University of Southern Denmark

Autumn 2020

Experts in Teams Innovation

Fence Inspection for Lorenz Technology

(Group 175)

Kenni Nilsson
kenil16@student.sdu.dk

Marcus Enghoff Hesselberg Grove
mgrov16@student.sdu.dk

Nikolaj Frederik Pihl Thomsen
nikso16@student.sdu.dk

Muhammad Owais Mehmood
mumeh19@student.sdu.dk

Associate Professor: Ulrik Pagh Schultz
E-mail: ups@mmmi.sdu.dk

Abstract

In this paper an automated drone solution for fence inspection was developed. This system will contribute to maintenance and heightened security of different high-end security places e.g. an airport. A test of the different sensors has been completed, to find the sensor most suitable for the task. Furthermore, an automated route planning algorithm has been developed with easy tuning for the end-user. This will help the end-user setup the drone for their companies specific needs. A deep analysis of different vision algorithms has been conducted to achieve the best possible outcome with data from the chosen sensor.

Moreover, Convolution Neural Network (CNN) has been developed and trained to locate breaches in the grid structure of the fences. This has been done using mask-rcnn. The network was trained with an augmented dataset of almost 7000 images which consisted of artificial snow, rain, fog and other kinds of noise in the images to replicate real-life scenarios. This yielded good results with an mean average precision (mAP) of 0.874 in the augmented test set of 1395 images as well a correct detection of all custom made breaches in the final acceptance test.

Contents

1 Introduction

This rapport will focus on the technical aspect of Expert in teams, based on our team's innovative idea. Furthermore, this case was given to us by Lorenz Technology, whom wishes an automated solution for fence inspection. This automated solution should be able to help e.g. airport security with maintenance as well as heightened security by inspecting their fence for breaches. The current workflow of this is a two person job and requires a car. These two will drive around the perimeter while one person inspects the fences for breaches. Not only is this a very tiresome job, it is also highly inefficient and contributes to pollution. Therefore, it is abundantly clear that drone solution with high automated capabilities would be desired. A solution such as this will contribute economically both for Lorenz Technology as well as their end-user, which in this case is the airport.

2 Project Requirements

This section describes the requirements for the final product of the autonomous detection of breaches in fences using a drone. However, only a prototype of this product will be created in this project with limitations of the design requirements for the final product. The parts of the requirements which are found feasible to incorporate into the prototype are stated below and an analysis of future work that can benefit this solution. This can be seen in section 9.

The vision algorithm should be able to detect a breach in the fence larger than $40x40$ cm as well as note the GPS coordinates of it for further use from the end-user.

When initialised, the drone should be able to fly autonomously without any human intervention. The result from such a flight needs to be analysed through the vision algorithm and return a result within the same day. An easy and short workflow/tutorial is made for the end-user.

The fence breach detection algorithm and the autonomous flight software are the two most important aspects for evaluating the effectiveness of this solution. The project is deemed a success if the drone is able to fly autonomously and capture data of the fence as well as having a vision algorithm capable of detecting breaches 90% of the time, in a tuned environment.

Defining the final acceptance test:
To test the final developed system with the different aspects of this project integrated together, a final acceptance test is defined here and is how we will verify that the solution is durable. This test entails an automated path that the drone need to follow, while maintaining a desired distance from the fence and a stable ground height. This is done to ensure repeatability of the data while inspecting the fence. Furthermore, this will ensure that the whole fence is visible by the mounted sensor and that the data between different flight are as closely related to each other as possible. Obviously, not taking weather conditions into account. This will make the algorithm to detect a breach more stable. The automated path is defined by two set of GPS coordinates, manually taken at the fence which will be turned into a short mission, where a limitation to the drone speed is set to $1m/s$ to ensure a stable video by reducing the blur and more time for the pilot to react if a failure is to occur. The sensor on the drone will start capturing data as soon as the drone reaches its initial waypoint and begins the planned mission.

While the drone is flying the planned mission, the mounted sensor should capture data of the fence for offline analysis of a breach.

This whole process is monitored by a member of the team acting as the pilot, ready to take over if an error were to occur. The pilot should be able to fly the drone, then via the controller swap flight mode into mission and the drone should be able to follow the uploaded mission. Furthermore, he should also be able to start/end the data capture via the controller, to make multiple tests in a row, either automated or manually. The result of this can be seen in section 8.

3 Agile Development Framework

We decided to adopt Scrumban via JIRA as the Agile development framework for the prototype development. Scrumban is a very effective tool for team collaboration and development of products with complex requirements. This framework ensures that we continue to deliver value and meet goals throughout the development period allocated for the prototype. It also helps us to get feedback on the completed deliverables and incorporate it in the design of the prototype.
We are utilising the scrum daily meeting every time we meet to discuss progress, potential issues and how to proceed.

4 Development schedule

In Figure 1, a Gantt chart of our task management and their associated time periods are shown. However, this is an initial schedule developed in the start of the project.

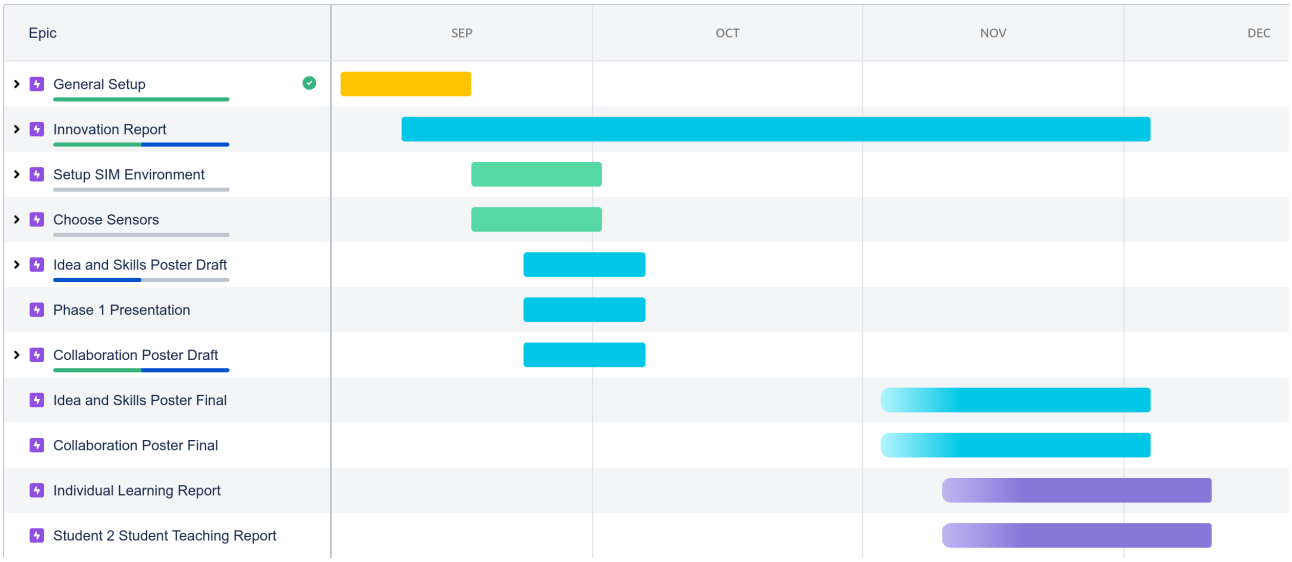


Figure 1: Overview of the time schedule

5 Workflow

This section will explain the workflow of the end-users, how to initialise the product for first time use. Furthermore, it will explain how the everyday usage of this product is done.

5.1 Setup workflow (First time setup)

When any airport or other facility incorporates the autonomous fence inspection solution they will need an initial setup. If the need arises to change the drone inspection route at any point this setup will be performed again. It comprises of the following steps:

1. Identify 4 corner points of the airport fence and record their GPS coordinates.
2. Insert the GPS coordinates in the UI/software to automatically get the flight path around the airport.
   Alternatively.
   The pilot will need to fly the drone manually along the desired inspection path and save the path using the UI/software available.

5.2 User workflow (Routine operation)

The solution will be able to fly autonomously a minimum of three times per day, depending on the end-user requirements. Given that the setup workflow has been performed. The user workflow will comprise of the following steps:

1. Autonomous flight three times per day at specific time intervals.
2. After each flight, employee performs an quality check for the drone.
3. Employee swap batteries on the drone.
4. Drone is now ready for the next upcoming flight.
5. This routine will continue for the amount of security check required per day.
6. If breach is detected an alert is sent to the employee with GPS coordinates.
7. The employee should act according to end-users security requirements and alert further authorities.
8. If a weather alert occurs, the employee must take charge and manually do the fence inspection.
9. In case of a critical failure of any kind, the drone will land at its given position, send an alert to the employee, with its current/last known position.
10. Critical failure: Employee should fetch the drone and place it at the landing/takeoff platform and inspect flight logs.

Following these instructions on a daily basis is required by a certified employee, for this product to be deemed a success.

6 Robotics

This section will explain in detail the robotics part of this project. Moreover, which hardware are used on the drone, how the offboard controller node is working, to automatically move the drone between waypoints. This was developed and tested in a simulated environment using Gazebo.

6.1 Hardware

A small overview of the drone and hardware components needed on a very high level can be seen in Figure 2a and Figure 3.

(a) Image of the drone with mounted camera

(b) Lume Cube Panel Mini [14]

Figure 2: Illustrations of the drone and Lume cube

Furthermore a Lume cube panel mini had been placed underneath the drone to make it able to illuminate the fence when it gets dark, this will ensure that the vision algorithm is still able to get some nice data during darkness [14].
This small piece of hardware, has a very nice illumination range and is very bright, meanwhile it is very lightweight. An illuminated image can be seen in Figure 14a.

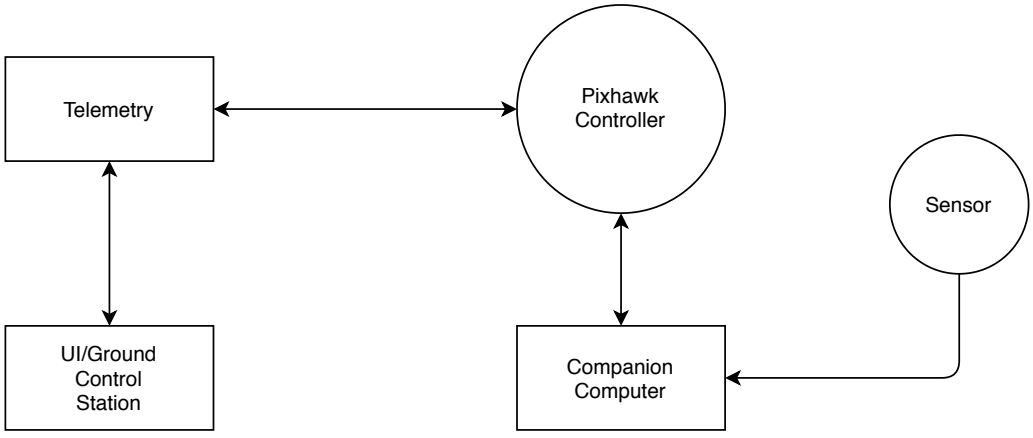


Figure 3: Overview of the different hardware components.

In Figure 3 an overview of the connected hardware are visualised. The sensor is directly fed through an USB to the raspberry PI model 3 B+ which then talks to the Pixhawk with the use of MavRos. A pair of telemetry antennas are used, with TX on the drone and the receiving unit e.g. on a laptop which enables communication and status updates on the drone while it is flying through e.g. QGroundControl.

6.2 Simulation and Software

Initially, a Gazebo server and client was set up to emulate the drone, fence and the different sensors. To ensure that the drone would be able to do autonomous flight and inspection, a ROS framework to control the drone was designed that could also handle the sensors put on the drone.

The ROS framework was based upon a modular structure, drawing inspiration from behaviour based robotics and the fundamental structure of ROS. The structure was based on a core, a message handler and distributed nodes with specialised functionality. A graphical representation of the framework can be seen in Figure 4.

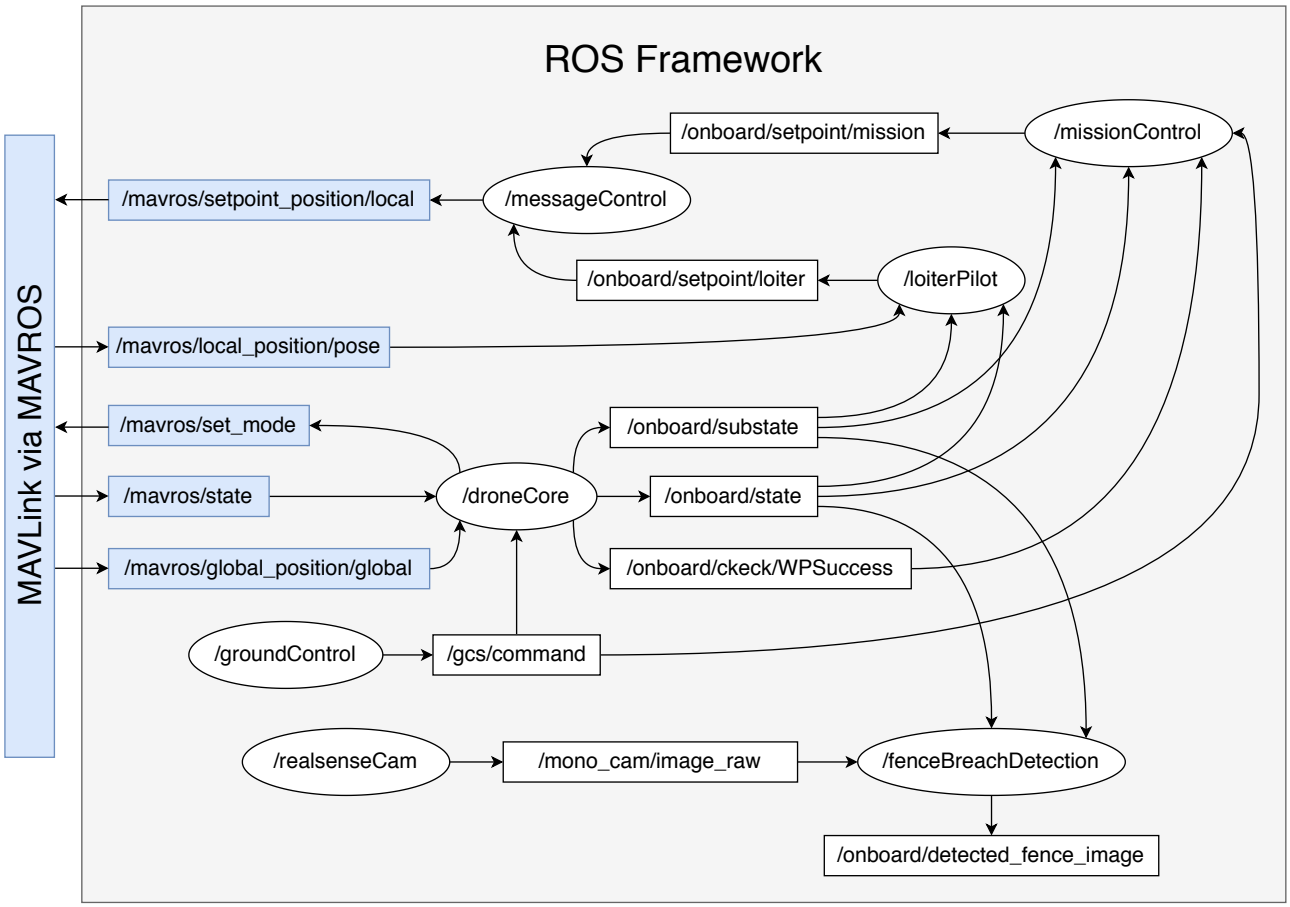


Figure 4: ROS framework for the Gazebo simulation

The oval and rectangular elements of Figure 4 represent ROS nodes and ROS topics respectively. The design consists of two main nodes that handles the core functionality of the framework. The dronecore and messageControl node, these nodes were designed to be the only nodes that could write to the PX4 related topics, where all nodes could read from those topics. This idea was considered to reduce the number of nodes that talks to the PX4 and give a clear separation between the PX4 and companion computer. This means that all nodes that would like to give positional information to the PX4 would have to be forwarded though the messageControl node. This will add a little more computational complexity to the system, but will enable the system to fully control what nodes are able to talk to the PX4 while also having access to different kinds of positional information. This could be beneficial if a waypoint needs to be corrected based on the distance to the fence. Meaning that it would be easy to add deviations to the flight paths.

droneCore: This node was designed to be the only node that could change settings on the PX4 flight controller and handles all state changes for both the PX4 and companion computer. It also takes commands from the ground control station, and contains core functionalities like takeoff, PX4 based landings and writes to the topic */onboard/WPComplete* if a waypoint have been reached.

A basic state machine was designed to handle different scenarios of drone flight seen in Figure 5.

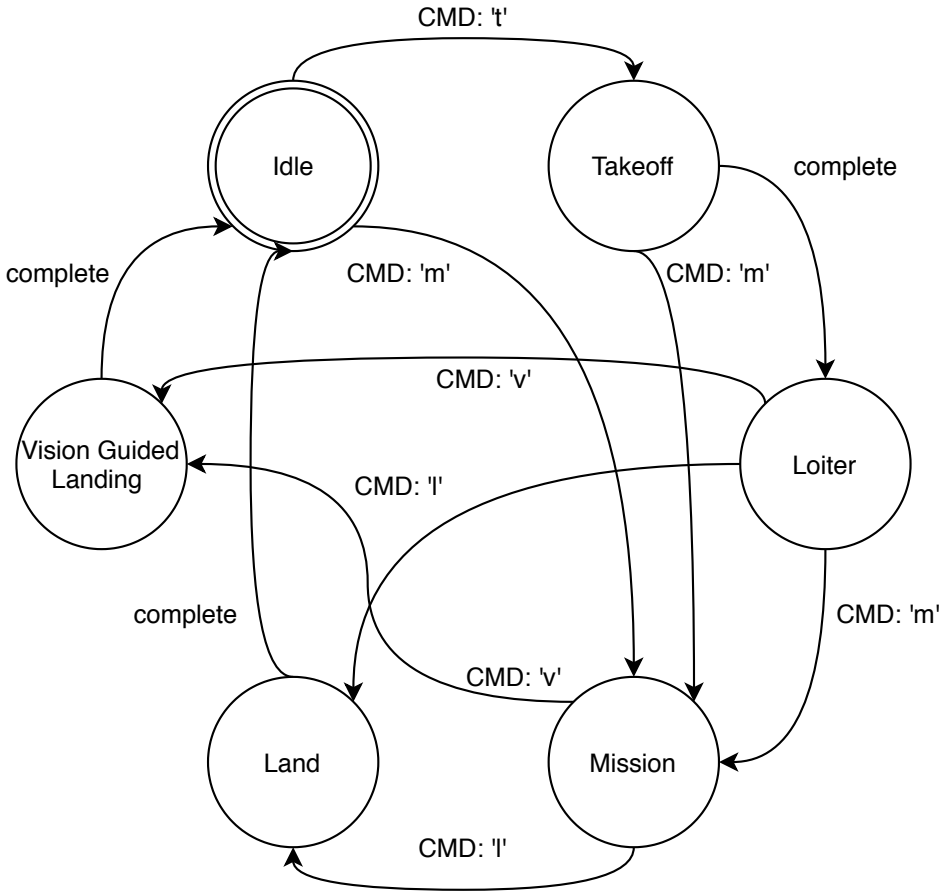


Figure 5: Illustration of the finite state machine for the drone companion computer

The state machine for the droneCore node can be seen on Figure 5. The state machine has six states; when the PX4 flight controller has the control of the drone (idle mode), a takeoff state to perform an automated takeoff, automated landing state, a loiter state where the loiter pilot will be enabled, a mission state where a mission will be executed and vision guided landing state where an autonomous vision based precision landing will be performed.
The vision guided landing feature was not intended to be something that would be implemented but preparing the system for the feature would make implementation easier.
The commands that initiate state transitions are commands from the ground control station. The commands will be discussed in combination with the ground control node below.

LoiterPilot: This node was designed to enable to drone to loiter while in offboard mode. The node takes advantage of the PX4 position hold functionality, and only resends the local position setpoint as the PX4 would otherwise think that companion computer have died. If it dies it will deploy a fail-safe routine. It was implemented with additional functions to alter the position of the drone for easy debugging in simulation. The following functions were implemented; moving along the x, y and z axis in increments of 0.5 m, and yaw rotation of the drone. The axis movement does not take the drones orientation into account.

MissionControl: This node was designed to execute a mission. Although the PX4 is capable of executing a mission it only specifies a flight path and some additional features was needed to perform an inspection. The idea was that a mission would be written as a file of commands, that would be executed. The following commands where added to the mission control node. *COMMAND* for performing a command. Could either be a ground control based command like takeoff/land or an internal command like start recording. *STATE* for changing the onboard state. *SUBSTATE* for changing the onboard substate like enabling a vision algorithm. *PARAM* for updating a PX4 parameter, e.g. max velocity. *WAYPOINT* for giving a waypoint to navigate to. Confirmation that the drone have moved in position will come from the droneCore. This was implemented using local coordinates and quaternions, but it was intended to be based on global GPS coordinates in a final version.
This node implements the same basic functionality of loiter pilot. One of the reasons for duplicating some of the functionality of the nodes, was to ensure that the droneCore could enable a different node and continue to have control over the drone if the current node crashes. If parts of the mission control crashes the loiter pilot would take over and keep the drone in the air at the current location. If everything fails then the PX4 will enable a fail-safe routine, e.g. land. In Figure 6, the state machine for the mission control node can be seen. It contains a state for each command, an idle state, a state that waits for a given state change, a state that waits for being in position and a state for fetching the next command. The only two states that does not lead directly back to fetching the next commands are send command and set waypoint, as these commands require a wait state. If a takeoff is being performed with a send command, then the mission control will wait for the system to change into loiter mode.

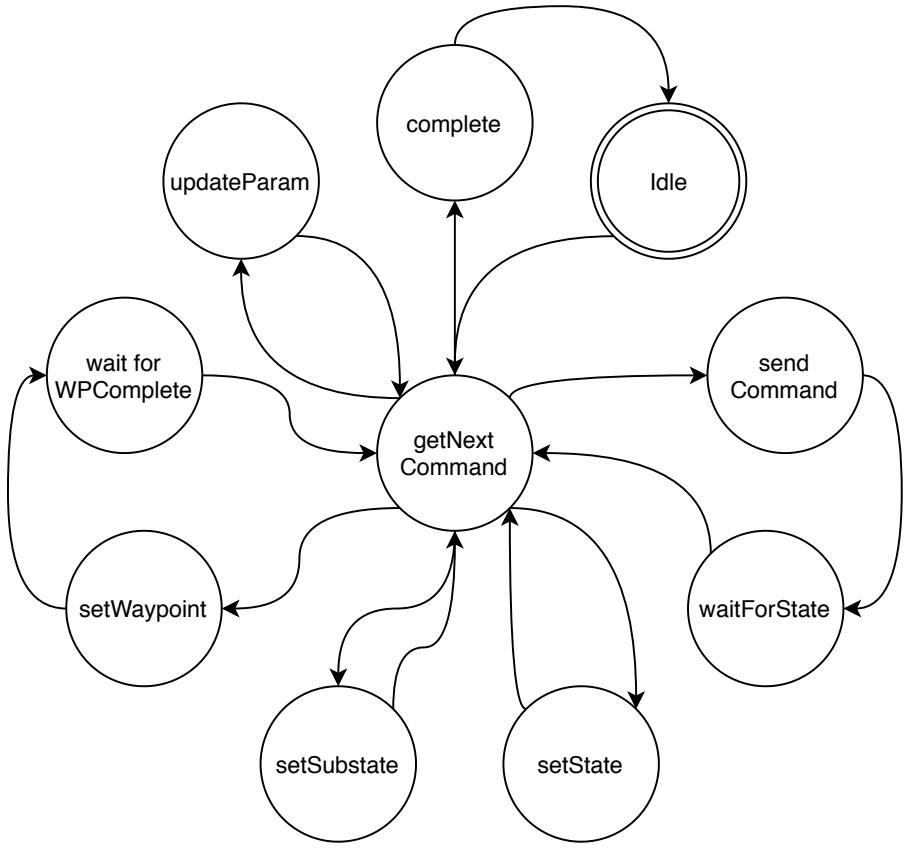


Figure 6: Finite state machine for mission control via the companion computer

MessageControl: This node was designed to relay all positional inputs from the different control nodes to the PX4, based on the overall state off the system. This architecture was chosen to ensure that as few node as possible have write access to the drones flight controller. This setup also enables the node to see if the current controller node is outputting positional information, and uses those topics as heartbeat signals for the controller nodes. If the current controller stops responding, the control will be given to the loiter pilot. This feature was added to give the companion computer time to resolve itself while being in control of the drone and only handing over the control of the drone to a PX4 fail-safe if necessary. Any functionality to restart a node was not implemented but intended to be a part of the droneCore node.

GroundControl: This node was designed to emulate a ground control station in the simulation. The node would take inputs from the keyboard and send them to the companion computer. This could be implemented on a drone with a telemetry link and some additional security to minimise the risk of hijacking. The commands that the drone can receive from the ground control node can be seen in Table 1.

Keypress	**Action**
t	droneCore: takeoff + offboard + loiter mode
o	droneCore: offboard + loiter mode
v	droneCore: vision guided landing (not implemented)
m	droneCore: mission
h	droneCore: move the drone to home and land
k	droneCore: kill switch (not implemented)
r	droneCore: reset ROS framework (not implemented)
l	droneCore: land at current location
wasd	loiterpilot: forwards, left, back, right
qe	loiterpilot: yaw ccw/cw (respectively)
zx	loiterpilot: decrease/increase altitude

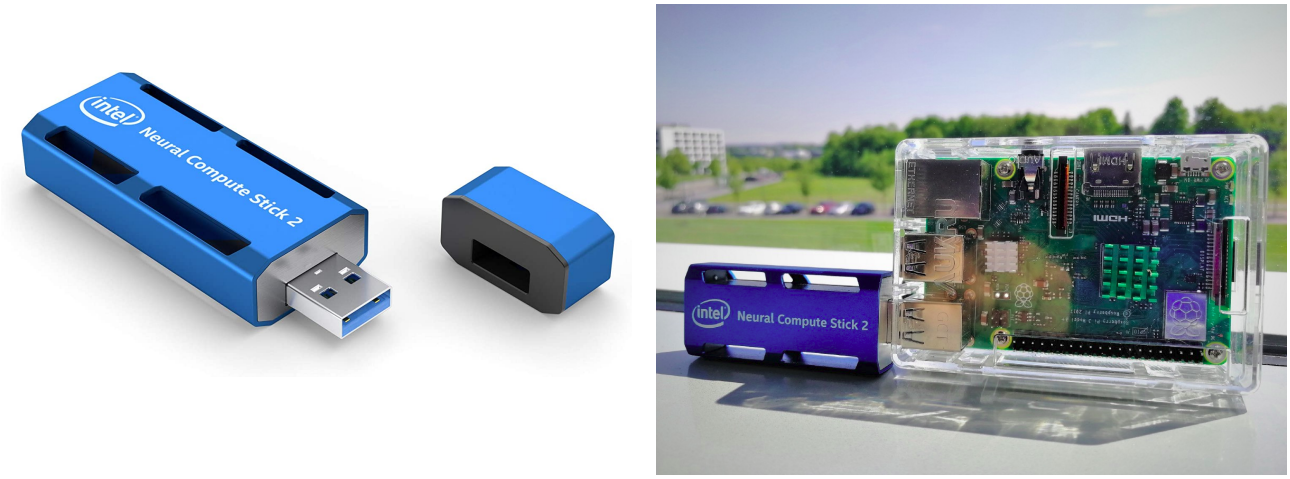
Table 1: Table of all possible commands from ground control station

The commands given to the companion computer was implemented with characters to make easy use of the keyboard.

RealsenseCam: This node was designed to function as a driver for the Intel Realsense camera and publishes the image data to a ROS topic. It also does a little pre-processing to get a depth image from the depth information that the camera gives. If software for adaptable camera settings will be needed it would be implemented in this node.

FenceBreachDetection: This node was designed to be the node for detecting fence breaches. An Fast Fourier Transformation (FFT) based detection approach was implemented in this node originally, that was able to run real-time on the Raspberry Pi. See subsubsection 7.2.1.
A more advanced deep-learning based approach have also been looked at, which would probably be too heavy computationally for the Raspberry Pi if run real-time. Therefore the approach for that solution would be to save the data and post-process it after landing.
An additional solution to this have been considered to enable real-time performance with an AI based approach. This solution requires some additional computational power for the AI inference model based on an Intel Neural Compute Stick 2.

In Figure 7a an illustration of the Intel Neural Compute Stick 2 can be seen. It is a small portable computational stick that are specially developed for deep-learning inference. It features a Movidius Myriad X Vision Processing Unit (VPU) with 16, 700 MHz SHAVE cores, that according to Intel are special CPUs with an instruction set tailored for deep neural nets and with a price of 595 DKK it was found to be a very capable device [11, 10, 9].
It can be used with an Raspberry by running Ubuntu and a Tensorflow based inference model. An illustration of this can be seen in Figure 7b. Additional sticks can be added to the system for additional computational power.



(a) Illustration of the Intel Neural Compute Stick 2. It is the size of a large USB thumb drive [3]

(b) Illustration of the Intel Neural Compute Stick 2 with a Raspberry Pi 3B+ [16]

Figure 7: Intel Neural Compute Stick 2

6.3 Final simulation test

Here a final simulation test will be documented. A simple fence was added to the optitrack Gazebo world, and a simple mission was written to arm and takeoff the drone. Then fly to the left of the fence, start the fence inspection, and setting the maximum xy velocity to 1 m/s to ensure smooth fly by. Thereafter flying the drone to the right of the fence, resetting velocity settings and returning home to land. The mission file used can be seen below.

```
COMMAND;t;loiter
STATE;mission
WAYPOINT;-5.0;-1.0;1.0;0.0;0.0;90.0
SUBSTATE;fence_breach_detection
PARAM;MPC_XY_VEL_MAX;1.0
WAYPOINT;5.0;-1.0;1.0;0.0;0.0;90.0
SUBSTATE;idle
PARAM;MPC_XY_VEL_MAX;12.0
PARAM;MPC_VEL_MANUAL;10.0
WAYPOINT;0.0;0.0;1.0;0.0;0.0;0.0
COMMAND;l;land
COMPLETE
```

In Figure 8 illustrations of the simulated flight can be seen with the drone taking off, moving left, slowly moving right and then returning to land.

7 Vision

In this section a complete walk through of the thoughts, methods and results of the different vision algorithms, that has been implemented are analysed and tested to locate a breach. The methods that will be analysed in this section include:

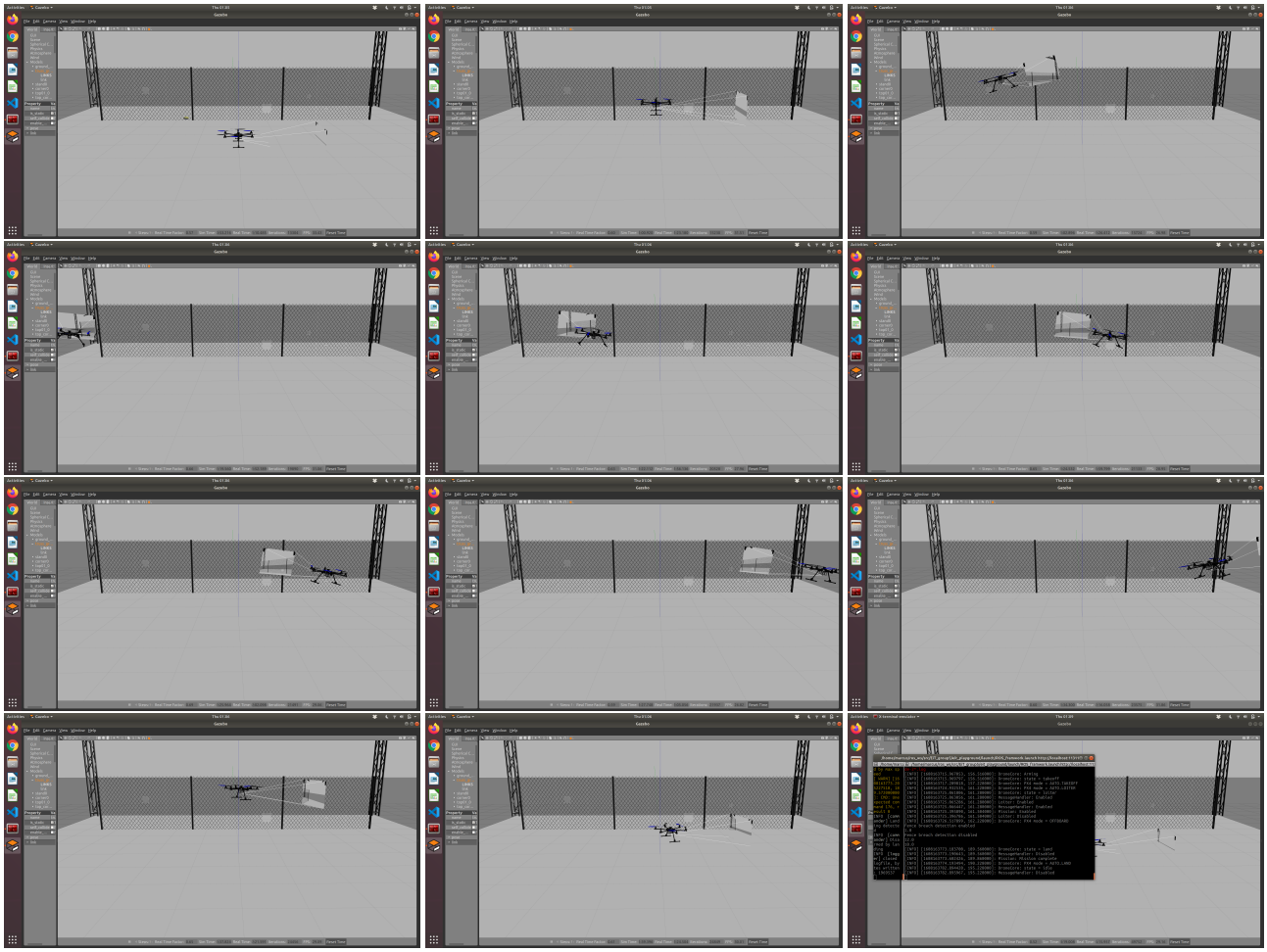


Figure 8: Illustration of a simple autonomous flight in the Gazebo simulation

Fourier Transformation to find the rectangular pattern in a fence structure, image segmentation using U-Net deep-learning method, a canny edge detection as well as a convolutional approach to segment the fence from the background and lastly a Convolutional Neural Network.

7.1 Analysing the different sensor options

Using a LiDAR could prove to be a powerful tool and a very interesting approach to follow. However, with the given drone, which has a current weight of 2500g, see Appendix A, and a maximum takeoff weight (MTOW) of 3200g, this lead to a total added weight from external sensors of: $3200g - 2500g = 700g$. For the lower end price of LiDAR's it is not uncommon that it weights over 1kg. Which is undesirable. It is possible to find a LiDAR below 700g. However, they come with a cost which is not suitable for this project. Thus, no further exploration of the use of LiDAR will be conducted.

A cheap and viable solution would be to use a Intel Realsense D435 Depth Camera. This will give access to multiple sensor within this one device, as well as a cheap cost of below 2500 DKK. It will grand access to a mono-RGB camera, depth sensor and even stereo cameras if needed [7].

7.1.1 Depth Camera Analysis

Depth camera is a powerful tool which provides depth and distance information for the objects in the images. The basic principle of depth camera is either projecting IR or structured light to get depth information or using closely placed cameras for capturing and comparing images (Stereo setup). Depth images are relatively easy to process, because of the depth

information associated with the pixels. We tested the Intel RealSense D435 Depth Camera which uses active IR stereo to generate depth images.

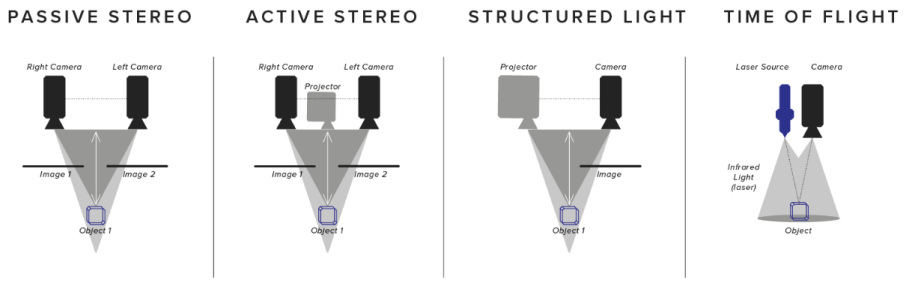


Figure 9: Depth Sensor Configurations

Initially the idea of using the depth camera was to generate a point cloud only consisting of points between 1 and 4 meters away. These point would then be used to reconstruct the grid structure of the fence and a breach could be detected. In theory this would work, but as we discovered, when dealing with actual data from the sensor, the data was unusable for this approach, see Figure 10.

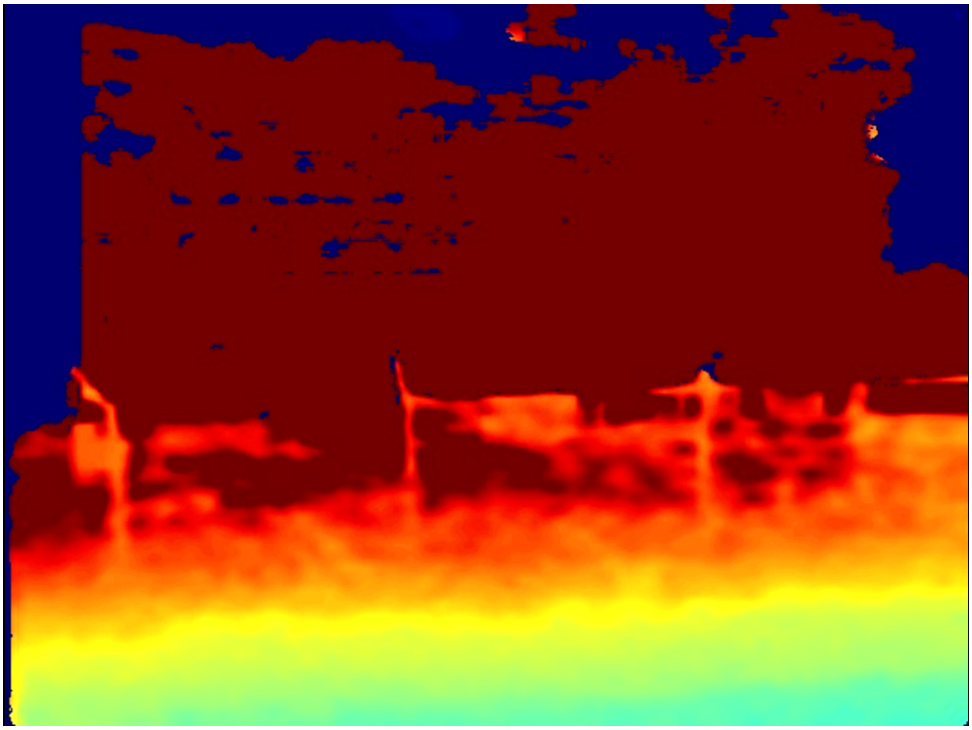


Figure 10: Real life data from depth sensor

This data is not usable for our approach, however a new hypothesis was made, that using some mean value distance of multiple points within the center of the image could perhaps be utilised to maintain a certain distance from the fence at all times.

7.2 Segmentation of The Image

This section will go through and analyse four different approaches to segment an image. During segmentation the goal is to remove as much as the background as possible, without compromising the fence structure. This is required to leave as little as possible to interfere with the screening algorithm. See subsubsection 7.3.1.

This algorithm will systematic go through the grid structure of the fence, locating intersections and detecting obstructions in this structure to verify a breach and its size.

7.2.1 Fourier Transformation

A way to separate the background from the fence was to use the Fourier Transformation which can decompose an image into its sine and cosine components. The output will be given in the frequency domain with the input given in the spatial domain. In this frequency domain each point represents a given frequency in the spatial image domain.

The two-dimensional Discrete Fourier Transformation (DFT) and its inverse is given in Equation 7.1 respectively.

$$F(k,l) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f(i,j)e^{\tau 2\pi\left(\frac{ki}{N}+\frac{lj}{N}\right)} \qquad f(a,b) = \frac{1}{N^2}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} F(k,l)e^{\tau 2\pi\left(\frac{ka}{N}+\frac{lb}{N}\right)} \tag{7.1}$$

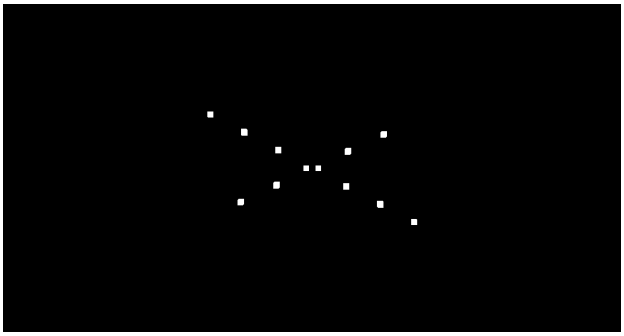
Here the $f(a,b)$ is the image in the spatial domain with the exponential as the basis function that corresponds to every point $F(k,l)$ in the Fourier transform. Hence the value for each point $F(k,l)$ is obtained by a multiplication of the spatial image with the corresponding base function and summing the results. The $F(0,0)$ is the DC-component (average brightness) and $F(N-1,N-1)$ is the highest frequency. The term $\frac{1}{N^2}$ is used as a normalisation term. The Fourier Transform is separable therefore it can be rewritten as seen in Equation 7.2.

$$f(a,b) = \frac{1}{N}\sum_{b=0}^{N-1} P(k,b)e^{\tau 2\pi\frac{lb}{N}} \qquad P(k,b) = \frac{1}{N}\sum_{a=0}^{N-1} f(a,b)e^{\tau 2\pi\frac{ka}{N}} \tag{7.2}$$

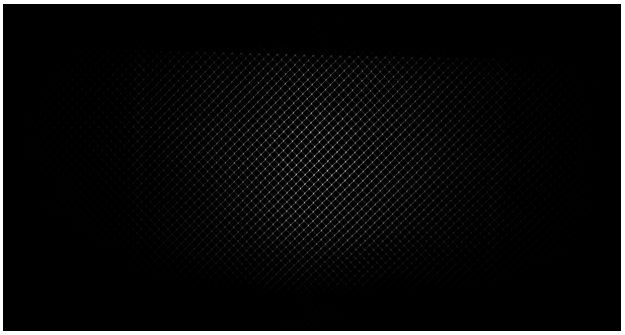
Calculating a two-dimensional Fourier Transformation can be done using a series of two one-dimensional calculations. This decreases the amount of required computations, yielding a $O(N^2)$ computational time. To improve further a Fast Fourier Transformation (FFT) could be used, which reduces the time complexity to $O(Nlog_2N)$. This was one of the main reasons for choosing the Fourier Transformation to segment the fence from the background. Since this enables the calculations to be done very fast and utilises the possibility of detecting breaches closer to real-time.[15]

Analysis:
The performance evaluation of the FFT will be based on images in different configurations. This is done to stress test the algorithm, in order to find its limitations. The code used for this algorithm was heavily inspired by a script given from Henrik Skov Midtiby (*hemi@mmmi.sdu.dk*) which involves FFT in Python and OpenCV. The procedure will follow the approach described in Section 7.2.1. Given the image in Figure 11a, the Fourier spectrum was found and visualised in Figure 11b. Here the structure of the fence is nicely highlighted as seen in the frequency spectrum of the FFT. The mask is now multiplied with the input image and the results can be seen in Figure 11c.

(a) Input image of fence without breaches.



(b) Fourier spectrum of input image (mask).

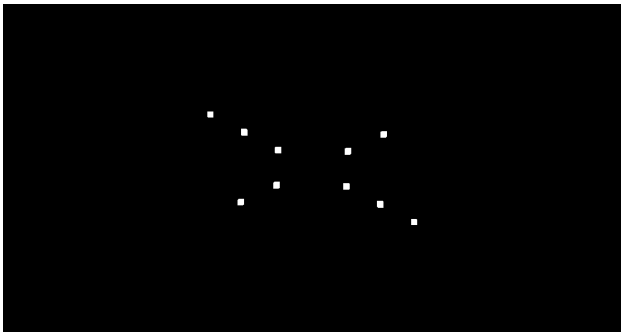


(c) Multiplication of mask and input image.

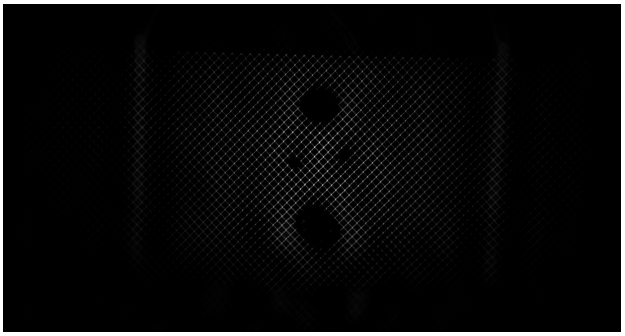
Figure 11: Illustration of the working of the Fourier transformation resulting in a background subtraction of the image in Figure 11c

The results is a segmented image where only the structure of the fence is visible. It may the noticed that this image is underexposed which results in a fine separation of the foreground (fence) and background (forest). This image is taken under ideal conditions in regard to optimising the performance of the FFT.

(a) Input image of fence with breach.



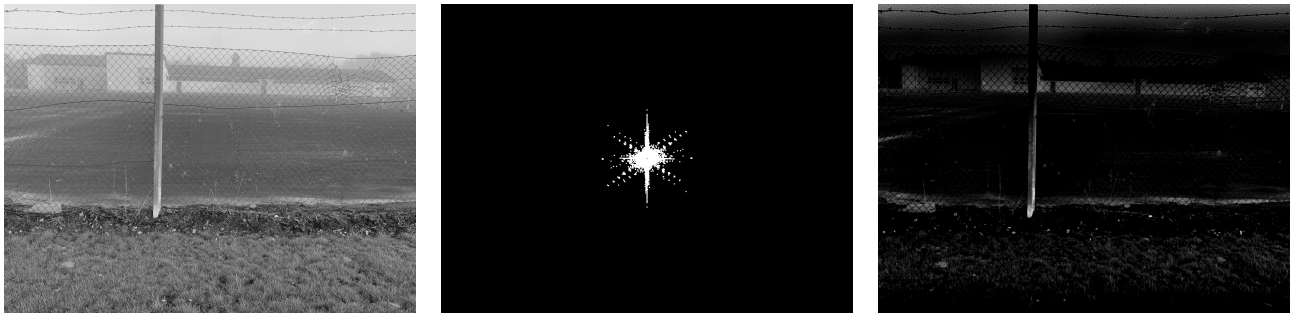
(b) Fourier spectrum of input image (mask).



(c) Multiplication of Mask and input image with breaches

Figure 12: Input image with breaches and the resulting background subtracted image suing Fourier transformation

Now giving an input image with customised breaches, the Fourier spectrum can be seen in Figure 12c. Applying this mask on the input image the resulting segmentation can be seen in Figure 12b. From this result it is possible to detect the breaches. The algorithm made to conquer this problem can be seen in Section 7.3.1.

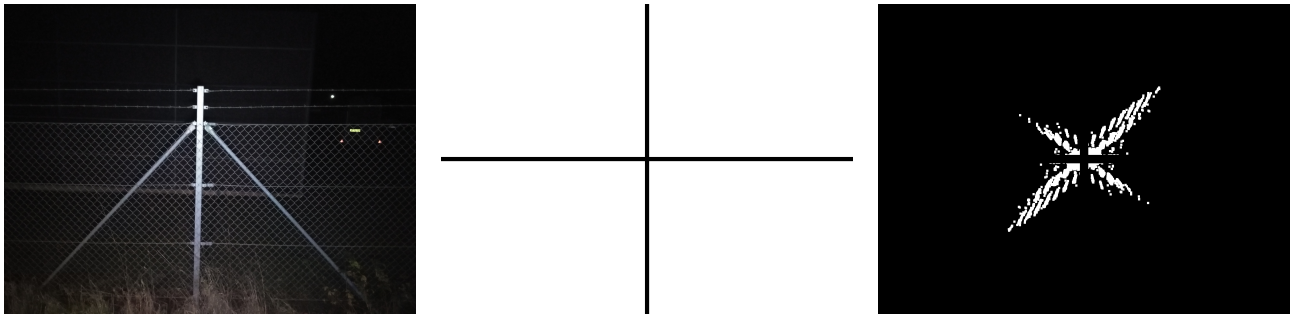


(a) Input image of fence *(b) Fourier spectrum (mask)* *(c) Segmented image*

Figure 13: Input image and the resulting background subtracted image using Fourier transformation

Because the FFT takes into account certain patterns in the image, it is highly sensitive to variations in light, background and other structures which shows some kind of symmetry. Especially if light is projected on structures in the image which is of no interest. Such an image can be seen in Figure 13a. Here the fence is hard to detect with a gray background. Moreover, the light is not projected on the fence the same way as in Figure 11a. This make the problem considerably more challenging. This Fourier spectrum can be seen in Figure 13b. One may notice that the Fourier spectrum includes a lot more frequencies then the ones got in Figure 12c. As it can be seen in Figure 13c, the background is not properly removed from the input image yielding a very bad segmentation.

Ways to deal with this problem could be to project a light source on the drone reflecting upon the fence. This could make a better separation of the fence from the background. Furthermore, the mask in the Fourier spectrum could be multiplied with a user defined mask excluding the frequencies which have no interest. A visualisation of these improvements can be seen in Figure 14.



(a) Light projected on the fence *(b) User defined mask* *(c) Resulting Fourier spectrum*

Figure 14: Input image and the resulting background subtracted image using Fourier transformation

Unfortunately, these improvements did not result in a general improvement of the segmentation of the image. The spectrum in the FFT still has to be tuned to fit the given conditions from where the image is taken. This is a considerable drawback of using this algorithm for segmentation.

7.2.2 *Segmentation using U-Net*

Another approach for using neural networks, is the fence segmentation using neural networks such as U-Net and then performing the breach detection using the fence re-construction methods discussed in the previous sections. U-Net configuration is particularly good at segmenting images. A typical U-Net used for image segmentation is shown in Figure 15.

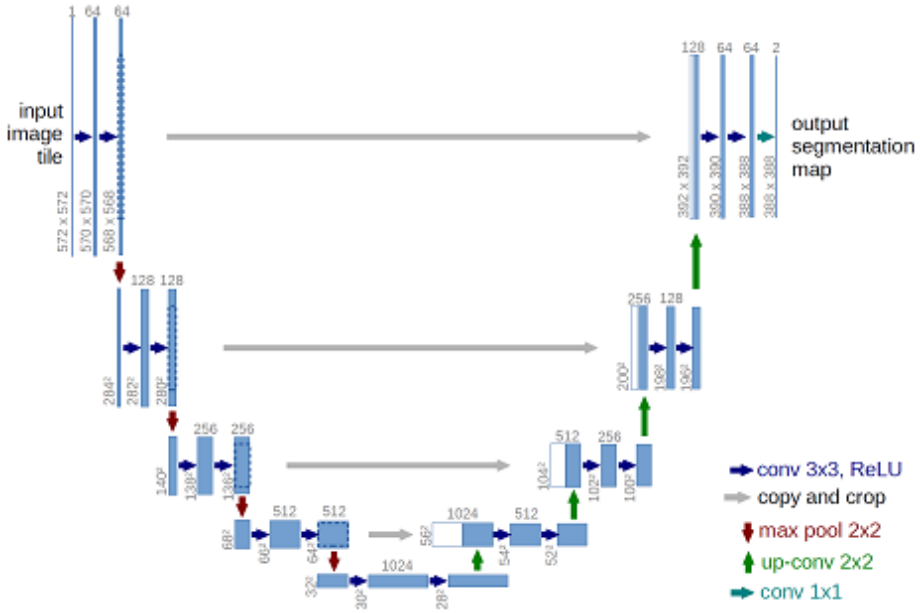


Figure 15: U-Net Architecture [20]

For each pixel in the input image the U-Net outputs a binary pixel classifying it as either background or foreground. This configuration, if properly trained, could provide a very strong segmentation tool. However, to train the U-Net hundreds of different fence images where needed with specified ground truth or mask of the image. This data was not readily available in our case. However, using generated artificial data sets, some level of performance from the U-Net approach could be achieved.

An example of the U-Net input image and ground truth is shown in Figure Figure 16.

Figure 16: U-Net Input Image and Mask

7.2.3 Convolution based segmentation

This method was based on a different idea. Instead of trying to segment the fence structure itself, the intersections in the fence structure would be detected instead resulting in a list of pixel coordinates. A breach detection algorithm could then be developed to find breaches in the structure of pixel locations.

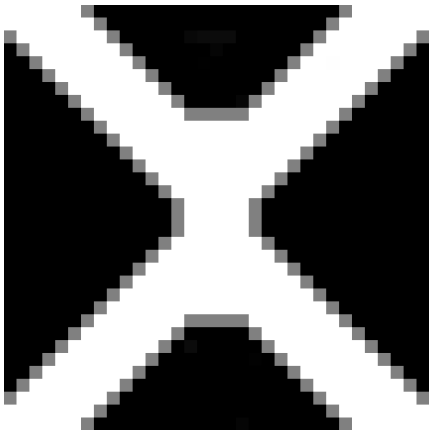


Figure 17: The kernel used for the method

Different approaches was considered but a convolution based solution was chosen. The algorithm would run sequential on each each step based on the blue colour channel. A 33 x 33 kernel was made to generate a high response when being in the center of an intersection. This kernel can be seen as an image in Figure 19. Black correspond to negative contribution, gray zero and white to positive contribution. This kernel was convoluted with the image to produce a response map featuring high responses were an intersection was found. This image was then normalised to values between 0 and 255 and then thresholded. Contours will then be found in the thresholded image and the centroids of those returned as the final pixel position.

In Figure 20 the different stages of the process can be seen. On the left is the blue colour channel as input, next to that the response from the convolution, then the thresholded image and finally the centroids for the found contours drawn onto the original image.
The tuning in this algorithm lies in the thresholding values for the binary thresholding. The methods was tuned to perform its best on the input image.

(a) Input blue channel *(b) Output convolution* *(c) Output thresholding* *(d) Output pixel positions*

Figure 18: Illustration of the processing steps from start to finish

In Figure 21 another lighter image was run though the method and was again tuned to give the best result. It was generally found that the method were relatively forgiving regarding the thresholds as long as the images were underexposed.

(a) Input blue channel *(b) Output convolution* *(c) Output thresholding* *(d) Output pixel positions*

Figure 19: Illustration of the processing steps from start to finish

The method handled grass really well but struggled a bit with tress. The result from this would need some filtering of the found points to be useful for detecting breaches.

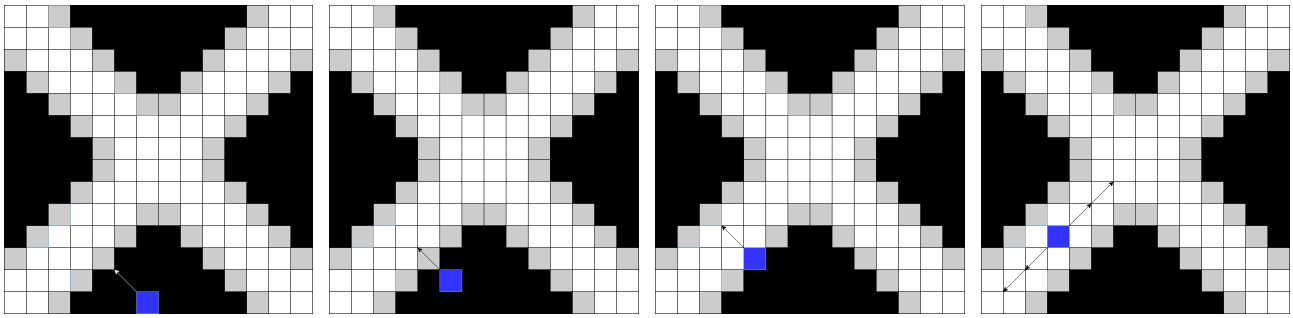
It was considered to maybe connect the points with lines to produce a perfect grid structure that could then be passed to the breach detection algorithm described in subsubsection 7.3.1.

7.3 Breach detection

7.3.1 Screening of segmented images

Given segmented images where the background has been removed with only the foreground (fence) left in the image, the resulting image can be screened for possible breaches. This is done using an algorithm with the following three stages namely *searching for a fence*, *fence following* and *peak and endpoint detection*. The peaks (junctions) is defined as the intersections where the elements of the fence meets to form a junction. This is the midpoint of the images seen in Figure 22.

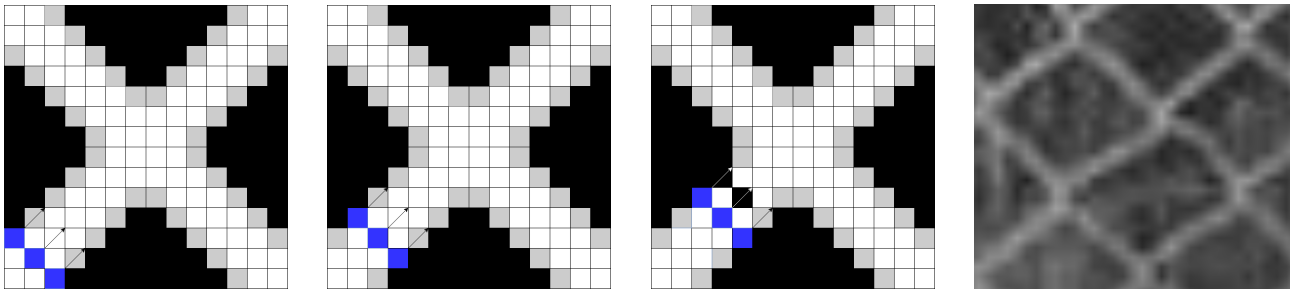
The basic idea is to find the fence starting from a random location in the middle of the image. The pointer can move in an either up-right, down-right, down-left or up-left direction. This is done so that the pointer initially ends up on the structure on the fence, which it now can follow. This is illustrated in Figure 22 where the fence is white, background black and the pointer labelled blue.



(a) Step one *(b) Step two* *(c) Step three* *(d) Step four*

Figure 20: Illustration of searching for a fence in the image. Fence (white), background (black), white arrows (illegal stopping criteria) and black arrows (valid stopping criteria)

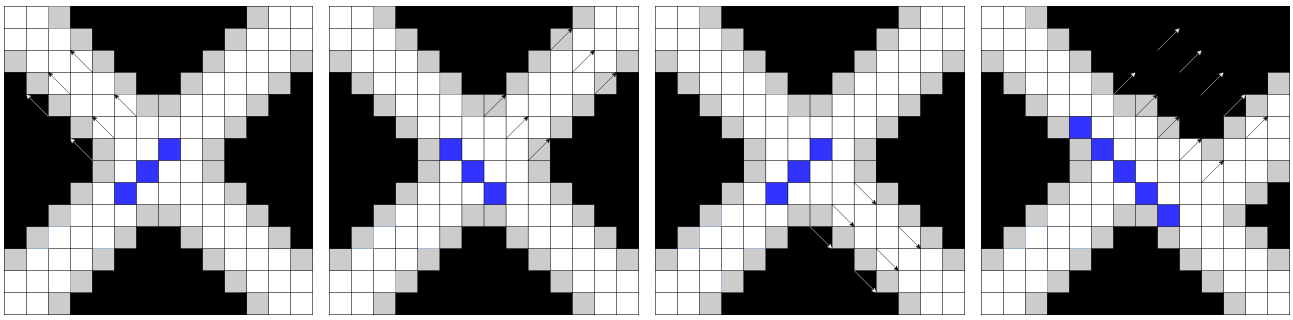
The pointer continues to move in a certain direction until the giving pixel value is above a threshold. When this happens, the pointer will now move in either two directions which is orthogonal to the last move direction. This can be seen in Figure 23. Now the pointer moves in a direction with a chosen step size for each move. The step size can be given as a list of values so that the pointer can move further away from its last position in each step. This is done to take into account low pixel values in the next step even thought the pointer is still on the fence. Moreover, the next step of the pointer can be moved in orthogonal directions according to the last move if an abnormal fence structure is seen like the one in Figure 23d.



(a) Step one *(b) Step two* *(c) Step three* *(d) Abnormal fence structure*

Figure 21: Illustration of the line (fence) follower approach. Fence (white), background (black), white arrows (illegal stopping criteria) and black arrows (valid stopping criteria)

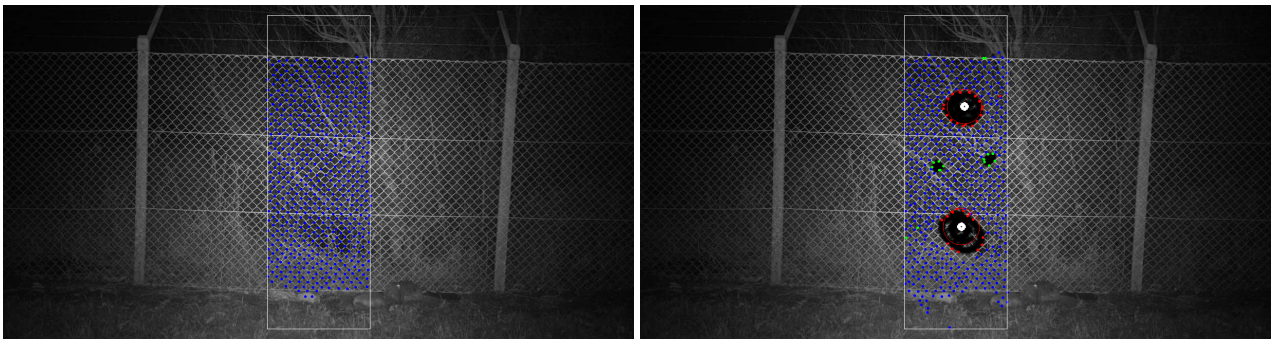
The pointer will continue to move until a peak location or end of fence is seen based on the pixel values. The way the algorithm detects peaks is illustrated in Figure 24. At every step in the fence following approach the algorithm will check for high pixel values in front of it as well as its orthogonal directions according to the last move. If the pixel value in all three directions is above a threshold it will be considered to be a peak location. The point in the image coordinates will be put on a list of already detected peak points. Now the algorithm will branch in the other directions at the peak location to continue its search for peaks and endpoints. This can be done either in an recursive or iterative manor.



(a) Step one *(b) Step two* *(c) Step three* *(d) Step four*

Figure 22: Illustration of searching for a peak (junction) in the fence. Fence (white), background (black), white arrows (illegal stopping criteria) and black arrows (valid stopping criteria)

Endpoints is detected if the fence following part of the algorithm sees only black pixels (pixels below a threshold) in the next steps to come. Hence, the algorithm will return from its given branch and put the endpoint on a list to be used in the analysis of breaches in a later step.
To make the algorithm faster a region of interest (ROI) was defined which can be seen in Figure 25. The recursion will stop if it goes beyond the ROI. As it can be seen, large custom made breaches can be found and small breaches or noise are neglected. The endpoints inside the ROI is grouped together using hierarchical clustering. This approach finds the distance between points. It will group points based on the distance from a given point to that of the mean of a cluster. This is both a fast and effective method in order to cluster points and make an approximation of the size of the breach knowing the cluster diameter.



(a) Fence without breaches *(b) Fence with custom made breaches*

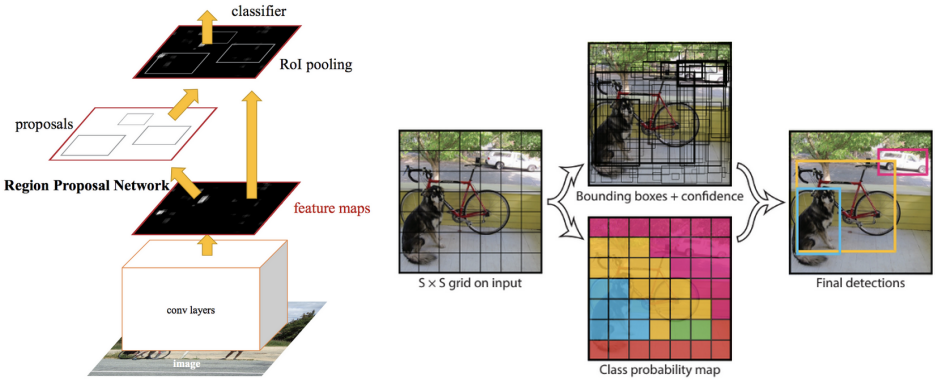
Figure 23: Illustration of screening a fence given images from the Fourier transformation. Peaks (blue), breaches (red), neglected small breaches or noise (green) and ROI (white)

The algorithm performs quite well given a nicely segmented input image. However, because the implementation is based of the need of a really good segmented input image, the algorithm will fail if the input is bad e.g. given a bad segmented image from the Fourier transformation. This is due to the fact that the pointer will follow the structure of the fence, but if the fence is badly segmented from the background, this algorithm will not work.

In theory, it is possible to find very small breaches using this screening approach of the image because it tries to find every peak and endpoint location in the image. The limitations is based on the need for a perfectly segmented input image.

7.3.2 *Image classification and boundary box regression using mask-RCNN*

A number of algorithms where taken into consideration in regard to deep learning. Faster-RCNN is an extension of the region-based convolution neural network (RCNN). The RCNN uses the selective search algorithm to extract proposal regions from the image from where these are given to a convolution neural network (CNN) which extracts a feature vector from the regions in question. Then these are given to a set of class specific support vector machines (SVM) where classification and offsets to boundary box regression is performed. Instead of using the selective search algorithm, the faster-RCNN makes a convolution feature map from the input image using a convolution network. A separate network is then used to find region proposals. This predicted region is reshaped using a region of interest pooling layer to classify the picture being within the proposed region and offsets values for the giving boundary boxes. **Mask-RCNN**, Figure 26a, extends to this idea by adding an additional branch being the objects mask. This enables the possibility to make instance segmentation to the found objects.[17] [13] [8]



(a) The faster-RCNN algorithm [17] *(b) The YOLO algorithm[8]*

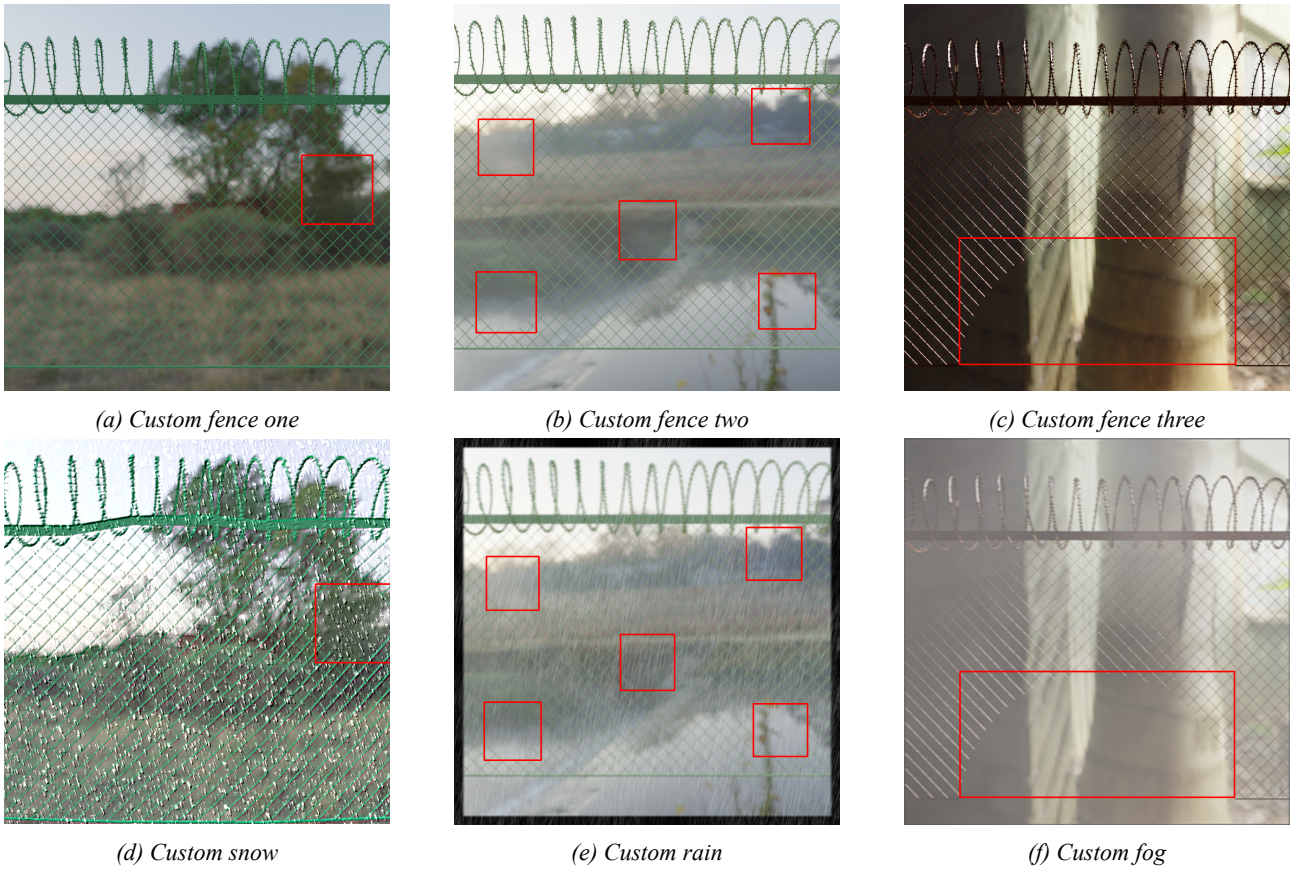
Figure 24: Illustrations of the considered algorithms to use in deep learning

In **YOLO** (you only look ones), Figure 26b, an image is split up into grids of size $N \times N$ where each grid consists of m bounding boxes. It uses a CNN to predict bounding boxes and class probabilities for each box. This algorithm is quite fast, but its comes with the expense of bad detection of small objects.

Given these methods it has been decided to move on with the Mask-RCNN because the precision is preferred instead of speed and the possibility of using the instance segmentation for estimating the size of the breach in the given image.

Data augmentation

In order to train the network a lot of data have to be collected in order to make a robust solution. Shared data from group 171 in the experts in teams course has been used. This data represents a lot of images where a 3D-model of a fence with artificial breaches has been placed in front of different backgrounds as seen in Figure 27a, 27b and 27c. This includes almost 500 images with fences in different configurations. However, more images are needed, so data augmentation is used. The augmented data takes into account different conditions like weather, lightening, blur and noise in the image. This has been done automatically in python where 15 new images have been made from each original image. Every image has been scaled and blurred differently each time. Moreover, rain, snow, fog, Gaussian noise and perspective transformations has randomly been given to each of these images. The result of three of them can be seen in Figure 27d, 27e and 27f.



(a) Custom fence one *(b) Custom fence two* *(c) Custom fence three*

(d) Custom snow *(e) Custom rain* *(f) Custom fog*

Figure 25: Illustration of some of the custom made fences with different backgrounds. The original form can be seen in Figure 27a, 27b, 27c and its augmented parts in Figure 27d, 27e and 27f respectively

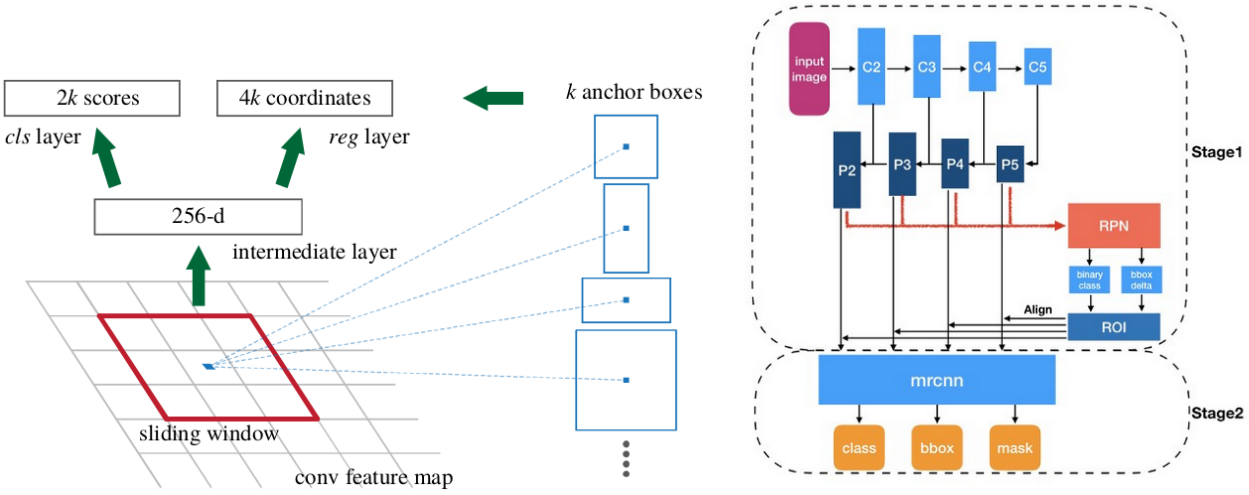
The data augmentation resulted in almost 7000 new images which is going to be used for training the deep neural network (DNN). 80% of the images will be used for training and 20% as validation/test images. The augmentation with snow, rain and fog has been used to take into account real life scenarios when the drone has to fly in different whether conditions and still make a robust estimation of possible breaches in the fences. [12]

Mask-RCNN

Mask-RCNN consist of two stages. The first stage begins with the bottom up path where a computation of the backbone (ResNet101 in this implementation) to compute a hierarchy of features at different scales with a scaling factor of two. This is denoted as *C2* (conv2), *C3* (conc3), *C4* (conv4) and *C5* (conv5) in Figure 28b. This illustrates the features activation outputs from the residual block by each last stage. The conv1 is excluded from the pyramid because of the size (memory). The top down path generates feature maps associated to each stage which is denoted *P2*, *P3*, *P4* and *P5* which is called a feature pyramid network (FPN). The use of FPN is due to its maintenance of strong semantic features at various resolution scales which makes the network scale-invariant for object detection.[19]

Now the top down pathway gets scanned for features using a lightweight neural network called region proposal network (RPN) in a sliding window fashion. The features is bound to raw image locations using anchors. An illustration of this can be seen in Figure 28a. The regions which are taking into account are the anchors. At each location of the sliding window, multiple region proposals are predicted according to the number of anchors k. Here the anchor is predicted to belonging to the foreground (possible object) or background. This is done by computing the intersection over region (IoU) between the anchor boxes and ground truth box locations. If the $IoU > 0.5$, it will be considered positive and being an object. If the $IoU < 0.5$, it will be considered negative and being the background. Moreover, the center coordinates (x,y) along with its width and height are predicted yielding $4k$ coordinates. The scores and coordinates are giving to the classification and regression layer respectively. If several of these anchors are overlapping each other, the anchor with the

highest foreground score will be kept using non-max suppression.



(a) Regional proposal network [17] *(b) Illustration of the mask-RCNN structure [21]*

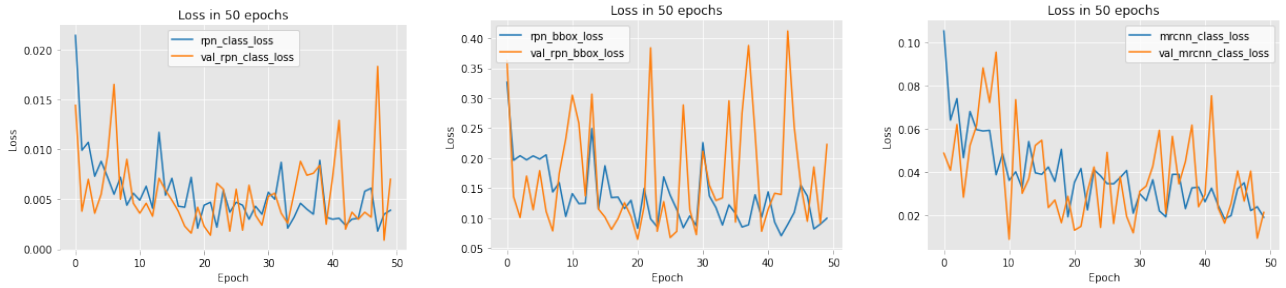
Figure 26: Illustrations of the RPN and associated anchors in Figure 28a and high level visualisation of the mask-rcnn in Figure 28b

In the second stage the region of interest (RoI) is taken into account. These are the anchor boxes got from the predicted objects from stage one using RPN. However, unlike the RPN which only predicted two classes (foreground/background), the network will now label possible objects to classes or disregard them as belonging to the background. Boundary box refinement will now take the RoIs into consideration and refine the box to encapsulate the possible objects of interest. Now RoI pooling is performed using RoIAlign. This is done to extract a small feature map of the RoI without any quantisation loss. This pooling is necessary because the fully convolution layers (FC) expects a fixed input dimension vector to be used in the last step. Now the classification, boundary box regression and instance segmentation will be evaluated. [21] [2]

The implementation of the mask-RCNN is based upon open source software [1] which is coded in python. This yields almost an exact implementation of the algorithm based on the original paper [13]. The training of the network has been done using Googles online platform, Colab, in order to increase the speed of the training process. The data sets have been uploaded to Dropbox from where the images where fetched to Colab. Every image is associated with an xml file which includes image id, image width and height and the boundary box of the custom made breaches. This is done so that the data can be properly loaded with the information to be used in Tensorflow and Keras. A number of parameters have been set when training the model. The training have been done using the following hyperparameters namely a learning rate of 0.001, momentum of 0.9 and weight decay of 0.0001 within the defined configuration model. Both the configuration and prediction model will be used with a *detection minimum confidence* of 0.7, meaning that the score of a given object prediction must be above a probability of 0.7 in order to be recognised as belonging to a certain class e.g breach in the fence. Furthermore, transfer learning has been used where pretrained COCO weights for the backbone ResNet-101 was implemented. This enables faster training using the custom made dataset [5]. This implementation of mask-RCNN gives the opportunity of using ResNet50 and ResNet101 which are 50 and 101 layers deep respectively. However, the ResNet101 is used because it have proved to have a better performance. This will come at the expense of a longer training time due to the dept of the network. [4]

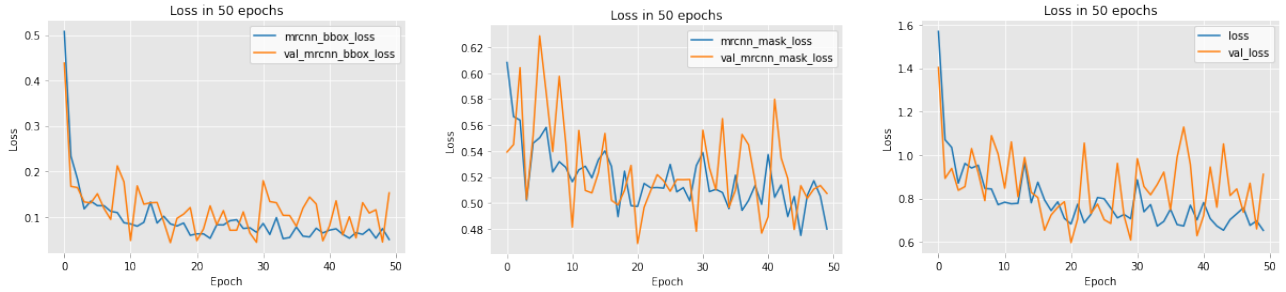
Analysis

The network will be trained with two different datasets namely the original one with 465 images and the augmented with 6975 images. The smaller dataset is given a batch size of 100 and the bigger 200 and both of 50 validation steps per epoch respectively. The learned weights of the network will be saved for later use. The results of training with the augmented dataset can be seen in Figure 29 and 30.



(a) RPN class loss *(b) RPN boundary box loss* *(c) Mask-rcnn class loss*

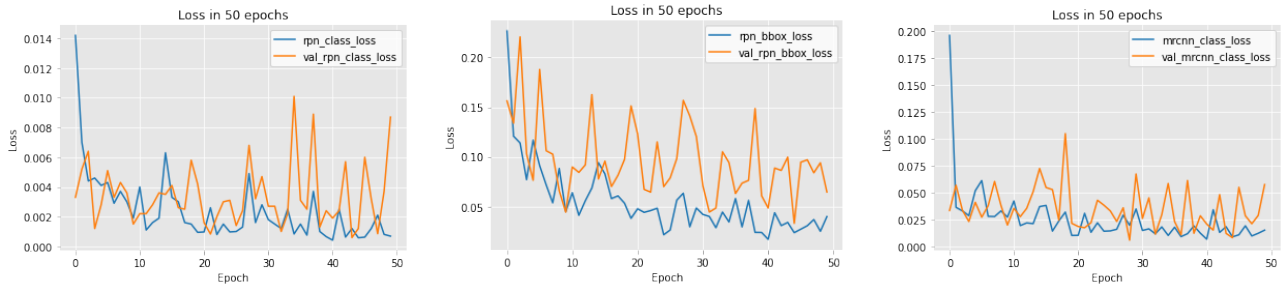
Figure 27: Illustrations of the loss during training in 50 epochs using 6975 images where 80% is used for training and 20% as validation/test



(a) Mask-rcnn boundary box loss *(b) Mask-rcnn mask loss* *(c) Total loss*

Figure 28: Illustrations of the loss during training in 50 epochs using almost 6975 images where 80% is used for training and 20% as validation/test

In Figure 30c the total loss can be seen. The training loss is seem to converge. However, the validation loss seems to fluctuate quite a lot. By studying Figure 29b and 30b, it is seen that the mean contributors to the total loss is the RPN boundary box loss and the mask-rcnn mask loss. The reason for the mask loss could be that no mask is given in the dataset yielding no training data for the mask besides the area on the boundary box which is not precise enough to properly train the network in regard to the instance segmentation. It is seen in Figure 29c and 30a that the mask-rcnn class and boundary box loss are converging as inspected with only minor fluctuations.



(a) RPN class loss *(b) RPN boundary box loss* *(c) Mask-rcnn class loss*

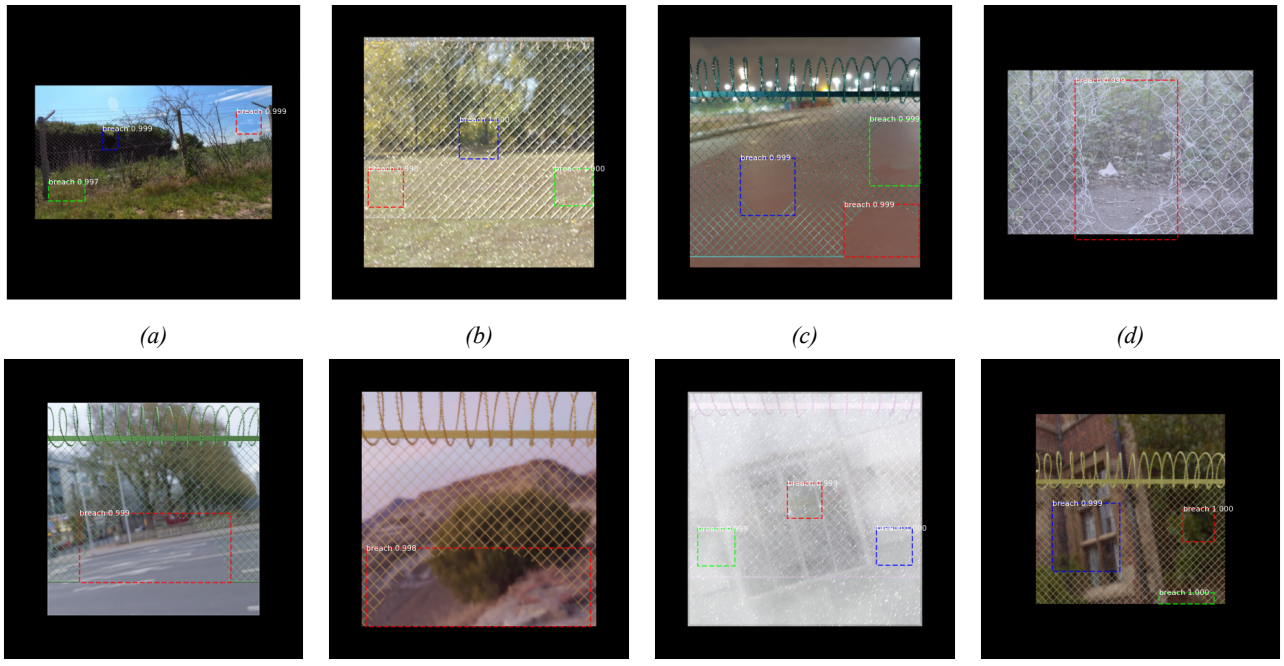
Figure 29: Illustrations of the loss during training in 50 epochs using 465 images where 80% is used for training and 20% as validation/test

(a) Mask-rcnn boundary box loss *(b) Mask-rcnn mask loss* *(c) Total loss*

Figure 30: Illustrations of the loss during training in 50 epochs using almost 465 images where 80% is used for training and 20% as validation/test

In Figure 31 and 32 similar conclusions can be made. Here the network has been trained using the small original dataset consisting of 465 images. It may the notised that the mask-rcnn class and boundary box loss is converging as seen in Figure 31c and 32a.

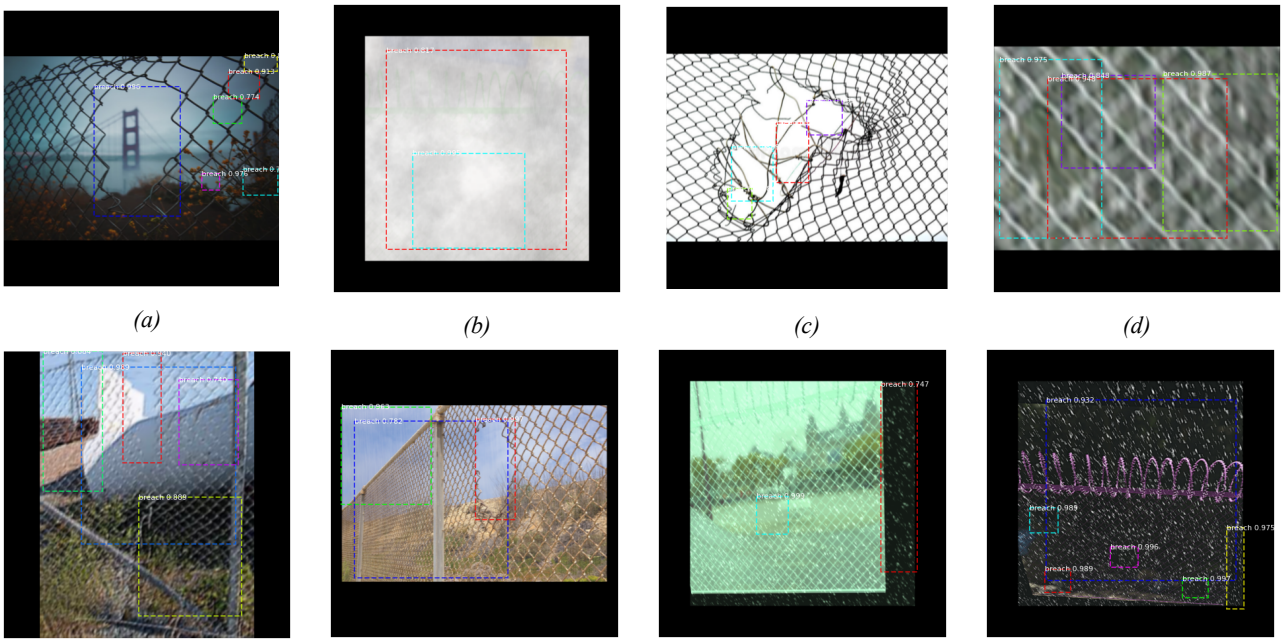
A subset of the augmented validation set along with the predicted breaches in each image can be seen in Figure 33. The breaches have all been found even in difficult conditions. This illustrates that the mask-rcnn performs quite well in estimating the boundary box and the image classification. This has been done using weights gained from training the network with the augmented dataset.



(a) *(b)* *(c)* *(d)*

(e) *(f)* *(g)* *(h)*

Figure 31: Illustration of the mask-rcnn performing well in finding some of the breaches in the validation set. Each breach is encapsulated within a bounding box with a probability score going from zero to one

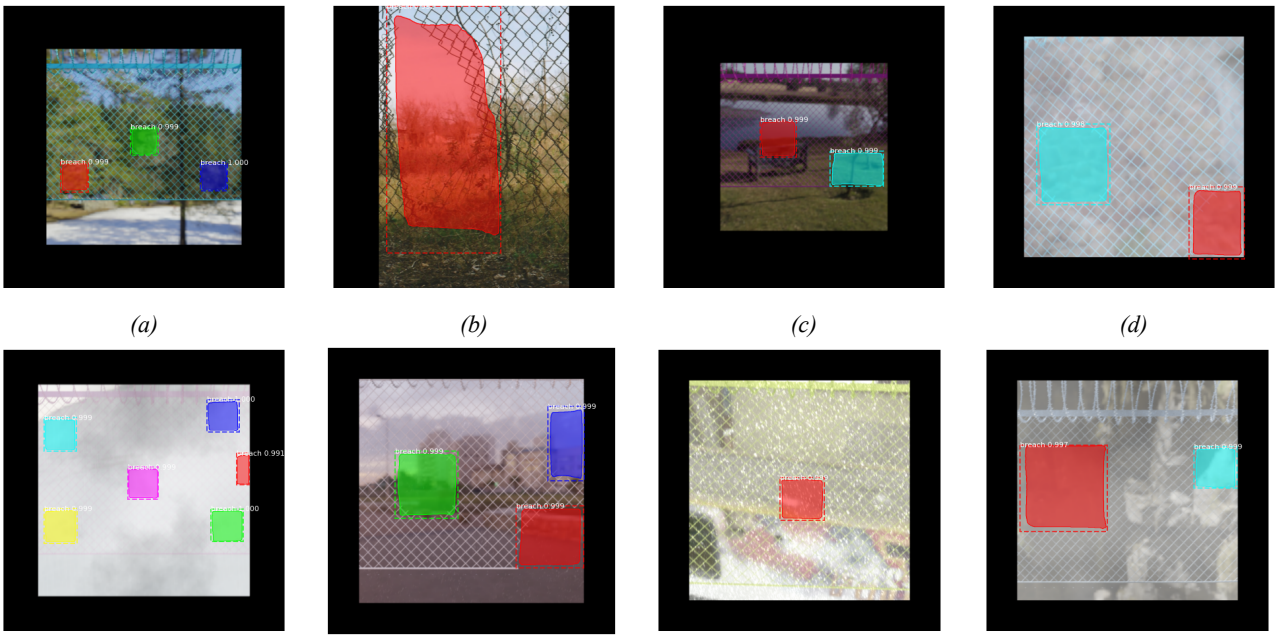


(a) *(b)* *(c)* *(d)*

(e) *(f)* *(g)* *(h)*

Figure 32: Illustration of the mask-rcnn performing bad in finding some of the breaches in the validation set. Each breach is encapsulated within a bounding box with a probability score going from zero to one

In Figure 34 the mask-rcnn do not perform quite as well. It can be seen in the figure that multiple wrong objects have been predicted as belonging to the breach class. However, it must be mentioned that the found breaches in Figure 34a and Figure 34c is taking very close to the fence which will not happen in real life using the drone. The algorithm is actually able to find the breaches, but with many wrong detections as a site note. Most of these could effectively have been sorted out by setting the limit of accepted objects to 0.9 or 0.9995 probability which will be done in section 8. Furthermore, by using a training set which includes many instances of close up fences, the algorithm would properly perform better in these cases.



(a) *(b)* *(c)* *(d)*

(e) *(f)* *(g)* *(h)*

Figure 33: Illustration of the instance segmentation using mask-rcnn. Each color encapsulated within a bounding box represents the area associated to the found breach

The results of instance segmentation can be seen in Figure 35. The area which belongs to the breaches have been nicely separated even in difficult conditions like the one in Figure 35b. However, it must be mentioned that the estimation of some instance segmentations have been seen to fluctuate quite a lot if the algorithm has trouble of finding the breach. But in most cases in works very good. Now by knowing the the distance to the fence, an approximation to the size of the breach can be found by using the instance segmentation if wanted. In Table 2, the different weights of the two different trained networks are shown, where the ”Small dataset weights” are trained from a pool of 465 images, and the ”Big dataset weights” are trained on the augmented dataset of 5580 images.

Table 2: Description of the mean average precision (mAP) using the original and augmented dataset

Size of dataset	mAP50
Original dataset *(372 train images)*	
Small dataset weights	0.839
Big dataset weights	0.959
Original dataset *(93 test images)*	
Small dataset weights	0.859
Big dataset weights	0.932
Augmented dataset *(1395 test images*	
Small dataset weights	0.660
Big dataset weights	0.874

The mean average precision (mAP) can be seen in Table 2. It can be seen that both networks trained with the original small dataset and the big augmented one performs quite well. However, the network trained with the augmented dataset performs better on every dataset. This has been done in only 50 epochs of training. A longer training time could possibly increase the mAP for both networks. However, due to time constraints, this has not been executed.

7.4 Part conclusion

During the development phase a few different vision algorithm were developed and analysed based on how they perform in segmenting an image and the possibility of taking a method further to actually distinguish the fence from its environment. A lot of time was spend making the Fourier Transformation, seen in subsection 7.2, as well as a screening method for this approach, see subsubsection 7.3.1. It has been shown that this method can be used to find the grid structure in an image and detect a breach in the fence. However, this method proved very unstable with different input images, so each new environment (e.g. different light conditions) has to be very specific fine tuned for it. Therefore this method was not tested and developed further upon, since it is very hard to make it versatile for every possible environment in which a solution might be required.

The problem with the Fourier Transformation method was that when a more bright input image of the fence was analysed, it performed quite poor on segmenting the image, which is a hard requirement for the screening approach. Therefore two other segmentation approaches were developed, a Canny edge detector and U-Net deep-learning segmentation algorithm. The Canny edge detector and convolutional based segmentation, see subsubsection 7.2.3 & subsubsection 7.2.4, had the same issues as the Fourier, making it work well in multiple different environment proved challenging, therefore an idea of using a deep learning to segment the image arose seen in subsubsection 7.2.2. However, to train this network, the method needs to know where the fence is in an image. Which kind of defeats the purpose of this algorithm, because if the fence location was known, then the segmentation of the fence would be complete and could be screened for breaches. Based on this knowledge we discuss that using another deep-learning algorithm could prove a success if chosen correctly. Therefore the final method developed in this project was a Masked Regional Convolutional Neural Network (Masked RCNN), see subsubsection 7.3.2. Where a region of interest is specified when training the network, which is desired, because we generate the training data ourselves with artificial breaches. The location of these artificial breaches is known, thus making it easy to put a region around, which the network will train and learn from.

Furthermore, using such an method, the network can be trained using multiple different environments, since the solution should be applicable in everyone of these. Artificial data was utilised to mimic snow, rain, cloudy day, sunny day and image with different blurs, thus making it very stable and versatile for different environments without having to fine tune it to each new environment.

8 Final Acceptance Test

In this section an evaluation and explanation of the final acceptance test are made:

Weather Conditions:
The day of the final acceptance test, the weather was cold, however the sun was still shining, there was no rain and a light breeze. These conditions are very nice for testing, since it was not too shiny/bright or windy.

8.1 Test Setup

The test was performed as stated in section 2. Two sets of GPS coordinates were manually captured using the GPS module on the drone. This was used as waypoints that the drone should follow along the fence. The path was visualized using QGroundControl's flight log import tool, see Figure 36 along with a image of the drone.

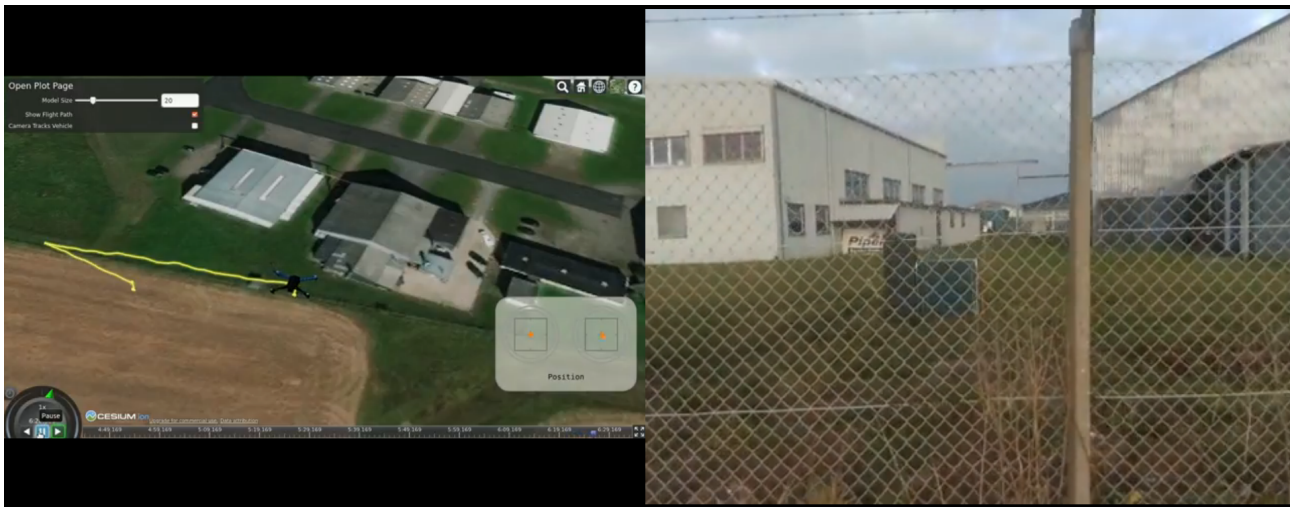


Figure 34: Drone trajectory and point of view

It can be seen that it follows the route as intended. However, by analysing the final acceptance test video, which can be found on GitHub [6], it was noticed that the drones flight height differs a lot. Multiple runs using the same mission was performed and the height changes from test to test, and it was estimated to be due to the fluctuation in GPS data. Therefore it could have been beneficial to utilise another sensor to maintain height to the ground, while also using another sensor to maintain a certain distance to the fence, to make sure that the whole fence is visible by the camera. It can also be seen in the Point of view (POV) video that the drone's distance to the fence fluctuate, and sometimes the bottom part of the fence is not visible, which is not desirable.
The mission was started by our pilot using the controller, the same applies for starting and ending the recording of data. Each image is saved with a GPS, time, and date stamp as a package for further use when alerting the end-user about the position and time of breach location.

8.2 Analysis of test

The mask-rcnn will be used in this section for breach detection which is explained in subsubsection 7.3.2. The video used in this analysis has been taken by the drone and split up into images which is going to be used in the mask-rcnn algorithm for breach prediction. This can be seen in Figure 37.
The drone was able to follow the desired path. However, the fluctuation in height and distance to the fence is not solved yet and is a troublesome problem that needs to be dealt with for this solution to be applicable. The pilot was able to start the mission and recording as intended. The camera was for the most time able to capture the whole fence but sometimes it flew a bit to high, which is unwanted.
As seen in Figure 37, the drone detects the placed custom breaches (cardboard) located on the fence. However, also wrong detections are predicted. Some errors are seen because the algorithm falsely detects some grass in front of the fence in Figure 37i and Figure 37k. The fence needs to be properly cleared of any obstructions in order to locate the breaches correctly. In Figure 37a, Figure 37b and Figure 37c windows are predicted to belonging to a breach class. This is due to the fact that the buildings matches the colour of the fence quite well and hence yields wrong detections. A way to accommodate for this could be to point the camera on the drone downwards to avoid seeing the buildings in the background and hence reduce the risk of wrong detections.

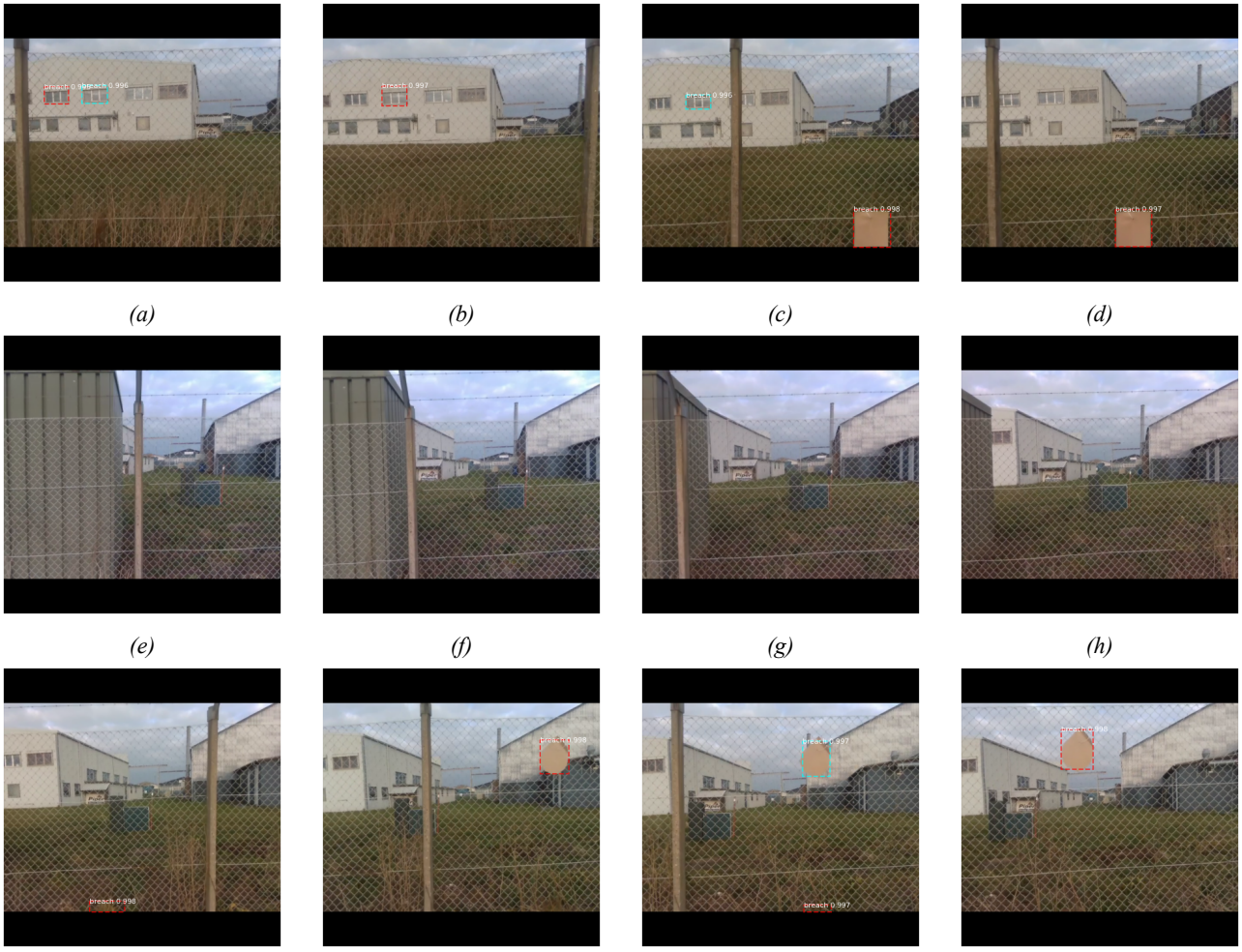


Figure 35: Illustration of some of the images where the mask-rcnn correctly detects the cardboard placed on the fence. However, also minor false detections can be seen

Overall the test went quite well, but with minor difficulties that needs to be addressed going forward.

9 Discussion and Future Work

This section will discuss some of the current features developed in this project as well as some additional thoughts and ideas. However, not all of these are covered in this rapport but would be the next tasks to focus on, for improving the product as a whole. Both in terms of succession rate of the vision algorithm and additional features which would make the product more attractive for potential investors and end-users.

Robotics:
Due to time constraints the setup of the offboard controller node was only done in simulation and not testing on the actual drone, which was undesirable. Furthermore, testing this on the drone would also have allowed us to work with the optitrack system which would have yielded a great estimation of the trajectory and robustness of the offboard controller node, before talking it outside to test it.

Hardware limitation:
The chosen sensor (Intel Realsense Depth Camera D435, see subsection 7.1) had some limitations. It is a great all round sensor also for testing different sensors at a relative low cost. However, to get full HD video at 30 fps a USB 3.0 is needed when using this specific sensor type. However, the companion computer (CC) mounted on the drone, for on-board computations is a Raspberry Pi 3 Model B+ which only support USB 2.0. This resulted in a resolution of only 640x480 if 30 fps was wanted. Upgrading the CC to a Pi 4 could help this issue.

Since, a Pi4 was not available, we should have made an automated path for the drone to follow with a lot of intermediate points where it stops, and snaps a full HD image instead, this would increase the overall flight time, however this is not of importance right now, so this could have been a stable solution to this problem. Furthermore, a manual setup of camera settings by analysing the exposure triangle could have proved beneficial, but this comes with its own issues since these value will change as the environment changes e.g. with different lightings or weather conditions.

Additional sensors for improving robustness:
During the final acceptance test, we identified a potential critical issue with the current system. Using GPS coordinates to make the automated mission for the drone to follow has its issues. This is due to the value of the GPS fluctuating over time. This could have been neglected by using a Ground Control Station (GCS) and a GPS module which supported Real Time Kinematics (RTK). This system potentially has an accuracy below 2cm.
Alternatively one or two additional sensors could be mounted on the drone to maintain a specific ground height as well as distance from the fence. Using these could yield a more stable data capturing sequence of the fence and ensure that the whole fence is completely recorded by the camera for a better end result.

Final Acceptance Test:
To fully test this, the acceptance test should be ran under different weather conditions but due to time constraints as well as corona limitations, only one day of testing was performed.

Additional features:
There are identified some requirements which would be nice to have, given more development time.

A lockable landing box, both to protect the drone from theft as well as weather. This box should be able to open and close automatically when the drone needs to do a perimeter inspection. The team did a initial idea creation for this feature, and would have liked to make the inside landing platform for the drone as an N-fold marker with a smaller ArUco maker inside it, for an automated and precise landing procedure. This feature will further increase the product value as it puts even less strain on the end-user when deployment is needed. This box can also be equipped with a rain and wind sensor to make sure the weather is flyable, otherwise the end-user is alerted and has to manually do a perimeter search. Using these landing boxes makes it easy for larger airports who have a lot more fence to cover, so multiple of these systems could be integrated together to cover the whole fence in one sweep. e.g. having the drone(s) fly from one box to the next by e.g. enabling them at the same time or making use a swarm technology.

Within this box a battery recharging or replacement station must be incorporated for this solution to be fully autonomous.

The mask-rcnn algorithm used to detect breaches in the fence should be trained on a very large (100.000 as a minimum) training set. However, it was quite time consuming to make the set of 7000 which were developed for this project, and it was enough to show a proof of concept. The more versatile the training data, the better the network would be in handling e.g. different lightings and weather conditions. Furthermore to increase effectiveness of the algorithm, a very simple solution is to fly a little higher with the camera tilted abit towards the ground. This will effectively reduce the amount of background noise you have and issues that come with this e.g. see Figure 37.

In case of hardware or software failure the drone, if possible, lands at it current position and alerts the end-user.

Additional camera facing forward could prove beneficial for obstacle avoidance or person detection in case of trespassing.

Building a new drone that is IP54 water resistant to handle light rain.

Integration with the cloud/stream real time data for processing.

Using high-end CC and sensors would make the system able to maintain a much higher fly speed while capturing data of the fence, which would increase the drones total fence clearance range.

10 SORA Assessment

10.1 Step 1: Concept of Operations

The operation consists of a drone provided by SDU employed for fence inspection at the perimeter of HCA Airport. The operation is fully autonomous with a pilot in-charge of monitoring the UAV during all stages of take-off, landing and inspection. The operation will be expectedly carried out 3 time a day.

A1.3 Type of operations:

The type of operation is visual line of sight (VLOS) as the pilot in-charge of the drone needs to monitor and control all the phases of drone operation. The operation is carried out in sparsely populated area of Beldringe Odense Denmark. The airspace is in the vicinity of an active small-scale airport in a fully integrated airspace.

- Maximum Flight height : 20 m
- Inspection height : 1-2.5 m
- Contingency volume : 50 m in all directions
- Ground Risk Buffer : 100 m (Due to vicinity of airport buildings)
- Air Risk Buffer : 100 m (Due to fence being in vicinity of the runway)

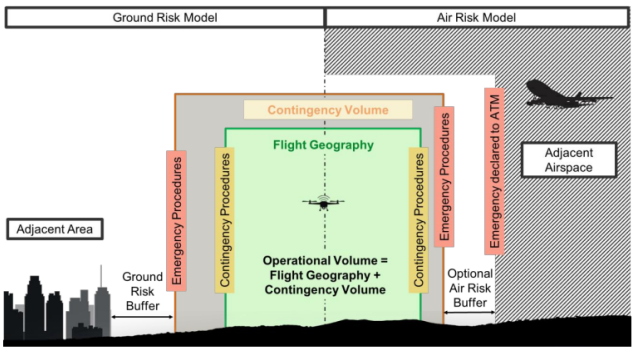


Figure 36: UAV Ground and Air Risk Model[18]

A.1.3.2 Normal operation strategy:

The operation is normally carried out by a certified drone pilot and the complete operation is owned by the airport authority of HCA Airport. The operator is expected to conduct inspection for 3 times a day maximum. Since the operation is done on behalf of airport authorities the pilot is in contact with the control tower for any coordination regarding incoming and outgoing traffic at HCA.

A1.4 Remote crew training:

The pilot in-charge for operating the drone is trained to intervene in case of any failure and deviation from standard inspection path of the drone.

A2.2 UAS Description:

Table 3: UAV Technical Specifications

Weight	2500 g
WxLxH	578 x 578 x 330 mm
Wingspan	832 mm
Material	Plastic/carbon fiber
Airframe	Hexacopter

Table 4: UAV Operational Specifications

Endurance	19 min
Payload capacity	700 g
MTOW	3200 g
Operating temperature	5 - 45 °C

Sensors: The drones sensors comprises of IMU, GPS and Intel RealSense D435 camera.

A.2.3 UAS control segment:

Navigation: The drone uses PX4 flight controller which uses GPS for location and navigation. The secondary means of navigation is manual operation of the drone by the remote pilot. In case of navigation system failure, the flight termination system will activate and try to land the drone using IMU feedback. The drone pilot is also expected to intervene in case such failure occurs. In normal operation, the route is pre-processed and uploaded as a mission to the drone.

A.2.4 Containment system:

In case of the drone deviating from the intended route the pilot is trained to act and take control of the drone to bring it back in the intended operating area or terminate the operation by deploying the flight termination system.

A.2.6 Command and control (C2) link segment:

The communication with UAV is a telemetry link and RC controller link. In case of the failure of these links the UAV will enter in a fail safe mode and initiate landing. Furthermore, in the future another communication link provided by Lorenz Technology called AI Link, which could further reduce the chances of communication failure with the drone.

A.2.9. Safety features:

The drone has all the safety features inherent in PX4 flight controller. In addition to this the monitoring by the remote pilot makes sure that the drone is operating as intended.

10.2 Step 2: Determination of the intrinsic UAS ground risk class (GRC)

The maximum dimension of the drone is 578 mm, maximums speed is 5 m/s and the maximum weight is 3200 g. This gives us an estimate of the typical kinetic energy of the drone: 40 J.

$$KE = 1/2m \cdot v^2 = 40J$$

Also the type of operation is VLOS over a ground area which gives us the **Intrinsic Ground Risk Class of 1** according to table 2 Annex I of the document. [18]

10.3 Step 3: Final GRC determination

The primary mitigation strategy for Ground Collision Risk is active monitoring by the drone pilot. Since the drone is flying at quite low speed 0.5-1.0 m/s at the height of 1-2 meter above ground in a sparsely populated area the need for mitigation of ground risk is deemed unnecessary. The final GRC is determined as below:

- M1: 0 (No strategic mitigation)
- M2: 0 (No mitigation for ground impact)
- M3: 0 (ERP in the form of operator takeover in case of drone malfunction)

Final GRC index is 1 + (0+0+0) = 1

10.4 Step 4: Determination of the initial air risk class (ARC)

The operation is carried out in Flight level less than 500ft AGL in the vicinity of the small-scale operational airport. Therefore, according to figure 4 of the document [18]) the initial ARC of the operation is determined to be ARC-d.

10.5 Step 6: TMPR (Tactical mitigation performance requirements) and robustness levels

In this step, we determine the tactical mitigation required to mitigate the air risk class identified in the previous step. Since the operation is always done in VLOS mode with active monitoring of the drone by the pilot, it is considered as sufficient means of compliance for mitigating the risk of air collision. The tactical mitigation during the fence inspection drone operation is:

- Operation in VLOS to avoid any manned air traffic
- Active collaboration with HCA tower controller during the operation where the operation is paused in case of any incoming/outgoing traffic.

These TMPR (Tactical Mitigation performance requirements) are satisfactory for the operational scenario.

10.6 Step 7: SAIL determination

SAIL determination				
	Residual ARC			
Final GRC	a	b	c	d
≤2	I	II	IV	VI
3	II	II	IV	VI
4	III	III	IV	VI
5	IV	IV	IV	VI
6	V	V	V	VI
7	VI	VI	VI	VI
>7	Category C operation			

Figure 37: SAIL Determination Chart

Residual GRC : 1
Residual ARC : d
SAIL : VI

The SAIL parameter is the consolidated measure of GRC and ARC and demonstrates the level of confidence that the operation will be under control. It further specifies which OSOs (Operational Safety Objectives) will be required for the operation.

10.7 Step 8: Identification of operational safety objectives (OSOs)

The following OSOs are required to be complied with for the operation of the fence inspection drone: Table attached in Appendix 5

10.8 Step 9: Adjacent Area/Airspace considerations

Since the drone is being operated in the vicinity of an operational airport, the complete operation has been designed and developed keeping the airspace considerations in mind.

11 Conclusion

A simulation of the drone and optitrack system were setup and used to develop the initial automated controller for the drones trajectory. This system is fully setup with ROS nodes to be able to utilise the offboard controller node, however no complete test on the actual system was performed due to time constraints.

Waypoints were selected manually besides the fence and connected for the drone to follow. From these points a mission was made utilising QGroundControl as the GUI.

We were able to start the mission as well as the video capture node running on the Pi, directly from the pilots controller, to the receiving unit on the drone. This setup worked quite well for doing multiple tests of the system.

Multiple different vision approaches were analysed and the mask-rcnn algorithm was found to be the best performing one to segment and find a breach in a fence. The network was trained with an augmented dataset of 5580 images which consisted of artificial snow, rain, fog and other kinds of noise in the images to replicate real-life scenarios. The network trained with this augmented dataset yielded quite good results with an mAP of 0.874 in the augmented test set of 1395 images.

This algorithm was able to find all custom made breaches (cardboard) placed on the fence in the final acceptance test. However, also false positives were found as breaches. This suggests that improvements have to be made on training the network with a bigger dataset for future use as well as changes to the camera setup and distance away from the fence.
The SORA process helped us in identifying the challenges in operation and design of the drone. This SORA will be presented to the authorities for the approval of the fence inspection drone operation.

References

[1] Waleed Abdulla. *Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow*. https://github.com/matterport/Mask_RCNN. Accessed: 17-12-2020. 2017.

[2] Waleed Abdulla. *Splash of Color: Instance Segmentation with Mask R-CNN and TensorFlow*. https://shorturl.at/wGV56. Accessed: 18-12-2020.

[3] Amazon. *Intel Neural Compute Stick 2*. https://www.amazon.com/Intel-Neural-Compute-Stick-2/dp/B07KT6361R. Accessed: 20-12-2020.

[4] Jason Brownlee. *How to Train an Object Detection Model with Keras*. https://machinelearningmastery.com/how-to-train-an-object-detection-model-with-keras/. Accessed: 20-12-2020.

[5] *COCO weights*. https://github.com/matterport/Mask_RCNN/releases. Accessed: 18-12-2020.

[6] *EiT group 175*. https://github.com/marcus-grove/EiT_group5. Accessed: 21-12-2020.

[7] Conrad Elektrinik. *Intel D435*. https://shorturl.at/cpsEN. Accessed: 21-12-2020.

[8] Rohith Gandhi. *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms*. https://shorturl.at/nAFN1. Accessed: 17-12-2020.

[9] Dustin Home. *Intel Neural Compute Stick 2*. https://shorturl.at/eivCG. Accessed: 20-12-2020.

[10] Intel. *Intel Movidius Myriad X Vision Processing Unit*. https://www.intel.com/content/www/us/en/products/processors/movidius-vpu/movidius-myriad-x.html. Accessed: 20-12-2020.

[11] Intel. *Intel Neural Compute Stick 2*. https://ark.intel.com/content/www/us/en/ark/products/140109/intel-neural-compute-stick-2.html. Accessed: 20-12-2020.

[12] Alexander B. Jung et al. *imgaug*. https://github.com/aleju/imgaug. Accessed: 17-12-2020. 2020.

[13] Piotr Dollar Kaiming He Georgia G. Kioxari and Ross Girshick. *Mask R-CNN*. https://arxiv.org/pdf/1703.06870.pdf. Accessed: 17-12-2020.

[14] Lumecube. *EiT group 175*. https://shorturl.at/dAFGM. Accessed: 17-12-2020.

[15] A. Walker R. Fisher S. Perkins and E. Wolfart. *Fourier Transform*. http://homepages.inf.ed.ac.uk/rbf/HIPR2/fourier.htm. Accessed: 09-12-2020.

[16] Towards data science. *Turning a Raspberry Pi 3B+ into a powerful object recognition edge server with Intel Movidius NCS2*. https://towardsdatascience.com/turning-a-raspberry-pi-3b-into-an-object-recognition-server-with-intel-movidius-ncs2-8dcfebebb2d6. Accessed: 20-12-2020.

[17] Ross Girshick Shaoqing Ren Kaiming He and Jian Sunk. *Faster R-CNN*. https://arxiv.org/pdf/1506.01497.pdf. Accessed: 17-12-2020.

[18] *SORA*. https://www.easa.europa.eu/document-library/agency-decisions/ed-decision-2020002r. Accessed: 18-12-2020.

[19] Ross Girshick Tsung-Yi Lin Piotr Dollar and Kaiming He. *Feature Pyramid Networks for Object Detection*. https://openaccess.thecvf.com/content_cvpr_2017/papers/Lin_Feature_Pyramid_Networks_CVPR_2017_paper.pdf. Accessed: 18-12-2020.

[20] *UNET Arch*. https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net. Accessed: 18-12-2020.

[21] Xiang Zhangk. *Simple Understanding of Mask RCNN*. https://alittlepain833.medium.com/simple-understanding-of-mask-rcnn-134b5b330e95. Accessed: 18-12-2020.

Section A

Appendices

This will be our appendices

A Table of drone specifications

Technical specifications		Flight specifications	
Weight	2500 g	**Endurance**	19 min
W x L x H	578 x 578 x 330 mm	**Payload capacity**	700 g
Wingspan	832 mm	**MTOW**	3200 g
Material	Plastic/carbon fiber	**Operating temperature**	5 - 45 °C
Airframe	Hexacopter		

Table 2: *Technical and flight specifications for the UAV platform*

Taken from the paper on BlackBoard ”Experts in Team Innovation / Project in Robotics” by: Jes Grydholdt Jepsen, Kristian Husum Terkildsen and Ulrik Pagh Schultz

B OSOs

Table for SORA operational safety objectives compliance:

Table 5: Operational Safety Objectives Compliance

OSO number	Technical issues with the UAS	SAIL VI required level	Method of compliance
OSO#1	Ensure the UAS operator is competent and/or proven	H	The airport authority as an operator will ensure compliance
OSO#2	UAS manufactured by competent and/or proven entity	H	The drone will be manufactured, tested and certified by the competent authority
OSO#3	UAS maintained by competent and/or proven entity	H	The airport authority and Lorenz technology will ensure the optimum maintenance of the drone
OSO#4	UAS developed to authority recognised design standards	H	This will be ensured that the drone comply with required design standards
OSO#5	UAS is designed considering system safety and reliability	H	The features and drone operation will be designed and developed keeping in mind the safety aspects
OSO#6	C3 link performance is appropriate for the operation	H	C3 Link performance is appropriate for this operation
OSO#7	Inspection of the UAS (product inspection) to ensure consistency with the ConOps	H	The drone operator will ensure inspection of UAS
OSO#8	Operational procedures are defined, validated and adhered to	H	The workflow and operational procedures will be developed and provided to the operator
OSO#9	Remote crew trained and current and able to control the abnormal situation	H	Training to the remote crew will be provided to enable safe operation of the drone
OSO#10	Safe recovery from a technical issue	H	The drone has features to recover in case of technical failures
Deterioration of external systems supporting UAS operations			

OSO#11	Procedures are in-place to handle the deterioration of external systems supporting UAS operations	H	External System include the Ground control station and RC controller. The drone will automatically land in case of malfunction of external systems
OSO#12	The UAS is designed to manage the deterioration of external systems supporting UAS operations	H	This is builtin in the flight controller
OSO#13	External services supporting UAS operations are adequate for the operation	H	External services are adequate for the operation
Human Error			
OSO#14	Operational procedures are defined, validated and adhered to	H	Adherence to operational procedures will be ensured by the operator
OSO#15	Remote crew trained and current and able to control the abnormal situation	H	Crew will be trained to operate the drone and handle abnormal situations
OSO#16	Multi-crew coordination	H	Not required in this operational scenario
OSO#17	Remote crew is fit to operate	H	This will be ensured by the drone operator
OSO#18	Automatic protection of the flight envelope from human error	H	This will be ensured by the pilot in-charge of operating the drone
OSO#19	Safe recovery from human error	H	This will be achieved through coordination with the airport control tower and automatic landing features
OSO#20	A human factors evaluation has been performed and the human machine interface (HMI) found appropriate for the mission	H	The HMI being used is standard QGroundControl which makes the evaluation unnecessary for this operational scenario.
Adverse operating conditions			
OSO#21	Operational procedures are defined, validated and adhered to	H	The drone will only be operated in favorable weather conditions.
OSO#22	The remote crew is trained to identify critical environmental conditions and to avoid them	H	The drone operator will be trained to avoid flying the drone in adverse environmental conditions

OSO#23	Environmental conditions for safe operations are defined, measurable and adhered to	H	The limits on operation of drone in different environmental conditions will be provided to the drone operator and implemented by the airport authority
OSO#24	UAS is designed and qualified for adverse environmental conditions	H	Since the UAS will only operate in favorable weather conditions, this design and qualification is not necessary.