MSc thesis in Geomatics

Localising objects with drones: A case study on the localisation of fisher boats in restricted areas

Lisa Yvette Geers

2023

MSc thesis in Geomatics

Localising objects with drones: A case study on the localisation of fisher boats in restricted areas

Lisa Yvette Geers

January 2023

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of Master of Science in Geomatics

The work in this thesis was carried out in cooperation with:

Geo-Database Management Centre
Delft University of Technology

Innovation and Portfolio Development
Esri Nederland

Supervisors:	Dr.ir	Martijn Meijers
	Dr.	Azarakhsh Rafiee-Voermans
	Drs.	Niels van der Vaart
Co-reader:	Ir	Edward Verbree

Abstract

Due to safety or preservation reasons, certain objects or areas need to be inspected regularly. Currently, the inspection of objects is mostly done in-person, which is labour-intensive and not very effective. With the use of drones, areas and objects can be inspected from new angles at a much faster rate. To effectively monitor these objects with drones, the position and location needs to be extracted from drone data in real time.

In this thesis, a case study is done on the localisation of fisher boats in restricted areas. Several components are integrated to create a prototype. The pretrained YOLOv3 detection model is trained on acquired nadir boat images, which makes it able to predict the bounding boxes of boats on images captured with drones. A positioning algorithm is constructed, which calculates the geographical coordinates from the pixel coordinates for images taken both in a nadir and an oblique angle. A real time connection is constructed between the drone and the prototype. This is done by creating a connection with Google Drive with the drone controller and the prototype. The positioned polygon bounding boxes are localised using a real time dashboard, which visualises the bounding boxes in a map with other relevant layers.

The results indicate that the performance of the components and prototype as a whole are satisfactory for this use case. To deploy this prototype in other object localisation use cases, it is recommended to train the pretrained model further, use a drone with more accurate equipment and run the prototype on the drone controller.

Acknowledgements

In this chapter I would like to thank multiple people who contributed to this thesis. Firstly, I want to express my gratitude to my two mentors Martijn Meijers and Azarakhsh Rafiee for regularly meeting with me, always providing useful feedback and pointing me in the right direction when needed. I also want to thank Edward Verbree for taking the time to be my co-reader.

I want to thank Niels van der Vaart for being an excellent mentor at Esri Nederland and giving me multiple opportunities to develop myself in the professional field. Additionally, I would like to thank my colleagues at Esri Nederland for always being open for questions and making me feel very welcome.

Furthermore, I would like to express my gratitude to the individuals from the NVWA involved in my thesis. I would especially like to thank Rob Broekman for providing me with the necessary contacts and equipment to carry out my thesis. I would also like to express large gratitude to Mike van Dooijeweert for flying the drone to gather data for my thesis and letting me borrow his equipment for my research.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AI	Artificial Intelligence	1
ANN	Artificial Neural Network	7
AP	Average Precision	43
API	Application Programming Interface	18
BVLOS	Beyond Visual Line Of Sight	6
CNN	Convolutional Neural Network	7
CRS	Coordinate Reference System	12
DEM	Digital Elevation Model	13
FMV	Full Motion Video	xii
FOV	Field of View	6
GIS	Geographical Information System	10
GNSS	Global Navigation Satelite System	31
GPS	Global Positioning System	1
IoU	Intersection over Union	xi
KPI	Key Performance Indicator	22
LiDAR	Light Detection And Ranging	5
MPP	Meters Per Pixel	xi
NVWA	Nederlandse Voedsel en Waren Autoriteit	1
RTK	Real Time Kinematic	53
SAR	Synthetic Aperture Radar	9
SfM	Structure from Motion	12
SLAM	Simultaneous Localization and Mapping	14
UAV	Unmanned Aerial Vehicle	5
VOC	Visual Object Classes	36
WFS	Web Feature Service	31
WGS 84	World Geodetic System 1984	19
XML	Extensible Markup Language	30

1. Introduction

Fishing is an important part of the Dutch economy and culture [Salz et al., 2008]. However, fishing can be harmful for several reasons, so it is not permitted to fish in all Dutch waters. The Dutch inland waters know a number of restricted areas where no fish is allowed to be caught because of the high dioxin degree in the water, which could have harmful effects when this fish is consumed [Leeuwen et al., 2002]. Moreover, both in the Dutch inland waters and in the sea, there are a number of areas where fishing is restricted due to the laws of the Natura 2000 agreements [van Oostenbrugge et al., 2010]. In these areas, fishing is restricted to conserve habitats and species named in the EU Birds and Habitat directives [Pedersen et al., 2008]. The monitoring of fishers and fisher boats in restricted areas is done by the Netherlands Food and Consumer Product Safety Authority, in Dutch Nederlandse Voedsel en Waren Autoriteit (NVWA). Inspections on boat equipment and fish catch are currently done by boat. Naturally, this is very labour-intensive, as this requires at least two inspectors per boat. Additionally, using this method, the checking speed is not very high. Also, on a boat, not every corner of the area may be visible. This results in most of the areas being left unchecked, large parts of the time. Nevertheless, in 2016, 80 cases of illegal fishing in inland waters were reported on eel fishing alone by the NVWA [Bos, 2018].

To increase efficiency and accuracy, a new method needs to be developed that can automatically find fishing boats in restricted areas. By using drones with imaging and Global Positioning System (GPS) equipment, fishing boats can be automatically detected with Artificial Intelligence (AI) and the boat position can be calculated using the drone GPS, followed by the localisation using a map and relevant layers. In this way, more fishing boats in illegal areas are expected to be found. This method needs to be carried out in real time, because visual inspection by inspectors is still needed to fine or prosecute offending fishers. Moreover, the real time aspect is also important to extend this method to other use cases, for example for the search and rescue of drowning victims. By developing a general, adaptable method for the real time localisation of objects with drones, this method can be directly used for a variety of other use cases. For example, keeping track of animals and plant types in forests or the localisation of drug waste. These are all inspection tasks carried out by the NVWA that could possibly be automated with such a method.

The automatic detection of objects with AI remains a challenging task, but has been researched by many. Recently, deep learning has been making major advances in the AI community, making it more efficient and accurate than most other techniques [LeCun et al., 2015]. Detection of ships on imagery with deep learning has been done mostly using 2-dimensional satellite images (i.e. Apoorva et al. [2020], Ciocarlan & Stoian [2021] and Voinov [2020]). Zhang et al. [2019] developed a method to track and localise objects in 3D with drones and Prayudi et al. [2020] researched a method to detect and localise fisher boats and derive hull plates from drone images. No research has been done to integrate detection, positioning and localisation in real time, which is a crucial aspect to catch offending fishers in the act.

The aim of this research is to develop a prototype for an integrated method that can detect, position and localise objects, in this case fishing boats, in real time with the use of drones. The focus areas of this research are the Dutch inland waters, as data acquisition and testing is more accessible there compared to on sea. Drone image data has been acquired from several inland water areas in the Netherlands. Using this data, a prototype is developed in Python using a variety of libraries that can detect, position and localise fisher boats in real time. To evaluate the developed method and prototype, several tests will be conducted in the field to gather necessary drone image data.

1.1. Research questions

The objective of this research is to develop a prototype for an integrated method to localise objects with drones. For the use case of this research, the prototype will be constructed with the goal of catching fishing boats in restricted waters with the use of drones. Boats need to be detected on images retrieved from a real time drone image stream, and the exact position of these boats needs to be derived to be able to determine if they are in restricted waters or not. The areas of interest are the Dutch internal waters, because there are a variety of restricted areas and data can be collected there more easily than on the sea.

To fulfil this objective, the following research question is defined:

To what extent can drones be used to localise objects in real time?

To answer the main question, the following sub-questions are defined:

- How can deep learning be used to detect objects on drone images?
- How can detected objects be automatically positioned in a geographical coordinate system?
- What hardware and software is needed for this method to be carried out in real time?

1.2. Scope and challenges

The challenges of this research lie in the positioning and localisation of the detected objects, as well as the real time aspect of this method. For this reason, besides training the model, no time will be spent on the extensive improvement of the detection accuracy. Additionally, determining if detected boats are fishing boats will be out of the scope of this research. Also, it will not be determined if people on the boat are fishing or not. Because a visual inspection of the images by an inspector is still necessary, detection of the boat type or the act of fishing is not crucial to implement and is thus left out of the scope.

A challenging aspect of the real time component is that a careful consideration has to be made between accuracy and speed of the prototype. These decisions are mainly made in the choice of the detection model architecture and type of positioning algorithm. Because the real time functionality is a big aspect of the project and visual inspection is still needed, speed will be favoured over accuracy in most cases.

1.3. Thesis overview

This graduation project document has the following outline:

- In Chapter 1, an introduction to the research project and its aim are given. Additionally, a description of the research questions, scope and challenges of this project are given
- Chapter 2 gives an overview of the theoretical background of several steps of this research. Also, a description of the related work is given.
- Chapter 3 explains the used methodology. An overview of this methodology is given. Moreover, the different steps in the research are outlined in separate sections.
- Chapter 4 provides the implementation details, which describes the pseudocode, used data, used tools and the ran experiments.
- Chapter 5 presents the results of the research. The results from the experiments and evaluation are visualised here.
- Chapter 6 describes the conclusion of this research as well as the future work recommendations. Also, the research outcomes are thoroughly discussed and the limitations are illustrated.

2. Theoretical background and related work

In this chapter, the relevant literature for this research project is presented. The related research that this project builds on, will also be addressed. In section 2.1, investigations and relevant findings on the use of drones in research are described. Section 2.2 outlines the theory behind deep learning and object detection. Additionally, related research on object detection is specified. Following, in section 2.3 relevant theory on the positioning of objects is detailed and relating research is reported on. Finally, a summary of the theoretical background and related work is given, where the research gap and the components that this research will build on are addressed.

2.1. Data acquisition with drones

Recently, drone technology for remote sensing has developed swiftly and has been increasingly used in military defence, monitoring, surveying, mapping, and disaster and emergency response [Yang et al., 2022]. Drones are Unmanned Aerial Vehicle (UAV)s, which are robots that fly remotely or autonomously and do not carry a human operator. Progress in fabrication, navigation, remote control and power storage increased the creation of a wide variety of drones fit for numerous situations [Hassanalian & Abdelkefi, 2017].

Most drones carry a camera, which contains both imaging and video capture capabilities. A GPS receiver to connect to GPS satellites is also common equipment present on a drone. Some drones even have Light Detection And Ranging (LiDAR) scanning capabilities. Over the years, the quality of drone cameras have increased. This results in very high resolution images, in which details can be seen that cannot be found in satellite imagery [Voinov, 2020], which has a lower resolution. Additionally, drones are more agile in data collection than satellites or aeroplanes. Compared to aeroplanes, gathering local data of a small area with drones is less expensive, which could lead to more data in smaller areas being able to be collected.

When using drones for data collection, one can set a predefined path, or fly on the go with a controller. Using a predefined path, a video, or images are taken every few seconds. The result is a collection of images or a video covering the whole area. When images are collected, these images can highly overlap. Several methods exist to remove this overlap, for example the method by [Dhanda et al., 2018] where they use the metadata to determine redundant overlap. When removing overlap, the amount of redundancy needs to be carefully chosen. To carry out 3-dimensional reconstruction, overlap is necessary. For other use cases, when only one image of the scene is required, all overlap can be removed. Another advantage of drones is that drones can collect data under circumstances when satellites are not useful, for example when the area is very cloudy. Although the GPS metadata is expected to be less accurate in this case, due to the fact that a connection with the GPS satellites is more difficult to make. The data collection times are very flexible for drones, a surveyor can for example decide to not sent out a drone under bad weather conditions [Joyce et al., 2019]. In the Netherlands, there are no restrictions to fly under any weather conditions. However, there exist some no-fly zones and the drone has to be kept in the visual line of sight at all times when flying. This also means that flying in the dark is not allowed. However, the NVWA is exempt from several no-fly zones to perform drone monitoring in these areas. Additionally, an exemption from the line of sight rule is also expected in the future. Currently, flying Beyond Visual Line Of Sight (BVLOS) is still in the testing phase in the Netherlands[1].

Collection of drone data over water bodies knows some additional difficulties compared to flying over land. Two factors that affect the image quality when investigating submerged features are sun glint and subsurface illumination [Mount, 2005]. However, these factors can also have an influence when investigating non-submerged features, because illumination and refraction differences between the water and objects on water can also influence detection accuracy. Sun glint can be avoided with calculated flight planning. The used parameters to calculate this are solar position, flight direction and camera angle [Joyce et al., 2019]. Figure 2.1 shows that sun glint will be significant when the solar position is at a high enough angle that it reflects into the drone camera Field of View (FOV). This means that when the sun is at a low angle, sun glint will be minimal. However, this would mean that data acquisition during the middle of the day would lead to a much lower detection accuracy. A solution for this would be to plan the flight plan as such that the drone is flying directly towards or away from the sun azimuth [Joyce et al., 2019].

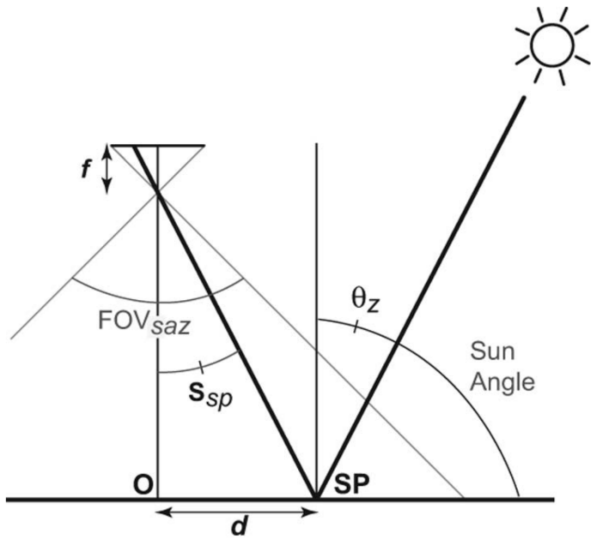


Figure 2.1.: "The angle S_{sp} at the focal point between the image centre O and the solar specular point SP is equal to the solar zenith angle θ_z. θ_z is complementary to sun angle (that is, 90 - z) and FOV_{saz} is the image FOV in the direction of the solar azimuth. Distance in image units d can be calculated with the focal length f and θ_z." From Mount [2005]

[1]https://unmannedvalley.nl/en/news/press-release-first-bvlos-corridor-in-the-netherlands/

2.2. Deep learning

The interest and innovations in machine learning have exploded over the last decade [Patterson & Gibson, 2017]. Machine learning is a type of artificial intelligence where machines are able to learn automatically and improve from experience without being explicitly programmed to do so [Nan, 2022a]. One very effective and accurate machine learning type is the Artificial Neural Network (ANN), which is based on neurons of the biological nerve system. ANN algorithms consist of a collection of connected nodes. The basic structure can be seen in Figure 2.2. It consists of an input, hidden and output layer. The input is a multidimensional vector, for example one containing image pixel values. The hidden layers make decisions based on the previous layer. In the process of learning, the effects of these decisions on the output are evaluated and the decisions are improved consequently. An architecture with multiple stacked hidden layers is called deep learning [O'Shea & Nash, 2015].

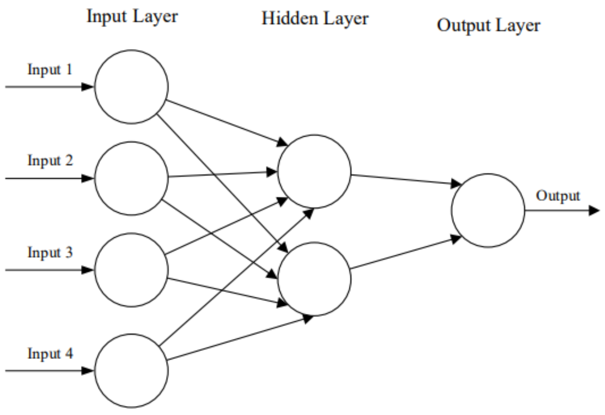


Figure 2.2.: Simple artificial neural network. From O'Shea & Nash [2015]

An advantage of deep learning over other machine learning methods is automatic feature extraction. Traditionally, features were created manually, which is very labour-intensive. A feature in this context is any value identifies certain attributes of the input data [Patterson & Gibson, 2017]. Features in images processing may for example be points, edges or objects in the image.

2.2.1. Convolutional Neural Networks

A Convolutional Neural Network (CNN) is an artificial neural network with an architecture created for image processing. CNNs contain a variable set of modules that transform the input at one level into a representation of a higher and more abstract level. With enough of these transformations, complex functions can be learned. The key aspect is that these transformations are not designed by humans, but are learned by the computer from training data [LeCun et al., 2015]. The structure can be seen in Figure 2.3. CNNs consist of three types of hidden layers [O'Shea & Nash, 2015]:

- Convolutional layer: Use filters to create feature maps, which indicate the presence of detected features in the input data.
- Pooling layer: Downsamples the spatial dimensionality of the input of the previous layer, thereby reducing the number of parameters.
- Fully connected layer: Produces class scores from the activations of the previous layer.

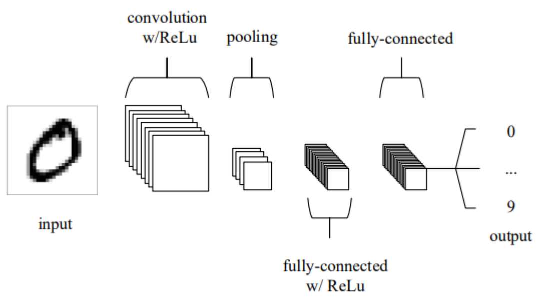
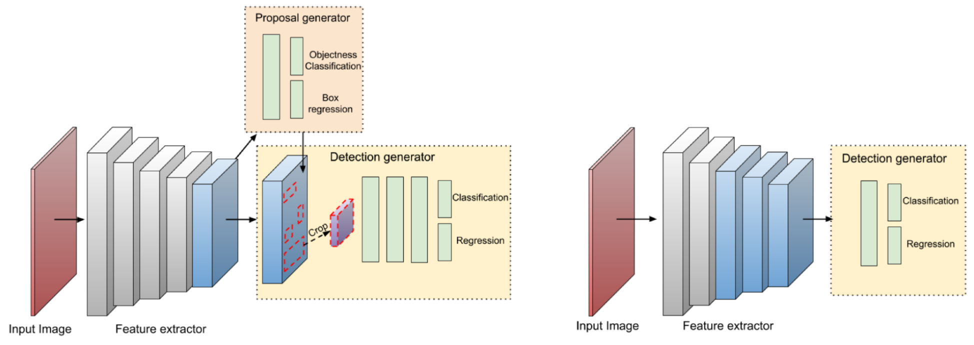


Figure 2.3.: Simple convolutional neural network. From O'Shea & Nash [2015]

2.2.2. Architectures

There are a variety of different deep learning architectures developed over the years. These can be divided into single stage and two stage algorithms. In single stage, there is only one pass through the network at inference, which makes detecting objects with this algorithm faster. Examples are YOLOv3 [Redmon & Farhadi, 2018], SSD [Liu et al., 2016] and RetinaNET [Lin et al., 2017]. In two stage algorithms, the classifications and bounding boxes are firstly proposed using a region proposal network and these are refined at a second pass. Examples include Faster R-CNN [Ren et al., 2017] and Mask-RCNN [He et al., 2017]. This makes these types of algorithms slower but more accurate at inference. Users thus have to consider what architecture to use for what type of problem. In this research, run time is most important, hence why a one stage architecture is used. One stage algorithms treat classification as a regression problem. It uses a single pass through a CNN to predict both the bounding boxes and the classes of object in the input image, similar to image classification. Figure 2.4 shows the most significant differences in architecture between the one-stage model RetinaNet and two-stage model Faster-RCNN. As can be seen, the inference process is divided in region proposal stage and a classification stage for the two-stage architecture. In the one-stage architecture, objects are directly detected [Carranza-García et al., 2020].

Between the two type of stages, there exist a variety of different architectures, which have quite a large difference in speed and accuracy amongst them. The Faster-RCNN architecture consists of a multitask learning procedure, which combines bounding box regression and classification for inference. It extracts high-level features from input images using a convolutional backbone. The architecture is two-staged, which consists of a region proposal network and a header network. The RetinaNet object detector is based on the SDD architecture. This architecture performs inference using a single feed-forward convolutional network. Because this architecture does not use per-proposal computations, it has a higher inference speed. To increase inference accuracy, it consists of feature maps of different resolutions.



(**a**) Two-stage Faster R-CNN (**b**) One-stage RetinaNet

Figure 2.4.: Example one-stage and two-stage architectures. From Carranza-García et al. [2020]

The YOLO architecture has been extensively used for real time inference. This architecture divides the input image into cells with the use of a grid. If the centre of an object falls into a certain grid cell, this cell is responsible for the detection of that object. Using this structure, the YOLO architecture achieves faster inference rates than other architectures [Carranza-García et al., 2020].

2.2.3. Research on the detection of objects

Because CNNs are structured to process image data, they can be used well to detect objects from drone data. The classification of objects on drone images has been researched before to find palm trees by Aburasain et al. [2021] and Htet & Sein [2021]. Aburasain et al. [2021] used the YOLOv3 object detector to detect palm trees of different resolution, sizes, spread and degree of overlap. They stated that the benefit of using drone images instead of satellite images is the increased image resolution, which leads to a more accurate classification.

Detecting ships with the use of deep learning has been researched several times, for example by Apoorva et al. [2020], Chang et al. [2019] and Voinov [2020]. Input images in these researches are imagery data collected by satellites. Apoorva et al. [2020] uses the TensorFlow deep learning library to detect ships on satellite images. Voinov [2020] proposes a method to detect vessels from optical satellite images using images from three different satellites. It is mentioned that the image resolution has a large impact on the detection accuracy. Chang et al. [2019] proposes a method to detect ships on Synthetic Aperture Radar (SAR) imagery. By using the YOLOv2-reduced deep learning architecture, a significant decrease of the computation time and a detection accuracy of 90% was achieved. Nonetheless, none of these methods used drone data and none implemented a real time or localisation functionality.

Prayudi et al. [2020] designed a surveillance system framework that can detect and localise ships and derive their hull plates. They used 2450 images for training and stated that ship detection can be challenging due to the difficulty of collecting training images. Nonetheless, the found average matching precision was 96%. The ship coordinates were found by taking the x and y image coordinates from the object bounding box and converting them to latitude and longitude coordinates. It is not clear what algorithm or library was used for this conversion.

2.3. Object Positioning

2.3.1. Terminology

It is important to note the difference between the terms localisation and positioning. Nominally, the terms are interchangeable [Groves, 2013]. However, they are commonly used to indicate two particular concepts. The term positioning is commonly used quantitively, for example to find numerical coordinates. Localisation is expressed qualitatively as the context of the position. A Geographical Information System (GIS) can match locations to positions [Groves, 2013]. In this research, positioning is used to denote the determination of the object coordinates. The term localisation is used to put the object in its geographical context.

2.3.2. Positioning

To position objects, position data is needed. In this research, the drone GPS and drone image metadata is used to position objects from images. The use of images to derive information from objects in these images is described by the science of photogrammetry [Schenk, 2005]. Images are created with a camera, which can be modelled in several levels of abstraction. One way is the camera lens model (Figure 2.5). Here f is the focal length. The light rays that originate from the world system are refracted by the lens so that they converge at a single point, the focal point.

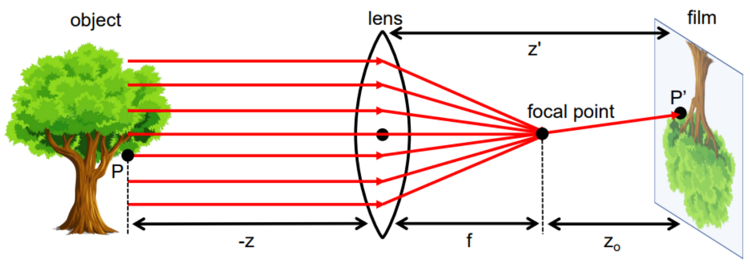


Figure 2.5.: Camera lens model. From Nan [2021a]

To find the geographical coordinates of an object in an image, one has to consider three coordinate systems. The one of the image plane and the the camera system and the world coordinate system [Schenk, 2005]. To calculate how a point M in the world coordinate system maps to a point m' on the image plane, the camera intrinsic and extrinsic parameters can be used. There exist five intrinsic parameters: two for focal length (α_x, α_y), one parameter for the skew coefficient between the x and the y-axis (γ) and two principal point offset parameters (u_0, v_0). Together, they form the intrinsic matrix K, as can be seen in equation 2.1. The extrinsic parameters consist of a rotation matrix R and a translation vector t, as can be seen at equations 2.2 and 2.3. The extrinsic parameters form the pose of the camera in relation to the world coordinate system. As can be seen at 2.4, the intrinsic and extrinsic parameters together form the camera matrix C. Equation 2.5 displays how a point M in the 3D world coordinate system maps to a point m' in 2D on the image plane [Nan, 2021a].

$$K = \begin{bmatrix} \alpha_x & \gamma & u_0 & 0 \\ 0 & \alpha_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.1}$$

$$R = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,1} & r_{3,2} & r_{3,3} \end{bmatrix} \tag{2.2}$$

$$t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \tag{2.3}$$

$$C = K[Rt] \tag{2.4}$$

$$m' = CM \tag{2.5}$$

Figure 2.6 shows the coordinate systems involved in the positioning problem. The extrinsic parameters t and R are used to map the camera's coordinate system C to the world coordinate system O. Coordinate system c is the image coordinate system. Using the camera specifications and the extrinsic parameters, we can calculate how a point M in the world coordinate system maps to a point m on the image plane [Heikkila & Silven, 1997].

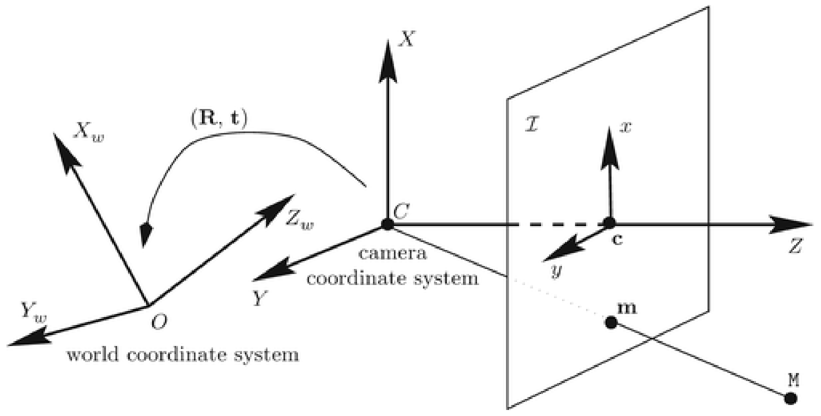


Figure 2.6.: Coordinate systems positioning From Xu & Zhang [1996]

There exist different world coordinate systems to map the camera coordinate system from. A Coordinate Reference System (CRS) uses numbers or coordinates to uniquely determine the position of a point in a Euclidean space [Tiberius et al., 2021]. CRSs can be divided into Cartesian coordinate systems and geographical coordinate systems. These are visualised in Figure 2.7. The geographical coordinates are represented by λ, ψ and r and the Cartesian coordinates are represented by X, Y and Z. In a Cartesian coordinate system, the coordinates are defined as the distance from an origin in X, Y and Z. To represent points on the surface of the earth, a 3-dimensional Cartesian coordinate system with the origin at the centre of mass of the earth using three coordinates would need to contain seven digits before the decimal point. A geographical coordinate system is more convenient to use to represent points on the surface of the earth [Tiberius et al., 2021]. In this system, coordinates are curvilinear in a unit of degrees and are defined on a sphere or ellipsoid approximating the earth's surface.

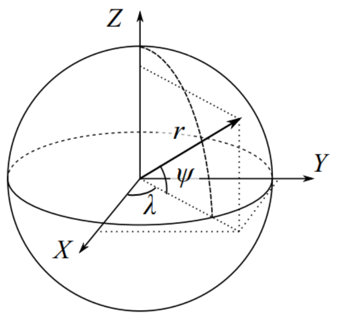


Figure 2.7.: Schematic view geographical and Cartesian coordinate system. From [Tiberius et al., 2021]

2.3.3. Positioning methods

One of the most fundamental problems in multiple view geometry is triangulation, which is the process of retrieving a 3-dimensional coordinates in the world coordinate system from its two local coordinates in two 2-dimensional images [Hartley & Zisserman, 2003].

The triangulation problem can be seen in Figure 2.8. Here, R and T are the camera extrinsic parameters and K and K' are the intrinsic parameter matrices of the two images. P is the 3-dimensional point that should be found and p and p' are the corresponding 2D points on the image planes. O_1 and O_2 are the camera origins. In theory, P could be calculated through the intersection of l and l', the two lines of sight. However, because observations are noisy and the camera parameters are not precise, finding this intersection point is problematic Nan [2021b].

A solution for this problem could be using more than two images. The Structure from Motion (SfM) method uses multiple views to determine the geometry of the scene and the camera parameters simultaneously. It consists of determining corresponding features in images and subsequently determining the motion of a feature in each image Nan [2021b]. Mlambo et al. [2017] used SfM on drone image data to monitor greenhouse gas emissions in forests.

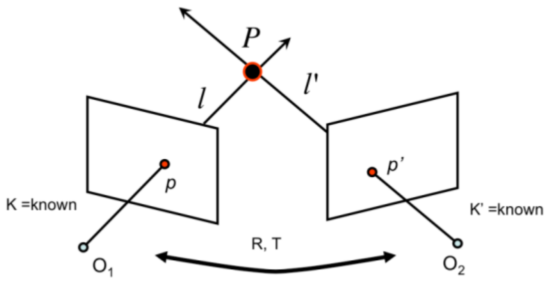


Figure 2.8.: Triangulation problem with two images. From Nan [2021b]

However, determining if two objects in two different images depict the same object, adds computational complexity, which is undesirable due to the real time aspect of this research. This arises the question whether positioning can be carried out by using only one image. Schenk [2005] states that the applications of single photographs in photogrammetry is limited, they cannot be used to reconstruct the object space. The reason for this is that even though the exterior parameters may be known, one cannot determine ground points without knowing the scale factor of each bundle ray. However, the goal in this research is to find the geographical coordinates of the bounding box of the object, not to reconstruct the whole image scene.

Bozzini et al. [2012] developed a tool to georeference oblique image data and extract vector features from these images. The needed inputs for this tool are images, a Digital Elevation Model (DEM) of the area and ground control points. Ground control points are points where the position is known for both the image coordinate and the 3-dimensional world coordinate on the DEM. Using this method, a ray is shot from the camera through each ground control point pair. The camera matrix is then defined and calibrated. The world coordinates can then be calculated for each pixel [Steiner, 2011]. The principle can be seen in figure 2.9. The ground control points are shown in green.

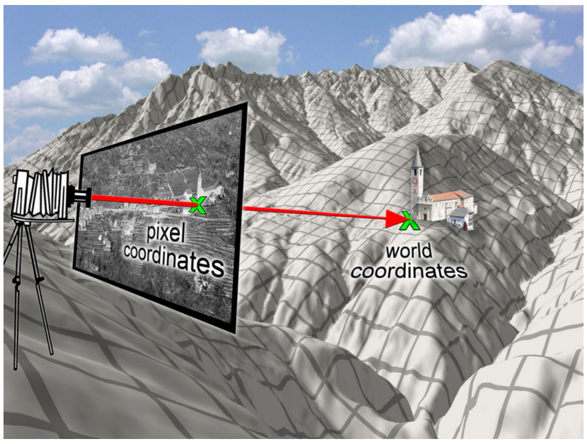


Figure 2.9.: "The monoplotting principle". From Bozzini et al. [2012]

2.3.4. Research on the positioning of objects

Bodensteiner et al. [2015] proposes a method to accurately georeference community drone photo and video data to register to 3-dimensional LiDAR data. This method relies on the SfM and Simultaneous Localization and Mapping (SLAM) techniques, which are combined with appearance and structure matching based on LiDAR data. The method can produce drift free drone image overlays at a speed of 30 frames per second with an average error of 0.4 meters. This method uses existing open drone data for processing and testing. However, it is not runtime optimised and real time execution has not been implemented.

Zhao et al. [2019] developed a framework for detecting, tracking and geolocating moving vehicles with drones. The detection of vehicles is carried out by the YOLOv3 detection model. Processing is done on a microcomputer mounted on the drone. The drone data used in this research is a video stream. This research performs real time detection and positioning, but does not localise the objects.

2.4. Summary

In this chapter, the relevant theory and research is described. It is found, that using drones for monitoring and data collection is advantageous due to the high flexibility and resolution. However, drone data acquisition can be inhibited due to weather conditions and flight restrictions. Object detection with deep learning is a topic that has been widely researched the past few years. There exist multiple different architectures. As found by other researches, the YOLOv3 architecture is the best fit for real time inference. Because it has been researched many times before, the use of deep learning for object detection is not the research goal. Instead, the aim is to research how this component can be integrated into the prototype most efficiently and accurately. For the positioning of objects, several methods have been specified. Because the method needs to be fit for real time use, little computational complexity is required. Reconstruction methods like SfM are thus not optimal. The method used in this research can be seen as a variation on monoplotting where one ray is shot through the middle of the image. Many of the researches that implement object detection and positioning, either do not perform this in real time or do not localise the results in real time. The different components of this research, the detection of objects, positioning, real time connection and localisation have each been researched before. This research aims to be innovative by integrating these components into one prototype.

3. Methodology

In this chapter, the methodology of this research is illustrated. Figure 3.1 shows an overview of the used methodology to construct the prototype. In section 3.1 de focus area and the data acquisition method is described. Thereafter, in section 3.2, the investigation to find and implement the deep learning model to detect fisher boats is illustrated. In section 3.3, the construction of the positioning algorithm for cameras with a nadir angle is presented. Following, in section 3.4 the construction of the real time connection with the drone is specified. In section 3.5, the developed dashboard that is constructed to localise the found objects is illustrated. After, in section 3.6, the prototype construction and improvement is described. The prototype improvement consists of the development of a positioning algorithm for objects on oblique images and the training of a deep learning detection model, respectively described in subsections 3.6.1 and 3.6.2. Finally, the validation method of the prototype is reported in section 3.7. In this section, a description is made how the three main components of the prototype are validated, which are the detection models, positioning algorithms and real time connection with the drone. The validation of each component is described in a separate subsection.

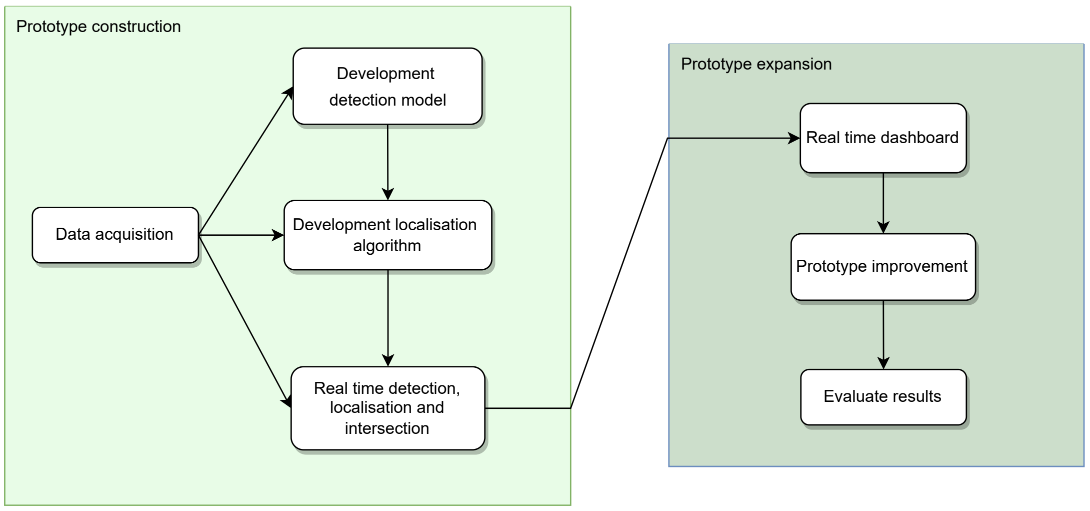


Figure 3.1.: Overview method of prototype construction

3.1. Focus area and data acquisition

As mentioned before, the focus area of this research area will be the Dutch inland waters. The Netherlands consist of a variety of rivers and freshwater lakes. Numerous areas are protected by the Natura 2000 directive. Figure 3.2 shows a map of these areas in the Netherlands. The different colours indicate the directives of the areas. *HR* is the habitat directive and *VR* is the bird protection directive. Additionally, in many areas, fishing is forbidden due to the high Dioxin degree in the water. Because drones have a limited flight time (usually under an hour) and the law enforces pilots to keep the drone in sight while flying, the choice was made to choose the inland water as the study area.

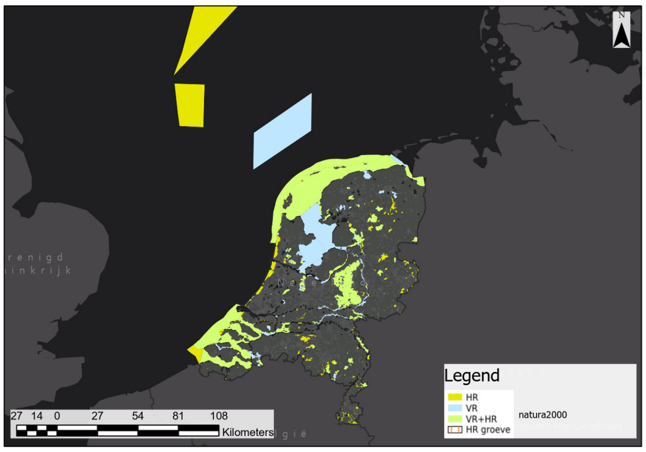


Figure 3.2.: Natura 2000 areas in the Netherlands

Data acquired for this research is done by drone. Data was collected at two instances with the DJI Mavic 2 Enterprise Advanced. Figure 3.3 shows an image of this drone. Because it contains a GPS receiver, the captured images indicate the drone position in the metadata. Data collection was done by inspectors of the NVWA, because a drone pilot licence is needed to fly drones heaver than 250 grams in the Netherlands [1]. For each of the two data acquisition instances, a flight plan was created to communicate the research area, research goal, necessary equipment and licences with all persons involved.

Figure 3.3.: DJI Mavic 2 Enterprise Advanced drone

[1] https://www.government.nl/topics/drone/applying-for-a-drone-pilot-licence

The first data acquisition instance took place at the Den Oever harbour in North-Holland, the Netherlands. Drone images were collected between 13.23 and 14.19 at July 15, 2022. A total of 290 images were collected at a nadir angle at a fixed pattern with a predefined flight path. The weather conditions were lightly cloudy, which lead to minimal sun glint. Figure 3.4 shows an example from this dataset. Only drone images were recorded and no GPS position of the photographed boats was measured.

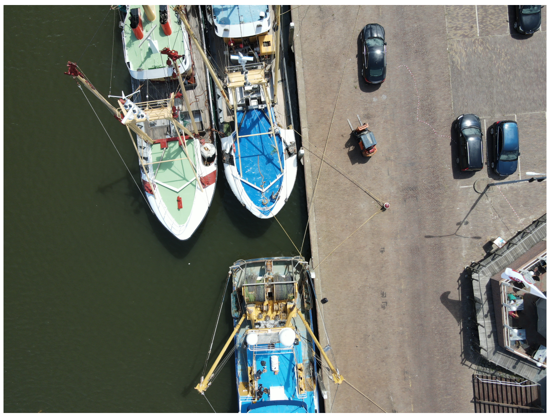


Figure 3.4.: Example drone image from dataset 1

The images from dataset 2 were recorded on November 9, 2022 between 10.29 and 10.41 AM in Ameide, the Netherlands. 2 video's and 61 images were captures of an inspection boat of the NVWA. More details can be found at section 4.2. In Figure 3.5 an example image of this dataset can be found.

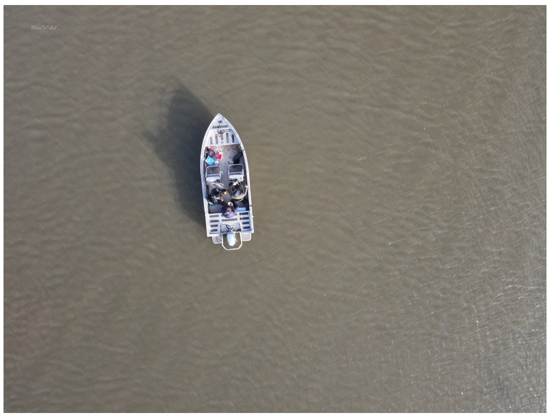


Figure 3.5.: Example drone image from dataset 2

The images from dataset 1 were used to test and develop the prototype. However, they could not be used to validate the positioning algorithm, as the real world GPS position of the boats was not recorded. The images of dataset 2 were used to evaluate the detection, positioning and real time functionalities.

3.2. Object detection model

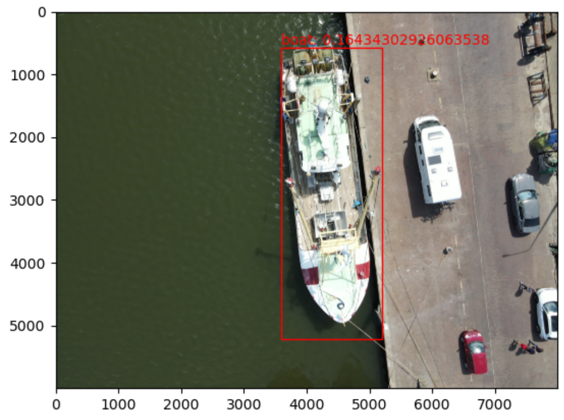
The next step in this research is the development of a deep learning detection model. The criteria these networks have to comply with are to be able to handle drone images, to be able to detect boats from these images and fast inferencing because of real time use. One can build a deep learning architecture from scratch, but data collection, labelling and model training takes a tremendous amount of time and effort. A more commonly used method is using a predefined architecture and training this model for the classes you want. But even then, creating training data requires a large amount of time, for both data acquisition and labelling. Because the development of the deep learning model is not the main focus of this research, the choice was made to investigate the feasibility of the use of a fully pretrained model, with a class that can represent fishing boats already present.

The next step is deciding on the architecture to be used. Section 2.2.2 describes a few of the most common ones. Research points out that the YOLO Architecture has the most optimal tradeoff between accuracy and speed [Carranza-García et al., 2020]. There exist multiple sources for pretrained deep learning algorithms. For example, the Keras Application Programming Interface (API) from the Python Tensorflow library. Keras contains many Applications[2], which are freely available deep learning models with pretrained weights. Among other things, these models can be used for prediction, feature extraction and fine-tuning. The Applications are trained on the ImageNet dataset, which does not contain a boat dataset. It does contain the fireboat and lifeboat classes, however these are not functional to use as class to detect fisher boats.

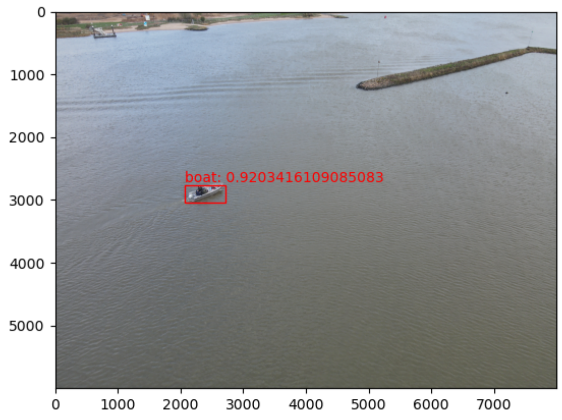
Another API source for deep learning packages is the ArcGIS API for Python library[3]. This API offers a variety of deep learning models and architectures, mainly focused on geographic use cases. The YOLOv3 architecture can be found in the API, pretrained on the COCO dataset. This dataset does contain a boat class that can represent the fisher boats.

Comparing the two options by how they fit to the research problem, the choice was made to use the YOLOv3 model of the ArcGIS API for Python library. The model was loaded in directly from the API in Python. A drone image can then be directly inputted into the predict function of the model. As stated before, no time will be spent on the improving of the detection, so no further research was done on the effect of downsizing the image on the accuracy. The outputs of the predict function are the image pixel coordinates of the bounding box, the prediction certainty and the class name. These are visualised in figure 3.6 for a nadir and oblique image. The cars and bus in image 3.6a are also detected by the model, because it consists of 80 classes. Because only the boat class is relevant to this research, predictions from other classes are filtered out and will not be used in further processing.

[2]https://keras.io/api/applications/
[3]https://developers.arcgis.com/python/



(a) Detection nadir image



(b) Detection oblique image

Figure 3.6.: Visualised detection model output

3.3. Construction of the positioning algorithm

The next step is to calculate the coordinates in the world coordinate system from the pixel coordinates of the bounding box. The world coordinate system in this research is considered to be the World Geodetic System 1984 (WGS 84) geographical coordinate system. As mentioned in 2.3.3, there are multiple techniques to calculate the world coordinates latitude and longitude from the pixel coordinates. In this research, we want to use a quick and simple algorithm, because the calculation needs to be done in real time. For this reason, stereo or multiple photogrammetry will not be used. Moreover, one image will be used because it is challenging and time-consuming to determine the same boat in multiple images. Formally, monoplotting is not used, because feature points have to be determined manually and the goal is to create an automated method. This algorithm can be seen as a variation on this, where a ray is cast from the drone through the image centre. The final algorithm is based on one image and uses the camera position and specification as input parameters. It will be explained below

Firstly, all images that contain detected objects are filtered to be further processed. From these images, the bounding box in pixel coordinates is known from the prediction output. The first step of the algorithm is to calculate the MPP of the image. Figure 3.7 gives an overview of this principle.

To simplify the algorithm for predefined flight paths, the assumption is made that the camera is facing in the nadir direction, so the camera is pointing towards the ground. In predefined drone flights, the drone camera is automatically pointed in the nadir direction at all times. The needed parameters for this problem are: camera resolution, camera lens height and camera field of view. Splitting these parameters for x and y will lead to the following parameters used in the algorithm:

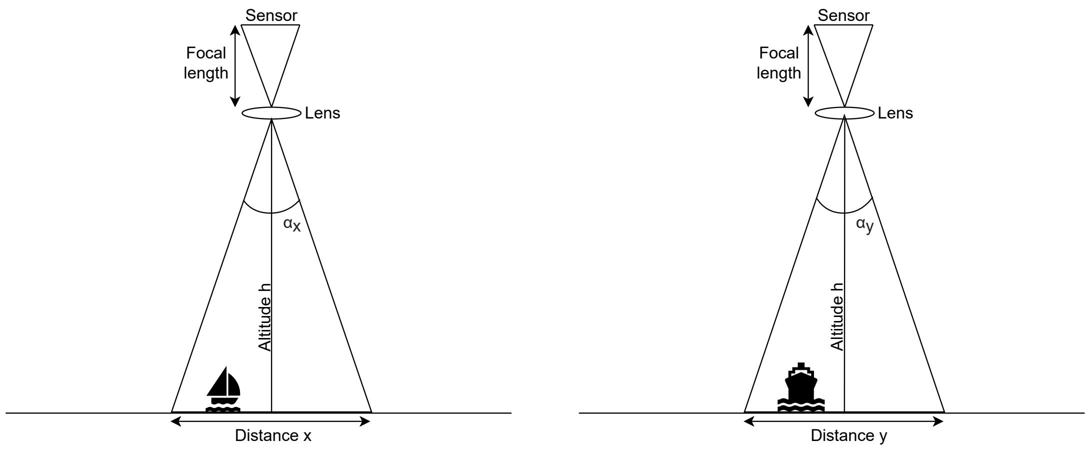


Figure 3.7.: Schematic view MPP calculation

- Ground distance $GD : [x, y]$
- Camera resolution $r : [r_x, r_y]$
- Camera lens altitude h
- Camera field of view angle $\alpha : [\alpha_x, \alpha_y]$

So, the aim is to calculate the ground distance in the x and y direction and divide it by the number of pixels to get the MPP. The h and r parameters can be derived from the camera metadata. The α can be calculated from the focal length f and the camera dimension d. The focal length can be derived from the metadata. The camera dimension can be found in the camera specification. Using trigonometry principles, we know that the tangent of half the view angle is equal to half of the ratio of the ground distance and height. This gives us the following functions:

$$\alpha = 2\arctan(\frac{d}{2f}) \tag{3.1}$$

$$\tan(\frac{\alpha}{2}) = \frac{GD}{2h} \tag{3.2}$$

$$GD = 2h\tan(\frac{\alpha}{2}) \tag{3.3}$$

$$\mu = \frac{GD}{r} = \frac{2h\tan(\frac{\alpha}{2})}{r} \tag{3.4}$$

Where $\mu : [\mu_x, \mu_y]$ is the MPP for both directions.

After the μ is found, a calculation needs to be made how many meters in the x and y direction the detected object is from the middle. This is done by calculating the number of pixels from the middle. The middle of the image represents the GPS coordinate that was measured by the drone and that is present in the metadata of the image. We know how many meters to shift the GPS in the x and y directions. However, the top of the image is not pointed north, so these x and y do not directly correspond to the latitude and longitude of the GPS. To fix this, the x and y are rotated using the camera yaw θ, as shown in the equation 3.5. As can be seen 3.8, the camera yaw represents the heading of the drone camera and thus the angle of the image from the north. When doing this for all the corners of the bounding box, a polygon of the object in the world coordinate system can be created.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.5}$$

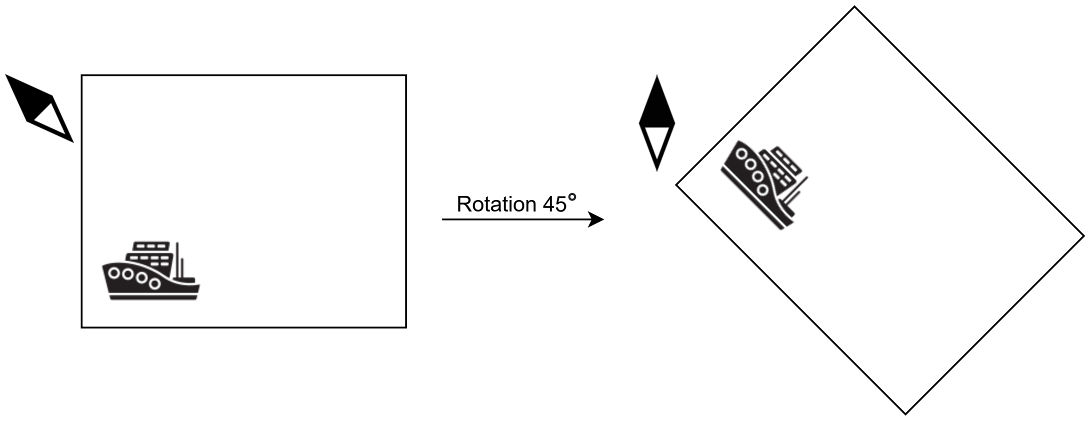


Figure 3.8.: Schematic view rotation with camera yaw

The final step is to convert the translation in meters m to a translation in degrees latitude and longitude $[S_{lon}, S_{lat}]$. The radius of the semi-major axis of the Earth at the equator is 6,378,137 metres. This axis is divided into 360 degrees of longitude, so each degree at the equator is 111,319.5 metres [Tiberius et al., 2021]. To convert meters to latitude, the number of meters needs to be divided by 1 degree latitude, as seen in equation 3.6. As one moves from the equator towards the poles, the longitude radius decreases. To find the radius for the longitude for conversion, the degree in meters needs to be multiplied by the cosine of the latitude Snyder [1987], as can be seen in equation 3.8. Y_{lat} and X_{lon} are the final outputs of the positioning algorithm.

$$S_{lat} = \frac{m_y}{111,319.5} \tag{3.6}$$

$$Y_{lat} = latitude + S_{lat} \tag{3.7}$$

$$S_{lon} = \frac{m_x}{111,319.5 * \cos(Y_{lat})} \tag{3.8}$$

$$X_{lon} = longitude + S_{lon} \tag{3.9}$$

3.4. Real time connection

As explained before, the boats need to be found while the drone is in the air. There thus needs to be a real time connection between the drone and the running prototype. To achieve this, a connection through the cloud service Google Drive was used. Figure 3.9 displays an overview of this construction. The FolderSync app is installed on the drone controller, which automatically uploads the images of a certain folder on the controller to Google Drive. With the Google Drive API for Python, these images are then collected and processed by the prototype. The pseudocode can be found at Algorithm 1.

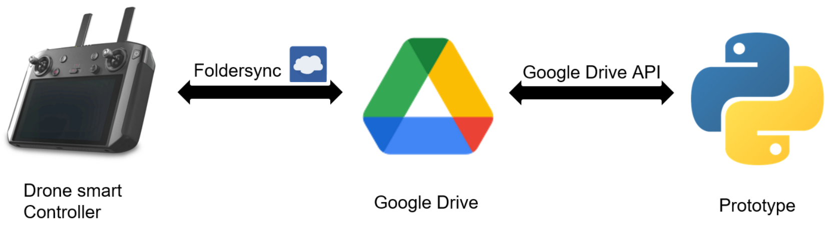


Figure 3.9.: Overview real time connection

3.5. Localisation with a dashboard

When all data is collected and processed, it needs to be visualised insightfully. Localising the positioned objects makes analysis and decision-making possible for inspectors. This is implemented using a dashboard, which is a graphical user interface which provides views of Key Performance Indicator (KPI)s. Dashboards provide the ability to identify and correct trends, measure accuracy and efficiency, and gain insight in multiple systems at once. Dashboard quality is related to the type of dashboard. For this use case, the dashboard is used to visualise the locations of the drones, forbidden areas and found boats. The goal is thus to visualise and not to analyse. Given the fact that the most of the fishery inspectors of the NVWA are no software experts, the dashboard needs to be simple and to the point for it to be effective.

The software used to create this dashboard is ArcGIS Dashboards. This software focusses on the spatial dashboards. It can be directly linked to a web map from ArcGIS Online and has several spatial functionalities.

Figure 3.10 shows a screenshot of the created dashboard. The main feature is the web map in the middle of the screen. This map shows the restricted areas, as well as the points where the drone images are taken and the polygons of the detected boats. The drone image points are shown to make insightful which areas have been covered by the drone. The boats are visualised as polygon to show the approximate size and heading of the boats. The colour red was chosen to make this most important layer stand out. With this dashboard, it is also possible to visualise the footprints of the captured images to see the inspected area.

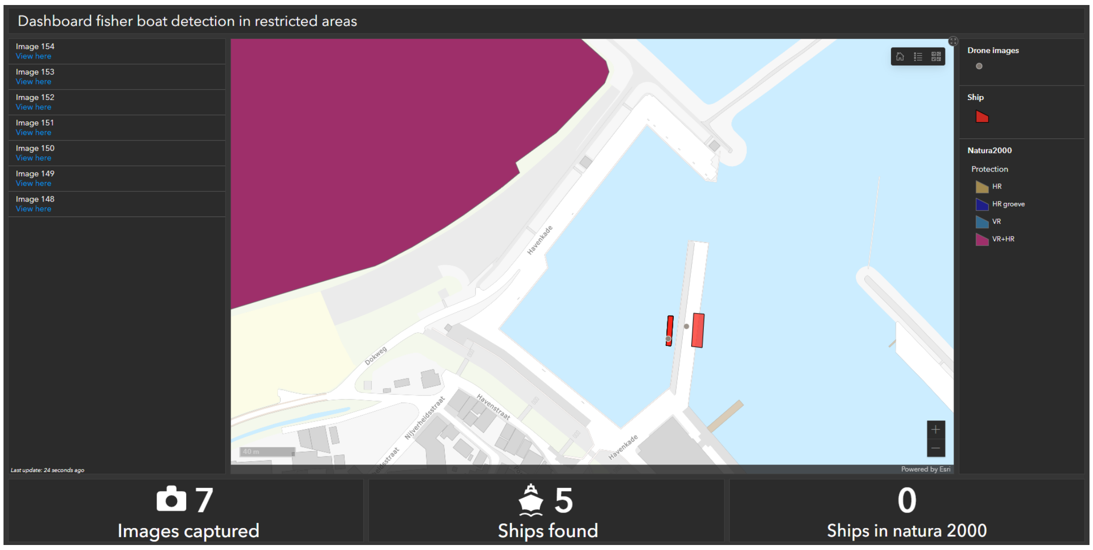


Figure 3.10.: Screenshot of created dashboard

To make the dashboard look more organised, the drone image and ship data are filtered to show only the data collected in the previous 24 hours. Additionally, to automatically show the collected data in the dashboard, an automatic refresh is added so that the layers are updated every 30 seconds.

3.6. Prototype improvement

3.6.1. Oblique positioning

To improve the prototype and to make it more useful for other use cases, a method was constructed to perform positioning for images of an oblique camera angle. In this situation, the drone camera can also be rotated around the pitch axis. Figure 3.11 shows the yaw and pitch axes of a DJI drone camera.

Because the algorithm is adapted from the nadir positioning at 3.3, similar parameters are used:

- Ground distance $GD : [x, y]$
- Camera resolution $r : [r_x, r_y]$
- Camera lens altitude h
- Camera field of view angle $\alpha : [\alpha_x, \alpha_y]$
- Camera principal axes pitch and yaw
- Pixel coordinates predicted bounding box centre $P : [P_x, P_y]$

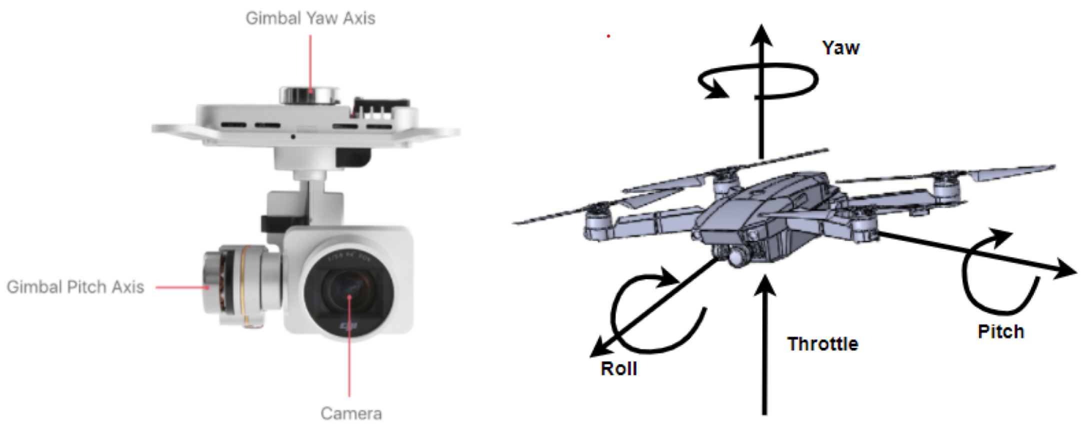


Figure 3.11.: Principal axes camera and drone. From [Keller & Ben-Moshe, 2022]

In the y direction, similar logic can be used as in Figure 3.7. Instead of the meters per pixel, the total meters from the drone over ground is calculated: GD. This is done because in this situation, the meter per pixel is not static, like in the nadir situation. Because the camera pointed in an oblique direction, a pixel close to the camera has a lower MPP than a pixel that depicts a scene far away from the camera. Figure 3.12 depicts the schematic view of the oblique positioning algorithm. As can be seen on the left side, pixels close to the boat have a lower MPP than pixels further away from the boat and drone. With the calculations below is shown how the parameters are used to calculate the ground distance from the drone to the predicted object in the y-direction. We define a new angle β using the field of view and the camera pitch, as seen in equation 3.11. This angle is then used in a trigonometric calculation with the height to calculate the y, as can be seen in 3.12.

$$\alpha = 2\arctan(\frac{d}{2f}) \tag{3.10}$$

$$\beta = pitch + (\frac{\alpha_y}{r_y} * P_y) \tag{3.11}$$

$$y = h\tan(\beta) \tag{3.12}$$

In the x direction, the same logic cannot be used, as the angle β is only valid in the y-direction. The right-hand size of figure 3.12 shows a top view of the calculation used for x. The MPP in the x-direction stays the same moving over the x-axis, but does increase as the y moves further away from the camera. The MPP thus needs to be calculated in the x-direction, while taking into account also the y value of the object. Firstly, the length of H_y is calculated, depicted by the dashed line on both sides of the figure, as seen in equation 3.13. The ground distance of half the triangle of the field of view can then be calculated using H_y and α_x. Finally, r_x and P_x are used to find to calculate the MPP and find x, as can be seen in equation 3.14.

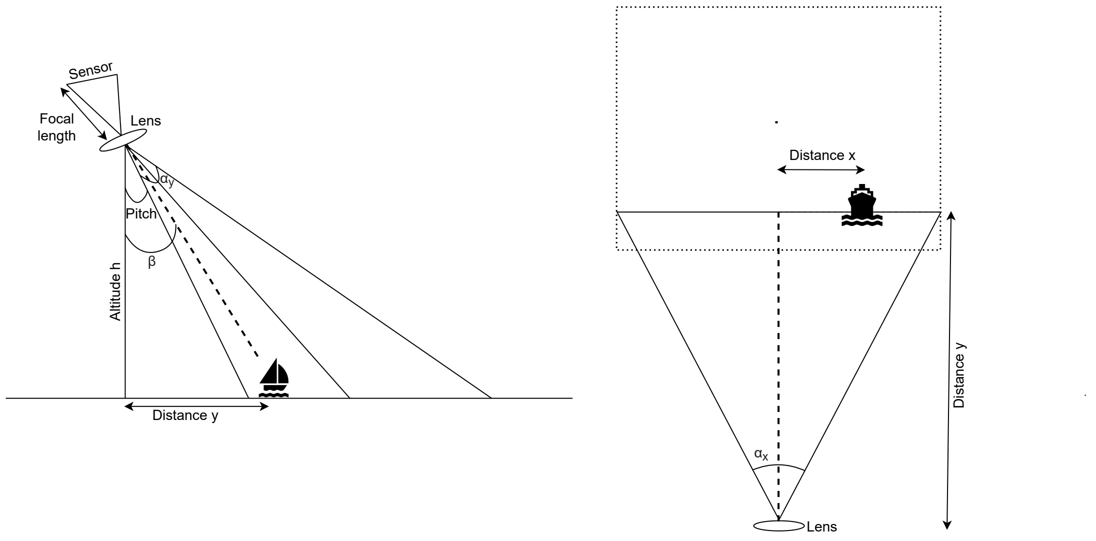


Figure 3.12.: Schematic view oblique positioning

$$H_y = \frac{h}{\cos(\beta)} \tag{3.13}$$

$$x = \frac{H_y \tan(\frac{\alpha_y}{2})}{0.5 r_x P_x} \tag{3.14}$$

After the x and y are found, these need to be rotated using the camera yaw. After this, the translation in meters needs to be converted to degrees latitude and longitude. These values are then used to translate the latitude and longitude coordinates of the drone, found in the metadata of the image. This is all done identically as described in 3.3.

3.6.2. Training the deep learning model

To improve the detection accuracy of the pretrained model described in section 5.1.1, this model was trained further using the acquired data from this research. To investigate if training the model further increases the accuracy, the same pretrained model needs to be used. This is the YOLOv3 model from the ArcGIS Python API, pretrained on the COCO dataset. ArcGIS Pro contains the functionality to train a deep learning model from a pretrained model from the ArcGIS Python API. Figure 3.13 displays the carried out steps to train the model.

Firstly, a raster needs to be generated from the collected images. This is done for the Den Oever dataset. Three images from the Ameide dataset are also included in the raster. Figure 3.14a shows the generated raster. Consecutively, polygon labels are drawn in over each boat on the raster, visualised at 3.14b.

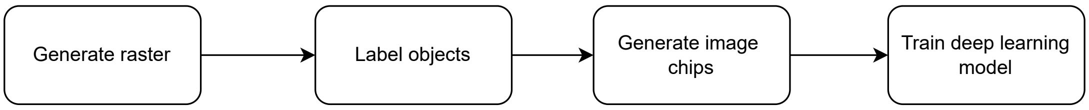
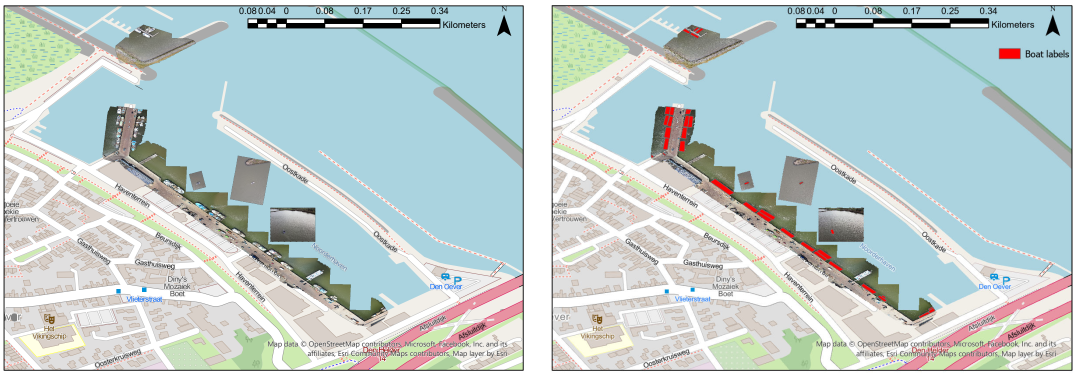


Figure 3.13.: Model training steps



(a) Generated training raster (b) Raster with ground truth labels

Figure 3.14.: Model training input

The labels and the raster are used to generate image chips and respective labels, which are smaller cutouts from the raster. These image chips and formatted labels are inputted into the tool that trains the deep learning model. The output will be a trained deep learning model in the Esri format. Several parameters were used in this method, which will be specified at section 4.4.

3.7. Validation

The created prototype will be evaluated quantitively based on three main criteria. These are related to the three subquestions of the research and will be explained in the following subsections.

3.7.1. Object detection accuracy

The first validation method relates to the first subquestion. Here, the detection accuracy is assessed. When evaluating the accuracy of a detection model, two concepts are essential: the real result y and the predicted result $\hat{y}$ of the model Nan [2022b]. The aim of the detection model is to generate an output $\hat{y}$ that corresponds to the true output y. The more the model output resembles the true output, the more accurate the model is. The model prediction can be classified into four categories regarding its performance:

- **True Positive (TP)**: The model predicts that there is an object and there is an object. The model output is correct
- **False Positive (FP)**: The model predicts that there is an object, while there is not. The model output is incorrect.
- **False Negative (FN)**: The model does not predict that there is an object while there is. The model output is incorrect.
- **True Negative (TN)**: The model does not predict that there is an object and there is not. The model output is correct

In the context of object detection, a true negative is not relevant, because there are an infinite number of bounding boxes in an image that should not be detected Padilla et al. [2020].

The true outputs y are found by manually labelling the validation dataset. These labels and the labels generated by the detection model are compared. Only the boat class is used to qualify the performance.

The following statistics are used: recall and precision. The recall calculation can be seen in equation 3.15. In words, this statistic clarifies if the model makes a prediction every time that the model should have made a prediction. The equation for precision can be seen at 3.16. This statistic defines how often the model makes a correct output, to how reliable the predictions are. In the context of this use case, a high recall is thus more important, as we do not want to miss any existing boats. When taking into account the confidence values of the detection when calculating the precision and recall, one can create a curve that displays a trade-off between these statistics [Padilla et al., 2020].

$$Recall = TP/(TP + FN) \tag{3.15}$$

$$Precision = TP/(TP + FP) \tag{3.16}$$

These statistics require a method that defines correct and incorrect detection. The IoU method calculates a ratio that is used as a threshold to determine whether a predicted outcome of the model is a true positive or a false positive. It measures the amount of overlap between the bounding boxes of the predicted (B_p) and true (B_{g_t}) objects [Padilla et al., 2020].

$$IoU = \frac{area(B_p \cap B_{g_t})}{area(B_p \cup B_{g_t})} \tag{3.17}$$

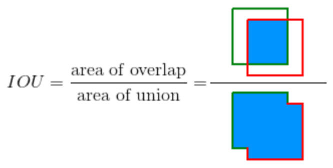


Figure 3.15.: Intersection over Union. From Padilla et al. [2020]

By weighing the IoU ratio against a threshold t, a detection output can be classified as being correct or incorrect. With $IoU \geq t$ being correct and $IoU < t$ being incorrect.

3.7.2. Positioning accuracy

This subsection describes the assessment of the positioning accuracy. During data acquisition, a GPS-tracker will be placed in the middle of the boat. These coordinates will be compared to the calculated coordinates of the positioning algorithm. The distance between the predicted coordinates $\hat{lon}$ and $\hat{lat}$ and the coordinates measured by the GPS-tracker longitude (lon) and latitude (lat) will be evaluated separately. After calculating the distance between the predicted and ground truth coordinates in degrees, this distance is converted to meters. This is done opposite of equations 3.6 and 3.8, so instead of a division, multiplication is carried out. Additionally, the total Euclidean distance in meters between the two points will be calculated and evaluated with equation 3.18. The average error will also be calculated, as seen in equation 3.19. Additionally, the average absolute error of both coordinates will be calculated. This is done by turning each distance into an absolute value and calculating the average of these values with equation 3.19.

$$d = \sqrt{(lon - \hat{lon})^2 + (lat - \hat{lat})^2} \tag{3.18}$$

$$Average = \frac{1}{n}\sum_{i=1}^{n} d_i \tag{3.19}$$

3.7.3. Speed real time connection

The third method assesses the real time speed of the prototype. The processing time of the different components of this prototype are measured separately. Uploading and extracting an image from Google Drive is measured, as well as the time to run the detection and positioning algorithm and the time from writing the data to file to it showing up on the dashboard. This is done for multiple images so that the average time per component can be calculated. The time is measured with a stopwatch for the drone controller component, and the remaining timers are implemented in the Python code.

To gather data for the evaluation, an experiment was conducted, as described in section 4.5.

4. Implementation

4.1. Tools

4.1.1. Software tools

In this section, the software tools used for this research are described. Table 4.1 gives an overview of the used software. The majority of the software used is from ArcGIS. This gives the advantage of interoperability, as all ArcGIS tools work closely together.

Table 4.1.: Software tools

Tool	Version	Description
ArcGIS Pro	3.0	Desktop Geographic Information Software used to visualise, analyse and process geographic information.
ArcGIS Online	10.3	Cloud-based mapping and analysis software.
ArcGIS Dashboards	10.9.1	Tool to create real time dashboards that can present location-based analytics using intuitive and interactive data visualisations.
ArcGIS Python API	2.0.0	Python library that is used for mapping, spatial analysis, data science, geospatial AI and automation.
Arcpy	3.0	Python site package that provides ArcGIS Pro tools in Python.
Drone2Map	2.3.0	ArcGIS desktop application that can be used to process and analyse drone data.
Google Drive	67.0	Shared cloud storage platform.
Google Drive API	V3	REST API that can interact programmatically with Google Drive.
Pillow	9.3.0	Python Imaging Library that contains a variety of image processing capabilities.
Foldersync	3.2.9	Android app that allows automatic syncing from local folders to cloud servers.
Open GPX Tracker	1.8.0	iPhone app that creates GPS traces with waypoints in the open GPX format.
CVAT	2.2.0	The Computer Vision Annotation Tool is a free and open source web-based image and video annotation tool.

The ArcGIS Pro desktop application is used during this research to visualise a variety of testing layers. To test different algorithms, different point and polygon layers were produced by the prototype. When creating this information as ArcGIS Pro layers, datasets could be tested locally, independent of the cloud. ArcGIS Pro is also used to train the deep learning model.
After testing the detection and positioning algorithms, the data created by the prototype are sent to ArcGIS Online. In ArcGIS Online, layers can be easily shared among others across the same organisation. When layers are hosted on ArcGIS Online, they can be made available in a web map, which is a map containing multiple layers. Functionalities like automatic updating and filtering are available here.
This web map can then be used to create a dashboard in ArcGIS Dashboards. The ArcGIS Dashboard software was chosen to be used to create the localisation dashboard for the inspectors. The reason for this was the advantage of the high interoperability with the other ArcGIS Software like ArcGIS Online. Additionally, the goal of this software is to create dashboards for location-based data, which fits well with the objective of this research.
To create the prototype, the ArcGIS python libraries Arcpy and the ArcGIS Python API were used. Arcpy was mainly used to create and write layers or data to ArcGIS Pro. The ArcGIS Python API was used to download and inference with the deep learning model. Additionally, it was used to send data to ArcGIS Online.
The ArcGIS desktop application Drone2Map was used to convert acquired drone images to an orthomosaic for use in model training, described at 4.4.

When images are downloaded to the drone controller, they are automatically synced with Google Drive. This connection is created using the FolderSync app. This is an android application that can be downloaded on the smart controller.
The cloud storage platform Google Drive is used to store the drone images on. A separate Google account was created for this research.
Using the Python Google Drive API, the images can be downloaded locally to the prototype. Firstly, the Drive is searched for images with the name DJI of type .jpg. This function retrieves all id's of the images correspond to this condition. Next, these id's are used to download the images in a byte format.
To get the metadata from the images in this format, they are converted to Image objects using the Pillow library. Using the getxmd and getexif functions, all necessary metadata can be easily extracted from the image.
To determine the accuracy of the positioning algorithm, the calculated coordinates need to be compared to the ground truth. To collect these ground truth coordinates, the Open GPX Tracker application was used, downloaded on an iPhone 8. Using this app, when pressing start, the latitude and longitude are determined and stored every 0-3 seconds. A timestamp is also recorded with the coordinates. The coordinates are stored in a .gpx format, which follows the Extensible Markup Language (XML) standard.
The web-based annotation tool CVAT was used to label all the images of acquired datasets. Only the boat objects are labelled.

4.1.2. Hardware tools

All data used in this research is collected by drones, as will be described in the next subsection. The DJI Mavic 2 Enterprise Advanced drone[1] was used to collect data. This drone has a 1/2" CMOS camera lens, which produces images with an 8000 by 6000 pixel size. Additionally, a Dell laptop with a 4 GB memory GPU is used to create the prototype and tests on. The smart controller of the drone is used to create a real time streaming connection on. This controller has an android environment. To acquire the ground truth coordinate data for dataset 2 to evaluate the algorithm in the results, a Geo 7x Trimble is acquired and taken to the testing area. This handheld Global Navigation Satelite System (GNSS) tracker would be able to collect the latitude and longitude with at least sub-meter accuracy. However, due to a defective receiver, the device could not achieve a first fix in the area. For this reason, the backup solution of using the iPhone application Open GPX Tracker was used.

4.2. Used data

To compare the found boat polygons against restricted areas, a polygon dataset of the Natura 2000 areas was used[2]. The dataset consist of 163 different areas both over land and over water. Firstly, the Web Feature Service (WFS) layer was used, which could be directly loaded into ArcGIS Online and ArcGIS Pro using the WFS link. However, it was found that the features retrieved from this link could not be used in further processing that was needed to show statistics on the ArcGIS Dashboard. For this reason, The Natura 2000 feature layer was downloaded from the Pdok website as a shapefile and subsequently hosted on ArcGIS Online as a feature layer.

Test data used in this research was collected with drones, as mentioned in 3.1. For the data collection in Den Oever, a DJI Mavic 2 Enterprise Advanced drone was used. In Den Oever, 190 images in the harbour were collected by this drone of several fish cutters. Figure 4.1 displays the data acquisition area of this dataset.

[1] https://www.dji.com/nl/mavic-2-enterprise-advanced

[2] https://www.pdok.nl/introductie/-/article/natura-2000

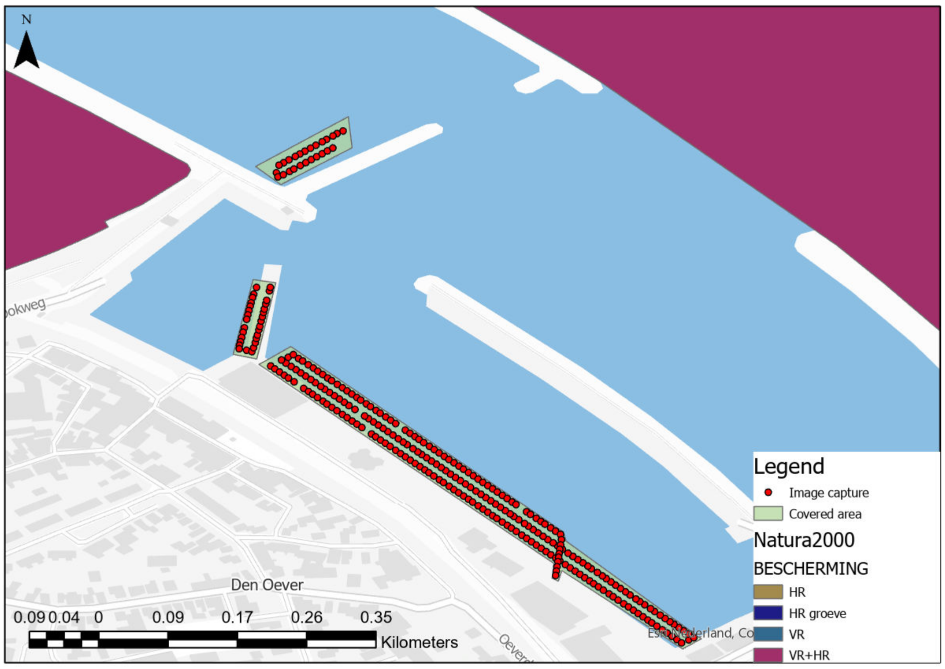


Figure 4.1.: Data acquisition area Den Oever dataset 1

At November 9, 2022 a test was conducted to gather data for the results. In Table 4.2, the different collected subdatasets of dataset 2 are shown. As can be seen, datasets were recorded with a varying boat movement and camera angle. The images recorded during this test were used to process the results. The videos were used to investigate the operability of using drone video for this method instead of images. This will be discussed in section 6.3.

Table 4.2.: Experiment parameters dataset 2

Test	Boat	Camera angle	Data type
1	Lying still	Nadir	Images
2	Lying still	Non-Nadir	Images
3	Lying still	Non-Nadir	Video
4	Moving	Nadir	Images
5	Moving	Non-Nadir	Images
6	Moving	Non-Nadir	Video

The test was conducted in the Dutch city Ameide, which is situated along the Lek river. Figure 4.2 shows the data acquisition area of dataset 2.

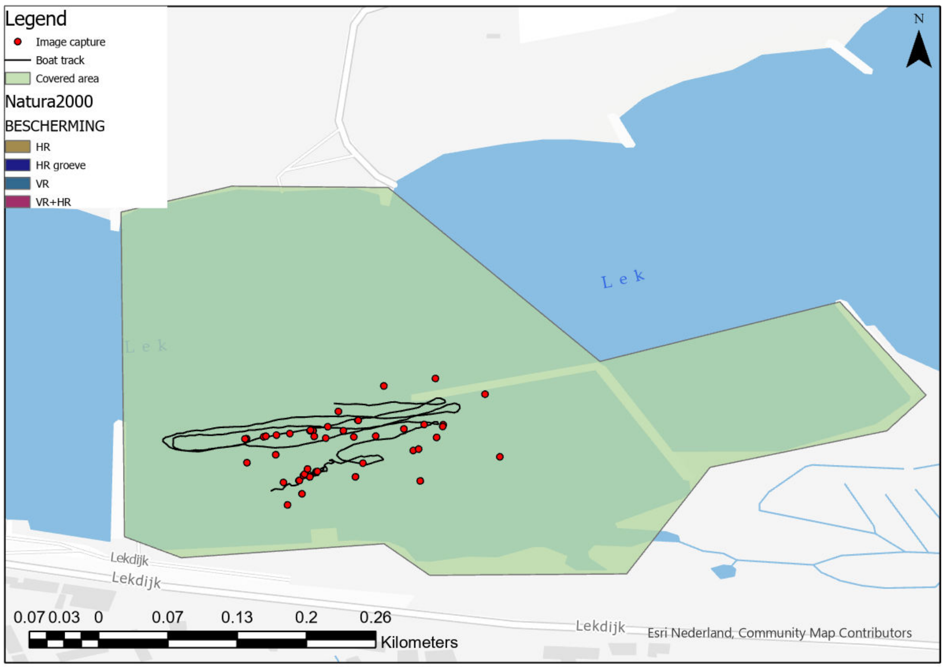


Figure 4.2.: Data acquisition area Ameide dataset 2

Drone images were collected for both datasets. Besides the pixel data, each image contains metadata that is needed to calculate the exact object position. The following metadata was used from the collected drone images:

XMP:

- GpsLatitude
- GpsLongitude
- RelativeAltitude
- GimbalYawDegree
- GimbalPitchDegree

Exif:

- FocalLength
- ImageWidth
- ImageLength
- DateTime

For the 2D positioning algorithm, the parameters GpsLatitude, GpsLongitude, RelativeAltitude, GimbalYawDegree, FocalLength, ImageWidth and ImageLength were used. To convert the algorithm to handle oblique images, the GimbalPitchDegree parameter was added as input. The DateTime parameter was added to process the validation of the results. The metadata also contains the FlightYawDegree and FlightPitchDegree, however these parameters correspond to the position of the drone while the Gimbal parameter corresponds to the position of the drone camera. The RelativeAltitude parameter calculates the height of the drone relative to its take-off height. The metadata also contains the AbsoluteAltitude parameter, however after investigation on its values, it was found that this parameter is highly inaccurate (10-100 m errors), so it cannot be used in this research. Because the relative altitude is used, it is recommended that the take-off point is as close to the ground as possible.

4.3. Code structure

In this section, the important parts of the created prototype code will be explained and shown in pseudocode. The created code can be found at GitHub[3].

Before the main loop is started, all used feature layers of ArcGIS Pro are completely cleared. These cleared layers are used for testing, which makes it clear that the content of the layer is the output of only the previous run. The layers in ArcGIS Online are for functional use and not for testing. Because only the features of the last 24 hours are filtered out, this layer does not need to be cleared. The purpose for this is that inspectors still have insight into previous flights, should the need arise for this.

After the test layers are cleared, the main loop function is started. The pseudocode can be seen at Algorithm 1. Before the while-loop, the Google Drive API connection is initialised, the stack is filled with the image IDs of the unprocessed images and the detection model is retrieved from the ArcGIS API. The while-loop runs until all images in the stack are processed. Each iteration, the first image ID is taken from the stack to be processed. Firstly, using the ID, the image is downloaded in byte-format using the Google Drive API. Secondly, the necessary metadata is extracted from the image. The objects in the images are then predicted using the prediction function from the detection model. Only the boat predictions are relevant to this use case, so these predictions are kept. Then, for each of the predicted boat features, the geographical coordinates of the bounding box are calculated and written to an ArcGIS Online layer as a polygon feature. Processing for this image is then finished, so it is removed from the stack. Because operation is done in real time, new images are added to Google Drive while processing. At the end of the loop, newly added images from Google Drive will be added to the stack. After this, a new image will be processed. When more images are retrieved than can be processed, the stack continuously grows. Because drones do not stay in the air forever and ultimately stop capturing new images, this is not expected to be an issue.

[3]https://github.com/Lisageers/OpsporingVissersboten

Algorithm 1 Main loop

```
Input: link to Google Drive
Output:
  $M \leftarrow$ deep learning model
  Fill stack with image IDs
  while IDs in stack do
    Get ID from stack
    Download image bytes $img$ with ID
    $p \leftarrow$ inference using $M$ and $img$
    filter boat features $b$ from $p$
    for each $b$ do
      $c \leftarrow$ positioning $b$
      write $c$ into online layer
    end for
    remove current ID from stack
    add new image IDs from drive to stack
  end while
```

In Algorithm 2, the positioning algorithm for the camera in the nadir direction is shown. The input data of this algorithm are the image metadata and the bounding box pixel coordinates of the prediction for the boat class. The first step is to calculate the field of view. This is done separately for both the X and Y directions. Subsequently, the meters per pixel is calculated for both directions. In the next step, the distances of the bounding box pixels to the image centre are calculated, because the image centre is where the geographical position from the metadata is located in the image. The pixels distances are then rotated using the camera yaw in radians. After the pixel coordinates are rotated, the distance in pixels of each corner of the bounding box to the image centre is known. Multiplying this distance to the meters per pixel, the total translation in meters is known for both directions. This number of meters is then converted to degrees latitude and longitude and added to the latitude and longitude of the image centre from the metadata. This leaves us with the geographical coordinates of the bounding box for the prediction.

Algorithm 2 Positioning algorithm nadir

```
Input: M: Metadata
       $P$: predicted bounding box pixel coordinates
Output: latitude $la$ & longitude $lo$ of bounding box
  $A \leftarrow$ angle field of view
  $mpp \leftarrow$ calculate meter per pixel using $M$ and $A$
  $D \leftarrow$ pixel distance of $P$ from image center
  $D_r \leftarrow$ rotate pixel distance $D$ using $yaw$ from $M$
  $u \leftarrow$ convert $(mpp * D_r)$ to $la$ & $lo$ degrees
  $la, lo \leftarrow la_m, lo_m$ from $M + u$
```

Algorithm 3 describes the algorithm for the positioning of objects on images taken with oblique camera angles. The algorithm is adapted from algorithm 2 and is thus similar. It uses the same input, the image metadata and the predicted bounding box. Because determining the object footprint is complex for oblique imagery, the geographical coordinates of the centre of the predicted bounding box are calculated. In the first step, the FOV angle is calculated. This angle is divided by the image width and height to get the FOV angle per pixel. Thereafter, the middle of the predicted bounding box and the distance of the middle from the image centre are calculated. Using the trigonometry functions described in 3.6.1, the ground distance in meters from the recorded latitude and longitude in the metadata are calculated. These distance values are then rotated using the camera yaw. Finally, the distances are converted to degrees latitude ang longitude and used to update the geographical coordinates. The output of this algorithm are the geographical coordinates of the centre of the object bounding box.

Algorithm 3 Positioning algorithm oblique

```
Input: M: Metadata
       $P$: predicted bounding box pixel coordinates
Output: latitude $la$ & longitude $lo$ of bounding box centre
  $A \leftarrow$ angle field of view
  $app \leftarrow A/$ image width $w$, height from $M$
  $D \leftarrow$ pixel distance of middle $P$ from image center
  $GD \leftarrow$ calculate ground distance y of $D$ from $la_m$, $lo_m$ using pitch, height, $w$ from $M$, $app$
  $GD_r \leftarrow$ rotate $GD$ using $yaw$ from $M$
  $u \leftarrow$ convert $GD_r$ to $la$ & $lo$ degrees
  $la, lo \leftarrow la_m, lo_m$ from $M + u$
```

4.4. Train Deep learning model

Before the deep learning model could be trained in ArcGIS Pro, considerable investigation and preprocessing was necessary. Firstly, it was investigated if it was possible to train the model using the labels from CVAT that were made for the validation of the detection model on the collected images. It was found that it is not possible to directly train the model using the labels from this program. Although the tool description states that it accepts labels in the PASCAL Visual Object Classes (VOC) format, which the labels on CVAT can be exported to[4], the tool did not accept these labels. After investigating the output of the ArcGIS Pro tool 'Export training data for deep learning', it was found that besides the .xmd label files in the Pascal VOC format, several other files and folders were created that are necessary to train the model on ArcGIS Pro. It was concluded that the generated files were too complex to generate without the tool, because a proprietary format was used for multiple files, like the 'esri_acculated_stats.json' and 'esri_model_definition.emd'. The option to train the model with a different application than ArcGIS Pro was briefly considered. However, the pretrained model used in this research is only available in an ArcGIS format, which makes it unusable for other applications. Using another pretrained model was considered, but this makes comparing the results of the ArcGIS pretrained model with the trained model not valid.

[4]https://pro.arcgis.com/en/pro-app/latest/tool-reference/image-analyst/train-deep-learning-model.htm

The input for the export training data tool is an imagery layer[5]. The collected images thus need to be converted to this format. The images of the Den Oever dataset can be converted to an orthomosaic raster layer with ArcGIS Drone2Map, because the images were collected in a predefined flight path at a regular interval. For the images collected in Ameide, this was not done, because the same boat was captured in each image, one image can be used as raster. Three random images were chosen from the Ameide dataset to be used for training, two nadir images and one oblique image, because the nadir accuracy needs to be increased. The images were georeferenced using the image metadata to the right position and scale. A translation was then carried out on the images to position them closer to the Den Oever raster. Figure 3.14a shows the raster. Positioning the rasters close together is more convenient when creating the image chips and the position is not important here. The next step is to combine the rasters into one dataset, this was done with the Append tool in ArcGIS Pro. Because the drone image rasters are of very high quality and model training is computationally heavy, the spatial resolution of the rasters needed to be decreased. This was done by increasing the cell size from 0.00559 to 0.0559 with the Resample tool in ArcGIS Pro. By decreasing the resolution by this number, the objects are still clearly visible and the used laptop is able to handle the model training.

The raster is now ready for the generation of the image chips. Previous to this, the objects need to be labelled. The labelling was done manually by drawing polygons over the boat objects using the 'Label Objects for Deep Learning' tool. The resulting polygons are saved in a feature layer. The next step is to generate the image chips and corresponding labels in the Esri format using the 'Export Training Data For Deep Learning' tool. This tool converts the raster sub-images, called image chips. Corresponding .xml files containing the feature labels are also created for each sub-image, as well as additional files in Esri format, like the model definition file. The raster and labelled polygons are the input for this tool. The chosen image format for the image chips is .JPEG, because this is the format of the collected drone images used in the research. The pixel size of each image chip was chosen to be 512, double the standard size of 256, to make sure an image chip can contain the entire feature. A stride of 128 was decided, which is the standard value, to make sure the dataset is covered well and there are enough samples to train the model. The metadata format of the output was selected to be PASCAL VOC, because this is the standard option and this format is fit for training the YOLOv3 model. After running this tool, 644 images with 1207 features were created out of the initial raster with 35 labels.

The final step is to train the pretrained YOLOv3 model with the 'Train Deep Learning Model' tool in ArcGIS Pro. The input of this tool is the folder with the image chips and labels created in the previous step. The model type was chosen to be YOLOv3, as this is the same pretrained model used in this research. The parameter chip_size was set to 512, as this is the size of the image chips. The number of epochs was increased from the standard 20 to 50. This is done to make sure the trained model is fitted to the input more closely. The processing batch size is decreased to 1. It was found that the laptop used to train the model could not handle batch sizes larger than this when training the model due to memory constraints. The training time with these parameters on the used laptop was 7 hours and 45 minutes. The output is the newly trained deep learning model in the Esri format. This model can be directly used in a Python script with the ArcGIS Python API and the path to the created Esri model definition file. The resulting accuracy of the trained model can be found in section 5.1.3.

[5]https://pro.arcgis.com/en/pro-app/latest/tool-reference/image-analyst/export-training-data-for-deep-learning.htm

4.5. Experiments

To come to the results, several experiments were created to process the acquired data and gather statistics. The experiments were subdivided between detection (4.5.1), positioning (4.5.2) and real time 4.5.3. Test were run separately to determine the statistics of the main components without them influencing each other.

4.5.1. Object detection

To determine the accuracy of the detection model for the specific use case of the fisher boats, the detected boat polygons $\hat{y}$ need to be compared with the real polygons y. Firstly, these real polygons need to be created by hand. This was done with the CVAT web tool. The following datasets were labelled:

- Dataset 1: All images from Den Oever
- Dataset 2: All images from test 1 and 4 in Ameide (all nadir images)
- Dataset 3: All images from test 2 and 5 in Ameide (all non-nadir images)

The images from these three datasets were uploaded separately to the CVAT web tool. One to be used 'boat' class was added to the three projects. Each image from all datasets was then investigated separately. A rectangle bounding box was drawn manually for each boat that was visibly present in the image. After all images were investigated, the labels could be downloaded locally in a variety of formats. After some experimenting, the format 'CVAT for images 1.1', which is a XML type format, was chosen to be the most optimal export format. In other formats like 'COCO 1.0' or 'YOLO 1.1' the bounding boxes and image names were in separate files or calculations had to be carried to get to the bounding box pixel numbers. Both of these would lead to more unnecessary programming complexity when processing these labels. The ground truth labels were then converted using a Python script, shown at Algorithm 4. For each image, a .txt file is created with the name of the image corresponding to the name of the file, so "imagename.txt". In this file, the labels are written in the following way: "classname xmin ymin xmax ymax" with a new line for each label. Converting the ground truth labels to this format allows them to be directly processed into the mAP program[6], created by [Cartucho et al., 2018]. This algorithm evaluates the detection accuracy and produces several statistics like the precision-recall curve and the average precision. Besides the ground truth labels, the program also requires the original images with the relating names and the bounding boxes predicted by the detection model. These predicted bounding boxes $\hat{y}$ should be should also be put into separate .txt files with corresponding filenames in the format "classname confidence xmin ymin xmax ymax". The pseudocode of the python script used to create this format for all the test images is shown at algorithm 5. This script was run three times for each dataset, resulting in nine different outputs. Firstly, it was run using the pretrained deep learning model. Secondly, it was run with the pretrained model and turning of the filter for the boat class, which is shown as the if-statment at Algorithm 5. Lastly, it was run using the model trained in this research.

[6]https://github.com/Cartucho/mAP

Algorithm 4 Process results ground truth

```
Input: P: path to annotation file
Output: .txt files with ground truth
  $GT \leftarrow$ read $P$
  for each $img$ in $GT$ do
    create .txt
    if $image.bbs$ != exist then
      continue
    end if
    $bbs \leftarrow$ get bounding boxes from $img$
    for $bb$ in $bbs$ do
      write $bb$ to .txt file
    end for
  end for
```

Algorithm 5 Process results detection

```
Input: P: path to images
Output: .txt files with predictions
  $M \leftarrow$ deep learning model
  for each $img$ in $P$ do
    create .txt
    $O \leftarrow$ inference with $M$ on $img$
    for each $Pred$ in $O$ do
      if $Pred$ == boat then
        write $Pred$ to .txt file
      end if
    end for
  end for
```

The ground truth .txt files should be put into a folder with the name ground-truth. Additionally, the images should be in a folder named images-optional and the predicted label .txt files should be put into a folder named detection-results, all in the mAP program folder. The mAP program is then ready to be run. This was done four times for each dataset, resulting in 12 accuracy outputs of the mAP program. Firstly, it was run using the detection files with the pretrained model. Secondly, using the pretrained model without filtering out the boat class. Finally, it was run two times for each dataset using the trained model and changing between IoU ratio of 0.5 and 0.1.

4.5.2. Object positioning

To determine the accuracy of the positioning algorithm, the ground truth needs to be compared to the predicted outcome. For this experiment, the ground truth data was collected with the Open GPX Tracker iPhone application. This was done for all tests of dataset 2 from Ameide. One of the boat passengers held the iPhone with the Open GPX Tracker application as still as possible. In figure 4.3 this can be seen. The red square in both images indicates where the phone was situated on the boat. As can be seen, the phone is not exactly in the middle of the boat, but approximately one meter towards the end of the boat. Because the positioning result aims to find the coordinates of the middle of the boat, this could explain errors up to 1 meter in the results.

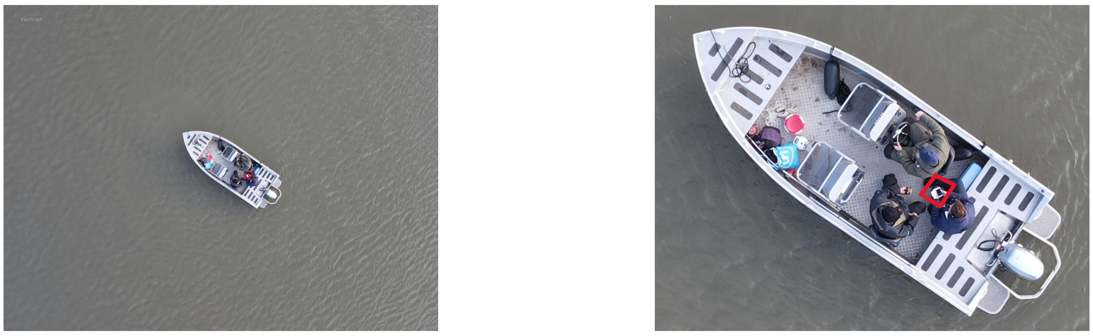

(a) Boat with GPS tracker (b) Zoomed-in boat GPS tracker

Figure 4.3.: Ground truth data collection positioning

After the GPS was tracked, these tracks were automatically saved to a .GPX file by the application. In these files, the recorded GPS points and the corresponding timestamp are stored. The files, which have an XML format, can be easily parsed in a Python script. The pseudocode can be seen at Algorithm 6 In the Python script, all images in each test are processed. For each image, the necessary metadata is retrieved. After, the recorded coordinate that is the closest in time to the image capture, is chosen to be the ground truth coordinate to compare the positioning output to. When the image time does not directly correspond to a recorded ground truth timestamp, a ratio is calculated for the interval the image falls in. This is done so that the ground truth coordinate is as close to reality as possible. After the ground truth coordinates are found, the ground truth boat bounding box is retrieved from the ground truth file with the same name as the image. The ground truth bounding box is used to measure the localisation algorithm without possible influence of the accuracy of the deep learning detection bounding boxes. For the middle of this ground truth box, the GPS coordinates are calculated using the positioning algorithm. These coordinates are then compared to the ground truth coordinates. In this way, ground truth and calculated coordinates are retrieved for each boat in each image (all images contain one boat with ground truth GPS). The distance between the ground truth and the calculated coordinates is then calculated, recorded and described in section 5.2.

Algorithm 6 Process positioning accuracy

```
Input: $P_p$: path to positioning ground truth file
       $P_d$: path to detection ground truth file
       $P_i$: path to image folder
Output: .txt files with positioning errors
        scatter plots positioning errors
  $GT_p, GT_d \leftarrow$ read $P_p$, $P_d$
  for $img$ in $P_i$ do
    $M \leftarrow$ get metadata from $img$
    $S \leftarrow$ smallest time difference
    for $C_{gt}$ in $GT_p$ do
      $T_i, T_p \leftarrow$ convert timestring to miliseconds from $M$, $C_{gt}$
      if $abs(T_i - T_p) < S$ then
        $S_{gt} \leftarrow C_{gt}$
      end if
    end for
    for $bb$ in $GT_d$ do
      $C_p \leftarrow$ use $M$ to position $bb$
      $E_{lat}, E_{lon} \leftarrow$ calculate difference $S_{gt}$ - $C_p$
      write $E_{lat}$, $E_{lon}$, $\sqrt{E_{lat}^2 + E_{lon}^2}$ to .txt file
    end for
    create scatterplot of all errors $E$
  end for
```

4.5.3. Speed real time connection

The speed of the real time connection and algorithm are also assessed. To measure the running speed, the prototype was subdivided into different key components. Table 4.3 displays the different components that were tested. For the first two rows, the speed was tested of the smart controller components with a stopwatch. The 'Start' and 'End' columns represent the moments where the stopwatch is started and stopped. For the remaining columns, the time is measured with the python Time library in the prototype code. A timestamp is recorded at the 'Start' and 'End' column points. The difference in seconds between these timestamps is then calculated, stored and described in section 5.3.2 The tests were run with a 50 mbit/s download speed and 20 mbit/s upload speed, which could have an influence on the outcome. At the time of testing, there were 8 devices connected to the same connection, which could have decreased the upload and download speed.

Table 4.3.: Prototype speed experiment

Test	Start	End
Download full size	Press download button in DJI Pilot app	Pop-up download complete in screen
To Drive	Press Sync in DJI Fly application	Foldersync marking the sync complete
From Drive	Start Drive API call	Image is retrieved in byte format
Detection	Start inference with model.predict() function	Prediction is outputted
Positioning	Positioning algorithm is started	coordinates are outputted
Write to file	Start searching for file in ArcGIS Online	File in ArcGIS Online is updated with new feature
Code total	Start while loop stack	End of while loop

5. Results

In this chapter, the results of this research are presented. In section 5.1 the accuracy of the pretrained and trained detection models is assessed. Section 5.2 describes the accuracy of the positioning algorithms. Finally, in section 5.3 the real time functionality and speed of the prototype is investigated and described.

5.1. Boat detection accuracy

In this section, the accuracy test results of the detection model will be displayed and analysed. The existing two datasets are split into three datasets. Dataset 1 contains all images from dataset 1, captured in Den Oever. Dataset 2 contains all images of the Ameide data collection from dataset 2 where the camera is pointed in the nadir direction. Dataset 3 contains the remaining images from data collection in Ameide where the camera was pointed in an oblique direction during image capture. The Ameide dataset was split to evaluate the performance of the detection models on both nadir and oblique imagery.

For all images of these three datasets, ground truth and detection labels were collected to calculate accuracy statistics. Specific implementation details can be found at 4.5.1. Using the Python code of [Cartucho et al., 2018] with some minor alterations, the performance parameters could be calculated. These minor alterations are made to mare visualisations more clear and to change the IoU ratio. This is done for the pretrained model and the trained model. Additionally, the accuracy is assessed taking into account all labels of the pretrained model without filtering out the boat label. Tests are run with a standard IoU of 0.5 unless specified otherwise. The results of the accuracy assessment consist of an Average Precision (AP) value and a precision-recall curve.

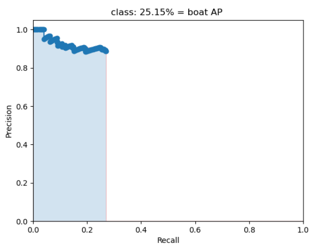
5.1.1. Results pretrained model

In this subsection, the accuracy of the pretrained model is evaluated. The produced results are table 5.1 and the 5.1 plots. As can be seen in the AP column at 5.1, the precision of dataset 1 and dataset 2 are quite unsatisfactory. The largest difference with dataset 3 is that the first two datasets were both taken with the camera in the nadir direction. In dataset 3 the camera was pointed at an oblique angle.

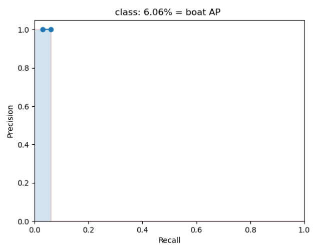
This result is also visualised in Figure 5.1. The AP is the area under the curve, shown in light blue. The precision on the y-axis shows how often the model predicts the bounding boxes correctly. As can be seen for all three datasets, the precision is quite high for an increasing recall, which indicates that when the model makes a prediction, this is a correct prediction. At 5.1c the precision starts to drop at a recall of 0.5. This would indicate that predictions made by the model with a probability of 0.5 or higher are always correct in this case, as the precision is 1.0. For predictions with a 0.5 or lower, the predictions are not always correct.

Table 5.1.: Detection results pretrained model

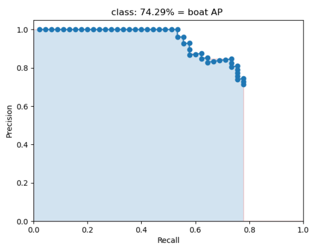
Dataset	AP	True Positives	False Positives	False Negatives	Ground truth objects	Detected objects
1	25.15%	127	16	344	471	143
2	6.06%	2	0	31	33	2
3	74.29%	35	14	10	45	49



(a) Dataset 1: Den Oever



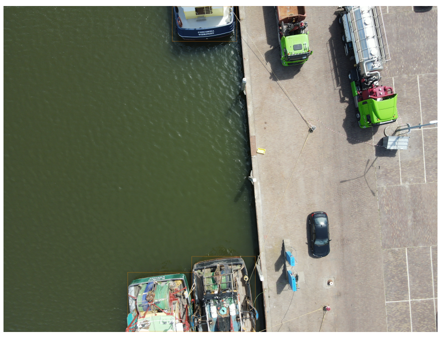
(b) Dataset 2: Ameide nadir



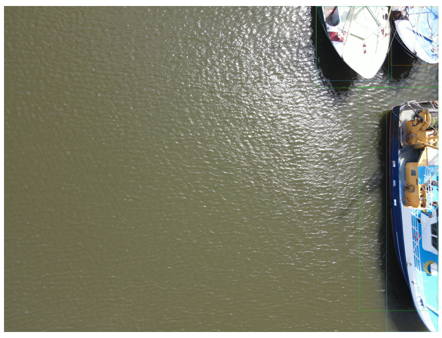
(c) Dataset 3: Ameide oblique

Figure 5.1.: Precision-recall curves pretrained model

The images from dataset 1 are recorded in a predefined flight path, with the camera in the nadir direction. Because data is collected in a harbour, many different boats are captured in this dataset, leading to a high boat variety. Also, because the drone flew at an altitude of approximately 30 meters, many large boats could not be fully captured in one image. As a result, many of the ground truth labels are boat objects that contain only part of the boat. Figure 5.2 shows some examples of these cases. This could partly explain the low accuracy of this dataset. Another explanation for the low accuracy could be that the model assigns the wrong class to the right bounding box. The YOLOVv3 model used here is pretrained on the COCO dataset, which contains 80 classes. This is investigated at section 5.1.2.



(a)



(b)

Figure 5.2.: Images with boat parts

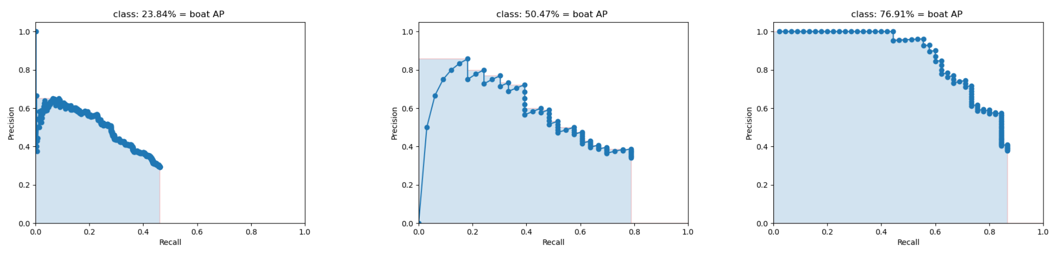
5.1.2. Results pretrained model with all classes

In this section, an investigation is done on whether the pretrained model could be predicting the right labels, but with the wrong class. In the previous section, only predictions of the boat class are taken into account. In this section, the accuracy is calculated using all predicted labels, regardless of class. All classes are thus seen as boat class. Table 5.2 displays the results. As anticipated, the number of false positives is quite high for all datasets. This is the result of additional objects in the images found by the model wrongly being seen as boat class in this method.

Table 5.2.: Detection results pretrained model all classes

Dataset	AP	True Positives	False Positives	False Negatives	Ground truth objects	Detected objects
1	23.84%	217	526	254	471	743
2	50.47%	26	50	7	33	76
3	76.91%	39	64	6	45	103

Figure 5.3 also displays this. For datasets 1 and 2 the recall is a lot higher, meaning that there are more correct predictions being made. This indicates that some bounding boxes around boats are predicted with the wrong class in 5.1.1. When comparing with figure 5.7, it is noticeable that the precision in 5.3 is a lot lower. This is the expected result of the high false positive rate. When the model makes a prediction, the chance is lower that this prediction is correct. In dataset 1, this results in the AP being lower than in 5.1.1. The reason for this is that this dataset was collected partly over land, resulting in many additional objects like cars being correctly predicted by the model. When all these additional classes are being converted to boat, this leads to many false positives. In dataset 2 this is not the case, because less additional objects were being predicted, so the precision is higher.



(a) Dataset 1: Den Oever (b) Dataset 2: Ameide nadir (c) Dataset 3: Ameide oblique

Figure 5.3.: Precision-recall curves pretrained model, all classes

Dataset 2 shows a much higher AP than in the previous section in Table 5.1. The other two datasets show little difference in AP. From this we can conclude that in dataset 2, many labels were in the right position around the boat, but contained the wrong class. Figure 5.4 shows some examples of this. The pretrained model predicted the bounding boxes in the right position, but with the wrong classes 'skateboard' and 'snowboard'.

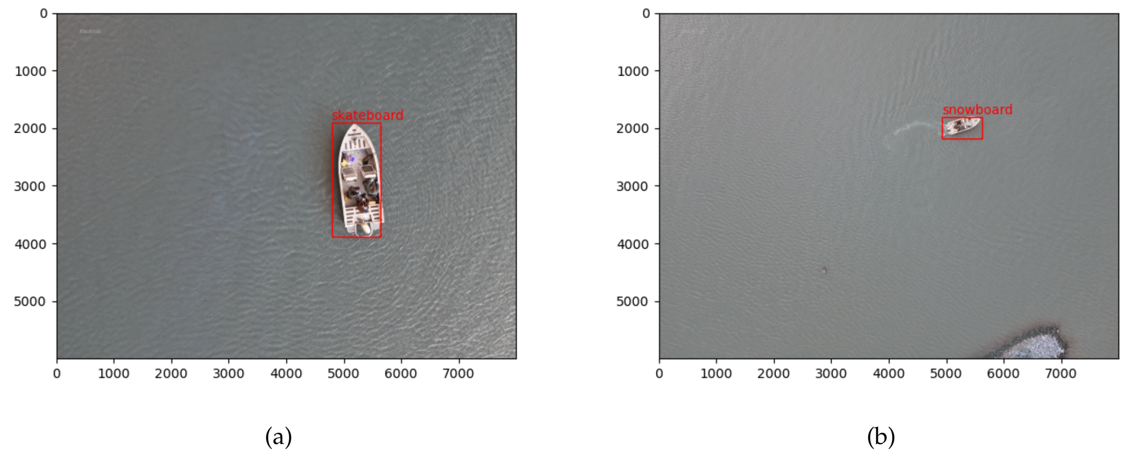


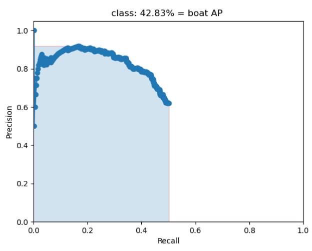
Figure 5.4.: Predictions with incorrect classes

5.1.3. Results trained model

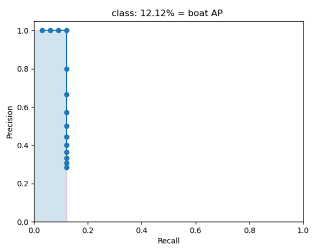
Table 5.3 and Figures 5.5 and 5.7 display the results of the model trained for this research. When looking at the average precision for the standard IoU ratio of 0.5, datasets 1 and 2 score significantly better than the pretrained model. Dataset 3 with oblique imagery scores significantly worse for this model, which is to be expected. The goal of training the model is to improve the nadir boat detection, hence why only one oblique image was added to the training dataset.

Table 5.3.: Detection results trained model

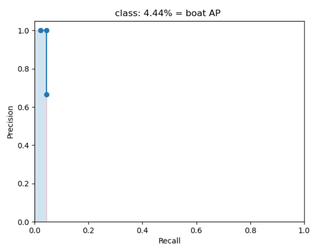
Dataset	IoU	AP	True Positives	False Positives	False Negatives	Ground truth objects	Detected objects
1	0.5	42.83%	236	144	235	471	380
1	0.1	68.55%	330	50	141	471	380
2	0.5	12.12%	4	10	29	33	14
2	0.1	42.42%	14	0	19	33	14
3	0.5	4.44%	2	1	43	45	3
3	0.1	4.44%	2	1	43	45	3



(a) Dataset 1: Den Oever



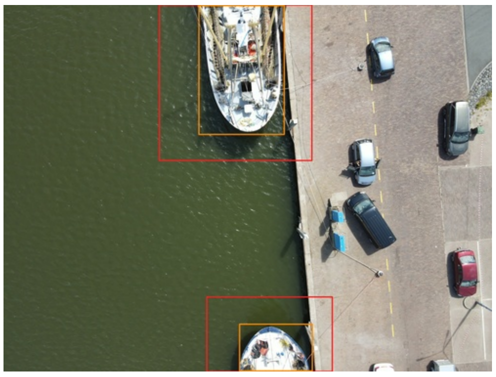
(b) Dataset 2: Ameide nadir



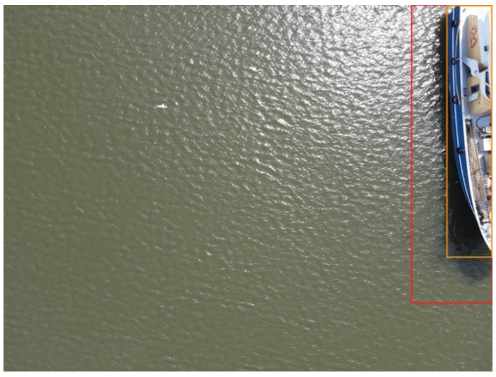
(c) Dataset 3: Ameide oblique

Figure 5.5.: Precision-recall curves trained model IoU 0.5

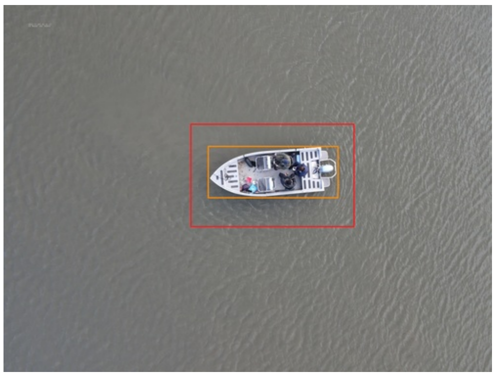
Noticeable is the large proportion of false positives in dataset 1 and 2. In Figures 5.5a and 5.5b this can also be seen. The precision stays below 1.0 and in figure 5.5b a sharp drop can be seen at a recall of 0.15. This high number of false positives could indicate that predicted the bounding are in the right position, but do not overlap the ground truth bounding boxes enough to be considered true positives. A visual inspection was done on the output of these datasets, and it was concluded that this is causing the large number of false positives. Figure 5.6 shows some examples of this. The predicted bounding boxes are shown in red and the ground truth in orange.



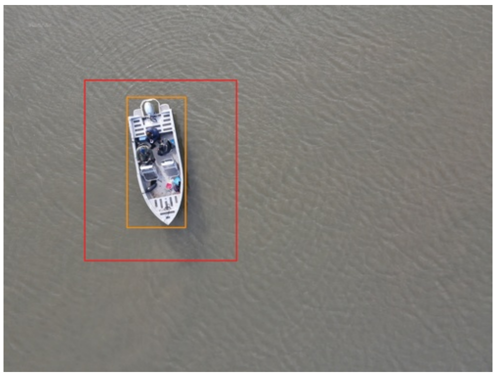
(a) Dataset 1



(b) Dataset 1



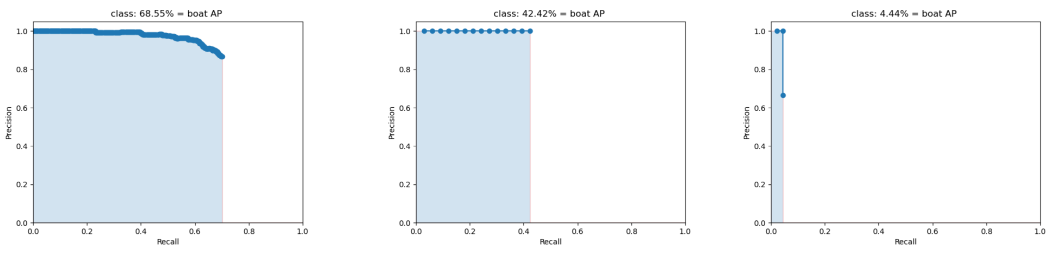
(c) Dataset 2



(d) Dataset 2

Figure 5.6.: Little overlap between predicted (red) and ground truth (orange) bounding boxes

To avoid predicted bounding boxes in the right position with little overlap being sorted into the false positives, the IoU ratio was decreased to 0.1. In that way, predicted bounding boxes need to be less overlapped with ground truth bounding boxes to be considered valid. In Table 5.3 it is shown that lowering the IoU increases the average precision notably. In 5.7, the increase in precision and recall is also noticeable. Figure 5.7a displays that the precision is a lot higher than in 5.5a and has a less steep downwards slope. In dataset 2 the increase in precision is even larger. In figure 5.7b the precision is at 1.0 at every recall value. This indicates that in datasets 1 and 2, the trained model predicted many bounding boxes with a low overlap to the ground truth. Figure 5.6 shows that the predicted bounding boxes are mostly larger than the ground truth bounding boxes.



(a) Dataset 1: Den Oever (b) Dataset 2: Ameide nadir (c) Dataset 3: Ameide oblique

Figure 5.7.: Precision-recall curves trained model IoU 0.1

5.2. Object positioning

In this section, the results of the positioning algorithms for the camera in the nadir and oblique direction are displayed. Dataset Ameide was split into 4 subsets to investigate the performance of the algorithms on difference camera angles and boat movement. All ground truth coordinates are compared to the calculated coordinates by the model. Table 5.4 shows the mean absolute error in meters of for the 4 subsets. More detailed results can be seen at tables A.1, A.2, A.3 and A.4. These tables show the difference in meters of the latitude, longitude and Euclidean distance between the ground truth and the calculated coordinates for each image. For each subset, the average and absolute average is also calculated. Four scatter plots are created to visualise the latitude and longitude errors of the four subsets. A line is drawn at 0 for both axes to divide the plot into four quadrants to visualise the distribution of the errors more clearly. In Table 5.4, it is shown that the errors of the positioning of a motionless boat are lower than for a moving boat. Additionally, the oblique algorithm has higher errors than the nadir positioning with the same boat movement. In sections 5.2.1 and 5.2.2 these results will be discussed further for the nadir and the oblique algorithms respectively.

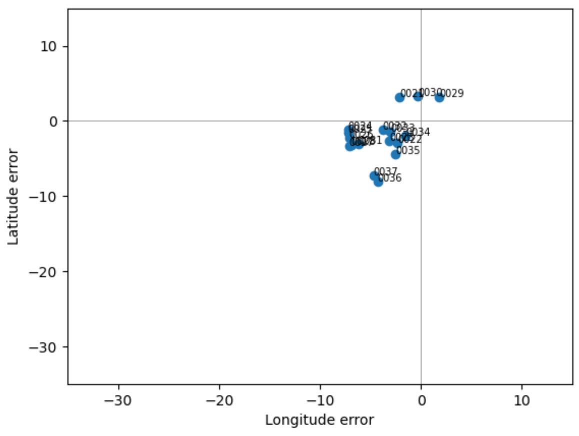
Table 5.4.: Average absolute errors in meters

Positioning accuracy in meters		
	Motionless	Moving
Nadir	5.6	9.7
Oblique	8.2	19.6

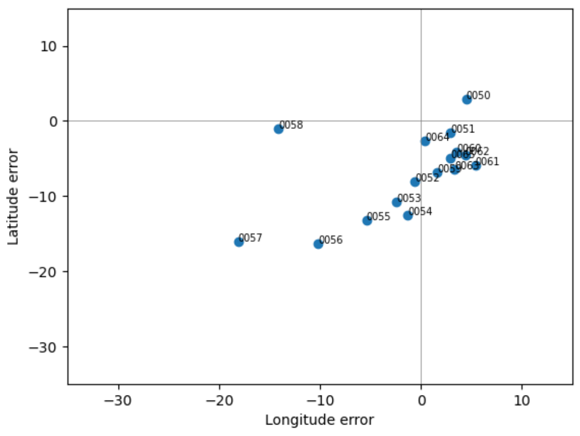
5.2.1. Results nadir positioning

Table A.1 shows the distance in meters between the ground truth and calculated coordinates for images of a motionless boat at a nadir position. There is a slight difference between the latitude and the longitude average. The error of the different images has quite a large range. The average shown in all positioning accuracy tables is the absolute average. Because the error values for latitude and longitude are squared in the Euclidean distance calculation, the total error values are all absolute values. This results in the average and absolute average being the same value at all times for this column. Figure 5.8a shows the scatter plot of the latitude and longitude errors. The errors seen highly clustered. Most of the errors seem to have a negative value for both the latitude and the longitude.

In Table A.2, the nadir results of a moving boat are shown. The errors are slightly higher than the Table A.1, which is expected, because a moving boat and drone could cause inconsistencies, both in the drone metadata and the ground truth data. The mean absolute latitude error is slightly higher than the longitude error, in contrast to Table A.1. Figure 5.8b visualises the errors of this subset. In contrast to the errors of the coordinates of the motionless boat, the errors seem spread out. Almost all the errors seem to have a negative value for the latitude. This indicates that the calculated latitude is higher than the ground truth latitude. This could be due to the fact that the algorithm overestimates the latitude shift that is needed from the metadata latitude to the latitude of the object.



(a) Camera in the nadir position, motionless boat



(b) Camera in the nadir position, moving boat

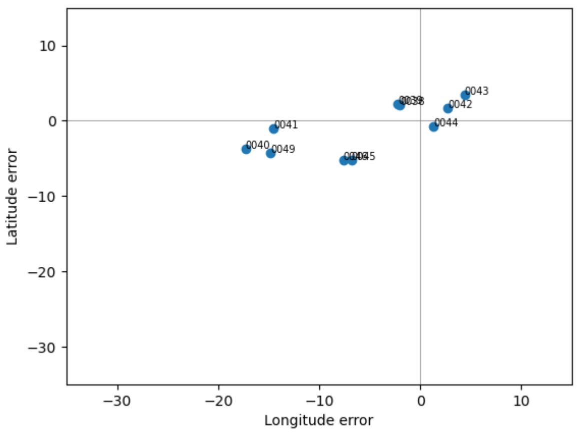
Figure 5.8.: Nadir positioning error in meters

5.2.2. Results oblique positioning

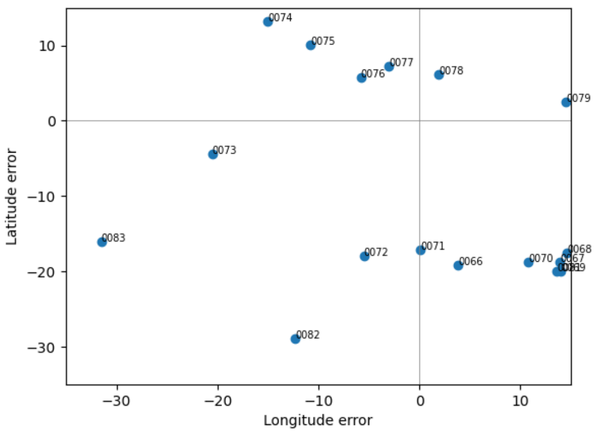
In this section, the results of the oblique positioning algorithm are shown. Table A.3 expresses the results of the errors of the drone images at an oblique angle, with the boat lying still. In Table A.4, the results with a moving boat are shown. Figure 5.9 shows the scatter plot of the latitude and longitude errors of both subsets.

The absolute average longitude error of the oblique algorithm with a motionless boat in Table A.3 is reasonably higher than the longitude error. However, in Table A.4, the absolute average latitude error is slightly higher than the longitude error, which does not indicate a pattern. In 5.9a, the errors of the calculated coordinates of the motionless boat are visualised. Like in Figure 5.8a, the errors seem more clustered. The most errors seem to be negative for both the latitude and longitude error, but this is not very distinct.

As expected, in Table A.4 the absolute average errors are higher than for a boat that is not moving, both in the latitude and longitude direction. Similar to Figure 5.8b, in Figure 5.9b the errors are very dispersed. For this subset, there is no distinct trend in negative or positive for both axes. When looking at the image names, it can be seen that images taken consecutively are in usually the same quadrant.



(a) Camera in the oblique position, motionless boat



(b) Camera in the oblique position, moving boat

Figure 5.9.: Oblique positioning error in meters

Concluding, there exists an error of several meters for all of the four subsets. As expected, the errors of the subsets with the moving boat are larger and more dispersed than the errors of the motionless boat. Because the boat is moving, the ground truth may be less accurate. Additionally, because of rapid movements of the drone to follow the moving boat, the image metadata quality is expected to decrease, which directly translates to the positioning accuracy. The errors of the oblique subsets are larger than its nadir counterparts. This is due to the fact that inaccuracies of the image metadata have a larger influence on the results when the translation is larger.

5.3. Real time connection

In this subsection, the real time functionality is assessed. Firstly, the connection of the drone controller with the cloud is assessed in section5.3.1. Then, the results of the prototype speed test are described in section 5.3.2.

5.3.1. Connection drone controller with Google Drive

As mentioned in section 3.4, the method used to make a real time connection to the prototype is to make a connection to and from Google Drive. The drone used for this research is the DJI Mavic 2 Enterprise Advanced, which is controlled with a smart controller. This controller can be connected to WiFi and has an android environment. As mentioned, the goal is to install a syncing app on the smart controller that syncs local folders to Google Drive. When trying to install this app called FolderSync, it was found that no apps could be installed through the verified Google Play Store, because it did not exist on this device. To install apps without the Google Play Store, websites that host the raw installation .apk files can be used. However, these websites are not verified and are thus more prone to computer viruses. After consultation with the owners of the drone, it was decided to install the app with the .apk hosting website Apkpure. The Foldersync app was installed successfully and linked to Google Drive and local data.

However, this brought to light two problems with this method. First of all, the cache folder is not filled directly when the images are made. It is filled at an unspecified time from the image capture, which could be as late as when the drone has already landed. A workaround for this problem was found. The person holding the controller could manually export images in the DJI Fly application to Google Drive. This needs to be done separately for every image, and is thus not desirable.

The second problem with this method is the image quality. The images that are received by the smart controller are of lower pixel size than the images stored on the drone its self. The drone images stored on the drone are of size 6000X8000 and the ones cached to the smart controller are of size 960X720 pixels. The lower pixel size does not have a critical influence on the output. The lowered image quality does have another crucial flaw: images received by the controller do not contain the GPS and camera metadata, which makes them unusable by the positioning algorithm. These findings are not mentioned in the DJI smart controller documentation. Finally, a solution was found for both of these problems to stream the images with the correct metadata to Google Drive. When an image is captured, it is also stored in the DJI fly app in the smart controller. In this app, the preview of these images can be seen. For each image, there is also a download button available. When pressing this button, the image is downloaded in full 6000X8000 size with the needed metadata. The image is downloaded to the ./djipilot/DJIPilot Album, which is synced to Google Drive with the Foldersync app. The cache folder is thus not needed. This app syncs the images in the folder every 5 minutes, which is the most frequent option this app contains. In this way, there is a near real time connection with the cloud and prototype. Although not fully automatic, because the person with the drone controller has to manually download the images in the DJI Fly application.

5.3.2. Speed of the real time connection

In Table 5.5, the speed in seconds of different components of the prototype method are recorded for 10 random images. For the columns 'Download full size' and 'To Drive' the time was measured with a stopwatch. In the First column, the time to download one image to get the full size 6000x8000 on the DJI Fly app was measured for 10 different images. In the next column, 'To Drive', the time was measured to upload an image from the controller to Google Drive by the Foldersync app.

The speed of the remaining columns were measured in Python with the time library. The start time and end time of different parts of the code of the prototype were recorded, subsequently the time difference between the start and end time could be calculated. For the 'Detection' column, the time to inference one image was measured. In the 'Positioning' column, the time to convert the pixel coordinates to latitude and longitude was measured. A 0.0 indicates that the time to carry out this code block was lower than 0.1 seconds. In the 'Write to file column' the time was recorded to add the point and polygon data to the layers in ArcGIS Online. The last column, 'Code Total', stores the time it takes for one image to be fully processed. So the total time for one image can be considered as 'Download full size' + 'To Drive' + 'Code total'. The average of this is $2.53 + 9.97 + 13.21 = 25.72 seconds$.

The components that rely on an internet connection have a large influence on the results. The number of images stored on Google Drive also has a large influence on the total running time. In each loop, these images need to be retrieved and for each image it needs to be determined if it has been processed or not. This means that the more images are stored on Google Drive, the longer processing takes in the code. At the time of testing, there were 10 images stored on the linked Google Drive.

Table 5.5.: Speed real time connection

Recorded speed in seconds							
Image	Download full size	To Drive	From Drive	Detection	Positioning	Write to file	Code total
1	3.26	8.32	0.29	2.19	0.0	3.68	13.84
2	3.19	8.75	0.30	0.99	0.0	4.01	9.34
3	2.29	12.48	0.29	0.96	0.0	3.13	12.23
4	2.30	13.50	0.31	0.99	0.0	3.08	14.70
5	2.64	9.53	0.37	0.97	0.0	4.13	12.66
6	2.22	7.06	0.26	0.97	0.0	2.99	13.50
7	1.80	9.35	0.30	0.98	0.0	3.42	13.19
8	3.05	10.46	0.32	0.98	0.0	3.38	13.58
9	2.43	9.30	0.30	0.98	0.0	3.18	12.96
10	2.10	10.99	0.31	0.98	0.0	3.89	16.11
Average	2.53	9.97	0.31	1.01	0.0	3.49	13.21

6. Conclusion and future work

6.1. Discussion & Limitations

In this section, the research project is discussed and the research limitations are described. Firstly, the results from Chapter 5 are thoroughly discussed and the limitations of the research method are described. After, the use of drones in this research project is evaluated. Finally, a discussion is made on the results and limitations of the use case of this research.

6.1.1. Discussion of the results

When looking at the average precision of the pretrained detection model on each of the three datasets, it is clear that it is too low to directly use in production. The difference in AP between the nadir and oblique datasets are noticeable. An expected explanation for this is that the boat class of the COCO dataset, which the model is trained on, has been trained mostly on images from an oblique angle. A method to improve the detection accuracy would be to collect data from the object of a nadir angle, label this data and train the model. To improve the performance of the prototype, the pretrained YOLOv3 model was trained further using mostly nadir imagery acquired in this research. This has been proven to increase the accuracy. It was also found that the pretrained model misclassifies some predictions. Taking these findings into account, it is recommended to train the pretrained model when using the prototype for other use cases. In that way, the accuracy is expected to be higher and misclassification is not possible due to that fact that the model is focused only on one class.
The pretrained detection model used for this research is trained on the COCO dataset, which consists of 80 classes. This model can be used for further training for other use cases where other objects present in the COCO classes list should be detected. Because this pretrained model is used, the prototype can be directly used for other use cases. Moreover, it was found that using the Python for ArcGIS API, detection models could be loaded in and directly used for inference. Using and training the model can be done in ArcGIS Pro, which provides a visual interface where intermediate steps of the training can also be shown on a map. This is an advantage for other use cases, where inspectors with little programming experience may need to train the model.

Although results are reasonable for positioning and useful for the use case of fisher boat detection, it is clear that due to some reasons, there is still a significant error in the calculated coordinates. One of the main reasons for this error is the inaccuracy of the metadata. As mentioned in section 4.2, certain metadata parameters are used as input in the positioning algorithm. When one or several of the parameters are less accurate, this directly translates to loss of accuracy of the positioning algorithm. As mentioned in 4.2, the AbsoluteAltitude parameter was so inaccurate, it was not usable to produce credible results. This could be improved in future projects by using a Real Time Kinematic (RTK) module on the drone. For

this research, the RTK module was unfortunately not available. Another option that could lead to improvement is to use a more professional drone. The drone used for this research was relatively light and inexpensive. A heavier drone would experience less influence of factors like wind. Also, a more expensive drone is expected to carry more accurate equipment to record the metadata, which would lead to an improvement of accuracy. Another solution which could improve the drone metadata accuracy is to determine the drone fly times based on the GPS constellation. The recorded metadata GPS position is gathered from a connection with the satellites in this constellation. During certain times of the day, more satellites could be in sight of the drone, thus a more accurate position can be recorded.
Besides the inaccurate metadata, the simplification of reality could also lead to a loss of accuracy. In the nadir positioning algorithm, the assumption is made that the camera is facing nadir at all times. Additionally, in both algorithms we assume that the terrain is entirely flat. In this use case, for inspection flights in The Netherlands over water bodies, these are credible assumptions. For autonomous flight, the DJI Fly app requires a predefined path with the camera facing downwards at all times. Moreover, the surface over inland water is mostly flat. When using this method on sea, waves could influence the flatness of the terrain and thus the accuracy of the output. When using this method to detect objects on land, the outcomes are highly dependent on the terrain. In the Netherlands, the terrain is relatively flat in most cases, so acceptable results are expected. When using this method in a mountainous or hilly area, it is expected that this could highly influence the results. This can be seen in Figure 6.1, the trigonometric approach for calculating the ground distance does not hold true for uneven terrain. It is recommended that the user should use a DEM to account for this uneven terrain.

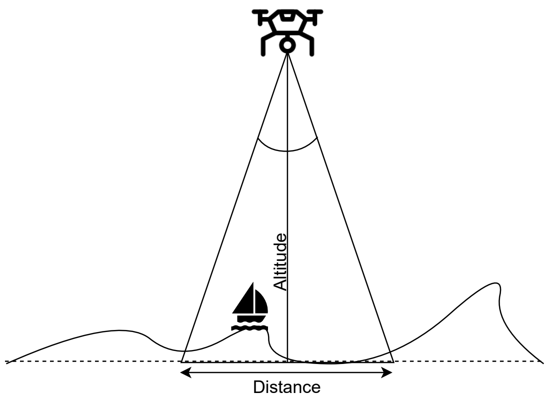


Figure 6.1.: Influence terrain on positioning approach

Ideally, the time for the prototype to process one image is less than the time between two image captures by the drone. The time between two images captures is 2.5 seconds when the drone is flying a predefined inspection flight plan. As mentioned in section 5.3.2, the average time to process one image is 25.72 seconds, much larger than the capture gap of two seconds. This means that images will be processed with a huge delay while the drone is in the air. For this use case, a delay of minutes is not ideal, but not a very big problem. Inspectors still need to inspect the images and make their way to the incident location. However, for other use cases, like search and rescue, this could lead to serious loss of usability. About half of the processing time can be accounted for by getting the images to and from Google Drive. An improvement on time could thus be to install the prototype on the controller and

leave out Google Drive. This could largely decrease the uploading and downloading time needed. Through ArcGIS Runtime, many ArcGIS functionalities, like writing data to layer files, are available on Android. Loading in the deep learning model can also be done offline by downloading the model.
A large part of the processing time of the Python code can be accounted for by the uploading and downloading of data. For uploading the data for the dashboard to ArcGIS Online, an ArcGIS function is used. Consequently, little can be altered in the code to improve the speed. When searching for new images in Google Drive with the API, the whole cloud is inspected to search for files that fit the requirements. With this API, it is not possible to limit the search to just one folder on Google Drive. This could potentially influence the speed when there are many files stored on the cloud.
As mentioned, the connection also has a large influence on the components of the prototype that use this connection to upload and download data. This can be an issue, because the connection can be unreliable in the field, which leads to unstable performance. This is expected to improve, because wireless connections continue to improve in the past years and are expected to do so in the future. Concretely, with the introduction of 5G in the upcoming years and the recently launched Starlink connection for remote areas.
Another limitation of the real time connection with the controller is that the connection could not be created fully automatically. A pilot still needs to press the download button to receive the drone images in high quality with the necessary metadata. There however exist a large variety of drones and drone controllers. It is possible that for other controllers, the downloading of the images to full quality would not be necessary. This research was limited to the drones and controllers available by the NVWA, for which it was concluded that it is not possible.
When the drone is flying a predefined path and capturing images every few seconds, it is a given that large parts of images overlap with each other. Not all images are thus crucial to process when other images can convey the same information. A method to decrease the delay in processing could thus be to increase the time gap between incoming images to be processed. A method for this is described in [Dhanda et al., 2018] where they use metadata to filter out redundant images.

It is important to consider that the detection results could influence the positioning results. In the experiments of this research, the detection and positioning were tested separately. This was done by using the ground truth bounding boxes as input for the positioning algorithm instead of the detected bounding boxes. Figure 5.6 displays the possible influence the detection output could have on the positioning result. In orange, the bounding box of the ground truth is given and in red, the bounding box of the detection result is shown. As can be seen, the predicted boat appears to be larger than it actually is. When using the prototype, the bounding box of the detection result would be directly translated to a polygon on the dashboard. The polygons on the dashboard are thus not guaranteed to directly relate to object size and have to be interpreted with caution. When the predicted bounding boxes do not directly overlap the ground truth, so the bounding boxes are not situated at the right position in the image, the positioning accuracy will also decrease as a result.

6.1.2. Use of drones

In this research, drones were used to detect and localise certain objects. This means that all data collection and testing needs to be carried out in the field. The use of a drone for research proved to be quite challenging, due to several reasons.

First of all, research with drones was challenging from a logistic point of view. Because a pilot licence is needed to fly the drones, drone pilots from the NVWA carried out the necessary data acquisition. There was considerable communication necessary for these data acquisition instances to take place. A drone pilot needed to be arranged who was willing to help with my research. Also, a date, time and fitting location needed to be discussed. Flying drones is not allowed in all Dutch airspace, so the location had to be carefully chosen. Also, a detailed flight plan for each data acquisition instance had to be created to let all involved parties know what the goal of the plan was. Additionally, for on ground testing with the drone and its controller, these items needed to be borrowed from the NVWA. When alterations needed to be done, these obviously had to be mutually agreed to by the drone owner. The persons involved with the NVWA were of very great help. But nevertheless, these aspects still made the research more time-consuming.

Secondly, it is important to note that fully autonomous flight is not allowed yet in the Netherlands. The current law imposes that the drone pilot should keep the drone in sight at all times. This means that autonomous inspection flights at interesting locations are restricted by law at this time. However, there exist signs that laws on this are about to change, at least for certain parties like the NVWA. Recently, a testing area has been fitted at Katwijk called the BVLOS corridor. This corridor runs from the former Valkenburg Air Base to the North Sea and was created to test autonomous drone flights without a pilot in the light of sight[1].

The original idea was to use drone video instead of images. However, due to the metadata not being available for videos taken with DJI drones, this method was not possible at this time. Section 6.3 describes a possible future method should the metadata be made available for videos.

6.1.3. Use case discussion

Although this method looks promising to automate the detection of fisher boats in restricted areas, some remarks need to be made. When using predefined inspection flight times, these need to be variable. Otherwise, fishers with bad intentions will know to avoid the restricted areas during inspection times. Additionally, inspections flights would also need to be carried out by night, as it is expected that many illegal activities would be carried out in a low light environment. To detect fisher boats during nighttime, an infrared camera is necessary. The detection algorithm would also need to be trained on this type of imagery. So with the current detection model, detection on infrared images is not possible yet. Also, to prevent boats from being detected, fishers could alter their boat to make it undetectable by the model, like using camouflage colours or high reflectance material. To prevent this from happening, the detection model needs to be trained for these types of situations as well.

The storage of all data is also important to discuss. An inspection flight will usually lead to hundreds of images in high quality, which all need to be stored. These images are being stored on Google Drive, but as mentioned before, increase in files on Google Drive leads to

[1]https://unmannedvalley.nl/en/news/press-release-first-bvlos-corridor-in-the-netherlands/

a large increase of the processing time of the prototype, because limiting the image search to a certain folder is not possible. It is therefore recommended to clean out the Google Drive connected to the prototype at least once every 24 hours and store the images elsewhere. The produced features added to ArcGIS online are added to a layer that has a filter to show only the features of the previous 24 hours. When features are continuously adding to the layer in ArcGIS Online, processing of the statistics on the dashboard and visualising the layer becomes more computationally expensive. Recommendations on when to back up and clear the feature layer depends on the number of incoming features and thus the number of inspection flights. It is expected that ArcGIS Online has no trouble handling feature layers with up to one million features.

To make this method more usable for other use cases, a tracking functionality needs to be added, that can track the detected objects. In the current method, images are processed separately, so the case where the object is present in both images is not handled yet. Several solutions to this issue were investigated and are described in section 6.3.

When working with visual data of the public space, privacy can be an issue. A solution for this could be to automatically blur faces on the drone images, this can be done with a deep learning model. Additionally, sensitive data can be prone to hacks. It is therefore recommended to limit access to servers where data is stored, like Google Drive and ArcGIS Online and use protected log-in like two-factor authentication. Google Drive at least as secure as other cloud platforms, as it uses a strong encryption when transferring files and has the option to use 2-factor authentication.

6.2. Conclusion

The aim of this research is to develop a prototype to detect and localise objects in real time. This was done by integrating several components, like the deep learning model, the positioning algorithm, the real time connection and the localisation dashboard. To carry out this aim, the following research question was defined: *To what extent can drones be used to localise objects in real time?* In the previous sections, the research method, implementation, results and analysis are described. Based on these sections, a conclusion of the research will be given. Firstly, answers to the subquestions and the main question will be given. Then, the contributions of this research are described.

6.2.1. Research Questions

In this subsection, the research questions are answered and evaluated.

1. **How can deep learning be used to detect objects on drone images?**

   To answer this research question, an investigation was done on the available deep learning options to detect objects. The YOLOv3 detection model from the Python for ArcGIS API was found to be the most fitting. The YOLOv3 model is the best fit for real time use, because it had the most optimal tradeoff between inference speed and accuracy. The model is pretrained on the COCO dataset, which makes it directly usable for 80 classes. A drone image can be inputted into the model, which predicts the bounding box around the object, the class and the prediction probability. Moreover, to increase the nadir boat detection accuracy, the model was trained with data acquired in this research. It was found that to increase accuracy and avoid misclassification, the pretrained model needs to be trained.

2. **How can detected objects be automatically positioned in a geographical coordinate system?**

   To investigate this research question, multiple methods were researched. Because of processing time constrains due to the real time functionality, an algorithm is implemented for images taken with a nadir angle that calculates the meters per pixel using the image metadata. The necessary input for this is: the drone GPS coordinates, drone camera angle and height. To make this method useful for more use cases, a positioning algorithm is developed for objects on images taken with an oblique camera angle. Because both algorithms directly use the image metadata, the accuracy has a very large influence on the results. It was found that the used drone has an acceptable metadata quality for positioning in this specific use case. However, if more accurate results are needed, the drone metadata quality needs to be improved. To visualise and localise the geographical coordinates of the object, the features are stored in a map layer in ArcGIS Online, which is visualised as a map on a dashboard.

3. **What hardware and software is needed for this method to be carried out in real time?**

   The drone used in this research has no real time image streaming functionality built in, so this had to be constructed. The hardware required for the constructed method is a controller that can be synced with the cloud and a device to run the prototype code on. The needed software a sync application, the cloud and the Python code of the prototype. To create a real time connection, a connection with the cloud is made through Google Drive. On the smart controller, an app is installed that automatically syncs local files with the cloud. The images can then be retrieved from Google Drive with Python code using the Google Drive API. Unfortunately, this method could not be made fully automatic because full quality images with the necessary metadata could only be stored locally on the controller after a manual download. Because of the multiple uploading and downloading, the speed of this method is dependent on the connection of the controller and the device running the prototype code. Current testing points out that with the current method, the time interval between image captures is much smaller than the duration to process one image, which leads to delays. Solutions for this problem are running the code directly on the controller or increasing the time interval between image captures by removing redundant images.

After the sub-questions are answered, the main research question can be answered. To what extent drones can be used to localise objects in real time depends on the specific use case and drone used. For the use case of finding fisher boats in restricted areas with the Mavic 2 Enterprise Advanced drone, it is possible, but with a few remarks. The expectation for this use case is that inspection flights with a predefined path will be carried out with a nadir pointed camera. As concluded, the detection accuracy of the boat class with pretrained model is not very high, and it was found that this can be increased by training the model. To increase detection accuracy and avoid misclassification, it is therefore recommended that the pretrained model is trained further for the class needed in the use case. Additionally, the accuracy of the image metadata will directly influence the accuracy of the positioning. However, the found accuracy is satisfactory for this use case, as visual inspection is still necessary. The real time functionality could not be made fully automatic for this drone and the image processing will be delayed due to the upload and download time. For this use

case, the objects do not need to be localised immediately on the dashboard, so this method is suitable.
For other use cases, these results could be less sufficient, depending on the need to have very accurate results for detection of boats, positioning or speed. Several recommendations to increase the accuracy of the results are given in 6.1. Nevertheless, this method seems a good starting point for the localisation of objects with drones that can be used for a variety of use cases.

6.2.2. Contributions

This research integrates several components to localise objects on a map in real time with the use of drones. The separate components have been researched before, but have not been integrated in such a way before to localise objects from drone images. With this integrated method, objects on drone images can be localised on a map in a dashboard while the drone is still in the air. The current methodology is highly adaptable for many use cases that require the geographical location of objects on drone images in real time. To carry out this method, a prototype was constructed using the programming language Python and a variety of software libraries. The prototype gives a good estimation on the position of detected objects. With this method, many in-person inspections could become redundant, saving money and time. Additionally, the created method allows multiple inspection flights to be processed and visualised at once. This research also provides insight in the usability of relatively light drones for research purposes.

6.3. Future work

When multiple images of the same boat produce a predicted polygon, it seems like there are many boats, while in truth there is only one. To solve this issue, the boats need to be tracked. Several methods were investigated. Two of these methods which seemed the most viable to do further research on are described.
The first method is based on template matching and can be implemented directly into the prototype. In this method, a smaller template image is compared to another image. This is done by sliding the template across the other image and comparing the pixel colour values. In the prototype, it can be used to compare objects from two sequentially captured images. For each previous image where a boat is detected, the boat bounding box is cut out. For this cutout, a match is attempted to be found in the next image. If there is a match, this means that the same boat is present at both images. The template matching functionality can be directly implemented using the OpenCV Python library.
Another proposed method for further research is the use of FMV. This is an extension for ArcGIS that that can geospatially analyse video data. When a local video or video stream is available together with a FMV-compliant metadata file, the dynamic field of view of the camera can be shown on a map. Figure 6.2 displays a screenshot of this functionality. This FMV image stream can be combined with a deep learning model for object tracking on video[2]. Once an object is found, it can be tracked, which is shown directly on the map as points connected by lines.

[2]https://pro.arcgis.com/en/pro-app/latest/help/analysis/image-analyst/object-tracking-in-motion-imagery.htm

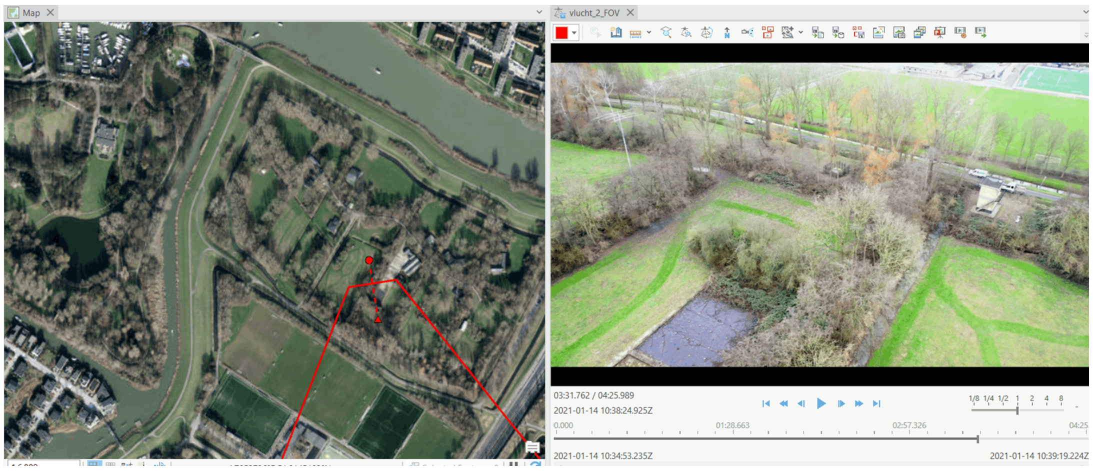


Figure 6.2.: Screenshot FMV functionality in ArcGIS Pro

For now, this functionality is available only for ArcGIS Pro, so created layers still have to be automatically uploaded to ArcGIS Online. The main issue why this has not been implemented yet is because many drones do not create a flight log for video captures. Using the DJI Fly app for the drone used for this research, no flight log was created for video. With other drone flying applications, like ArcGIS Site Scan a flight log is created after flight, on which still numerous alterations had to be made to make it fit for FMV use. The fact that many drone fly apps produce a flight log for videos only after landing, if at all, poses implications for the real time functionality. Nevertheless, this method has high potential because of the detection and oblique tracking capabilities on video stream. It therefore is recommended to look into further if drone or fly apps exist where the drone flight is logged in real time in a format easily fit for FMV. If the flight log of the drone is not available, it would be interesting to research if the predefined flight plan could be converted to the right flight log format for FMV. In this flight plan, the start and end time, as well as the flight track is recorded. The camera is always in the nadir position, so in theory all the necessary metadata is recorded in the flight plan to be able to convert it to a flight log.

Several potential solutions to limitations, mentioned in section 6.1, could also be considered interesting topics to research further in the future. It would be appealing to research in what way the positioning accuracy could be improved. For example, the quantification on how the metadata quality influences the positioning accuracy could be researched. Also, the influence of a drone with a RTK module or more precise equipment on the positioning accuracy could be investigated.
As theorised in 6.1, running the prototype on the controller or on the drone itself could decrease the processing time. It would be interesting to research the feasibility of this theory and what software and equipment is needed to achieve this.
Finally, it would be worthy to research different drones and controllers to find the drone equipment that is most fit for these type of monitoring use cases. The quality of this drone was scarcely suitable for this use case, but would not be fit for other monitoring use cases. As mentioned in this section, a drone which records a flight log with drone streaming capabilities fit for FMV would be most optimal.

A. Positioning errors

Table A.1.: Positioning accuracy with nadir camera and motionless boat

Positioning accuracy in meters			
Image number	Longitude error	Latitude error	Total error
21	-2.1	3.2	3.8
22	-2.3	-2.9	3.7
23	-3.1	-2.7	4.1
24	-7.2	-1.2	7.3
25	-7.2	-1.6	7.4
26	-7.1	-2.3	7.5
27	-7.1	-3.3	7.8
28	-6.9	-3.2	7.6
29	1.8	3.2	3.7
30	-0.3	3.3	3.3
31	-6.2	-3.0	6.9
32	-3.8	-1.2	4.0
33	-3.0	-1.4	3.3
34	-1.5	-1.9	2.4
35	-2.5	-4.4	5.1
36	-4.3	-8.0	9.1
37	-4.7	-7.3	8.7
Average	-4.0	-2.0	5.6
Absolute average	4.2	3.2	5.6

Table A.2.: Positioning accuracy with nadir camera and moving boat

Positioning accuracy in meters			
Image number	Longitude error	Latitude error	Total error
50	4.5	2.9	5.4
51	2.9	-1.5	3.3
52	-0.6	-8.1	8.1
53	-2.4	-10.7	11.0
54	-1.3	-12.5	12.6
55	-5.4	-13.2	14.3
56	-10.2	-16.3	19.2
57	-18.1	-16.0	24.2
58	-14.1	-1.0	14.1
59	1.6	-6.9	7.1
60	3.5	-4.1	5.4
61	5.4	-5.9	8.0
62	4.4	-4.5	6.3
63	3.3	-6.4	7.2
64	0.4	-2.6	2.6
65	2.9	-4.9	5.7
Average	-1.5	-7.0	9.7
Absolute average	5.1	7.3	9.7

Table A.3.: Positioning accuracy with oblique camera and motionless boat

Positioning accuracy in meters			
Image number	Longitude error	Latitude error	Total error
38	-2.0	2.1	2.9
39	-2.2	2.2	3.1
40	-17.3	-3.7	17.7
41	-14.5	-1.0	14.5
42	2.7	1.7	3.2
43	4.4	3.5	5.6
44	1.3	-0.8	1.5
45	-6.8	-5.2	8.6
46	-7.6	-5.2	9.2
49	-14.8	-4.2	15.4
Average	-5.7	-1.1	8.2
Absolute average	7.4	3.0	8.2

Table A.4.: Positioning accuracy with oblique camera and moving boat

Positioning accuracy in meters			
Image number	Longitude error	Latitude error	Total error
66	3.8	-19.1	19.5
67	13.9	-18.8	23.4
68	14.6	-17.5	22.8
69	14.0	-20.0	24.4
70	10.8	-18.8	21.7
71	0.1	-17.1	17.1
72	-5.5	-17.9	18.7
73	-20.5	-4.4	21.0
74	-15.0	13.2	20.0
75	-10.8	10.1	14.8
76	-5.8	5.8	8.2
77	-3.0	7.2	7.8
78	1.9	6.1	6.4
79	14.5	2.5	14.7
80	19.8	-9.5	22.0
81	13.6	-19.9	24.1
82	-12.3	-28.9	31.4
83	-31.5	-16.0	35.3
Average	0.1	-9.1	19.6
Absolute average	11.7	14.0	19.6

Bibliography

Aburasain, R. Y., Edirisinghe, E. A., & Albatay, A. (2021). Palm tree detection in drone images using deep convolutional neural networks: Investigating the effective use of YOLO v3. In *Digital interaction and machine intelligence* (pp. 21–36). Springer International Publishing. Retrieved from https://doi.org/10.1007/978-3-030-74728-2_3 doi: 10.1007/978-3-030-74728-2_3

Apoorva, A., Mishra, G. K., Sahoo, R. R., Bhoi, S. K., & Mallick, C. (2020, July). Deep learning-based ship detection in remote sensing imagery using TensorFlow. In *Algorithms for intelligent systems* (pp. 165–177). Springer Singapore. Retrieved from https://doi.org/10.1007/978-981-15-5243-4_14 doi: 10.1007/978-981-15-5243-4_14

Bodensteiner, C., Bullinger, S., Lemaire, S., & Arens, M. (2015, December). Single frame based video geo-localisation using structure projection. In *2015 IEEE international conference on computer vision workshop (ICCVW)*. IEEE. Retrieved from https://doi.org/10.1109/iccvw.2015.136 doi: 10.1109/iccvw.2015.136

Bos, O. (2018). *Report on the eel stock and fisheries in the netherlands 2016/2017* (Tech. Rep.). Retrieved from https://doi.org/10.18174/445173 doi: 10.18174/445173

Bozzini, C., Conedera, M., & Krebs, P. (2012, September). A new monoplotting tool to extract georeferenced vector data and orthorectified raster data from oblique non-metric photographs. *International Journal of Heritage in the Digital Era*, *1*(3), 499–518. Retrieved from https://doi.org/10.1260/2047-4970.1.3.499 doi: 10.1260/2047-4970.1.3.499

Carranza-García, M., Torres-Mateo, J., Lara-Benítez, P., & García-Gutiérrez, J. (2020, December). On the performance of one-stage and two-stage object detectors in autonomous vehicles using camera data. *Remote Sensing*, *13*(1), 89. Retrieved from https://doi.org/10.3390/rs13010089 doi: 10.3390/rs13010089

Cartucho, J., Ventura, R., & Veloso, M. (2018, October). Robust object recognition through symbiotic deep learning in mobile robots. In *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (p. 2336-2341). IEEE. Retrieved from https://doi.org/10.1109/iros.2018.8594067 doi: 10.1109/iros.2018.8594067

Chang, Y.-L., Anagaw, A., Chang, L., Wang, Y., Hsiao, C.-Y., & Lee, W.-H. (2019, April). Ship detection based on YOLOv2 for SAR imagery. *Remote Sensing*, *11*(7), 786. Retrieved from https://doi.org/10.3390/rs11070786 doi: 10.3390/rs11070786

Ciocarlan, A., & Stoian, A. (2021, October). Ship detection in sentinel 2 multi-spectral images with self-supervised learning. *Remote Sensing*, *13*(21), 4255. Retrieved from https://doi.org/10.3390/rs13214255 doi: 10.3390/rs13214255

Dhanda, A., Remondino, F., & Quintero, M. S. (2018, May). A METADATA BASED APPROACH FOR ANALYZING UAV DATASETS FOR PHOTOGRAMMETRIC APPLICATIONS. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XLII-2*, 297–302. Retrieved from https://doi.org/10.5194/isprs-archives-xlii-2-297-2018 doi: 10.5194/isprs-archives-xlii-2-297-2018

Groves, P. (2013). *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems, Second Edition*.

Hartley, R., & Zisserman, A. (2003). *Multiple View Geometry in Computer Vision*. Cambridge University Press.

Hassanalian, M., & Abdelkefi, A. (2017, May). Classifications, applications, and design challenges of drones: A review. *Progress in Aerospace Sciences*, *91*, 99–131. Retrieved from https://doi.org/10.1016/j.paerosci.2017.04.003 doi: 10.1016/j.paerosci.2017.04.003

He, K., Gkioxari, G., Dollar, P., & Girshick, R. (2017, October). Mask R-CNN. In *2017 IEEE international conference on computer vision (ICCV)*. IEEE. Retrieved from https://doi.org/10.1109/iccv.2017.322 doi: 10.1109/iccv.2017.322

Heikkila, J., & Silven, O. (1997). A four-step camera calibration procedure with implicit image correction. In *Proceedings of IEEE computer society conference on computer vision and pattern recognition*. IEEE Comput. Soc. Retrieved from https://doi.org/10.1109/cvpr.1997.609468 doi: 10.1109/cvpr.1997.609468

Htet, K. S., & Sein, M. M. (2021, April). Toddy palm trees classification and counting using drone video: Retuning hyperparameter mask-RCNN. In *2021 7th international conference on control, automation and robotics (ICCAR)*. IEEE. Retrieved from https://doi.org/10.1109/iccar52225.2021.9463466 doi: 10.1109/iccar52225.2021.9463466

Joyce, K. E., Duce, S., Leahy, S. M., Leon, J., & Maier, S. W. (2019). Principles and practice of acquiring drone-based image data in marine environments. *Marine and Freshwater Research*, *70*(7), 952. Retrieved from https://doi.org/10.1071/mf17380 doi: 10.1071/mf17380

Keller, A., & Ben-Moshe, B. (2022, April). A robust and accurate landing methodology for drones on moving targets. *Drones*, *6*(4), 98. Retrieved from https://doi.org/10.3390/drones6040098 doi: 10.3390/drones6040098

LeCun, Y., Bengio, Y., & Hinton, G. (2015, May). Deep learning. *Nature*, *521*(7553), 436–444. Retrieved from https://doi.org/10.1038/nature14539 doi: 10.1038/nature14539

Leeuwen, S., Traag, W., Hoogenboom, L., & De Boer, J. (2002). Dioxins, furans and dioxin-like PCBs in wild, farmed, imported and smoked eel from the Netherlands. *Organohalogen compounds*, *57*, 217–220.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollar, P. (2017, October). Focal loss for dense object detection. In *2017 IEEE international conference on computer vision (ICCV)*. IEEE. Retrieved from https://doi.org/10.1109/iccv.2017.324 doi: 10.1109/iccv.2017.324

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). SSD: Single shot MultiBox detector. In *Computer vision – ECCV 2016* (pp. 21–37). Springer International Publishing. Retrieved from https://doi.org/10.1007/978-3-319-46448-0_2 doi: 10.1007/978-3-319-46448-0_2

Mlambo, R., Woodhouse, I., Gerard, F., & Anderson, K. (2017, March). Structure from motion (SfM) photogrammetry with drone data: A low cost method for monitoring greenhouse gas emissions from forests in developing countries. *Forests*, *8*(3), 68. Retrieved from https://doi.org/10.3390/f8030068 doi: 10.3390/f8030068

Mount, R. (2005, December). Acquisition of through-water aerial survey images. *Photogrammetric Engineering and Remote Sensing*, *71*(12), 1407–1415. Retrieved from https://doi.org/10.14358/pers.71.12.1407 doi: 10.14358/pers.71.12.1407

Nan, L. (2021a). *Camera models.* https://3d.bk.tudelft.nl/courses/geo1016backup/handouts/01-camera_models.pdf.

Nan, L. (2021b). *Reconstruct 3D geometry.* https://3d.bk.tudelft.nl/courses/geo1016backup/handouts/04-reconstruct_3D_geometry.pdf.

Nan, L. (2022a). *Introduction to machine learning.* https://3d.bk.tudelft.nl/courses/geo5017/handouts/01-Introduction.pdf.

Nan, L. (2022b). *Performance metrics for classification.* https://3d.bk.tudelft.nl/courses/geo5017/handouts/09-ClassificationMetrics.pdf.

O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Padilla, R., Netto, S. L., & da Silva, E. A. B. (2020, July). A survey on performance metrics for object-detection algorithms. In *2020 international conference on systems, signals and image processing (IWSSIP).* IEEE. Retrieved from https://doi.org/10.1109/iwssip48289.2020.9145130 doi: 10.1109/iwssip48289.2020.9145130

Patterson, J., & Gibson, A. (2017). *Deep learning: A practitioner's approach.* O'Reilly Media, Inc.

Pedersen, S. A., Fock, H., Krause, J., Pusch, C., Sell, A. L., Böttcher, U., ... Rice, J. C. (2008, December). Natura 2000 sites and fisheries in german offshore waters. *ICES Journal of Marine Science*, *66*(1), 155–169. Retrieved from https://doi.org/10.1093/icesjms/fsn193 doi: 10.1093/icesjms/fsn193

Prayudi, A., Sulistijono, I. A., Risnumawan, A., & Darojah, Z. (2020, September). Surveillance system for illegal fishing prevention on UAV imagery using computer vision. In *2020 international electronics symposium (IES).* IEEE. Retrieved from https://doi.org/10.1109/ies50839.2020.9231539 doi: 10.1109/ies50839.2020.9231539

Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. *arXiv preprint arXiv:1804.02767*. Retrieved from https://arxiv.org/abs/1804.02767 doi: 10.48550/ARXIV.1804.02767

Ren, S., He, K., Girshick, R., & Sun, J. (2017, June). Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(6), 1137–1149. Retrieved from https://doi.org/10.1109/tpami.2016.2577031 doi: 10.1109/tpami.2016.2577031

Salz, P., Hoefnagel, E., Bavinck, M., Hoex, L., Bokhorst, J., Blok, E., & Quaedvlieg, J. (2008). *Maatschappelijke gevolgen van de achteruitgang in de visserij* (Tech. Rep.). LEI. Retrieved from https://edepot.wur.nl/44406

Schenk, T. (2005). Introduction to photogrammetry. *The Ohio State University, Columbus*.

Snyder, J. P. (1987). *Map projections–a working manual* (Vol. 1395). US Government Printing Office.

Steiner, L. (2011). Reconstruction of glacier states from geo-referenced, historical postcards. *Master's diss. Eidgenossische Technische Hochschule Zurich*.

Tiberius, C., van der Marel, H., Reudink, R., & van Leijen, F. (2021). *Surveying and mapping*. Netherlands: TU Delft Open. doi: 10.5074/T.2021.007

van Oostenbrugge, J., Bartelings, H., & Buisman, F. (2010). *Distribution maps for the north sea fisheries; methods and application in natura 2000 areas*. LEI, part of Wageningen UR.

Voinov, S. (2020). *Deep learning-based vessel detection from very high and medium resolution optical satellite images as component of maritime surveillance systems* (Unpublished doctoral dissertation). Universität Rostock.

Xu, G., & Zhang, Z. (1996). *Epipolar geometry in stereo, motion and object recognition: a unified approach*. Springer Science & Business Media.

Yang, Z., Yu, X., Dedman, S., Rosso, M., Zhu, J., Yang, J., ... Wang, J. (2022, September). UAV remote sensing applications in marine monitoring: Knowledge visualization and review. *Science of The Total Environment*, *838*, 155939. Retrieved from `https://doi.org/10.1016/j.scitotenv.2022.155939` doi: 10.1016/j.scitotenv.2022.155939

Zhang, H., Wang, G., Lei, Z., & Hwang, J.-N. (2019, October). Eye in the sky. In *Proceedings of the 27th ACM international conference on multimedia.* ACM. Retrieved from `https://doi.org/10.1145/3343031.3350933` doi: 10.1145/3343031.3350933

Zhao, X., Pu, F., Wang, Z., Chen, H., & Xu, Z. (2019). Detection, tracking, and geolocation of moving vehicle from UAV using monocular camera. *IEEE Access*, *7*, 101160–101170. Retrieved from `https://doi.org/10.1109/access.2019.2929760` doi: 10.1109/access.2019.2929760

Colophon

This document was typeset using LaTeX, using the KOMA-Script class `scrbook`. The main font is Palatino.