NTNU

Norwegian University of
Science and Technology

Detecting Location of Free Range Sheep

Using Unmanned Aerial Vehicles and
Forward Looking Infrared Images

Even Arneberg Rognlien
Tien Quoc Tran

Master of Science in Informatics
Submission date: June 2018
Supervisor: Svein-Olaf Hvasshovd, IDI

Norwegian University of Science and Technology
Department of Computer Science

Problem Description

Locating missing free range sheep at the end of the summer is a tedious process. Often, up to several hundred hours is spent out in the eld before all sheep are found. Previous work performed at IDI at NTNU has proven potential for using forward looking infrared (FLIR) images to locate sheep. As a rst step in creating an automatic sheep localization system using Unmanned Aerial Vehicle (UAV) and thermal imaging, the task will be to nd a way for collecting FLIR images of a dened area and nd a way to detect images containing sheep.

Abstract

In Norway, approximately two million sheep and lambs are released on open ranges to graze each summer. Locating the sheep after each grazing period is a tedious and challenging process. An experiment conducted by IDI at Norwegian University of Science and Technology (NTNU) shows that thermal imaging and Unmanned Aerial Vehicles (UAVs) can be an effective tool for locating sheep.

This thesis covers two modules. The first module is to develop an algorithm used for generating coverage paths for acquiring forward looking infrared (FLIR) images. The algorithm is capable of generating efficient routes for any user-defined area, with the possibility of omitting lakes to reduce flight time. Using a UAV with an attached FLIR camera and an onboard auto flight system, images can be acquired autonomously using the generated route.

The second module is to develop and test different image processing methods for detecting sheep in FLIR images. The primary focus was on developing an algorithm using classic image processing. This method was evaluated against two machine learning methods: Convolutional Neural Network (CNN) and RetinaNet. The results showed that classic image processing produced a better result than the state of the art machine learning methods. The algorithm was able to detect 83.3% of the sheep in the test set, which included images with a severe amount of background clutter. In areas with dense forest canopy, the algorithm was able to detect most animals with a low rate of false positives. It was found that the available dataset was too sparse to be used for machine learning based object detection.

Preface

This thesis was carried out as a Master's thesis at NTNU as part of the MSc programme in Informatics during the autumn 2017 and spring 2018. The work has been supervised by Svein-Olaf Hvasshovd.

We would like to thank our supervisor for the guidance, input and motivation throughout the thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The use of natural open pastures during summer is an essential part of the Norwegian sheep farming industry. Every year approximately two million sheep and lambs are released on open ranges for 90-100 days. During this period the sheep and lambs are allowed to graze freely, with some supervision. While the sheep are grazing outside the farmland, the grass within the farmland is allowed to grow freely, harvested and stored as food for the coming winter season. One of the main concerns with this practice is the dispersion of sheep and lambs across the open range, making it difficult for a farmer to locate and gather them [5, 28, 9].

1.1 Motivation

The sheep gathering process can often last one or more weeks [52], and the final phase is often the most time consuming and stressful part. The first 90% of the sheep usually flocks in large groups and are found after one to two weekends. The remaining 10% are often split into groups of 2-4 and scattered over a larger area [43]. Due to death, some sheep are never found, and the remaining sheep usually dies during the winter [41, 43].

A vital aid in sheep gathering is the bell which is mounted around the neck and worn by all mother sheep. A loud sound can be heard within a 1 km radius whenever the sheep are moving. The drawbacks with bells are that they can be torn off during the gazing period [41], and they are usually only worn by adult sheep. Lambs usually stay together with their mother who carries a bell, but if the lambs lose their mother, they become much harder to find.

Few digital aids are available to help with the sheep gathering process. One commercially available system is Telespor [22]. Each sheep wears a collar that uses the Global Positioning System (GPS) to calculate its position and communicates with the farmer using the

Global System for Mobile Communications (GSM). Looking at the coverage maps from the three largest cellular network providers in Norway, the mountain landscapes which the sheep often seek has limited or no coverage at all [23, 21, 7]. A High Frequnecy (HF) finder must be bought separately for use in areas without GSM coverage. Besides this, the common problems associated with collars that was mentioned above, is still persistent.

An industry which has benefited from the development of battery and sensor technology is Unmanned Aerial Vehicles (UAVs), commonly known as drones. Some UAVs are capable of flying with high speed for several hours and can be used for searching large areas in short time [43]. Using one of several available autopilot systems [27, 17, 11], the UAVs can be programmed to follow a pre-defined route specified by GPS coordinates.

In a recent experiment conducted by people from the Department of Electronic Systems at Norwegian University of Science and Technology (NTNU) in Trondheim, a UAV, and a thermal infrared camera were used to capture images of a flock of sheep from different altitudes [41]. The experiment showed that sheep was easily identified in the thermal infrared images, even from higher altitudes (50-100 meters), and demonstrated a high potential for using UAVs and thermal imaging to search for sheep. However, to spot the sheep, a manual human inspection was required, and for large areas with potentially hours of video recordings, this method is highly time-consuming and prone to human errors. Because of this, it was suggested to develop a system that automatically analyzes the images and detects potential sheep. A survey of the advancements in computer vision shows that image processing and machine learning can detect targets with a high accuracy [29]. It was also suggested to develop a system for planning the flight, as existing freely available tools were not optimal for this specific use case.

1.2 Objectives

Based on the observations made in the previous section it was concluded that the current tools used in the sheep gathering process are insufficient. Digital tracking system struggle with poor performance within areas with poor or no GSM coverage, while bells require a considerable amount of manual labor concerning organizing and executing a search party. The primary objective of this thesis was focused on overcoming these shortcomings by developing a new tool based on the concept of using UAV and thermal imaging to search and locate sheep. Towards reaching this goal the following research questions were formed;

RQ 1: How can UAVs be used to survey areas for missing sheep? We wanted to find a method for efficiently acquiring thermal images of an area specified by a user. This includes the selection of equipment and finding a way to generate search routes that can be automatically followed by an autopilot system.

RQ 2: How can computer vision be used to detect sheep in FLIR images? We wanted to find out how computer vision can be used to detect sheep in the images surveyed by the UAV automatically, and if it is accurate enough to be used in a real use case.

1.3 Research method

The research conducted in this thesis is based on principles from Design Science [42]. Design Science is a research paradigm with the purpose to establish credibility for research done in the Information Systems (IS) field. Design science research provides the guidelines for understanding, designing and building new innovative artifacts, to improve the environment of a specific domain. Figure 1.1 shows an overview of the design science research cycles. The following three design science research cycles were used in this research project: the relevance cycle, the rigor cycle, and the design cycle.

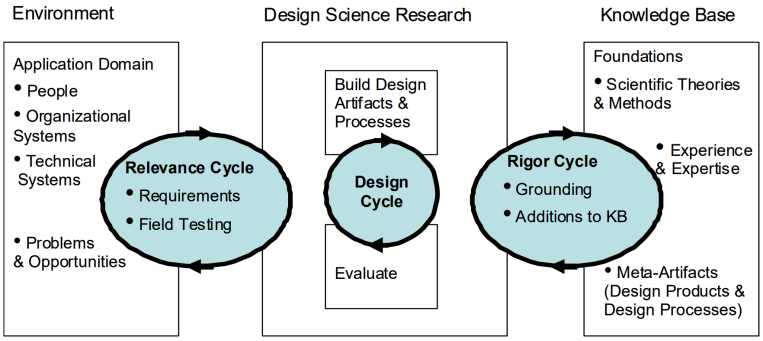


Figure 1.1: Design science research cycles [42].

The relevance cycle initiates the design science research by identifying opportunities and problems within the application environment. This cycle also defines different criteria/requirements that are used to measure the improvement of the application environment, as a result of the research. In our case, the environment consists of the sheep gathering process, image acquisition, and the process of analyzing the thermal images.

The rigor cycle brings past knowledge, experiences, and expertise into the research project and ensures that the result of the research extends the existing knowledge base. Our literature review gave us an overview of the existing knowledge base, which is used and referenced throughout this thesis. At the end of the project, our work brought new research contributions to the knowledge base.

In the design cycle, different design alternatives are generated and evaluated against the requirements which are provided by the relevance cycle. This is iterated until a satisfactory design is achieved.

1.4 Contribution

The first contribution is a method for automatically generating efficient flight routes used for acquiring images within a defined geographic boundary. A basic software for generating search routes has been developed.

The second contribution of this thesis, which is the primary focus, is a system for automatic detection of sheep in aerial thermal images. A prototype was developed as a proof of concept.

Finally, the third contribution is this thesis itself, where we present and discuss relevant previous work, background theory, our implementation and our contribution to the problem domain.

1.5 Thesis Outline

The thesis is structured as follows.

In Chapter 2, *Previous Work*, the initial knowledge base is presented. This includes other research projects that address the same, or parts of the problem domain that this thesis does. This is not limited to sheep gathering, but also includes research which addresses similar problems in regards to computer vision and route generation.

Chapter 3, *Image Acquisition*, addresses *RQ 1*, and presents a method for generating efficient coverage paths for any area.

Chapter 4, *Automatic Sheep Detection*, is the core of the project, and answers *RQ 2*. This chapter describes a method for automatic detection of sheep in thermal images, primarily using classical computer vision, but machine learning is also covered.

In Chapter 5, *Implementation*, the technological choices that was made when implementing the algorithms are accounted for. This includes the choice of programming languages, frameworks and third-party libraries.

Chapter 6, *Result and Discussion*, presents and discuss the results of the research.

Chapter 7, *Future work*, summarize what was not covered during the work on this thesis, by suggesting different topics that should be further explored to add more knowledge to the problem domain.

In Chapter 8, *Conclusion*, a short summary of the thesis is given, where the research questions are answered.

Chapter 2

Previous Work

This chapter presents previous works related to sheep localization using UAV and FLIR imaging. However, since this is still a relatively new and uncharted field of research, not many directly related previous works were found. Because of this, we had to search for other work that could be adapted to our problem domain. The first sections will present previous work directly related to our problem, then previous work that is not directly related but can be adapted to our problem, is presented.

2.1 The use of UAV in Sheep Localization

The motivation and foundation for this thesis is based on the previous work mentioned in Section 1.1 where an experiment was conducted, testing whether or not UAVs and thermal cameras could be used to locate sheep. The experiment used a small group of sheep dispersed across a farm located in Oppdal, Norway. A thermal camera mounted on a quadcopter was used to capture footage of a group of sheep from different altitudes. This experiment resulted in four videos; two from a 50m height above ground, one from 60m and one from 100m. Although the videos contained a significant amount of background clutter due by sun radiation, sheep could easily be spotted by the camera from a high altitude [40].

One paper directly related to sheep gathering and UAVs, describes a prototype developed in 2013 at Harper Adams University in England [36]. The prototype only served as a proof of concept and was not capable for use in a real-world application. First, a multirotor UAV was programmed to automatically detect and follow a red circular target. Using WiFi, live video from the color imaging camera on the UAV was sent to the base station (laptop), and the base station could send control inputs back to the UAV. Image analysis was run on the base station to detect the red spot, and control inputs were sent to the UAV attempting to keep the spot centered in the camera view. For developing and testing a

sheep tracking algorithm, a monochrome camera was mounted at a static point, 2-3 meters above a flock of sheep. A live video stream was sent to a base station on the ground where the image analysis was performed. Objects were extracted from the images using thresholding, where every object brighter than a certain level was further processed. The area of each object was measured, and if the area were within a particular interval, the object was classified as a sheep. The output of the algorithm was the sheep coordinates in the image, found by calculating the center of gravity of the object's contours. The results showed that the method had difficulty distinguishing sheep from other objects with the same color and area. One of the methods that were suggested to resolve this problem was to detect arcs in the outline of the sheep, such as the back and the head. In our scenario, there are difficulties with this method. When the camera is mounted significantly higher, and the camera resolution is low, which it is in our case, the level of details is too low for this solution. Also, using a regular monochrome camera or color camera relies on all sheep to have a natural bright color, which is not always the case.

2.2 Wildlife Population Monitoring

Several research papers were found on the use of UAVs for wildlife monitoring [32, 46, 51]. Some of the papers rely on manual human inspection, while some address automatic recognition. The most relevant papers are described below.

In a paper from 2017 [51] a fixed-wing UAV, thermal imagery and computer vision were used to estimate seal population at different islands in eastern Canada. The *senseFly eBee*, a commercially available electric fixed wing UAV with an integrated thermal infrared camera, was used. The camera is self-calibrating and was measured to be accurate within 1°C. The UAV was equipped with an auto flight system and followed a pre-programmed route, generated by the *eMotion 3* software. After the flight, the images were first analyzed manually, where each seal was counted and classified as either adult or pup. Then an automated detection was run using the thermal infrared images. The algorithm first used a threshold to find all objects warmer than 9°C. In some cases, seals were tightly clustered in groups, giving some objects an aggregated irregular shape. Common for the aggregations was that they had a higher degree of concavity than individuals. The minimum convex hull area was found and compared with the original object's area to decide whether a selected object was an individual seal or an aggregation of seals. The aggregations were then separated using edge detection on the thermal image, where the detected edges were used to separate the seals. Finally, each individual object was classified as adult or pup, based on average measured temperature and area. Figure 2.1 shows an overview of the process. The results showed that the algorithm was able to detect between 95-98% of the human estimates, and the algorithm found even a few seals that were missed by the human analysis. The image processing algorithm worked out in this thesis will have a similar pipeline. However, adult sheep and lambs will not be differentiated.

Another article which addresses wildlife monitoring is [46] from 2017, where multiple species were of interest. Unlike [51] which was conducted on islands without any high vegetation, this survey was conducted on a landscape where forest covered over 80% of

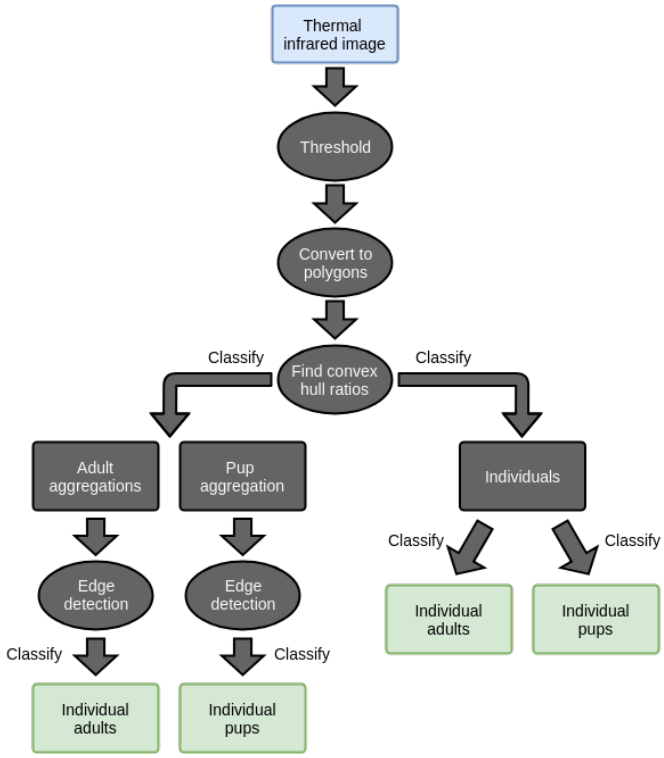


Figure 2.1: Overview of the seal detection algorithm [51]

the area. Although our dataset does not contain any images of forest covered areas, it is still a highly relevant topic for our case. The forest was a combination of coniferous and deciduous forest of various age and degrees of canopy cover. Since the flights were done in April, the trees were leafless. A coverage path was generated using the *Horizon*^{MP}[1] software and flown by the UAV's auto-flight system. Three different species were searched for; red deer, wild boar and roe deer. The project relied on human analysis of the images, which sometimes was challenging due to the low amount of details, especially for small animals like the roe deer. However, the paper suggested that the image information could be sufficient for recognition by machine learning techniques. Using other characteristics of the thermal signature, such as the distribution of pixel temperatures, was also suggested for classification.

The animal sighting rate varied between the different habitats. The number of detected animals was highest in the deciduous forests, where the animals in most cases were possible to spot between the leafless branches. In the coniferous forests, the ground surface was significantly obscured by the tree crowns. However, in areas without full canopy cover, several animals standing under the tree crowns being obscured, became visible to the camera in the next frames, as the UAV was passing. Flying in closer located trajectories would

[1] `https://www.micropilot.com/products-horizonmp.htm`

increase the detection rate in forests. For areas with full coniferous canopy cover, it was suggested to not use UAVs. Video samples of different sightings in different habitats was available as supplemental material [45]. This allowed us to test the developed algorithm on different forest environments, in addition to the video samples provided by NTNU.

2.3 Object Detection

In [31] a method for automatic target detection and tracking in FLIR image sequences is proposed. In this method, connected morphological operators are used to extract and track targets of interest based on size, connectivity and motion criteria. This is done using spatial intraframe and temporal interframe information. In the first step, a sequence of images is filtered separately on a frame by frame basis, removing background clutter and enhancing targets of interest, resulting in a new set of images. The next step uses the new image sequence to perform a motion analysis based on the spatiotemporal changes in each image in the sequence. This method was tested on real FLIR data, which was characterized by a substantial target and clutter variability. The result showed that it was able to detect the targets of interest and eliminating background clutter with high accuracy.

In [48] a method for detecting and extracting blobs in infrared images was proposed. This method uses an edge detector and a size criteria to detect and extract objects that have a shape similar to a blob.

These two methods have been adapted to the problem and are used in the proposed sheep detection algorithm.

Two methods based on supervised machine learning, are Convolutional Neural Network (CNN) and RetinaNet. These can be trained to detect or classify certain types of objects, based on labeled images. CNN is used to classify images. In [34] seven state of the art CNNs were tested on the MNIST handwritten digits dataset. The result proved that the CNNs was capable of classifying handwritten digits accurately. In [47], RetinaNet, an automatic object detection method based on CNN was proposed. The results showed that it was able to detect objects and their location in an image accurately.

In this project, we wanted to test and find out if a CNN can be used as a classification step in the worked out algorithm for sheep detection. We also wanted to test if RetinaNet can replace the whole algorithm. The dataset provided for this thesis was limited. Because of this, these methods were not part of our main focus, and are only described on a high level.

2.4 Path Planning

In this section, previous work related to UAVs and Coverage Path Planning (CPP) is reviewed. CPP is the task of finding a path that covers an entire bounded area. We wanted to be able to generate a flight path, also called flight mission, that covers a predefined

closed area. From this point onward, the term waypoint is used about a point in the flight route.

2.4.1 Ground Control Station

Ground Control Stations (GCSs) are computer programs that are used to communicate and control UAVs. These programs run on a computer located on the ground and are used to retrieve and display real-time data from the UAVs. It is also used to send orders to the UAV, like updating a flight mission. These programs also have the tools for creating and editing flight missions. Two of the most feature rich and popular open source GCSs were tested; APM Planner 2.0[1] and Mission Planner[12]. During our test we found out that the former only supported manual creation of flight mission, meaning that the user had to define each waypoint manually. The latter had an own function for creating a flight mission for surveying purposes. Figure 2.2 shows an example of the process and results of generating a flight mission using the Mission Planner GCS program. Figure 2.2a shows the first step which is used to define an area that the user wants to survey. The area is marked using a polygon. Figure 2.2b shows the second step which is to generate the flight path. In this step, the user only needed to configure different options, camera type, viewing angles, altitude and etc, and the route would be generated automatically. From the results shown in Figure 2.2b it was concluded that the route generated is not optimal for our use case. There are two reasons for this; (1) The generated path is not optimal and covers large areas that are not marked by the polygon. (2) The generated path does not consider lakes.

2.4.2 UAV and CPP

A paper from 2013 [53] propose a method for surveying aerial images used for 3D terrain reconstruction. This method aims to capture overlapping images covering an entire pre-defined area. A georeferenced polygon is used to delimit an area of interest. A coverage path is generated for the polygon. For convex polygons, a simple algorithm is used, while concave polygons require an additional step that decomposes the polygon into smaller convex sub-polygons. Using the same algorithm that is used for convex polygons, a route is generated for each sub-polygon.

[37] presents an algorithm for decomposing concave polygons into multiple convex sub-polygons. An interesting feature is that it can decompose polygons that contain holes, which are defined as polygons inside the main polygon. The paper states that the problem of decomposing a polygon with holes into a minimum number of convex pieces is known to be NP-hard, meaning that it is not considered to be solvable in polynomial time. The algorithm presented does not guarantee a minimum number of convex pieces, but is said to be computationally quick. By feeding the algorithm with lake polygons, defined as "holes", it provides a set of smaller polygons that can be searched individually to avoid the lakes.

(a) Survey area marked as a polygon.

(b) Generated flight path (flight mission).

Figure 2.2: Generating flight path using Mission Planner

2.5 Summary

The presented previous work is part of the existing knowledge base. The algorithms developed in this thesis is mainly based on this work, and are used to update the current knowledge base with new discoveries and results.

Chapter 3

Image Acquisition

This chapter presents a method for collecting thermal image data for a defined area, and the theory behind. The primary focus is the generation of a search route.

3.1 Setup

The most vital components in the image acquisition process are the camera and the UAV.

3.1.1 Camera

The camera is mounted on the UAV and captures images perpendicular to the ground. A FLIR camera was chosen because of the following reasons; (1) Thermal imaging does only depict the emitted surface temperature of objects. In practice, this means that the camera can search for sheep in both illuminated and unilluminated environments. (2) The color of the sheep is irrelevant, and all sheep appear in images only based on their temperature. This means that sheep will remain visible, even when their colors blend into the surroundings. The image processing algorithm, which is described in Chapter 4, relies on thermal infrared images.

3.1.2 Fixed Wing or Rotary Wing

From this point onward, the term UAV is defined as a pilotless aircraft controlled by an autopilot system. There exists many different types of UAVs with different configurations, each one having their strengths and weaknesses. Choosing the right type of UAV is hence very important for optimal results. The two most relevant types of UAVs for our use case

are called rotary-wing and fixed-wing aircraft. The main difference between the two types is how they generate lift. In order for a wing to generate lift, a certain amount of airflow above and below the wing is required. Rotary-wing aircraft, such as helicopters, achieves this by rotating their wings. Fixed-wing aircraft, such as commercial passenger airplanes, use forward motion to provide the required airflow. This fundamental difference has a significant impact on how the two types operate and are used in practice. When comparing the two types, rotary-wing UAVs are agiler and capable of performing advanced maneuvers such as vertical takeoff and landing, hover and sharp turns. Fixed-wing UAVs are dependent on a forward airspeed to generate lift, resulting in reduced agility and maneuverability. However fixed-wing UAVs are usually the most power efficient. Rotary-wing UAVs uses a significant amount of the engine power to actively keep the aircraft airborne, whereas on the fixed-wing UAVs the engine power is used to maintain forward airspeed while the wings passively ensure lift. This gives the fixed-wing a considerably longer flight time, as well as a higher cruising speed. It can also carry heavy payload using less energy than a typical rotary wing. This makes the fixed-wing UAV an excellent choice for surveying large areas and was considered the most suitable for our application.

(a) Airplane

(b) Quadcopter

Figure 3.1: Examples of fixed wing and rotary wing UAV. Both images are from the UAV-Lab at NTNU [15].

3.2 Flight Planning

A complete flight plan is generated based on the area the sheep farmer wants to search. The flight plan solves the well known CPP problem by ensuring no area is left unseen by the camera. In [38], CPP algorithms are classified as either offline or online. Offline algorithms rely on static information known before the mission, while online algorithms utilize real-time sensor measurements to decide the path. Because the search is performed at an altitude well free of physical obstacles such as trees an power lines, the environment is considered as static. The route is therefore generated before the flight starts and uploaded to the UAV's autopilot.

Figure 3.2 gives an overview of the proposed route generation algorithm. First, the desired search area is defined by the user. If the area is convex, a coverage path is generated and output. If the search area is concave, or it contains holes, decomposition is performed.

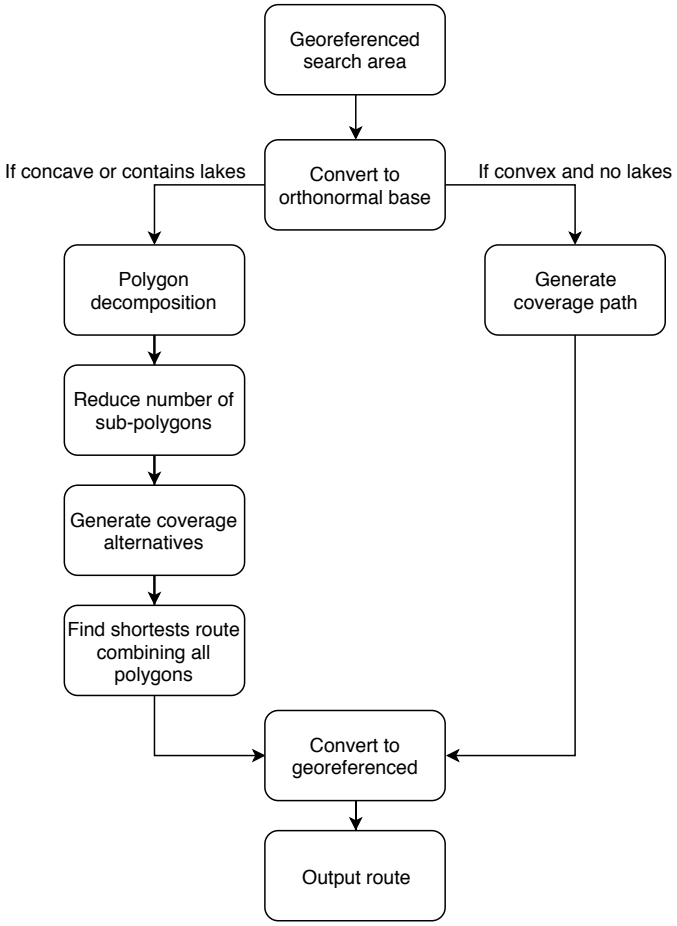


Figure 3.2: Route generation algorithm overview.

After the decomposition, an attempt is made to reduce the number of polygons by merging adjacent ones, when possible. For each resulting polygon a coverage path is generated, and finally, a single route is found connecting and covering all polygons. The algorithm is described in greater detail in the following sections.

3.2.1 Search Area

A geographically referenced polygon defines the bounds of the search area. A polygon is a geometric figure formed by a set of vertices and edges. In 2D space, a vertex is a point defined by an x and y coordinate. An edge, or a segment, is a straight line connecting two vertices. In a simple polygon, the line segments form a chain closing in a loop with no crossing line segments. In a geographic coordinate system, instead of x and y a vertex is defined by degrees longitude and latitude. The algorithms proposed in this paper requires that shapes are described in an orthonormal base, which is not the case with longitude and latitude since the degree of change in longitude shrinks as the latitude increases. See Figure 3.3. To solve this problem, the formula described by [50] is used to convert the longitude and latitude coordinates to orthonormal base:

$$\begin{aligned} \triangle y &= 110574 + |\varphi| \cdot 12.4444[m] \\ \triangle x &= 111304 \cdot cos(\varphi) + 81.2583[m] \end{aligned} \tag{3.1}$$

φ is the middle latitude of the area to be searched. This is found by:

$$\varphi = \frac{\varphi_{max} + \varphi_{min}}{2} \tag{3.2}$$

where φ_{max} is the latitude of the northernmost vertex, and φ_{min} is the latitude of the southernmost vertex of the polygon. The number 110574 refers to the distance of one degree of latitude in meters at the equator, while 111304 refers to the distance of one degree of longitude at the equator. The final conversion to orthonormal based coordinates is found by multiplying all longitude and latitude values with $\triangle x$ and $\triangle y$, respectively. The resulting x and y describes the coordinates in meters instead of degrees. The same $\triangle x$ and $\triangle y$ are used to convert the generated orthonormal based points back to latitude and longitude when the route generation is complete.

3.2.2 Coverage Path Generation

As mentioned in Chapter 2, the coverage path algorithm used in this paper is based on the algorithm described in [53]. The first step of the algorithm is to find the distance, d, separating each row. See Figure 3.4. For this, the width of the camera footprint, w, and the desired vertical overlap in the captured images are required. Using an overlap during the image acquisition, the chance of missing areas due to slight deviations from the route caused by, for example, sudden wind changes or turbulence is lowered. The camera

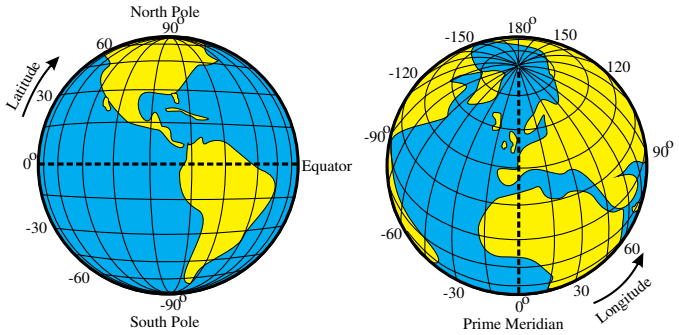


Figure 3.3: Illustration of geographic latitude and longitude of the earth [35]

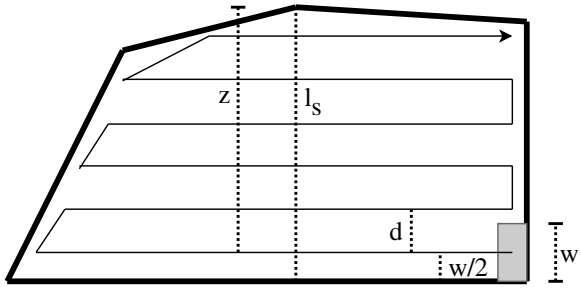


Figure 3.4: The values used for generating the coverage path

footprint is calculated using the camera's field of view and the height of the UAV. Given the camera's horizontal field of view, θ, and the height above the ground, h, the following equation is used to find the horizontal distance w covered by the captured image:

$$w = tan(\frac{\theta}{2}) \cdot h \cdot 2 \tag{3.3}$$

As an example, the camera that was used to record the primary dataset used in this thesis has a field of view of $45° \times 37°$ and a resolution of 640×512 pixels. The images were taken from approximately 50 meters above ground. Based on this a theoretical camera footprint of 41.42×33.45 meters is calculated. Once the width w of the camera footprint and the desired overlap v is known, the distance d between each line is found using

$$d = w \cdot (1 - v) \tag{3.4}$$

The exception is the first line which, as shown in Figure 3.4, has the distance of $w/2$ from the starting edge of the polygon. The number of turns depends on d, w and l_s, where l_s is the length of the so-called optimal line sweep direction. An intermediate value z is found by $z = l_s - w/2$, which is the distance from the first row to the end of l_s. Then $[z/d]$ is the

number of rows. For each segment in the coverage path, two turning points are needed. The total number of turns, n is given by

$$n = \begin{cases} 2 \cdot [z/d], & \text{if } z \mod d \leq w/2 \\ 2 \cdot ([z/d] + 1), & \text{if } z \mod d > w/2 \end{cases} \tag{3.5}$$

Since d is fixed for a specific ground sampling resolution, it is clear that n depends on the value of z. z, in turn, depends on the value of l_s. Therefore, in order to get the minimum amount of turns, the shortest possible l_s must be found. This is the optimal line sweep direction.

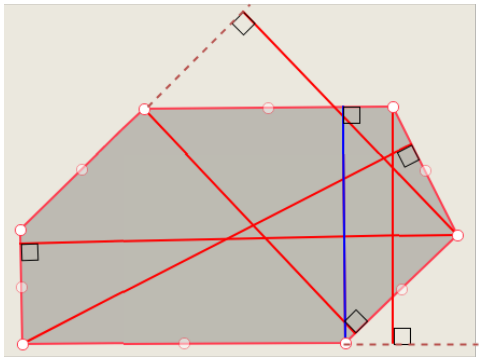


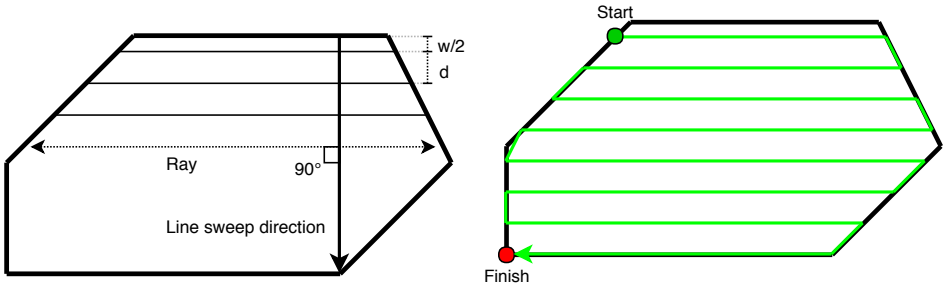
Figure 3.5: Longest distances from each edge to a polygon vertex. The blue line (shortest) is the optimal line sweep direction.

For a given polygon, the optimal line sweep direction is the shortest line that goes across the polygon. This gives the lowest l_s, resulting in the least number of turns. First, all possible line sweep directions are found; for each edge of the search area, a line is drawn from the edge to the vertex that is located furthest away from that edge. Figure 3.5 shows an example, where all the red lines are the possible line sweep directions. When this has been done for all edges, the length of each line sweep direction is compared and the shortest line is the optimal line sweep direction. For the polygon shown in the example image, the blue line is selected as the optimal sweep direction.

Once the optimal line sweep direction is found, the coverage path is generated. This process is illustrated in Figure 3.6. For each row, an infinitely long "ray" is cast in each direction, perpendicular to the line sweep direction. Where the ray intersects the polygon, a point is created. Finally, a route is created by connecting the points in the same way as shown in Figure 3.6b.

3.2.3 Handling Concave Search Areas

As opposed to a convex polygon, a concave polygon is a polygon that is hollowed inward. More specifically, in a concave polygon, at least one of the internal angles are between



(a) Generation of sweep route along the optimal line sweep direction

(b) Generated coverage path

Figure 3.6: Coverage path generation

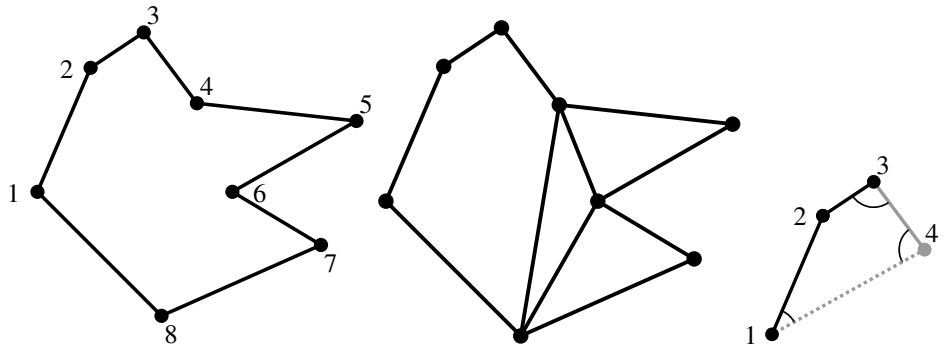
180 and 360 degrees. As described above, when generating coverage paths for a concave polygon, there is a chance that the path is interrupted by the polygon's edges. This can be handled in multiple ways. In some cases, choosing a different line sweep direction is sufficient. However, this is not always the best solution as it might lead to an unnecessarily high number of extra turns. A different option is to decompose the area into smaller convex polygons, which are then covered individually. Using this method an optimal sweep direction can be found for each sub-polygon.

In addition to simple search areas, this thesis also examines omission of lakes. In Section 2.4.2, a brief introduction was given to the algorithm described in [37], which supports decomposition of polygons containing holes. In our case, a hole can represent a lake that should be omitted. To start with, decomposition of polygons without holes is explained, then decomposition of polygons with holes.

Decomposing polygons without holes

The purpose of the algorithm is to subdivide the concave polygon into separate convex polygons. Figure 3.7a shows a concave polygon, P, defined by eight vertices, v_1 to v_8. The polygon is decomposed using the following method.

A list L is used to store the vertices of a potential new convex polygon. First, the very first line segment of P is added to this list, i.e. v_1 and v_2. The last vertex that was added is called v_i. The next vertex in the polygon, v_{i+1}, is then tentatively added to L. Let $ang(a, b, c)$ be the angle between line segment ba and line segment bc. To check that the new point, v_{i+1}, does not cause L to form a concave polygon, the angles $ang(v_{i-1}, v_i, v_{i+1})$, $ang(v_i, v_{i+1}, v_1)$ and $ang(v_{i+1}, v_1, v_2)$ are measured. See Figure 3.7c. If all three angles are less than or equal to $180°$, v_{i+1} is kept in L. This is repeated until all vertices of P are in L or the angle-test fails. In the first case, the entire polygon is already convex, and the algorithm can stop. In the latter case, a diagonal line is created between the last vertex in L, and the first vertex, v_1, creating a closed convex polygon. A check is made to see if the new polygon surrounds any vertices from $P \setminus L$ (i.e all points



(a) A concave polygon **(b)** Convex decomposition **(c)** Trying to add v_4 to the L-polygon.

Figure 3.7: Example of decomposition

in P that are not in L). If a vertex, v, is found inside the bounds of the polygon generated by L, the last added vertex, v_k, is removed from L together with all other the vertices of L that are located in the half-plane generated by v_1v, that contains v_k. This is repeated with the new L until no vertex is inside the polygon generated by L.

When no more points can be added in the clockwise direction of P, the algorithm attempts to add more points to L in the counterclockwise order. The process is similar to the steps described above, but L is used backward. In the example polygon (Figure 3.7a), v_8 is first tentatively added at the beginning of L. In the three tests v_{i+1} will be v_8, v_i will be v_1 and v_{i-1} will be v_2. If v_8 is successfully added, the same steps are done with v_7 and so on. After this, if L contains more than two vertices and at least one of the last and first vertices of L is a notch, the polygon is saved. A notch is a point where the inner angle is more than $180°$. If not, a polygon is not generated in this call. Next time the method is called, the initial vertex (previously v_1), will be the first notch (in clockwise order) that is found starting from the last vertex of the last convex polygon generated. If no polygon is generated it go to the next notch and try to use that as the initial vertex.

Figure 3.7c shows the vertices of L, in black, after v_3 has been added. The figure also shows the three angles that are tested before v_4 is added. In this case, the addition of v_4 does not cause L to be concave, thus can v_4 be added. Figure 3.7b shows the final decomposition of P.

Decomposing polygons with holes

Consider a polygon, P, containing one or more hole polygons, $H_1, ..., H_h$. To generate sweep paths that omit these holes, the polygon is decomposed into smaller polygons surrounding the holes, where each sub-polygon will be covered separately. To accomplish this, the holes are first "absorbed" by P, resulting in one long polygon consisting of all vertices in P and all vertices of each hole. The algorithm for this is described below.

The decomposition of P works similar to the algorithm for polygons without holes. A list, Q, contains the remaining points of P still to be decomposed. Unlike the previously described algorithm, every time a new diagonal, $d = v_i v_f$, is about to be created, a check is made to see if any holes intersect the diagonal, or if the new polygon that it forms (L) contains any holes. If it intersects with a hole, a diagonal is drawn from v_i to the vertex closest to v_i, on the line segment that it intersects. The new diagonal is $d' = v_i v'_f$ and H' is the hole containing v'_f. Using d' as a "bridge" between the border polygon, P, and H', the vertices from H' is absorbed by Q and becomes part of the border polygon. This works by inserting each point from H' starting from v'_f, in a counterclockwise order, after v_i.

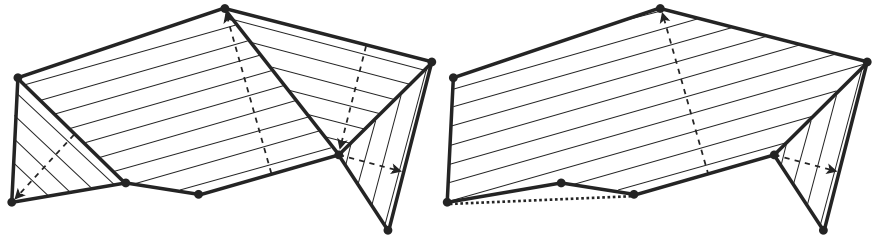
3.2.4 Reducing the Number of Decompositions

The polygon to be covered does not necessarily have to be convex, as long as the generated sweep path is not interrupted. With that in mind, a post-processing step is proposed for the decomposition algorithm described above. The purpose of the algorithm is to reduce the number of sub-polygons by merging adjacent polygons whenever possible. By doing this, the number of turns and transition segments between the polygons are reduced. The algorithm we propose works as follows.

For each polygon, all adjacent polygons are found. In this context, adjacent polygons have at least one edge in common. For a given polygon, the algorithm attempts to merge one of the adjacent polygons. To check if two polygons can be merged, a temporary merge is created first. Using the convex hull, instead of the polygon itself, of the temporary polygon the optimal line sweep direction for the resulting polygon is found as described in Section 3.2.2. If one of the generated rays (see Section 3.2.2) happens to interest with more than two edges, it means that the path has been interrupted. If this is the case, the merge is dropped. If not, the two polygons that were merged are removed from the collection of polygons and substituted by the new polygon. The new polygon is then checked against all its adjacent polygons for new possible merges, and if none is possible, the algorithm proceeds with the next polygon in the collection. This is repeated until there are no more possible merges in the collection of polygons. Figure 3.8a shows the output polygons output from the convex decomposition algorithm described in Section 3.2.3. The dashed arrows illustrate the line sweep direction for the sweep path. Figure 3.8b shows the result of the polygon merging. The thick dotted line illustrates the edge of the minimum convex hull of the largest polygon, which was used to find the optimal line sweep direction (dashed arrow) after the merge. Notice that in this example, the number of turns is reduced from 50 to 28.

3.2.5 Coverage Path for Multiple Polygons

When the UAV must visit multiple polygons, which is the case after convex decomposition, it is important to find a route that gives short transitions between each polygon. A transition is a line going from the end of one coverage path to the start of the next coverage path.



(a) Convex decomposition

(b) Convex decomposition with merged polygons

Figure 3.8: Example of polygon merging

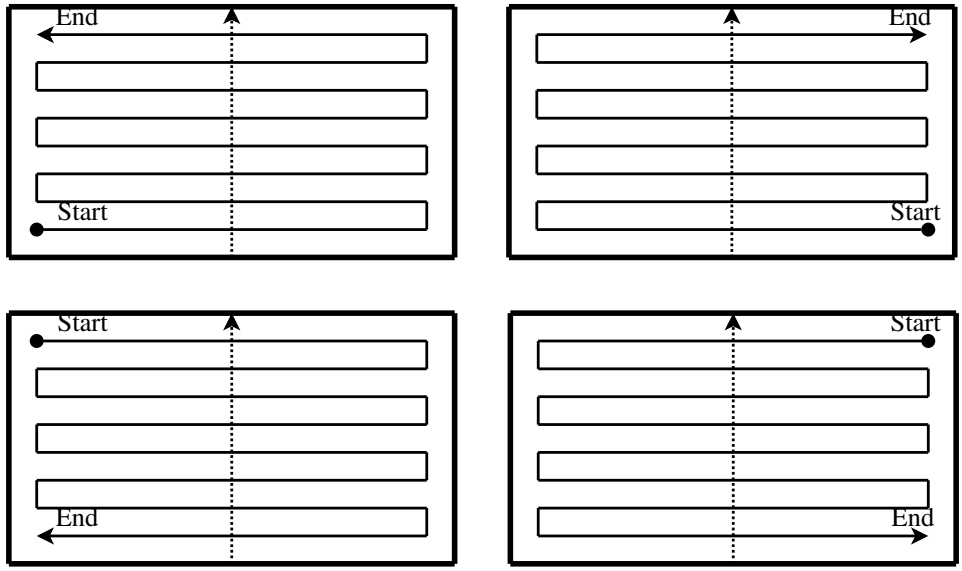


Figure 3.9: The four different alternatives to cover a region. The dashed line indicates the optimal line sweep direction. Top left: clockwise and current line sweep direction. Top right: counterclockwise and current line sweep direction. Bottom left: clockwise and opposite line sweep direction. Bottom right: counterclockwise and opposite line sweep direction.

According to [53], there are four different alternatives for flying the coverage path for a single polygon. These alternatives are based on two criteria. The first criteria is whether the current or the opposite direction of the optimal line sweep direction is used. The second criteria is the way that the coverage path is constructed: clockwise or counterclockwise. In the case of clockwise, the first turn made is to the right, while in case of counterclockwise the first turn is to the left. This results in four possible entry and exit points for each polygon. Figure 3.9 shows the four alternatives for a simple rectangular area.

In order to minimize the total distance of a path, that visits all sub-polygons, the algorithm in [53] generates all possible combinations of visit orders and coverage alternatives, and

selects the combination that generates the shortest complete path covering the entire area. In a set of n polygons there are $\frac{(n-1)!}{2} \cdot 4^n$ possible permutations. As seen later in this thesis, the polygon decomposition can produce a large number of sub-polygons, especially when used with holes. Because of this, this algorithm does not scale well enough to be usable for the potentially high number of polygons in the use case of this thesis.

The algorithm proposed in this thesis is a simple, greedy, "first and best"-approach. First, all four combinations of starting points and end points are found for each polygon. The algorithm checks two and two polygons and their four coverage alternatives and finds the combination that gives the shortest transition from the first polygon to the next. The starting point of the coverage alternative in the first polygon is selected as the starting point of the route. From the second polygon, at the endpoint in the selected coverage alternative, the algorithm checks all the other polygons that have not been visited yet and their coverage alternatives, to find the polygon which should be visited next. The polygon that has a coverage alternative with the closest starting point to the ending point in the previously visited polygon is selected. This is repeated until all polygons have been visited.

3.2.6 Exporting Route

When the route generation is done, all generated waypoints are converted to geo-referenced latitude and longitude points, using the same $\triangle x$ and $\triangle y$ that was found in the first step, described in Section 3.2.1. The resulting waypoints can then be exported to a file format that can be uploaded to the UAV's autopilot system.

Chapter 4

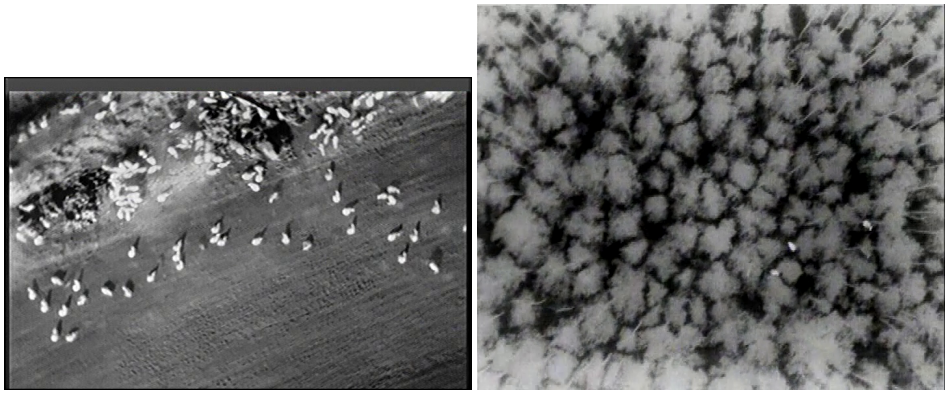
Automatic Sheep Detection

4.1 Data

The footage in the primary dataset was captured and transmitted from a quadcopter using radio signals to a nearby ground station. Unfortunately, the footage contains a severe amount of noise caused by radio interference, rendering most of the footage useless. Because our project was conducted after the end of the grazing period and ended shortly after the new grazing period started, there was no opportunity for collecting any new datasets. Instead, a collection of still images were extracted from the existing videos, at points where the amount of noise is at a minimum. This collection contains 25 still images from the two videos captured at 50m. These can be found in the attachments of this thesis. The footage in the videos taken from 60m and 100m are too cluttered by noise and does not contain much footage of sheep. Besides, the altitude that the videos was recorded from makes the sheep appear with too few details. Because of this, these videos were not used.

In addition to the mentioned dataset, FLIR video samples from [45] were used. Although this dataset does not contain images of sheep, some of the recorded animals have a similar appearance to sheep, making the dataset relevant for this thesis. This dataset was discovered at a late stage in the project and was not used during the development of the sheep detection algorithm described in this chapter. However, it was used to test the performance of the algorithm on images taken in a forest covered landscape, and to test how well the algorithm keeps track of objects in a video sequence. The video files that were used are also available in the attachment.

Figure 4.1 shows one example image from each dataset. Figure 4.1a is from the primary dataset, showing multiple sheep scattered along the edge of a farmland field. Figure 4.1b is from the secondary dataset, and shows wild boars in a coniferous forest [46]. The first image covers most of the challenges faced during the development of the algorithm and will be used in examples throughout the rest of this chapter.



(a) Image from the primary dataset **(b)** Image from the secondary dataset [45]

Figure 4.1: Example images from the two datasets

The extracted images are grayscale images. A grayscale image has a depth of 8 bit. Each pixel can have a value ranging from 0 to 255, where 0 is black, and 255 is white. This means that a grayscale image only contains different shades of black, where the values between 0 and 255 represent the intensity level.

4.2 Classic Image Processing

Based on the previous work presented in Section 2.2 and 2.3 an algorithm for automatic sheep detection is proposed. Figure 4.2 shows an overview of the sheep detection algorithm, equivalent to the pipeline that is presented in Section 2.2. The first step after image acquisition, *pre-processing*, is used to enhance sheep appearance and suppress image noise and background clutter. The *segmentation* step separates the potential sheep objects in the image, from the background. The *convert to polygon* step converts the segmented objects and represents their shape as polygons. *Find geometric properties* finds the length, width and area of each polygon and determines whether the polygon is an aggregation of multiple sheep or an individual sheep. An aggregation is defined as a group of sheep standing so close that they are mistakenly segmented as one large object. In case of aggregation, an attempt is made to split the group into individual objects. The last steps, *classification*, determines whether or not each individual object is a sheep.

4.2.1 Pre-processing: Noise Reduction

A varying amount of periodic noise caused by radio interference is seen in the dataset. Median blur is used to reduce the noise. First, a neighborhood, also known as a mask, filter or kernel, with a given size, is defined. The mask can be seen as a quadratic window that is systematically moved across the entire image. Each time the mask moves, the

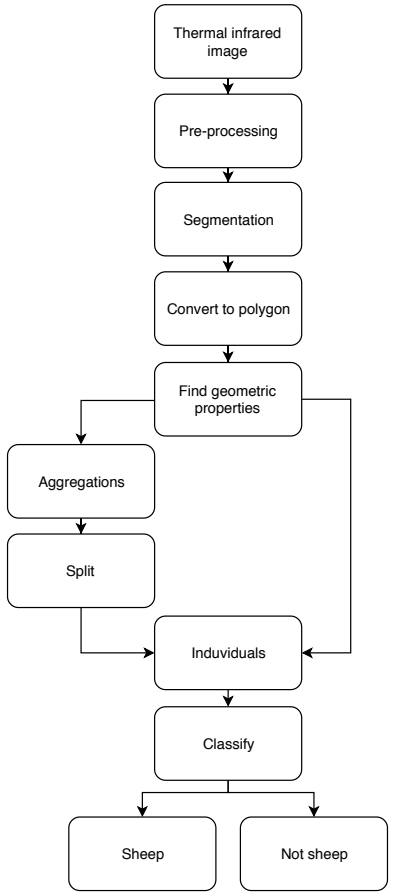
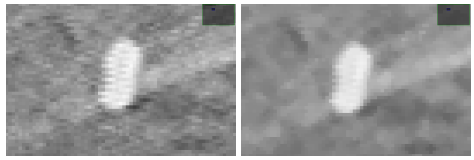


Figure 4.2: Sheep detection algorithm overview.

median intensity value of all pixels contained in the image area encompassed by the mask, is found. This value is assigned to the corresponding pixel in the output image. Figure 4.3 shows the effect of using median blur on one of the images in our dataset.



(a) Original noisy image **(b)** Median blur of (a)

Figure 4.3: Example of median blur.

4.2.2 Pre-processing: Opening by Reconstruction Top-Hat

In computer vision, the background is defined as everything in the image that is not of interest. In this case, this is the terrain surrounding the sheep. Based on different thermal properties and sun exposure rates, the terrain areas and objects are heated differently. This causes high-intensity regions to appear in the background, making it harder to distinguish the sheep from all the background, as the sheep also appear as bright spots.

A method called opening by reconstruction top-hat [31] is used to reduce the background clutter. This method attempts to remove or reduce the intensity of any objects larger than a given size. The algorithm uses well known morphological operators called erosion, dilation and opening [39]. Morphological reconstruction is used to ensure that the shapes of the objects of interest are preserved. These operations are described below.

Erosion

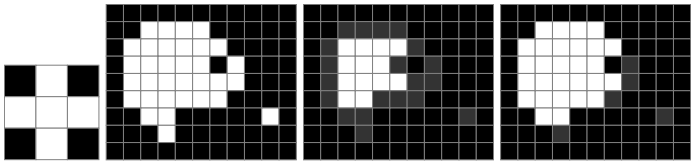
Erosion is a method for reducing the size of objects with high intensity. The process can be denoted as $f \ominus b$ where f is the input image and b is a structuring element. The erosion of image f by the structuring element b is defined as the minimum value contained in the region enclosed by b. This can be expressed using equation 4.1 where x and y are incremented through all values required so that the origin of b visits every pixel in f. The erosion of any location is determined by selecting the minimum value of f from all the values of f contained in the region coincident with b [39] (Section 9.6.1).

$$[f \ominus b](x, y) = \min_{(s,t)\epsilon b} \{f(x + s, y + t)\} \tag{4.1}$$

In general, high-intensity areas that the structuring element does not fit inside are removed from the image. Figure 4.4 shows an example of the erosion of a binary image. The structuring element is a 3×3 image, with values of 1's forming a white cross. The center pixel is the origin of the structuring element. Starting from the top left of the input image, the structuring element's origin is moved one by one pixel through the image. At all locations where the structuring element fits within the white region in the input image, the corresponding pixel is set to 1 in the output image. Figure 4.4c shows the final output, where the removed pixels are shaded dark gray for reference. As shown in the example, the operation removes the vertical pier at the bottom of the shape and the single white pixel in the bottom right. Figure 4.6b shows an example of the erosion of a grayscale image using a circular structuring element with a diameter of 24 pixels. The operation creates an output image where the size of dark features are increased, while the size of bright features are decreased.

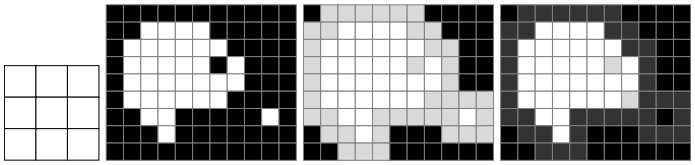
Dilation

Dilation works in the opposite way of erosion. The dilation of image f by the structuring element b, is defined as the maximum value contained in the region defined by b. In



(a) Structuring element, b **(b)** Input image, f **(c)** Erosion output, $f \ominus b$ **(d)** Opening, $f \circ b$

Figure 4.4: Example of erosion and opening on a binary image.



(a) Structuring element, b **(b)** Input image, f **(c)** Dilation output, $f \oplus b$ **(d)** Closing, $f \bullet b$

Figure 4.5: Example of dilation and closing on binary a image.

a binary image this means all points where b overlap f with at least one element [39] (Section 9.6.1). Similar to Equation 4.1 the equation for dilation is given by

$$[f \oplus b](x, y) = \max_{(s,t)\epsilon b} \{f(x + s, y + t)\} \tag{4.2}$$

The result is that dark area are reduced in size, or removed. Figure 4.5 shows an example of dilation on the same binary image that is used in Figure 4.4. The structuring element is a 3×3 image, where all pixels have value 1. In the same way as erosion, the structuring element is moved across all pixels in the input image. Wherever the structuring element overlaps the input image with at least one element, the pixel is marked with 1 in the output image. As shown in Figure 4.5c, the operation has filled the hole in the large shape and connected the single pixel in the bottom right. The added foreground pixels are shaded light gray, for reference.

Opening and closing

Two operations that rely on erosion and dilation are opening and closing. An opening is often used to smoothen the contour of an object, remove narrow lines or small objects. Closing is often used to fill small holes and gaps in the contour. The opening of image f by structuring element b is defined as the erosion of f by b, followed by a dilation of the result by b:

$$f \circ b = (f \ominus b) \oplus b \tag{4.3}$$

Similarly, the closing of image f by structuring element b is defined as the dilation of f by b, followed by an erosion of the result by b [39](Section 9.3):

$$f \bullet b = (f \oplus b) \ominus b \tag{4.4}$$

An example of opening and closing can be seen in Figure 4.4d and 4.5d respectively.

Morphological reconstruction

Morphological reconstruction is used to extract marked objects in an image, without changing the size or shape of the object. The process requires two images; one marker image, f, which contains the starting points for the reconstruction, and one mask image, g, which defines the constraints of the reconstruction.

Central to the morphological reconstruction is geodesic dilation. Geodesic dilation works by first computing the dilation of f by b, then at every point (x, y), selecting the minimum between the result and the mask. In [39] (Section 9.5.9) geodesic dilation of size 1 is defined as

$$D_g^{(1)}(f) = (f \oplus b) \wedge g \tag{4.5}$$

where f is the marker image, g is the mask image and $\wedge$ denotes the point-wise minimum operator. During reconstruction, geodesic dilation is performed on the marker image and repeated on its own output until the output is no longer changing.

As defined in [39] (Section 9.6.4), reconstruction by dilation of the mask, g, by the marker image, f, is the geodesic dilation of f with respect to g, iterated until stability is reached:

$$R_g^D(f) = D_g^{(k)}(f) \tag{4.6}$$

Opening by reconstruction

In opening by reconstruction, the input image g is first eroded by the structuring element b to get the marker image, f. Then, using g as mask, and the same structuring element b, the reconstruction is run as described above. In [39] (Section 9.6.4), the opening by reconstruction of size n of an image f is defined as the reconstruction by dilation of f from the erosion of size n of f:

$$O_R^{(n)}(f) = R_f^D[(f \ominus nb)] \tag{4.7}$$

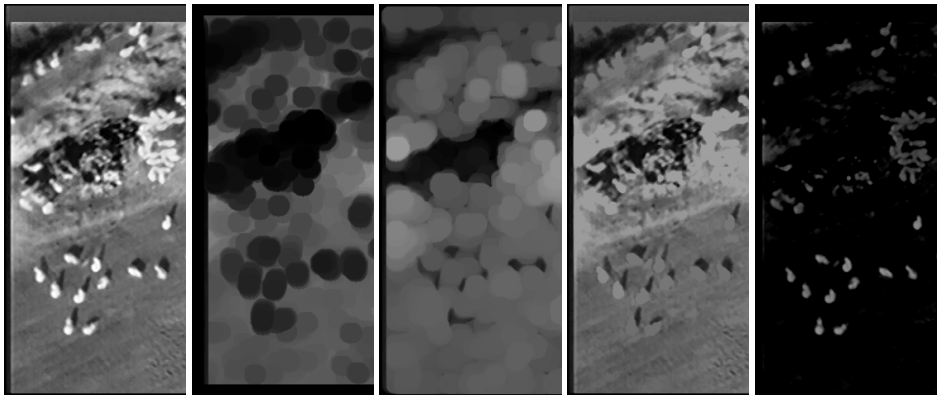
In the reconstructed image, all objects that remained after the erosion are preserved.

Opening by reconstruction top-hat

Finally, the so-called top-hat is found by subtracting the opening by reconstruction image from the input image. All pixel values in the input image are subtracted by the pixel intensity at their corresponding location in the reconstructed image. All details contained in the opening by reconstruction image will be attenuated in the top-hat output.

Figure 4.6 shows five cutouts from different steps of the opening by reconstruction top-hat. Based on the largest sheep that was expected to be found in the image, a circular structuring element with a diameter of 24 pixels was chosen. This is to ensure that all sheep disappear when the image is eroded, which can be seen in (b). This defines the marker image. (c) shows the first reconstruction step, which, as described above, is the minimum of the dilation of (b) and the input image (a). In each reconstruction step, everything in the input image darker than the marker is kept, while everything brighter than the marker is attenuated. This can be seen in the final reconstructed image, (d), where most of the sheep have blended into the background, while the background is kept. The final output is the result of (a) subtracted by (d). When comparing (a) and (e), it is evident that that much of the background clutter is removed, and the contrast between the sheep and the background is increased.

The algorithm described in [31] also uses closing by reconstruction top-hat to extract dark ("cold") features. However since the entire sheep, as seen from the UAV, will appear warmer than the background, this is not relevant in this context.



(a) Input image **(b)** Markers (erosion) **(c)** First reconstruction step **(d)** Reconstructed image **(e)** (a) minus (d)

Figure 4.6: Opening by reconstruction Top-Hat (cutouts from original images)

4.2.3 Segmentation using Double Threshold

After the reprocessing, the next step is the actual segmentation where foreground and background are separated. As seen in the previous section, the opening by reconstruc-

tion top-hat removes a significant amount of background clutter and enhances the contrast between warm spots of a given size and the background. Some background clutter still remain, however, most of this has a relatively low intensity compared to the sheep. This can be removed using thresholding. Thresholding requires two parameters: an input image and a threshold value. The algorithm checks every pixel in the input image, and all pixels with an intensity value greater than the threshold value are considered as foreground. All other pixels are considered as background. The algorithm outputs a binary image where all foreground pixels have the value 1 and all background pixels have value 0. The threshold of an image is defined as

$$g(x,y) = \begin{cases} 1 & \text{if } f(x,y) > T \\ 0 & \text{if } f(x,y) \leq T \end{cases} \tag{4.8}$$

where $f(x,y)$ is the input image, T is the threshold value and $g(x,y)$ is the output binary image [39].

Adaptive double threshold

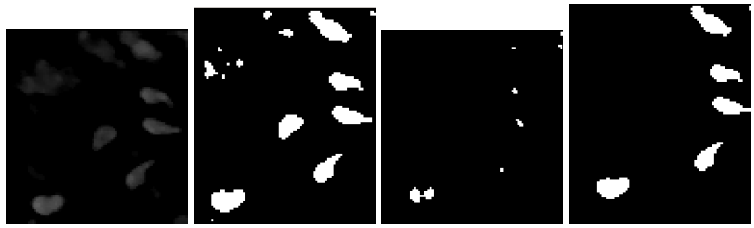
Thresholding using a pre-determined fixed threshold value is very sensitive, as noise or slightly abnormal intensity levels in the image can lead to objects not being picked up, or unwanted clutter being included. Adaptive double thresholding can be used to overcome this problem. As the name implies, two thresholds are performed, where each threshold value is found adaptively for each image. The method is based on the algorithm described in [31], which also describes the opening by reconstruction top-hat.

The first threshold is used to create a binary mask image, and the second threshold is used for creating a binary marker image. In this context, a marker is a single pixel or a group of adjacent pixels forming a connected component. Morphological reconstruction is then used to reconstruct the full contour of the brightest objects, which are marked by the marker image. The following values are used in the adaptive double threshold:

M = maximum gray scale value in top-hat image
c = some value between 0 and 1
ϵ = some small value between 0 and 1
$d = c + \epsilon$
$t_1 = c \cdot M$
$t_2 = d \cdot M$

The algorithm works as follows. First, the threshold t_1 is found for the given input image by assigning c a value that will ensure t_1 includes most shapes of interest in the image. See the example in Figure 4.7b. The output of the threshold using t_1 is the mask image, which will be used to restrain the binary reconstruction. While t_1 is determined once, t_2 is adjusted experimentally to give the desired number of markers for the reconstruction. Given the expected number of detected objects, T, d is incremented by ϵ and a new t_2 is calculated. This is repeated until the number of markers is greater than or equal to T. The

value T is decided by the number of detected objects in the previous video frame. Initially, this value is set to 1. In Figure 4.7, the final t_2 is set to 73, giving the marker image in Figure 4.7c. The reconstruction of the mask image, shown in 4.7b, by the marker image, shown in 4.7c, results in the output shown in 4.7d.



(a) Top-hat **(b)** Mask (t_1 = 29) **(c)** Marker (t_2 = 73) **(d)** Reconstructed shapes

Figure 4.7: Example of adaptive double threshold

4.2.4 Segmentation using Blob Detection

Another method for segmenting the sheep after the pre-processing step, is blob detection [48]. Compared to the adaptive double threshold which extracts high-intensity objects, this method aims at segmenting objects with a shape that resembles a circular blob. The blob detection does not depend on objects being of certain minimum intensity, as long as the object is brighter than the background. The main difference from the algorithm described in this section, and the original one, is the preprocessing step. As with the double threshold segmentation, opening by reconstruction top-hat is used for preprocessing. The original blob detection algorithm uses a method called dc notch filtering, which is less efficient for removing background clutter.

Edge Detection

The first step in the blob detection algorithm is to detect edges, which can be expressed as directions. For this, convolution is used. Convolution is a spatial filtering technique used to modify or enhance an image using filters. It works by moving a filter mask over an image and computing the sum of products at each location. At each point (x, y) in an image, the filter output is the sum of products of the filter coefficients and the image pixels encompassed by the filter. The convolution of an image $w(x, y)$ and a filter $f(x, y)$ is given by:

$$w(x,y)\bigstar f(x,y) = \sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)f(x-s,y-t) \tag{4.9}$$

In this case, convolution is used to find edges, which are sharp intensity changes in the image. The Sobel operators, shown in Figure 4.8, are used as filters, where the center element is the origin of each filter. By convolving the two filters with the image in Figure 4.9a, two output maps are produced; one for the horizontal gradient component, and one for the vertical gradient component. The two maps can be visualized as shown in Figure 4.9b and 4.9c, where bright pixels corresponds to positive values, dark pixels correspond to negative values and grey corresponds to zero. Using the ratio of the vertical to horizontal component, the arctangent is used to determine the direction vector of the gradient at each pixel location. Each computed vector points in the direction with the largest intensity increase. This means that all vectors along the edge of a blob point inwards. Each direction is rounded to the nearest $45°$, resulting in eight directions: $0°$, $\pm 45°$, $\pm 90°$, $\pm 135°$ and $180°$. This produces the vector image, G. The weakest edges that were found by the two Sobel operators are considered to be background clutter. These are removed by first finding the absolute magnitude of the two Sobel derivatives. A result of this is seen in Figure 4.9d. Every pixel from this image with a lower intensity than a certain level is set to 0 in G. An example of the resulting G is seen in Figure 4.9e. Notice that there are eight shades of gray along the object edges, where each shade represents a gradient angle. Rotating every angle in G by $90°$ counterclockwise gives the vector edge image E. Each pixel in E is an edge element which points in a direction so that the intensity to the right of the edge is greater than that to the left.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

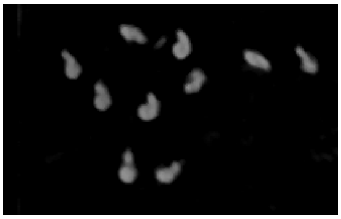
(a) Horizontal edge filter **(b)** Vertical egde filter

Figure 4.8: Sobel operator

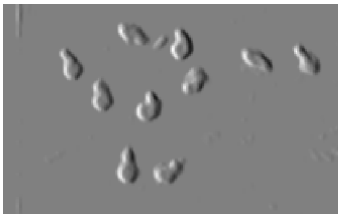
Spoke Filtering

The next step is the spoke filter, which is where the blobs are detected. First, a spoke register image, R, is created. Each pixel of G generates a straight line segment pointing in the direction of the gradient. An example of this is shown in Figure 4.10. For illustration purpose, the blob in the figure is represented as a binary image, and spokes are only shown for 14 of the edge pixels. The numbers on each edge pixel represent the gradient direction.

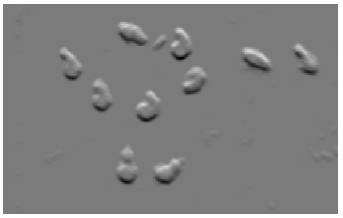
The next step is to check each pixel's neighborhood and count the number of unique spoke directions that intersects the neighborhood. In Figure 4.10, the gray 3×3 square represents the neighbourhood. In this example, at least four unique spoke directions intersects the neighbourhood: $0°$, $45°$, $90°$, $-45°$. The highest possible number of unique spoke directions is eight. If the number exceeds a given threshold, a blob is found. A binary image



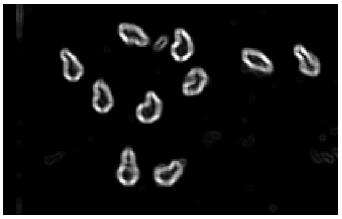
(a) Opening by reconstruction top-hat



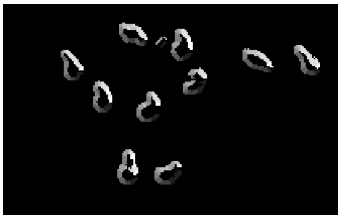
(b) Sobel gradient in x-direction



(c) Sobel gradient in y-direction



(d) Absolute value of vertical and horizontal Sobel gradient



(e) Visualization of gradient directions rounded to nearest $40°$

Figure 4.9: Edge detection steps visualized.

is created where all pixels with a blob-neighborhood are assigned the value 1, while all other pixels are set to 0. Figure 4.11a shows the spokes generated based on the image in Figure 4.9e. Figure 4.11b illustrates the number of spoke directions in each pixel's 3×3 neighbourhood. Red means that eight unique spoke directions intersect with the pixel's neighborhood, while green corresponds to 7 and blue 6.

Figure 4.11c shows the markers for the detected blobs. In this example, all pixels with a neighborhood being intersected by 6 or more spoke directions, are colored white.

Gradient based segmentation

When the spoke filtering is complete, and all blobs have been identified, the segmentation is run. The segmentation works by finding an optimum threshold for each blob, which ensures that the shape is entirely segmented. First, an initial threshold value is found in the input image, at the area where the blob is detected. The input image is then thresholded, with a gradually decreasing threshold value, causing the shape of the blob of interest to

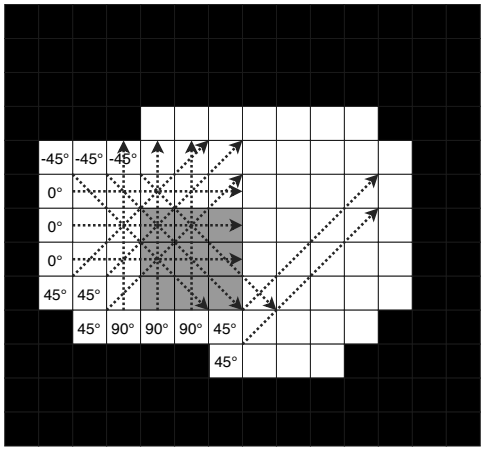
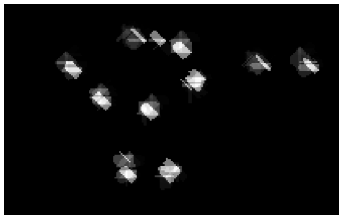
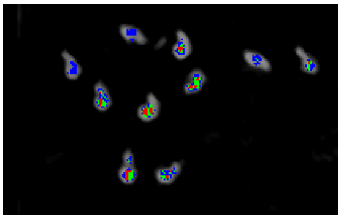


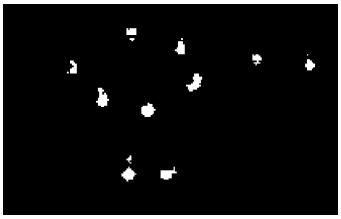
Figure 4.10: Spoke filtering



(a) Generated spokes



(b) Spoke directions in neighbourhood: red = 8, green = 7, blue = 6



(c) Marker: 6 spoke directions or more

Figure 4.11: Spoke filtering steps visualized.

grow gradually. At each new threshold value, the shape of the blob is analyzed. The boundary, B_1, of the blob is found. Additionally, two boundaries B_0 and B_2 are found, which are located immediately inside and outside B_1, respectively. The method presented in [48] compares the directions of each edge pixel in B_0, B_1 and B_2, with the corresponding edge pixels in image E, to determine whether the thresholded shape is a good segmentation. The implementation used in this thesis, however, compares the three borders with the gradient magnitude image (Figure 4.9d). The average pixel value inside the area covered by

the three boundaries is calculated and used as a score. Once the perimeter of B_1 exceeds a pre-defined value, or it touches the edge of the image, the threshold for the given blob stops and the threshold value that produced the best score is selected for segmenting the final blob shape. This is illustrated in Figure 4.12, where the final threshold is the one that produced the highest score. The upper row shows the segmented contour, while the second row shows the corresponding boundaries, overlayed on the edge magnitude image. The three boundaries, B_0, B_1 and B_2 are colored red, green and blue, respectively.

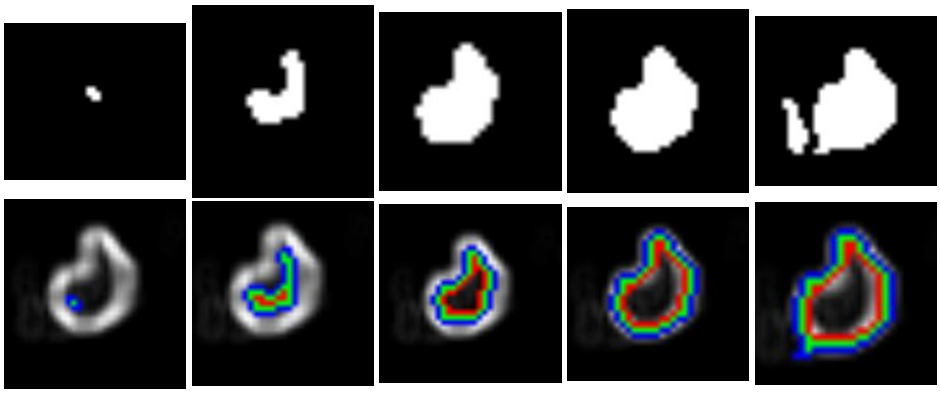


Figure 4.12: Gradient based segmentation. The contour matching the gradient image best is selected.

4.2.5 Classification

After the image is segmented using double threshold or blob detection, each segmented object is individually analyzed to decide whether the object is a sheep, an aggregation of sheep, or no sheep. Before the analysis begins, the following values are calculated.

A_{min} = minimum sheep area in m^2
A_{max} = maximum sheep area in m^2
a_{min} = $A_{min} \cdot (px/m^2)$
a_{max} = $A_{max} \cdot (px/m^2)$
p_{min} = min perimeter in pixels
p_{max} = max perimeter in pixels

The values are based on the camera's field of view, the image resolution and the height above ground. The area and perimeter values are found using the formulas for an ellipse.

The first check of the classification is whether the area of the object is within the range defined by a_{min} and a_{max}. If so, the ratio between the area of the contour, and the area of the contour's convex hull is found. Figure 4.13 illustrates the difference between the non-convex (concave) and the convex hull of an object. The ratio between the two areas enclosed by the two contours gives an indicator of how complex the shape is. A high ratio

(a) Object contour **(b)** Convex hull

Figure 4.13: Concave and convex hull of an object.

means that the object is relatively simple, while a low ratio means that there is a possible aggregation of sheep, which is the case in Figure 4.13. If the ratio is greater than a given threshold (between 0 and 1), the object is classified as a "sheep". All other objects are classified as "not sheep".

For the objects classified as "not sheep", a new check is made to check whether the object is an aggregation. If the object has a convex hull area ratio lower than a given threshold, and the contour area is larger than a_{min}, the object is classified as an "aggregation" and further processed later.

4.2.6 Aggregation Splitting

Particularly during the double threshold segmentation, sheep standing very close or touching each other, are often extracted as a single, aggregated, object. Different methods can be applied to attempt splitting the aggregations into single sheep. [51] use edge detection to find borders that can be used to form a continous line that separate the entire sheep. In several cases in our dataset, the level of detail is too low to find edges that are strong enough to separate the entire sheep. A different method, which is commonly used for separating objects, is the watershed algorithm [39] (Section 10.5).

Watershed

Morphological watershed is a powerful segmentation algorithm, especially helpful for separating touching objects.

Watershed splitting works as follows: consider a grayscale image as a topographic map where the intensity of each pixel is mapped to the elevation of a landscape. Dark pixels represents low elevations, such as valleys, and bright pixels represents high elevations, such as hilltops and ridges. The contours of the aggregation are used to extract the shape from the opening by reconstruction top-hat image. To get the desired topographical map, the image is inverted, so that bright pixels become dark and dark pixels become bright. Figure 4.14 shows the result, together with a 3D representation. The watershed algorithm can be illustrated by making a "hole" in the bottom of each valley, then raising the water

level from underneath the surface. As the valleys will become filled, at some level, the water from one valley will meet the water from another valley. A watershed line is created where the water meets. When the entire area (the image) has been flooded, the result is a set of connected edges separating the different valleys [39].

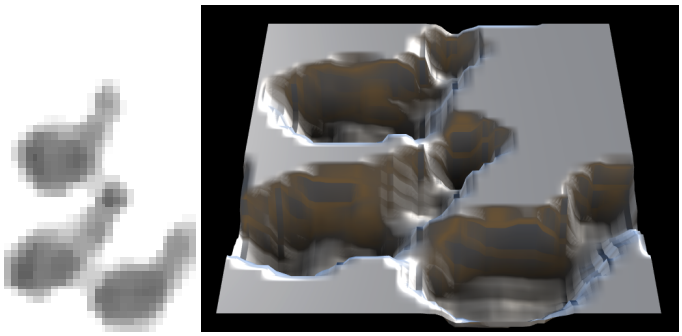


Figure 4.14: Topological representation of edge image. 3D image generated at: `http://cpetry.github.io/NormalMap-Online/`

In our case, the markers, or "holes", where the water starts flowing is defined by the darkest areas inside the grayscale image. Additionally, external markers can be created to restrict the algorithm to work inside specific regions. As an example, applying the watershed on the image in Figure 4.14 without external markers will result in watershed lines being created on the flat area on top of the valleys. To avoid this, the shape of the aggregation is used as to create an outer marker around the three large "valleys". A watershed line will be created where the water reaches the external marker.

Concavity based

The second method that is implemented for splitting aggregations, is inspired by the algorithm described in [44]. In Section 6.2.2, the output is compared with watershed splitting. The method uses concavity analysis to split images of clustered cells. The first step is to find concavity points in the shape. Due to potentially noisy contours, the shape is first smoothed to reduce the number of small bumps in the shape that are mistakenly detected as concavity points. This is done using a polygonal approximation. The most concave points are found using a sliding window which visits each contour point, counting the number of white pixels in the area encompassed by the window. Figure 4.15 shows how this works. The arrows show the travel path of the bounding window, which is the red square, and the black dots mark each stop, i.e. contour points. If the ratio between the number of white pixels and the window area is above a certain minimum, the point is considered a concavity point. In this example, 80% of the pixels encompassed by the window are white. When all concavity points are identified, the distance between each set of concavity points is measured. If this is longer than a minimum length, a check is made to see if the line between the two points intersects with any background pixels (the black area). If not, the

perimeter of each new object created by the potential split is measured. If at least one of the objects has a perimeter longer than a certain minimum, the split is saved.

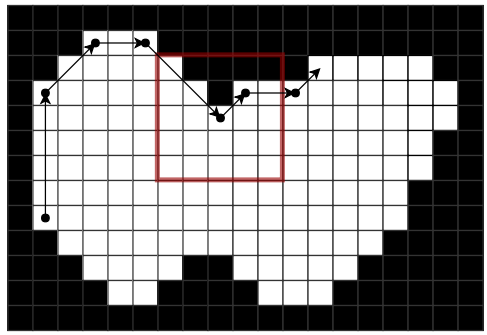


Figure 4.15: Sliding window searching for concave points.

4.3 Machine Learning

4.3.1 Convolutional Neural Network

CNN is a supervised learning method, meaning that it requires a labeled dataset [2, 6]. In our case, the dataset consists of labeled images of sheep and not sheep. Using this dataset an image classifier using CNN was trained. The CNN shown in Figure 4.16 is divided into two stages: feature extraction and classification. The first part extracts features in an image while the second part uses these features to classify an image.

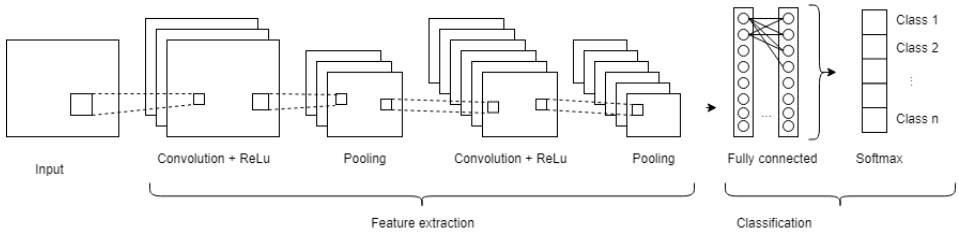


Figure 4.16: CNN [2]

Feature extraction

A layer is defined as a series of operations. The input image is first passed through the convolutional layer. In this layer, the features of an image are extracted using the convolution operator. In Section 4.2.4 an image of sheep is convolved with two Sobel filters to extract the horizontal and vertical edge features of the sheep. Different types of features can be

extracted by changing the filter coefficients. Figure 4.17 shows an example of how a filter can be used to extract diagonal lines. Image 4.17a shows a pixel representation of an image of a X. The grayed squared areas show two filter locations during the convolution process. By calculating the sum of products of the corresponding filter coefficients and the pixels in the first filter neighborhood a score of 2250 is calculated indicating that a diagonal feature has been found. Doing the same for the second filter neighborhood results in a score of 0 indicating that no feature is found. The convolutional layer uses a predefined number of filters with different filter coefficients to extract features. The greater the number of filters the more features are extracted. The size of the filters is also configurable. The result of this layer is a set of images called activation maps. The activation maps are a result of convolving the input image with every filter.

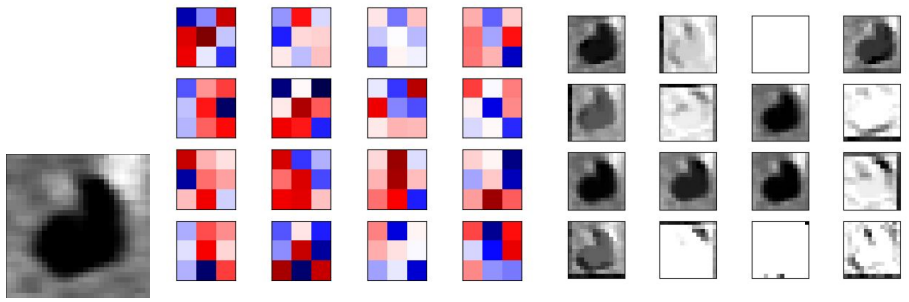
50	0	0	0	0	0	0	0	50
0	50	0	0	0	0	0	50	0
0	0	50	0	0	0	50	0	0
0	0	0	50	0	50	0	0	0
0	0	0	0	50	0	0	0	0
0	0	0	50	0	50	0	0	0
0	0	50	0	0	0	50	0	0
0	50	0	0	0	0	0	50	0
50	0	0	0	0	0	0	0	50

0	0	15
0	15	0
15	0	0

(a) Image of a X **(b)** 3x3 filter

Figure 4.17: Diagonal feature extraction.

Figure 4.18 shows an image inputted into a convolutional layer containing 16 3x3 filters. The filter values are represented using red and blue. Red filter value indicates a positive reaction to a black pixel while blue indicates a negative reaction. The results of convolving the filters with the input image show that different features of the input image are found.



(a) Input image **(b)** 3x3 filters **(c)** Result from convolving 4.17a with 4.17b.

Figure 4.18: Example of a convolutional layer.

After the convolutional layer, the activation maps are passed through the normalization layer. This layer adds non-linearity to the CNN by using an operation known as Rectified Linear Unit (ReLu) which is an element-wise operation, meaning that it is applied to every pixel in the image. Its output is given by $Output = Max(0, Input)$. In practice, this operation replaces all negative values in an activation map with 0s.

After the normalization layer, the resulting activation maps are passed through a pooling layer. In this layer, the activation maps are sampled while retaining the most important data. In our case, an operation known as max pooling was used. First, a window size and stride is defined. The window is moved across the entire image using the defined stride length. The max value for each window location is selected and stored in a new sampled image. Figure 4.19 shows an example of max pooling using a 2x2 window and a stride length of 2.

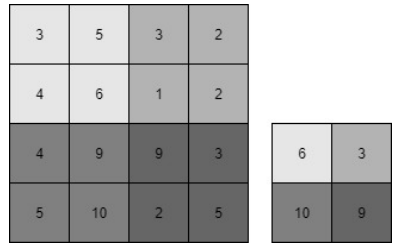


Figure 4.19: Max pooling

After the pooling layer, the same sequence of convolution, normalization and pooling is repeated once more where the sub-sampled activation maps are inputted into the second convolutional layer. In the second convolutional layer, every input image is convolved with every filter resulting in more abstract features, e.g. composition of two features found in the previous convolutional layer. The result of this layer is inputted into a second normalization and pooling layer.

Classification

The classification of the input image is done in the fully connected layer. The activation maps from the previous layer are flattened into a 1-dimensional vector, where each index gets a weight to a corresponding class. A higher weight indicates that this feature is more frequently found in that particular class. In practice, this layer looks at the activation maps and determines which features most correlates to a particular class and computes probabilities for each class based on this.

Training

When a CNN is first created the filter values or weights are initialized using random values. In this state, the CNN does not know how to differentiate between the two classes and

needs to be trained using backpropagation. The first step is to input a training image into the CNN. The second step is to use a loss function to calculate an error rate for the classification. The third step is to calculate the partial gradient for each weight with respect to the error rate. The fourth step is to increment the weights with a constant called a learning rate in the opposite direction of the gradient. Every time this process is repeated, the weights are adjusted in a manner that allows them to classify images better.

Chapter 5

Implementation

This chapter describes the implementation of the path generation and sheep detection algorithm. This includes an explanation of the technological choices, the motivation behind them and how they are used in the project.

5.1 Path Generation

5.1.1 Development Environment

Platform and dependency handler

NodeJS is a runtime platform built on Google Chrome's JavaScript engine [14]. The main motivation for this choice is the active community surrounding it, which has resulted in a very mature and reliable platform with a rich variety of tools. Dependencies can easily be installed using its own package system which is the worlds largest ecosystem of open source libraries.

User interface

Electron is a tool used to build cross-platform apps with JavaScript, HTML and CSS [4]. In combination with Electron, React, Redux and WebPack were used. React was used to simplify the creation of HTML components [19]. Redux was used to simplify the handling of different system states [20]. WebPack was used to bundle the entire project including all its dependencies into a single JavaScript file providing live reloading during development [25]. Google Maps API was used to handle map related operations such as defining a search area [26].

Test framework

The algorithm for generating the coverage path contains many mathematical functions. It was important that every one of them worked as expected. To enforce this, the Chai Assertion Library [3], was used to create tests for every mathematical function thoroughly testing them.

5.1.2 Coverage Path Generation

Figure 5.1 shows an example of how to use the system to generate a coverage path for a delimited area omitting lakes. The first section shows how the required classes are included. The next section defines the required arguments used to generate a coverage path. `coordinates` are a list of latitude and longitude coordinates used to create a georeferenced polygon. `lakes` are created using a geojson file containing a list of selected lakes in Norway. The geojson was extracted from the N50 Geodatabase provided by The Norwegian Mapping Authority [13]. The camera settings are specified using `cameraWidth` and `overlap`. The last section shows how the `PathGenerator` is used to generate a coverage path using the defined arguments. The returned `path` object contains the original polygon, decomposed polygon, distance and the generate path, transition lines and sweep direction represented in latitude and longitude coordinates. A path without omitting lakes can be generated by not including the lake argument.

```
2  const Polygon = require('./polygon');
3  const Lakes = require('./lakes');
4  const PathGenerator = require('./PathGenerator');

6  var coordinates = [
7      {lng: 11.48, lat: 62.36},
8      {lng: 11.44, lat: 62.35},
9      {lng: 11.53, lat: 62.33}
10 ];
11 var area = new Polygon(coordinates);
12 var lakes = new Lakes("./lakes.geojson");
13 var cameraWidth = 50;
14 var overlap = 0.1;

16 var path = PathGenerator.generate(area, cameraWidth, overlap, lakes)
```

Figure 5.1: Example of generating a coverage path.

5.2 Sheep Detection: Classic Image Processing

5.2.1 Development Environment

Programming language

The sheep detection system is implemented using Python 3, which is a powerful and fast programming language [18]. Python is widely used, especially in the research field resulting in a very mature varied scientific stack. Python is also very easy to work with and fast to set up which is very useful when prototyping different image processing methods.

Libraries

Open Source Computer Vision Library (OpenCV) is an open source computer vision and machine learning library. OpenCV has a very active community and is very reliable and fast. It has more than 2500 optimized algorithms covering most of the image processing techniques used in our algorithm [49]. All basic image processing techniques such as convolving an image with a filter are implemented using this library. Figure 5.2 shows an example of how OpenCV is used to load and erode an image using an eclipse.

```
1 import cv2

3 image = cv2.imread("image path", 0)
4 ksize = 5
5 kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (ksize,
      ksize))
6 eroded_image = cv2.erode(image, kernel)
```

Figure 5.2: Erosion using OpenCV.

NumPy is a fundamental package needed for scientific computing. NumPy provides a powerful N-dimensional array object and basic linear algebra. Figure 5.3 shows an example of using NumPy to find the max value in an image.

```
1 import numpy as np

3 highest_intensity_value = np.max(image)
```

Figure 5.3: Finding max value in an image using numpy.

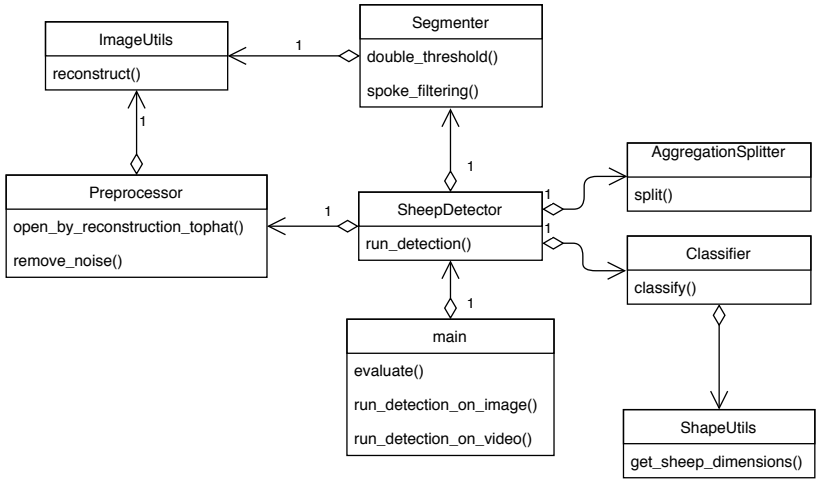


Figure 5.4: Class overview of sheep detection using classic image processing.

5.2.2 Overview

The class diagram in Figure 5.4 shows an overview of all classes in the sheep detection system. To make the system modular, each step described in 4.2 was derived into separate classes. This allowed us to implement, add and test new steps easily. `SheepDetector` is the most important class and assembles the sheep detecting algorithm using different classes for each step in the algorithm. This class has a function called `run_detection`. Figure 5.5 shows how this function works. The first argument is the image to be analyzed in grayscale format. The second argument is the expected sheep dimensions, which are calculated in `get_sheep_dimensions` based on camera properties and height. type detection method to be used. This function returns an image with a bounding box for each object detected. The `seg_type` is the name of the segmentation type to be used; `"blobs"` or `"double"`. `expected_sheep` is an integer used in the double threshold, and is based on the number of detections in the previous frame.

```
1 from ShapeUtils import get_sheep_dimensions
2 import SheepDetector as sd
3 #...
4 sheep_dimensions = get_sheep_dimensions(cam_height, sheep_props
      , camera_props)
5 found_sheep = sd.run_detection(image, sheep_dimensions,
      seg_type, expected_sheep)
6 #...
```

Figure 5.5: Code example of using the sheep detector

`main.py` uses this function to allow the user to test the sheep detection algorithm. Figure

5.6 shows how the script is used.

```
$ python main.py --image ./image.png \
  --json ./image.json --altitude 50 --expected 88

$ python main.py --image ./image.png \
  --json ./image.json --altitude 50 --segtype "blobs"

$ python main.py --video ./video.mp4 --altitude 50
```

Figure 5.6: How to use the sheep detection system.

5.3 Sheep Detection: Machine Learning

Python 3 was chosen using the same motivations mentioned previously. Additionally, multiple open source machine learning frameworks come with a Python API. TensorFlow [24] was used for implementing the CNN model. This framework was chosen due to its popularity within the field of machine learning, good documentation and the variety of online tutorials.

The 2D plotting library, Matplotlib [10], was used to visualize nodes and layer outputs during the execution of the CNN-model. This was helpful for debugging and testing out various parameters used when training the model.

5.3.1 Keras RetinaNet

Keras RetinaNet is a state of the art R-CNN based on RetinaNet described in the paper Focal Loss For Dense Object Detection [8, 47]. A prediction model was trained using Keras RetinaNet and a dataset containing infrared images of sheep and corresponding CSV files containing the coordinates of every sheep in each image. The dataset was created using LabelImg, a graphical image annotation tool used to mark sheep in images.

Chapter 6

Result and Discussion

In this chapter, we present and discuss the obtained results. We will start with path generation followed by sheep detection using classic image processing and machine learning.

6.1 Path Generation

The results of the coverage path generation are presented using a series of polygons with different generation settings. Every coverage path is generated using the same specification, e.g., same distance between each sweep line.

6.1.1 Path without Lake Omission

Figure 6.1 shows a generated coverage path without omitting the lake contained within the polygon. The green circle is the starting point of the path while the red circle is the ending point. The violet arrows show the optimal sweep direction. The green lines are the coverage pattern and follow the sweep direction. As explained in Section 3.2.3, because the polygon is concave, the polygon has been decomposed into two convex polygons. Each of the polygons has their own optimal sweep direction and a coverage pattern. The black arrow shows the approximated optimal path between each sub-polygon. This arrow starts at the ending point of a coverage pattern and points to the nearest starting point of a new coverage pattern. The distance between each line in the coverage pattern is 45m, which is 5m less than the camera footprint. This is the result of the 10% overlap. The total distance for this path is 602km.

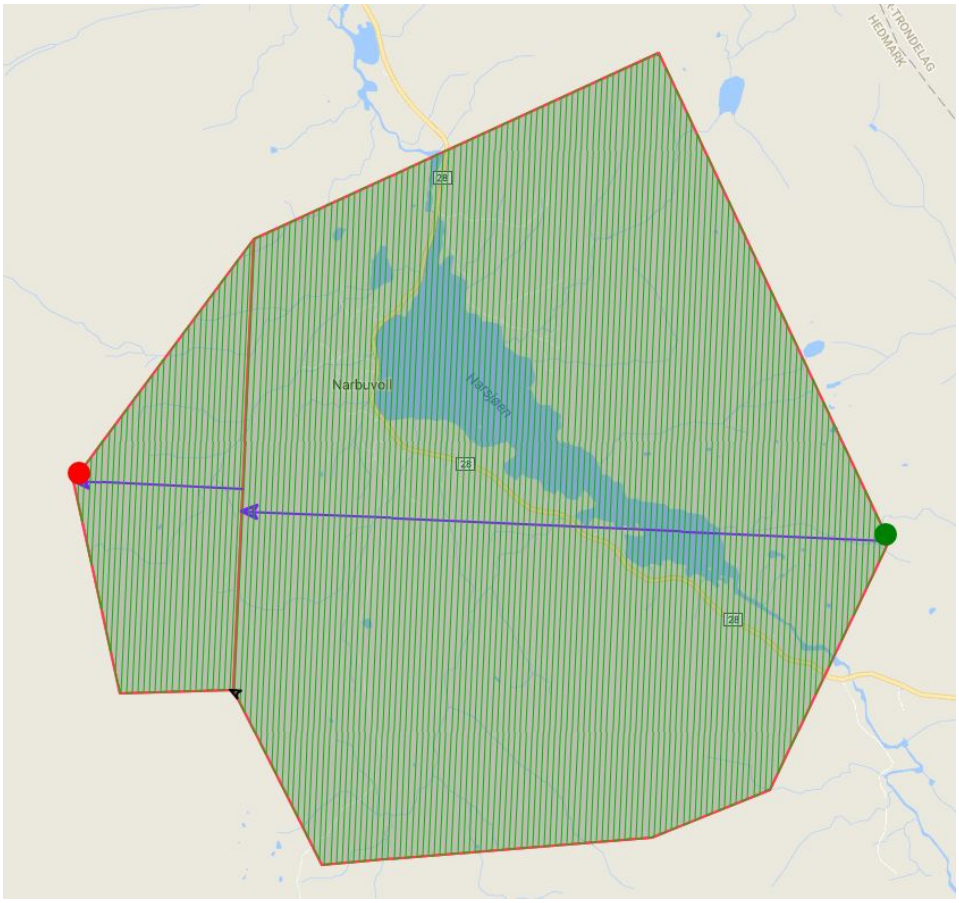


Figure 6.1: Distance: 602km

6.1.2 With Water Omission

Figure 6.2 shows a generated coverage path for the same polygon as the one shown in Figure 6.1, but with the largest lake omitted. The distance for this path is 583km which is 19km, or 3.1%, shorter than when the lake is not emitted.

Figure 6.3 shows two different coverage paths for a polygon covering an area with multiple smaller lakes. The first path, shown in Figure 6.3a, does not omit the lakes, while the second path, shown in Figure 6.3b, omits the lakes. In this case, omitting the lakes results in an increase of 8.1%, or 5 km, in total flight distance. The primary contributing factor to the increased distance, is the transition legs from the end of the sweep path in one

polygon, to the beginning of the sweep path in the next polygon. In the case with small lakes, several of the transitions legs are even longer than the biggest lakes, and the lakes are narrow enough to be covered by two or three sweeps. Additionally, there is a high increase in the number of sweep lines, and as mentioned in the previous section, the extra cost of turning from one sweep line to the next is at least 45m.

This shows that omitting lakes is not necessarily a good choice. However, in cases where the lakes require a significant number of sweeps to be covered, the total flight distance will be reduced.

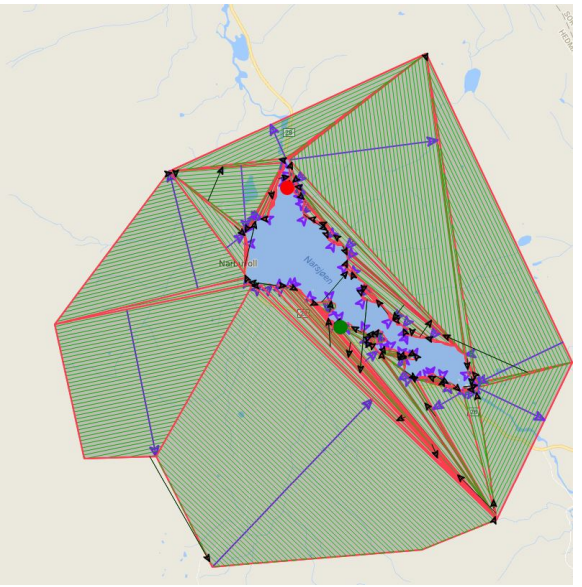


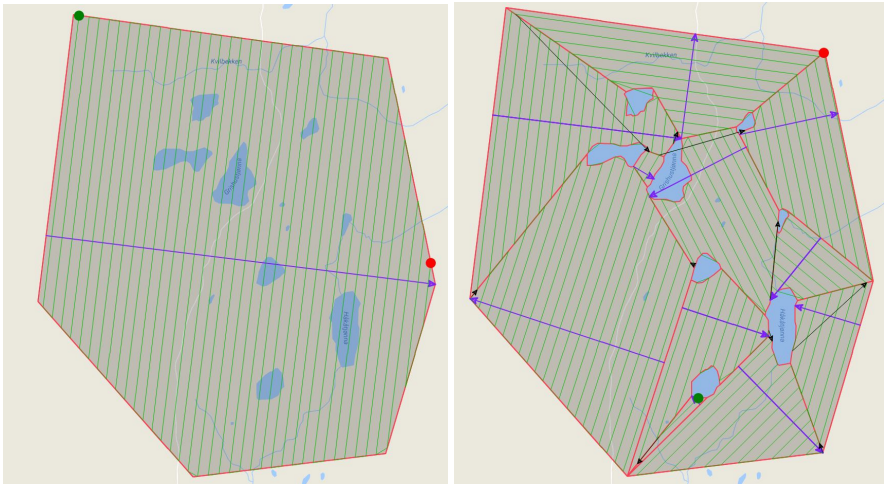
Figure 6.2: Distance: 583km

6.1.3 Polygon Merging

The coverage path shown in Figure 6.4a is a merged version of the one shown in Figure 6.1, where the two polygons have been merged into a single polygon. This results in a reduction of 0.5%, or 2.1km, of the total distance.

The coverage path shown in Figure 6.4b is a merged version of the one shown in Figure 6.2. In this scenario, the effect of the polygon merging is more evident. Multiple polygons are merged, and the overall complexity of the coverage path is significantly reduced. The polygon merging results in a reduction of 2.2%, or 13km. Comparing the new route with the route generated without lake omission, the distance is reduced by 5.3% or 32km.

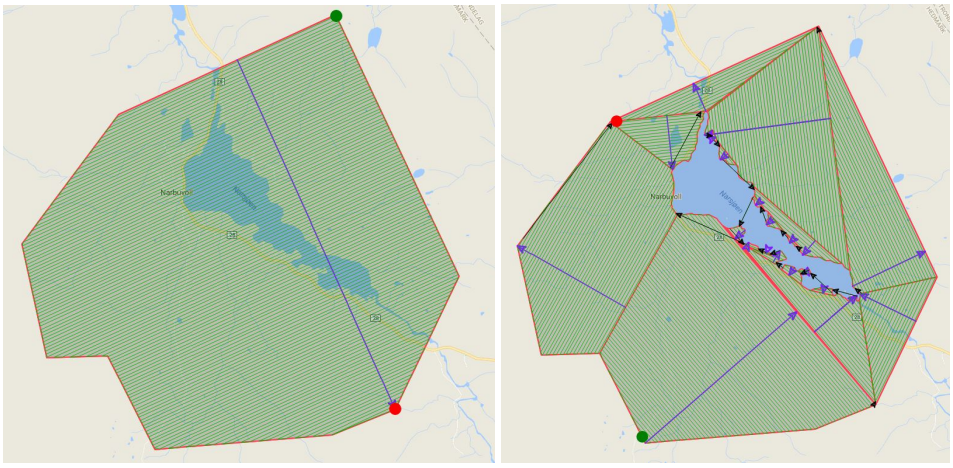
Polygon merging contributes to reducing the complexity of generated coverage path by reducing the number of sub-polygons. This has the most effect on the total distance of



(a) Distance: 62km **(b)** Distance: 67km

Figure 6.3: Coverage path omitting multiple small lakes

the transition legs, but also the number of sweep lines, both contributing to a shorter route.



(a) Distance: 599km **(b)** Distance: 570km

Figure 6.4: Coverage paths with merged polygons

6.1.4 Compared with Mission Planner

Figure 6.5 shows the generated coverage path using the GCS Mission Planner [12], for a polygon identical to the one in Figure 6.4. The coverage path is generated using the built-in survey coverage path function with a line separation specified to 45 meters. Both of the coverage paths, i.e., with and without omitting the largest lake, generated by the algorithm proposed in this thesis, produces shorter distances than the one generated by Mission Planner. Comparing the numbers from the route generated by Mission Planner with the first route, which also ignores the water, Mission Planner generates a route that is 2.0%, or 12 km, longer. One factor causing this difference is the choice of sweep direction, resulting in a different number of sweep lines.

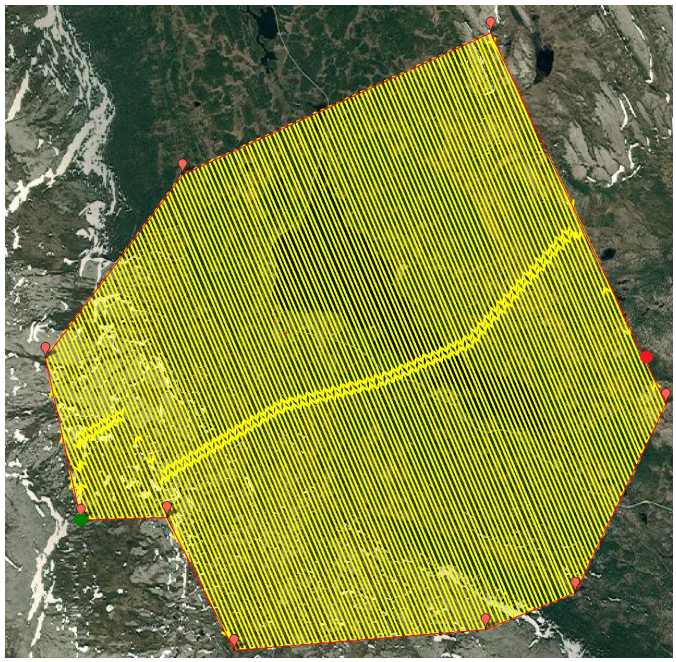
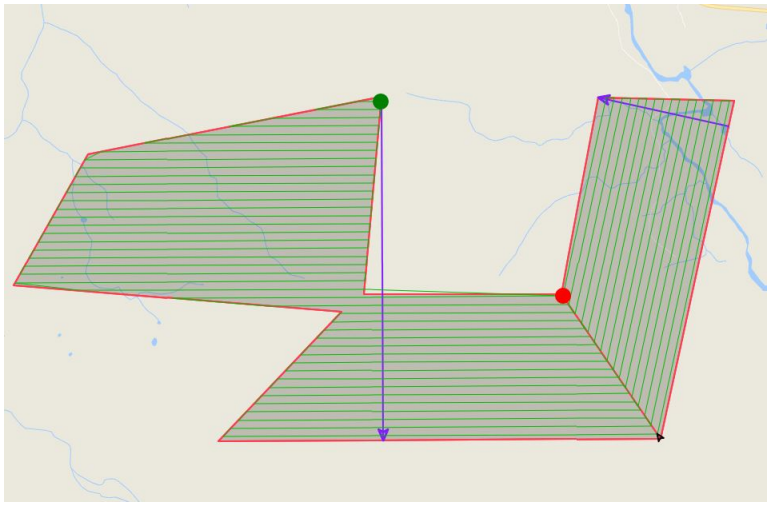
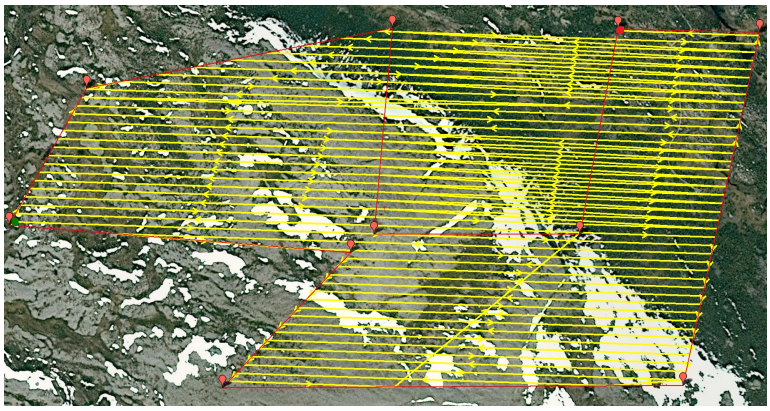


Figure 6.5: Generated by Mission Planner. Distance: 611km

In Figure 6.6, a different concave polygon is used. Figure 6.6a shows the route generated by our algorithm, while Figure 6.6b shows the route generated by Mission Planner. The coverage path generated by Mission Planner uses a single sweep direction for the whole polygon and ignores the polygon edges that interrupts the coverage path. This results in a route covering additional areas that are not part of the polygon. The algorithm proposed in this thesis avoids this by decomposing the polygon and covering each sub-polygon separately, finding the optimal sweep direction for each. The route generated by Mission Planner is 89.9%, or 89 km, longer.



(a) Distance: 99km



(b) Distance: 188km

Figure 6.6: Coverage path for a special scenario

6.1.5 Discussion

The decomposition of polygons produces extra transition paths and sweep lines, resulting in additional route distance. Since the method used for combining the coverage paths for all polygons into one route, only looks one polygon ahead at a time, it tends to produce some long transition legs after a few iterations. However, in most cases where water omission is not used, the algorithm produces good results, as the number of sub-polygons is low.

The decomposition algorithm does not always produce a minimum number of convex polygons. However, the post-processing step, where polygons are merged when possible,

highly improves this result.

The coverage paths generated the algorithm proposed in this thesis, is generally shorter than the ones generated by Mission Planner. The reason for this is the polygon decomposition and optimal sweep direction used by our algorithm. The results also show that only big lakes are worth omitting, as omitting small lakes results in additional transitions and sweep line, resulting in a longer route than without water omission.

6.2 Classic Computer Vision

Height Approximation

As described in Section 4.2.5, the height of the camera is used to determine the expected sheep size in the image. In a real use case scenario, the height is measured using sensors on the UAV and stored together with the recorded image. Because no accurate height data was available in our dataset, a manual height estimation was done for each test image. This was done by manually finding the largest sheep and reversing the process of calculating estimated sheep size based on height. Equation 6.1 was used, where f_w is the image resolution in x-direction (width), l_{max} is the longest found sheep in image pixels, L_{max} is a constant for the longest expected sheep length in meters, θ is the field of view in degrees in the x-direction. L_{max} was set to 1.4m for all images.

$$h = \frac{f_{fov_x}}{2 \cdot tan(\frac{\alpha}{2})} = \frac{\frac{f_w}{px/m}}{2 \cdot tan(\frac{\alpha}{2})} = \frac{\frac{\frac{f_w}{l_{max}}}{L_{max}}}{2 \cdot tan(\frac{\alpha}{2})} \tag{6.1}$$

6.2.1 Pre-processing: Opening by Reconstruction Top-Hat

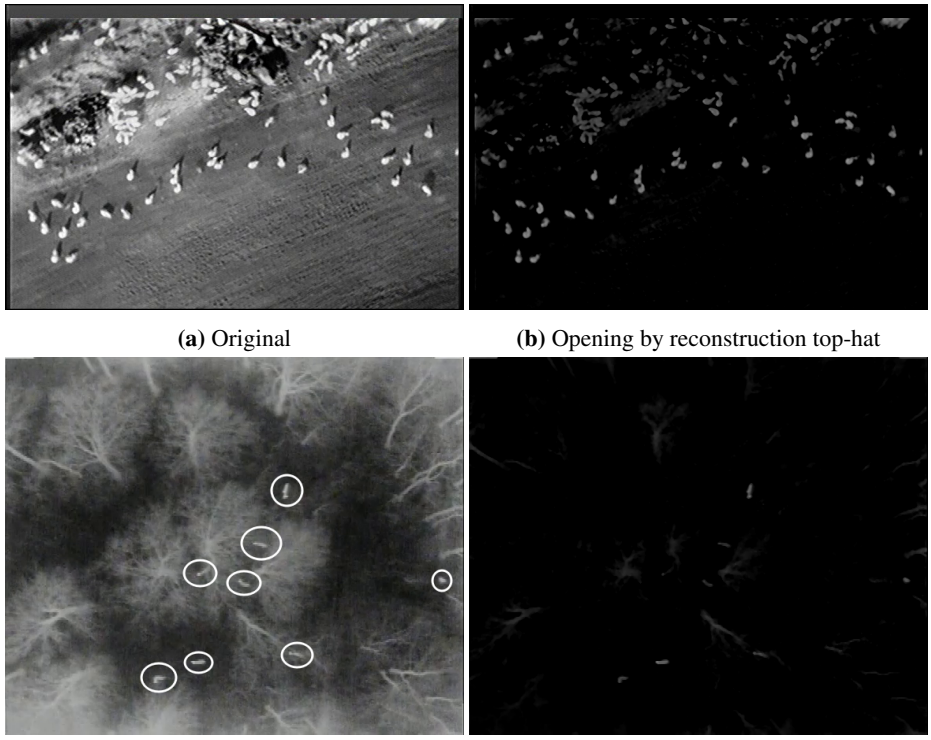
The opening by reconstruction top-hat, described in Section 4.2.2, is efficient for removing background clutter based on size criteria. In all images, a circular kernel was used to remove all objects larger than the maximum sheep size.

Figure 6.7b shows the result of applying noise reduction and opening by reconstruction top-hat on the image in Figure 6.7a. A height of 50 meters was calculated, resulting in a kernel diameter of 20 pixels. The results show that objects larger than 20 pixels, such as most of the evenly colored field, are filtered out (becomes black). Objects that are 20 pixels or less in diameter remains intact. This includes sheep, but also small heated objects on the ground, such as small rocks.

Figure 6.7d shows a different example. This image is extracted from the dataset, [45], provided by [46]. The image is taken in a forest covered area, where eight red deer are visible (circled for reference). The same height setting is used for this image. Although this image was taken 06:45 in the morning, the forest canopy appears relatively warm. However, the result of the pre-processing shows that most of the tree canopies are removed,

and the high-intensity vignetting along the image borders are eliminated. Another thing to notice with the vignetting is that, although it is eliminated, the underlying objects are still intact, which for example can be seen in the top right corner of Figure 6.7c and 6.7d. This means that if animals appear in the corner areas, they remain intact. Tree trunks and branches that are about the same thickness as the kernel diameter or smaller remains, together with some background clutter on the open (dark) areas.

One thing to notice in the examples from both datasets is that objects located on a higher intensity background become more attenuated than the objects located in lower intensity areas. The reason for this is that these objects are not eroded to the same degree since there is a lack of nearby pixels with low intensity. This complicates the double threshold segmentation since a lower threshold value must be used to pick up the darkest objects, potentially causing more background clutter to be included. For the blob segmentation, this is less of a problem, as each detected blob is thresholded individually.



(a) Original **(b)** Opening by reconstruction top-hat

(c) Original **(d)** Opening by reconstruction top-hat

Figure 6.7: Result of opening by reconstruction top-hat

Table 6.1: Final classification results using double threshold with varying T-factors

T-factor	Expected sheep	True positives	False positives
4	108	90	21
3	108	90	19
2.5	108	90	17
2	108	**90**	**15**
1.5	108	77	7
1	108	65	5

6.2.2 Segmentation

Double threshold

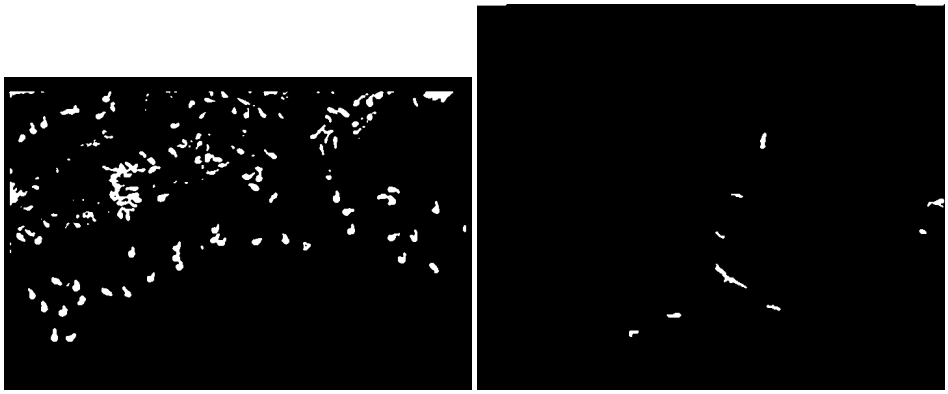
As described in Section 4.2.3, the double threshold segmentation requires an expected number of objects. To allow for a growth in the number of detected objects, this value is multiplied by a factor, giving a value T, which is used to limit the number of reconstruction markers. The values $c = 0.25$ and $\epsilon = 0.02$ were found appropriate for our data set. This means that the first threshold, used to find the mask image, is 25% of the maximum pixel value in the pre-processed image. Starting from this value, the second threshold is gradually incremented by 2% to find the best marker image. The c-value was chosen so that as many as possible of the objects of interest were extracted with a low amount of background clutter. ϵ was chosen to give good accuracy, keeping in mind that a lower value increases the calculation time.

Table 6.1 shows the classification results using double thresholding with different T-factors. The numbers are the totals after running the algorithm on all annotated images in the dataset. This includes the images in Figure 6.7. The table shows that a T-factor of 2 gives a good tradeoff between true and false positives. For the rest of the results presented in this chapter, this is the value that is used.

Figure 6.8a and Figure 6.8b shows the result of the double threshold segmentation performed on the two top-hat outputs shown in Figure 6.7b and Figure 6.7d, respectively. The result in Figure 6.8a is good, however, some background clutter can be seen in the upper left quarter of the image. Most of these are caused by different terrain characteristics heated by the sun. In Figure 6.8b the segmentation result is slightly better, as most of the background clutter appear darker than the deer. Only two false positives are segmented, which is the tree trunks in the lower midsection and on the middle of the right image edge. One deer is not segmented, as it appears too dark in the top-hat image. Using a lower c the last deer can be included, however, with the risks mentioned above.

Blob detection

The blob detection algorithm, described in Section 4.2.4, depends on three parameters for spoke generation; the threshold for ignoring weak edges, the spoke length and spoke



(a) Sheep in an open area **(b)** Deer in forest area

Figure 6.8: Double threshold with a T-factor of 2

separation distance from the edges. Additionally, two parameters are needed for selecting interesting areas in the spoke image; the minimum number of unique spoke directions in a pixel's neighborhood and the neighborhood size. Similar to the pre-processing step, the spoke length is based on the camera's height and the expected sheep size. To limit the segmentation to only include blobs not larger than a sheep, a spoke length half the length of the largest expected sheep was selected. To reduce the possibility of closely located or touching blobs being detected as a single blob, the spokes were set to start one pixel from the object edges. The threshold for removing weak edges was set to 95, where all pixels in the edge magnitude image with intensity larger than 95 were kept. This was found to be the value that removed most false positives while keeping a high number of true positives.

The minimum number of unique spoke directions in the pixel neighborhood, and the neighborhood size was found experimentally. Table 6.2 shows the final results of the sheep detection algorithm using different values for neighborhood size and number of spoke directions. For our dataset, a neighborhood of 2×2 and minimum 6 spoke directions resulted in a relatively low number of false positives and a high number of true positives. The value of 6 allows for some irregularities in the blob shapes. Using a lower minimum number of spoke directions will potentially result in more non-blob objects being segmented, while a higher number than 6 makes the algorithm too picky laving too many blobs undetected. As described in Section 4.2.4, the final step after the blobs have been identified, is to find the best threshold for each blob. The threshold stops when the perimeter of the object is too large. Using a large neighborhood to check for unique spoke directions results in closely located blobs being detected as one, causing difficulties when finding the optimal threshold, as the perimeter will grow too large. This means that blob filtering is not well suited for tightly connected aggregations of sheep.

Figure 6.9 shows the segmentation result of Figure 6.7a, when searching for minimum 6 spoke directions in a 2×2 neighbourhood, compared to minimum 8 spoke directions in a 5×5 neighbourhood. One evident difference between the two can be seen in the

Table 6.2: Final classification results using blob detection with varying neighbourhood sizes and spoke directions.

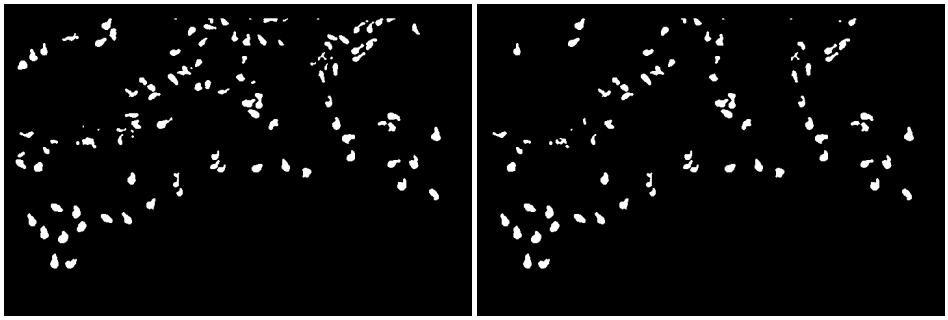
Neighbourhood	Spoke directions	True positives	False positives
2×2	≥ 6	**89/108 (82.41%)**	16/105
3×3	≥ 6	**89/108 (82.41%)**	20/109
4×4	≥ 6	82/108 (75.93%)	20/102
5×5	≥ 6	79/108 (73.15%)	21/100
2×2	≥ 7	63/108 (58.33%)	10/73
3×3	≥ 7	75/108 (69.44%)	13/88
4×4	≥ 7	80/108 (74.07%)	15/95
5×5	≥ 7	79/108 (73.15%)	16/95
2×2	≥ 8	36/108 (33.33%)	**2/38**
3×3	≥ 8	48/108 (44.44%)	8/56
4×4	≥ 8	61/108 (56.48%)	8/69
5×5	≥ 8	**73/108 (67.59%)**	**9/82**

upper left corner, where several sheep are left undetected in the second case. Also in the middle of the left edge in the image, and middle top to top right, several differences can be seen.

Comparing the blob detection segmentation with the double threshold, one important difference is the accuracy in the reconstruction of the object shapes. Comparing Figure 6.8a with Figure 6.9a some sheep contours are better defined in the latter case. As explained in Section 4.2.3, double thresholding uses a global threshold value for all sheep, causing some sheep with variable intensity to become partly segmented. Thus, in many cases, the blob detection provides a better foundation for classification based on contours.

Another difference is the handling of groups/aggregations. Some of the objects that are mistakenly connected in Figure 6.8a are separated in Figure 6.9a. One explanation for this is that the start of a spoke is offset by one pixel from the edge of the blob. However, in cases where objects are not automatically split, there is a problem. The blob detection algorithm does not create segmentations that are larger than the maximum expected sheep perimeter. This prevents background clutter from being included in the segmentation of the object, but it also prevents aggregations from being fully segmented. This can be avoided by increasing the expected perimeter.

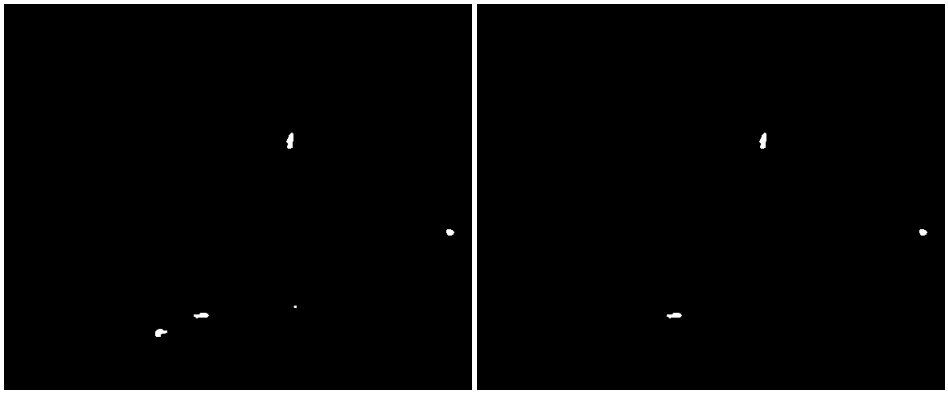
Figure 6.10 show the corresponding results of the segmentation applied on Figure 6.7d. Even with six minimum spoke directions, some animals are left unnoticed. This can be explained by the undetected objects having too weakly defined edges. When the threshold used to remove weak edges from the edge image is run, these edge spokes are not generated. Lowering the value of this threshold results in the rest of the deer being detected, however, it also introduces more false positives.



(a) $2x2$ neighbourhood, minimum 6 spoke direction

(b) $5x5$ neighbourhood, minimum 8 spoke direction

Figure 6.9: Blob segmentation of image in Figure 6.7a



(a) $2x2$ neighbourhood, minimum 6 spoke direction

(b) $5x5$ neighbourhood, minimum 8 spoke direction

Figure 6.10: Blob segmentation of image in Figure 6.7c

6.2.3 Aggregation Splitting

Section 4.2.6 describes the two splitting algorithms that were implemented: watershed-based and concavity-based. As explained in Section 6.2.2, splitting is most relevant when double thresholding is used for segmentation, as this method tends to produce more aggregations than blob detection.

The watershed algorithm uses the actual pixel intensities, from the input image, for watershed growing. Figure 6.11 shows the result of the two methods. The watershed algorithm works well for shapes where there is a clear line between the objects, e.g., the first, second and last row. When all objects appear with a more or less even intensity level, like in the fourth row, the watershed has no well defined growing boundaries. When a raw video feed is provided, the level of detail will potentially be higher, making the watershed process

more robust.

In comparison, the results of our implementation of the concavity split tend to give results that are very different from the desired split. The concavity method only considers the contour of the aggregation, and with a high variety of possible shapes, the watershed is better suited for this use case. If the shapes had a more predictable appearance, like a constant geometric shape, the concavity splitting could be an option.

Although the watershed was selected as the optimal splitting algorithm, there was a problem finding good seeds/markers for the watershed growing. The consequence of this is evident in the third row of Figure 6.11. The sheep in the lower part of the image is successfully separated from the object in the top of the image, but it is also split in half, resulting in the sheep to be undetected since the two parts are too small to pass the size check in the classification.

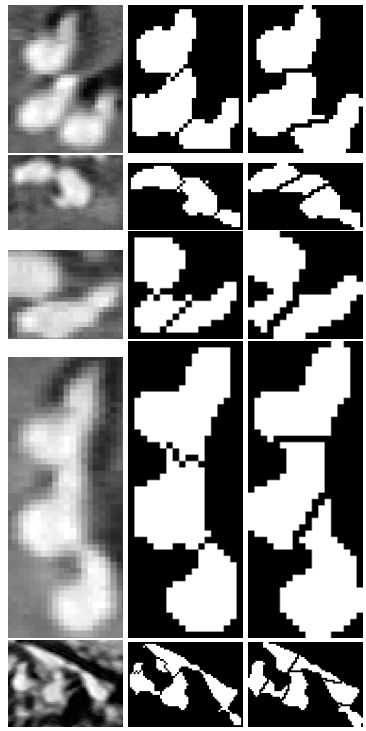


Figure 6.11: Comparison of aggregation splitting methods. Left: original image of aggregation. Middle: splitting using watershed on input image. Right: shape smoothing followed by splitting based on concavity points.

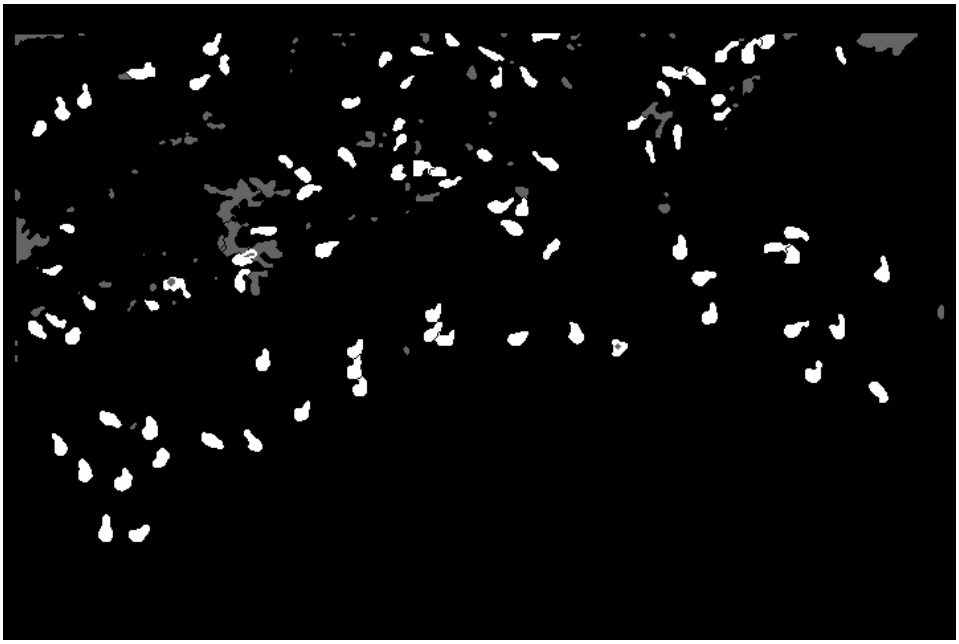


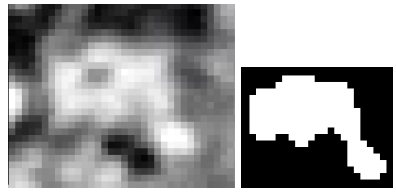
Figure 6.12: Example of classification on double threshold segmentation. Positives are white and negatives are grey.

6.2.4 Classification

As explained in earlier sections, the segmentation outputs a binary image representing the segmented objects, which is what is used for classification. Values such as max/min length and area, used for filtering the segmented contours, are determined by the camera's properties and height. Additionally, a fixed value is used to reject objects that are too concave. This was set to a threshold of 0.7, meaning that the object's area must cover at least 70% of the object's convex hull area.

The classification process assumes that the height parameter is correct and that the output from the segmentation process resembles the actual shape of the object. As discussed in Section 6.2.2, the double threshold segmentation will in some cases only segment parts of the objects, and in other cases, sheep and background clutter are segmented together as a single object. Both cases often result in the object being rejected by the classification, either due to the size, or an irregular shape. When blob detection is used, this problem appears to be less frequent.

Figure 6.12 shows the effect of classification on the double threshold output after watershed splitting has been applied on aggregations. The white objects are the ones that have been classified as sheep, while the grey objects are the rejected ones. Although the classification algorithm used is relatively simple, it significantly reduces the number of false positives. Still, there are several cases where it fails. One example is shown in Figure 6.13.



(a) From input image **(b)** Segmentation

Figure 6.13: Example of a false positive.

The main problem is that algorithm used only analyses the binary contour of the segmented object. This does not tell anything about the texture structure or intensity. One common characteristic of the sheep thermal signatures is that the intensity is relatively even across the entire surface. By analyzing the texture of the objects, the classification accuracy can be improved. Using the object contour as a mask, the texture can be extracted exclusively from either the original image or the opening by reconstruction top-hat, to delimit the analysis area.

Using the numbers from Table 6.1 where a T-factor of 2 was used, 90 out of 105 objects that were classified as sheep were classified correctly. This gives an accuracy of 85.7%. The result shown in the top row of Table 6.2, shows a similar result, where 89 of 105 objects were classified correctly, giving an accuracy of 84.7%.

6.2.5 Detection Results

In this section, a series of figures showing the final result of the sheep detection algorithm using the values discussed above is presented. For every figure, a white rectangular bounding box and circle is used to mark potential sheep detected by the algorithm. Rectangular bounding boxes represent detected sheep segmented by the double threshold, while the bounding circle represents sheep detected using blob detection. The black dot is used to mark the objects that were manually annotated. During the annotation, only the objects that we were absolutely sure was a sheep were marked.

Figure 6.14 shows an image of a sheep on open farmland. The blob detection was able to segment the sheep. This is because the sheep has a circular shape with strong edges, and there is a sharp contrast between background and sheep. Detection using double threshold did not manage to detect this sheep, this is because there are other brighter objects in the image, which are segmented out instead of the sheep. These can be seen in the upper left of the image, and the bottom of the left edge. Note that the classifier filtered out these objects, so no false positive detections were made. For this particular case, increasing the T-factor will fix the problem, but this adjustment will affect the overall performance on the rest of the dataset.

Figure 6.15 shows the results of an image containing multiple sheep and a varying amount of background clutter. The detection rate using double threshold resulted in 77/88 true

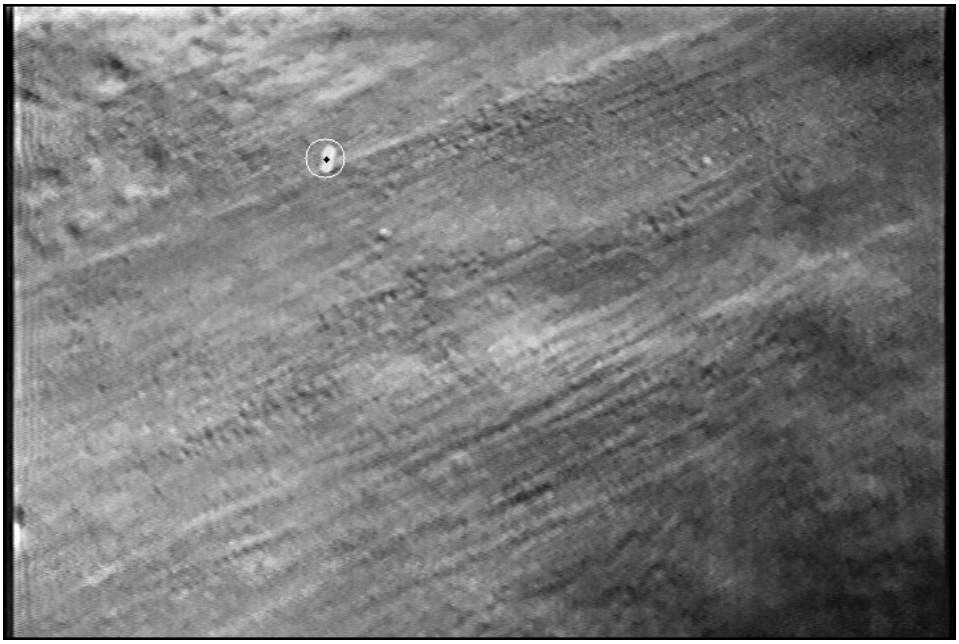


Figure 6.14: White bounding box: double threshold, Black bounding box: blob detection

positives and 17/94 false positives, while the detection rate for blob detection resulted in 75/88 true positives and 11/86 false positives. In the image not all sheep were annotated, this is because some sheep had too much background noise which made it hard to classify them as either sheep or background. These cases include; (1) large groups of sheep that are close to each other, which can be seen in the image as a single large white region. (2) Sheep that are located on the heated ground surface. In these cases, the sheep blends in with its surroundings making it hard to see the boundaries of the sheep clearly. For both segmentation methods, all of the false positive detections were made in the upper part of the image, where most of the background noise resides. The false positive figures are in reality a bit lower than whats measured, as some of them are sheep that were, due to uncertainty, not marked during the manual annotation.

The marked region, 1, shows a group with an uncertain number of sheep tightly clustered together. In this case, the sheep shapes blend into each other and form a large white irregularly formed shape, which is difficult to handle. Only the double threshold segmentation managed to segment the entire group. However, the segmentation splitting failed in most cases. The blob detection only segmented some of the sheep located in the outer part of the group, as these had more clearly defined edges.

The sheep marked as two shows a case were the blob detection was able to detect a sheep while double threshold was not. This is because the sheep is darker than the other sheep, which resulted parts of the sheep, the head, was left out by the segmentation process. This can be seen in Figure 6.12. Even if the sheep is darker than the other sheep, the blob

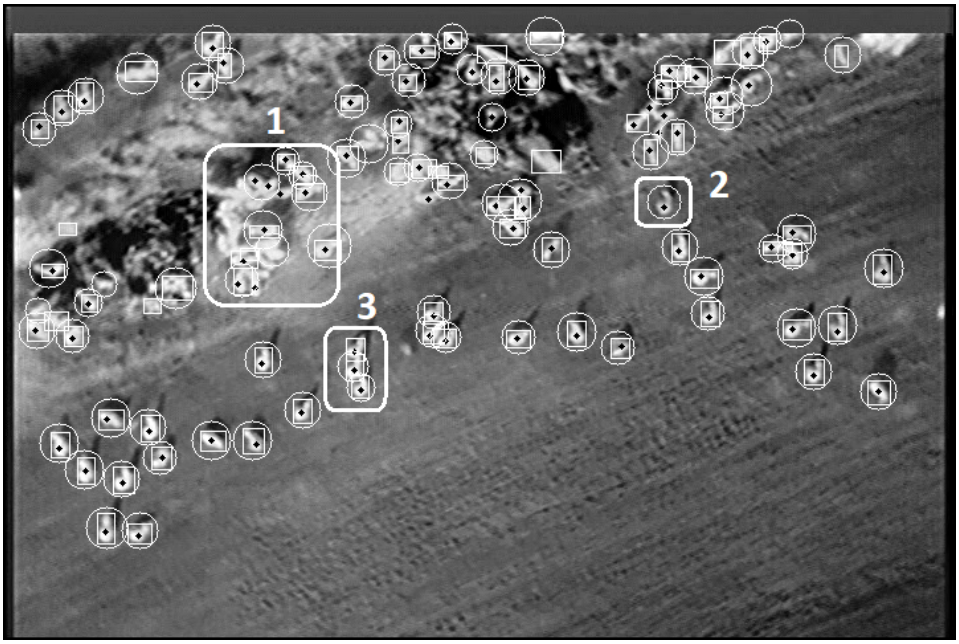


Figure 6.15: White bounding box: double threshold, Black bounding box: blob detection

detection was still able to detect it because it still has clear edges. This is one of the advantages of using blob detection over the double threshold. Another thing to notice is that all detection done by the blob detection has an oval shape, while double threshold has a bit more varied shapes.

In 3 a group of three sheep is close to each other. They appear as an extended vertically connected white region. This group is easier to split compared to the previously discussed group. Blob detection automatically splits the sheep as it only extracts a blob with the max size of a sheep. This works when the marker is in the middle of the sheep, but if the marker is located between two sheep, it does not work as it will grow and segment out half of both sheep. This can be fixed by increasing the max perimeter size during the blob segmentation. The sheep closest to the bottom of the image shows the former case while the upper two sheep shows the latter case. This resulted in only 2 of 3 got detected. Compared to the double threshold which was able to detect and split all 3 using watershed.

Figure 6.16 and 6.17 shows two examples of a case with no sheep and a high amount of background noise. In the case shown in Figure 6.16, double threshold managed to produce no false positive detections, this is because there are no bright objects that fit the size requirements. On the other hand, blob detection resulted in one false positive detection. This is because the detected object has defined edges, generating at least 6 spoke directions. A higher requirement for spoke directions, such as 7 or 8, would likely eliminate this false positive. Figure 6.17 shows another example containing a higher amount of small bright

objects. In this case, the double threshold produced one false positive. At first glance the detected object looks like a sheep, but if one study it further it is more likely an object heated by the sun. Blob detection produced four false positives. Overall both of the methods produced good results, considering the high amount of background noise produced by sun radiation. We can see that the sun radiation contributes to heating the ground surface as there are many dark regions in an image created by shadows. As mentioned before, by doing the flight during night time or before sunrise will reduce the background clutter significantly. By looking at the shadows in Figure 6.17, particularly in the bottom right corner, the effect of the sunlight is evident. In the darkest areas, there are much fewer details than in the surrounding areas. Note, however, that some of this effect is also caused by the gain settings on the camera.

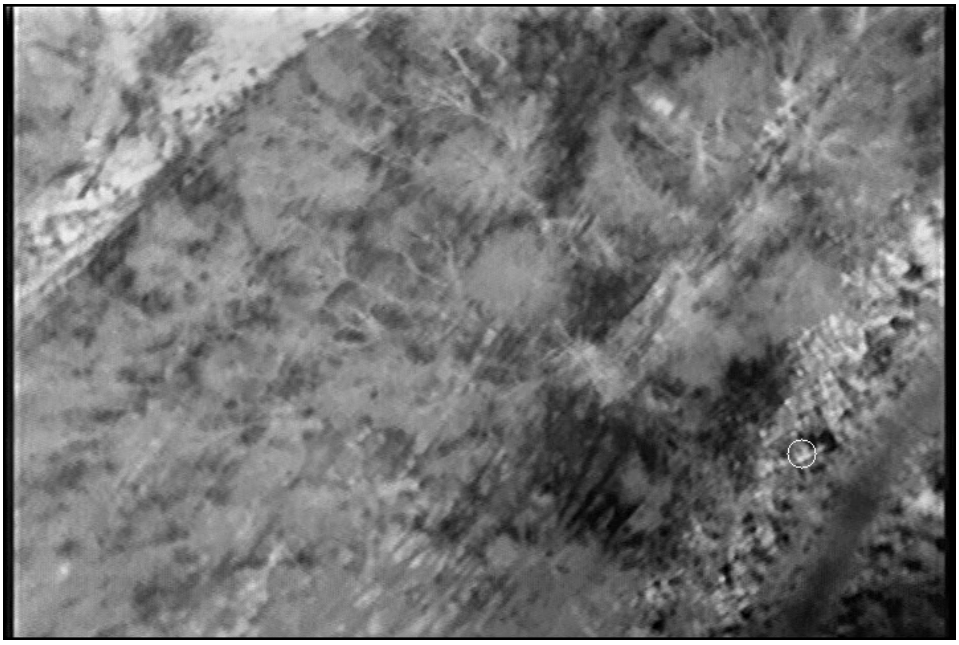


Figure 6.16: White bounding box: double threshold, Black bounding box: blob detection

Detection in forests

Figure 6.18 shows three wild boars in a pine-dominated forest, with moderate canopy cover captured. This image was captured at 22:55 [46]. Both methods managed to detect 2 of 3 wild boars. In case of blob detection, the third wild boar had too weak edges to be detected. In case of double thresholding, the surroundings were too bright, and the opening by reconstruction top-hat failed to strengthen its intensity sufficiently. As this method segments based on intensity levels, the darker target did not get segmented. However, the results show that the algorithm is capable of detecting objects in a forest with a moderate canopy in certain cases.

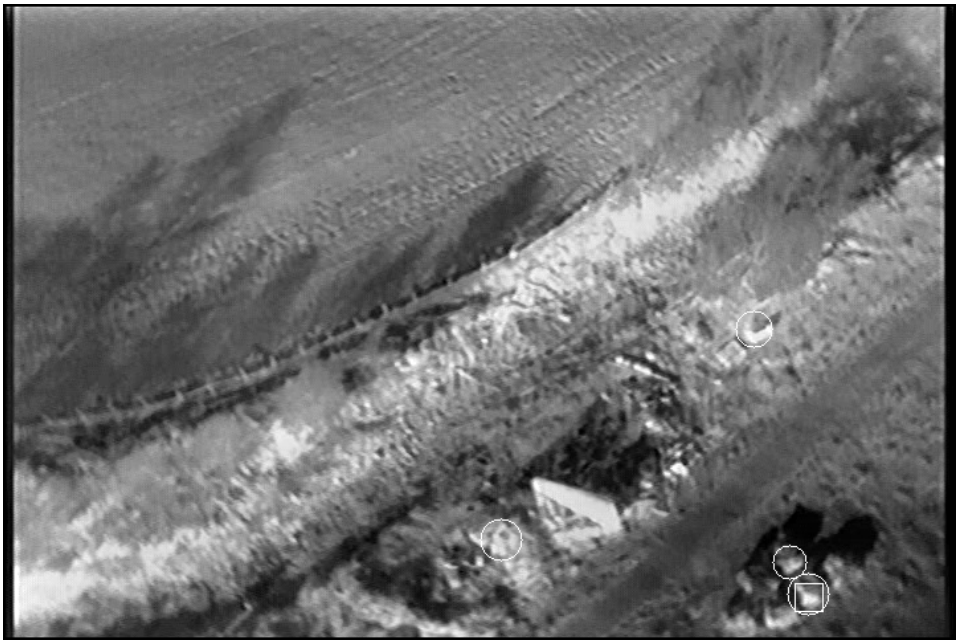


Figure 6.17: White bounding box: double threshold, Black bounding box: blob detection

Figure 6.19 shows another example taken from [45]. The image shows animals obscured by tree canopy in coniferous forest [46] captured at 19:40. Only 3 out of 5 animals were visible in this image. The missing two animals was obstructed by trees. Both segmentation methods were able to segment all three visible animals, and all were classified correctly. No false positive detection was detected.

Figure 6.20 shows a harder case where tree branches obstruct some of the animals. Comparing the results, double thresholding produced better results. This is because the shape of these animals appears more like a long rectangle instead of a blob. The fact that they are relatively narrow also reduces the chance of finding adequate spoke directions. No false positives were generated.

6.2.6 Image Sequence

The previous section looked at how the algorithm performed in isolated images. This section discuss how they perform on a sequence of images. Because of the run-time constraints of the blob detection, only double threshold was tested. The videos used was; one from the main dataset captured at 50m altitude 1.1 and two from [45]. For each video, the sheep detection algorithm using double threshold was used to detect sheep. The subsequent figures show the most important frames from each video.

Figure 6.21 shows the most essential frames from applying the algorithm on a video from

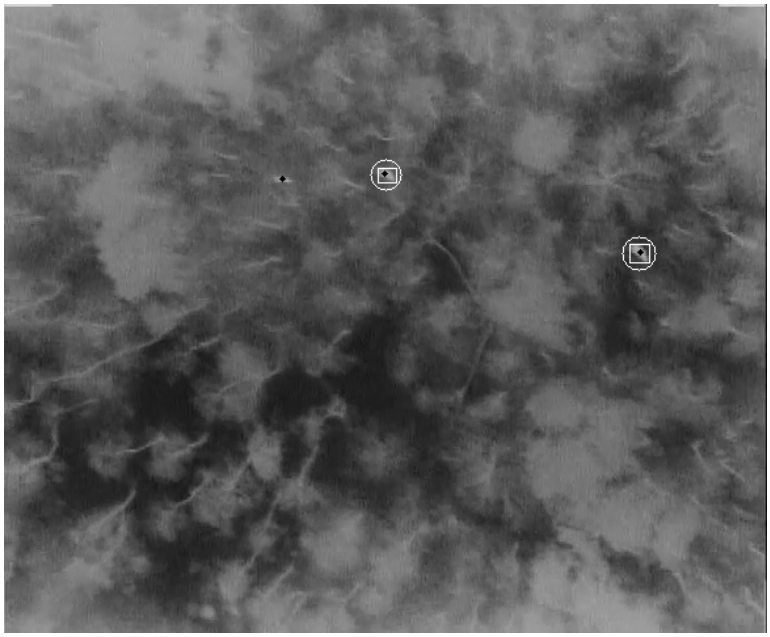


Figure 6.18: White bounding box: double threshold, Black bounding box: blob detection. Images from [45]

the primary dataset, introduced in Chapter 1.1, containing a significant number of sheep. The first three image sequence shown demonstrate how the $expected$ value performs. The expected value started at 1 and grew as more sheep got detected. When more sheep left the image the expected value decreased. This worked very well and provided good results. Because of noise, the expected value was kept for at least three consecutive frames before decreasing it, if less sheep were detected. Looking closely at the image sequence, the objects that look similar to a sheep gets frequently detected, while false positive objects only get detected a few times. Even though not all sheep were detected in a single image, they got detected throughout the image sequence. In a real scenario, a sheep would appear in dozens of pictures before exiting the view, giving multiple opportunities for detection. Because of this, it is more important to adjust the algorithm to minimize false positive and assume that the sheep is detected in at least one of the frames.

Figure 6.22 shows the most essential images resulting from applying the sheep detection algorithm on a video from [45]. This sequence demonstrates how the algorithm performs in a setting where targets are obscured by a tree canopy in a coniferous forest [46].

A square represents a detected target using the algorithm. The undetected targets are manually marked using a circle, to make it easier to follow their movements. Every target is also given a unique identifier. As shown in the sequence, the algorithm manages to detect every target except for target no. 4. This is because this target was partially covered by a tree in all frames, and it also appears a bit darker compared to the other targets.

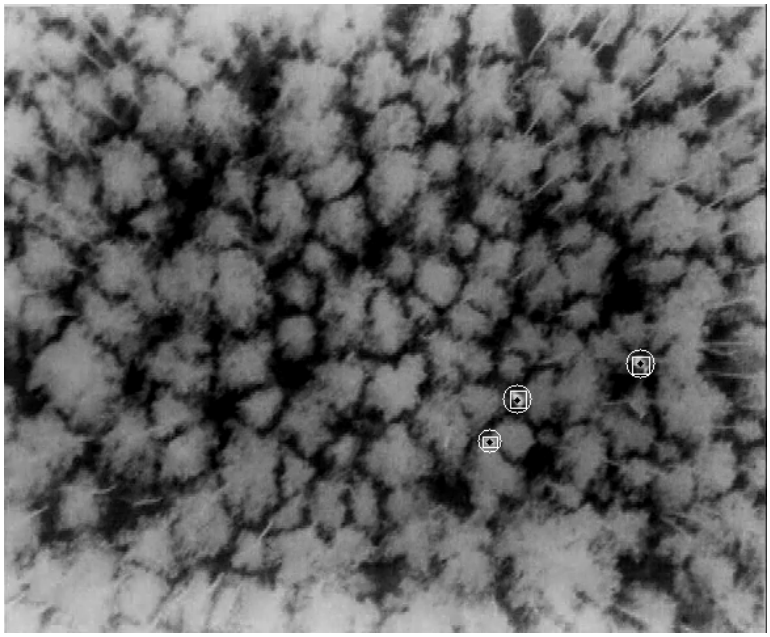


Figure 6.19: White bounding box: double threshold, Black bounding box: blob detection. Images from [45]

Throughout the video, there were occurrences of false positive detection, but as in the previous cases they only got detected once or twice, while the true targets were frequently detected.

Figure 6.23 shows an image sequence of a different video from [45]. The video was recorded during sunrise in a beech forest, with a full canopy cover [46]. In this case, two of the animals, marked 1 and 3 in the first frame, were detected. The beech canopy appears to add a fog-like effect to the images, reducing the contrast between the animals and ground surface. However, unlike the coniferous canopy, it is possible to partly see through the beech canopy, as long as it is leafless. The third animal (number 2) appeared too weak to be fully segmented in any of the frames. This case is difficult to spot even with the human eye and is therefore not considered a limitation of our algorithm. The result also shows that the algorithm is able to detect moving targets without problems. From the sequence, one can see that one of the animals is moving towards the bottom edge of the view. If individual object tracking were to be implemented, these situations would have been taken into consideration, as the object does not appear on the same location in the scene between each frame. Despite the low contrast and tree trunks and branches that are easy to confuse with the animals, no false positives were detected in this sequence.

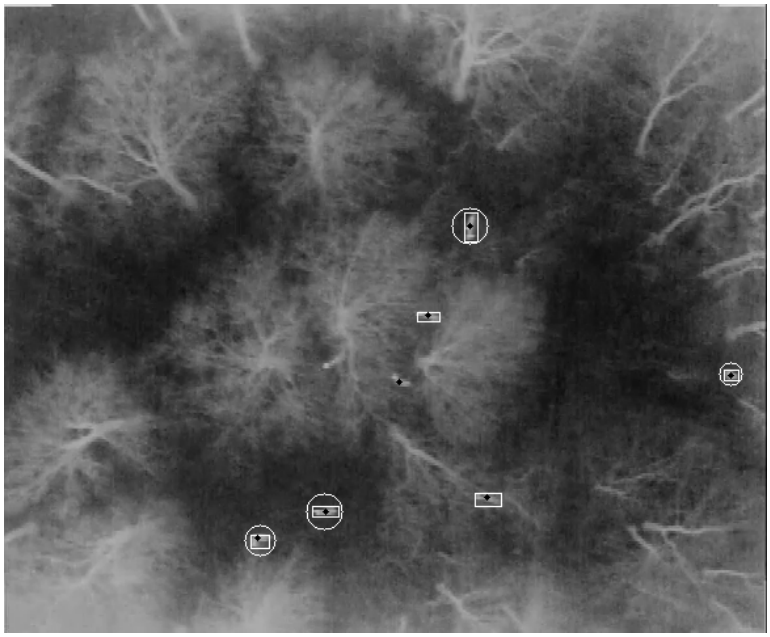


Figure 6.20: White bounding box: double threshold, Black bounding box: blob detection. Images from [45]

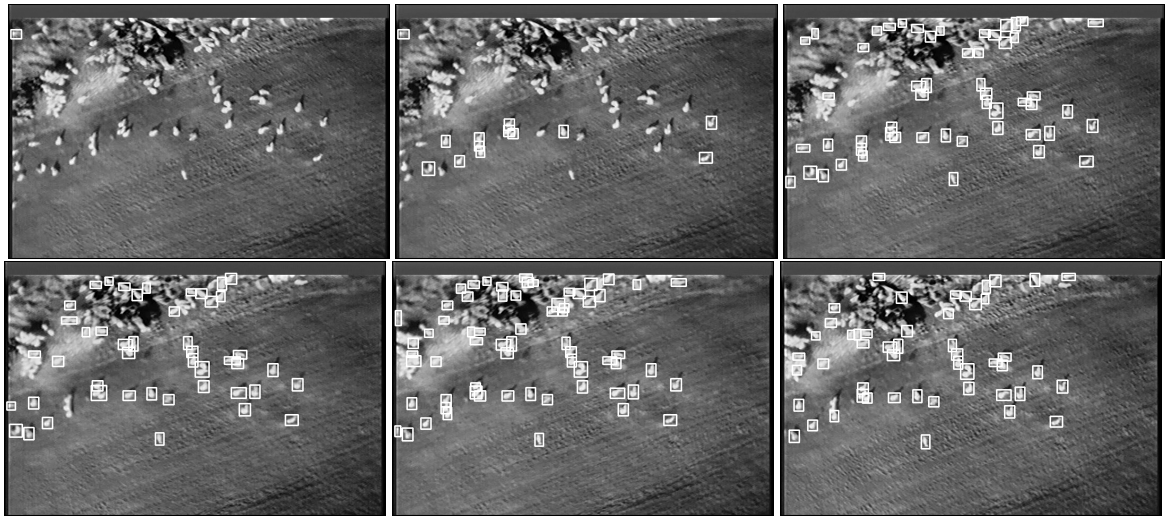


Figure 6.21: Image sequence from sheep 50m.

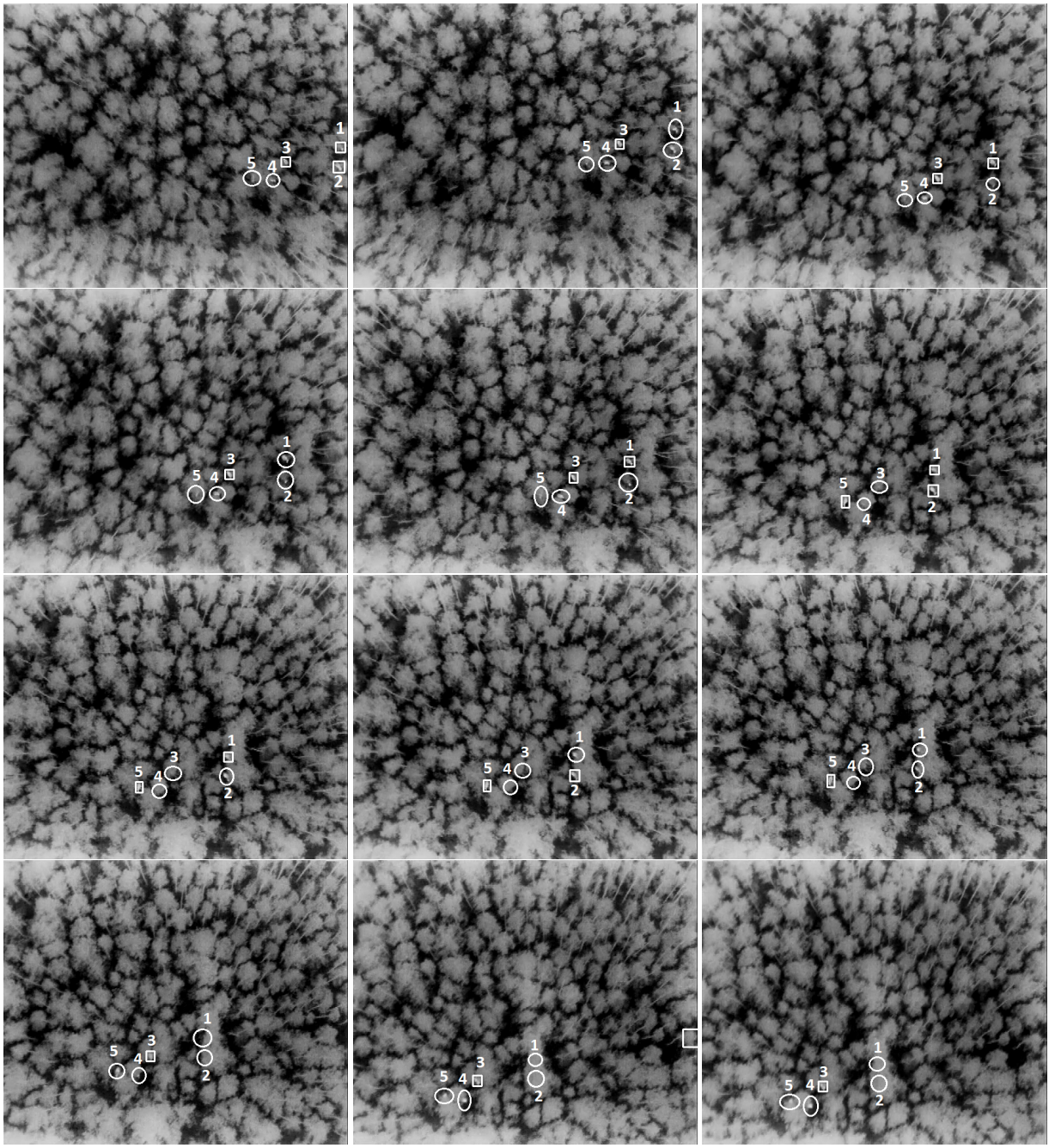


Figure 6.22: Coniferous forest, 19:40 during sunset [46]. Images from [45]

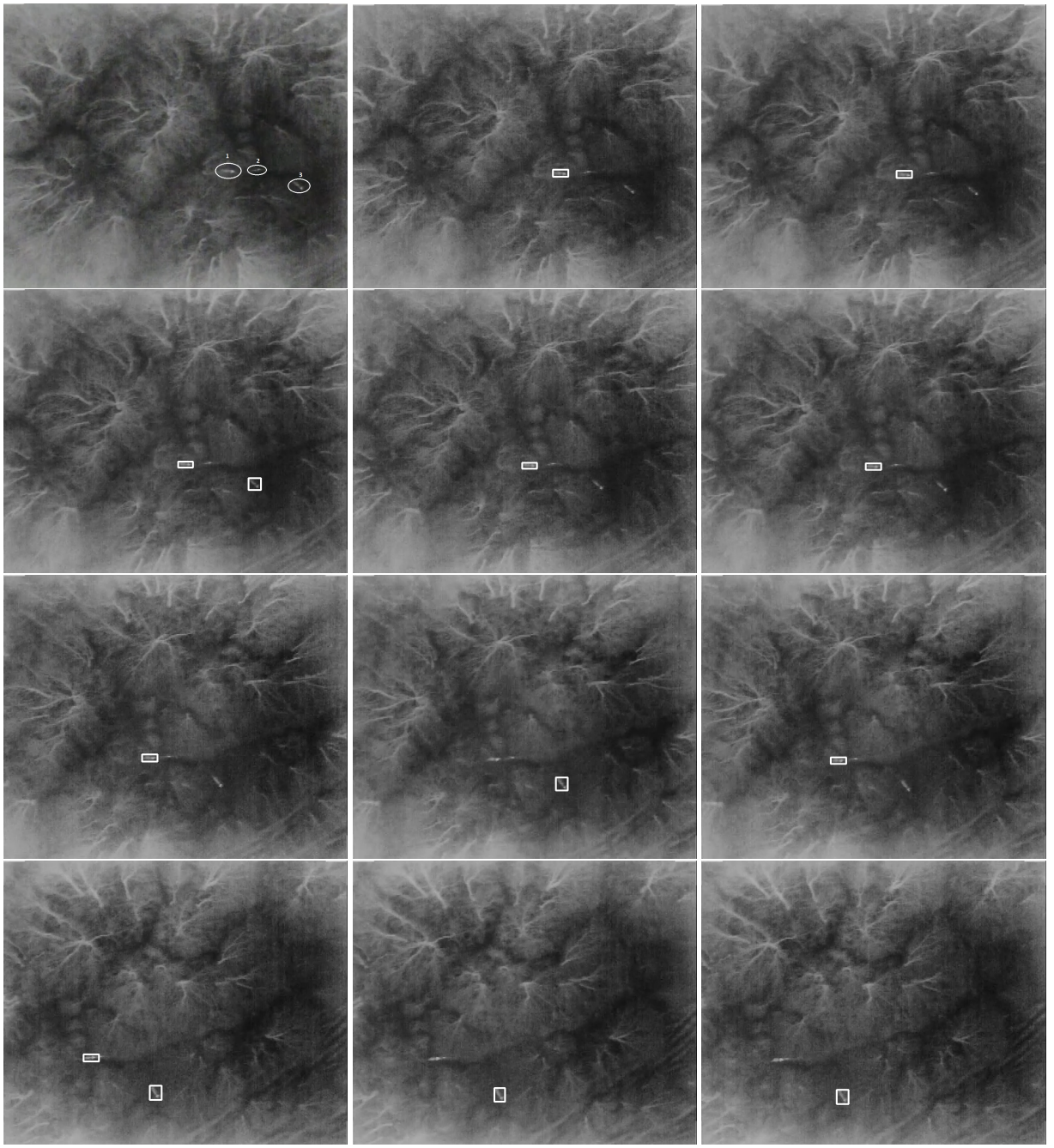


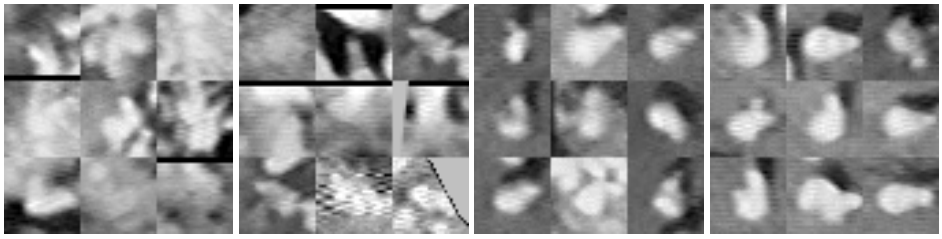
Figure 6.23: Beech forest, 06:45 during sunrise [46]. Images from [45].

6.3 Machine Learning

Machine learning requires a massive dataset which is used during the training of the network. In the case of this project, the amount of data (images of sheep) was limited, which is a strong limitation when trying to develop and test machine learning methods. Because of this, only the surface of this topic was scratched, and machine learning was not prioritized. The goal was to test whether it is worth investing time in gathering a bigger dataset and developing a machine learning based sheep detection method.

6.3.1 CNN

CNNs can be used in place of the classical image processing classification, described in 4.2.5. One challenge with the use case in this thesis is the fact that there is only one class of interest. At the time of writing, CNNs for single class classification appears as a relatively uncharted area. Because of this, there was created a dataset containing two classes; "sheep" and "not sheep". The images in the dataset were generated from the true and false positive detections made by the classic computer vision algorithm, and inspected and sorted manually to ensure that the images were correctly labeled as "sheep" or "not sheep". The dataset contained 564 images of sheep and 247 images of not sheep and was divided into two sub-datasets; one used for training and one for testing. The training dataset contained approximately 75% of the unique sheep, while the test sets contained the remaining 25%. In the dataset, some sheep appeared in multiple images with slight variations, so unique sheep means that reappearing sheep were counted as a single sheep. This was done to prevent the network from training and testing on the same objects, which could look identical.



(a) Training not sheep **(b)** Test not sheep **(c)** Training sheep **(d)** Test sheep

Figure 6.24: Example of training and test images

In the generated dataset, all images, regardless of quality, from the two classes were used. For the "sheep"-class, this included both images containing clearly distinct sheep shapes on a plain dark background, and more irregularly shaped sheep with a cluttered background. For the "not sheep"-class, any extracted image not containing sheep were included. Figure 6.24 shows different portions of the dataset.

The CNN has a set of hyperparameters that are adjustable. These include; the size and number of filters in each convolutional layer and the window size and the stride used during

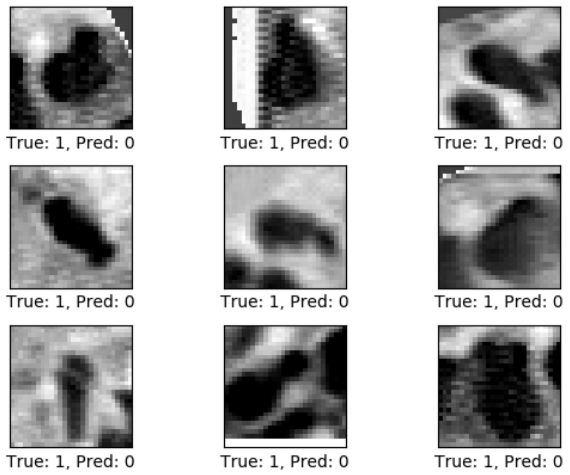


Figure 6.25: Example of false detection. The two classes sheep and not sheep is represented using 1 and 0 respectively

max pooling. The CNN used was configured with a 3x3 filter size for both convolutional layers. The first layer had 16 filters while the second layer had 32 filters. A window size and stride of 2x2 was used for max pooling.

After training the CNN model, it was able to classify objects with an accuracy of 89-94%, varying between each time it was trained. The result is surprisingly good, considering the small dataset. In comparison, the classical computer vision classification had an overall score of 85.7% on the same images that the dataset for the CNN was extracted from. However, these numbers do not necessarily resemble the results in a real scenario, as the test set and training set was taken from very similar environments. Also, the model was only tested on a small portion of the dataset. Still, it was able to distinguish between images containing sheep with a cluttered background, and images of only cluttered background. However, CNN was not implemented in the final algorithm, as it is uncertain how well it performs on other datasets than the one it was trained on. In comparison, the implemented classical algorithm produces overall good results for any situation.

The results of the CNN, indicates that replacing the existing classification with CNN, is a realistic option. This implies using one of the discussed segmentation methods, like before, then cropping the segmented objects from the image and running CNN to classify them.

6.3.2 RetinaNet Object Detection

The training and test dataset were created using the extracted still images. Because the datasets contained multiple images of the same sheep taken at slightly different angles, we divided the training and test set based on unique sheep. Since the RetinaNet is not

rotation irrelevant, all the training data was rotated from 0 to 360 degrees using 30 degrees increments.

Figure 6.26 shows the result of the RetinaNet trained for 15 epochs using the dataset mentioned above. From the results, we can see that the performance is not too good. This is because not enough data was provided, resulting in a prediction model that only handles a few scenarios of sheep appearance. The sheep inside the marked area are the test sheep (these sheep has not been trained on). The sheep outside are the training sheep used during the training of the prediction model.

Training the model for 15 epochs was optimal, as training the model further resulted in overtraining. Overtraining occurs when the prediction model is trained so much that it is only able to detect the exact sheep in the training set, resulting in the model not being able to detect sheep with a slight variation.

6.3.3 Summary

Automatic sheep detection using classic image processing showed best results. The algorithm proves to work well in areas covered by dense coniferous and deciduous forests, as well as open areas. Although not all animals in the video samples from [45] are detected in every frame, most animals that are clearly visible are detected in at least a few frames during the pass of the UAV, while most false positives only appear randomly for one or two frames. Classifying objects based on the number of reoccurring detections is suggested to give a low false positive rate. Using CNN for object classification produced better results than the classical classification methods and should be tested in future work. With the available dataset, RetinaNet was not able to reach any usable results, even with simple data augmentation.

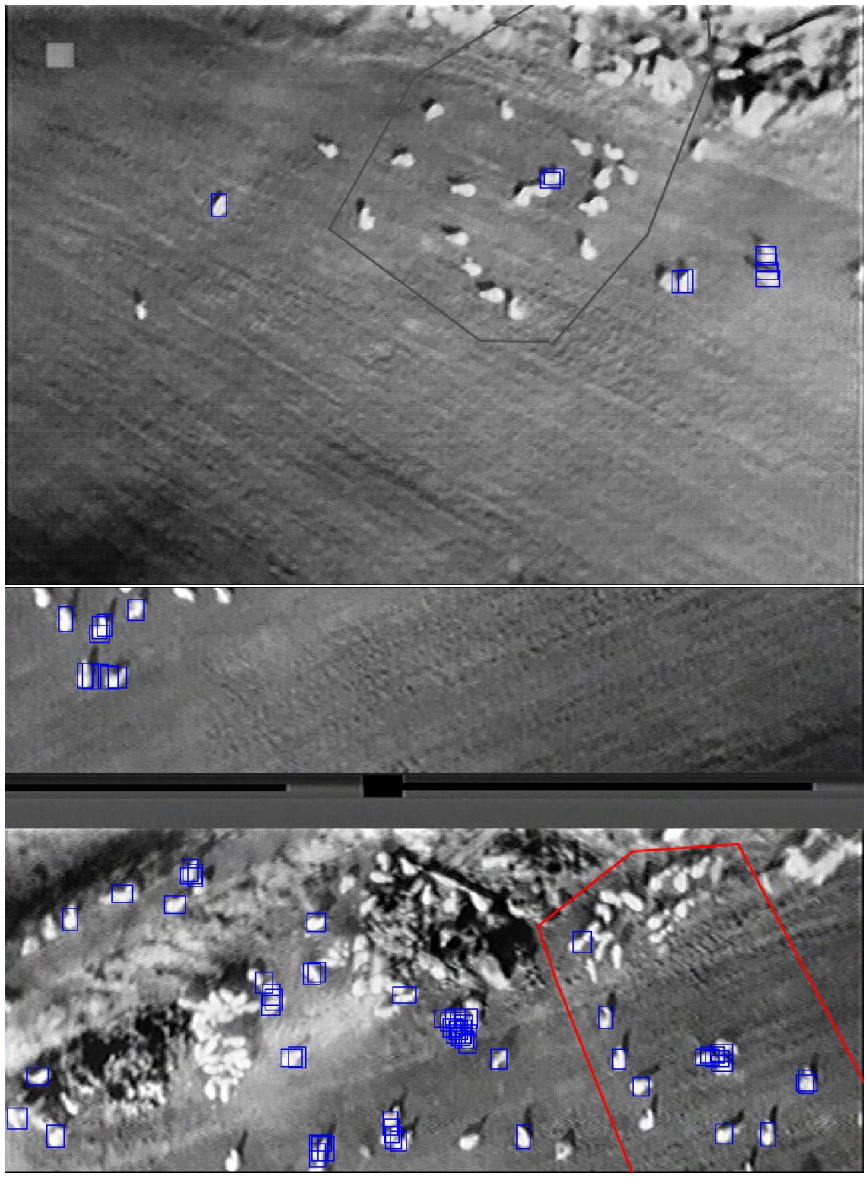


Figure 6.26: Example of prediction results after 15 epochs

Chapter 7

Future work

7.1 Sheep Detection

Several areas can be further explored to improve the results of the object detection. A good start would be to collect more images of sheep in different environments, at different times of the day.

7.1.1 Segmentation

Both segmentation methods were implemented as a proof of concept and have a great potential for optimization. Particularly, the blob detection algorithm can be optimized to run faster.

If a camera with thermographic capabilities is used, the process of segmenting objects can be done based on a constant temperature range. In our case, the pixel values did not correspond to the emitted temperatures, as the camera was configured to use automatic gain control, resulting one level of intensity representing different temperatures from one frame to another.

Aggregation splitting

The marker-based watershed algorithm was found to be an accurate method of splitting aggregations. The difficult task, however, was finding a good marker for each sheep in the aggregation. This is a topic that should be further explored.

7.1.2 Object Classification

Minimum number of detection

As presented in 6.2.4, when testing our algorithm on the video samples in [45], most objects were detected several times during the frame sequence, and false positives were usually only detected once. However, the same object was rarely detected consecutively for more than two frames. Instead of requiring a continuous track, a loss of track can be accepted by instead counting the total number of times each object is detected. Optical flow tracking [30] can be used to predict where the lost object will appear next time it is detected. Functionality for this is already available in OpenCV [16], and was conceptually tested for avoiding double counting of sheep. It was however not implemented in the final solution, as it was not considered needed. This method should be effective for eliminating false positives, which could mean that the sensitivity of the segmentation can be increased, for example by increasing the T-factor.

Classification using heat signature

Another feature to analyze if accurate temperature information is available is the way that the temperature changes towards the center of the object. For this, a method similar to the one used in [33] could be tested. This method uses an expected temperature curve for each class, together with the k-nearest-neighbor algorithm to classify objects.

Convolutional Neural Networks

Both CNN and Keras RetinaNet has potential if enough training data is collected. Some basic training augmentation was used to improve the results, but the effect was minimal. More advanced data augmentation can be tested to get better results. A Python library called `imgaug`[1] converts a set of input images to a new set of altered images, using multiple augmentation methods such as rotation, sharpening, blurring, contrast adjustments, flipping and different types of noise generation.

Mapping warm terrain objects

Another possible approach to reduce the number of false positives is to maintain a registry of objects that tend to appear as false positives during the searches. Because false positives normally remain stationary, doing multiple flights over multiple days, these objects can be identified. Farmers search the same areas each year, and over multiple iterations the number of false positives can be reduced using the registry.

[1] `https://github.com/aleju/imgaug`

Combine with color imaging

By combining FLIR imaging and color imaging, more information, such as texture and color can be extracted from each object and used for classification.

7.2 Route Generation

7.2.1 Reducing Transition Distances

A more sophisticated algorithm for finding the shortest travel path between sub-polygons.

7.2.2 Vegetation

More GIS data can be used to handle other types of ground surfaces, such as forest areas. In case of forests, increasing the sweep overlap can increase the detection rate.

Chapter 8

Conclusion

The objective of this thesis was to develop a tool specifically targeted towards the use of UAVs and thermal imaging.

In our first research question, we asked how UAV can be used to survey an area for missing sheep. Our approach was to use a fixed-winged UAV with a forward mounted infrared thermal camera. An algorithm for generating a flight path suitable for surveying an area for missing sheep was developed. This path can be exported and uploaded to existing autopilot systems. It was found that the tool produced more efficient routes, in terms of flight distance than other freely available tools. Omitting lakes was efficient only for larger sized lakes.

The second research question was targeted towards the image processing, where we asked how it could be used and if it is reliable enough to help in a real scenario. We developed and tested three methods for detecting sheep in FLIR images. After optimizing and testing the three methods, it was found that the method based on classical computer vision is the best in terms of accuracy, and would be sufficient for detecting potential sheep in a real scenario. The algorithm was found to work well also in forests with dense canopy coverage.

Bibliography

[1] Apm planner 2 - home. http://ardupilot.org/planner2/. Accessed: 2018-03-20.

[2] A beginner's guide to understanding convolutional neural networks. https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/. Accessed: 2018-06-20.

[3] Chai assertion library. http://www.chaijs.com/. Accessed: 2018-06-15.

[4] Electron. https://electronjs.org/. Accessed: 2018-06-15.

[5] Fanesak tap av sau på beite. https://www.dyrebeskyttelsen.no/tap-sau-pa-beite/. Accessed: 2018-01-15.

[6] How to make a tensorflow image classifier. https://github.com/llSourcell/How_to_make_a_tensorflow_image_classifier_LIVE/blob/master/demonotes.ipynb. Accessed: 2018-06-19.

[7] Icenet dekningskart. https://www.ice.no/private/coverage. Accessed: 2018-01-15.

[8] Keras retinanet. https://github.com/fizyr/keras-retinanet. Accessed: 2018-01-15.

[9] Litt om sauedrift. http://www.trollsteingaard.no/sauedrift.htm. Accessed: 2018-01-15.

[10] Matplotlib: Python plotting. https://matplotlib.org/.

[11] Micropilot - home. https://www.micropilot.com. Accessed: 2018-06-16.

[12] Mission planner - home. http://ardupilot.org/planner/index.html. Accessed: 2018-03-20.

[13] N50 kartdata. https://www.kartverket.no/data/Last-ned-norgeskartet-som-database/. Accessed: 2018-04-12.

[14] Nodejs. https://nodejs.org/. Accessed: 2018-06-15.

[15] Ntnu unmanned aerial vehicles laboratory. https://www.itk.ntnu.no/english/lab/unmanned. Accessed: 2018-06-22.

[16] Opencv: Optical flow. https://docs.opencv.org/3.3.1/d7/d8b/tutorial_py_lucas_kanade.html.

[17] Px4 pro drone autopilot - home. http://px4.io/. Accessed: 2018-06-16.

[18] Python. https://www.python.org/. Accessed: 2018-06-21.

[19] React. https://reactjs.org/. Accessed: 2018-06-15.

[20] Redux. https://redux.js.org/. Accessed: 2018-06-15.

[21] Telenor asa dekningskart. https://www.telenor.no/privat/dekning/#map. Accessed: 2018-01-15.

[22] Telespor as - home. https://telespor.no/. Accessed: 2018-06-16.

[23] Telia dekningskart. https://telia.no/dekningskart. Accessed: 2018-01-15.

[24] Tensorflow. https://www.tensorflow.org/.

[25] webpack. https://webpack.js.org/. Accessed: 2018-06-15.

[26] Welcome to google maps platform. https://cloud.google.com/maps-platform/. Accessed: 2018-06-15.

[27] ArduPilot. Ardupilot open source outopilot. http://ardupilot.org/, 2016.

[28] L. Asheim and I. Mysterud. The norwegian sheep farming production system. *Options Méditerranéennes. Série A: Séminaires Méditerranéens (CIHEAM)*, 1999.

[29] S. Bambach. A survey on recent advances of computer vision algorithms for egocentric video. *arXiv preprint arXiv:1501.02825*, 2015.

[30] J.-Y. Bouguet. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation*, 5(1-10):4, 2001.

[31] U. M. Braga-Neto, M. Choudhury, and J. I. Goutsias. Automatic target detection and tracking in forward-looking infrared image sequences using morphological connected operators. *Journal of Electronic Imaging*, 13(4):802–814, 2004.

[32] L. Chrétien, J. Théau, and P. Ménard. Wildlife multispecies remote sensing using visible and thermal infrared imagery acquired from an unmanned aerial vehicle (uav). *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 40(1):241, 2015.

[33] P. Christiansen, K. A. Steen, R. N. Jørgensen, and H. Karstoft. Automated detection and recognition of wildlife using thermal cameras. *Sensors*, 14(8):13778–13793, 2014.

[34] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Convolutional neural network committees for handwritten character classification. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1135–1139. IEEE, 2011.

[35] f. W. C. Djexplo [CC0]. Latitude and longitude of the earth, 2018. [Online; accessed 20-June-2018].

[36] M.-A. Favier, R. C. Green, and A. Linz. The potential for uav technology to assist in sheep man-agement in the scottish highlands. 2013.

[37] J. Fernández, B. Tóth, L. Cánovas, and B. Pelegrín. A practical algorithm for decomposing polygonal domains into convex polygons by diagonals. *Top*, 16(2):367–387, 2008.

[38] E. Galceran and M. Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous systems*, 61(12):1258–1276, 2013.

[39] R. C. Gonzalez. *Digital Image Processing*. Pearson Education, Inc., 2008.

[40] B. Hansen. Varmesøkende droner finner sauen. *http://gardsdrift.no*, 2015.

[41] S. Hansen. Drones help find lost sheep, Feb 2016.

[42] A. R. Hevner. A three cycle view of design science research. *Scandinavian journal of information systems*, 19(2):4, 2007.

[43] S.-O. Hvasshovd. Internet of sheep (ios) og litt til. 2017.

[44] C. Indhumathi, Y. Cai, Y. Guan, and M. Opas. An automatic segmentation algorithm for 3d cell cluster splitting using volumetric confocal images. *Journal of microscopy*, 243(1):60–76, 2011.

[45] A. Z. . M. C. Julia Witczuk, Stanisław Pagacz. Exploring the feasibility of unmanned aerial vehicles and thermal imaging for ungulate surveys in forests - preliminary results. *International Journal of Remote Sensing*, 2017.

[46] A. Z. . M. C. Julia Witczuk, Stanisław Pagacz. Exploring the feasibility of unmanned aerial vehicles and thermal imaging for ungulate surveys in forests - preliminary results. *International Journal of Remote Sensing*, 2017.

[47] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*, 2017.

[48] L. G. Minor and J. Sklansky. The detection and segmentation of blobs in infrared images. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(3):194–201, 1981.

[49] OpenCV. Opencv library. `https://opencv.org/`, 2018.

[50] G. Öst. Search path generation with uav applications using approximate convex decomposition, 2012.

[51] A. Seymour, J. Dale, M. Hammill, P. Halpin, and D. Johnston. Automated detection and enumeration of marine wildlife using unmanned aircraft systems (uas) and thermal imagery. *Scientific Reports*, 7, 2017.

[52] R. Stølsmark and E. Tøssebro. Reducing energy consumption in a sheep tracking network using a cluster-based approach. *Proc. The 6th Int. Coif. Sensor Technologies and Applications, 20 1 3*, pages 129–135, 2012.

[53] M. Torres, D. A. Pelta, J. L. Verdegay, and J. C. Torres. Coverage path planning with unmanned aerial vehicles for 3d terrain reconstruction. *Expert Systems with Applications*, 55:441–451, 2016.

Appendix

Appendices

A Test Images