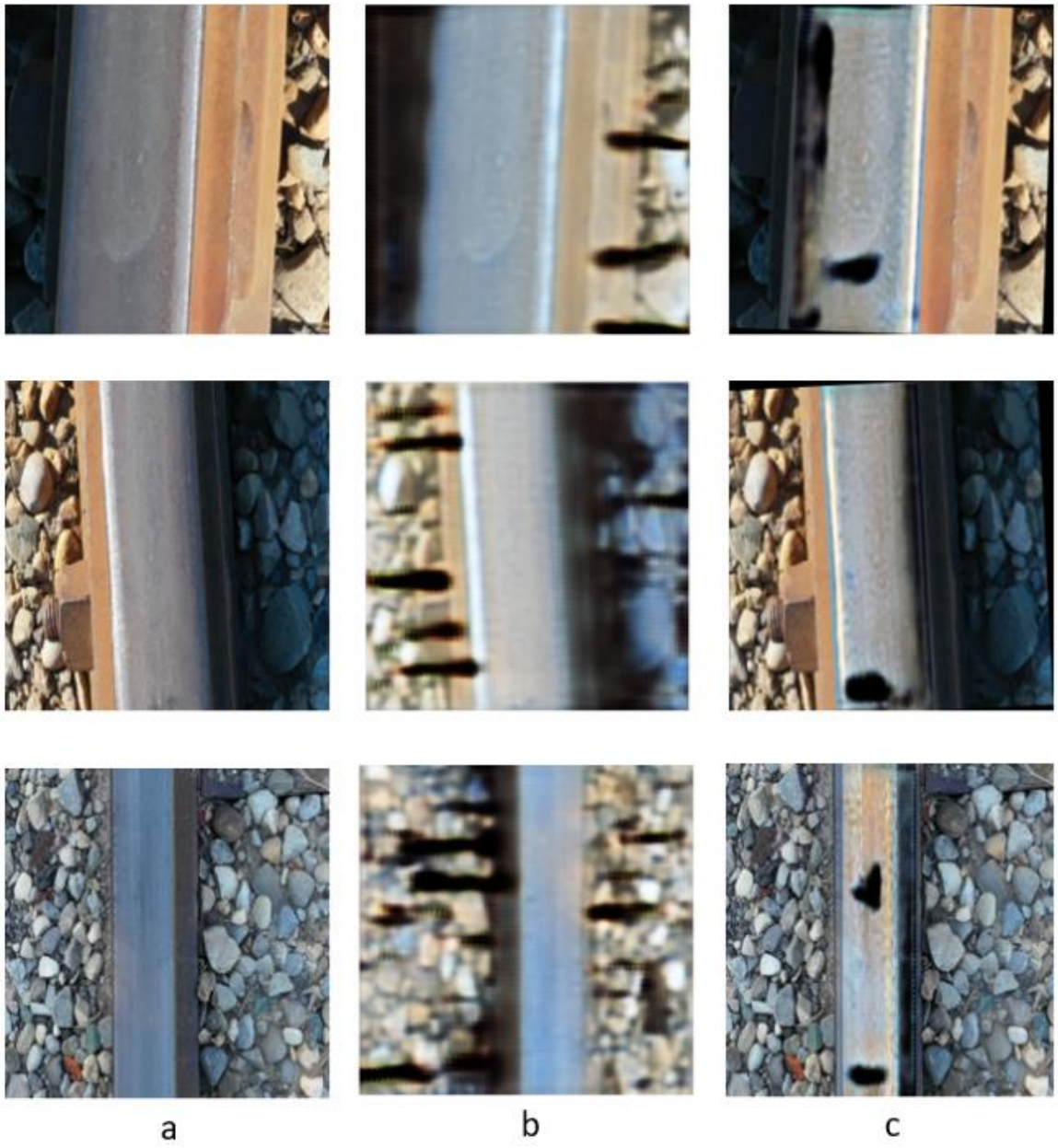

Figure 36: Three example images with different lighting conditions and angles: (a) live image of railway track; (b) results from the original CycleGAN model; (c) results from the RailGAN model.

6.3 Defect identification algorithm

Using the RailGAN algorithm, a total of 100 defect images with different crack patterns and lighting conditions were generated. Then the defects in these fabricated images were labelled by Roboflow, and the dataset was split into training and validation sets in a 9:1 ratio. Some examples of the labelled dataset are demonstrated in Figure 37.

Figure 37: Example of labelled images

These fabricated images, coupled with the coordinates of corresponding label bounding box in each image, were sent to the YOLO-v5 network by Roboflow for training. The training process was originally set with 2000 epochs with learning rate of 0.0001, and Roboflow automatically saved the model with the best performance. For YOLO, the performance of the model was evaluated by precision, recall, mean average precision with threshold over 0.5 (mAP50). The precision measures the accuracy of the prediction, and is defined by equation 40 [90]:

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP+FP}} \tag{40}$$

On the other hand, recall measures the model's ability to find the positives, and is defined as equation 41 [90]:

$$\boldsymbol{Recall} = \frac{\boldsymbol{TP}}{\boldsymbol{TP+FN}} \tag{41}$$

From the precision and recall values obtained for a certain class, the precision-recall (PR) curve for this particular class can be obtained. Figure 38 shows the PR curve for the class defect:

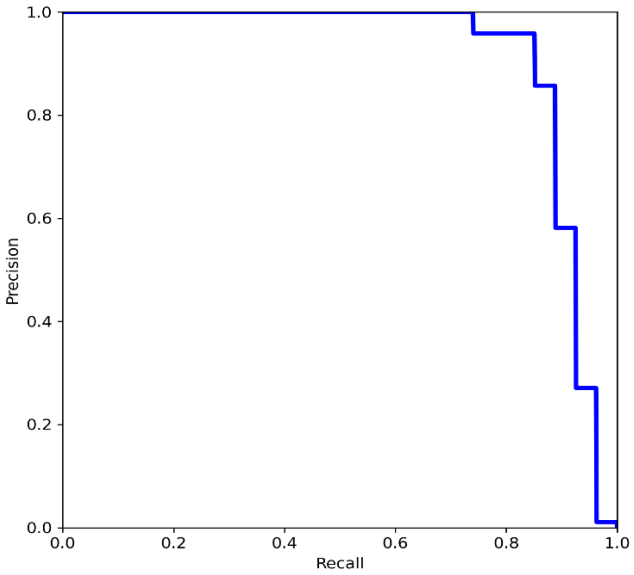

Figure 38: PR curve for YOLO training

Suppose there are k number of classes in total, the mAP can be defined as [91]:

$$\boldsymbol{mAP@t = \frac{1}{k}\sum_{0}^{k-1} AP@t} \quad \textbf{(42)}$$

where t is the threshold, and AP is the area under the PR curve.

The loss, precision, recall and mAP50 during training are visually presented in Figure 39.

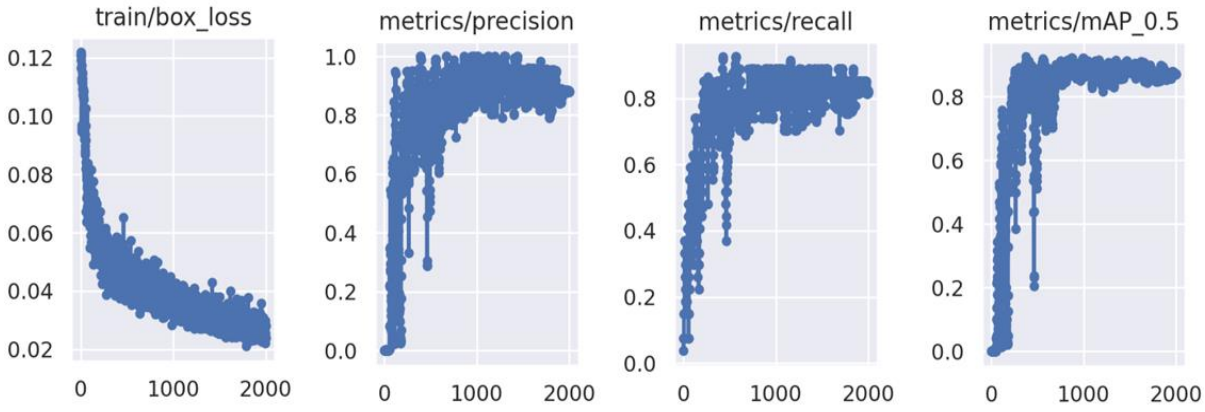

Figure 39: Loss, precision, recall and mAP50 during training

During the training process, the loss decreased while the precision, recall, and mAP 0.5 increased. However, a sign of overfitting was observed as the precision, recall and mAP 0.5 slightly decreased towards the end of the training. The model reached the best overall performance after 1348 epochs, which is automatically selected by Roboflow, by having 95.8% precision, 83.6% recall and 91.1% mAP50. This model was saved out for testing and validation.

This trained YOLO model was then applied to the original defect images to verify the effectiveness of the algorithm on real defect images for testing. Among 10 real defect images with cracks found online, the trained YOLO model was able to identify 9 defects, as shown in some examples in Figure 14.

Figure 40: Example of crack defects identified by YOLO algorithm

Chapter 7
Conclusion and Future Work

7.1 Conclusion

Considerable amount of work has been made in this project of achieving autonomous railway defect recognition. There are different types of railway defects, and this project aimed to tackle the surface defect that can develop and initiate other types of defects.

The literature review in Chapter 2 identified multiple attempts to tackle railway defect recognition. And among different NDT methods, visual inspection was considered to be the most suitable since it is portable and does not require external excitation applied to the whole railway track. The drone was then selected as the most suitable platform for visual inspection. Thus, the whole project was divided into the drone navigation on the railway and defect identification algorithm.

In Chapter 3, the drone hardware and software components were discussed, along with the navigation system that enables autonomous flight. For the drone navigation, the key process is to find the railway track in the images. In the project, two different ways were proposed. The first method is template matching, which use a template of a small section of the railway track to find the railway track region. The second method is to apply different filters to achieve railway edge detection. By comparing both methods, the latter one has a more consistent performance since template matching requires the drone flying at a fixed height, and the threshold needs to be adjusted according to the intensity value of the railway track.

Chapter 4 proposed RailGAN model to generate artificial defect images of railways. The images first went through the track segmentation, which could separate railway track from the background. Two different methods were considered to perform this task. In the first method, image filters such as the Gaussian filter and Sobel filter were used to extract the railway track surface, while the second method adopted the CNN method U-Net. To train U-net, 120 railway images along with their labeled masked images were used. After 200 epochs, the model reaches an accuracy over 96% and a dice score close to 0.7, which is acceptable for image segmentation method. Comparison of the results from these two methods indicated that the U-Net model could achieve more consistent results, while the threshold values for the image filters were highly dependent on the pixel intensity values of the railway track surface.

Then, the trained U-net model is used to perform image segmentation on railway track surface which is later used for the defect generation. The defect generation method selected was based on CycleGAN, followed by splicing the background section back to finish the whole RailGAN model process. During the training process, the CycleGAN model is saved every 120 epochs, and the best model is accepted at 1800 epochs. By applying the RailGAN to live footages taken by a drone, it was evident that RailGAN performed significantly better than the original CycleGAN model. RailGAN could successfully generate defects only on the railway surface, while the original Cyclegan model tended to generate defects in the background.

With the RailGAN model, the defective railway images are generated. These images are labeled by RoboFlow, and the images as long as the bounding boxes are sent to the YOLO-v5 model for the final defect identification algorithm. And the result show by applying the trained model to the defective railway track images, YOLO can successfully identify 90% of all the defects.

7.2 Future work

Although this project achieved significant progress in the development of the RailGAN model and defect detection algorithm, there is still more work to be done to successfully build a prototype for autonomous railway defect recognition. Due to the time constraints, the project mainly focused on these key elements, but future work should also consider other important aspects such as drone autonomous flying.

In addition, the developed defect detection algorithm can also be implemented on other platforms such as a railway cart, which can provide more flexibility and accuracy in detecting defects. Furthermore, the development of RailGAN also inspires future projects in terms of image generation, which can be applied to other areas such as medical imaging and computer vision. Therefore, future work can explore the potential of the developed technologies in other fields and improve their performance and efficiency.

7.2.1 Drone Control

Currently, all the track detection and autopilot algorithms were developed and simulated on a powerful desktop PC in Python scripts; however, they have not been integrated with the live drone control. The immediate next step for this regard is to migrate the developed algorithms and functions to the drone controller with the SDK provided by DJI. SDK enables a development of a mobile

application. Then the mobile device needs to connect to the controller through wire connection, and the controller will communicate the command from SDK to the drone through radio signal. The current progress to achieve this is getting the video live stream from the drone camera. The video will be processed by the track detection algorithm with the power of the mobile device. Once all these algorithms are compiled and loaded on the drone, the iterative process of R&D to fine tune the models and algorithms will take place. Online and offline testing on the drone will be carried out. To help streamline the process, the team will also try to implement an on-board WIFI emitter to establish live wireless communication between the drone and the ground control.

7.2.2 Defect Detection Algorithm

Currently, all the algorithms developed have shown the potential to achieve defect recognition. First some fine-tunning still need to be done for all these networks. For example, the U-net still achieve a relatively low dice score, so more research will be done on the activation function to increase the performance of the model. Next, the project will be focusing on generating a large patch of defective railway images by RailGAN network. As introduced in Chapter 6, the current RailGAN model have only generated 120 defective images, and we are looking at having over one thousand images as the defective dataset, and YOLO-v5 model will be retrained with this larger dataset.

As mentioned above, the whole program will be implemented on the drone, so once the defect detection algorithm is trained and validated, the network will be saved as json file in order to be used in JAVA, and the SDK application developed will use this algorithm to achieve real-time defect detection.

References

[1] "Annual Report 2019," 2019. Accessed: Oct. 31, 2021. [Online]. Available: https://media.viarail.ca/sites/default/files/publications/2019%20VIA%20RAIL%20AR_ENGLISH.pdf

[2] "Annual Report 2020 ," 2020. Accessed: Oct. 31, 2021. [Online]. Available: https://media.viarail.ca/sites/default/files/publications/Annual_report_2020_EN.pdf

[3] "Rail transportation occurrences in 2020 - Statistical Summary - Transportation Safety Board of Canada." https://www.bst-tsb.gc.ca/eng/stats/rail/2020/sser-ssro-2020.html#1.0 (accessed Oct. 31, 2021).

[4] D. Paul, "Some Aspects of the Hither Green Rail Disaster:," http://dx.doi.org/10.1177/002581727003800303, vol. 38, no. 3, pp. 62–73, Jun. 2016, doi: 10.1177/002581727003800303.

[5] "Hither Green rail crash." https://amp.freejournal.info/5158697/1/hither-green-rail-crash.html (accessed Oct. 31, 2021).

[6] "Railway Investigation Report R15M0034 - Transportation Safety Board of Canada." http://www.tsb.gc.ca/eng/rapports-reports/rail/2015/R15M0034/R15M0034.html (accessed Nov. 01, 2021).

[7] N. S. Mishra, V. Ramaswamy, and S. Mishra, "Defects in rails," vol. 9, pp. 345–369, 1986.

[8] Nordco Rail Flaw Defects Identification Handbook. Accessed: Oct. 31, 2021. [Online]. Available:https://www.nordco.com/Media/Assets/GeneralFiles/NordcoRailFlawDefectsIdentificationHandbook.pdf

[9] "Nondestructive Evaluation Techniques." https://www.nde-ed.org/NDETechniques/ (accessed Oct. 31, 2021).

[10] C. Ibarra-Castanedo, J. R. Tarpani, and X. P. Maldague, "Nondestructive testing with thermography," *European Journal of Physics*, vol. 34, no. 6, 2013.

[11] "CNDT Thermography." https://www.cityu.edu.hk/seam/CNDT.Thermography.html (accessed Oct. 31, 2021).

[12] C. Tuschl, B. Oswald-Tranta, and S. Eck, "Inductive Thermography as Non-Destructive Testing for Railway Rails," Applied Sciences, vol. 11, no. 3, Jan. 2021, doi: 10.3390/app11031003.

[13] D. Peng and R. Jones, "Modelling of the lock-in thermography process through finite element method for estimating the rail squat defects," Engineering Failure Analysis, vol. 28, pp. 275–288, Mar. 2013, doi: 10.1016/J.ENGFAILANAL.2012.10.024.

[14] U. Netzelmann, G. Walle, A. Ehlen, S. Lugin, M. Finckbohner, and S. Bessert, "NDT of railway components using induction thermography," *AIP Conference Proceedings*, 2016.

[15] A. Sharma and A. K. Sinha, "Ultrasonic testing for mechanical engineering domain: present and future perspective," International Journal of Research in Industrial Engineering, vol. 7, no. 2, pp. 243–253, Sep. 2018, doi: 10.22105/RIEJ.2018.100730.1018.

[16] W. Gong, M. F. Akbar, G. N. Jawad, M. F. Mohamed, and M. N. Wahab, "Nondestructive Testing Technologies for Rail Inspection: A Review," *Coatings*, vol. 12, no. 11, p. 1790, 2022.

[17] R. Singh, "Ultrasonic testing," *Applied Welding Engineering*, pp. 347–358, Jan. 2020, doi: 10.1016/B978-0-12-821348-3.00028-8.

[18] H. Yılmaz and Z. Öztürk, "Investigation of rail defects using an ultrasonic inspection method: a case study of Aksaray-Airport Light Rail Transit Line in Istanbul," Jun. 2015. doi: 10.2495/UT150561.

[19] S. Mariani *et al.*, “Noncontact ultrasonic guided wave inspection of rails,” *Structural Health Monitoring*, vol. 12, no. 5–6, Sep. 2013, doi: 10.1177/1475921713498533.
[20] N. A. Sharif, R. Ramli, A. Z. Mohamed, and M. Zaki Nuawi, “Theory and development of magnetic flux leakage sensor for flaws detection: A review,” *International Journal of Advances in Applied Sciences*, vol. 8, no. 3, p. 208, Sep. 2019, doi: 10.11591/IJAAS.V8.I3.PP208-216.
[21] M. R. Kandroodi, B. N. Araabi, M. N. Ahmadabadi, F. Shirani, and M. M. Bassiri, “Detection of natural gas pipeline defects using magnetic flux leakage measurements,” May 2013. doi: 10.1109/IranianCEE.2013.6599681.
[22] A. G. Antipov and A. A. Markov, “Detectability of Rail Defects by Magnetic Flux Leakage Method,” *Russian Journal of Nondestructive Testing*, vol. 55, no. 4, Apr. 2019, doi: 10.1134/S1061830919040028.
[23] Y. Jia, K. Liang, P. Wang, K. Ji, and P. Xu, “Enhancement method of magnetic flux leakage signals for rail track surface defect detection,” *IET Science, Measurement & Technology*, vol. 14, no. 6, pp. 711–717, 2020.
[24] E. Deutschl, C. Gasser, A. Niel, and J. Werschonig, “Defect detection on rail surfaces by a vision based system.” doi: 10.1109/IVS.2004.1336435.
[25] X. Zhang, N. Feng, Y. Wang, and Y. Shen, “Acoustic emission detection of rail defect based on wavelet transform and Shannon entropy,” *Journal of Sound and Vibration*, vol. 339, pp. 419–432, Mar. 2015, doi: 10.1016/J.JSV.2014.11.021.
[26] J. BOHSE and A. J. BRUNNER, “Acoustic emission in delamination investigation,” Delamination Behaviour of Composites, pp. 217–277, 2008.
[27] “DJI Developer.” https://developer.dji.com/payload-sdk/ (accessed Oct. 31, 2021).
[28] “Four drone manufacturers providing SDKs - RIIS.” https://riis.com/blog/four_drone_sdks/ (accessed Oct. 31, 2021).
[29] “Mavic Air 2 - Specifications - DJI.” https://www.dji.com/ca/mavic-air-2/specs (accessed Oct. 31, 2021).
[30] “Mavic 2 - Product Information - DJI.” https://www.dji.com/ca/mavic-2/info#specs (accessed Oct. 31, 2021).
[31] “Waypoint Flights.” https://www.measure.com/help/autonomous-flight-with-waypoint (accessed Oct. 31, 2021).
[32] “waypoint | National Geographic Society.” https://www.nationalgeographic.org/encyclopedia/waypoint/ (accessed Oct. 31, 2021).
[33] “DJI Waypoints Mode Explained.” https://www.tomstechtime.com/waypoints-mode (accessed Oct. 31, 2021).
[34] M. B. Hisham, S. N. Yaakob, R. A. A. Raof, A. B. A. Nazren, and N. M. Wafi, “Template Matching using Sum of Squared Difference and Normalized Cross Correlation,” Dec. 2015. doi: 10.1109/SCORED.2015.7449303.
[35] A. Basulto-Lantsova, J. A. Padilla-Medina, F. J. Perez-Pinal, and A. I. Barranco-Gutierrez, “Performance comparative of opencv template matching method on jetson TX2 and Jetson Nano Developer kits,” *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, 2020.
[36] “Object detection,” *OpenCV*. [Online]. Available: https://docs.opencv.org/3.4/df/dfb/group__imgproc__object.html#gga3a7850640f1fe1f58fe91a2d7583695da5382c8f9df87e87cf1e9f9927dc3bc31. [Accessed: 03-Mar-2023].

[37] "Template matching," *OpenCV*. [Online]. Available: https://docs.opencv.org/4.5.2/d4/dc6/tutorial_py_template_matching.html. [Accessed: 03-Mar-2023].
[38] VictorLegrosVictorLegros 37011 gold badge44 silver badges1313 bronze badges and alkasmalkasm 21.5k44 gold badges7777 silver badges9494 bronze badges, "Understanding and evaluating template matching methods," *Stack Overflow*, 01-Aug-1966. [Online]. Available: https://stackoverflow.com/questions/58158129/understanding-and-evaluating-template-matching-methods. [Accessed: 03-Mar-2023].
[39] "Road Lane Line Detection using Computer Vision models - KDnuggets." https://www.kdnuggets.com/2017/07/road-lane-line-detection-using-computer-vision-models.html (accessed Oct. 31, 2021).
[40] C. Kanan and G. W. Cottrell, "Color-to-Grayscale: Does the Method Matter in Image Recognition?," *PLoS ONE*, vol. 7, no. 1, Jan. 2012, doi: 10.1371/journal.pone.0029740.
[41] "Spatial Filters - Gaussian Smoothing." https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm (accessed Oct. 31, 2021).
[42] M. Wang, S. Zheng, X. Li, and X. Qin, "A new image denoising method based on Gaussian filter," *2014 International Conference on Information Science, Electronics and Electrical Engineering*, 2014.
[43] E. Akbari Sekehravani, E. Babulak, and M. Masoodi, "Implementing canny edge detection algorithm for Noisy Image," *Bulletin of Electrical Engineering and Informatics*, vol. 9, no. 4, pp. 1404–1410, 2020.
[44] C. K.S., S. G.S., N. K.R., and C. A.L., "Analysis of image quality using Sobel filter," *2019 Third International Conference on Inventive Systems and Control (ICISC)*, 2019.
[45] "Machine Vision - University of South Florida." [Online]. Available: https://cse.usf.edu/~r1k/MachineVisionBook/MachineVision.pdf. [Accessed: 03-Mar-2023].
[46] "Lines Detection with Hough Transform | by Socret Lee | Towards Data Science." https://towardsdatascience.com/lines-detection-with-hough-transform-84020b3b1549 (accessed Oct. 31, 2021).
[47] Dagao Duan, Meng Xie, Qian Mo, Zhongming Han, and Yueliang Wan, "An improved Hough Transform for line detection," *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, 2010.
[48] P. S. Rahmdel, R. Comley, D. Shi, and S. McElduff, "A review of Hough Transform and line segment detection approaches," *Proceedings of the 10th International Conference on Computer Vision Theory and Applications*, 2015.
[49] F. Zeng, X. Cai, and S. S. Ge, "Low-Shot Wall Defect Detection for Autonomous Decoration Robots Using Deep Reinforcement Learning," *Journal of Robotics*, vol. 2020, pp. 1–7, Sep. 2020, doi: 10.1155/2020/8866406.
[50] X. Chen, J. Lv, Y. Fang, and S. Du, "Online Detection of Surface Defects Based on Improved YOLOV3," *Sensors*, vol. 22, no. 3, p. 817, Jan. 2022, doi: 10.3390/s22030817.
[51] X. Xie, "Transmission line surface defect detection method based on uav autonomous inspection," *Journal of Physics: Conference Series*, vol. 2132, no. 1, p. 012030, Dec. 2021, doi: 10.1088/1742-6596/2132/1/012030.
[52] I. J. Goodfellow *et al.*, "Generative Adversarial Networks," Jun. 2014.
[53] S.-W. Park, J.-S. Ko, J.-H. Huh, and J.-C. Kim, "Review on Generative Adversarial Networks: Focusing on Computer Vision and Its Applications," *Electronics (Basel)*, vol. 10, no. 10, p. 1216, May 2021, doi: 10.3390/electronics10101216.

[54] A. Dash, J. Ye, and G. Wang, “A review of Generative Adversarial Networks (GANs) and its applications in a wide variety of disciplines -- From Medical to Remote Sensing,” Oct. 2021.
[55] J. Gui, Z. Sun, Y. Wen, D. Tao, and J. Ye, “A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications,” Jan. 2020.
[56] A. Radford, L. Metz, and S. Chintala, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks,” Nov. 2015.
[57] N. Inkawhich, “DCGAN TUTORIAL.” https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html (accessed Mar. 10, 2022).
[58] Q. Liu, M. Wang, Z. Liu, B. Su, and N. Hanajima, “Defect Detection of Micro-Precision Glass Insulated Terminals,” *Journal of Robotics, Networking and Artificial Life*, vol. 8, no. 1, p. 18, 2021, doi: 10.2991/jrnal.k.210521.005.
[59] S. Lin, Z. He, and L. Sun, “Defect Enhancement Generative Adversarial Network for Enlarging Data Set of Microcrack Defect,” *IEEE Access*, vol. 7, pp. 148413–148423, 2019, doi: 10.1109/ACCESS.2019.2946062.
[60] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-Image Translation with Conditional Adversarial Networks,” Nov. 2016.
[61] J. Brownlee, “How to Develop a Pix2Pix GAN for Image-to-Image Translation,” Aug. 02, 2019. https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/ (accessed Mar. 10, 2022).
[62] S. Mertes, A. Margraf, C. Kommer, S. Geinitz, and E. André, “Data Augmentation for Semantic Segmentation in the Context of Carbon Fiber Defect Detection using Adversarial Learning,” in *Proceedings of the 1st International Conference on Deep Learning Theory and Applications*, 2020, pp. 59–67. doi: 10.5220/0009823500590067.
[63] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks,” Mar. 2017.
[64] S. Niu, B. Li, X. Wang, and H. Lin, “Defect Image Sample Generation With GAN for Improving Defect Recognition,” *IEEE Transactions on Automation Science and Engineering*, pp. 1–12, 2020, doi: 10.1109/TASE.2020.2967415.
[65] D.-M. Tsai, S.-K. S. Fan, and Y.-H. Chou, “Auto-Annotated Deep Segmentation for Surface Defect Detection,” *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–10, 2021, doi: 10.1109/TIM.2021.3087826.
[66] T. Karras, S. Laine, and T. Aila, “A Style-Based Generator Architecture for Generative Adversarial Networks,” Dec. 2018.
[67] J. Brownlee, “A gentle introduction to stylegan the style generative Adversarial Network,” May 10, 2020. https://machinelearningmastery.com/introduction-to-style-generative-adversarial-network-StyleGAN/ (accessed Mar. 10, 2022).
[68] “Papers with code - The methods corpus.” https://paperswithcode.com/method/StyleGAN (accessed Mar. 10, 2022).
[69] Z. Situ, S. Teng, H. Liu, J. Luo, and Q. Zhou, “Automated Sewer Defects Detection Using Style-Based Generative Adversarial Networks and Fine-Tuned Well-Known CNN Classifier,” *IEEE Access*, vol. 9, pp. 59498–59507, 2021, doi: 10.1109/ACCESS.2021.3073915.
[70] F. A. Saiz, G. Alfaro, I. Barandiaran, and M. Graña, “Generative Adversarial Networks to Improve the Robustness of Visual Defect Segmentation by Semantic Networks in Manufacturing Components,” *Applied Sciences*, vol. 11, no. 14, p. 6368, Jul. 2021, doi: 10.3390/app11146368.

[71] S. Zhao, Z. Liu, J. Lin, J.-Y. Zhu, and S. Han, “Differentiable Augmentation for Data-Efficient GAN Training,” Jun. 2020.
[72] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” May 2015.
[73] A. Zafar, M. Aamir, N. Mohd Nawi, A. Arshad, S. Riaz, A. Alruban, A. K. Dutta, and S. Almotairi, “A comparison of pooling methods for Convolutional Neural Networks,” *Applied Sciences*, vol. 12, no. 17, p. 8643, 2022.
[74] A. F. Agarap, “Deep Learning using Rectified Linear Units (ReLU),” Mar. 2018.
[75] J. Zhang, “UNet line by line explanation,” *Towards Data Science*, Oct. 18, 2019. https://towardsdatascience.com/unet-line-by-line-explanation-9b191c76baf5 (accessed Jul. 16, 2022).
[76] A. Gupta, “Optimizers in Deep learning: A comprehensive guide,” *Analytics Vidhya*, 03-Mar-2023. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/. [Accessed: 03-Mar-2023].
[77] “Adam,” *Adam - Cornell University Computational Optimization Open Textbook - Optimization Wiki*. [Online]. Available: https://optimization.cbe.cornell.edu/index.php?title=Adam. [Accessed: 03-Mar-2023].
[78] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” Dec. 2014.
[79] B. Xu, N. Wang, T. Chen, and M. Li, “Empirical Evaluation of Rectified Activations in Convolutional Network,” May 2015.
[80] J. Qi, J. Du, S. M. Siniscalchi, X. Ma, and C.-H. Lee, “On Mean Absolute Error for Deep Neural Network Based Vector-to-Vector Regression,” Aug. 2020, doi: 10.1109/LSP.2020.3016837.
[81] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, “Activation Functions: Comparison of trends in Practice and Research for Deep Learning,” Nov. 2018.
[82] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” Sep. 2014.
[83] I. Kouretas and V. Paliouras, “Simplified hardware implementation of the Softmax activation function,” *2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, 2019.
[84] J. Brownlee, “Softmax activation function with python,” *MachineLearningMastery.com*, 23-Jun-2020. [Online]. Available: https://machinelearningmastery.com/softmax-activation-function-with-python. [Accessed: 03-Mar-2023].
[85] L. Du, R. Zhang, and X. Wang, “Overview of two-stage object detection algorithms,” *J Phys Conf Ser*, vol. 1544, no. 1, p. 012033, May 2020, doi: 10.1088/1742-6596/1544/1/012033.
[86] S. Srivastava, A. V. Divekar, C. Anilkumar, I. Naik, V. Kulkarni, and V. Pattabiraman, “Comparative analysis of deep learning image detection algorithms,” *J Big Data*, vol. 8, no. 1, p. 66, Dec. 2021, doi: 10.1186/s40537-021-00434-w.
[87] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” Jun. 2015.
[88] T. Diwan, G. Anirudh, and J. V. Tembhurne, “Object detection using yolo: Challenges, architectural successors, datasets and applications,” *Multimedia Tools and Applications*, vol. 82, no. 6, pp. 9243–9275, 2022.
[89] P. Shi, M. Duan, L. Yang, W. Feng, L. Ding, and L. Jiang, “An improved U-Net Image segmentation method and its application for metallic grain size statistics,” *Materials*, vol. 15, no. 13, p. 4417, 2022.

[90] D. M. W. Powers, “Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation,” Oct. 2020.
[91] B. Wang, “A Parallel Implementation of Computing Mean Average Precision,” Jun. 2022.

Appendix A

Image Tracking

```
import math
import statistics
import cv2
import numpy as np
import matplotlib.pyplot as plt
# video=cv2.VideoCapture("Along_Rail.MP4")
# while True:
#     ret,frame=video.read()
#     if not ret:
#         video=cv2.VideoCapture("")
#         continue
#
#     cv2.imshow('frame',frame)
#
#     key=cv2.waitKey(25)
#     if key==27:
#         break
# video.release()
# cv2.destroyAllWindows()
image_original = cv2.imread('DJI_0010.JPG')
print(image_original.shape[0:2])
height,width=image_original.shape[0:2]
#resize the picture
image_resized=cv2.resize(image_original,(0,0),fx=0.25,fy=0.25)

#convert to greyscale
image_gray = cv2.cvtColor(image_resized, cv2.COLOR_BGR2GRAY)
cv2.imwrite('greyscale.jpg', image_gray)
#image2=cv2.imshow('Gray',image_gray)
#cv2.waitKey(0)
#apply Guarssian filter
img_blur = cv2.GaussianBlur(image_gray,(3,3), cv2.BORDER_DEFAULT)
cv2.imwrite('blurred.jpg', img_blur)
#image2=cv2.imshow('blurred',img_blur)
#cv2.waitKey(0)
sobelx2 = cv2.Sobel(img_blur, cv2.CV_16S, 1, 1, ksize=3)
abs_sobel64f = np.absolute(sobelx2)
sobel_x = np.uint8(abs_sobel64f)

kerSize = 20
kernel = np.ones(shape=(kerSize, kerSize)) * 0.1
img_sobel_filtered = cv2.filter2D(src=sobel_x, ddepth=-1, kernel=kernel)

img_binary = cv2.threshold(img_sobel_filtered, 200, 255,
cv2.THRESH_BINARY_INV)[1]

# cv2.imshow('flt',img_sobel_filtered)
cv2.imshow('BIN',img_binary)
cv2.imwrite('filtered.jpg', img_binary)
```

```
#plt.plot(img_sobel_filtered[1,:])

#plt.show()
#image4=cv2.imshow("sobel",sobel_x)
#Double Threshold
filtered_image = cv2.Canny(sobel_x, 90,100)
Lines=cv2.HoughLinesP(img_binary,1,math.pi/180,300,10,300)
print(Lines)
x1 = np.zeros(Lines.shape[0])
y1 = np.zeros(Lines.shape[0])
x2 = np.zeros(Lines.shape[1])
y2 = np.zeros(Lines.shape[1])
for i,line in enumerate (Lines):
    x1[i],y1[i],x2[i],y2[i]=line[0]
    #cv2.line(image_resized,(x1,y1),(x2,y2),(0,255,0),3)

x3 = x1.tolist()+x2.tolist()

'''
x3.sort()
x_11 = int((x3))
for i in range(len(x3)-1):
    if (x3[i+1]-x3[i]) > 100:
        x_12 = int(x3[i])
        x_21 = int(x3[i+1])
        break
x_22 = int(max(x3))
'''

x3.sort()
x_mean=int(statistics.mean(x3))
for i in range(len(x3)-1):
    if (x3[i]>x_mean):
        x_11=int(x3[0])
        x_12=int(x3[i-1])
        x_21=int(x3[i])
        x_22=int(x3[-1])
        break

cv2.line(image_resized,(x_11,900),(x_12,0),(0,255,0),3)
cv2.line(image_resized,(x_21,900),(x_22,0),(0,255,0),3)
# Plot output
image1= cv2.imshow('filtered image',filtered_image)
image7=cv2.imshow('line',image_resized)
cv2.waitKey(0)
```

Appendix B

Video Tracking

```
import math
import statistics
from PIL import Image, ImageDraw
import cv2
import numpy as np
import matplotlib.pyplot as plt

#import video
video=cv2.VideoCapture("Along_Rail.mp4")
while True:
    ret,frame=video.read()
    if not ret:
        video=cv2.VideoCapture("Along_Rail.mp4")
        continue
#resize the picture
    img_resized=cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)
    height=img_resized.shape[0]
    width=img_resized.shape[1]
#convert to greyscale
    image_gray = cv2.cvtColor(img_resized, cv2.COLOR_BGR2GRAY)
#apply Guarssian filter
    img_blur = cv2.GaussianBlur(image_gray,(7,7), 7)
#sobel filter
    #sobelx2 = cv2.Sobel(img_blur, cv2.CV_16S, 0, 1, ksize=3)
    sobelx2 = cv2.Sobel(img_blur, cv2.CV_16S, 1, 0, ksize=3)
    cv2.imshow("haha", img_blur)
    abs_sobel64f = np.absolute(sobelx2)
    sobel_x = np.uint8(abs_sobel64f)
    kerSize = 5
    kernel = np.ones(shape=(kerSize, kerSize)) * 0.2
    #cv2.imshow('',sobel_x)
    img_sobel_filtered = cv2.filter2D(src=sobel_x, ddepth=-1,
kernel=kernel)
    #cv2.imshow('',img_sobel_filtered)
# Double Threshold
    img_binary = cv2.threshold(img_sobel_filtered, 70, 130,
cv2.THRESH_BINARY_INV)[1]
#find lines

Lines=cv2.HoughLinesP(img_binary,1,math.pi/180,200,minLineLength=400,maxLi
neGap=4)
    x1 = np.zeros(Lines.shape[0])
    y1 = np.zeros(Lines.shape[0])
    x2 = np.zeros(Lines.shape[0])
    y2 = np.zeros(Lines.shape[0])
    for i,line in enumerate (Lines):
        x1[i],y1[i],x2[i],y2[i]=line[0]
```

```
    x3 = x1.tolist()+x2.tolist()
    x3.sort()
    x_mean=int(statistics.mean(x3))
    w=width/2
    if x_mean<w:
        distance=w-x_mean
        T="Should move{} pixel to the left".format(distance)
        cv2.putText(img_resized, T,
                    (100,100),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    1,
                    (250,0,0),
                    2)
    elif x_mean>w:
        distance=x_mean-w
        T="Should move {} pixels to the right".format(distance)
        cv2.putText(img_resized, T,
                    (100, 100),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    1,
                    (250, 0, 0),
                    2)
    #cv2.imshow("haha", img_resized)
    '''for i in range(len(x3)-1):
        if (x3[i] > x_mean):
            x_11=int(x3[0])
            x_12=int(x3[i-1])
            x_21=int(x3[i])
            x_22=int(x3[-1])
            break
    '''
    cv2.line(img_resized, (x_mean, 900), (x_mean, 0), (0, 255, 0), 7)
    #cv2.line(img_binary, (x_mean, 540), (x_mean, 0), (0, 255, 0), 7)
    #cv2.line(img_resized, (x_21, 900), (x_22, 0), (0, 255, 0), 3)
    sel = 1

    if sel == 1:
        trg = img_resized
    elif sel == 2:
        trg = img_binary
    elif sel == 3:
        trg = img_sobel_filtered
    elif sel ==4:
        trg = sobel_x

    cv2.imshow("haha", trg)

    key=cv2.waitKey(25)
    if key==27:
        break
video.release()
cv2.destroyAllWindows()
```