CRANFIELD UNIVERSITY

PANAGIOTIS TSILIVIS

AUTONOMOUS INDOOR INSPECTION OF AIRCRAFT WING PANEL WITH UAV: IMAGE PROCESSING & LOCALISATION

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING
Autonomous Vehicle Dynamics & Control

MSc
Academic Year: 2016–2017

Supervisor: Prof. Antonios Tsourdos & Dr. Al Savvaris & Mr. Jose Gonzalez
August 2017

CRANFIELD UNIVERSITY

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING
Autonomous Vehicle Dynamics & Control

MSc

Academic Year: 2016–2017

PANAGIOTIS TSILIVIS

Autonomous Indoor Inspection of Aircraft Wing Panel with UAV: Image Processing & Localisation

Supervisor: Prof. Antonios Tsourdos & Dr. Al Savvaris & Mr. Jose Gonzalez
August 2017

This thesis is submitted in partial fulfilment of the requirements for the degree of MSc.

Abstract

Abstract

This particular project, entitled ”Autonomous Indoor Inspection of Aircraft Wing Panel with UAV: Image Processing & Localisation”, constitutes a Research and Development application concerning an algorithmic analysis and implementation of image processing and precise localisation methods; which enable the drone to perform efficiently the operation of indoor autonomous inspection in narrow environment. In particular, the application of this thesis is focussed more on the precise positioning of the drone around a wing panel and the detection of potential flaws with the use of Ultraviolet light. The application is to guide precisely the drone around a wing panel, which is in the process stage of a Non-Destructive Testing, and inspect the panel for potential defections. Following that, a post-processing automatic detection and classification of the wing panel's defects is performed with advanced image-processing techniques. The experimental application was carried out at Cranfield University, in cooperation with the Airbus Group, as part of Autonomous Vehicle Dynamics and Control (AVDC) MSc Course during the academic year 2016 - 2017.

Keywords: UAV, Inspection, Non-Destructive Testing, Precise Positioning, Drones, Aircraft Wing Panel, Image Processing, Ultraviolet light.

Contents

List of Figures

List of Tables

List of Abbreviations

SATM	School of Aerospace, Technology and Manufacturing
SEEA	School of Energy, Environment and Agrifoods
UAV	Unmanned Aerial Vehicle
MAV	Micro Aerial Vehicle
RTLS	Real-Time Location System
GPS	Global Positioning System
PFD	Penetrant Flow Detection
NDT	Non-Destructive Testing
UV	Ultraviolet
LED	Light Emitting Diode
USB	Universal Serial Bus
ToF	Time of Flight
ToA	Time of Arrival
AoA	Angle of Arrival
TDoA	Time Difference of Arrival
RSS	Received Signal Strength
UWB	Ultra Wideband
INS	Inertial Navigation System
SLAM	Simultaneous Localisation and Mapping
RFID	Radio Frequency Identification
LIDAR	Laser Identification Detection and Ranging

MIC	Morphological Image Cleaning
PDF	Probabilistic Density Function
HOG	Histogram of Oriented Gradients
SIFT	Scale-Invariant Feature Transform
SURF	Speeded-Up Robust Features
ICP	Iterative Closest Point

Acknowledgements

I would first like to thank my thesis supervisors Prof. Antonios Tsourdos and Dr. Al Savvaris of Cranfield University for their support. The office doors both of the supervisors were always open whenever I needed some further advise. I would also like to thank the Airbus supervisor Jose Gonzalez for his continuous support and excellent cooperation. Finally, I must express my very profound gratitude to my parents and friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Chapter 1

Introduction

With the advent and the sharp advance of the modern technology, the Unmanned Aerial Vehicles (UAVs) have taken a significant position in regard to several applications; either military or industrial, civil. Apart from advanced military applications, civil operations constitute an also important field of research. Applications such as industrial surveillance, precise agriculture, fire-fighting or remote sensing and monitoring, are only the most common operations performed by various UAV types [12]. However, the requirements, along with the difficulties of these operations, are of utmost important to be researched. In particular, the operating environment of the UAV poses several challenges differing from operation to operation. The unknown environment of an outdoor area constitutes a challenge that needs to be coped with continuous update of the UAV's path. Collision avoidance algorithms are also a part of the aforementioned unknown environment challenge, that have to be contained in the UAV's system. For indoor environments, the GPS data are unavailable; a challenge that needs be to dealt with, in order to achieve optimum control of the position of the drone. Options such as laser sensors or indoor Real-Time Localisation System provide the drone's system with valuable data that assist in the localisation.

One of the most essential requirements concerning the manufacturing of an aircraft, is the built quality of each of the equipped parts. With the view of achieving the optimum safety quality in regard to an aircraft build, several tests specialised on every different part of the aircraft,

Figure 1.1: Drone used by EasyJet for Aircraft Damage Inspection [1]

are performed during the manufacturing process. Namely, one the important processes during the manufacturing of an aircraft wing, is called Penetrant Flow Detection (PFD). Shortly, the PFD provides assurance that the wing panel is defection free prior to the step of the anodising and the painting. This work, focusses on the research and development of an innovative method that replaces the ongoing technique with an autonomous operation with the use of a UAV. Specifically, in this thesis the subject to be researched and analysed is oriented on the precise indoor positioning of a drone and the autonomous navigation of the UAV around an Airbus wing panel; with the view of inspecting it to detect potential defections. The benefits that this application provides are important for the advance of the manufacturing. In particular, it significantly reduces the amount of time needed for inspecting the wing panel by a technician and also increases the accuracy of the inspection, while it provides a 3-D model for post-processing evaluation.

This section states the aims and objectives of this individual research work, and introduces the literature review regarding prior research of suitable algorithms and hardware implementations.

1.1 Aims and Objectives

The aim of this individual research project is oriented around the research of potential algorithms that enable indoor navigation and mapping for inspecting an under-construction aircraft wing; the essential feature of focus, is the precision in positioning. In other words, the target of this research is to develop a drone application, in regard to the hardware configuration and software implementation; with the view of inspecting an aircraft wing panel using the method of Penetrant Flaw Detection (PFD), to detect potential flaws before the panel's painting and anodising process. The aim of the literature review, which follows in the next section, is to state suitable researched techniques concerning precise positioning in indoor environments, computer vision methods regarding feature detection and extraction under UV light, and potential path-planning methods with the view of developing an efficient way-point navigation. The objectives of this research project are listed as follows:

- Evaluate the potential risks of the entire process, focussing on the safety of the wing, and the demands of environment, with respect to the hardware configuration of the used UAVs.
- Assess the possible difficulties that need to be coped with, concerning the indoor positioning, visual structure detection, among others stated in the following sections.
- Simulate the Penetrant Flow Detection process in laboratory environment, in order to perform experiments with the UAV.
- Study and implement suitable algorithms, concerning high accuracy in localisation and computer vision and image recognition techniques, focussed on the feature detection and extraction for identifying and mapping the environment of the UAV.
- Simulate the autonomous navigation application in 3-D software environment.
- Perform a flight demonstration with the use of the DJI Matrice 100 UAV, with the view of a complete autonomous inspection of an Airbus wing panel.

- Discuss the feasibility of the project with respect to the resulting UAV performance in tight indoor environment along with the high risk assessment of the mission.

Chapter 2

Literature Review

In this chapter, the software solutions concerning the computer vision and image analysis for inspection of structures, such as an aircraft wing panel, under UV light are to be described. Additionally, localisation techniques and hardware options, based on prior researches, are to be stated and explained; focussing more on GPS-denied methods.

2.1 Localisation - Theoretical Implementations & Sensors

This sub-section concerns the description of indoor localisation algorithms along with relevant used sensors. Specifically, suitable sensors for indoor navigation are bound to be illustrated, while positioning algorithms are to be described.

The structures of the UAVs generally vary on each other, in several aspects. However, when the drone has to perform an indoor operation, the requirements to meet are of utmost importance and very strict. The UAV has to feature lightweight characteristics with the view of advanced manoeuvrability; the lightweight feature along with powerful motors and sharp processor, constitutes an optimal combination for a UAV precise controllability. However, depending on the operation that the UAV has to complete, the sensors and techniques, regarding the positioning of the drone, changing on focus. In this particular application, in which the drone has to perform an indoor inspection through a narrow environment, namely the wing panel inspection room, the most striking requirement consists the accurate GPS-denied localisation of the drone. Several researches have been carried out, in order to conclude on optimum solutions; various papers contain solutions or, in other words, localisation implementations of algorithms along with suitable indoor positioning sensors.

Apart from the fact that an indoor inspection demands high levels of manoeuvrability, hence, lightweight structures with optimum sizes, requirements such as sensory precision or accuracy in navigation are of utmost importance. To achieve this general requirements, the combination of sensors and navigation algorithms must collaborate to the optimum. From the view of the hardware of a drone, the requirement that must be met is the accuracy of the sensors. The bias created by the sensors constitutes a significant factor that may affect the performance of the drone; which also contains the risk of collision. Additionally, the indoor environment is a GPS-denied place that leads the approach of the application towards other localisation solutions.

2.1.1 Vision-Based Positioning - Monocular Camera

Vision-based localisation is one of the most common approaches in indoor and outdoor environments. At present, due to the quality of image recorded by modern cameras, several applications have been relied on vision-based positioning. It is worth-mentioning that apart from numerous benefits that this technique provides, the most essential drawback that has to be coped with, is the low levels of indoor lighting which affects the quality of the results. Fortunately, various sub-solutions concerning the issue of lighting in machine vision, have been proposed in several academic papers. For instance, an installation of LEDs close to the drone's camera, constitutes a constant light projection that the drone can process via image-processing algorithms. An illustration of similar techniques is presented by Daryl Martin [13], while a deeper look into this issue is described in the Chapter 3 of the *Handbook of Machine Vision* [14]. Apart from a hardware modification, image processing techniques, such as the simple gamma correction filter, can improve the visual factor of the image or video and enable the processing to better results. A camera-based positioning technique has been proposed and implemented to a similar to this application, concerning *Inspection of Industrial Facilities* [7]. In the aforementioned application, the UAV performs inspections in enclosed industrial environments with the use of monocular cameras. Specifically, the drone uses a pair of monocular cameras arranged in a classical stereo configuration, which provide the system with essential feature-based information concerning the localisation and mapping process. It is worth noting that the cameras are equipped with exposure synchronous LED flash, to eliminate the factor of low level lightning.

2.1.1.1 Monocular Camera along with Laser Sensor

A combination of visual-based navigation and laser range finder is also a possible solution for indoor navigation. This particular solution is described in the paper, entitled *Autonomous Indoor Hovering with a Quadrotor* et al. [G. Angeletti] [15], which illustrates a performance of a monocular camera, as the vision system, and a laser range finder sensor with the view of detecting the surrounding objects. Similar to this project's technique, is described in the thesis report, entitled *Autonomous Flight in Unstructured and Unknown Indoor Environments* et al. [A. Bachrach] [16]. In particular, the thesis follows the methods of a standalone Laser configuration, a pair of monochrome USB cameras, as well as the combination of the two options. The optimum solution, is given by the combination of the two sensors due to the fact that in overcomes standalone issues. Problem such as shadowing or illumination of the camera is coped with the laser beacon, while issues such as incorrect measurements from inaccurate laser beams are dealt with vision-based featured extraction and description methods. Nevertheless, a comparison of both sensors depicts that laser sensor maintains essential advantages in the accuracy of the measurements along with the corresponding propagation.

Sensor - Approach	Computation Time	Bandwidth
Laser Sensor (Scan-Matching)	5 msec/measurement	170 KB/sec
Monocular Camera (Visual Odometry)	65 msec/measurement	1300 KB/sec

Table 2.1: Comparison between laser and vision requirements and bandwidth

The table above, estimated in the thesis report [17], presents the transmission time of both sensors. The laser data requires less propagation time in contrast to the camera. This, results in reduced time of the scan-matching algorithm and the process of incoming data; which leads to less delay in the position estimates. This delay is of utmost importance for the stability of the drone. However, the visual odometry provides a critical benefit, on contrary to the laser scan-matching, of estimating all 6 degrees of freedom.

2.1.2 Real Time Location System - RTLS

Positioning accuracy to an indoor environment can be achieved by a combination of multiple sensors mounted in the perimeter of the environment. In other words, the Real-Time Location System (RTLS) uses technology of Radio Frequency (RF), or even optical (infrared) or acoustic (ultrasound) media, in order to communicate with every sensor and, hence, create a local positioning system. The principle of operation of the RTLS, states that several anchor sensors are mounted to the corners of the indoor environment, while a tag sensor is equipped on-board the desired vehicle that needs to be positioned. The tag sensor, which is mounted to the object-to-be-located, radiates signals, called beacons, towards the indoor environment. These beacons are received by the wall-mounted reader sensors, calculating the relative distance between the anchor and the tag [18]. Furthermore, each of these readers (called anchors) sends the distance information to the main processor of the system, in order to enable it calculate the relative coordinates of the tag based on the anchors. However, with respect to the variety of the manufacturers, every RTLS hardware differs from each other; the estimation of the relative coordinates may be performed from the tag to the anchor, in terms of the position of the processor. The figure 5.1 illustrates the KIO RTLS, manufactured by Eliko Tehnoloogia [2], which contains 4 anchors and 1 tag. The tag is connect via serial USB cable to the ground processor of the system, while it sends all the data for the estimation of the relative coordinate system. This particular system of sensors is being described in detail in the next chapter.

The technique used by KIO sensor for estimating the relative distance of the tag, is called Time-of-Flight (ToF) or Time-of-Arrival (ToA). The operating principle, states that the relative distance is estimated through the travel time of the radio signal between the receiver and transmitter sensors. Similarly, other possible distance measurement techniques are **1)** the Angle of Arrival (AoA), **2)** the Time Difference of Arrival (TDoA) and **3)** the Received Signal Strength (RSS) [19]. These methods are divided in two main categories, namely one-way and two-way ranging methods [20].

Figure 2.1: Real-Time Location System - KIO Evaluation Kit [2]

The one-way ranging method is a simple technique in terms of implementation, however, the provided accuracy fluctuates at low levels. In particular, this category of ranging estimation uses the strength of the received signal with the view of estimating the distance between the receiver and transmitter. The Received Signal Strength method (RSS) belongs to this category of ranging methods. On the other hand, the two-way ranging method uses the radio technology in order to estimate the delay of two transmitted signals. By way of explanation, theses techniques use the delay that naturally occur during a signal broadcast in order to determine the range between the two stations; namely, the reader (anchor) and the tag. The calculation of the range is performed by the propagation delay; focussed on the acknowledgements of each station, the calculation algorithm takes as inputs the acknowledgement of the returned signal, which contains the signal propagation delay and the processing delay, and use these parameters to estimate the range. The final stage of these algorithms, after the range estimation, is the transmission of a verification signal which contains the range of these station along with an average value. A characteristic algorithm that uses a two-way ranging estimation is the Time Difference of Arrival (TDoA). In particular, it estimates the time difference between two nodes, while it uses this information as input to calculate the location of the sensor. At first, a time-synchronised broadcast towards the readers provides the tag with the information of the ranging. Then, the processor of the system calculates the position of the tag, with respect to the converted received distance between the readers and the tag, using the hyperbola curve. As follows, a simple illustration of both techniques provides a describing image of them [3].

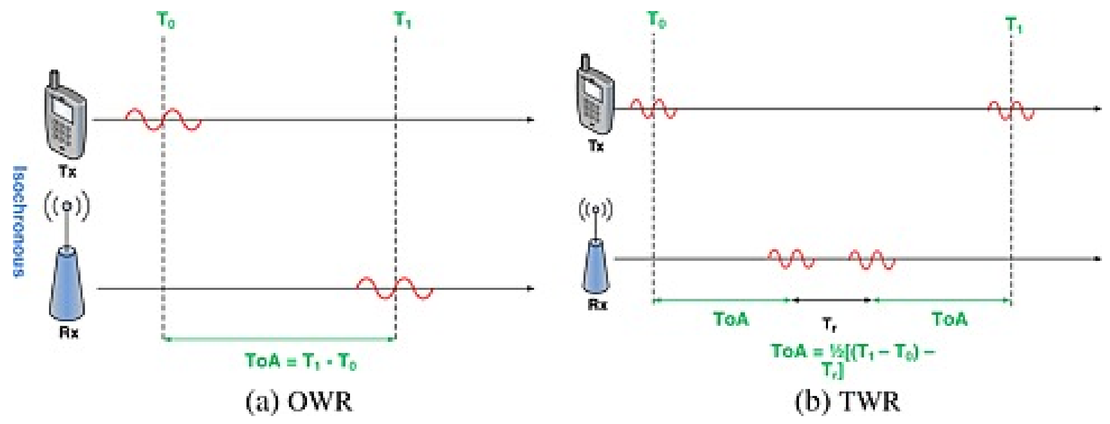


Figure 2.2: Operating Principle of One-Way and Two-Way Ranging Methods [3]

2.1.3 Ultra-Wideband Positioning

The Ultra-Wideband is a radio technology that provides a system with signal propagation, as similarly mentioned in the previous sub-section 2.1.2. However, the robustness of the signal communication among the sensors along with the improved accuracy of the distance estimations, renders this radio technology as worth-describing.

As stated in the thesis [Rainer Mautz] [21], the significant feature that makes the Ultra-Wideband (UWB) stand out from the other radio frequency technologies, is the large bandwidth that maintains; which is translated into an essentially improved resolution in time, and as a result, in range. This range can be estimated by the ratio of the speed of the wave front (v) related to the bandwidth (b).

$$ratio = \frac{v}{2b} \tag{2.1}$$

Furthermore, the UWB radio technology includes the techniques of two-way ranging, described above, such as the Time of Arrival (ToA) and the Time Difference of Arrival (TDoA). These techniques rely on time measurements; thus, the accuracy of the measurements in every mission maintains significant factor for the integrity of the results. Therefore, the following three categories of frequency bands specify the different uses of this radio technology. In particular, the **Continuous Waves** enable the precision on the ranging measurements, still, this technology demands large antennas for the broadcast of the signal. This requirement is translated to an incapability of using this technology to small devices; the wider the frequency range, the larger the physical size of the antenna. On contrary, the **UWB Impulse Radio** technology provides even more robustness of the signal transmission and fast distance measurements. The UWB-IR uses ultra-short pulses (duration of nanoseconds) in order to transmit the data, which provides better overall resolution of the information; due to the fact that it is protected against the interference of the signal from multipath broadcast. One more advantage of this technology, is the low power consumption due to less energy demand. An implementation regarding a localisation of an industrial robot is described by Segura et al. [22]. Last but not least, the **Pseudo Noise Modulation** technology is based on Maximum Length Sequences, or M-sequences, which are random binary sequences with short duration. The advantage of this technology is the small sizes of antennas that they use; an appealing feature for the mobile and Micro Aerial Vehicles (MAVs) companies. However, the large processing demand for the estimation of the ranging, consists an essential drawback that needs to be improved.

Concerning the localisation implementations using the UWB technology, the methods are divided in **Passive UWB Localisation**, which includes an omnidirectional emitter antenna and multiple listener antennas, **UWB Virtual Anchors**, which contains a physical transmitter and uses the walls as virtual anchors to receive delays of the signals, and **UWB Direct Ranging**, which are based on the techniques of ToA and TDoA. The figure 2.3 illustrates a UWB indoor positioning system, similar to the one chosen for this project implementation.

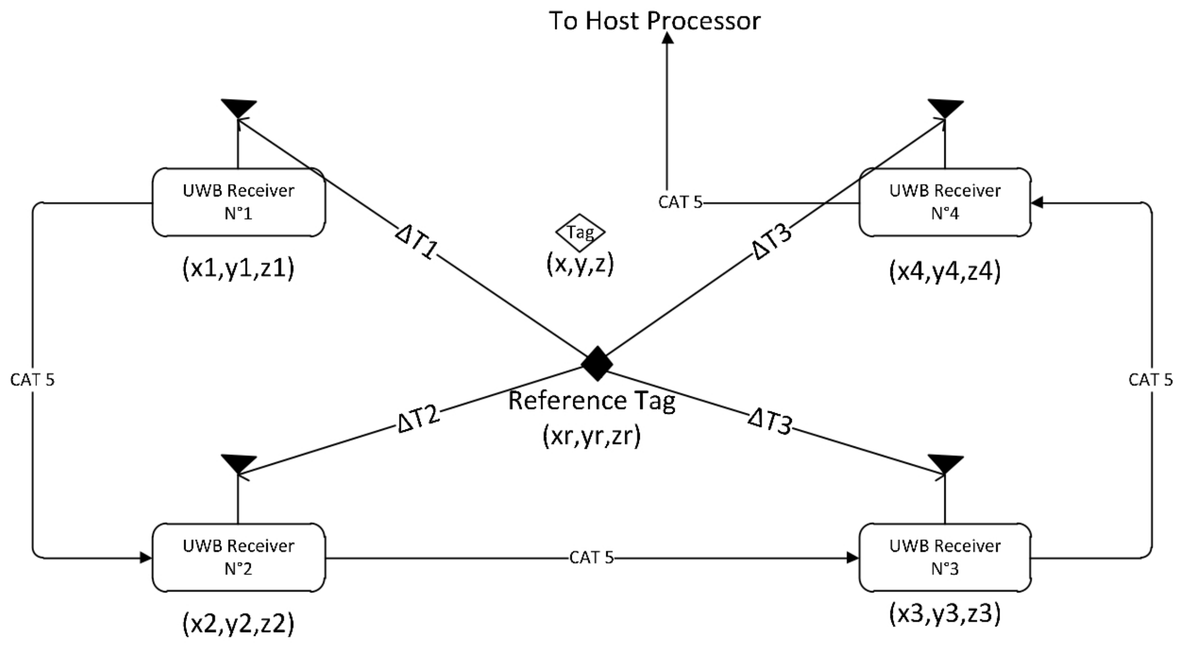


Figure 2.3: UWB Passive Positioning System - 4 Readers and 1 Tag [4]

2.1.4 Supporting Sensors

The sub-section of the supporting sensors is related to the hardware that provides the autonomous vehicle with supporting data, yet sometimes necessary. Sensors such as the Inertial Navigation System (INS), the Ultrasonic sensor or an Altimeter are mandatory sensors that provide the drone with essential information; these are to be stated as follows.

2.1.4.1 Inertial Navigation System (INS)

The Inertial Navigation System is a system every drone contains, due to the simplicity regarding the positioning that it provides. The algorithm of INS uses the data acquired from the Inertial Measurement Unit (IMU) in order to estimate the position of the drone based on the movement itself. Due to the shift of the IMU data, usually the INS fuses these information along with inputs from other complementary sensors with the view of calculating more accurate position. The particular information provided by the INS is an estimation of the drone's position, velocity and orientation towards the operating environment. At this point, it is worth-noting that an initial belief of the drone's state is a requirement of utmost importance for the INS to maintain less bias in its estimations. Following that an initial position is provided, the Dead Reckoning method is being applied in order to calculate the position of the drone based on the velocity and movements that the vehicle performed. This technique does not require external information; a feature that characterises this position estimator as a robust and rapid localisation system. In the article of Bong-Su Cho et al. [23], the INS technique and the Dead Reckoning approach is being used in order to estimate the position of mobile robots, yet with the addition of a Kalman Filter for improving the orientation and velocity estimation.

2.1.4.2 Altitude Estimation Sensor

The solution concerning the vertical localisation of the drone, that is the altitude ranging, constitutes the altimeter sensor in combination with ultrasonic sensor; a dominant option due to the accuracy and robustness that this fusion provides. The aforementioned paper et al. [G.

Angeletti] [15], mentions the technique of sonar-based altitude control, in which the ultrasonic sensor is equipped at the ceiling of the drone in order to avoid potential obstacles that may affect the output of the drone's real height. Additionally, in the publication of G. Szafranski et al. [24], the given solution describes the option of using the combination of an Altimeter along with an Ultrasonic sensor for the estimation of the vehicle's altitude. In particular, the Altimeter includes a barometric pressure sensor, connected to a micro-controller that converts the analogue measurements to digital signal. In addition to this, an ultrasonic proximity finder estimates the altitude of the vehicle between 0 and 7 meters. Specifically, it uses the transmitted sound waves in order to estimate the distance based on the evaluated echo. The main advantage of the proximity sensor, in comparison to the altimeter, is the operation in any environmental conditions; there is no effect atmospheric effect against the sensor. However, the range limitation along with the restricted size of the detected objects, consists an essential drawback that demands careful placement of the sensors on-board the vehicle.

2.1.5 Positioning using Theoretical Approaches

The computer vision algorithms, stated in the next section, conduce to the overall positioning of the drone when combined with the appropriate localisation sensors. In other words, the analysis of the drone's vision constitutes a fundamental step for the extraction of the position data. Hence, the SLAM algorithms along with their image processing methods, are of utmost importance for the localisation of the drone; while the precision of the positioning is based on the accuracy of the methods along with the gathered data. For instance, the application of indoor navigation with the use of a Micro Aerial Vehicle (MAV) et al. [D. Sobers] [5], uses an algorithm called "CoreSLAM" as the basic tool for the precise localisation and mapping. The following figure 2.4 illustrates the resulting map of the above mentioned application, generated in simulation environment with the use of a laser sensor.

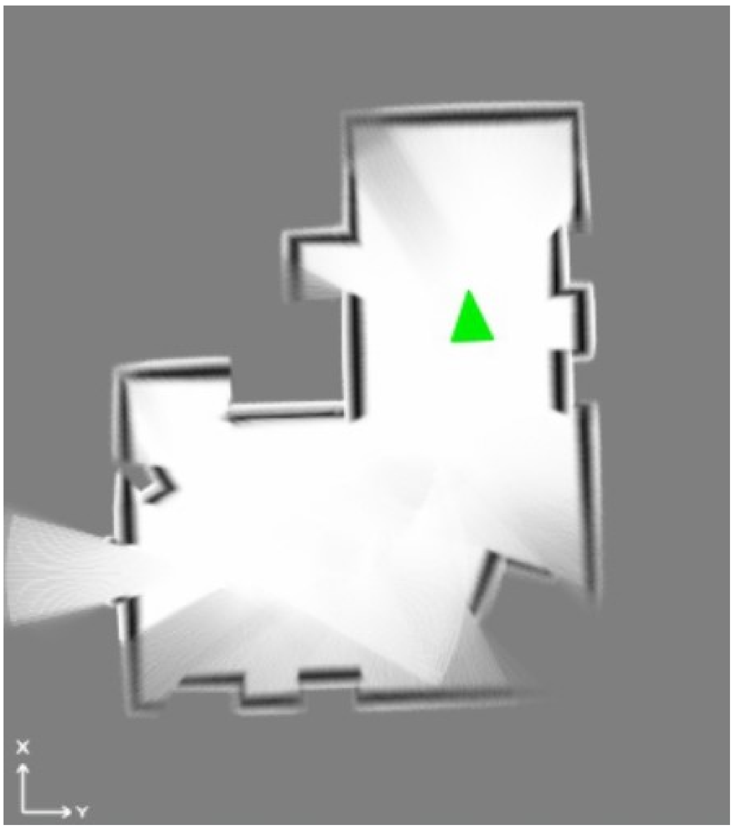


Figure 2.4: Laser-based map in simulation environment generated from CoreSLAM [5]

Following the method of laser or optical sensors, different approaches use wireless indoor communication for the propagation of the position data based on known map. These approaches are categorised into long distance, middle distance and short distance wireless technology. The *long distance* technology contains methods such as FM signal propagation and GSM/CDMA. The *middle distance* wireless technology includes the WiFi technology along with the ZigBee. Last but not least, the *short distance* technology is about signal propagation via Bluetooth, Ultra-Wide Band (UWB) and Radio Frequency Identification (RFID). These technologies are used widely in the UAV indoor navigation, with the view of improving or even optimising the localisation of the drone. The technical report et al. [Junjie Liu] [25], summarises and describes these techniques in detail.

Furthermore, the most common theoretical techniques used for the estimation of the position are the proximity and triangulation or trilateration. In particular, the *proximity* method is based on a known location, while the drone tries to estimate or predict its position in regards to it. For a wireless localisation approach, the UAV can estimate its position based on the position of the access point. Unfortunately, due to its high variance this technique is no longer implemented in UAV developments. On the other hand, the *triangulation* method uses geometric data to obtain the location of the drone. Similarly, the *trilateration* method uses the euclidean distances among several points in order to define the distance from the intersection point; the difference from the triangulation technique is that it does not take involve the measurements of angle. Yet, the position of the drone is estimated either relatively to the distance of a fixed point measurement, or by the received signal angle [25]. Finally, the triangulation method is divided in three sub-categories, namely, time-based triangulation, angle-based and RSS-based.

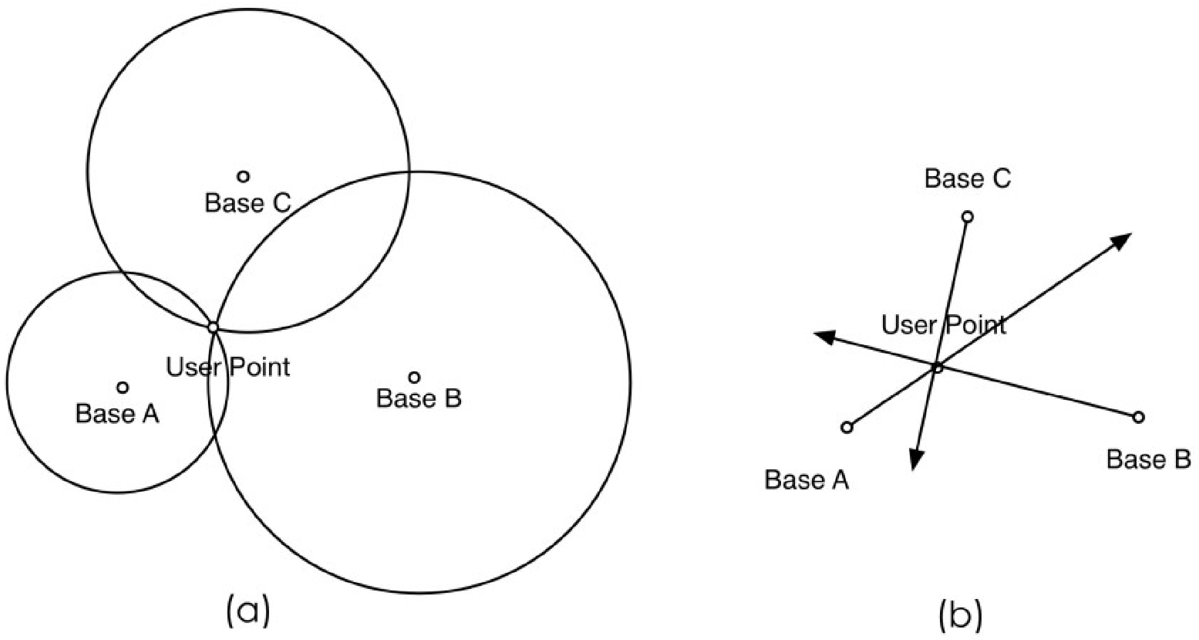


Figure 2.5: Triangulation technique using (a) distance and (b) angle

The combination of several wireless techniques for the improvement of the position accuracy, constitutes an advance and robust approach. In the publication regarding the integration of RFID and WLAN for indoor positioning, et al. [A. Papapostolou] [26], the WiFi is used to guide the reader and elimination any potential RFID issue, while the RFID is used to foresee the following mobile node and increase the performance of the handover latency among the mobile nodes [25].

2.2 Computer Vision - Positioning & Image Analysis

The inspection techniques vary among several non-destructive testing (NDT) methods, such as visual inspection, radar or LIDAR, UV and thermography. Hence, based on the infrastructure to be inspected, the method differs in terms of the equipment, that is to be used, along with the image processing technique [27]. However, the differences among the aforementioned sensors - techniques are essential. In particular, the optical and infrared option provide only 2-D data, while the radar and LIDAR sensors offer the significant parameter of depth information. The absence of depth in the optical and infrared options, assuming that the camera is monocular and not stereo, is coped with the method of Inverse Depth Parametrisation [28]. This method is extensively used in the Monocular Simultaneous Localisation and Mapping (SLAM) algorithms, in which it solves the essential problem of depth and the inaccurate identification of close distance features in the captured image. This section, describes the techniques used in Computer Vision concerning the localisation based on image analysis, in addition to filters and detector algorithms that provide automation in defect detection to this project.

2.2.1 Image Analysis Techniques

A large amount of automations regarding industrial processes, demand the important task of image analysis. Image processing is the procedure of analysing an image, a series of images or even a video, with the view of acquiring either a processed image or a set of characteristics and key parameters related to the particular project. The software package, may contain several implementations of algorithms regarding the detection of key points in the image, such as edges or corners, detection of colour variations or combinations of them. Several algorithms, either digital filters or intelligent detectors, have been developed with the view of optimising the automation in the analysis.

2.2.1.1 Digital Filters

The category of digital filters constitutes a very wide field of interest; thence, this sub-section focusses more on image editing with the view of the resolution improvement in concern of the project of automatic defect detection. The improvement of the resolution includes several factors, or in other words, various filters that react on the image's output. The main categories that someone may divide these filters are related to the operation of the analysis. **Noise reduction** filters consist the most useful and essential filter category; the filters significantly reduce the noise of an image or video, thus, improve the quality of the input information, as well as the output. However, the risk of removing significant features, because of wrong justification, is a drawback that always sets the optimum in higher level. In particular, in the publication of [R. Peters] et al. [29], a Morphological Image Cleaning (MIC) algorithm is being described as "careful" cleaner. Specifically, this algorithm preserves the small and thin features during the implementation of noise reduction filters. It uses the calculations of residuals between the original and the morphologically processed image, while it analyses regions of the image and discards these that it estimates as noisy regions. After the justification of the residuals, the output of the algorithm is a combination of the processed residual images, which is a smoothed image with carefully reduced noise. Furthermore, simpler filter that provides noise reduction as well, is the Median filter; a technique of digital filtering that processes the image's signal entry by entry. Specifically, the statistic-based filter processes the signal of the image and replaces every noisy entry of the signal with the median of the neighbouring entries [30]. This mask contains the pixels of the image, which are ranked with respect to the gray level variations. The Media Filter is widely known for its noise reduction performance, while it is worth-mentioning that it is a non-linear filter, hence, complex to mathematically described it.

On the other hand, the blur in an image consists an important noisy feature that demands different approach from the aforementioned filters. One of the most widely used **Image Correction Filters** is the Gaussian filter or Gaussian smoothing technique. Before describing the filter, it is of utmost importance the understanding of the Gaussian noise. This noise is created by the acquisition of the sensors' estimations; e.g. poor illumination or shadowing. Gaussian noise is a statistical noise, which maintains a Probability Density Function (PDF) equal to that of the normal distribution, or Gaussian distribution. In order to reduce the Gaussian noise, a spatial filter which contains noise reduction filters has to be applied. Usually, a spatial filter includes a noise reduction filter, such as the Median Filter, and an image smoother. The drawback of this implementation is the processing of the image with these filters. The Gaussian smoothing operation, which aims on decreasing the noise of the image, reduces also the quality of image result. Hence, the implementation of a spatial filter, included the noise reduction filter and the Gaussian filter, demands an application of image resolution improvement algorithm.

2.2.2 Simultaneous Localisation and Mapping (SLAM) - Image Processing

Following the reference of the SLAM algorithms, several application of drones development solve the computational problem of SLAM with fast processing algorithms. Regarding the image processing of the SLAM algorithms, the general procedure maintains important steps; namely, the **feature extraction** and **feature description** methods. These methods are oriented more in the optical approach of the SLAM. In brief, the *feature extraction* algorithms analyse the captured or recorded image of the drone based on techniques that focus on various features; namely, edge detectors and corner detectors. The edge detectors are used in order to detect sharp edges from the image that is divided in intensity gradients. The dominant step for the detection is the application of the double threshold binarization technique [31]. Applications that use the edge detectors are oriented more in the precise inspection, while this method is useful to recognise the boundaries or details of an infrastructure; namely, a wind turbine rotor blade.

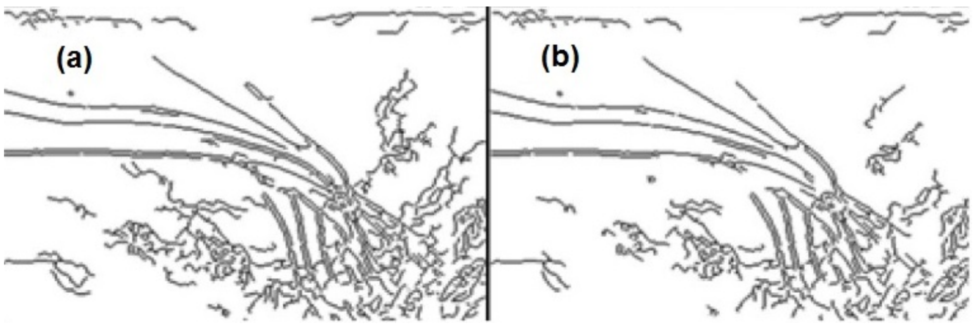


Figure 2.6: Edge detection images (a) Canny method with default threshold (b) Canny method with optimal threshold

The automatic detection of surface cracks in wind turbine rotor blades application et al. [H. Zhang] [32] makes use of the edge detection algorithms via support of the MATLAB function `edge()`. This function uses several edge detectors, such as the Canny edge detector [33]. Figure 2.6 illustrates the results of the edge detector algorithm in the aforementioned application.

The *feature description* algorithms are the following step of the feature extraction. The methods used for the description of the features, are of utmost importance for the system to input the features. This techniques not only describe the detected features into the system, but spot the most important key points on the total data. The dominant feature descriptors are the **Histogram of Oriented Gradients (HOG)** method, the **Scale-invariant Feature Transform (SIFT)** method and the **Speeded Up Robust Features (SURF)** method. In the publication et al. [I. Mondragon] [34], the application uses as dominant method the SIFT approach along with several others, in order to perform an experiment among them. It tracks several objects, including a building's windows, with the view of detecting and evaluating the matching of the keypoints. It is worth mentioning that SIFT method is used widely in the field of inspection, due to the use of the Hough line transform algorithm that enables the detection of straight line features, such as the frame of a wind turbine rotor blade [6].

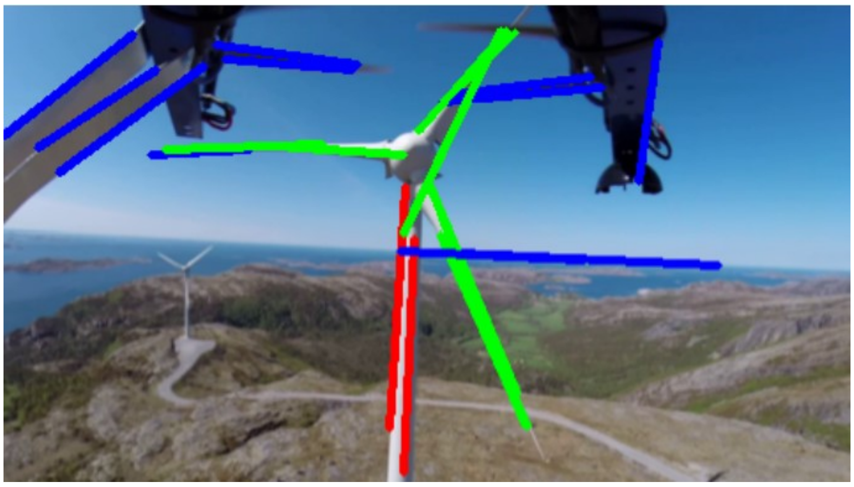


Figure 2.7: Hough Line Transform implemented to detect the rotation of the wind turbine rotor blades [6].

2.2.3 Scan Matching Techniques

Last but not least, the scan matching techniques are commonly used with the equipment of a laser or optical sensor and provide relative position information. These essential methods are divided in the categories of feature-to-feature (F2F), point-to-point (P2P) or even feature-to-point (F2P). The aforementioned techniques are based on the analysis of the image in features or points, such as corners, edges, lines or simple points. The main difference between the features and the points is that features are more accurate, but more difficult to extract them from the image, while points can be easily extracted but also contain lower information quality. Hence, an F2F (feature to feature) method scans the captured image of the environment and matches the features of the first image with the features of the next image. In contrast, a P2P (point to point) method uses the Iterative Closest Point (ICP) [35] technique in order to match each initial point with the closest final belief point. The outcome of these methods is the estimation of the relative position of the drone on the map. The publication in Sensors journal et al. [J. Tang] [36], uses the P2P scan matching method with the view of the indoor positioning of an Unmanned Ground Vehicle (UGV), equipped with laser sensor.

2.3 Platform Advanced Solutions

Based on the special demands of the project, this sub-section contains a brief description of platform solutions suitable for these kind of narrow operations. In particular, two platforms, namely a Collision-Tolerant UAV and a Spherical UAV are to be described as follows.

2.3.1 Multi-rotors with Collision-Tolerant Equipment

The most common approach of an indoor inspection, is by using multi-rotors, such as quadrotor or hexarotor, equipped with fail-safe frame around its propellers. This frame, constitutes an essential part that enables the drone to perform collision-tolerant missions. The aforementioned project from ETH Zurich et al. [J. Nikolic] [7], uses a quadrotor equipped with a fail-safe frame, which enables the drone to perform effectively indoor inspection of industrial facilities. In particular, Figure 2.8 illustrates the drone configuration that this research has used, which performs inspection inside a boiler.

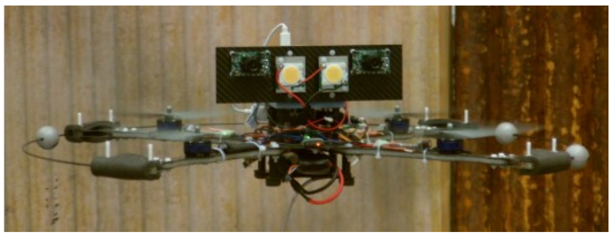


Figure 2.8: Prototype UAV used for inspection in a boiler [7]

A commercial yet effective solution is described in the paper, entitled *Indoor Autonomous Navigation of Low-Cost MAVs using Landmarks and 3D Perception* et al. [L. Apvrille] [37], which illustrates the techniques of navigation via landmarks and 3D perception with the use of a Micro Aerial Vehicle (MAV) for indoor navigation. The commercial open-source UAV used in this research is call Parrot AR Drone [38] and constitutes a lightweight, manoeuvrable drone focussed on indoor navigation. However, due to its low cost, the performance of the processor can not support computational demanding operations, such as reliable precise positioning; yet, the hardware of the drone is not open-source, hence, the it does not support laser positioning.

2.3.2 Spherical Drone - Advanced Solution

The shape of an omnicopter is similar to a ball, which contains the vital parts of the drone. It benefits from the normal drones in the aspect of safety, due to the barriers that surround the UAV, and in the high levels of manoeuvrability derived from the multi-directional movement capabilities. It constitutes an optimal configuration for precise indoor inspection, due to its high levels of safety; it is a collision-tolerant UAV that prevents any potential damage towards the inspecting subject. The company *Flyability SA.* [8] produces an innovative UAV in the shape of an omnicopter, called **Elios**. The Figure 2.9 illustrates the structure of the Elios drone, while the specifications state that it is equipped with a live video feedback module and a payload capable of lifting an HD camera along with a thermal imagery sensor. With respect to the structure, the Elios UAV is capable of performing precision indoor operations, without the risk of damaging the under-inspection object during a possible contact or collision.

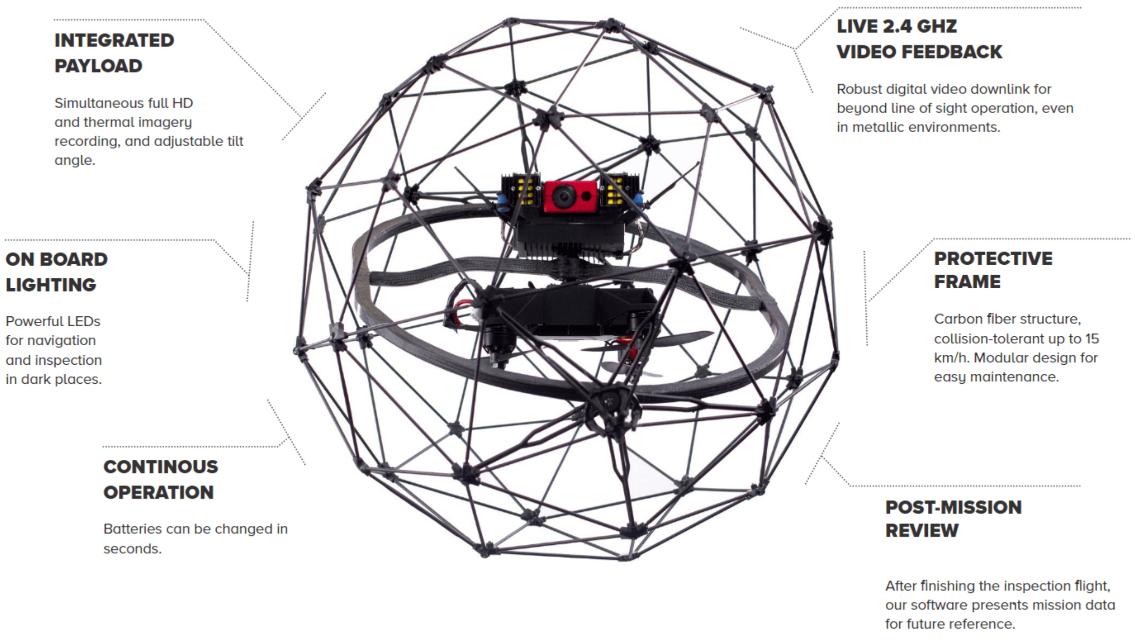

INTEGRATED PAYLOAD
Simultaneous full HD and thermal imagery recording, and adjustable tilt angle.
LIVE 2.4 GHZ VIDEO FEEDBACK
Robust digital video downlink for beyond line of sight operation, even in metallic environments.
ON BOARD LIGHTING
Powerful LEDs for navigation and inspection in dark places.
PROTECTIVE FRAME
Carbon fiber structure, collision-tolerant up to 15 km/h. Modular design for easy maintenance.
CONTINOUS OPERATION
Batteries can be changed in seconds.
POST-MISSION REVIEW
After finishing the inspection flight, our software presents mission data for future reference.

Figure 2.9: Elios - Collision-Tolerant Aerial Vehicle [8]

Chapter 3

Non-Destructive Testing - Penetrant Flow Detection

This chapter concerns the explanation of the Non-Destructive Testing (NDT), focussed more on the Penetrant Flow Detection (PFD) method. Specifically, this section describes the meaning of the NDT, while explains the method of Penetrant Flow Detection; the fully automated operation of this method is the dominant aim of this project.

3.1 Non-Destructive Testing - Theory

The Non-Destructive Testing (NDT) is the procedure of testing, evaluating and inspecting a material, component or structure with the dominant feature of not destroying it. In other words, after the process of the NDT the material or component can still be used. On the other hand, the destructive techniques are used in order to determine the physical properties of the material, such as the tolerance, ultimate tensile strength and fatigue strength, among others.

At present, the NDT is widely used in the industry of manufacturing with the view of optimising the quality of the process. The NDT improves the reliability of the product along with the integrity, while it lowers the overall production costs and provides the benefit of controlling the manufacturing process. It ensures the safety of the product by evaluating several manufacturing stages; a fully quality controllable procedure without wasting products for inspecting and evaluating them.

The NDT method vary among the process of testing and evaluation. In particular, based on the equipment used for the testing or the penetrant material, the NDT method maintains a similar name. Some of the most common Non-Destructive Testing techniques are listed below:

- Radiographic Testing (RT)
- Ultrasonic Testing (UT)
- Electromagnetic Testing (ET)
- **Liquid Penetrant Testing (PT)**
- Laser Testing Methods (LT)
- Magnetic Particle Testing (MT)
- Visual Testing (VT)
- Thermal/Infrared Testing (IR)
- Vibration Analysis (VA)

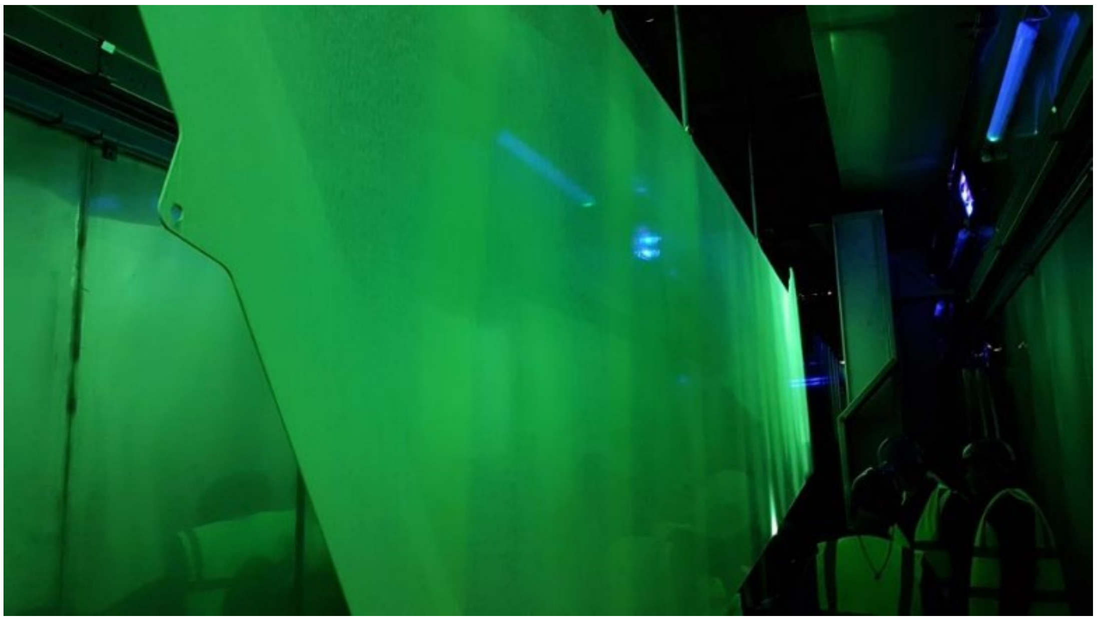


Figure 3.1: Penetrant Testing on Wing Panel: Inspection Stage

As it is aforementioned, each of the techniques holds a characteristic name with respect to the method of inspection or evaluation. For instance, the Thermal/Infrared testing, or infrared thermography, uses infrared radiation to map surface temperatures by receiving the heat radiation of the objects and environment. The equipment required for this technique is dominantly a thermal camera, which is commonly called ”infrared camera”. Despite the various implementations among the techniques, the main purpose of all these is to locate and characterise a material’s or component’s condition along with its potential flaws; with the view of deterring any future chance of malfunction to the overall system. This evaluation process has essential impact to the safety of the overall product; a small flaw in the manufacturing process of a wing panel, may lead to a crack on the wing and, as a result, to an aeroplane’s crash. Hence, the significance of the Non-Destructive Testing maintains exponential trend with respect to the safety requirements of the product.

3.2 Penetrant Testing - Theory

This Individual Research Project focuses on the NDT method of Penetrant Testing. This technique is applied by Airbus Group on wing panels, prior to the stage of anodising and painting. The aim of this testing is to detect potential defects or cracks on the under-manufacturing panel, before the stages of assemble.

The Penetrant Testing method makes use of a low viscosity penetrant material; the fundamental principle is when the component is coated with this material, which contains a visible or fluorescent dye. As soon as the penetrant solution has been applied to the component, it penetrates into the defects leaving an excessive portion on the surface. The excess penetrant has to be removed from the surface of the component, in order to reveal only the potential defects. Thence, the penetrant is trapped into the variations of the surface, revealing a defection that might be a significant flaw in the manufacturing of the component. The inspection process, after the penetrant application stage, varies between the two penetrant material categories. In other words, the penetrant might be ”visible”, which allows the inspection under normal light, or fluorescent material, which requires ”black” light for the inspection stage. The process that this project focus on is under ”black” light, specifically ultraviolet light, with the use of fluorescent penetrant material.

The stages of the fluorescent penetrant testing are listed as follows:

- Pre-Cleaning
- Application of Penetrant
- Removal of Excess Penetrant
- Application of Developer
- Inspection
- Post-Cleaning

At this point, it is worth-mentioning some key details of the testing, while an extended description of the procedure will be stated in the next sections. First and foremost, the surface should have been carefully cleaned prior to the testing, with the view of eliminating any chances of foreign materials or liquids to block the penetrant from entering the voids or fissures of the component. Similarly, the excess penetrant removal has to carefully performed, in order not to clear away penetrant from any void or fissure; in any other case, the procedure of this testing has to be repeated. Additionally, a dwell time between the applications of materials are of utmost importance for the reason of enabling the material to be applied efficiently on the surface of the product.

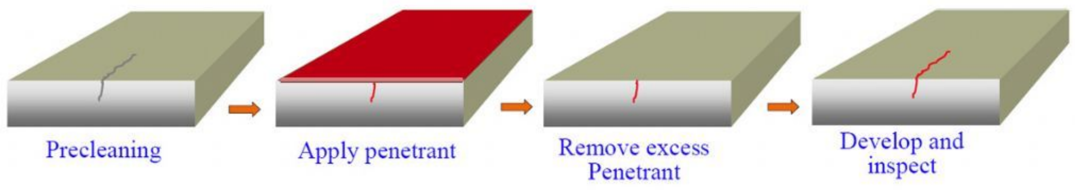


Figure 3.2: Fluorescent Penetrant Testing Procedure [9]

The excess penetrant removal method vary among three different techniques, which are based on the used penetrant: **(1)** with Solvent Penetrant, **(2)** with Water-Washable penetrant and

(3) with Post-Emulsification. The solvent solution is usually painted or brushed on the surface of the product, while after the dwell time the solvent is removed with a cloth damped with penetrant cleaner. Regarding the water-washable solution, the product has to be placed into a rinse station after the penetrant dwell time, in order to be cleaned from the excess penetrant with the use of a course water spray. This method is the most common cleaning technique, which provides fast and efficient results. The last category is a combination of a cleaner application along with the water-washable technique. In particular, post-emulsifiable penetrant is applied to the surface for a prescribed period of time, also called emulsifier dwell time, to remove the excess penetrant. Emulsifiers can be lipophilic (oil-based) or hydrophilic (water-based). After the emulsifier dwell time, the product is subjected to water wash in order to clean both the excess penetrant and the emulsifier.

3.3 Penetrant Testing - Airbus Implementation

As it aforementioned, the process of fluorescent penetrant testing is used by Airbus Group in order to evaluate the build quality of the wing panels, prior to the stage of painting and anodising. The procedure that Airbus follows is the general process mentioned above, while it makes use of advanced systems for better results. Below, the Airbus-followed process will be described with the support of a process map and a block diagram as well.

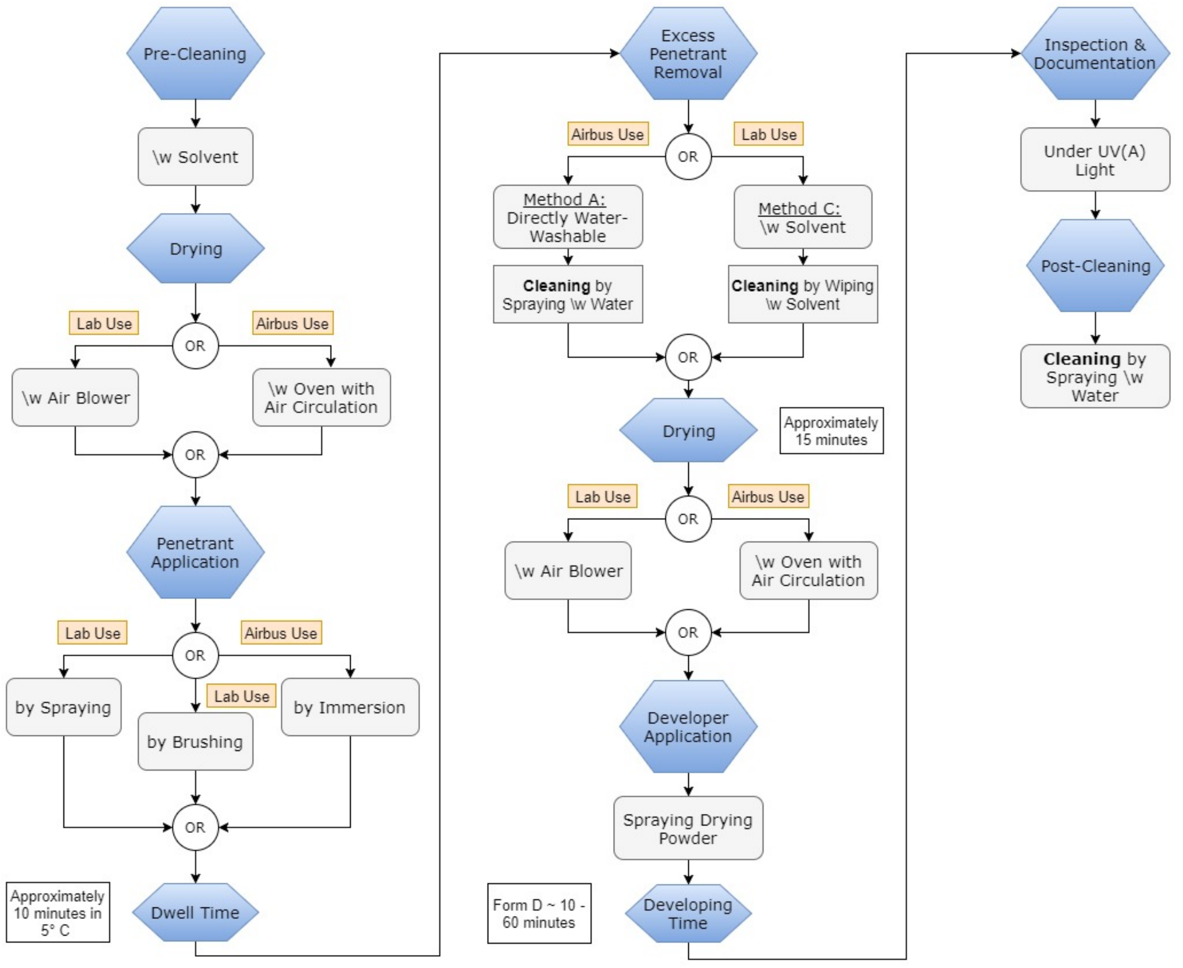


Figure 3.3: Block Diagram of the Penetrant Testing

The initial step of the procedure is of utmost importance; the **pre-cleaning** stage, provides a clean surface ready to be tested, while it reduces, or even eliminates, the chances of other foreign materials to affect the result of the testing. The surface should be free from oil, grease, water or other materials in order to proceed to the next step of the testing. The material used for the cleaning step, is a solvent material, namely ARDROX 9PR5, which is applied to the surface of the panel by spraying, prior to the testing procedure. Following the pre-cleaning step, the **drying** process is responsible for removing any humidity from the product. The procedure is applied with the use of a special-structured oven that performs air circulation, while the time of drying is essentially restricted above 15 minutes and in environmental temperature. After the drying stage, the **application of the penetrant** step is being performed. In particular, the panel is placed in a tank full of penetrant, and with the use of the immersion technique it is applied on the surface of the product. The **dwell time** is an essential step of the testing, because it provides the penetrant with time to be drawn or seep into a defect. The penetrant dwell time, with respect to the NDT Resource Centre [10], is the overall time that the penetrant is in contact with part of the surface. This time varies among the materials used in each operation, while a common range is between 5 to 60 minutes. Airbus procedure includes 10 minutes of dwell time, whenever this stage is needed. Nevertheless, the dwell time can be adjustable as long as the penetrant is not allowed to dry.

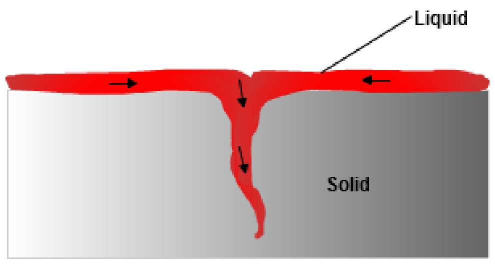


Figure 3.4: Penetrant Application Step [10]

Following the dwell time, the **excess penetrant removal** step maintains high levels of importance. During this stage, the excess penetrant must be carefully removed from the surface of the panel, while removing as fewer as possible from the potential defects. This step is performed with the use of the *Method A: Water-Washable*; direct rinsing with water is applied to the surface of the panel, thus removing the unnecessary penetrant. The Figure 3.5 illustrates the method of using a solvent and a cloth in order to remove the excess penetrant with less equipment and more careful.

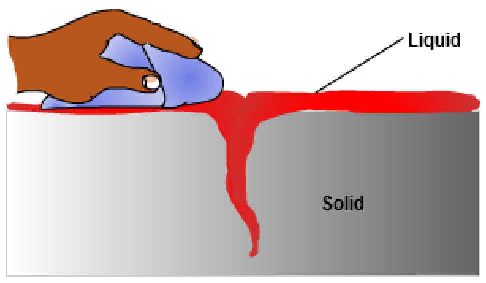


Figure 3.5: Excess Penetrant Removal [10]

After a drying stage due to the water-washable technique, next step is the **developer appli-**

cation. The developer is a material that enables the process of drawing the penetrant from the defect back to the surface, where is will be visible for the inspection. These developers come in a variety of forms and can be applied by spraying (wet developers), dusting (dry powder) or dipping. The process that Airbus follows includes an application of developer with the use of the dusting technique. In particular, the system used for dusting maintains high levels of autonomy while it automatically applies dust all over the wing panel. The developing time is the only parameter that has significant weight to this step. The developer has to stay for a predefined time on the surface of the wing, sufficient to enable the extraction of the trapped penetrant towards the surface of the panel. Common developing time fluctuates around 10 minutes, while the duration of 10 minutes is also performed by Airbus' procedure.

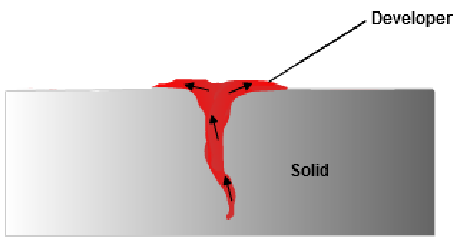


Figure 3.6: Developer Application Effect [10]

Final step of the Penetrant Testing, is the inspection process. Airbus procedure includes special trained technicians to be inspecting the wing panel after the completion of the aforementioned stages. This inspection takes from 1 hour until 4 hours, depends on the length of the wing panel, to be completed. The process is performed in a special designed room (Figure 3.7), under fully dark conditions, only under the light of Ultraviolet hand-lamps. The technicians wander around the wing panel with the UV light and inspect carefully the panel with the view of locating even the smallest possible defect. After the acquiring of the inspecting information and the completion of the Penetrant Testing, the panel is post-cleaned with the same technique mentioned in the pre-cleaning stage.

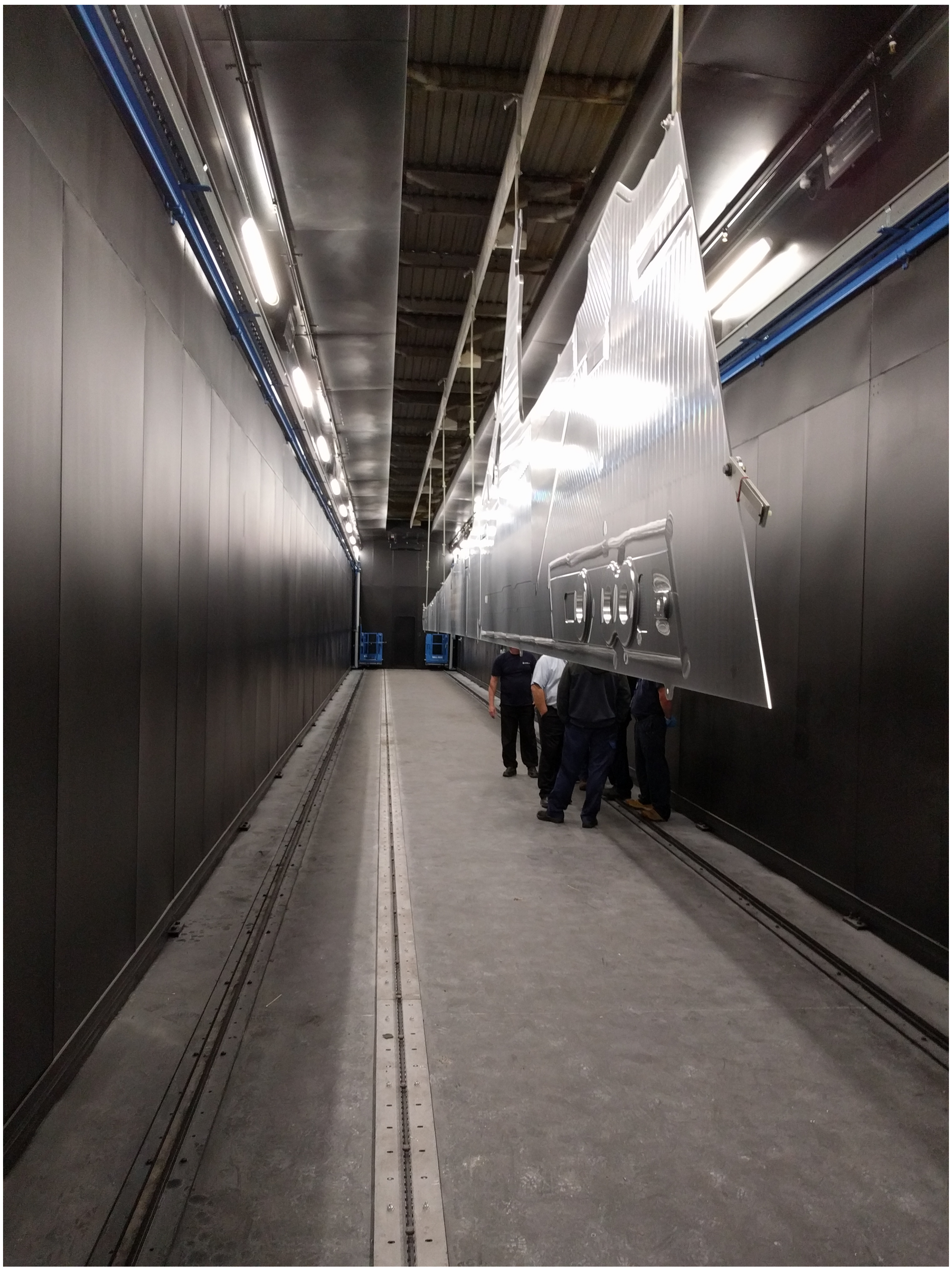
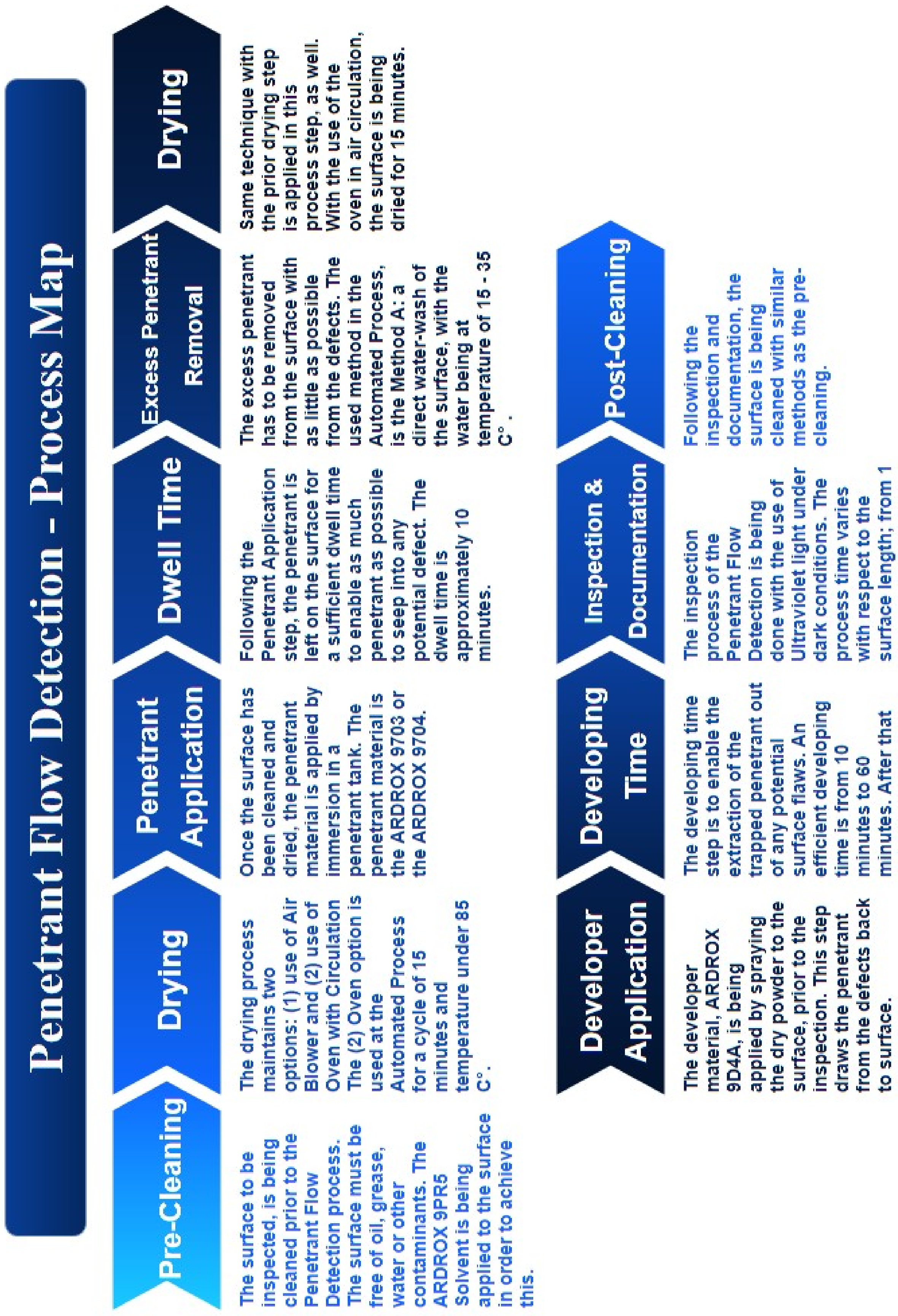


Figure 3.7: Airbus Inspection Room

In the following page, a process map (Figure 3.8) of the above-described procedure of the penetrant testing is illustrated as a stage-by-stage graph.



Penetrant Flow Detection - Process Map

Pre-Cleaning

The surface to be inspected, is being cleaned prior to the Penetrant Flow Detection process. The surface must be free of oil, grease, water or other contaminants. The ARDROX 9PR5 Solvent is being applied to the surface in order to achieve this.

Drying

The drying process maintains two options: (1) use of Air Blower and (2) use of Oven with Circulation. The (2) Oven option is used at the Automated Process for a cycle of 15 minutes and temperature under 85 C°.

Penetrant Application

Once the surface has been cleaned and dried, the penetrant material is applied by immersion in a penetrant tank. The penetrant material is the ARDROX 9703 or the ARDROX 9704.

Dwell Time

Following the Penetrant Application step, the penetrant is left on the surface for a sufficient dwell time to enable as much penetrant as possible to seep into any potential defect. The dwell time is approximately 10 minutes.

Excess Penetrant Removal

The excess penetrant has to be removed from the surface with as little as possible from the defects. The used method in the Automated Process, is the Method A: a direct water-wash of the surface, with the water being at temperature of 15 - 35 C° .

Drying

Same technique with the prior drying step is applied in this process step, as well. With the use of the oven in air circulation, the surface is being dried for 15 minutes.

Developer Application

The developer material, ARDROX 9D4A, is being applied by spraying the dry powder to the surface, prior to the inspection. This step draws the penetrant from the defects back to surface.

Developing Time

The developing time step is to enable the extraction of the trapped penetrant out of any potential surface flaws. An efficient developing time is from 10 minutes to 60 minutes. After that time, the developer has to re-applied.

Inspection & Documentation

The inspection process of the Penetrant Flow Detection is being done with the use of Ultraviolet light under dark conditions. The process time varies with respect to the surface length; from 1 hour to 4 hours inspection time.

Post-Cleaning

Following the inspection and documentation, the surface is being cleaned with similar methods as the pre-cleaning.

Figure 3.8: Penetrant Testing - Airbus Implementation

3.4 Penetrant Testing - Laboratory Simulation Implementation

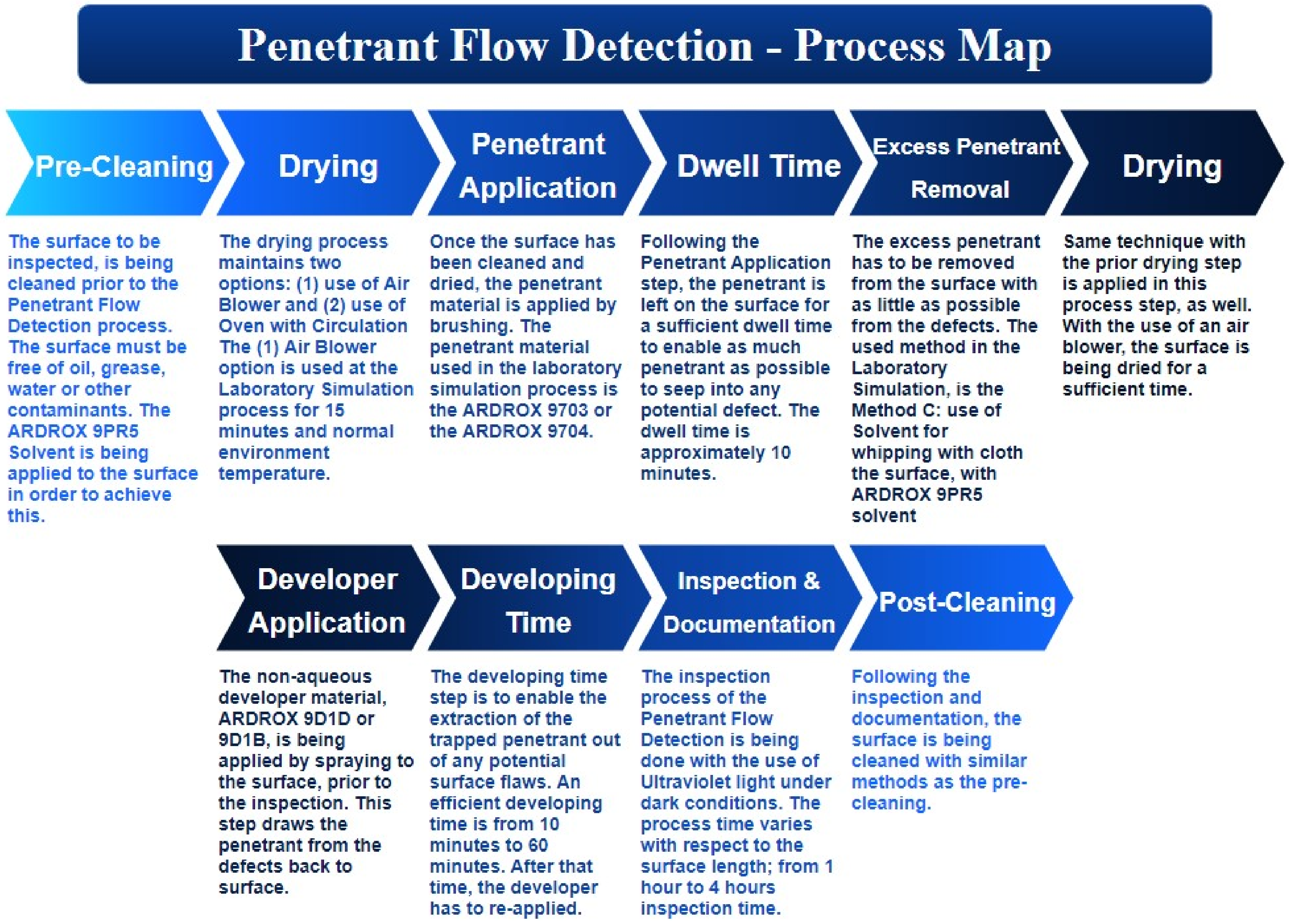
The simulation of similar to the aforementioned procedure had to be implemented in the laboratory of Cranfield University, in order to create similar results that can be analysed by the image analysis software package, described in the next chapter. Hence, the principles and requirements of the Penetrant Testing were adapted to the lab's standards in order to enable the inspection process. The similarities of the above-described procedure with the simulation technique are several, yet many steps were implemented with different technique due to absence of appropriate industrial equipment.

As one can notice in the Figure 3.3, there are two ways of the penetrant testing. The "Lab Use" branches of the block diagram are devoted to the laboratory simulation. Hence, a first image of the differences can be acquired just by looking the process diagram 3.3 stated in previous page.

Starting from the beginning, the **pre-cleaning** step is performed with the same option as before. The solvent is applied to the surface of the panel, with the purpose of cleaning it from any foreign material such as oil, grease or water. Following that, the **drying** process performed by an Air-Blower, instead of oven, due to lack of similar equipment. The stage of drying maintained similar results, yet the drying time lasted around 15 minutes, which is more time than the oven. Next, the **penetrant application** step was implemented by brushing the penetrant material to the surface of the wing panel; the material of use was the type ARDROX 9703 or 9704. Similar to the Airbus implementation, the dwell time stages maintained similar yet reduced duration, because of the increased temperature of the laboratory environment. The most important stage of the testing, the **excess penetrant removal** performed differently from the Airbus standards, due to the lack of suitable equipment. In particular, the *Method C: Solvent Application* is used in order to remove the excess penetrant, by whipping the surface of the panel with the use of the solvent ARDROX 9PR5. This technique were used by Airbus Group in their procedure as well, yet it is not cost efficient, due to the cost and great amount of solvent consumption, and not so effective as the water washable. Moreover, the **developer application** stage performed by spraying the material ARDROX 9D1D to the wing panel. As aforementioned in the previous section, Airbus performs this stage by spraying dry powder towards the wing, something more effective than the spray material used for the simulation experiments. A clear conclusion of the simulation-used material is the undesirable glare that it creates, which reduces the effectiveness of the UV inspection. Finally, the same techniques for the rest of the processes are the same implemented by Airbus.

Hence, comparing the simulation method followed in the laboratory with the method implemented in the Airbus plant, one can say that the industrial way is more effective than the simulation. Nevertheless, the difference during the inspection process seemed to be equally acceptable, yet the common measuring point differs from wing panel to wing panel. Hence, the performance evaluation by implementing the simulation method once and having only theoretical knowledge of the Airbus implementation, is something difficult to justify.

In the next page, Figure 3.9 illustrates, in similar pattern as above, the process map of the simulation procedure of the penetrant testing on a wing panel. Following that, photographs from the simulation process that illustrate the followed procedure are stated in the Appendix A.3.


Penetrant Flow Detection - Process Map

Pre-Cleaning

The surface to be inspected, is being cleaned prior to the Penetrant Flow Detection process. The surface must be free of oil, grease, water or other contaminants. The ARDROX 9PR5 Solvent is being applied to the surface in order to achieve this.

Drying

The drying process maintains two options: (1) use of Air Blower and (2) use of Oven with Circulation The (1) Air Blower option is used at the Laboratory Simulation process for 15 minutes and normal environment temperature.

Penetrant Application

Once the surface has been cleaned and dried, the penetrant material is applied by brushing. The penetrant material used in the laboratory simulation process is the ARDROX 9703 or the ARDROX 9704.

Dwell Time

Following the Penetrant Application step, the penetrant is left on the surface for a sufficient dwell time to enable as much penetrant as possible to seep into any potential defect. The dwell time is approximately 10 minutes.

Excess Penetrant Removal

The excess penetrant has to be removed from the surface with as little as possible from the defects. The used method in the Laboratory Simulation, is the Method C: use of Solvent for whipping with cloth the surface, with ARDROX 9PR5 solvent

Drying

Same technique with the prior drying step is applied in this process step, as well. With the use of an air blower, the surface is being dried for a sufficient time.

Developer Application

The non-aqueous developer material, ARDROX 9D1D or 9D1B, is being applied by spraying to the surface, prior to the inspection. This step draws the penetrant from the defects back to surface.

Developing Time

The developing time step is to enable the extraction of the trapped penetrant out of any potential surface flaws. An efficient developing time is from 10 minutes to 60 minutes. After that time, the developer has to re-applied.

Inspection & Documentation

The inspection process of the Penetrant Flow Detection is being done with the use of Ultraviolet light under dark conditions. The process time varies with respect to the surface length; from 1 hour to 4 hours inspection time.

Post-Cleaning

Following the inspection and documentation, the surface is being cleaned with similar methods as the pre-cleaning.

Figure 3.9: Penetrant Testing - Airbus Implementation

Chapter 4

Automatic Detection of Defects

The Chapter of ”Automatic Detection of Defects” is focussed on the introduction and description of the chosen image analysis techniques, concerning the automatic detection of defects under ultraviolet light, the explanation of the relative developed code and the illustration and explanation of the acquired results. Specifically, the chapter will divided in the description of the filters and detectors used for the development of this project, the extended description of the developed code and the illustration of the results and the performance evaluation.

4.1 Theoretical Approach - Image Analysis

This section states the approaches of image processing filter, detectors and other algorithms used for the implementation of the image processing software package. Besides the description of the theoretical algorithms, a practical explanation with the use of the developed code will be stated as well; the library that used for the image processing is the OpenCV - Open Computer Vision library and coded in Python language. It is worth-mentioning that the developed script for Video Processing is stated in the Appendix A.1.1 section. The architecture of the code is similar to the Image Processing code, detailed as follows, with the difference of using each frame of the video instead of one image.

4.1.1 Filters

Starting from the background of the image, the **Gamma Correction** filter enables the algorithm to adjust the colours of the image to the project's standards, while it edits the background in order to reduce the unimportant features that might affect the detection outcome. Specifically, the non-linear filter is used in order to encode and decode the luminance in the image and it is generally characterised by the following gamma equation:

$$V_{out} = AV_{in}^{\gamma} \tag{4.1}$$

where: V_{in} is the real input value, γ is the gamma correction parameter, A is a constant value and V_{out} is the output value.

The principle of the gamma correction filter adopts the power function that characterises the human perception of light and colour. This power function describes the brightness perception, with sensitivity to the variations between dark and light tones. Hence, changing the input image by gamma correction can cancel the non-linearity and adjust the luminance levels of the image to the demands of the project. The formula for calculating the resulting output is as follows:

$$I' = 255 \cdot \left(\frac{I}{255}\right)^{1/\gamma} \tag{4.2}$$

Hence, based on this equation, the developed function focusses on building a lookup table to mapping the pixel values [0, 255] to their adjusted gamma values. In particular, the line 2 defines the inverted gamma for the equation, while the lines 3 and 4 implement the aforementioned equation and maps each pixel value. The output of this function is a gamma corrected image with the use of the created look up table.

```
1 def adjust_gamma(image, gamma=1.0):
2         invGamma = 1.0 / gamma
3         table = np.array([((i / 255.0) ** invGamma) * 255
4                 for i in np.arange(0, 256)]).astype("uint8")
5
6         # apply gamma correction using the lookup table
```

```
        return cv2.LUT(image, table)
```

At this point it is worth-mentioning that a conversion of the original coloured image into a **grayscale**, is made in the beginning of the code. A grayscale digital image contains only information regarding the intensity variation, while each pixel has a value of a simple sample. The technique of converting the original image into grayscale is commonly used in the image processing, because it enables the segmented image analysis and provides with faster results concerning the processing of the intensity variations. The following line of code, illustrates the implementation command of converting an coloured image into grayscale one with the use of OpenCV library and Python language. In particular, it converts the processed with gamma correction image, as mentioned above, to a grayscale one.

```
gray = cv2.cvtColor(adjusted, cv2.COLOR_BGR2GRAY)
```

Following that, a technique of detecting the amount of blur into an image has been implemented as well. The used method is the **Variance of Laplacian** [39]. This method utilises the Laplacian variance of the image in order to estimate the focus measure. With respect to the publication of [J. Pachero] et al. [40], it is based on the second derivative operator technique "for passing the high spatial frequencies that are associated with sharp edges". Hence, the outcome of the second derivative operator, in which case the Laplacian operator, is an estimate of the focus measure. The Laplacian kernel is the key matrix that can estimate the operator:

$$L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{4.3}$$

The method that estimates the focus measure is the calculation of the variance of the absolute values. In other words, it makes use of the Laplacian estimation, stated in the [40], and calculates the variations of the values.

$$Laplacian \quad Variance(I) = \sum\sum[|L(m,n) - \bar{L}|]^2 \tag{4.4}$$

Providing this method, the implementation in code language with the use of the OpenCV library is much simpler. The sub-library of OpenCV, called ”Laplacian”, provides with this technique, while the developed code is as follows.

```
def variance_of_laplacian(image):
        # compute the Laplacian of the image and then return the focus
        # measure, which is simply the variance of the Laplacian
        return cv2.Laplacian(image, cv2.CV_64F).var()
```

Specifically, the function takes as input the image and returns the focus measure calculated based on the above mentioned technique. The assumption is that if the image contains high variance as indicated from the output of the function, there will be extensive outcomes which leads to in-focus image. On contrary, if the image contains low variance the responses will be a few, indicating small amount of edges in the image. This means that in the image, not many edges can be detected and the image is blurry.

In this project of automatic detection of defects, the filters of **dilation and erosion** were used as well. These morphological operations apply a structuring element to an input image and generate an output image. Specifically, the dilation and erosion operations reduce the noise of the image, join or isolate elements and locate the intensity bumps or holes of the image. In the particular project, these operations are used with the view of reducing the image noise and joining the gaps between close detected features.

The *dilation* operation is performed by the convolution of the image with a predefined kernel, which can have any size or shape convenient to the image. The command used in the developed code, with the use of the OpenCV library, edits the processed image from the previous filters in order to reduce the noise and close the gaps of the features. The ”edged” image is after the application of Canny edge detection algorithm.

```
edged = cv2.dilate(edged, None, iterations=1)
```

The *erosion* operation is the supplement of the dilation, by computing the local minimum over the area of the kernel. Specifically, as the kernel is scanned over the image, the minimal pixel value is computed while it replaces the image pixel under the anchor point. Similarly, with the support of the OpenCV library, the command used to apply erosion to the processed image, follows.

```
edged = cv2.erode(edged, None, iterations=1)
```

4.1.2 Detectors

As stated in the Chapter of Literature Review, the detector algorithms are basic methods that process the image and extract useful structural information, such as the edges and corners. After the implementation of the aforementioned filters, the application of a **Canny Edge Detector** can easily identify the key features, in other words the defects, inside the captured image.

The multi-stage algorithm of Canny edge detector [33] is divided in 5 main steps:

- The application of Gaussian filter to smooth the image and reduce the noise.
- The estimation of the image's intensity gradients.
- The application of a non-maximum suppression to get rid of spurious response to edge detection.
- The edge tracking with the use of the Hysteresis method.

The Gaussian filter, as stated in the chapter of the Literature Review 2.2.1, is the most common filter that reduces the noise and smooths the image. This filter is used by Canny edge detector to prevent false detections caused by noise. Following that, the finding of the image's intensity gradients determines the direction of the detected edge. In other words, a detected edge in an image may point in various directions, hence, one of the stages of the Canny detector uses four filters in order to track the horizontal, vertical and diagonal edges. Every filter of these, calculates the value of the first derivative in the horizontal and vertical direction, which are used to determine the edge gradient and direction.

$$G = \sqrt{G_x^2 + G_y^2} \tag{4.5}$$

$$\Theta = \arctan 2(G_y, G_x) \tag{4.6}$$

The non-maximum suppression consists an edge thinning technique. Following the application of the gradient calculation, the edge that is extracted from the gradient value is blurry. Thus, the non-maximum suppression provides support of suppressing the gradients to 0, except the local maximum that indicates location with the sharpest change of intensity value. The method of performing such filtering is by comparing the edge strength of the under-evaluation pixel, with the edge strength of the pixel in the positive and negative gradient directions. Hence, if the edge strength of the current pixel is larger than the value of the other pixels in the mask with the same direction, the value of it will stay constant; otherwise, the value will be suppressed. The nest step of the multi-stage Canny algorithm, is the implementation of a double threshold. This filter, reduces the noise in the resulting edges by filtering out the low value gradient edges and preserve these with high gradient value. Last but not least, the final step of the Canny edge algorithm is the edge tracking with the use of hysteresis. In order to prevent the detection of weak edges, blob analysis is applied to the image by analysing the weak edge pixel along with its 8 neighbouring pixels. If the analysis of the neighbouring pixels shows up a strong pixel included, then the weak pixel can be identified as one that should be preserved.

In the library of OpenCV the application of Canny edge detector is as simple as one command line.

```
edged = cv2.Canny(gray, 50, 50)
```

In the following Figure 4.1, the grayscale processed image is illustrated side-by-side with the output of the Canny edge detector.

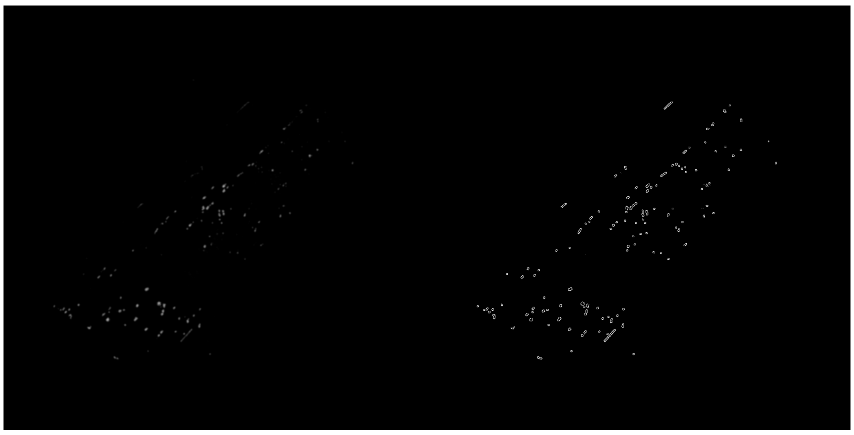


Figure 4.1: Comparison view of grayscale image with the output of the Canny edge detector

4.2 Defects Classification Technique

After the detection of the defects with the algorithms and implementations described above, the determination of the size and position of the defects is of utmost importance. Based on the size of the detected defect, the technician that will review the report will give more emphasis to the size of the defects; while if he/she desires to inspect this defect in person, the coordinates with respect to the overall size of the panel will guide him/her to the defect.

The method followed to define the size and position of each defects is based on the contour analysis of the image. A ratio of the detected width with a known width (from an initialisation) outputs the number of pixels per a given metric. The implementation of the size detection will be described by analysis the developed code step-by-step.

Starting from the beginning of the implementation, the code loops over each contour in order to analyse the included points.

```
1  for c in cnts:
2          # if the contour is not sufficiently large, ignore it; initial: 100
3          if cv2.contourArea(c) < 5:
4                  continue
5  
6          # compute the rotated bounding box of the contour
7          orig = adjusted.copy() # remove .copy() to illistrate full detected image
8          box = cv2.minAreaRect(c)
9          box = cv2.cv.BoxPoints(box) if imutils.is_cv2() else cv2.boxPoints(box)
10         box = np.array(box, dtype="int")
11 
12         # order the points in the contour such that they appear
13         # in top-left, top-right, bottom-right, and bottom-left
14         # order, then draw the outline of the rotated bounding
15         # box
16         box = perspective.order_points(box)
17         cv2.drawContours(orig, [box.astype("int")], -1, (0, 255, 0), 2)
```

In the Lines 3-4, the condition of ContourArea(c) to be smaller than the value of 5, constitutes a threshold to determine the lowest contour to be inspected. In order to detect and calculate the size of even the smallest ones, the threshold value of 5 has been chosen. Following that, the Lines 8-10 are responsible for the calculation of the rotated bounding box, while the Lines 16-17 are sorting out the points in the contour from the top-left to the bottom-right and drawing the outline of the calculated bounding box. The computed bounding box contains the local coordinates of the contours, which represent the detected defects on the panel. These local coordinates are translated to universal coordinates with the combination of the wing panel size and the current position acquired from the localisation sensor.

```
# loop over the original points and draw them
for (x, y) in box:
        cv2.circle(orig, (int(x), int(y)), 5, (0, 0, 255), -1)
```

The loop condition is responsible for drawing a circle over each original point of the contours.

```
# unpack the ordered bounding box, then compute the midpoint
# between the top-left and top-right coordinates, followed by
# the midpoint between bottom-left and bottom-right
# coordinates cv2.imwrite("adjusted.jpg",adjusted)

(tl, tr, br, bl) = box
(tltrX, tltrY) = midpoint(tl, tr)
(blbrX, blbrY) = midpoint(bl, br)

# compute the midpoint between the top-left and top-right points,
# followed by the midpoint between the top-righ and bottom-right
(tlblX, tlblY) = midpoint(tl, bl)
(trbrX, trbrY) = midpoint(tr, br)

# draw the midpoints on the image
cv2.circle(orig, (int(tltrX), int(tltrY)), 5, (255, 0, 0), -1)
cv2.circle(orig, (int(blbrX), int(blbrY)), 5, (255, 0, 0), -1)
cv2.circle(orig, (int(tlblX), int(tlblY)), 5, (255, 0, 0), -1)
cv2.circle(orig, (int(trbrX), int(trbrY)), 5, (255, 0, 0), -1)

# draw lines between the midpoints
cv2.line(orig, (int(tltrX), int(tltrY)), (int(blbrX), int(blbrY)),
        (255, 0, 255), 2)
cv2.line(orig, (int(tlblX), int(tlblY)), (int(trbrX), int(trbrY)),
        (255, 0, 255), 2)
```

Proceeding to the calculation of the midpoints, this is made by the function stated in the beginning of the developed code and illustrated in the following two lines.

```python
def midpoint(ptA, ptB):
        return ((ptA[0] + ptB[0]) * 0.5, (ptA[1] + ptB[1]) * 0.5)
```

The *midpoint* function returns the middle of two points of contour. In the lines 7-13 of the code attached above, the midpoints of the top-left, top-right, bottom-left and bottom-right points are being calculated. In the following lines 16-19, the computed midpoints are being shaped to the image with a circle shape. The purpose of this, is the illustration of the midpoints of the contours to the image. Similarly, in the lines 22-25 the vertical and horizontal lines of the midpoints are being shaped for the shape of visualisation.

```python
1 # compute the Euclidean distance between the midpoints
2 dA = dist.euclidean((tltrX, tltrY), (blbrX, blbrY))
3 dB = dist.euclidean((tlblX, tlblY), (trbrX, trbrY))
4
5 # compute the size of the object
6 dimA = dA / pixelsPerMetric
7 dimB = dB / pixelsPerMetric
```

The most essential calculations of this developed code is contained in the above script. The lines 2-3 are responsible for calculating the euclidean distance between the pre-computed midpoints; the top-left-right with the bottom-left-right, and the others respectively. The lines 5-6 are of utmost importance; they make use of the euclidean distance and the initialised pixel-per-metric value to compute the size of the detected defect.

4.3 Results Illustration & Evaluation

The resulting images of a sample provided by Airbus Group follow, along with an explanation of them. Unfortunately, further experiments that test the reliability of the code along with further improvements regarding video processing or processing time in general, have to be included to future work due to the absence of necessary components that made impossible to advance this project in-time.

Figure 4.2 illustrates the captured image under UV light, with a material that has been tested with the method of penetrant testing. This material consist a simulation component that contains numerous defects detected only by the non-destructive testing that Airbus' technicians performed.

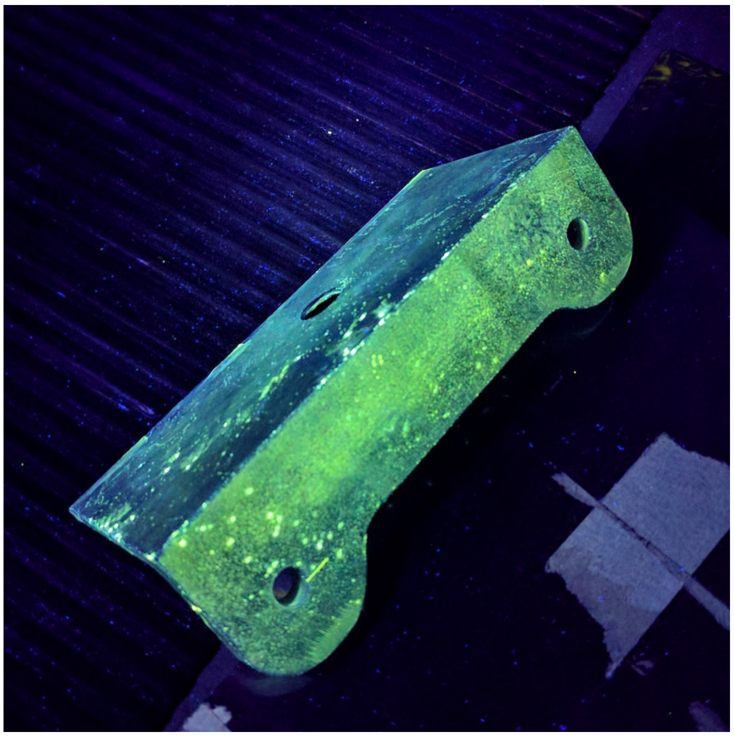


Figure 4.2: Sample after Penetrant Testing with numerous defects

Based on this sample, the developed code explained in the previous sub-section maintains results that prove the novelty of the project and implementation. Specifically, in Figure 4.3 the gamma corrected image is displayed, while it shows the numerous defects in a more clear to the eye way.

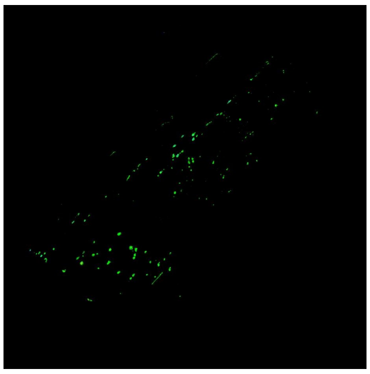


Figure 4.3: Adjusted image with Gamma Correction

The Figure 4.3 displays only the corrected image after the processing with the gamma correction filter. The Canny edge detector that follows in the script, detects the above illustrated defects and provides with input the classification code. Figure 4.4 illustrates this effect and shows that each of the defects shown in the adjusted image, are detected from the Canny edge detector.

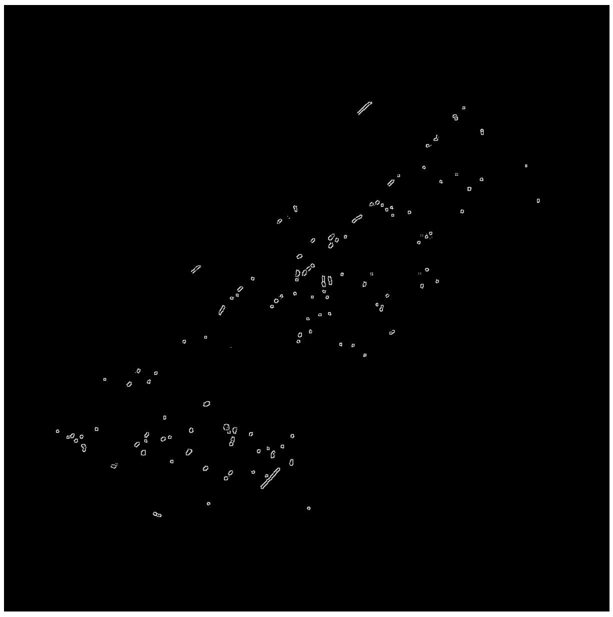


Figure 4.4: Defects detected with the use of Canny edge detector

Following that, the developed code that estimates the size and position of the defects is depicted in the Figure 4.5. The code detects each of the defects and calculates their relative size and position to the image. Then the code translates the relative coordinates with respect to the wing panel and stores them in an 2 dimensional array. This combined figure is consisted by two captions of the detection and size calculation process. The outcome of the developed code is that this component, based on the provided sample image, contains 114 detected defects from 0.1 inch size to 0.5 inches. However, to evaluate this result along with the reliability of the code, a comparison of the outcome with the opinion of an expert is of utmost importance. Unfortunately, this has to be included also to the future work due to lack of provided evaluation data, until the data of the thesis submission.

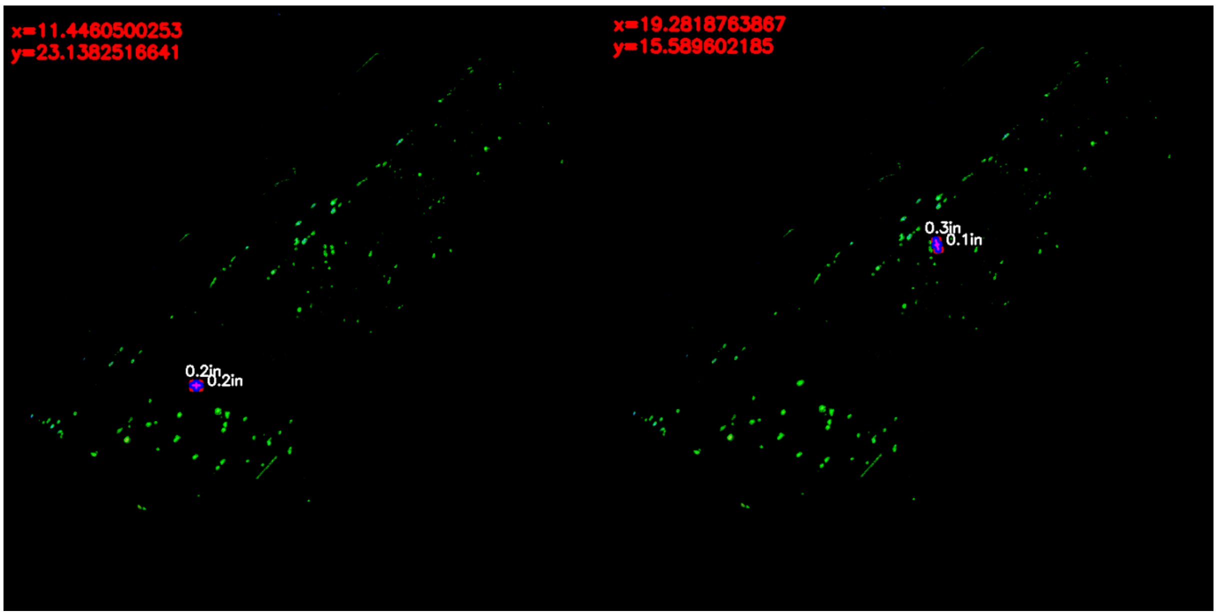


Figure 4.5: Illustration of size and position estimation of the detected defects

Chapter 5

Hardware Implementation - Localisation

The Chapter of Hardware implementation includes the description of the hardware components used for the implementation of this project. In particular, the KIO RTLS indoor positioning system, the UAV platform along with the chosen processor are to be stated and described as follows. Furthermore, the Robot Operating System (ROS) along with its implementation and purpose of use to the project, will be stated as well. Last but not least, the code implementation for the RTLS in order to communicate with the ROS and processor, is also to be analysed.

5.1 Localisation Sensor - RTLS

Concerning this project, the requirements for the option of the localisation sensors were several. The most important factor is the GPS-denied environment, while the narrow room of inspection poses the provided accuracy of the sensor to a highest level of significance. Additionally, the parameters of cost and delivery time consisted factors of consideration, while the project was supposed to be delivered to a specific date arrange by Cranfield University and Airbus Group. Hence, the decision of KIO Real-Time Location System (RTLS) was the ending one, due to the specifications that it provides.

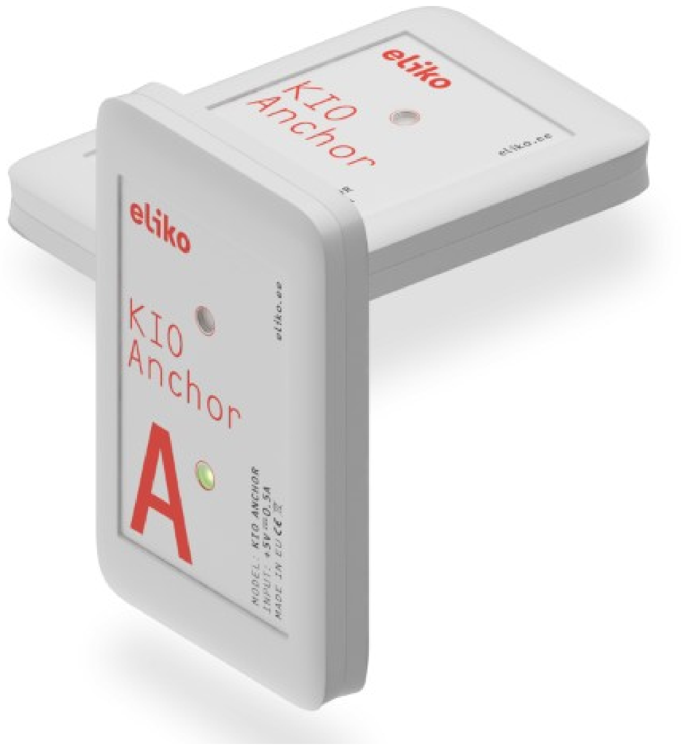


Figure 5.1: KIO RTLS Evaluation Kit [2]

5.1.1 Description of the Sensor

The KIO RTLS positioning system [2], as previously referred to the chapter of Literature Review 2.1.2, uses the Ultra-Wideband technology to locate the provided tag sensor to the environment. Specifically, the system is capable of detecting the tag hardware with Line-of-Sight (LOS) or Non-Line-of-Sight, due to its architecture of using UWB at frequency of 3.1 - 4.8 GHz. It is worth-noting that the UWB positioning technology is stated in the section 2.1.3.

The Evaluation Kit that consists the RTLS sensor includes 1 tag and 4 anchors; the tag is placed on-board the UAV, while the anchors are placed in the corners of the inspection room. With the use of the Time-of-Flight method the system calculates the distance between the anchors and the tag, and converts this euclidean distance into a coordinate system with centre of interest the position of the tag. This procedure happens multiple times per second due to the higher frequency of UWB technology that is used. Furthermore, the immunity against interfering obstacles is a characteristic that increases the effectiveness of the results. The power consumption along with the lightweight feature constituted an key parameter, as well. Each of the the anchors and the tag maintain a power capacitor or battery, that enables the wireless operation of these without the necessity of power charging; to charge them each hardware has to be connected via USB cable to a power charger.

The accuracy provided by this sensors fluctuates around the 30 [cm] with the use of the regular cell configuration, while it can reach the level of 5 cm with the small cell configuration. The accuracy is strongly depended on the clearness of the Line-of-Sight. Based on the specifications of the product, it is claimed that the error is 0.1-0.3 [cm], however this numbers are in function of the indoor environment size and the interference from other radio frequency sensors.

5.1.2 Trilateration Algorithm

As mentioned in the Literature Review chapter 2.1.5, the trilateration method combines the euclidean distances from 2 or more sensors, in order to estimate the position of the intersection point. It is worth-mentioning that in order to calculate a 3D position, it is needed to use 3 or reader sensors (anchors). With respect to this simple technique, the RTLS sensor, used in this project, can estimate the position of the tag based on the anchors position.

In particular, the anchors are fixed-placed in the corners of the operating room, with known coordinates based on a pre-defined point of origin. Hence, the anchors with the technology of UWB estimate their distances towards the receiver sensor (tag). The following equation describes the euclidean distance of the anchors from the tag.

$$r_i^2 = (x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 \qquad \text{for i = 1,2,3,4} \rightarrow \text{Number of anchors} \tag{5.1}$$

where: r_i is the distance, x, y, z are the unknown tag coordinates and x_i, y_i, z_i are the constant coordinates of each anchor.

Therefore, solving the 3 equations with 3 unknowns problem, results in the output of the receiver's coordinates.

Based on this theoretical approach, the development of a simple code that receives the distances from the anchors and calculates the tag's coordinates, follows along with its description.

```
def callback(self, data):

        # rospy.loginfo(data.data)
        info = data.data
        info = info.split(" ")

        # define anchor IDs and acquire distances
        #----------------------- Anchor A --------------------------------------#
        anchor_a = info[1]
        self.distance_A = float(info[2])
        #----------------------- Anchor B --------------------------------------#
        anchor_b = info[3]
        self.distance_B = float(info[4])
        #----------------------- Anchor C --------------------------------------#
        anchor_c = info[5]
        self.distance_C = float(info[6])
        #----------------------- Anchor D --------------------------------------#
        anchor_d = info[7]
        self.distance_D = float(info[8])

        distance = [self.distance_A,self.distance_B,self.distance_C,self.distance_D]

        self.coordinates = fsolve(self.solver,self.initial)
        print(self.coordinates)

        self.talk_x.publish(str(self.coordinates[0]))
        self.talk_y.publish(str(self.coordinates[1]))
        self.talk_z.publish(str(self.coordinates[2]))

        self.serial_pub()
```

Starting from the `callback()` function, it continuously receives from the Publisher script, stated in the Appendix A.2.1, the euclidean distances of the anchors from the tag. These information have to be split from the `String` format into 3 values. The `info.split(" ")` function detects the gap between the values of the string line and splits them into 3 strings. Following that, the distance values `distance_A` to `distance_D` are float converted values that contain the euclidean distance of each anchor, respectively. Hence, the distance values are published to 3 different ROS topics, with the view of sharing the information to other scripts. The function `fsolve(self.solver, self.initial)` is the responsible function that solves the 3 equations problem.

```python
def solver(self, coordinates):
    x = coordinates[0]
    y = coordinates[1]
    z = coordinates[2]

    # distance = callback()
    self.F[0] = pow((x - self.xan[0]),2) + pow((y - self.yan[0]),2) + pow((z - self.zan
        [0]),2) - self.distance_A
    self.F[1] = pow((x - self.xan[1]),2) + pow((y - self.yan[1]),2) + pow((z - self.zan
        [1]),2) - self.distance_B
    self.F[2] = pow((x - self.xan[2]),2) + pow((y - self.yan[2]),2) + pow((z - self.zan
        [2]),2) - self.distance_D

    return self.F
```

The `solver()` function initialises the x, y, z coordinate values. Moreover, in this function the euclidean equations are defined with respect to the received distances. The output of this function is an array $F[]$ that contains the three equations. With the use of the python function `fsolve()`, the 3 equations are solved and the output is the tag coordinates x, y, z.

5.1.3 Installation - Environment & Drone

The installation of this particular sensors to the environment or on-board the drone is simple.

Starting from the drone, the tag that will be continuously located is connected to the processor of the drone, in this case a Raspberry Pi 3, via USB. The purpose of this is to direct transmit the positioning information towards the processor in order to be processed and used to the operation. Additionally, this connection provides with the advantage of not depending to the battery of the tag, while it is draining the needed energy from the large capacity battery of the drone. The integration of the data to the system is being done with the use of the Robot Operating System (ROS), which is to be described in the next sections.

The installation of the anchors to the corners of the environment is a procedure that demands more cautiousness. Firstly, a point of origin in the room has to be determined in order to evaluate the system. This point is the zero coordinate point in the XYZ-axis, (0,0,0), while it provides support for initialising the position of the anchors in the environment. Following that, the anchors are placed in the corners of the room, as showed in the Figure 5.2, in which way it provides the 3-D position.

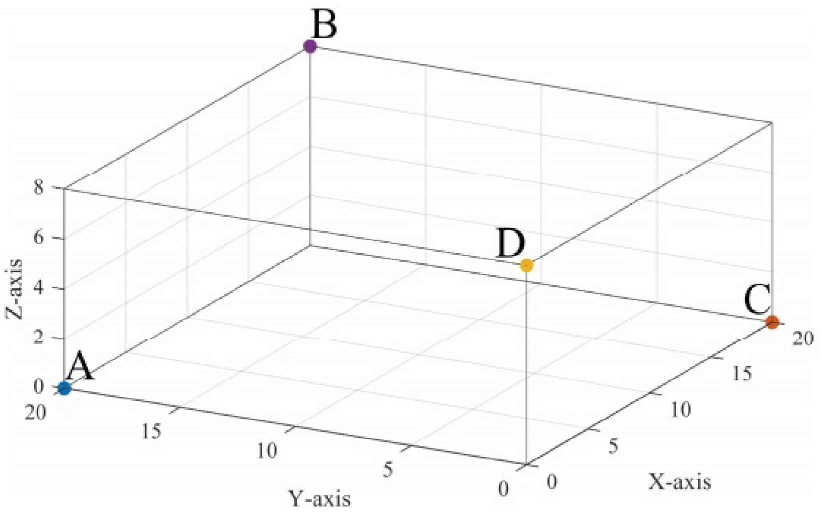


Figure 5.2: KIO RTLS Anchor Positions

The key factor that needs caution in the installation of the RTLS is the position at least of one of the anchors close to the main processor. One the anchors should be the ”chief anchor”, which sends the location data to the processor that performs the trilateration method (described in 2.1.5) in order to determine the distances of the tag and the anchors. Thus, after placing the anchors to the corners of the inspection room, the computing of their distance in relation to the zero coordinate point is the so-called calibration procedure. This determines the X, Y, Z coordinates of each of the anchors in relation to a known starting point, which improves the accuracy of the distance calculations.

KIO RTLS works sort of like an indoor GPS; by extrapolating positioning data with the system of anchors and tags, the 4 anchors define the tracking area while the tag is located by calculating its relative euclidean distance to each of the anchors in real-time. These raw data of calculations are fed up to the main processor, in which they are combined to create a coordinate system of X, Y, Z in relation to the position of the tag. Figure 5.3 provides with an illustration of the simple yet efficient technique that this system provides. Moreover, the implementation of the API provided by the company to the algorithm of the project is open-source. This enabled the developed path-planning code along with the valuable real-time positioning data to be implemented to the scripts of the project. The current position of the drone is used to the image processing algorithm in order to cross-check the captured image position and to translate correctly the detected defects to a universal coordinate system in relation to the real wing panel length.

At this point, it is worth-mentioning that the testing of this particular sensor is included to the short-term future work of the project, due to its delivery very close to the submission date of this thesis.

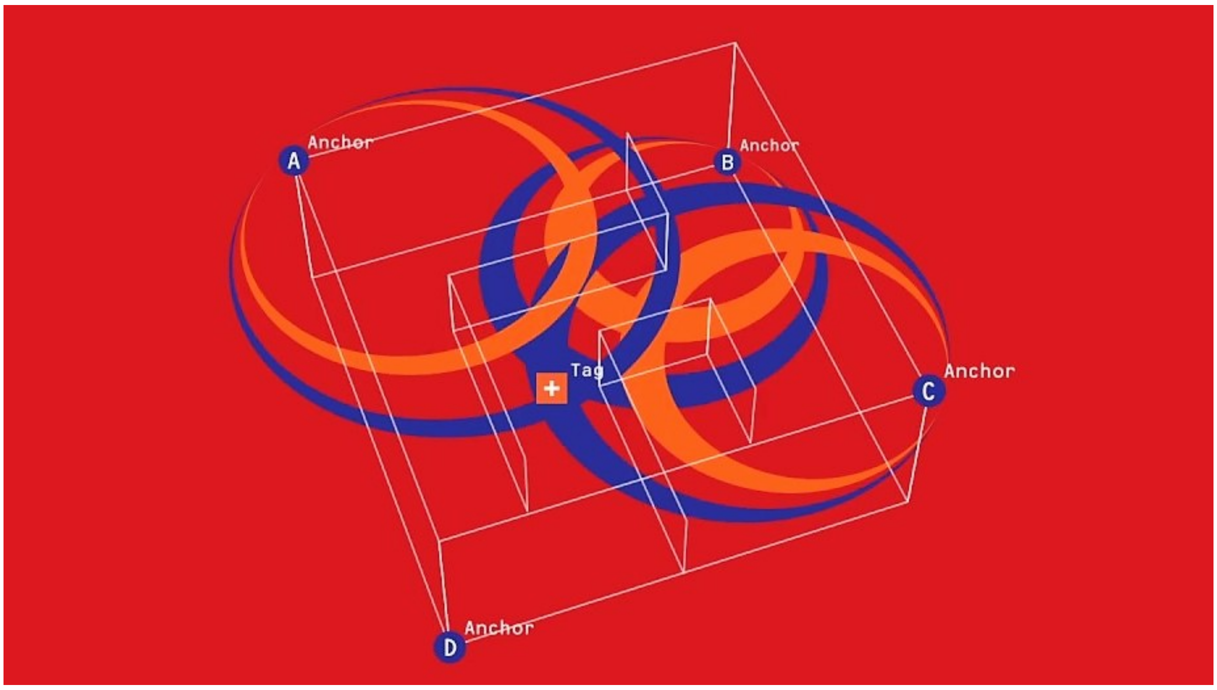


Figure 5.3: KIO RTLS Localisation Method

5.2 UAV Platform

The chosen UAV platform for the given operation of the project is the DJI Matrice 100. However, due to several drawbacks with compatibility of other sensors, the DJI Matrice 100 had been modified with components that will be listed and described as follows. The Figure displays the Matrice 100 platform as it is made from the company.

Figure 5.4: DJI Matrice 100 UAV Platform [11]

The components equipped on-board the UAV are listed below:

- Raspberry Pi 3 as processor of the drone
- Pixhawk Flight Controller as the autopilot
- Ultrasonic Sensor for the altitude control
- Ultraviolet light for the inspection operation
- KIO RTLS Tag for the localisation of the drone
- RC Receiver for emergency piloting over RC Controller

The **Raspberry Pi 3** constitutes the main processor of the drone that runs the Operating System of Linux Ubuntu Mate. It provides with a processing power of 1.2 GHz per core, while its architecture contains a Quad-Core processor with a RAM of 1Gb. This processor coordinates the positioning data towards the autopilot, while it receives the captured images from the image processing algorithms and sends them to the ground computer for post-processing.

THe **Pixhawk** autopilot includes one ST Micro L3GD20H 16 bit gyroscope, one ST Micro LSM303D 14 bit accelerometer / magnetometer, one Invensense MPU 6000 3-axis accelerometer/gyroscope and one MEAS MS5611 barometer. The IMU provides full real-time data with the status of the drone in flight, that are send back to the controller of the UAV in order to maintain the correct tuning via the feedback system.

The rest of the sensors provide supplementary information regarding the position of the drone in the vertical axis, the ultrasonic sensor, or the continuous position of the drone in the 3D plane, from the KIO RTLS. The fusion of the altitude sensors feedback data and the RTLS data provide a cross-checking for optimising the altitude control of the drone. The following Figure displays the final version of the drone during the calibration process of the controller, stated in the thesis of F. Plumacker [41].

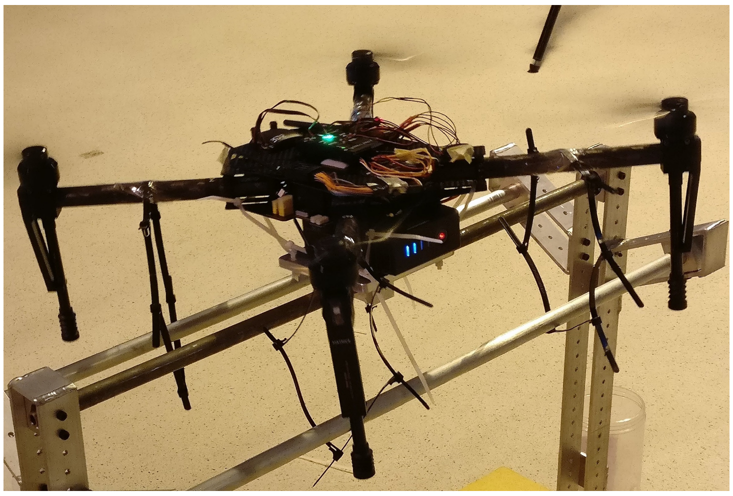


Figure 5.5: Final Version of UAV Platform during Controller Calibration Process

5.3 Robot Operating System - ROS

The Robot Operating System (ROS) is one of the most widespread meta-operating systems for developing robotic software applications. This OS provides the developer with an essential tool, called Hardware Abstraction Layer (HAL), that enables the user to access the hardware of the robot by commands. Additionally, ROS enables the communication among nodes and services of the robot or drone. In particular, with the use of its messaging system, the developed algorithm can subscribe or publish information from or to a specific topic and enable the real-time data transferring among the services of the drone. The administration of these nodes and services maintains the ROS Master, which is a central system that allows to the nodes of the system to communicate each other.

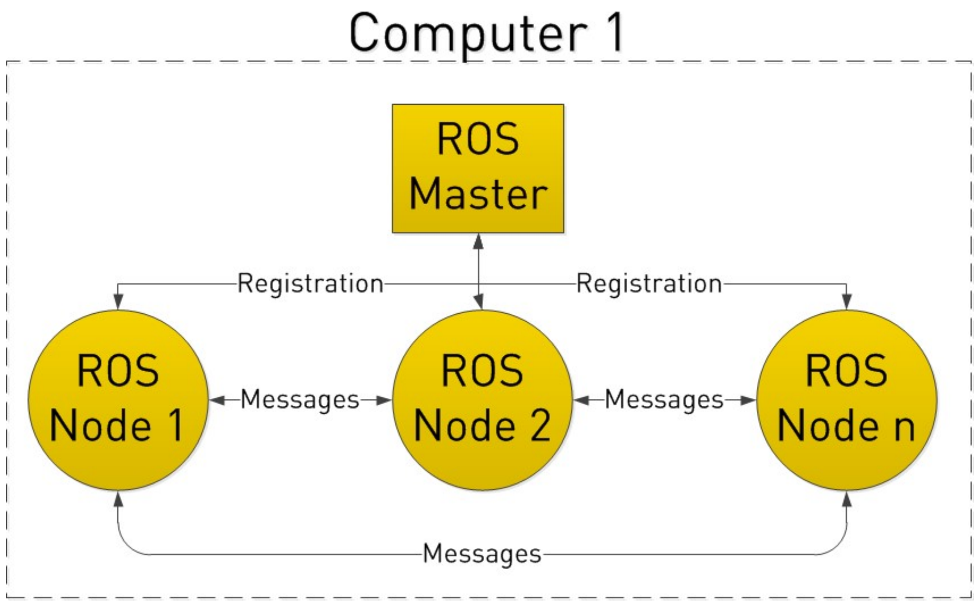


Figure 5.6: Robot Operating System - Nodes Inter-Communication

Hence, in the particular project the communication of the positioning system directly to the autopilot, enables the handshake of valuable information for feedback control with no delay. Additionally, providing access to the image output of the camera, the software script of the image processing is able to obtain the captured image and process it directly, while it receives real-time stamp of the position of the drone; this enables the image processing algorithm to identify the position of the captured image in relation to the wing panel. In the appendix A.2.2 and A.2.1, the Subscriber and Publisher script concerning the positioning data acquiring from the RTLS sensor are stated. The listener script subscribes to the relevant topic of the RTLS sensor in order to obtain the data points in string and point version. The talker scripts is responsible for publishing these data towards the system of the drone.

The essential benefit that the ROS provides is the rapid interaction among the nodes of the drone and the transmission of data among several services.

Chapter 6

Discussion

This study set out to access the feasibility of automatising the process of Airbus Penetrant Testing to aircraft wing panels. Specifically, the novelty of the project was to replace the old-fashioned method of inspecting the wing panel, with the presence of human factor, with a fully autonomous process using the technology of UAVs. The importance of this novelty would focus on a significant operating time reduction, since the 1 to 4 hours inspection by technicians was supposed to substantially decreased to an estimate of 15 minutes with the use of an autonomous UAV, while an essential annual cost reduction provides with a strong motivation to complete this project.

The main findings of the thesis is the feasibility of performing such delicate operation with the use of a UAV, along with the technical capability of detecting efficiently potential defects with the use of advanced image processing algorithms. Hence, the results stated in the previous chapter, concerning the image processing development, illustrate for the first time the capability of automatically detecting the defects on the wing panel for an inspection operation. Specifically, the image processing software package is capable of detecting efficiently even small defects, while a threshold can be adjusted to the standards of experts in order to ignore too small features that might be detected. A simple classification technique, identifies the size and relative position of the detected defects. This constitutes a task of utmost importance, which enables the technician to record the tracked defects and categorise them by size. Additionally, a translation of the relative position to a universal coordinate system, in relation to the wing panel size, provides the technician with the benefit of knowing in physical location the detected defect. Hence, these features improve not only inspection process due to the autonomous and automatic characteristic of the process, but facilitate the technician by providing essential tools that can build an efficient report of every inspected wing panel. Moreover, the software provides with the ability of review again the inspected wing panel after the completion of the process. With this feature, the report and inspection of the wing panel maintains enhanced reliability due to more cautious classification and evaluation of the defects.

The option of the RTLS sensor benefited the project by providing a decent accuracy of indoor location. The provided compatibility with the ROS, along with the decision of using this meta-operating system, enabled the system to continuously communicate with the localisation sensor, thence, obtain positioning stamps in real-time. Thus, the KIO RTLS sensor constituted an valuable option in terms of accuracy, compatibility and normal cost.

Unfortunately, this study maintained numerous limitations in terms of experimenting feasibility. The decreased number of results are caused by the absence of significant to the operation components. The RTLS localisation sensor was accessible to be tested a few days prior to the thesis submission, something that made the implementation to the drone system and the installation impossible. Additionally, the Ultraviolet light delayed considerably, in a level of receiving it the previous week of the thesis submission. Due to these delays, experiments with the UAV in the laboratory of Cranfield University were unable to be performed, which constitutes the dominant reason of the absence of results in the particular task of the project.

The findings of this project suggest that the approach of autonomous operation of this non-destructive testing would also be beneficial in other sectors of the manufacturing procedure. Namely, the maintenance sector can be taken advantage of such operation, by enabling the autonomous inspection of an aircraft-under-maintenance. The UAV equipped with a fully autonomous navigation and inspection system, can enhance the maintenance procedure with respect to time decrease and overall cost reduce. Additionally, the difficult operation of inspecting an aircraft in its upper levels, can easily be performed by this autonomous UAV.

Although this study conducted in the region of aircraft wing manufacturing, several studies and projects have researched and developed similar inspection packages concerning wing turbine rotor blades inspection or buildings and antennas. Hence, the idea of autonomous inspection with the use of UAV provides with several benefits worth investing for.

Last but not least, several question remained to be solved; the performance of the drone during the inspection process is something that demands testing, while a better estimation of the time of inspection with the use of the autonomous UAS needs for be determined. More research to this area is necessary before the industrial application of this project, yet the starting results show the technical feasibility of implementing such automation.

Chapter 7

Conclusions & Future Work

In this chapter, a brief reference to potential conclusions of the project will be listed, while a future work will also be stated as follows.

7.1 Conclusions

- The feasibility of performing autonomously the inspection during the penetrant testing is the dominant objective of this thesis. The results of the image processing along with the specification provided by the RTLS sensor illustrated the capability of performing such operation with an autonomous UAV.
- The detection of defects with the use of several image and video analysis algorithms were succeeded, while the classification of the detected defects consisted an important feature.
- The Real-Time Location System consisted a decent choice in relation to the provided accuracy and cost.
- The evaluation of the RTLS system was a limitation for this thesis project, since the delay in delivery.
- The Non-Destructive Testing technique, the Penetrant Flow Detection, displays valuable results in detecting potential defects on an aircraft wing panel.

- The efficiency of fusing several image analysis filters and detectors, provided great results concerning the correct and automatic detection of the defects.

- The size and position determination of the detected defects constitutes a beneficial tool for the improvement of the classification technique along with the facilitation of the operators/technicians.

- The effectiveness of the Penetrant Testing simulation, in comparison with the industrial implementation, differs significantly in the Developer application, due to the glare that it is created.

7.2 Future Work

This project's work maintained several limitation due to delivering times of significant components, such as the Real-Time Location System and the Ultraviolet light. Hence, essential tasks for the evaluation of the project's performance should be included to the future work section.

The experimental part of the project, which includes operational flights with the use of the UAV in order to evaluate the performance of the drone during the flight, is one of the tasks that should take place in the next steps. Additionally, the image processing algorithm should be evaluated in concern to its performance, with the support of expert to these operations. Important task for the improvement of the project is the automatic classification of the defects based on their size and significance. Part of the classification, the size calculation has been done, yet the completion of the fully automatic classification and the discard of non-defects are of utmost importance for the advance of the project. Furthermore, the implementation of the localisation sensor on-board the drone has to be implemented in order to be ready for a demonstration flight. Last but not least, the most essential future work concerning this project is the completion of the package. The project should perform complete autonomous mission of inspection around the wing panel, inspect the panel in order to detect potential defects and export the to a 3D wing model for illustrating them in relation to their size and position.

This future work maintains short-term characteristics as the Airbus Group has strong interest for the completion of this project.

Appendix A

Appendix

A.1 Developed Code

A.1.1 Video Processing Algorithm

```
from __future__ import print_function
from scipy.spatial import distance as dist
from imutils import perspective
from imutils import contours
from imutils import paths
from matplotlib import pyplot as plt

from std_msgs.msg import String
import rospy

from PIL import Image, ImageDraw
from PIL import ImageFilter

from scipy import signal
from scipy import ndimage

from skimage import feature, filter
from skimage.filter import threshold_otsu
from skimage.color import rgb2gray

import numpy as np
import argparse
import imutils
import cv2
import sys, os, time
from math import tan


class VideoProcessing():

        def __init__( self ):

                # construct the argument parse and parse the arguments
                self.ap = argparse.ArgumentParser()
                self.ap.add_argument("-i", "--image", required=False,
                self.ap.add_argument("-w", "--width", type=float, default=0.5,
                self.ap.add_argument("-t", "--threshold", type=float, default=50,
                self.args = vars(self.ap.parse_args())

                # subscribe to topics of position to get the location values to the
                    script
                self.tagX = rospy.Subscriber( "/tag_x", String, self.tag_x )
                self.tagY = rospy.Subscriber( "/tag_y", String, self.tag_y )
                self.tagZ = rospy.Subscriber( "/tag_z", String, self.tag_z )


                # initialisation of wing panel
                self.panel_length = 660
                self.panel_height = 240

                self.photos = []

                self.frames_number = 0

                self.cap = cv2.VideoCapture("cam_2.avi")
                # self.cap = cv2.VideoCapture(0)

                self.core()

        # receive position x - function
        def tag_x(self, data):
                info = float(data.data)
                self.tagx = info

        # receive position x - function
        def tag_y(self, data):
                info = float(data.data)
                self.tagy = info

        # receive position x - function
        def tag_z(self, data):
                info = float(data.data)
                self.tagz = info

        def adjust_gamma(self, image, gamma=1.0):
                # build a lookup table mapping the pixel values [0, 255] to
                # their adjusted gamma values
                self.invGamma = 1.0 / self.gamma
                self.table = np.array([((i / 255.0) ** self.invGamma) * 255
                for i in np.arange(0, 256)]).astype("uint8")

                # apply gamma correction using the lookup table
                return cv2.LUT(self.image, self.table)

        def midpoint(self, ptA, ptB):
                return ((ptA[0] + ptB[0]) * 0.5, (ptA[1] + ptB[1]) * 0.5)

        def variance_of_laplacian(self, image):
                # compute the Laplacian of the image and then return the focus
                # measure, which is simply the variance of the Laplacian
                return cv2.Laplacian(self.image, cv2.CV_64F).var()

        def fov_calculation(self, aov, distance):
                # calculates the field of view (the width of the image), based on the
                # angle of view and the distance of the UAV from the panel.
                # Variables: 1) aov in [degrees], 2) distance in [cm]
                self.fov = 2 * self.distance * tan(self.aov/2)
                return self.fov



        def core(self):

                while(self.cap.isOpened()):

                        self.position = [self.tagx, self.tagy, self.tagz]

                        self.ret, self.image = self.cap.read()

                        if self.args.get("video") and not self.ret:
                                break

                        cv2.imwrite("frame.jpg", self.image)
                        self.img = cv2.imread("frame.jpg",0)

                        # cv2.imshow("Images", image)
                        # quit = cv2.waitKey(0)
                        # if quit == "q":
                        #       QuitProgram()


                        # # load the image and resize it
                        # self.img = Image.open(self.image)
                        # self.width, self.height = self.img.size

                        if cv2.countNonZero(self.img) == 0:
                                break
                        else:
                                self.gray = cv2.cvtColor(self.image, cv2.COLOR_BGR2GRAY)

                        # compute the focus measure of the image, using the Variance of the Laplacian
                        self.focus_measure = self.variance_of_laplacian(self.gray)

                        # check whether the image is blurry or not and decide to continue
                        # if focus_measure < args["threshold"]:
                        #       print("Blur Scale:", focus_measure, "in comparison to the threshold value:", args["threshold"])
                        #       print("Blurry Image: Cannot Processed")
                        #       # break
                        # else:
                        #       print("Blur Scale:", focus_measure, "in comparison to the threshold value:", args["threshold"])

                        self.gamma = 0.5
                        self.adjusted = self.adjust_gamma(self.image, self.gamma)
                        # cv2.putText(adjusted, "g={}".format(0.03), (10, 30),
                        #       cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 3)
                        # cv2.imshow("Images", self.adjusted)
                        # self.quit = cv2.waitKey(0)
                        # if self.quit == "q":
                        #       QuitProgram()

                        # convert it to grayscale, and blur it slightly
                        self.gray = cv2.cvtColor(self.adjusted, cv2.COLOR_BGR2GRAY)
                        self.gray = cv2.GaussianBlur(self.gray, (7, 7), 0)

                        # perform edge detection, then perform a dilation + erosion to
                        # close gaps in between object edges3
                        self.edged = cv2.Canny(self.gray, 50, 50)
                        cv2.imshow("Images", self.edged)
                        self.quit = cv2.waitKey(0)
                        if self.quit == "q":
                                QuitProgram()

                        self.edged = cv2.dilate(self.edged, None, iterations=1)
                        self.edged = cv2.erode(self.edged, None, iterations=1)

                        # find contours in the edge map
                        self.cnts = cv2.findContours(self.edged.copy(), cv2.RETR_EXTERNAL,
                        cv2.CHAIN_APPROX_SIMPLE)
                        self.cnts = self.cnts[0] if imutils.is_cv2() else self.cnts[1]
                        if len(self.cnts) == 0:
                                break
                                print("Error")

                        # sort the contours from left-to-right and initialize the
                        # "pixels per metric" calibration variable
                        (self.cnts, _) = contours.sort_contours(self.cnts)
                        self.pixelsPerMetric = None

                        # calculates the screen width based on the field of view
                        self.aov = 170
                        self.d = 100
                        self.screen_width = self.fov_calculation(self.aov, self.d)

                        self.photos.append(self.edged)

                        # loop over the contours individually
                        for c in self.cnts:
                                # if the contour is not sufficiently large, ignore it; initial: 100
                                if cv2.contourArea(c) < 5:
                                        continue

                                # compute the rotated bounding box of the contour
                                self.orig = self.adjusted.copy() # remove .copy() to illistrate full detected image
                                self.box = cv2.minAreaRect(c)
                                self.box = cv2.cv.BoxPoints(self.box) if imutils.is_cv2() else cv2.boxPoints(self.box)
                                self.box = np.array(self.box, dtype="int")

                                # order the points in the contour such that they appear
                                # in top-left, top-right, bottom-right, and bottom-left
                                # order, then draw the outline of the rotated bounding
                                # box
                                self.box = perspective.order_points(self.box)
                                cv2.drawContours(self.orig, [self.box.astype("int")], -1, (0, 255, 0), 2)

                                # loop over the original points and draw them
                                for (self.x, self.y) in self.box:
                                        cv2.circle(self.orig, (int(self.x), int(self.y)), 5, (0, 0, 255), -1)

                                # unpack the ordered bounding box, then compute the midpoint
                                # between the top-left and top-right coordinates, followed by
                                # the midpoint between bottom-left and bottom-right coordinates cv2.imwrite("adjusted.jpg",adjusted)

                                (self.tl, self.tr, self.br, self.bl) = self.box
                                (self.tltrX, self.tltrY) = self.midpoint(self.tl, self.tr)
                                (self.blbrX, self.blbrY) = self.midpoint(self.bl, self.br)

                                # compute the midpoint between the top-left and top-right points,
                                # followed by the midpoint between the top-righ and bottom-right
                                (self.tlblX, self.tlblY) = self.midpoint(self.tl, self.bl)
                                (self.trbrX, self.trbrY) = self.midpoint(self.tr, self.br)

                                # draw the midpoints on the image
                                cv2.circle(self.orig, (int(self.tltrX), int(self.tltrY)), 5, (255, 0, 0), -1)
                                cv2.circle(self.orig, (int(self.blbrX), int(self.blbrY)), 5, (255, 0, 0), -1)
                                cv2.circle(self.orig, (int(self.tlblX), int(self.tlblY)), 5, (255, 0, 0), -1)
                                cv2.circle(self.orig, (int(self.trbrX), int(self.trbrY)), 5, (255, 0, 0), -1)

                                # draw lines between the midpoints      cv2.imwrite("adjusted.jpg",adjusted)

                                cv2.line(self.orig, (int(self.tltrX), int(self.tltrY)), (int(self.blbrX), int(self.blbrY)),
                                (255, 0, 255), 2)
                                cv2.line(self.orig, (int(self.tlblX), int(self.tlblY)), (int(self.trbrX), int(self.trbrY)),
                                (255, 0, 255), 2)

                                # compute the Euclidean distance between the midpoints
                                self.dA = dist.euclidean((self.tltrX, self.tltrY), (self.blbrX, self.blbrY))
                                self.dB = dist.euclidean((self.tlblX, self.tlblY), (self.trbrX, self.trbrY))

                                # if the pixels per metric has not been initialized, then
                                # compute it as the ratio of pixels to supplied metric
                                # (in this case, inches)
                                if self.pixelsPerMetric is None:
                                        self.pixelsPerMetric = self.dB / self.args["width"]

                                # compute the size of the object
                                self.dimA = self.dA / self.pixelsPerMetric
```

```python
                                self.dimB = self.dB / self.pixelsPerMetric

                                # register the coordinates of the contour
                                self.coordinate_system[0].append(self.x)
                                self.coordinate_system[1].append(self.y)


                                # draw the object sizes on the image
                                cv2.putText(self.orig, "{:.1f}in".format(self.dimA),
                                (int(self.tltrX - 15), int(self.tltrY - 10)), cv2.FONT_HERSHEY_SIMPLEX,
                                0.65, (255, 255, 255), 2)
                                cv2.putText(self.orig, "{:.1f}in".format(self.dimB),
                                (int(self.trbrX + 10), int(self.trbrY)), cv2.FONT_HERSHEY_SIMPLEX,
                                0.65, (255, 255, 255), 2)
                                cv2.putText(self.orig, "x={}".format(self.x), (30, 80),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 3)
                                cv2.putText(self.orig, "y={}".format(self.y), (30, 110),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 3)


                                # # show the output image
                                # cv2.imshow("Image", orig)
                                # # orig = adjusted --> uncomment to illustrate full detected image
                                # cv2.waitKey(0)

                        # print("Number of detected defects: ", len(self.coordinates_f))


                        # calculate the total number of frames
                        self.frames_number += 1

                # store every frame in one array
                self.all_frames = np.array(self.photos)

                # print frames total number
                print(self.frames_number)

                self.cap.release()
                cv2.destroyAllWindows()
                exit()

def main():
        rospy.init_node( "video_processing" )
        try:
                vp = VideoProcessing()
                rospy.spin()
        except KeyboardInterrupt:
                print("Keyboard interrupted")

if __name__ == "__main__":
        main()
```

A.1.2 Image Processing Algorithm

```
from __future__ import print_function
from scipy.spatial import distance as dist
from imutils import perspective
from imutils import contours
from imutils import paths
from matplotlib import pyplot as plt

from PIL import Image, ImageDraw
from PIL import ImageFilter

from scipy import signal
from scipy import ndimage

from skimage import feature, filter
from skimage.filter import threshold_otsu
from skimage.color import rgb2gray

import numpy as np
import argparse
import imutils
import cv2
import sys, os, time

def adjust_gamma(image, gamma=1.0):
        # build a lookup table mapping the pixel values [0, 255] to
        # their adjusted gamma values
        invGamma = 1.0 / gamma
        table = np.array([((i / 255.0) ** invGamma) * 255
                for i in np.arange(0, 256)]).astype("uint8")

        # apply gamma correction using the lookup table
        return cv2.LUT(image, table)

def midpoint(ptA, ptB):
        return ((ptA[0] + ptB[0]) * 0.5, (ptA[1] + ptB[1]) * 0.5)

def variance_of_laplacian(image):
        # compute the Laplacian of the image and then return the focus
        # measure, which is simply the variance of the Laplacian
        return cv2.Laplacian(image, cv2.CV_64F).var()


# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
        help="path to the input image")
ap.add_argument("-w", "--width", type=float, required=True,
        help="width of the left-most object in the image (in inches)")
ap.add_argument("-t", "--threshold", type=float, default=75.0,
        help="focus measures that fall below this value will be considered blurry")
args = vars(ap.parse_args())

# load the image and resize it
image = cv2.imread(args["image"])
image = cv2.resize(image, (1236, 1237))
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# compute the focus measure of the image, using the Variance of the Laplacian
focus_measure = variance_of_laplacian(gray)

# check whether the image is blurry or not and decide to continue
if focus_measure < args["threshold"]:
        print("Blur Scale:", focus_measure, "in comparison to the threshold value:", args
            ["threshold"])
        print("Blurry Image: Cannot Processed")
        exit()
else:
        print("Blur Scale:", focus_measure, "in comparison to the threshold value:", args
            ["threshold"])

adjusted = adjust_gamma(image, 0.02)
cv2.putText(adjusted, "g={}".format(0.03), (10, 30),
        cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 3)
cv2.imshow("Images", adjusted)
quit = cv2.waitKey(0)
if quit == "q":
        QuitProgram()

# convert it to grayscale, and blur it slightly
gray = cv2.cvtColor(adjusted, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (7, 7), 0)

# perform edge detection, then perform a dilation + erosion to
# close gaps in between object edges3
edged = cv2.Canny(gray, 50, 50)
edged = cv2.dilate(edged, None, iterations=1)
edged = cv2.erode(edged, None, iterations=1)

# find contours in the edge map
cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
cnts = cnts[0] if imutils.is_cv2() else cnts[1]

# sort the contours from left-to-right and initialize the
# pixels per metric calibration variable
(cnts, _) = contours.sort_contours(cnts)
pixelsPerMetric = None

coordinate_system = [[],[]]

# loop over the contours individually
for c in cnts:
        # if the contour is not sufficiently large, ignore it; initial: 100
        if cv2.contourArea(c) < 5:
                continue

        # compute the rotated bounding box of the contour
        orig = adjusted.copy() # remove .copy() to illistrate full detected image
        box = cv2.minAreaRect(c)
        box = cv2.cv.BoxPoints(box) if imutils.is_cv2() else cv2.boxPoints(box)
        box = np.array(box, dtype="int")

        # order the points in the contour such that they appear
        # in top-left, top-right, bottom-right, and bottom-left
        # order, then draw the outline of the rotated bounding
        # box
        box = perspective.order_points(box)
        cv2.drawContours(orig, [box.astype("int")], -1, (0, 255, 0), 2)

        # loop over the original points and draw them
        for (x, y) in box:
                cv2.circle(orig, (int(x), int(y)), 5, (0, 0, 255), -1)

        # unpack the ordered bounding box, then compute the midpoint
        # between the top-left and top-right coordinates, followed by
        # the midpoint between bottom-left and bottom-right coordinates cv2.imwrite("
            adjusted.jpg",adjusted)

        (tl, tr, br, bl) = box
        (tltrX, tltrY) = midpoint(tl, tr)
        (blbrX, blbrY) = midpoint(bl, br)

        # compute the midpoint between the top-left and top-right points,
        # followed by the midpoint between the top-righ and bottom-right
        (tlblX, tlblY) = midpoint(tl, bl)
        (trbrX, trbrY) = midpoint(tr, br)

        # draw the midpoints on the image
        cv2.circle(orig, (int(tltrX), int(tltrY)), 5, (255, 0, 0), -1)
        cv2.circle(orig, (int(blbrX), int(blbrY)), 5, (255, 0, 0), -1)
        cv2.circle(orig, (int(tlblX), int(tlblY)), 5, (255, 0, 0), -1)
        cv2.circle(orig, (int(trbrX), int(trbrY)), 5, (255, 0, 0), -1)

        # draw lines between the midpoints      cv2.imwrite("adjusted.jpg",adjusted)

        cv2.line(orig, (int(tltrX), int(tltrY)), (int(blbrX), int(blbrY)),
                (255, 0, 255), 2)
        cv2.line(orig, (int(tlblX), int(tlblY)), (int(trbrX), int(trbrY)),
                (255, 0, 255), 2)

        # compute the Euclidean distance between the midpoints
        dA = dist.euclidean((tltrX, tltrY), (blbrX, blbrY))
        dB = dist.euclidean((tlblX, tlblY), (trbrX, trbrY))

        # if the pixels per metric has not been initialized, then
        # compute it as the ratio of pixels to supplied metric
        # (in this case, inches)
        if pixelsPerMetric is None:
        pixelsPerMetric = dB / args["width"]

        # compute the size of the object
        dimA = dA / pixelsPerMetric
        dimB = dB / pixelsPerMetric

        # register the coordinates of the contour
        coordinate_system[0].append(x)
        coordinate_system[1].append(y)

        # draw the object sizes on the image
        cv2.putText(orig, "{:.1f}in".format(dimA),
                (int(tltrX - 15), int(tltrY - 10)), cv2.FONT_HERSHEY_SIMPLEX,
                0.65, (255, 255, 255), 2)
        cv2.putText(orig, "{:.1f}in".format(dimB),
                (int(trbrX + 10), int(trbrY)), cv2.FONT_HERSHEY_SIMPLEX,
                0.65, (255, 255, 255), 2)
        cv2.putText(orig, "x={}".format(x), (10, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 3)
        cv2.putText(orig, "y={}".format(y), (10, 60),
                cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 3)


        # show the output image
        cv2.imshow("Image", orig)
        # orig = adjusted --> uncomment to illustrate full detected image
        cv2.waitKey(0)

print("Number of detected defects: ", len(coordinate_system[0]))
```

A.2 Positioning Scripts

A.2.1 Publisher Script

```
import roslib
import rospy
from geometry_msgs.msg import Point
from std_msgs.msg import String
import serial

ser = serial.Serial("/dev/ttyACM2", 9600)

def talker():
    pub = rospy.Publisher("kio_rtls_points", String, queue_size=10)
    rospy.init_node("kio_rtls_talker", anonymous=True)
    rate = rospy.Rate(10) # 10Hz

    while not rospy.is_shutdown():

        data= ser.readline()
        if (str(data).startswith("data: TargetNew")):
            data = data.split(" ")
            distance = data[4]

        # rospy.loginfo(data)
        pub.publish(String(data))


if __name__ == "__main__":

    talker()
```

A.2.2 Subscriber & Trilateration Script

```
import sys
import rospy

from std_msgs.msg import String, Int32MultiArray
from numpy import *
from scipy.optimize import *
```

```python

import serial
import numpy as np

import warnings
warnings.filterwarnings("ignore", "The iteration is not making good progress")

class Trilateration():

    def __init__( self ):

        # rospy.init_node("kio_rtls_listener", anonymous=True)

        rospy.Subscriber("/kio_rtls_points", String, self.callback)
        self.talk_x = rospy.Publisher("tag_x", String, queue_size=10)
        self.talk_y = rospy.Publisher("tag_y", String, queue_size=10)
        self.talk_z = rospy.Publisher("tag_z", String, queue_size=10)

        self.F = empty((3))
        self.initial = array([1,1,1])

        self.xan = [1.24, 14.73, 14.72, 1.250]
        self.yan = [16.15, 16.25, -1.225, -1.24]
        self.zan = [3.1, 0.3, 3.08, 0.34]

        self.ser = serial.Serial(
            port="/dev/ttyUSB0",
            baudrate=57600,
        )

    def callback(self, data):

        # rospy.loginfo(data.data)
        info = data.data
        info = info.split(" ")

        # define anchor IDs and acquire distances
        #------------------------ Anchor A --------------------------------------#
        anchor_a = info[1]
        self.distance_A = float(info[2])
        #------------------------ Anchor B --------------------------------------#
        anchor_b = info[3]
        self.distance_B = float(info[4])
        #------------------------ Anchor C ---------------------------------------#
        # anchor_c = info[5]
        # self.distance_C = float(info[6])
        # #------------------------ Anchor D ---------------------------------------#
        anchor_d = info[7]
        self.distance_D = float(info[8])

        # distance = [self.distance_A,self.distance_B,self.distance_C,self.distance_D]

        self.coordinates = fsolve(self.solver,self.initial)
        print(self.coordinates)

        self.talk_x.publish(str(self.coordinates[0]))
        self.talk_y.publish(str(self.coordinates[1]))
        self.talk_z.publish(str(self.coordinates[2]))

        self.serial_pub()


    def serial_pub(self):

        if self.ser.isOpen():
            self.ser.write(str(self.coordinates))
            # self.ser.write("6")

    def solver(self, coordinates):
        x = coordinates[0]
        y = coordinates[1]
        z = coordinates[2]

        # distance = callback()
        self.F[0] = pow((x - self.xan[0]),2) + pow((y - self.yan[0]),2) + pow((z - self.
            zan[0]),2) - self.distance_A
        self.F[1] = pow((x - self.xan[1]),2) + pow((y - self.yan[1]),2) + pow((z - self.
            zan[1]),2) - self.distance_B
        self.F[2] = pow((x - self.xan[2]),2) + pow((y - self.yan[2]),2) + pow((z - self.
            zan[2]),2) - self.distance_D

        return self.F

def main():
        rospy.init_node( "drone_position" )
        af = Trilateration()
        try:
                rospy.spin()
        except KeyboardInterrupt:
                print "Keyboard interrupted"
                # af.usb_service()

if __name__ == "__main__":
        main()
```

A.3 Penetrant Testing

Figure A.1: Fluorescent Penetrant Material

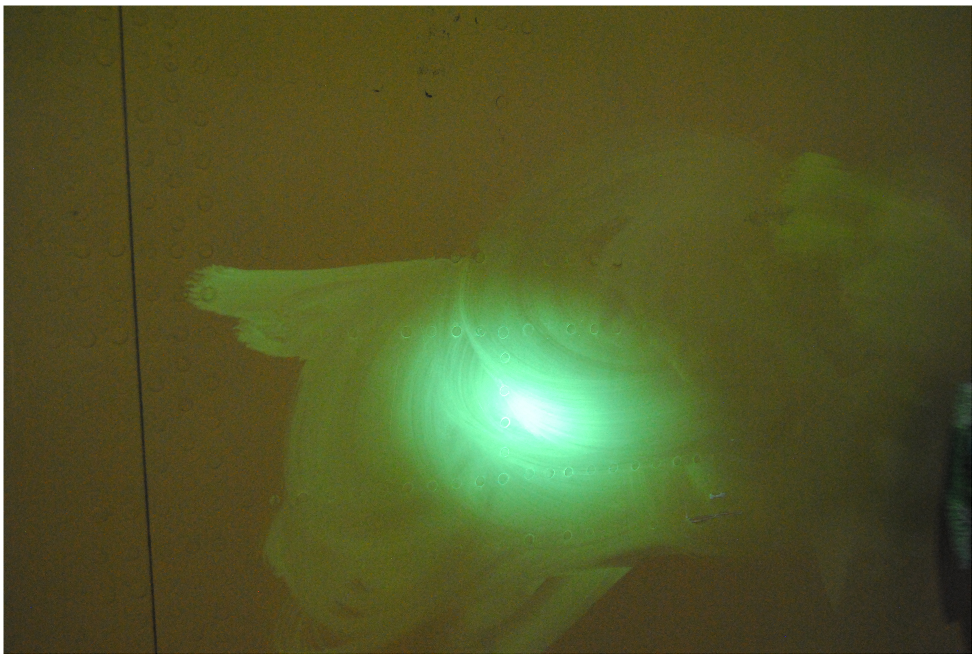


Figure A.2: Penetrant Application Stage

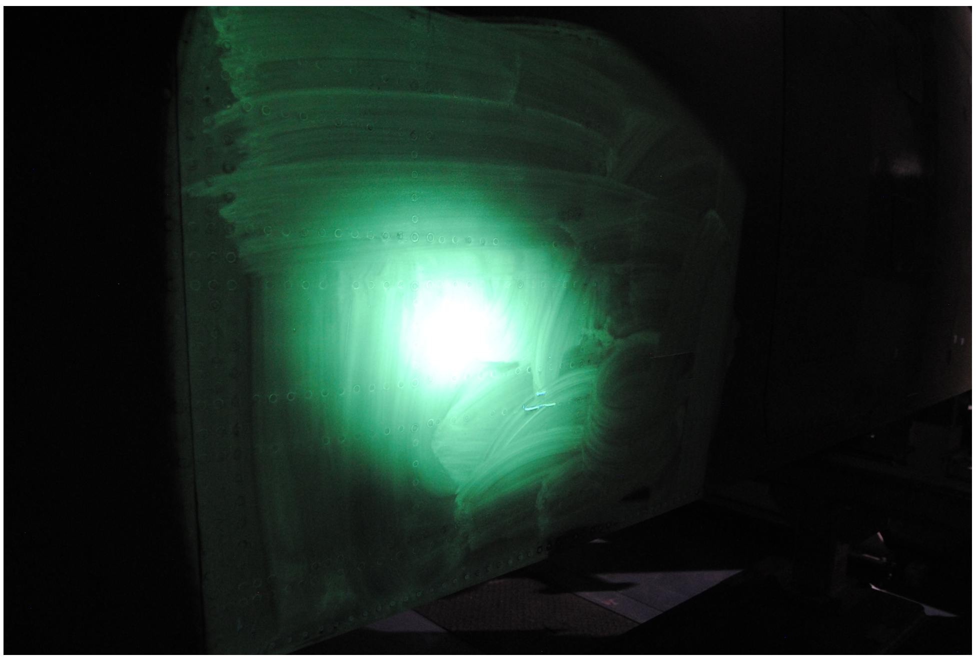


Figure A.3: Penetrant Application Stage - Under UV Light

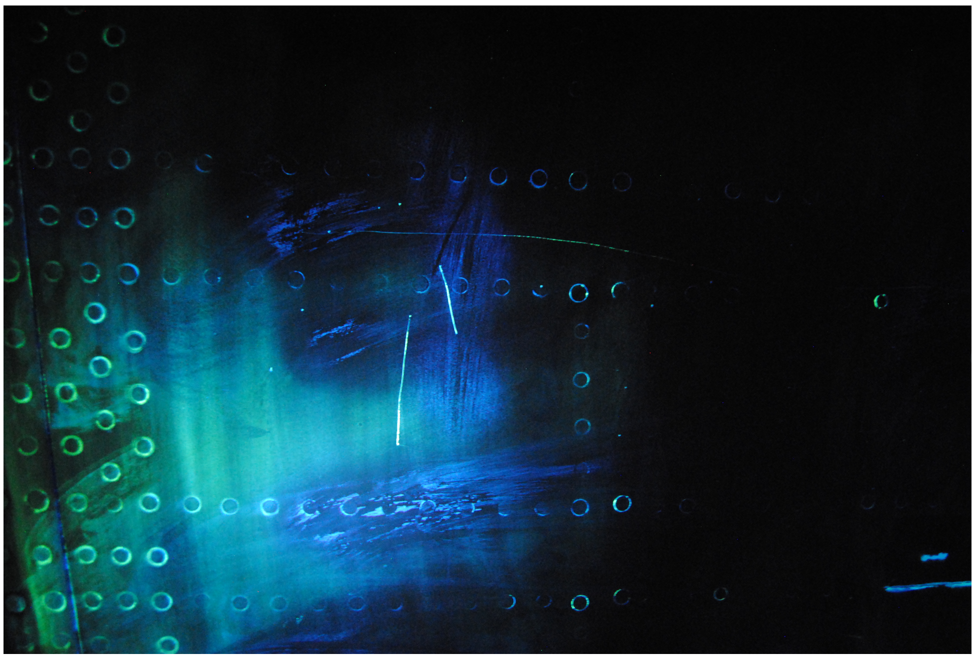


Figure A.4: Inspection Sample Photograph - Detected Defects

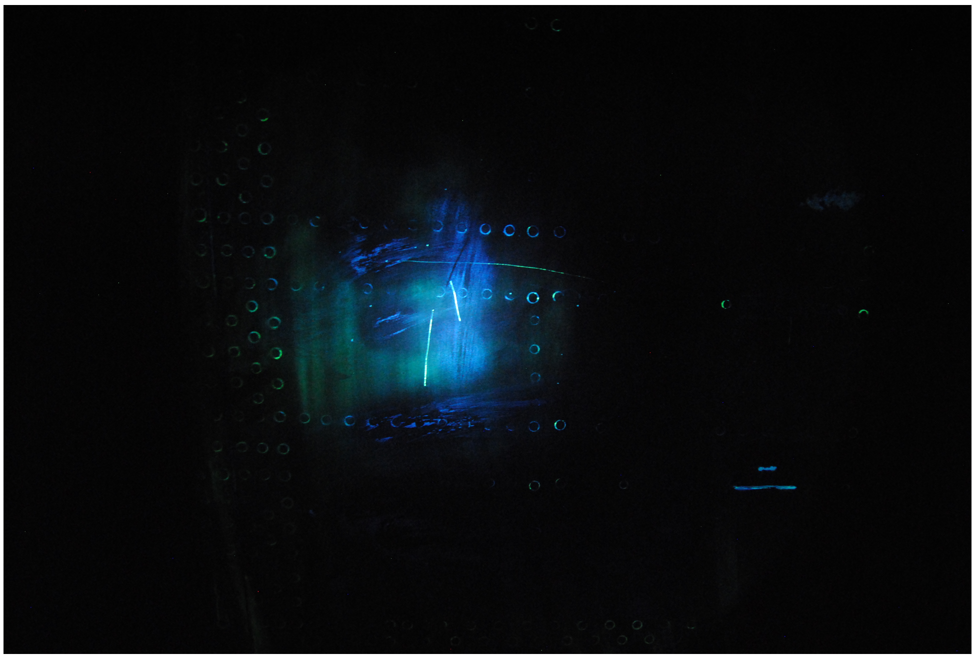


Figure A.5: Inspection Sample Photograph - Detected Defects

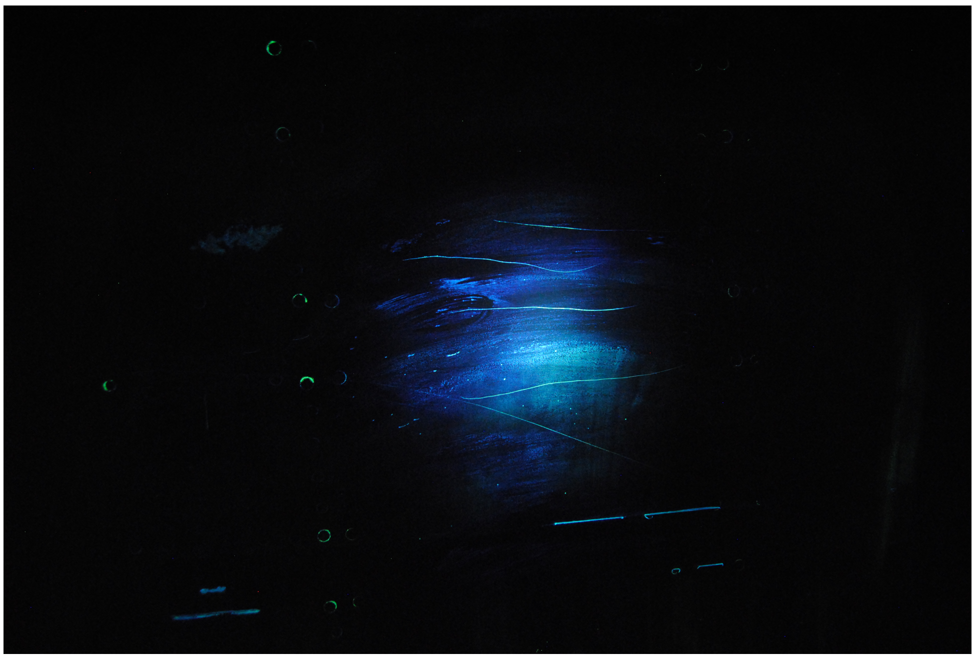


Figure A.6: Inspection Sample Photograph - Detected Defects

Bibliography

[1] PCWorld. (2015, Jun) Easyjet testing aircraft inspection by drone. [Online]. Available: http://www.pcworld.com/video/53587/easyjet-testing-aircraft-inspection-by-drone

[2] E. Tehnoloogia. (2017, Aug) eliko.ee. [Online]. Available: http://www.eliko.ee

[3] M. Laaraiedh, S. Avrillon, and B. Uguen, "Lower bounds for non-hybrid and hybrid localisation techniques in wireless networks," *Trans. Emerging Telecommunications Technologies*, vol. 23, pp. 268–280, 2012.

[4] A. Giretti, A. Carbonari, and M. Vaccarini, "Ultra wide band positioning systems for advanced construction site management," in *New Approach of Indoor and Outdoor Localization Systems*, F. B. Elbahhar and A. Rivenq, Eds. Rijeka: InTech, 2012, ch. 05. [Online]. Available: http://dx.doi.org/10.5772/48260

[5] D. Sobers and E. Johnson, "Indoor Navigation for Micro Air Vehicles," *AIAA Infotech@Aerospace 2010*, no. April, pp. 1–33, 2010. [Online]. Available: http://arc.aiaa.org/doi/abs/10.2514/6.2010-3401

[6] M. Stokkeland, K. Klausen, and T. A. Johansen, "Autonomous visual navigation of Unmanned Aerial Vehicle for wind turbine inspection," *2015 International Conference on Unmanned Aircraft Systems, ICUAS 2015*, pp. 998–1007, 2015.

[7] J. Nikolic, "A uav system for inspection of industrial facilities," ETH Zurich, Autonomous Systems Lab, IEEE Paper, 2012.

[8] F. SA. (2017) Elios. [Online]. Available: http://www.flyability.com/elios/

[9] mechanicalintegrity.com. (2017, Sug) Liquid penetrant testing and inspection. [Online]. Available: http://www.mechanicalintegrity.com/LPT.html

[10] N. R. Centre. (2017, Aug) Liquid penetrant testing. [Online]. Available: https://www.nde-ed.org/EducationResources/CommunityCollege/PenetrantTest/Principles/liquidpi.htm

[11] DJI. (2017, Aug) Dji matrice 100. [Online]. Available: https://www.dji.com/matrice100

[12] E. Schnipke, S. Reidling, J. Meiring, W. Jeffers, M. Hashemi, R. Tan, A. Nemati, and M. Kumar, "Autonomous Navigation of UAV through GPS-Denied Indoor Environment with Obstacles," *AIAA Infotech @ Aerospace*, no. January, pp. 1–7, 2015. [Online]. Available: http://arc.aiaa.org/doi/abs/10.2514/6.2015-0715

[13] D. Martin, "Basic lighting techniques for machine vision," Certified Vision Professional.

[14] I. Jahr, *Lighting in Machine Vision*. Wiley-VCH Verlag GmbH and Co. KGaA, 2007, pp. 73–203.

[15] L. I. D. N. G. Angeletti, J. R. Pereira Valente, Ed., *Autonomous Indoor Hovering with a Quadrotor*, ser. Workshop Proceedings of SIMPAR 2008, Sapienza University of Rome, Dept. of Computer and System Science. Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS, Nov 2008.

[16] A. G. Bachrach, "Autonomous flight in unstructured and unknown indoor environments," MSc Thesis, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, Department ofElectrical Engineering and Computer Science, Sep 2009.

[17] A. Bachrach, "Autonomous Flight in Unstructured and Unknown Indoor Environments," *International Journal of Micro Air Vehicles*, vol. 1, no. 4, p. 126, 2009. [Online]. Available: http://dspace.mit.edu/bitstream/handle/1721.1/54222/599812275.pdf?sequence=1

[18] H. Choi, S. Son, J. Kim, and Y. Baek, "Rf-based indoor locating system for nlos environment," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. IEEE, 2010, pp. 628–633.

[19] P. Piotrowski, S. Kubal, M. Kowal, and R. J. Zielinski, "Algorithm for signal detection in time based distance measurements," in *2010 Fifth International Conference on Broadband and Biomedical Communications*, Dec 2010, pp. 1–5.

[20] B. K. Sagar Patil, "Rf-based indoor asset tracking," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 5, no. 4, pp. 933–935, Apr 2016.

[21] R. Mautz, "Indoor Positioning Technologies," *Institute of Geodesy and Photogrammetry, Department of Civil, Environmental and Geomatic Engineering, ETH Zurich*, no. February 2012, p. 127, 2012. [Online]. Available: http://e-collection.library.ethz.ch/eserv/eth:5659/eth-5659-01.pdf

[22] M. Segura, H. Hashemi, C. Sisterna, and V. Mut, "Experimental demonstration of self-localized ultra wideband indoor mobile robot navigation system," in *2010 International Conference on Indoor Positioning and Indoor Navigation*, Sept 2010, pp. 1–9.

[23] B.-S. Cho, W.-s. Moon, W.-J. Seo, and K.-R. Baek, "A dead reckoning localization system for mobile robots using inertial sensors and wheel revolution encoding," *Journal of Mechanical Science and Technology*, vol. 25, no. 11, pp. 2907–2917, Nov 2011. [Online]. Available: https://doi.org/10.1007/s12206-011-0805-1

[24] G. Szafranski, R. Czyba, W. Janusz, and W. Blotnicki, "Altitude estimation for the uav's applications based on sensors fusion algorithm," in *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, May 2013, pp. 508–515.

[25] J. Liu and R. Jain, "Survey of Wireless Based Indoor Localization Technologies," *Washington University in St. Louis*, pp. 1–17, 2014.

[26] A. Papapostolou and H. Chaouchi, “Integrating rfid and wlan for indoor positioning and ip movement detection,” *Wireless Networks*, vol. 18, no. 7, pp. 861–879, 2012. [Online]. Available: http://dx.doi.org/10.1007/s11276-012-0439-y

[27] C. Eschmann, C.-M. Kuo, C.-H. Kuo, and C. Boller, “High-Resolution Multisensor Infrastructure Inspection With Unmanned Aircraft Systems,” *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XL-1/W2, no. September, pp. 125–129, 2013. [Online]. Available: http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XL-1-W2/125/2013/

[28] J. Civera, A. J. Davison, and J. M. M. Montiel, “Inverse Depth Parameterization for Monocular {SLAM},” *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 932–945, 2008.

[29] R. A. Peters, “A new algorithm for image noise reduction using mathematical morphology,” *IEEE Transactions on Image Processing*, vol. 4, no. 5, pp. 554–568, May 1995.

[30] Y. Zhu and C. Huang, “An improved median filtering algorithm for image noise reduction,” *Physics Procedia*, vol. 25, pp. 609 – 616, 2012, international Conference on Solid State Devices and Materials Science, April 1-2, 2012, Macao. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1875389212005494

[31] Q. Chen, Q. sen Sun, P. A. Heng, and D. shen Xia, “A double-threshold image binarization method based on edge detector,” *Pattern Recognition*, vol. 41, no. 4, pp. 1254 – 1267, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S003132030700413X

[32] H. Zhang and J. Jackman, “Feasibility of Automatic Detection of Surface Cracks in Wind Turbine Blades,” *Wind Engineering*, vol. 38, no. 6, pp. 575–586, 2014.

[33] J. Canny, “A computation approach to edge detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679 – 698, Nov 1986.

[34] I. F. Mondragon, P. Campoy, J. F. Correa, and L. Mejias, “Visual model feature tracking for UAV control,” *2007 IEEE International Symposium on Intelligent Signal Processing, WISP*, 2007.

[35] I. Systems, “Object Modeling by Reg strat ion of Multiple Range Images,” no. April, pp. 2724–2729, 1991.

[36] J. Tang, Y. Chen, A. Jaakkola, J. Liu, J. Hyypp??, and H. Hyypp??, “NAVIS-An UGV indoor positioning system using laser scan matching for large-area real-time applications,” *Sensors (Switzerland)*, vol. 14, no. 7, pp. 11 805–11 824, 2014.

[37] B. R. Ludovic Apvrille, Jean-Luc Dugelay, “Indoor autonomous navigation of low-cost mavs using landmarks and 3d perception,” Institut Mines-Telecom Telecom ParisTech, EURECOM, Tech. Rep.

[38] P. SA. (2017) Parrot ar drone 2.0. [Online]. Available: https://www.parrot.com/uk/drones/parrot-ardrone-20-elite-edition

[39] S. Pertuz, D. Puig, and M. A. Garcia, “Analysis of focus measure operators for shape-from-focus,” *Pattern Recognition*, vol. 46, no. 5, pp. 1415 – 1432, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320312004736

[40] J. L. Pech-Pacheco, G. Cristóbal, J. Chamorro-Martínez, and J. Fernández-Valdivia, “Diatom autofocusing in brightfield microscopy: A comparative study,” in *Proceedings of the International Conference on Pattern Recognition - Volume 3*, ser. ICPR ’00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 3318–. [Online]. Available: http://dl.acm.org/citation.cfm?id=876865.877098

[41] F. Plumacker, “Autonomous indoor inspection of aircraft wing panels using uav technology: Control and path following,” MSc Thesis, Cranfield University, Aug 2017.