Autonomous Path Planning and Landing Site Selection for Aquatic UAV

A dissertation submitted in partial fulfilment of the requirements for the degree of Master of Science.

by
Pieter Fiers

Master's Degree in Advanced Computer Science

Cardiff School of Technology

Cardiff Metropolitan University, Cardiff

August 2022

Abstract

Drones capture the interest of science and industry in countless domains. These autonomous flying robots can be deployed to observe wildlife, take samples, or monitor the environment from up high. Missions like these would benefit from longer flight durations: to take more measurements or to observe their subject for longer. Solar cells on the drone can help with this. However, they cannot extend mission lengths indefinitely. To do so, the drone would still need to land now and then to recharge its batteries. The high-level planning software necessary for this has not yet been discussed in the literature.

This dissertation addresses this by developing a system that can autonomously select suitable landing sites to recharge on multi-hop routes. To achieve this, a semantic segmentation algorithm was trained to identify lakes and buildings in aerial imagery. The output from this network is used by a newly developed optimized visibility graph algorithm. With the constructed graph, a novel path planner is able to generate multi-hop shortest routes to any location, with recharge stops on lakes when necessary.

The semantic segmentation algorithm was selected as the best of nine potential networks, and achieved an 96% accuracy. Furthermore, the optimized visibility graph algorithm is able to process large maps more than two times faster than the standard method, while the planner algorithm can always generate a near-optimum path. This complete high-level planning system can create a map from just aerial imagery and use it to generate multi-hop paths with recharge stops. When deployed on an aquatic UAV with solar cells, this platform will be capable of extending research missions from just hours to multiple days.

Word count: 9724
Version: 22-8-8

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This opening chapter discusses the background behind this dissertation. It also presents a problem description that will guide the later *Literature Review* and *Design & Implementation* chapters (chapters 2 & 3).

1.1 Background

Small unoccupied aerial vehicles (UAVs, colloquially 'drones'[1]) have myriad applications in biological and environmental research. From remote sensing applications like counting cattle, kangaroos, or penguins[2] and preventing poaching[3], to listening to bats[4] or checking surface water quality[5]. They can even work together; as communication relays[6] or in a swarm[7].

All of these applications would benefit from longer flight durations. One way to achieve this is with solar/photovoltaic (PV) cells on the UAV. These cells can recharge the drone's batteries, both in flight and when landed. Combined with an efficient design, a drone like this can even achieve multi-day flight (Oettershagen et al. 2015). However, this requires excellent conditions for an extended period of time. Eventually, the drone would always need to land; whether due to weather conditions or simply because the battery did not last the night.

For multi-day deployments or deployments in remote areas, it might not be feasible to land manually. Instead, this process needs to be autonomous. Such an autonomous solution would allow the drone to land, recharge its batteries, and subsequently continue its mission (illustrated in figure 1.1 below).

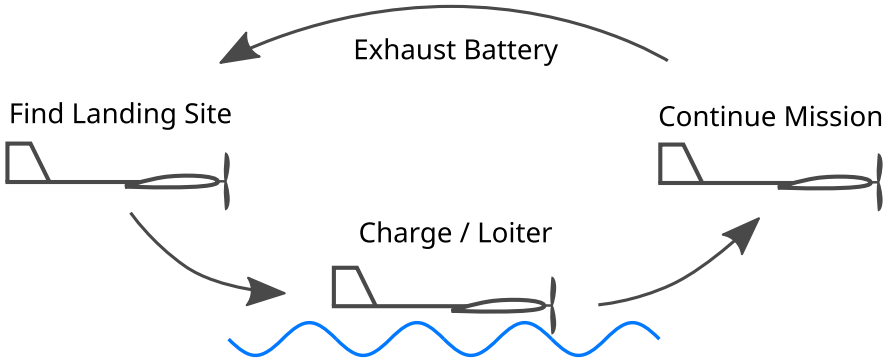

Figure 1.1: Mission stages of the proposed system

Even a perfect automatic landing system would still work best on large, open spaces. This way, the system would have the chance to avoid obstacles when landing. One ubiquitous type of open space is bodies of surface water (lakes, ponds, canals...). In many parts of the world, surface waters large enough to

[1]See appendix A: *Drone Terminology* for more information on drone/UAV terminology used in this dissertation.

[2]Cattle: Andrew, Greatwood, and Burghardt 2019; Barbedo et al. 2020; Kellenberger, Volpi, and Tuia 2017. Kangaroos: Brunton, Leon, and Burnett 2020. Penguins: Bird et al. 2020.

[3]Vuuren et al. 2019; Puri and Bondi 2021

[4]Kloepper and Kinniry 2018; Fu, Kinniry, and Kloepper 2018

[5]Sibanda et al. 2021Zang et al. 2012Koparan, Koc, Privette, et al. 2018

[6]Boyang Li et al. 2016; Chen, Feng, and Zheng 2018; Pinkney, Hampel, and DiPierro 1996

[7]L. He et al. 2018; Campion, Ranganathan, and Faruque 2019

land a UAV are rarely far apart (illustrated by figure 1.2 below). They are the perfect candidate to serve as an intermediate recharge site.

The hopping from one lake to the next to recharge is the namesake of *Lakehopper*, a work-in-progress UAV and its supporting systems (e.g., ground station software, communications). Lakehopper is meant to serve as a long-duration mission platform for research like mentioned above.

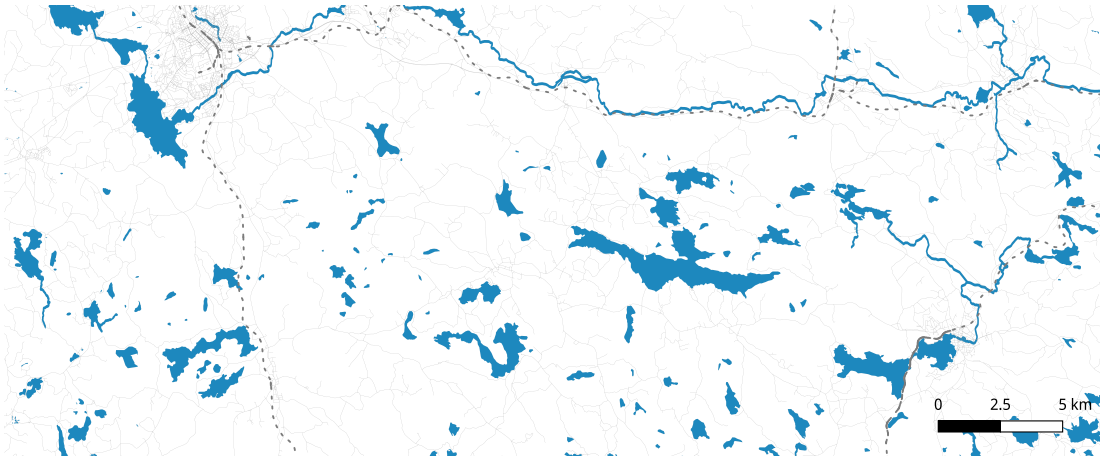

Figure 1.2: Map of surface water bodies around Vetlanda, Sweden
Created using QGis. Map data © OpenStreetMap contributors.

1.2 Problem Description

One of the biggest puzzle pieces in Lakehopper's design is the software system responsible for high-level planning. This system needs to decide where to land and how to get there. Solving this planning problem is the primary goal of this dissertation.

1.2.1 Landing Site Selection

When Lakehopper's battery is low and it needs to land, it has to select an appropriate body of surface water to land. For this, the system needs a map with all the landable waters. This map could be constructed from existing landuse and topographic maps[8]. However, maps like these do not consider overhanging vegetation or floating structures. Figure 1.3 shows two examples of this. Additionally, surface waters are influenced by weather and seasonal changes, so ideally the map would need several variations.

These issues can be addressed by instead using a computer vision algorithm to identify surface water from aerial imagery. This way, only the lakes and rivers where landing would be possible are included. With this technique, seasons can

[8]Candidates include OpenStreetMap (osm.org/about) and the European Commission's *Global Surface Water* project (publications.jrc.ec.europa.eu/repository/handle/JRC109054, Pekel et al. (2016))

Figure 1.3: Water obscured by vegetation (left) and a bridge (right)

also be taken into account because aerial imagery is available for various times throughout the year.

Besides as a way to generate a map of landing sites beforehand, this algorithm could also be used in-flight. Using a camera on board, it would allow the UAV to verify the state of a site before landing and to check for obstacles like boats or debris.

Because this dissertation is only concerned with the high-level planning software, this second use case does not fall into scope. The first, however, does and forms one of two components that will be implemented.

The first of these components is the 'planner'. The planner includes the algorithm to select a landing site from the map of surface waters. The second is the 'vision' component. It consists of the computer vision algorithm to identify surface water from aerial imagery, as well as the functionality to convert detections to the map.

1.2.2 Path Planning

After an appropriate surface water is found, the UAV would need to plan a path to the landing site.

A direct path might cross over areas where UAV flight is forbidden, for example around airports, military sites, or restricted areas. The path should also avoid regions where flight is discouraged, like over built-up districts or sensitive habitats. These areas can be recorded in the same map that the UAV uses to select a landing site. This is illustrated in figure 1.4.

Though out of scope of this dissertation, an even more dynamic path planner could also take into account the position of other nearby aircraft, and avoid them. Positional data of other aircraft can be collected through receivers on the UAV itself or through a connection to online services.

Autonomously planning a path while avoiding airspace restrictions and built-up areas is part of the planner component.

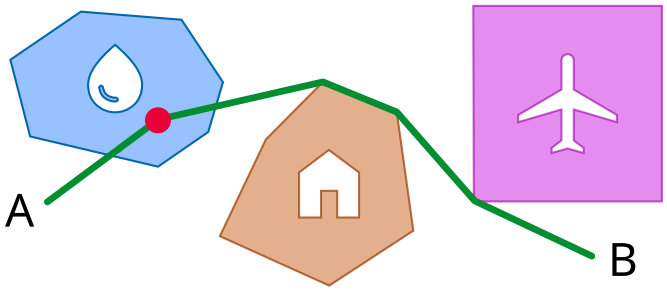

Figure 1.4: Shortest flight path among obstacles, landing on water to recharge along the route
Blue: water Brown: buildings Purple: restricted airspace

1.3 Summary of Introduction

This chapter has been an introduction to this dissertations' background and to the problem it aims to solve. The latter being: how can an autonomous UAV select an appropriate landing site to recharge and plan a path to it?

This dissertation aims to solve this problem by detecting bodies of water from aerial imagery, selecting a landing site from a map of this data, and finally planning the path based on this map. This functionality will be split into two components: the vision component and the planner.

Later chapters will discuss the design, implementation, and evaluation of these components.

Chapter 2

Literature Review

This chapter will discuss the existing literature surrounding UAVs, semantic segmentation of aerial imagery, landing site selection, path planning, and possible research applications. This review will be the foundation for the later chapter 3: *Design & Implementation.*

2.1 UAV Design

As discussed before, UAVs have opened up many new possibilities in research. To provide background for, and investigate the viability of Lakehopper, existing designs and applications will be discussed. Particular attention will be given to aquatic drones and photovoltaics.

2.1.1 In General

There are two main types of UAVs: rotary-wing and fixed-wing. Rotary-wing drones fly using direct upward thrust, conventionally generated by propellers (i.e., rotors). Fixed-wing drones instead use a power system to produce forward thrust. This forward thrust is partially converted into vertical lift using wings. Figure 2.1 below shows examples of both types.

In general, fixed-wing drones are more efficient for long-distance flight, as well as for larger payloads (Paredes et al. 2017). As Dong et al. (2019) and Polonelli et al. (2020) have shown, this makes fixed-wing drones especially suitable for remote sensing applications like those discussed in 1.1: *Background*.

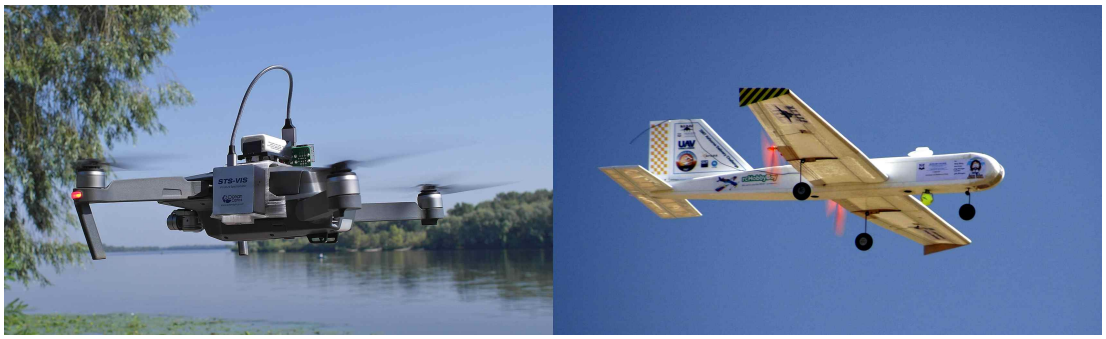

Figure 2.1: Two UAV types: rotary (quadcopter, left) and fixed-wing (plane, right)

When equipped with the appropriate sensors (like GPS/GNSS receivers), drones can follow a provided path entirely autonomously[1] (Hadi et al. 2014). This is made possible by flight controller software like PX4[2] and ArduPilot[3], which will no doubt be an important part of the Lakehopper prototype.

[1] **Note:** 'Autonomous flight' in this section refers to missions high above surrounding obstacles. '3D' autonomous flight among obstacles (usually based on computer vision and/or 'simultaneous localization and mapping' [SLAM]) is a broad and interesting area of research, but not directly applicable to this dissertation. For examples, see any of Xin Zhou et al. (2022), Lu et al.'s (2018), Smolyanskiy et al.'s (2017), or Hadi et al.'s (2014) work.

[2] https://px4.io

[3] https://ardupilot.org

2.1.2 Photovoltaics

Photovoltaics/solar cells have become efficient and cheap enough to be deployed on research drones. Oettershagen et al.'s (2015) 5.65m-wide experimental airplane for example was capable of a continuous 12h 22m flight, powered by 88 solar modules mounted on its wings. More recently, Dwivedi et al. (2018) achieved an even longer 18 hour flight time, with a slightly smaller 5.35m wingspan, clearly demonstrating the ever-increasing efficiency of PV cells.

Deployments like these are also possible on a much smaller scale, like Chu et al.'s (2021) 2m-wide retrofitted commercial airplane. In this case, just 12 solar cells were already enough to achieve a 22% efficiency gain. The tests for this example were also not conducted under the excellent conditions needed to achieve the results of Oettershagen et al. or Dwivedi et al.

If used as a platform for other missions, photovoltaics would also just be a nice-to-have, as opposed to the main focus. For Lakehopper's applicable use cases and for the foreseeable future, results like those of Chu et al. will likely remain the status quo. That is: long duration, but never indefinite flight.

2.1.3 Aquatic UAVs

Lakehopper's entire premise – loitering on water to recharge – of course banks on the assumption that a drone can be made to land on, and take off from, water. This is fortunately the case, and incidently the subject of much research.

Water-landing/take-off capable drones can be divided into four categories: multicopters with floats, fixed-wing planes with floats (i.e., seaplanes), vertical take-off and landing planes (VTOLs), and active/dynamic vertical take-off planes.

Multicopters with floats are the subject of the vast majority of research on rotary-wing UAVs. As discussed, these drones use vertical thrust to stay in the air, and those that are aquatic usually use bottom-mounted floats for landings. Examples of these are the quadcopter described by Agarwal and M. K. Singh (2019) or the hexcopter by Koparan, Koc, Privette, et al. (2020). Both of these examples were used – among other purposes – to conduct water quality measurements. This use-case will be discussed further in section 2.5: *Applications*.

VTOLs, or 'vertical take-off and landing' aircraft, are a second category of aquatic multicopters. These are in a sense *both* rotary-wing *and* fixed-wing. An example of a VTOL aircraft possessing both rotors and fixed wings is that of Bustamante et al. (2019), which uses ducted-fans embedded in the plane's wings. Alternatively, a transforming design can be used, like in M. Hsu's (2020) thesis. This thesis used a quadcopter structure capable of rotating from a horizontal to a vertical orientation, rotating with it the direction of thrust from pointing upwards to pointing forwards.

Active vertical take-off planes use vertical thrust from a single propeller to lift themselves (or 'shoot') out of the water. Once in the air, they transition to a normal flight mode and use the same propeller to provide horizontal thrust.

This is most commonly achieved using a rotating propeller, like in Tetreault, Rancourt, and Lussier Desbiens (2020) or Waldau (2019). Alternatively, the plane can use a fixed propeller but start its take-off from under the water to achieve the necessary angle with the surface on exit, as shown by Rockenbauer et al. (2021). Interestingly, Tetreault, Rancourt, and Lussier Desbiens's design also includes solar panels on the plane's wings, with the same goal as previously described: recharging between flights. Similarly, Waldau's design would loiter as for example a communications relay.

Seaplane UAVs are the fixed-wing equivalent of multicopters with floats, using floats under their wings to land and take off. They are the scaled down version of how most full-sized airplanes achieve aquatic capabilities. This concept is demonstrated by Lou et al.'s (2019) preliminary design.

2.1.4 Summary of UAV Design

The Lakehopper prototype can be designed to have enough solar cells to recharge its batteries when landed and to fly for longer than would be possible with just a battery. Take-off and landing from water would also be possible, maybe through an active vertical take-off mechanism or acting as a VTOL.

2.2 Semantic Segmentation of Aerial Imagery

Semantic segmentation is the process of dividing an image into different parts based on the features present (see figure 2.2 below for an example). In aerial imagery these features (also 'classes') can be buildings, roads, agricultural fields, or of particular interest for this system: bodies of surface water (M. Wu et al. 2019; Wurm et al. 2019). If a particular pixel belongs to a road for example, a semantic segmentation algorithm might pick up on the uniform texture of the surrounding area and the dark colour when compared to the rest of the image. For each pixel and class, the algorithm then outputs a confidence level indicating how likely it is that said pixels belongs to that class. Training a semantic segmentation model requires examples of pixel-class pairings (labels) and is thus a type of supervised machine learning (Y. Guo et al. 2018).

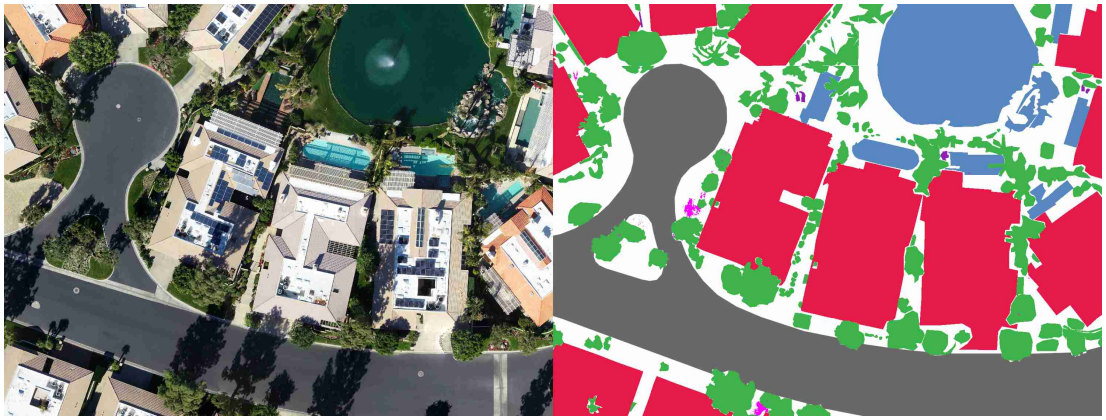

Figure 2.2: Semantic segmentation of aerial imagery
Left: aerial imagery Right: semantically segmented mask

The most common semantic segmentation algorithms either follow a traditional machine learning approach or are an encoder-decoder convolutional neural network (CNN). As shown in reviews by W. Liu et al. (2017), and Aloysius and Geetha (2017), models in the CNN category have seen a particular focus in recent years. Though, there is no single 'best' approach for all use cases[4].

Nonetheless, some broad distinctions can be made to guide one's choice. For example, in general, traditional machine learning approaches require feature engineering. This extra step involves extracting properties like edges, contrast changes, pixel entropy, etc. from the image before passing it to the model (as done for the random forest model of Kang and Nguyen (2019) and the models of Bhatnagar, Gill, and Ghosh (2020)).

CNNs on the other hand nearly always operate directly on the input image. This difference means CNNs are favourable in cases where the input dataset is too varied and unstructured for feature engineering to be effective (Mahony et al. 2020). The strength of being able to work with large datasets is however a double-edged

[4]This is also known as the 'no free lunch' theorem (Wolpert and Macready 1997).

sword, as it is not always possible to gather the required amount of samples. So, for small datasets, a traditional approach might perform better.

In the case of the vision component of Lakehopper, a general and adaptable algorithm is required. This is because it might need to be expanded with new feature classes (e.g., power lines or roads) or because it should work in new environments (e.g., snowy or arid). Additionally, aerial imagery datasets are widely available, so creating a large enough dataset should not be a problem.

So while a traditional machine learning approach might be preferable for a system with a limited scope and expansion possibilities, for Lakehopper, a semantic segmentation CNN is likely the best option. Ideally both approaches would be evaluated and compared, but in the interest of limiting the scope of this dissertation, only a handful of CNN variants will be considered.

2.2.1 Convolutional Neural Networks

The defining feature of convolutional neural networks is that they are made up of 'convolutional' layers. A convolutional layer uses sliding windows (also 'kernels') to "recognize features regardless of their position in the image" (Albawi, Mohammed, and Al-Zawi 2017, p. 1). Such a window might, for example recognize a downward gradient or a horizontal line. The output of a single convolutional layer is a matrix indicating the likelihood that the feature, the kernel is tuned to look for is present in a particular part of the image.

A complete CNN combines many of these layers (along with other operations) one after the other. Training this network is an automatic process – including a step called 'backpropagation' – that, for each image in the dataset, tunes the kernels so that the network output best matches the true label. (XueFei Zhou 2018).

2.2.2 CNN Encoders

One often-used configuration for a CNN is as a so called 'encoder', made up of a series of convolutional layers combined with layers that reduce the resolution of the image. This reduction (i.e., 'downsampling') is usually achieved using max pooling, which only keeps the most important feature in a certain window.

The output of an encoder is a dense representation of the input. While low in resolution (sometimes just 4 pixels wide and high), this representation describes many features (represented as channels/dimensions). These features are not designer-architected, but they do typically indicate some recognizable trait, like how 'wavy' the area is or how vegetation or water-like the colours are.

The dense representation can be used as the input to a classifier, which outputs a probability array indicating the likelihood that a particular class is present anywhere in the image. The first item in this array might, for example, indicate that there is a 88% chance of a building being in the image.

In the case of CNNs for semantic segmentation, the dense representation is used

as the input for a second stage: the decoder (Ji et al. 2021). This stage expands the dense representation back to the resolution of the input image, forming a sort of reverse funnel which mirrors the encoder stage (see later figure 2.3). The result is a collection of probability matrices indicating the likelihood that a certain pixel belongs to its matrix' corresponding class. Colouring each pixel according to which matrix has the highest probability results in a mask like that of figure 2.2-right (page 10).

The exact arrangement of layers in the encoder varies for every CNN. Some of the most used architectures are VGG (Simonyan and Zisserman 2015), ResNet (K. He et al. 2015), InceptionNet/InceptionResNet (Szegedy et al. 2016), MobileNet (Howard et al. 2017), and EfficientNet (Tan and Le 2020).

Bhatnagar, Gill, and Ghosh (2020) compare the performance of the first two of these for semantic segmentation of drone imagery of a bog in Ireland. They find that overall, ResNet50 performs better for their dataset, but note that VGG16 can more effectively handle noise in the images.

In another example, Girisha et al. (2019) use a VGG16 architecture on imagery of suburban roads and surrounding vegetation in India. They later expand on this work with a ResNet50-based model on an extended version of the dataset, finding that the variants tested achieved a similar performance (Girisha et al. 2021).

Three of the most recent encoders: MobileNet, InceptionResNetV2, and EfficientNet, were evaluated by Parmar et al. (2020) on the *DroneDeploy* dataset (this dataset will later be considered in section 2.2.5: *Datasets*). Of these three encoders, the latter two outperformed MobileNet.

Besides segmentation performance, encoders should also be evaluated by their complexity and computational cost, as done by Bahl et al. (2019) for power consumption.

Overall, a case-by-case analysis is required to decide on the best architecture for a particular system. For the semantic segmentation CNN of Lakehopper, this analysis will include ResNet50, InceptionResNetV2, and EfficientNet. All three of these show promise based on the above results.

2.2.3 Semantic Segmentation Decoders

As mentioned, CNNs for semantic segmentation have not only an encoder, but also a decoder stage. The design of this stage, as well as the connection between the two stages varies significantly between models. Some of the most used architectures in this regard include UNet (Ronneberger, Fischer, and Brox 2015), Fully Convolutional Networks (FCNs; Shelhamer, Long, and Darrell 2016), SegNets (Badrinarayanan, Kendall, and Cipolla 2016) Feature Pyramid Networks (FPNs; Lin et al. 2017), Pyramid Scene Parsing Networks (PSPNets; Zhao et al. 2017), and LinkNets (Chaurasia and Culurciello 2017).

All of the above architectures use some form of 'skipping' between layers of the encoder and decoder. The goal of skips is to include not only the coarse features of

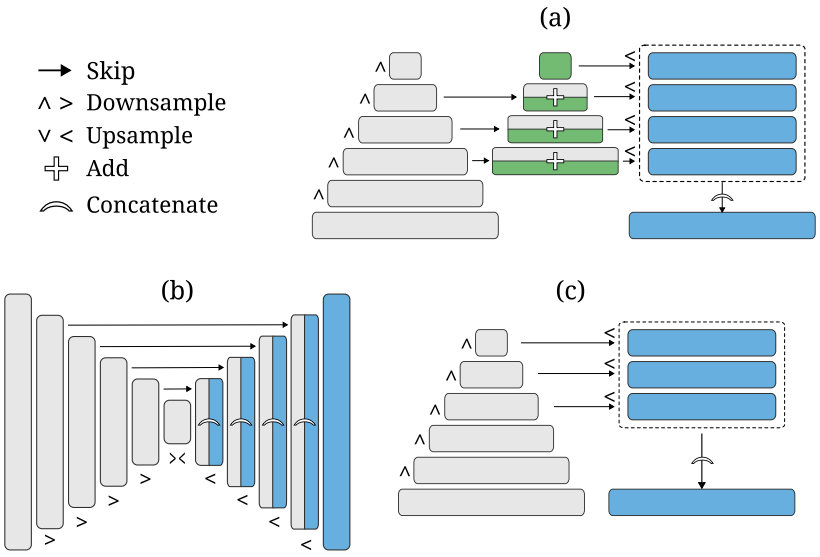

Figure 2.3: Semantic segmentation decoder architectures
(a): FPN (b): UNet (c): FCN Grey/green/blue layers: downsample/intermediate/upsample layers

the dense representation but also the finer edges of segmented objects. The final layer of the encoder might for example point to a water feature being present in the center of the image, while the second to last (higher resolution) layer indicates the oval shape of that feature. These two data points are then combined in the decoder to form an oval lake. Exactly how this is achieved varies between the models. LinkNets for example arithmetically add the skipped layer to the previous layer in the decoder, while UNets and FCNs instead concatenate the layers and subsequently extract the most important features.

A second common aspect of decoders beside skips is the presence of upsampling layer(s) to bring the final dense layer of the encoder up to the same resolution as the input image. In UNets and LinkNets this is done in steps. FCNs on the other hand use a single layer to handle this task, while PSPNets and FPNs use an intermediate pyramid-shaped representation feeding into a single final layer.

Figure 2.3 above illustrates the layout of skips and upsampling layers visually for the FPN, UNet, and FCN architectures.

The previously discussed analyses by Bhatnagar, Gill, and Ghosh (2020, bog Ireland) and Girisha et al. (2019, road/vegitation India) also compared UNet to respectively SegNet and FCN. For their particular use cases, they found that an FCN outperformed UNet, followed by SegNet.

UNet was also compared to the three most recent architectures mentioned (FPNs, PSPNets, and LinkNets) by Parmar et al. (2020, DroneDeploy dataset). This study found FPN outperforms the three other architectures, but was closely followed by UNet.

Based on these results, UNet, FCN and FPN will be considered in later sections.

2.2.4 Transfer Learning

As discussed before, CNNs often require large training datasets to reach an acceptable level of performance. One way to reduce this need is to use transfer learning. In transfer learning, another – often unrelated – dataset is used to initially train the model. This primes the model to recognize general shapes and features. It can then be trained a second time with the actual dataset for fine-tuning. The initial dataset can be a much more general one, like *Imagenet*[5], which contains classes completely unrelated to drone semantic segmentation (e.g., 'leopard' or 'artichoke').

The concept of transfer learning is illustrated by Wurm et al. (2019) for the detection of slums. In this study, the CNN is initially trained with a detailed low-altitude dataset and afterwards fine-tuned to work with larger-scale imagery. Similarly, Girisha et al.'s (2021) segmentation model was first trained on the Cityscapes[6] dataset and subsequently retraining on self-collected drone imagery. Lakehopper's CNN will also use transfer learning, as will be discussed in section 3.2.2: of the next chapter: 3: *Design & Implementation.*

2.2.5 Datasets

Labelling the images required to train a CNN can be a tedious task. However, using an existing labelled dataset can – partially – alleviate this.

A first requirement of such a dataset is that it should contain enough examples of the classes that we want to identify. If not, predictions from the model might get skewed towards other classes (known as class imbalance). Therefore, datasets like the one used by Bhatnagar, Gill, and Ghosh (2020) or Robicquet et al. (2016) cannot be used, as they contain no water.

Additionally, the dataset must contain colour images, as captured by the camera that will be used on the Lakehopper drone. This is as opposed to, for example, infrared pictures, or the polarimetric synthetic-aperture radar (PolSAR) images used in W. Wu et al. (2019).

Finally, the images must be captured from the same perspective as that of Lakehopper's camera (top-down), making the oblique images of Lyu et al. (2020) for example inapplicable.

Given all these requirements, the DroneDeploy segmentation dataset as well as the dataset used by Girisha et al. (2021) can be used to train the CNN, and will later be discussed in detail in section 3.2.1: *Preprocessing.*

[5]https://www.image-net.org/
[6]https://www.cityscapes-dataset.com/

2.3 Landing Site Selection

To select an appropriate landing site from its surroundings, the planner naturally needs a map of all of them (or at least of those close-by). This map can be generated dynamically (i.e., in-flight), by using semantic segmentation for example, as done by X. Guo et al. (2014, aerial imagery) and Marcu et al. (2018, in simulation). Alternatively, it can be created beforehand, as shown by Ayhan et al. (2019) for satellite imagery. This last example closely resembles Lakehopper's approach.

Given a map of sites, the next step to selecting the best one is to filter out any candidates that are too small to land. What 'too small' means of course depends on the drone in question. A seaplane-style UAV would need a runway-shaped rectangle to land, while a multicopter only needs a spot as large as itself (and some margin). For the purposes of this dissertation, Lakehopper will be assumed to be either a multicopter, VTOL, or dynamic takeoff plane, all of which require simply a circular landing site.

For each body of water, the question then boils down to a circle-in-polygon problem: does the minimum landing area (the circle) fit into the landing site (the polygon). A simple solution that comes to mind is to find the furthest removed point from all sides of the polygon, known as the 'pole of inaccessibility'. If that point is further removed from the nearest side than the circle's radius, the circle fits. The pole of inaccessibility can be found using for example the *Polylabel* algorithm, developed by *MapBox*[7], a web cartography company (Agafonkin 2016).

A problem with this approach is that it would reduce large lakes down to a single landable point. Instead, we ideally want a map of polygons where landing is possible, with the necessary landing circle already accounted for. Creating these polygons is simply a matter of shrinking each body of water by the radius of the landing circle (i.e., negatively buffering it). This simple approach will be the method of choice for Lakehopper, as will be discussed in section 3.1. Along with the point of inaccessibility approach, it is also illustrated in figure 2.4 below.

Figure 2.4: Two approaches to the circle-in-polygon problem
Left: pole of inaccessibility Right: negative buffer

Once a list of landable polygons has been drawn up, which of these counts as the best one depends on the flight path the planner wants to take. This will be discussed in the next section.

[7]MapBox: mapbox.com

2.4 Path Planning

To be used as a platform for long duration and possibly remote missions, Lakehopper must be able to fly completely autonomously between points of interest. As discussed before, the planned path should avoid obstacles like buildings and restricted airspace. Additionally, the planner must be able to decide if an intermediate stop is necessary to recharge the battery.

A complete model of this problem would require navigation in 3-dimensional space, to avoid power lines for example, mountains, and elevation-based airspace restrictions. However, a simpler model in two dimensions ('in the plane') can be used in an initial analysis. In this model, height-dependent obstacles have to be reduced to simple regions based on whether they apply to the UAV's cruising altitude.

Given this representation of the environment, one method of finding the shortest path is through a two-phase algorithm. An initial 'learning' phase constructs a graph representing all possible paths through the environment, after which a second 'query' phase finds the shortest path over this graph (Yang et al. 2014).

The graph can be constructed using randomly placed points, with an edge between every two points that does not intersect any obstacles. This method, known as 'probabilistic roadmap' (PRM), has been studied extensively, for example by Bohlin and Kavraki (2000), Belghith et al. (2006), and Francis et al. (2020). A problem with this approach is that it has trouble navigating through narrow passages, due to the random placement of points, although this can be mitigated by placing points near obstacles (Boor, Overmars, and van der Stappen 1999; Amato et al. 1998) or by dynamically increasing the number of points near these passages (D. Hsu, Jiang, et al. 2003; D. Hsu, Sanchez-Ante, and Sun 2005).

An alternative to PRMs is to simply use the obstacle points as the vertices for the graph, while similarly keeping only edges that are visible to each other. The resulting graph is known as a 'visibility graph', and is demonstrated by Emo Welzl

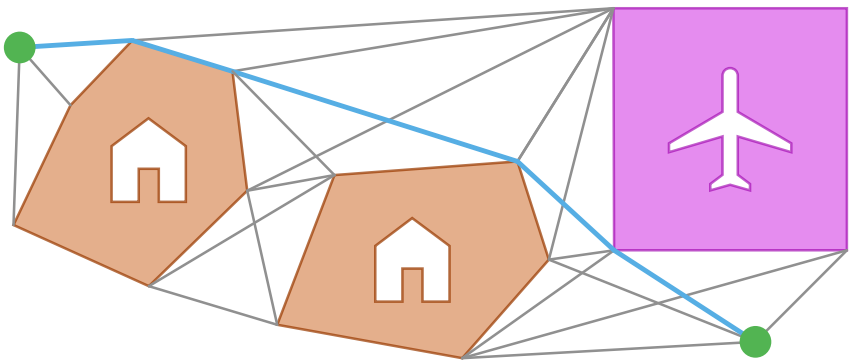

Figure 2.5: Visibility graph among three obstacles, and a shortest path over it
Orange & purple: obstacles Grey lines: visibility graph edges
Blue line: shortest path between green points

(1985), Alt and E. Welzl (1988), and de Berg et al. (2008). Figure 2.5 on the previous page illustrates such a graph and a shortest path over it. Both in the case of PRMs as for visibility graphs, points can be added or removed later, to add a start and end point, or to model dynamic obstacles (Ghambari et al. 2020).

Once the graph is constructed, the query phase can be implemented using Dijkstra's well-known shortest path algorithm (Dijkstra 1959). However, since nodes in the navigation graph correspond to geographical points, the geographical distance to the goal node can be used as a heuristic to guide the search. This modified version of the shortest path algorithm is known as A*, conceived at the *Stanford Research Institute* for the *Shakey* autonomous robot (Hart, Nilsson, and Raphael 1968).

In case the obstacles are not polygonal or too detailed, a grid-based approach is possible (voxel-based in three dimensions, see Yan, Y.-S. Liu, and Xiao 2013). This is demonstrated by Radmanesh et al. (2018) for the potential field, Floyd-Warshall and, Multi-Step Look-Ahead Policy algorithms, as well as by Bo Li et al. (2020) for ant-colony optimization (ACO).

For the purposes of Lakehopper, all obstacles are polygonal, so it is possible to use the visibility graph algorithm combined with A* as discussed before. This method always produces the optimal path, but can be slow for larger areas. Optimizations can later be applied if this proves to be a problem. Section 3.1 of the next chapter, *Design & Implementation*, will detail the development of both the visibility graph generator and the shortest path algorithm.

2.5 Applications

As discussed in the *Background* section (section 1.1), there are endless areas of research that could benefit from Lakehopper as a platform. To discuss them all would not be possible. Instead, this section will show a single case study; water sampling, which is a particularly common application of aquatic drones. The case study will illustrate how existing missions can be adapted to work in the constraints of Lakehopper.

2.5.1 Introduction

Projects that employ UAVs for water sampling can be subdivided into two categories: those that collect samples for later analysis and those that measure properties of the water in-situ (i.e., during the mission).

Sample collection for later analysis is usually achieved using either a long tube and pump, or with a mechanism to submerge & fill a vial. Some other missions instead perform in-situ measurements, possibly without taking samples.

2.5.2 Long Tube Sampling Mechanism

A first example of the long tube method is the UAV system developed by Banerjee et al. (2020), which uses six pumps and respective tubes. This system could benefit from Lakehopper as a platform in two ways. First, it could enable sampling in remote regions where the drone cannot be released close to the target body of water. Second, it could enable sampling for longer periods of time.

For the second use case however, the system of Banerjee et al. might need the possibility to collect more than its current capacity of six samples. If not, multi-day sampling as enabled by Lakehopper might not be useful.

A similar example is the UAV system of Ore et al. (2015), which also uses a long tube and supports collecting multiple samples. However, it uses a single pump combined with a rotating mechanism that 'chooses' a vial to deposit the sample into. This method is likely easier to adapt to multi-day missions, as more vials can just be added without much added complexity.

2.5.3 Submerge & Fill Sample Mechanism

The second common collection technique; using vials that are submerged and filled, is demonstrated by Doi et al. (2017) as well as Koparan, Koc, Privette, et al. (2018) (also Koparan, Koc, Privette, et al. 2019). The former two examples both use a single cartridge suspended by a cable. The last example uses a more complicated mechanism, involving a servo motor that opens or closes one of three cartridges suspended as a single package.

In an alternative configuration, the cartridge can be mounted on the bottom of the UAV, like demonstrated by Koparan, Koc, Privette, et al. (2020).

Like the long tube method, long-distance missions of these systems could benefit from Lakehopper as a platform, but multi-day missions would require modifications to collect additional samples.

2.5.4 In-situ Measurements

To collect in-situ measurements, sensors on the drone itself are used. This technique has the benefit that, in contrast to the previous examples, multi-day missions are possible without major modifications.

The water sampling UAV of Koparan and Koc (2016) for example uses such sensors on the drone to measure dissolved oxygen content (DO), temperature, electrical conductivity (EC), and chloride levels of the water. Koparan, Koc, Privette, et al.'s (2020) implementation of this technique also measures pH level and turbidity (beside the bottom-mounted sample collection discussed before).

2.5.5 Summary of Applications

This case study has demonstrated that Lakehopper could be used as a research platform for existing mission types. Although, it also shows that modifications and expansions might be necessary in some cases.

2.6 Summary of Literature Review

This thorough review of the existing literature has provided the foundation needed to develop the high-level planning software for Lakehopper. Additionally, it has proven the viability of Lakehopper, both with respect to its development and as a research platform.

Chapter 3

Design & Implementation

This chapter discusses the techniques and methods used to create the planner and vision components for Lakehopper. The performance of these components, as well as learnings from their creation, will be discussed in chapter 4: *Results*. Possible extensions, as well as the steps required to integrate them into a complete system, will be discussed in section 5.2: *Future work* of chapter 5: *Discussion & Conclusion*.

A Note on Source Code

All source code for both components is available as an online Git repository via github.com/ubipo/lakehopper. For an overview of the structure of this repository, see appendix B: *Source Code Overview*. References to specific files or directories are written as "`path/to/file.xy`", starting from the root of the repository.

3.1 Planner

The planner component (`planner/`) of the Lakehopper system is responsible for generating the optimal flight path between points, while avoiding obstacles and optionally stopping to recharge.

Because the program that makes these calculations will be used continuously in flight, performance is paramount. Not only to optimize the energy consumption of the computer on the drone, but also to ensure that a safe path is always available in time. The program must also be reliable, with as little unhandled failure conditions as possible. After all, the drone might be difficult to retrieve if it gets stuck in a remote area.

The Rust programming language was specifically designed with such requirements in mind. It is a compiled language without a garbage collector, which pays dividends in its runtime performance. Additionally, it is completely memory-safe and has tooling which forces every possible error to be handled (or explicitly ignored) (Perkel 2020; Balasubramanian et al. 2017).

While the performance and reliability requirements could just as well be met using practically any other language, Rust makes focusing on these goals easy. A notable trade-off of using Rust is slightly more resistance while writing code, while programming in more dynamic languages often feels smoother (Ardito et al. 2021). Forcing all cases and possible errors to be considered – something dynamic languages do not usually require – can indeed be tedious.

This program will serve as the brains of the Planner, and will be executed on the drone itself: close to where its decisions are needed. In web development terms, this part of the system would be called the backend. Its source is located under `planner/src/`.

Apart from the backend, the planner also needs a user interface (UI); to visualize the current state of, and to control, the system. This frontend will run on the ground station computer, so, a webbrowser-based UI was developed to support all modern devices. As the code of the frontend does not have the same stringent performance and reliability requirements as the backend, it was written in Typescript, a more dynamic language than Rust. Typescript also integrates well with modern browsers, as it builds on JavaScript, the de facto programming language for the web. The source code of the UI is stored under `planner/ui/` and figure 3.1 below shows it in use.

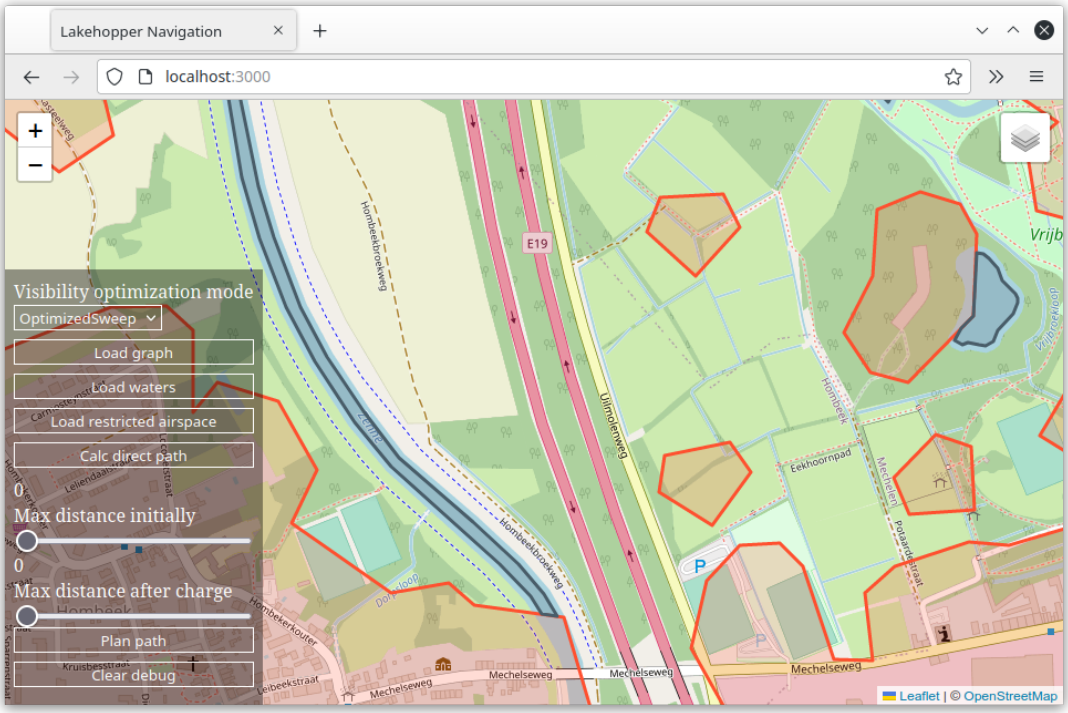

Figure 3.1: Frontend browser UI for the planner

When actually deployed on Lakehopper, there will need to be a continuous communication channel between the UI on the ground station and the backend on the UAV (e.g., for location or battery level updates). For this reason a websocket connection is used, which provides a two-way channel than can easily be adapted to work over a standard drone telemetry connection[1] (Fiers 2021).

The eventual deployment of the navigation system will use data from the imagery segmentation component. For testing during the development phase, however, a temporary dataset was created. This dataset is comprised of building geometries from the Flemish geographical information agency (*AGIV*[2]), combined with surface water body geometries from OpenStreetMap[3]. A safety buffer was added around the buildings, and a negative buffer to the water bodies (as discussed in section 2.3: *Landing Site Selection* of the literature review). Figure 3.1 above shows this dataset loaded in the browser UI via the websocket connection to the navigation backend (red indicating the building buffer, blue the bodies of water).

[1] For example, the SiK telemetry radio by HolyBro: http://www.holybro.com/product/transceiver-telemetry-radio-v3/

[2] *Agentschap voor Geografische Informatie Vlaanderen*: geopunt.be/over-geopunt/extra/glossary/a/agentschap-voor-geografische-informatie-vlaanderen

[3] osm.org/about

3.1.1 Airspace Restrictions

In addition to buildings and bodies of water, the planner's aggregated map also contains airspace restrictions. As an initial proof of concept, this functionality was implemented for Belgium's *UAS geographical zones*. UAS (unmanned aerial systems) geographical zones are a standardized representation of airspace restrictions for UAVs in the European Union[4], as well as – partially – the United Kingdom. Because of this standardization, it would be possible to extend the implementation to other European countries.

The publisher of UAS geographical zones in most of Belgium is *Skeyes*[5], the Belgian air navigation and traffic service provider. Skeyes publishes their geozones – among other channels – through an ArcGIS REST service[6]. To load the geozones, a simple client was developed for this service (`planner/src/droneguide.rs`). This client connects to the relevant endpoints via HTTP and retrieves both the polygons of the geozones, and their restrictions. The restrictions are quite complex, depending on the type of drone, height above sea level, time of day, temporary notices etc.. Therefore, for the purposes of this initial implementation, just the permanent restrictions relevant to Lakehopper's cruising altitude were considered.

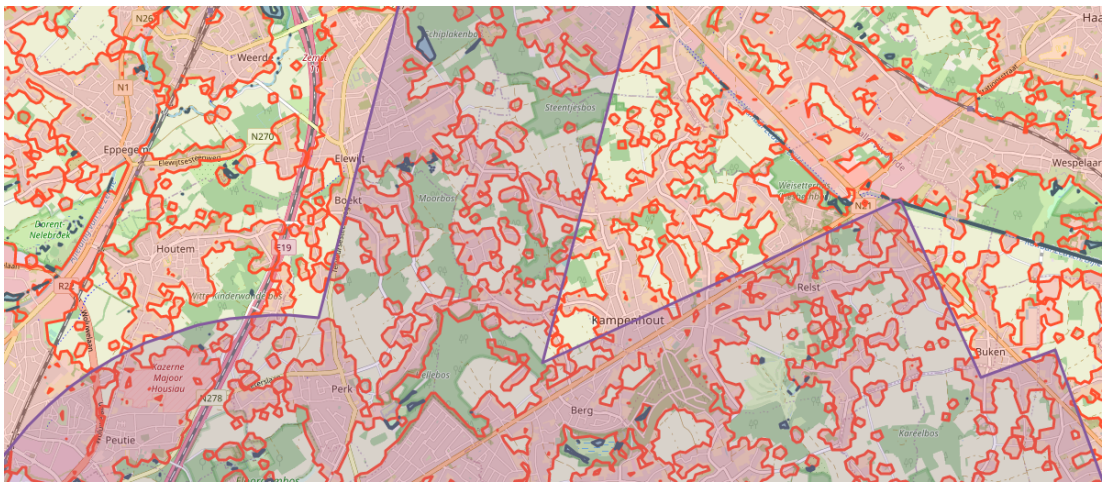

Figure 3.2: Restricted airspace data in the planner
Red: buildings *Blue: water* *Purple: restricted airspace*

An example of the geozone data in the planner UI is shown in figure 3.2 above. Figure 3.1 of the browser UI on page 23 shows the button by which the data can be loaded (labelled "Load restricted airspace").

[4]As defined by article 15 of the Commission Implementing Regulation (EU) 2019/947 of 24 May 2019 on the rules and procedures for the operation of unmanned aircraft (The European Commission 2022).

[5]Skeyes: skeyes.be, the *geozone manager*, as appointed by the Belgian Civil Aviation Authority (BCAA): mobilit.belgium.be/nl/luchtvaart.

[6]Available at services3.arcgis.com/om3vWi08kAyoBbj3. See developers.arcgis.com/rest for more information on ArcGIS REST services.

3.1.2 Visibility Graph

As decided in the literature review section on path planning (section 2.4), a visibility graph approach is used to calculate shortest paths. Examples of a part of this graph are shown in figure 3.3 below. As the calculated graph is extremely dense, this figure only shows the edges for a single node.

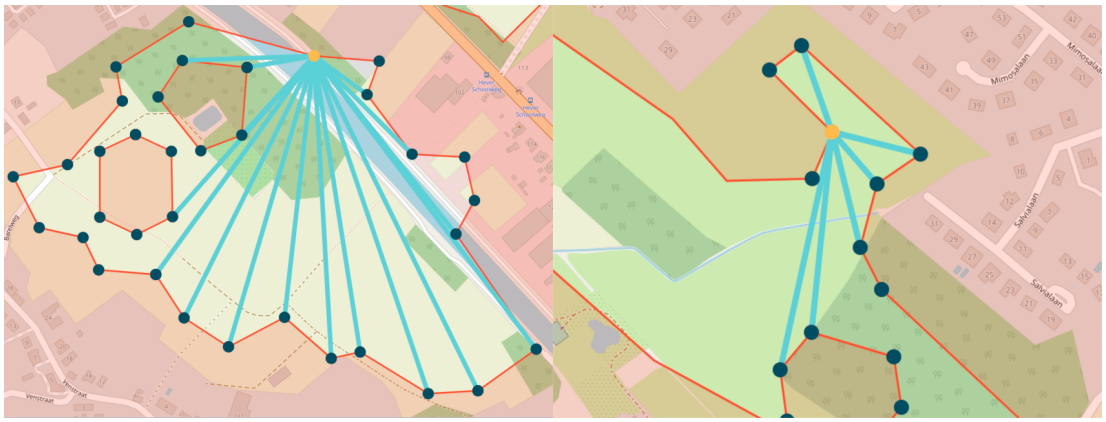

Figure 3.3: Two example nodes of the visibility graph, as well as their edges
Node: orange Graph edges/visibility lines: light blue Possibly visible points: dark blue

A naive approach to calculate this graph would be to check the visibility of every possible combinations of points. Checking whether a point is visible from another point requires checking for an intersection with all line segment on the obstacles, the time complexity of which is $O(n)$ (where n is the number of obstacle points). Performing this check for every possibly visible point gives a complexity of $O(n^2)$, which becomes $O(n^3)$ when applied to every node in the graph.

A better approach, described in de Berg et al.'s *Computational Geometry: Algorithms and Applications* (2008, pp. 326-330), is to use a rotational sweep while checking visibilities. This limits the number of line segments that need to be checked for an intersection to just those intersected by the sweep ray (illustrated in figure 3.4), lowering the intersection checking step's time complexity from $O(n)$ to constant time.

In order to perform this sweep, all points must first be ordered according to their angle with the current point. This step takes at worst $O(n \log(n))$ time, and needs to be done for every node in the graph. Overall, the time complexity of this improved visibility graph algorithm is therefore $O(n^2 \log(n))$.

Further improvements can be made to limit the number of points that need to be considered for visibility. First, only points that are 'in front' of the current node can possibly be visible. A visibility line to any point 'behind' the current node would intersect the obstacle that the current node is a part of. This is visible in the right example of figure 3.3. Notice how the points behind the orange node are not marked with a dot, indicating that they cannot possibly be visible.

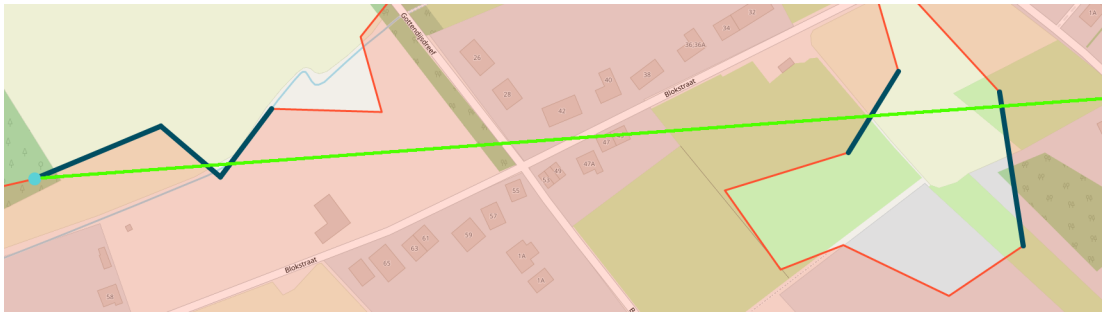

Figure 3.4: Sweeping ray of the visibility graph algorithm
Node: light blue Ray: green Possibly visibility-blocking line segments: dark blue

Additionally, points that are part of the inner ring of an obstacle can also never be visible if the current node is part of the outer ring (and vice-versa for the outer ring when the current node is part of an inner ring).

Three versions of the visibility graph algorithm were implemented. These are: the naive approach, the sweep algorithm from de Berg et al. (2008), and the sweep algorithm with optimizations. As shown in figure 3.1 of the browser UI (page 23), it is possible to switch between these implementations on the fly. As the implementation is quite complex, it is split into several files under `planner/src/nav_graph/`.

The performance of all three algorithms will be compared in section 4.1 of the next chapter: *Results*.

3.1.3 Shortest Path

As discussed in the literature review, shortest path queries can be answered using the A* algorithm (Hart, Nilsson, and Raphael 1968) over the generated visibility graph.

The cost of edges in the graph is their geographical length. The heuristic that guides A*'s search is the geographical distance between the node in question and the goal node. Because the optimal path disregarding obstacles would just be the line between the start and the goal node, the heuristic will always predict a shorter (or equal) distance to the goal as compared to the optimal path's length. This property is called 'admissibility' and guarantees that the returned path will always be the optimal one.

The implementation of A* in the planner is adapted from the original paper by Hart, Nilsson, and Raphael (1968), as well as the *petgraph*[7] Rust library. It is illustrated in figure 3.5 on the next page for two different queries. Its source code is available as `planner/src/nav_graph/bounded_astar/mod.rs`.

[7]Petgraph library: https://docs.rs/petgraph/latest/petgraph/

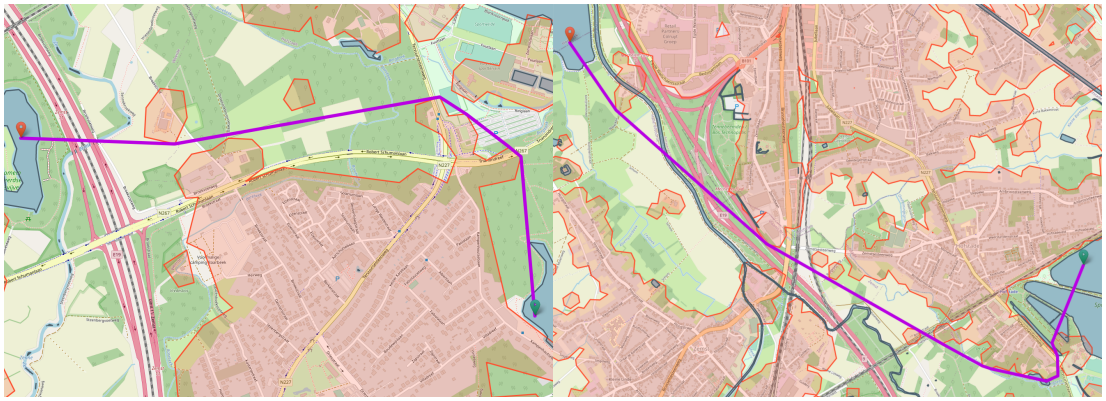

Figure 3.5: Shortest paths between two points
Shortest path: purple

3.1.4 Landing to Recharge

As mentioned before, the planner should not just be able to calculate the shortest path between two points, but it should also be able to generate a path with stops to recharge.

If and when a recharge stop is necessary depends on many factors, including the remaining battery charge, power consumption, and even wind speed. For the purposes of the planner though, these factors can be approximately represented by a 'remaining distance' metric. Similarly, the potential range after charging can be represented by a 'remaining distance after charge' metric (both visible as slider parameters in figure 3.1).

Given these two values, the navigation planner should produce a series of paths (or 'hops') from the start point, to recharge points, and eventually to the end point. This problem can be decomposed into a series of smaller problems: given a start and endpoint, produce a path that either reaches the end point or stops at the last reachable recharge point. By repeatedly solving this smaller problem with a start point ever closer towards the end, a complete path with multiple hops can be generated.

For a given iteration of this smaller problem, there can be multiple 'last reachable' recharge points. There might be, for example, two lakes where the UAV can land: one to the left and one to the right. Whether the planner decides to land on the left or on the right lake depends on which of the resulting total paths would give the shortest flight time (including recharge time). Therefore, finding an optimal solution to the total route problem would require considering all possible total routes. This total grows for each possible hop, akin to the travelling salesman problem, with a worst-case naive time complexity of $O(n!)$ where n is the number of recharge points, plus the start and end point. Although, just like the travelling salesman problem, this can be reduced to exponential time using dynamic programming (Y. Li 2014). An even simpler solution, however, is to use a heuristic to decide between the possible next recharge points, without exploring all of them.

A simple example of such a heuristic is to try to stick to the shortest path *without* recharge hops (the 'ideal' path). In other words, if the ideal path verges to the left, the left recharge point should be selected. To encourage progressing towards the goal, recharge points that are further along the ideal path should also be prioritized. These two goals can be achieved by ordering on a single metric: the distance to the last reachable point along the ideal path. This both encourages sticking to that path, and progressing towards the goal at the same time. Figure 3.6 below illustrates a three-hop route planned using this heuristic. The last reachable points along the ideal path are also shown. Figure 3.7 on the next page gives two more route examples, with respectively four and five hops. The planner algorithm is implemented in `planner/src/nav_graph/planning.rs`.

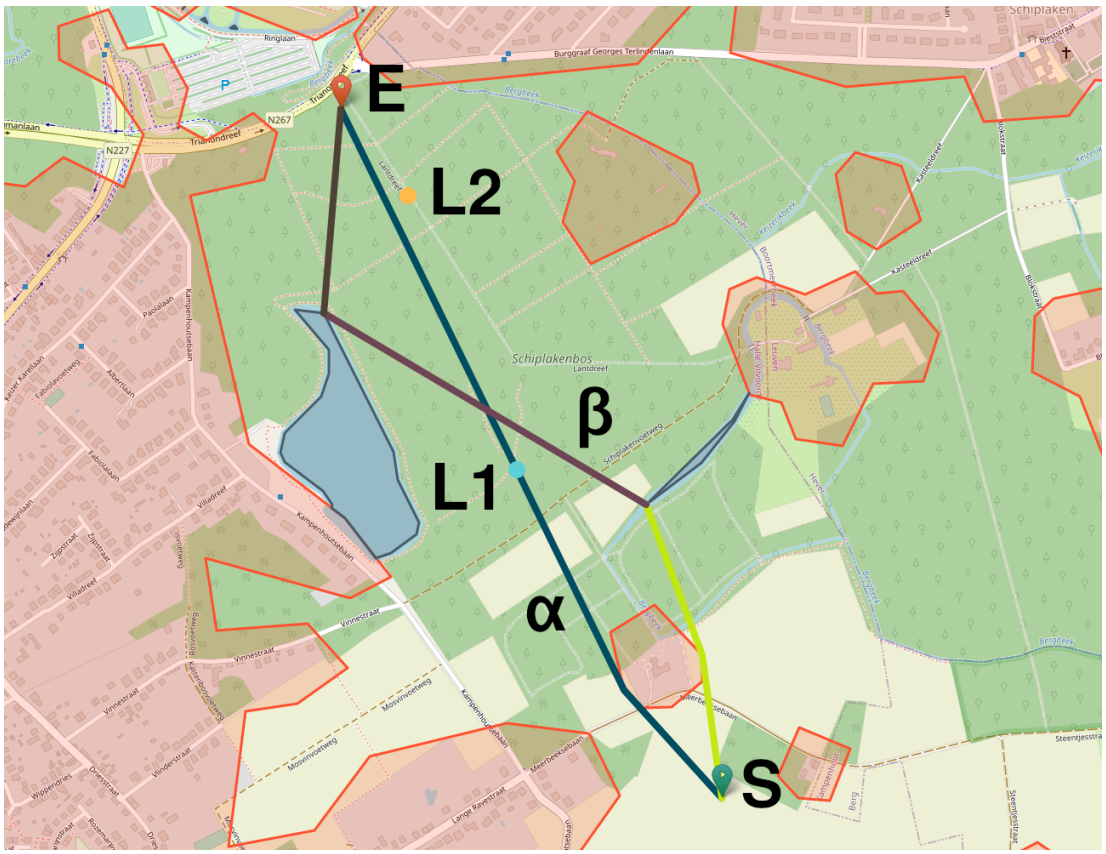

Figure 3.6: Planned route with two recharge stops
S: start node E: end node L1/L2: last reachable point along ideal path of first and second leg respectively α: ideal path of first leg β: three-legged path with recharge stops

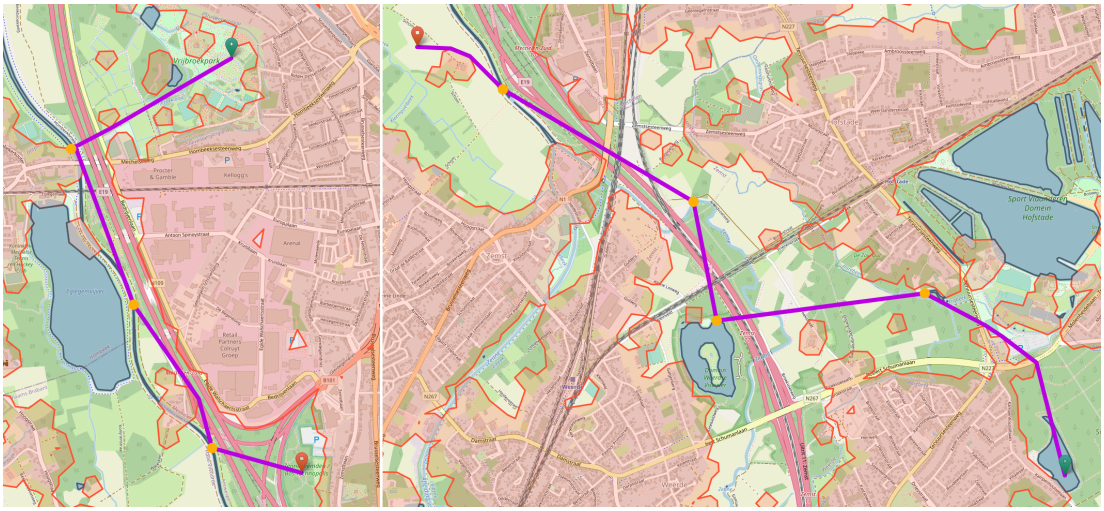

Figure 3.7: Two planned routes with multiple recharge stops *Green/red pin: start/end node Purple: planned path Orange points: recharge stops*

3.1.5 Summary of Planner

Thanks to the browser-based user interface discussed in this section, the operator of Lakehopper is able to view obstacles and water detected by the vision component. The operator can also use the interface to instruct the backend to calculate the shortest path between points, potentially stopping to recharge on water along the way. This functionality is backed by both an efficient visibility graph algorithm and a multi-hop route planner.

3.2 Vision

The vision component consists of three parts. The first of these are the preprocessing scripts. These scripts have two functions: transforming the raw datasets into a format usable for training, and helping in the labelling process. The second part of the vision component is the training notebook, responsible for actually training the neural networks using the processed data. The third and smallest part is the map generation script. This script uses the best performing model to generate maps from aerial imagery for the planner to use.

3.2.1 Preprocessing

As discussed in the literature review section on UAV semantic segmentation datasets (2.2.5), the existing *DroneDeploy Segmentation* dataset[8] as well as the dataset used by Girisha et al. (2021) could be used to train the segmentation model for Lakehopper. Of these, however, only the former could actually be downloaded[9].

The DroneDeploy Segmentation dataset consists of 55 georeferenced drone images, as well as corresponding coloured label masks. An example snippet of these is shown in figure 3.8 below.

Figure 3.8: Sample of the *DroneDeploy Segmentation* dataset
Left: image shot from drone Right: colour label mask

Because the images are very large, both in resolution (4,873 by 4,873 pixels) and geographically ($2km^2$ and up), they need to be divided into smaller 'chips'. These chips have a resolution that is on par with what a small camera on board of the drone would produce (300 by 300 specifically), as well as a field of view that better reflects the target cruising altitude (capturing a few houses or just part of a lake). This process is handled by a small Python script: `vision/src/lakehopper_semseg/preprocess/chipify.py`.

[8]Available at: https://github.com/dronedeploy/dd-ml-segmentation-benchmark
[9]The authors of Girisha et al. (2021) did not respond to a request for data access.

This script also filters out chips that contain sections that should be ignored (e.g., image artefacts or private data). Additionally, it converts the coloured label mask to a binary format better suited to training. Finally, it maps some superfluous classes to ones relevant to Lakehopper. 'Vegetation' for example is mapped to 'background', as for now, Lakehopper does not need to segment vegetation in input images – only water and buildings.

In total, the script extracts 10,325 such chips, nearly all of which show at least some ground. However, only 14% (1,466) contain any buildings, while an even smaller 10% (1,051) contain water. This imbalance in the presence of classes is discussed in the next section.

The DroneDeploy dataset contains enough examples to train a performant model. Nonetheless, it lacks some features common to Western Europe that Lakehopper is likely to encounter, like fields, forests, and canals.

Therefore, in addition to the DroneDeploy dataset, a completely new dataset was created focussing specifically on these features. By combining the two datasets, the aggregated training images are much more diverse. The next section will also discuss 'augmentations' to increase diversity even further.

The new dataset consists of aerial images[10] from the Flemish geographical information agency (AGIV), along with coloured label masks. These masks were created by a Python script that converts AGIV water body and building outlines[11] into masks, which were then edited manually to, for example, remove bridges and overhanging vegetation (see figure 1.3 on page 4) or to add missing buildings. Manual editing was done using both the free tier of a commercial web-based labelling tool[12] and the well-known *Labelme* program[13].

The same *chipify.py* script created to divide the DroneDeploy dataset into chips was also used on this new dataset, resulting in 2,112 chips, with 44% (931) containing buildings and 18% (372) containing water. With that, the aggregated dataset contains 12,437 chips (19%/2397 with buildings and 11%/1423 with water).

3.2.2 Training

As mentioned, the training was performed in a notebook, a *Jupyter Notebooks* in particular (`vision/src/lakehopper_semseg/train.ipynb`). Jupyter Notebooks are a data science tool that combines code with text and visual outputs. In addition to this notebook, some small utility scripts are used (e.g., `vision/src/lakehopper_semseg/visualize.py`).

[10] From the 25cm orthographic colour images dataset: download.vlaanderen.be/Producten/Detail?id=1545

[11] Water bodies: download.vlaanderen.be/Producten/Detail?id=6566, Buildings: download.vlaanderen.be/Producten/Detail?id=6566

[12] Hasty.ai: hasty.ai

[13] Labelme: github.com/wkentaro/labelme

Class Imbalance

As evident by the ratio of chips containing water and buildings, the dataset contains a serious class imbalance. Furthermore, individual chips that do contain water often only do so in small areas.

To counteract the first source of imbalance, only chips that contain at least buildings or water were used for training. This reduces the dataset to 3,583 chips, with 67% (2,397) containing buildings and 40% (1,423) containing water.

Class imbalance within individual chips was addressed through a weight mask. Pixels in this bytewise mask have a high value if they correspond to an under-represented class (water) and a low value if they correspond to an over-represented one (ground). During training, the loss function is weighed by this mask to place more emphasis on sparse classes.

Augmentations

To increase the diversity of the dataset, random augmentations were applied to every image. This way, if batches repeat themselves, they are never exactly the same. Training on this larger dataset makes the model more robust to noise and lighting variation.

The three augmentations used are: flips/mirroring (horizontal and vertical), changes in hue, and changes in brightness. Three samples with random amounts of each augmentation (as passed to the model) are shown in figure 3.9 below.

Figure 3.9: Random augmentations of a sample image
Leftmost: base image Others: sample augmentations

Models

As discussed in section 2.2.2: *CNN Encoders*, three CNN encoders hold particular promise for Lakehopper's vision system: ResNet50, InceptionResNetV2, and EfficientNetB3. Each of these was integrated into the three semantic segmentation decoders discussed in section 2.2.3 *Semantic Segmentation Decoders*: UNet, Fully Convolutional Networks (FCNs), and Feature Pyramid Networks (FPNs). In total, this results in 9 specific models. The performance of each combination of encoder and decoder will be discussed in section 4.2: *Vision* of the next chapter.

Just like the auxiliary data processing scripts, the model definitions and training code were written in Python. Python has a vast array of data science and

scientific libraries (NumPy, Matplotib, SciPy, OpenCV[14] etc.), and it is the most well-supported language of the Tensorflow/Keras[15] framework, which was used to create the models. The `vision/src/lakehopper_semseg/models` directory contains both the encoder and decoder definitions.

Training

Training of the models was accelerated using *Tensorflow Processing Units* (TPUs), provided through Google's *TPU Research Cloud* programme[16] for students and researchers. TPUs are application specific integrated circuits (ASICs) designed for machine learning applications. They cut training time down from possibly several days on a laptop, to under an hour, or to even mere minutes.

The encoder stage of every model was pretrained on ImageNet[17]. This transfer learning also speeds up training significantly (see section 2.2.4: *Transfer Learning* of the previous chapter).

Despite augmentations and an extended dataset, overfitting may still occur. To address this, early stopping was also implemented. This mechanism stops training if loss over the validation dataset does not improve for a certain number of epochs.

Loss curves and training results will be discussed in the next chapter.

3.2.3 Map Generation

As discussed in the problem description (section 1.2 *Problem Description*), the goal of the vision component is twofold: a) to generate maps from aerial imagery for the planner component to use, and b) to, in a later version of Lakehopper, be used in-flight.

The former goal, map generation, is handled by a small Python script that uses the best trained model to predict/'infer' water and buildings from unseen aerial imagery datasets (`vision/src/lakehopper_semseg/map/chips_to_map.py`). It does this for every chip of some larger image. It then converts these outlines to polygons and simplifies them. The resulting map is a drop-in replacement for the test dataset used during development of the planner component.

Using this script, a map from practically any area in North America or Europe can be generated using just aerial imagery.

3.2.4 Summary of Vision

To transform the datasets for the vision component, several preprocessing scripts were created. These, for example, split the data into chips and convert it into

[14]Python scientific libraries: numpy.org, matplotlib.org, scipy.org, opencv.org
[15]Tensorflow/Keras: tensorflow.org / keras.io
[16]TPU Research Cloud: sites.research.google/trc
[17]ImageNet: image-net.org

a format better suited to training. With this data, a Jupyter Notebook trained nine models, each with a different encoder-decoder pair. Finally, a script was created to, using the best model, generate a map of buildings and landable bodies of surface water from aerial imagery.

3.3 Summary of Design & Implementation

Chapter 3 has given an overview of the design and implementation of the high-level planning software for Lakehopper. This design followed decisions made in chapter 2 *Literature Review* and has fulfilled a solution for the problems described in the introductory section 1.2 *Problem Description*.

Chapter 4

Results

Following the implementation of the two system components in chapter 3: *Design & Implementation*, this chapter will discuss the performance results for each component. It will be the basis for the final conclusions and recommendations in chapter 5: *Discussion & Conclusion*.

4.1 Planner

The visibility graph algorithm was implemented in three variations: a naive version, a version closely following de Berg et al.'s (2008) rotational sweep approach, and a final version building upon this sweep approach with various optimizations. As discussed in the literature review (section 2.4: *Path Planning*), this naive version has a time complexity bounded by $O(n^3)$, while both sweep versions are bounded by $O(n^2 \log n)$. n in all cases refers to the number of vertices in the visibility graph.

To test the performance of the algorithms in real-world conditions, each was applied to seven regions of increasing area and centred on the same randomly chosen point (50.995° N, 4.506° E). Table 4.1 below shows the number of nodes in each region.

Area (km^2)	500	1500	2000	2500	3000	3400	3500
Number of vertices	54	516	836	1303	1728	2206	2711

Table 4.1: Test areas for visibility graph algorithms

The processing time of every run was measured using the system's high-resolution monotonic clock. Figure 4.1 below shows the results of these tests. This graph also shows the – constant-factor corrected – big O bound for each variation, demonstrating that the algorithms more or less follow their expected time complexity curve.

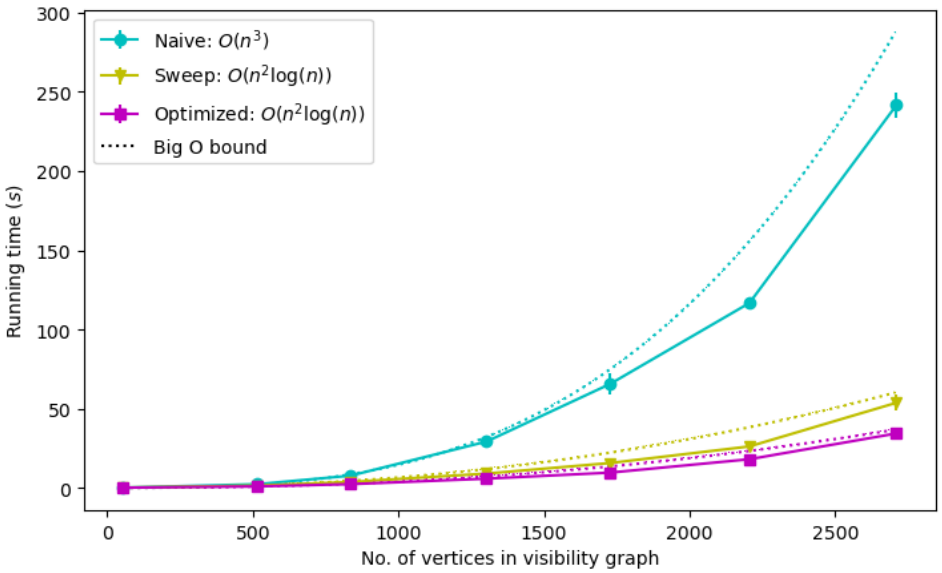

Figure 4.1: Running time of visibility graph algorithms

While the difference between the base and the optimized versions of the sweep approach is initially small, the gap between them does grow quadratically (see

bottom right). Unfortunately, even the running time of the optimized algorithm already grows to over 20 seconds above 2500 nodes. More optimizations and a divide-and-conquer approach might be able to tame this growth, but it is worth considering other approaches to calculate shortest paths that do not rely on a visibility graph. This will be further discussed in section 5.2: *Future work* of the final chapter.

Manual tests of the multi-hop planner algorithm showed that a route could always be found in under a second. As the algorithm uses backtracking in case it gets stuck after a hop, it will always find a route when possible.

4.2 Vision

As discussed in the previous chapter, models of all nine encoder-decoder combinations were trained on a 3,583-sample large dataset of both DroneDeploy and AGIV aerial imagery.

All models were trained to a complete 40 epochs, after which validation loss did not improve (see figure 4.2.a below). Figure 4.2.b shows that even after saturating the training loss, overfitting did not occur (notice how the validation loss did not deviate up at the end).

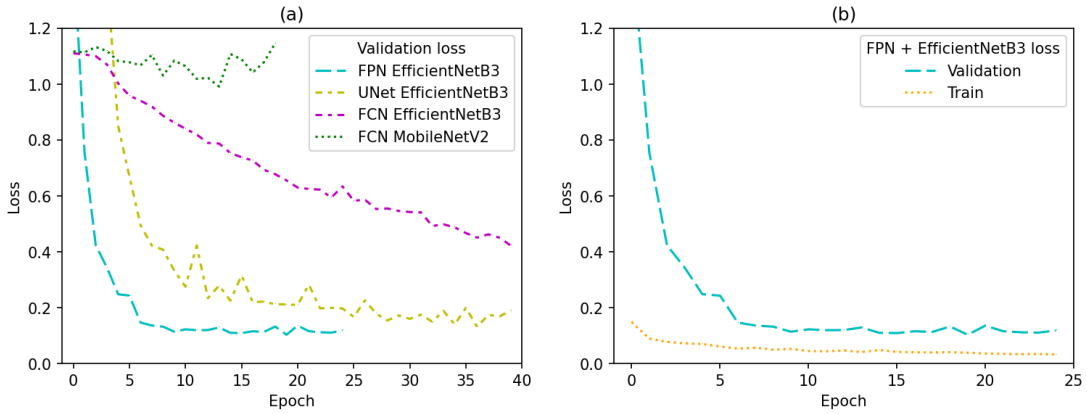

Figure 4.2: (a): Validation loss during training of four segmentation models (b): Validation and training loss for the FPN + EfficientNetB3 model

To evaluate the performance of the models, five common metrics were calculated over the entire test dataset: accuracy, precision, recall, F1 score, and the mean intersection of union (MIoU) over all classes. While the IoU of the different classes did vary somewhat (favouring the background class), this variation was consistent across all models.

Table 4.2 on the following page shows these metrics for all encoder-decoder combinations. The model based on a feature pyramid network (FPN) and the EfficientNetB3 encoder performed the best on all metrics (shown in bold). Based on results of previous research (see section 2.2.2: *CNN Encoders* of the literature

review), it is not surprising that EfficientNetB3 performs better than ResNet or MobileNetV2. What is surprising, is the poor performance of the FCN-based network with a MobileNetV2 encoder. The learning rate for MobileNetV2-based networks stagnated much sooner than models with the other encoders. For UNet, FCN, and FPN this was at epoch 20, 13, and 10 respectively. The FPN+EfficientNetB3 model on the other hand trained until epoch 39 before stagnating. The particular implementation used might also hurt the performance of MobileNetV2 networks. Section 5.2: *Future work* of the next chapter will discuss this further.

Decoder	Encoder	Acc.	Precision	Recall	F1-score	MIoU
UNet	MobileNetV2	0.92	0.92	0.91	0.91	0.72
	InceptionResNetV2	0.94	0.94	0.94	0.94	0.77
	EfficientNetB3	0.96	0.96	0.95	0.96	0.81
FCN	MobileNetV2	0.69	0.47	0.15	0.22	0.49
	InceptionResNetV2	0.92	0.93	0.90	0.92	0.75
	EfficientNetB3	0.95	0.96	0.93	0.94	0.81
FPN	MobileNetV2	0.91	0.92	0.91	0.91	0.71
	InceptionResNetV2	0.94	0.94	0.94	0.94	0.76
	EfficientNetB3	**0.96**	**0.96**	**0.96**	**0.96**	**0.83**

Table 4.2: Prediction performance metrics of segmentation models

The two next best models after FPN + EfficientNetB3 are respectively UNet and FCN with EfficientNetB3. Figure 4.3 on the next page shows a comparison of the predictions of all three of these models as compared to the ground truth, as well as to the worst model: FCN + MobileNetV2. It shows that the best two models struggle with strongly defining the shape of buildings, wrongly including many surrounding pixels. Interestingly, the third best model: FCN+EfficientNetB3, performs better in this regard, showing smooth outlines of both buildings and water. The biggest obstacle of the FCN+MobileNetV2 model is differentiating buildings from their surrounding clutter. The predictions this model makes for water are however satisfactory if somewhat course. Overall, the performance of all models is acceptable as small wrongly identified patches of water are in any case not large enough to land. Over-prediction of buildings is also preferable over under-prediction.

A thorough analysis of the computational complexity and power consumption of the models is currently not possible. This is because the deployment conditions of the vision component are not yet known. For use on board of the drone, for example, the CPU of a single board computer (SBC) could be used. Alternatively, the model might be run on an external machine learning accelerator (e.g., the Coral.ai USB Accelerator[1] or the Intel Neural Compute Stick 2[2]).

[1]Coral.ai USB Accelerator: coral.ai/products/accelerator

[2]Intel Neural Compute Stick:
intel.com/content/www/us/en/developer/tools/neural-compute-stick/overview

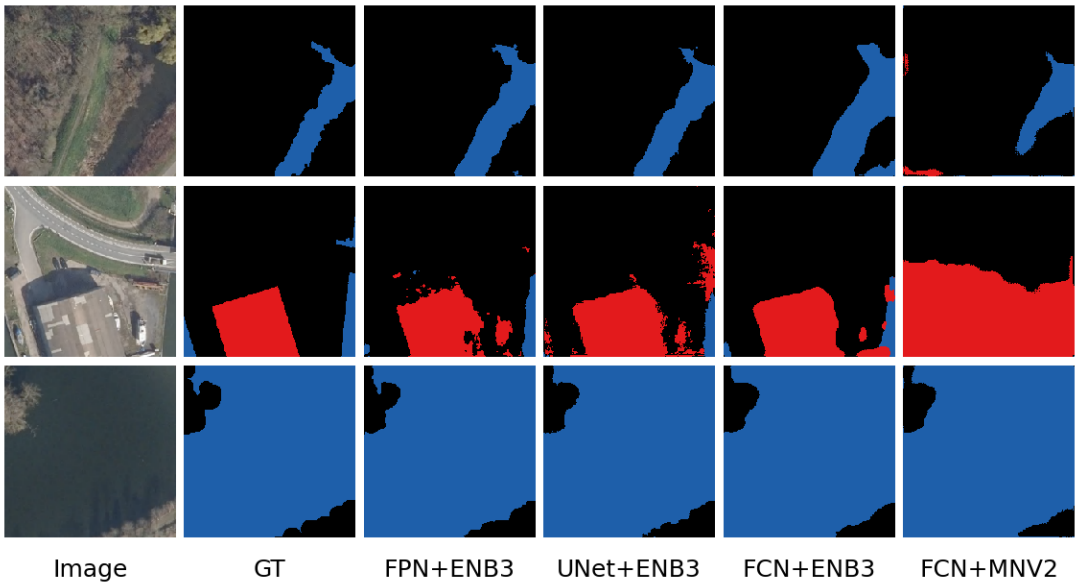

Figure 4.3: Segmentation model predictions for sample images
GT: Ground Truth ENB3: EfficientNetB3 MNV2: MobileNetV2

4.3 Summary of Results

This chapter has discussed the performance and results of both the planner and the vision component of Lakehopper, as implemented in chapter 3: *Design & Implementation*. It found that, while further research could improve performance, the current implementation has met its goals as set in the problem description (section 1.2).

Chapter 5

Discussion & Conclusion

This final chapter will summarize the process and findings of this dissertation. To conclude, it will review possible future avenues of research as well as the work that needs to be done to complete the Lakehopper prototype.

5.1 Conclusion

This dissertation set out to solve the high-level planning problem for Lakehopper, a work-in-progress UAV prototype and its supporting systems. Using the software created to solve this problem, Lakehopper will be able to extend the duration of missions to, for example, observe wildlife or monitor the environment. It will achieve this by hopping between lakes, recharging its batteries with solar cells at each stop. This way, Lakehopper can travel between far removed points of interest or loiter in one area for long periods of time.

To develop this software system, a comprehensive literature review was conducted. This review uncovered the most promising algorithms and techniques for the two software components of the high-level planning system: the planner and the vision component.

The planner was developed with an efficient visibility graph algorithm, which builds on existing work with optimizations and added functionality. This optimized version can process large areas up to two times faster than the existing algorithm it was based on. Using the generated graph, the planner's route solver can effectively calculate near-optimal multi-hop paths. These paths avoid built-up areas and airspace restrictions.

The vision component was created to construct the map of lakes and buildings that the planner relies on. It does this by using a computer vision algorithm able to classify sections of aerial imagery as water, buildings, or ground with an accuracy of 96%. This approach does not rely on existing maps, thus ensuring that landing sites are neither obstructed by vegetation, nor dried up due to seasonal changes. In other words, it can generate suitable navigational maps for any area in North America or Europe, using just aerial imagery. This algorithm will also be adaptable to an in-flight deployment to verify the state of a site before landing and to check for obstacles on the water.

5.2 Future work

This dissertation can serve as the starting point for a great variety of possible future research, from machine vision and robotics to computational geometry. The following sections will give an overview of some of these possible directions, firstly in research and secondly specifically with the goal to complete Lakehopper.

5.2.1 Research

As discussed in this dissertation's introduction, the planner could be extended to support dynamic obstacles. Additionally, it can be developed to take a weighted area cost into account. This would allow the planner to take more nuanced decisions. For example, it could plan a path around a wildlife conservation area, even if flying over it is technically possible.

The previous chapter (chapter 4: *Results*) showed that even the best visibility graph algorithm in this dissertation has an quadratic-logarithmic time complexity. As mentioned, this might be improved using a divide-and-conquer approach, where only a subset of the graph is considered at a time. Alternatively, the navigation graph could be generated using probabilistic roadmaps or a dynamically expanding graph. It might also be possible to calculate shortest paths directly using computational geometry instead of over a graph. Any of these revisions could offer significant improvements in performance. If proven necessary by the real-world application of the planner component, they should be explored further.

Also for the vision component, performance can be improved in various ways. For one, the existing model implementations can be further fine-tuned. Alternatively, completely different CNN architectures like Convnext (Dollár, M. Singh, and Girshick 2021) and Regnet (Z. Liu et al. 2022) should be considered. Finally, the existing dataset could be expanded, and perhaps more importantly, its labels should be improved.

5.2.2 Lakehopper

In order to deploy the systems developed in this dissertation, the physical Lakehopper prototype must of course be built. This requires further research into UAV design. Apart from the high-level planner, lower-level control routines must also be developed (e.g., for low passes, takeoff, and landings). Finally, all software components, as well as ground station communications, have to be integrated into a complete system. A solution like the *Robot Operating System*[1] (ROS) might be useful for this.

[1]Robot Operating System: ros.org

Bibliography

Agafonkin, Volodymyr (July 2016). *Polylabel: A Fast Algorithm for Finding the Pole of Inaccessibility of a Polygon*. URL: https://github.com/mapbox/polylabel (visited on 08/02/2022).

Agarwal, Pankaj and Mukesh Kr. Singh (Feb. 2019). "A Multipurpose Drone for Water Sampling & Video Surveillance". In: *2019 Second International Conference on Advanced Computational and Communication Paradigms (ICACCP)*. 2019 Second International Conference on Advanced Computational and Communication Paradigms (ICACCP). Gangtok, India: IEEE, pp. 1–5. ISBN: 978-1-5386-7989-0. DOI: 10.1109/ICACCP.2019.8883017. (Visited on 05/07/2022).

Albawi, Saad, Tareq Abed Mohammed, and Saad Al-Zawi (Aug. 2017). "Understanding of a Convolutional Neural Network". In: *2017 International Conference on Engineering and Technology (ICET)*. 2017 International Conference on Engineering and Technology (ICET), pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.

Allan Poe, Edgar (1845). *The Raven*. URL: https://www.poetryfoundation.org/poems/48860/the-raven (visited on 05/06/2022).

Aloysius, Neena and M. Geetha (Apr. 2017). "A Review on Deep Convolutional Neural Networks". In: *2017 International Conference on Communication and Signal Processing (ICCSP)*. 2017 International Conference on Communication and Signal Processing (ICCSP), pp. 0588–0592. DOI: 10.1109/ICCSP.2017.8286426.

Alt, H. and E. Welzl (May 1, 1988). "Visibility Graphs and Obstacle-Avoiding Shortest Paths". In: *Zeitschrift für Operations Research* 32.3, pp. 145–164. ISSN: 1432-5217. DOI: 10.1007/BF01928918. (Visited on 07/12/2022).

Amato, Nancy M. et al. (Aug. 1, 1998). "OBPRM: An Obstacle-Based PRM for 3D Workspaces". In: *Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics on Robotics : The Algorithmic Perspective: The Algorithmic Perspective*. WAFR '98. USA: A. K. Peters, Ltd., pp. 155–168. ISBN: 978-1-56881-081-2.

Andrew, William, Colin Greatwood, and Tilo Burghardt (Nov. 2019). "Aerial Animal Biometrics: Individual Friesian Cattle Recovery and Visual Identi-

fication via an Autonomous UAV with Onboard Deep Inference". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Macau, China: IEEE, pp. 237–243. ISBN: 978-1-72814-004-9. DOI: 10.1109/IROS40897.2019.8968555. (Visited on 03/17/2022).

Ardito, Luca et al. (Feb. 26, 2021). "Evaluation of Rust Code Verbosity, Understandability and Complexity". In: *PeerJ Computer Science* 7, e406. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.406. (Visited on 07/14/2022).

Ayhan, Bulent et al. (Aug. 2019). "Semi-Automated Emergency Landing Site Selection Approach for UAVs". In: *IEEE Transactions on Aerospace and Electronic Systems* 55.4, pp. 1892–1906. ISSN: 1557-9603. DOI: 10.1109/TAES.2018.2879529.

Bacha, Andrew et al. (2008). "Odin: Team VictorTango's Entry in the DARPA Urban Challenge". In: *Journal of Field Robotics* 25.8, pp. 467–492. ISSN: 1556-4967. DOI: 10.1002/rob.20248. (Visited on 05/06/2022).

Badrinarayanan, Vijay, Alex Kendall, and Roberto Cipolla (Oct. 10, 2016). *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation*. arXiv: 1511.00561 [cs]. URL: http://arxiv.org/abs/1511.00561 (visited on 07/10/2022).

Bahl, Gaetan et al. (2019). "Low-Power Neural Networks for Semantic Segmentation of Satellite Images". In: Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops, pp. 0–0. URL: https://openaccess.thecvf.com/content_ICCVW_2019/html/LPCV/Bahl_Low-Power_Neural_Networks_for_Semantic_Segmentation_of_Satellite_Images_ICCVW_2019_paper.html (visited on 03/17/2022).

Balasubramanian, Abhiram et al. (May 7, 2017). "System Programming in Rust: Beyond Safety". In: *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*. HotOS '17. New York, NY, USA: Association for Computing Machinery, pp. 156–161. ISBN: 978-1-4503-5068-6. DOI: 10.1145/3102980.3103006. (Visited on 07/14/2022).

Banerjee, Bikram Pratap et al. (July 2, 2020). "Development of a UAV-mounted System for Remotely Collecting Mine Water Samples". In: *International Journal of Mining, Reclamation and Environment* 34.6, pp. 385–396. ISSN: 1748-0930. DOI: 10.1080/17480930.2018.1549526. (Visited on 05/07/2022).

Barbedo, Jayme Garcia Arnal et al. (Apr. 10, 2020). "Counting Cattle in UAV Images—Dealing with Clustered Animals and Animal/Background Contrast Changes". In: *Sensors* 20.7, p. 2126. ISSN: 1424-8220. DOI: 10.3390/s20072126. (Visited on 03/17/2022).

Belghith, Khaled et al. (Jan. 1, 2006). "Anytime Dynamic Path-planning with Flexible Probabilistic Roadmaps." In: vol. 2006, pp. 2372–2377. DOI: 10.1109/ROBOT.2006.1642057.

Bhatnagar, Saheba, Laurence Gill, and Bidisha Ghosh (Jan. 2020). "Drone Image Segmentation Using Machine and Deep Learning for Mapping Raised Bog Vegetation Communities". In: *Remote Sensing* 12.16 (16), p. 2602. ISSN: 2072-4292. DOI: 10.3390/rs12162602. (Visited on 07/02/2022).

Bird, Clara N. et al. (Jan. 2020). "A Semi-Automated Method for Estimating Adélie Penguin Colony Abundance from a Fusion of Multispectral and Thermal Imagery Collected with Unoccupied Aircraft Systems". In: *Remote Sensing* 12.22 (22), p. 3692. ISSN: 2072-4292. DOI: 10.3390/rs12223692. (Visited on 05/07/2022).

Bohlin, R. and L.E. Kavraki (Apr. 2000). "Path Planning Using Lazy PRM". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065).* Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065). Vol. 1, 521–528 vol.1. DOI: 10.1109/ROBOT.2000.844107.

Boor, V., M.H. Overmars, and A.F. van der Stappen (May 1999). "The Gaussian Sampling Strategy for Probabilistic Roadmap Planners". In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C).* Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C). Vol. 2, 1018–1023 vol.2. DOI: 10.1109/ROBOT.1999.772447.

Brunton, Elizabeth A., Javier X. Leon, and Scott E. Burnett (June 2020). "Evaluating the Efficacy and Optimal Deployment of Thermal Infrared and True-Colour Imaging When Using Drones for Monitoring Kangaroos". In: *Drones* 4.2 (2), p. 20. ISSN: 2504-446X. DOI: 10.3390/drones4020020. (Visited on 05/07/2022).

Bustamante, Juan M. et al. (Sept. 2019). "Design and Construction of a UAV VTOL in Ducted-Fan and Tilt-Rotor Configuration". In: *2019 16th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE).* 2019 16th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE), pp. 1–6. DOI: 10.1109/ICEEE.2019.8884533.

Cambridge English Dictionary (2022). *Drone.* In: *Cambridge English Dictionary.* URL: https://dictionary.cambridge.org/dictionary/english/drone (visited on 05/06/2022).

Campion, Mitch, Prakash Ranganathan, and Saleh Faruque (June 2019). "UAV Swarm Communication and Control Architectures: A Review". In: *J. Unmanned Veh. Sys.* 7.2, pp. 93–106. ISSN: 2291-3467. DOI: 10.1139/juvs-2018-0009. (Visited on 05/07/2022).

Chaurasia, Abhishek and Eugenio Culurciello (Dec. 2017). "LinkNet: Exploiting Encoder Representations for Efficient Semantic Segmentation". In: *2017*

IEEE Visual Communications and Image Processing (VCIP), pp. 1–4. DOI: 10.1109/VCIP.2017.8305148. (Visited on 07/10/2022).

Chen, Yunfei, Wei Feng, and Gan Zheng (Feb. 2018). "Optimum Placement of UAV as Relays". In: *IEEE Communications Letters* 22.2, pp. 248–251. ISSN: 1558-2558. DOI: 10.1109/LCOMM.2017.2776215.

Chu, Yauhei et al. (May 24, 2021). "Development of a Solar-Powered Unmanned Aerial Vehicle for Extended Flight Endurance". In: *Drones* 5.2, p. 44. ISSN: 2504-446X. DOI: 10.3390/drones5020044. (Visited on 05/07/2022).

De Berg, Mark et al. (2008). *Computational Geometry: Algorithms and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-77973-5 978-3-540-77974-2. DOI: 10.1007/978-3-540-77974-2. (Visited on 06/16/2022).

Dijkstra, E. W. (Dec. 1, 1959). "A Note on Two Problems in Connexion with Graphs". In: *Numer. Math.* 1.1, pp. 269–271. ISSN: 0945-3245. DOI: 10.1007/BF01386390. (Visited on 07/12/2022).

DJI (2022). *DJI Mavic 3*. URL: https://www.dji.com/be/mavic-3 (visited on 05/06/2022).

Doi, Hideyuki et al. (Nov. 2017). "Water Sampling for Environmental DNA Surveys by Using an Unmanned Aerial Vehicle: *Drone Water Sampling for eDNA*". In: *Limnol. Oceanogr. Methods* 15.11, pp. 939–944. ISSN: 15415856. DOI: 10.1002/lom3.10214. (Visited on 05/07/2022).

Dollár, Piotr, Mannat Singh, and Ross Girshick (Mar. 11, 2021). *Fast and Accurate Model Scaling*. DOI: 10.48550/arXiv.2103.06877. (Visited on 07/31/2022).

Dong, Fangyun et al. (May 2019). "Energy-Efficiency for Fixed-Wing UAV-Enabled Data Collection and Forwarding". In: *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*. 2019 IEEE International Conference on Communications Workshops (ICC Workshops), pp. 1–6. DOI: 10.1109/ICCW.2019.8757098.

Dwivedi, Vijay Shankar et al. (Aug. 2018). "MARAAL: A Low Altitude Long Endurance Solar Powered UAV For Surveillance and Mapping Applications". In: *2018 23rd International Conference on Methods Models in Automation Robotics (MMAR)*. 2018 23rd International Conference on Methods Models in Automation Robotics (MMAR), pp. 449–454. DOI: 10.1109/MMAR.2018.8485805.

Federal Aviation Administration (June 28, 2016). *CFR Title 14 Part 107 - Small Unmanned Aircraft Systems*. URL: https://www.ecfr.gov/current/title-14/chapter-I/subchapter-F/part-107 (visited on 05/06/2022).

Federal Aviation Administration (2022). *What Is a NOTAM?* Federal Aviation Administration. URL: https://www.faa.gov/about/initiatives/notam/what_is_a_notam (visited on 05/08/2022).

Fiers, Pieter (Feb. 2021). *Autonomously Following Forest Paths with a Mobile Robot Using Semantic Segmentation*. URL: https://paddy.pfiers.net/README.html (visited on 03/18/2022).

Francis, Anthony et al. (Aug. 2020). "Long-Range Indoor Navigation With PRM-RL". In: *IEEE Transactions on Robotics* 36.4, pp. 1115–1134. ISSN: 1941-0468. DOI: 10.1109/TRO.2020.2975428.

Fu, Yanqing, Morgan Kinniry, and Laura N. Kloepper (June 2018). "The Chirocopter: A UAV for Recording Sound and Video of Bats at Altitude". In: *Methods Ecol Evol* 9.6. Ed. by Graziella Iossa, pp. 1531–1535. ISSN: 2041-210X, 2041-210X. DOI: 10.1111/2041-210X.12992. (Visited on 03/18/2022).

Garber, Steve (Aug. 2012). *Style Guide for NASA History Authors and Editors*. URL: https://history.nasa.gov/printFriendly/styleguide.html (visited on 07/30/2022).

Garmin (2022). *Garmin Autonomí*. URL: https://discover.garmin.com/en-GB/autonomi/ (visited on 05/06/2022).

Ghambari, Soheila et al. (Dec. 2020). "UAV Path Planning in the Presence of Static and Dynamic Obstacles". In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 465–472. DOI: 10.1109/SSCI47803.2020.9308340.

Girisha, S. et al. (June 2019). "Semantic Segmentation of UAV Aerial Videos Using Convolutional Neural Networks". In: *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*. 2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), pp. 21–27. DOI: 10.1109/AIKE.2019.00012.

Girisha, S. et al. (2021). "UVid-Net: Enhanced Semantic Segmentation of UAV Aerial Videos by Embedding Temporal Information". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 14, pp. 4115–4127. ISSN: 2151-1535. DOI: 10.1109/JSTARS.2021.3069909.

Guo, Xufeng et al. (Nov. 2014). "Automatic UAV Forced Landing Site Detection Using Machine Learning". In: *2014 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*. 2014 International Conference on Digital Image Computing: Techniques and Applications (DICTA). Wollongong, New South Wales, Australia: IEEE, pp. 1–7. ISBN: 978-1-4799-5409-4. DOI: 10.1109/DICTA.2014.7008097. (Visited on 03/17/2022).

Guo, Yanming et al. (June 1, 2018). "A Review of Semantic Segmentation Using Deep Neural Networks". In: *Int J Multimed Info Retr* 7.2, pp. 87–93. ISSN: 2192-662X. DOI: 10.1007/s13735-017-0141-z. (Visited on 05/08/2022).

Hadi, Ghozali et al. (Jan. 1, 2014). "Autonomous UAV System Development for Payload Dropping Mission". In: Journal of Instrumentation, Automation and Systems. Vol. 1, pp. 72–77.

Hart, Peter E., Nils J. Nilsson, and Bertram Raphael (July 1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107. ISSN: 2168-2887. DOI: 10.1109/TSSC.1968.300136.

He, Kaiming et al. (Dec. 10, 2015). *Deep Residual Learning for Image Recognition*. arXiv: 1512.03385 [cs]. URL: http://arxiv.org/abs/1512.03385 (visited on 07/09/2022).

He, Lvlong et al. (Jan. 1, 2018). "Feedback Formation Control of UAV Swarm with Multiple Implicit Leaders". In: *Aerospace Science and Technology* 72, pp. 327–334. ISSN: 1270-9638. DOI: 10.1016/j.ast.2017.11.020. (Visited on 05/07/2022).

Howard, Andrew G. et al. (Apr. 16, 2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. arXiv: 1704.04861 [cs]. URL: http://arxiv.org/abs/1704.04861 (visited on 07/09/2022).

Hsu, D., Tingting Jiang, et al. (Sept. 2003). "The Bridge Test for Sampling Narrow Passages with Probabilistic Roadmap Planners". In: *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*. 2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422). Vol. 3, 4420–4426 vol.3. DOI: 10.1109/ROBOT.2003.1242285.

Hsu, D., G. Sanchez-Ante, and Zheng Sun (Apr. 2005). "Hybrid PRM Sampling with a Cost-Sensitive Adaptive Strategy". In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. Proceedings of the 2005 IEEE International Conference on Robotics and Automation, pp. 3874–3880. DOI: 10.1109/ROBOT.2005.1570712.

Hsu, MoWei (Nov. 2020). "AquaFly: A Tilt-rotor Vertical Take-off and Landing Aquatic Unmanned Aerial Vehicle". Toronto: University of Toronto. URL: https://tspace.library.utoronto.ca/bitstream/1807/103640/1/Hsu_MoWei_202011_MAS_thesis.pdf (visited on 05/07/2022).

Ji, Yuzhu et al. (Feb. 2021). "CNN-based Encoder-Decoder Networks for Salient Object Detection: A Comprehensive Review and Recent Advances". In: *Information Sciences* 546, pp. 835–857. ISSN: 00200255. DOI: 10.1016/j.ins.2020.09.003. (Visited on 08/02/2022).

Joyce, Karen E., Karen Anderson, and Renee E. Bartolo (Mar. 2021). "Of Course We Fly Unmanned—We're Women!" In: *Drones* 5.1 (1), p. 21. ISSN: 2504-446X. DOI: 10.3390/drones5010021. (Visited on 05/06/2022).

Kang, Byeongkeun and Truong Q. Nguyen (July 2019). "Random Forest With Learned Representations for Semantic Segmentation". In: *IEEE Transactions on Image Processing* 28.7, pp. 3542–3555. ISSN: 1941-0042. DOI: 10.1109/TIP.2019.2905081.

Kellenberger, Benjamin, Michele Volpi, and Devis Tuia (July 2017). "Fast Animal Detection in UAV Images Using Convolutional Neural Networks".

In: *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. 2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS). Fort Worth, TX: IEEE, pp. 866–869. ISBN: 978-1-5090-4951-6. DOI: 10.1109/IGARSS.2017.8127090. (Visited on 03/17/2022).

Kloepper, Laura N. and Morgan Kinniry (May 17, 2018). "Recording Animal Vocalizations from a UAV: Bat Echolocation during Roost Re-Entry". In: *Sci Rep* 8.1 (1), p. 7779. ISSN: 2045-2322. DOI: 10.1038/s41598-018-26122-z. (Visited on 03/17/2022).

Koparan, Cengiz and Ali B. Koc (July 17, 2016). "Unmanned Aerial Vehicle (UAV) Assisted Water Sampling". In: *2016 ASABE International Meeting*. 2016 ASABE International Meeting. American Society of Agricultural and Biological Engineers. DOI: 10.13031/aim.20162461157. (Visited on 05/07/2022).

Koparan, Cengiz, Ali B. Koc, Charles V. Privette, et al. (Mar. 3, 2018). "In Situ Water Quality Measurements Using an Unmanned Aerial Vehicle (UAV) System". In: *Water* 10.3, p. 264. ISSN: 2073-4441. DOI: 10.3390/w10030264. (Visited on 03/18/2022).

Koparan, Cengiz, Ali B. Koc, Charles V. Privette, et al. (Mar. 2019). "Autonomous In Situ Measurements of Noncontaminant Water Quality Indicators and Sample Collection with a UAV". In: *Water* 11.3 (3), p. 604. ISSN: 2073-4441. DOI: 10.3390/w11030604. (Visited on 05/07/2022).

Koparan, Cengiz, Ali B. Koc, Charles V. Privette, et al. (Mar. 2020). "Adaptive Water Sampling Device for Aerial Robots". In: *Drones* 4.1 (1), p. 5. ISSN: 2504-446X. DOI: 10.3390/drones4010005. (Visited on 05/07/2022).

Lakdawalla, Emily (2022). *Finding New Language for Space Missions That Fly without Humans*. The Planetary Society. URL: https://www.planetary.org/articles/10050900-finding-new-language (visited on 07/30/2022).

Li, Bo et al. (2020). "Trajectory Planning for UAV Based on Improved ACO Algorithm". In: *IEEE Access* 8, pp. 2995–3006. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2962340.

Li, Boyang et al. (Oct. 13, 2016). "Development and Testing of a Two-UAV Communication Relay System". In: *Sensors* 16.10, p. 1696. ISSN: 1424-8220. DOI: 10.3390/s16101696. (Visited on 03/18/2022).

Li, Yunpeng (2014). "A New Exact Algorithm for Traveling Salesman Problem". Version 4. In: DOI: 10.48550/ARXIV.1412.2437. (Visited on 07/25/2022).

Lin, Tsung-Yi et al. (Apr. 19, 2017). *Feature Pyramid Networks for Object Detection*. arXiv: 1612.03144 [cs]. URL: http://arxiv.org/abs/1612.03144 (visited on 07/10/2022).

Liu, Weibo et al. (Apr. 19, 2017). "A Survey of Deep Neural Network Architectures and Their Applications". In: *Neurocomputing* 234, pp. 11–26. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2016.12.038. (Visited on 05/08/2022).

Liu, Zhuang et al. (Mar. 2, 2022). *A ConvNet for the 2020s*. DOI: 10.48550/arXiv.2201.03545. (Visited on 07/31/2022).

Lou, Bin et al. (Dec. 2019). "Preliminary Design and Performance Analysis of a Solar-Powered Unmanned Seaplane". In: *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* 233.15, pp. 5606–5617. ISSN: 0954-4100, 2041-3025. DOI: 10.1177/0954410019852572. (Visited on 05/07/2022).

Lu, Yuncheng et al. (Jan. 2, 2018). "A Survey on Vision-Based UAV Navigation". In: *Geo-spatial Information Science* 21.1, pp. 21–32. ISSN: 1009-5020, 1993-5153. DOI: 10.1080/10095020.2017.1420509. (Visited on 05/07/2022).

Lyu, Ye et al. (July 1, 2020). "UAVid: A Semantic Segmentation Dataset for UAV Imagery". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 165, pp. 108–119. ISSN: 0924-2716. DOI: 10.1016/j.isprsjprs.2020.05.009. (Visited on 03/17/2022).

Mahony, Niall O' et al. (2020). *Deep Learning vs. Traditional Computer Vision*. Vol. 943. DOI: 10.1007/978-3-030-17795-9. (Visited on 07/09/2022).

Marcu, Alina et al. (2018). "SafeUAV: Learning to Estimate Depth and Safe Landing Areas for UAVs from Synthetic Data". In: Proceedings of the European Conference on Computer Vision (ECCV) Workshops, pp. 0–0. URL: https://openaccess.thecvf.com/content_eccv_2018_workshops/w7/html/Marcu_SafeUAV_Learning_to_estimate_depth_and_safe_landing_areas_for_ECCVW_2018_paper.html (visited on 05/08/2022).

Merriam-Webster Dictionary (2022). *Drone*. In: *Merriam-Webster Dictionary*. URL: https://www.merriam-webster.com/dictionary/drone (visited on 05/06/2022).

Oettershagen, Philipp et al. (May 2015). "A Solar-Powered Hand-Launchable UAV for Low-Altitude Multi-Day Continuous Flight". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015 IEEE International Conference on Robotics and Automation (ICRA), pp. 3986–3993. DOI: 10.1109/ICRA.2015.7139756.

Ore, John-Paul et al. (Dec. 2015). "Autonomous Aerial Water Sampling: Autonomous Aerial Water Sampling". In: *J. Field Robotics* 32.8, pp. 1095–1113. ISSN: 15564959. DOI: 10.1002/rob.21591. (Visited on 05/07/2022).

Paredes, Juan Augusto et al. (Aug. 2017). "Study of Effects of High-Altitude Environments on Multicopter and Fixed-Wing UAVs' Energy Consumption and Flight Time". In: *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*. 2017 13th IEEE Conference on Automation Science and Engineering (CASE), pp. 1645–1650. DOI: 10.1109/COASE.2017.8256340.

Parmar, Vivek et al. (July 6, 2020). *Exploration of Optimized Semantic Segmentation Architectures for Edge-Deployment on Drones*. DOI: 10.48550/arXiv.2007.02839. (Visited on 07/02/2022).

Pekel, Jean-François et al. (Dec. 15, 2016). "High-Resolution Mapping of Global Surface Water and Its Long-Term Changes". In: *Nature* 540.7633, pp. 418–422. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature20584. (Visited on 08/01/2022).

Perkel, Jeffrey M. (Dec. 1, 2020). "Why Scientists Are Turning to Rust". In: *Nature* 588.7836 (7836), pp. 185–186. DOI: 10.1038/d41586-020-03382-2. (Visited on 07/14/2022).

Pinkney, M.F.J., D. Hampel, and S. DiPierro (Oct. 1996). "Unmanned Aerial Vehicle (UAV) Communications Relay". In: *Proceedings of MILCOM '96 IEEE Military Communications Conference*. Proceedings of MILCOM '96 IEEE Military Communications Conference. Vol. 1, 47–51 vol.1. DOI: 10.1109/MILCOM.1996.568581.

Plumet, Frederic et al. (Apr. 2015). "Toward an Autonomous Sailing Boat". In: *IEEE J. Oceanic Eng.* 40.2, pp. 397–407. ISSN: 0364-9059, 1558-1691, 2373-7786. DOI: 10.1109/JOE.2014.2321714. (Visited on 05/06/2022).

Polonelli, Tommaso et al. (2020). "A Flexible, Low-Power Platform for UAV-Based Data Collection From Remote Sensors". In: *IEEE Access* 8, pp. 164775–164785. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3021370.

Puri, Anika and Elizabeth Bondi (2021). *Space, Time, and Counts: Improved Human vs Animal Detection in Thermal Infrared Drone Videos for Prevention of Wildlife Poaching.*

Radmanesh, Mohammadreza et al. (Apr. 2018). "Overview of Path-Planning and Obstacle Avoidance Algorithms for UAVs: A Comparative Study". In: *Un. Sys.* 06.02, pp. 95–118. ISSN: 2301-3850. DOI: 10.1142/S2301385018400022. (Visited on 05/06/2022).

Robicquet, Alexandre et al. (2016). "Learning Social Etiquette: Human Trajectory Understanding In Crowded Scenes". In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Vol. 9912. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 549–565. ISBN: 978-3-319-46483-1 978-3-319-46484-8. DOI: 10.1007/978-3-319-46484-8_33. (Visited on 06/08/2022).

Rockenbauer, Friedrich M. et al. (July 2021). "Dipper: A Dynamically Transitioning Aerial-Aquatic Unmanned Vehicle". In: *Proceedings of Robotics: Science and Systems XVII*. Robotics: Science and Systems Conference (RSS 2021). Robotics: Science and Systems Foundation, p. 48. ISBN: 978-0-9923747-7-8. DOI: 10.15607/RSS.2021.XVII.048. (Visited on 05/07/2022).

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (May 18, 2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation.* arXiv: 1505.04597 [cs]. URL: http://arxiv.org/abs/1505.04597 (visited on 07/10/2022).

Shelhamer, Evan, Jonathan Long, and Trevor Darrell (May 20, 2016). *Fully Convolutional Networks for Semantic Segmentation*. arXiv: 1605.06211 [cs]. URL: http://arxiv.org/abs/1605.06211 (visited on 07/10/2022).

Sibanda, Mbulisi et al. (Sept. 2021). "Application of Drone Technologies in Surface Water Resources Monitoring and Assessment: A Systematic Review of Progress, Challenges, and Opportunities in the Global South". In: *Drones* 5.3 (3), p. 84. ISSN: 2504-446X. DOI: 10.3390/drones5030084. (Visited on 03/17/2022).

Simonyan, Karen and Andrew Zisserman (Apr. 10, 2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv: 1409.1556 [cs]. URL: http://arxiv.org/abs/1409.1556 (visited on 07/09/2022).

Sliwka et al. (July 12, 2009). "Autonomous Robotic Boat of Ensieta". In: IRSC. Vol. 1.

Smolyanskiy, Nikolai et al. (Sept. 2017). "Toward Low-Flying Autonomous MAV Trail Navigation Using Deep Neural Networks for Environmental Awareness". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Vancouver, BC: IEEE, pp. 4241–4247. ISBN: 978-1-5386-2682-5. DOI: 10.1109/IROS.2017.8206285. (Visited on 05/07/2022).

Szegedy, Christian et al. (Aug. 23, 2016). *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. arXiv: 1602.07261 [cs]. URL: http://arxiv.org/abs/1602.07261 (visited on 07/10/2022).

Tan, Mingxing and Quoc V. Le (Sept. 11, 2020). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. arXiv: 1905.11946 [cs, stat]. URL: http://arxiv.org/abs/1905.11946 (visited on 07/09/2022).

Tetreault, Etienne, David Rancourt, and Alexis Lussier Desbiens (2020). "Active Vertical Takeoff of an Aquatic UAV". In: *IEEE Robot. Autom. Lett.*, pp. 1–1. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2020.3003296. (Visited on 05/07/2022).

The European Commission (Apr. 4, 2022). *Commission Implementing Regulation (EU) 2019/947 of 24 May 2019 on the Rules and Procedures for the Operation of Unmanned Aircraft*. URL: http://data.europa.eu/eli/reg_impl/2019/947/2022-04-04/eng (visited on 05/06/2022).

Thrun, Sebastian et al. (2006). "Stanley: The Robot That Won the DARPA Grand Challenge". In: *Journal of Field Robotics* 23.9, pp. 661–692. ISSN: 1556-4967. DOI: 10.1002/rob.20147. (Visited on 05/06/2022).

United States Air Force (2022). *MQ-1B Predator*. URL: https://www.af.mil/About-Us/Fact-Sheets/Display/Article/104469/mq-1b-predator/ (visited on 05/06/2022).

Vuuren, Marlice van et al. (June 5, 2019). "On the Effectiveness of UAS for Anti-Poaching in the African Arid Savanna". In: p. 660126. DOI: 10.1101/660126. (Visited on 03/17/2022).

Waldau, Leonard (2019). "Development of an Aquatic UAV Capable of Vertical Takeoff from Water". MA thesis. KTH, Naval Systems. 52 pp.

Welzl, Emo (May 10, 1985). "Constructing the Visibility Graph for N-Line Segments in O(N2) Time". In: *Information Processing Letters* 20.4, pp. 167–171. ISSN: 0020-0190. DOI: 10.1016/0020-0190(85)90044-4. (Visited on 07/12/2022).

Wolpert, D.H. and W.G. Macready (Apr. 1997). "No Free Lunch Theorems for Optimization". In: *IEEE Transactions on Evolutionary Computation* 1.1, pp. 67–82. ISSN: 1941-0026. DOI: 10.1109/4235.585893.

Wu, Ming et al. (2019). "Towards Accurate High Resolution Satellite Image Semantic Segmentation". In: *IEEE Access* 7, pp. 55609–55619. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2913442.

Wu, Wenjin et al. (June 2019). "PolSAR Image Semantic Segmentation Based on Deep Transfer Learning—Realizing Smooth Classification With Small Training Sets". In: *IEEE Geosci. Remote Sensing Lett.* 16.6, pp. 977–981. ISSN: 1545-598X, 1558-0571. DOI: 10.1109/LGRS.2018.2886559. (Visited on 06/08/2022).

Wurm, Michael et al. (Apr. 1, 2019). "Semantic Segmentation of Slums in Satellite Images Using Transfer Learning on Fully Convolutional Neural Networks". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 150, pp. 59–69. ISSN: 0924-2716. DOI: 10.1016/j.isprsjprs.2019.02.006. (Visited on 03/17/2022).

Yan, Fei, Yi-Sha Liu, and Ji-Zhong Xiao (Dec. 1, 2013). "Path Planning in Complex 3D Environments Using a Probabilistic Roadmap Method". In: *Int. J. Autom. Comput.* 10.6, pp. 525–533. ISSN: 1751-8520. DOI: 10.1007/s11633-013-0750-9. (Visited on 07/12/2022).

Yang, Liang et al. (June 2014). "A Literature Review of UAV 3D Path Planning". In: *Proceeding of the 11th World Congress on Intelligent Control and Automation.* Proceeding of the 11th World Congress on Intelligent Control and Automation, pp. 2376–2381. DOI: 10.1109/WCICA.2014.7053093.

Yu, Zhenyu, Xinping Bao, and Kenzo Nonami (2008). "Course Keeping Control of an Autonomous Boat Using Low Cost Sensors". In: *Journal of System Design and Dynamics* 2.1, pp. 389–400. DOI: 10.1299/jsdd.2.389.

Zang, Wenqian et al. (June 2012). "Investigating Small-Scale Water Pollution with UAV Remote Sensing Technology". In: *World Automation Congress 2012.* World Automation Congress 2012, pp. 1–4.

Zhao, Hengshuang et al. (Apr. 27, 2017). *Pyramid Scene Parsing Network*. arXiv: 1612.01105 [cs]. URL: http://arxiv.org/abs/1612.01105 (visited on 07/10/2022).

Zhou, Xin et al. (May 4, 2022). "Swarm of Micro Flying Robots in the Wild". In: *Sci. Robot.* 7.66, eabm5954. ISSN: 2470-9476. DOI: 10.1126/scirobotics.abm5954. (Visited on 05/07/2022).

Zhou, XueFei (Apr. 2018). "Understanding the Convolutional Neural Networks with Gradient Descent and Backpropagation". In: *J. Phys.: Conf. Ser.* 1004, p. 012028. ISSN: 1742-6596. DOI: 10.1088/1742-6596/1004/1/012028. (Visited on 08/02/2022).

Glossary

CNN Convolutional Neural Network. A type of neural network that uses sliding windows to recognize features in images.

GNSS Global Navigation Satellite System. Uses satellites to provide worldwide positional information for receivers on the ground. Examples are GPS, GLONASS, Beidou and Galileo.

NOTAM Notice to Airmen. A "notice containing information essential to personnel concerned with flight operations but not known far enough in advance to be publicized by other means" (Federal Aviation Administration 2022, §1).

RPAS Remote Piloted Aircraft System. A UAS where the pilot controls the plane externally. See also appendix A.

UAS Unoccupied Aircraft System. Conventionally, '*Unmanned* Aircraft System'. Includes not only the UAV, but also any other equipment used. See also appendix A.

UAV Unoccupied Aerial Vehicle. Conventionally, '*Unmanned* Aerial Vehicle'. See also appendix A.

VTOL Vertical Take-Off and Landing. VTOL aircraft can take off vertically and resume normal horizontal flight afterwards.

Appendix A

Drone Terminology

Choosing a term for a flying vehicle without a pilot is – perhaps surprisingly – not trivial.

Let's start with a description of the concept; what are we trying to name?

The vehicle of interest in this dissertation is both flying / aerial, unoccupied (no pilot, nor passengers), and autonomous.

Figure A.1 illustrates the relationship between these three categories. Lakehopper falls in the center set A. Examples of vehicles in set B are autonomous cars without passengers Thrun et al. (2006), and Bacha et al. (2008), autonomous boats Plumet et al. (2015), Sliwka et al. (2009), and Yu, Bao, and Nonami (2008)or technically Lakehopper while it is recharging on the water. Vehicles in set C are for example a full-size fixed-wing airplane using an autopilot system (e.g., 'Garmin Autonomí', 2022) and with passengers on board. Vehicles in category D are for example toy radio-controlled quadcopters (like the DJI Mavic, 2022) or military drones with a flight crew in an external command & control unit (like the Predator Drone, 2022).

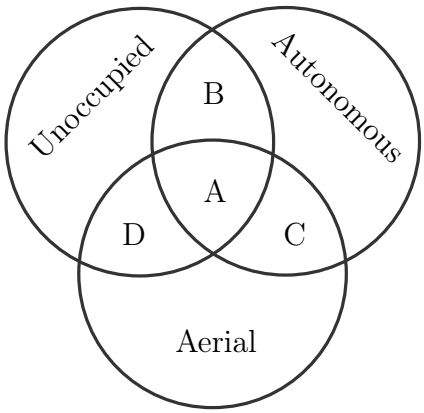

Figure A.1: Venn diagram of vehicle categories

With the concept that we want to name in mind (vehicles in set A), let's define and discuss some common terms.

Drone

The online Cambridge English Dictionary defines a drone as: "An aircraft that does not have a pilot but is controlled by someone on the ground, used especially for dropping bombs or for surveillance" (Cambridge English Dictionary 2022).

In a slight juxtaposition, the definition places a particular emphasis on military applications, but a picture next to the definition shows a toy quadcopter.

The online Merriam-Webster Dictionary provides a slightly more exact definition: "an uncrewed aircraft or ship guided by remote control or onboard computers" (Merriam-Webster Dictionary 2022).

As indicated by the Merriam-Webster Dictionary, in its widest sense, 'drone' can include any unoccupied vehicle, regardless of where it operates (in the air, around water, or anywhere else). The term also does not differentiate between remotely piloted and autonomous vehicles.

Laws and regulations generally avoid the term altogether. The United States Federal Aviation Administration's[1] regulation on "Small Unmanned Aircraft Systems" ('PART 107') for example, never uses the term once, instead using for example 'unmanned aircraft' and 'unmanned aircraft system' (UAS) (Federal Aviation Administration 2016). The same is true for the European Union's regulation on "the rules and procedures for the operation of unmanned aircraft"(The European Commission 2022).

In figure A.1, all vehicles in the sets within 'Unoccupied' are drones. Importantly, unoccupied vehicles that are not autonomous nor even aerial are also drones.

UAV

For an aerial drone, perhaps the most widely used term in the literature is 'Unmanned Aerial Vehicle' or UAV.

All UAVs fall into either the A or the D sets in figure A.1. While Lakehopper would fly autonomously most of the time (and thus fall into region A), it can still be piloted remotely by a human (and so fall into category D).

As indicated, the first letter of 'UAV' usually stands for 'Unmanned'. This term however has the unnecessary implication that the occupant(s) of the aircraft (if they were to exist) would be men (Joyce, Anderson, and Bartolo 2021).

As an alternative for 'manned', NASA has introduced the term 'crewed' (Lakdawalla 2022; Garber 2012). 'Uncrewed', like 'Unmanned' starts with a 'U'. However, it could imply that the vehicle has no crew. This is not necessarily true (e.g., the Predator Drone's command & control unit with a multi-person crew (United States Air Force 2022)).

The best alternative term seems to be 'Unoccupied', as it states exactly what we want to convey, and nothing more (Allan Poe 1845). As such, save for this

[1] https://www.faa.gov/

section, all occurrences of 'UAV' in this dissertation mean 'Unoccupied Aerial Vehicle'.

MAV

A 'Micro Aerial Vehicle' (MAV) is simply a miniature UAV. All MAVs are UAVs and fall into the same sets in figure A.1 (A & D).

UAS

An 'Unoccupied Aircraft System' includes not only the UAV, but also any other equipment used. This equipment is usually limited to the electronics to control the aircraft, like a radio remote. In a broad sense however, it could also include another UAV acting as a communication relay (Boyang Li et al. 2016).

An aircraft that is part of a UAS naturally falls into the same sets as a UAV does (A and D).

RPAS

A 'Remote Piloted Aircraft System' is a UAS that is specifically not autonomous. While this term could technically also include systems with aircraft carrying passengers [2], this is not usually what's referred to.

In figure A.1, the aircraft of an RPAS is part of set D.

Conclusion

'Drone' is likely the best term for the field of unoccupied vehicles in general, as many techniques are applicable to all types of drones, whether they are aquatic, aerial, or part of any other medium. Path planning, for example, is broadly applicable to aerial and ground vehicles alike, as are ground station communication technologies.

Nonetheless, a more specific term would always be preferable. 'UAV' fills this role, including all drones that are also (at least partly) aerial. This dissertation uses both terms interchangeably to refer to Lakehopper.

The most widely used term to specify that a UAV is also autonomous seems to simply be 'Autonomous UAV'. Any compound initialism like 'AUAV' seemingly hasn't caught on.

[2] In general this concept is usually referred to as 'teleguidance'.

Appendix B

Source Code Overview

All source code written in connection with this dissertation is available as an online Git repository via github.com/ubipo/lakehopper. All datasets used for development and machine learning can be obtained by contacting Pieter Fiers <pieter@pfiers.net>.

The planner component's backend was written in approximately 1,816 lines of Rust code, spread over 31 files (not including blank lines, comments, etc.). Its browser UI was written in approximately 535 lines of TypeScript code (and some HTML/CSS).

The vision component was written in approximately 1,331 lines of Python code.

The (abridged) structure of the Git repository is as follows:

```
Root of Git repository
├─ design/: design of drone; not applicable to this dissertation
├─ planner/: planner component
│  ├─ ui/: browser user interface (HTML/CSS/Typescript)
│  ├─ src/: backend websocket server / planner algorithms (Rust)
│  │  ├─ server/: websocket handler functionality
│  │  ├─ nav_graph/: visibility graph / shortest path / planner algorithms
│  │  ├─ mpi/: MultiPolygonIndex (aids with computational geometry problems)
│  │  ├─ main.rs: server entrypoint
│  │  ├─ droneguid.rs: Skeyes geozone restricted airspace client
│  │  ├─ crs.rs: coordinate reference system conversions
│  │  └─ ...: other source files
│  ├─ scripts/: miscellaneous scripts (Python)
│  ├─ data/: planner development dataset, not included (too large)
│  └─ ...: various auxiliary files
├─ vision/: vision component
│  ├─ src/: vision component source code (Python)
│  │  └─ lakehopper_semseg/: vision component Python module
│  │     ├─ preprocess/: preprocessing scripts
│  │     ├─ models/: model definitions
```

- `map/`: *map generation*
- `visualize.py`: *visualization helper functions*
- `train.ipynb`: *training notebook*
- `tpu.py`: *TPU connection functions*
- `dataset.py`: *dataset loading functions*
- `...`: *other source files*

- `scripts/`: *miscellaneous scripts (Python)*
- `datasets/`: *machine learning datasets, not included (too large)*
- `...`: *various auxiliary files*

- `...`: *various auxiliary files*