Marit Gjøstøl Ytterland
Tone Kathrine Ervik Winsnes

Retrieval of Sheep Using Unmanned Aerial Vehicles

Master's thesis in Computer Science
Supervisor: Svein-Olaf Hvasshovd
May 2019

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Kunnskap for en bedre verden

Marit Gjøstøl Ytterland
Tone Kathrine Ervik Winsnes

Retrieval of Sheep Using Unmanned Aerial Vehicles

Master's thesis in Computer Science
Supervisor: Svein-Olaf Hvasshovd
May 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

Farmers spend a lot of time every fall collecting sheep from the mountains. This process needs to be done more efficient using modern information technology, as existing solutions are not satisfactory. In this paper we propose a system analyzing images taken with an Unmanned Aerial Vehicle(UAV) to detect sheep-like objects, and this system could be used with any off-the-shelf UAV. We present different solutions for object detection using the OpenCV library with traditional computer vision methods and image processing that can be helpful in retrieving sheep more efficiently. In these solutions we study the outcome of applying different filters and thresholding methods to the images, as well as detecting colours and contours. We also discuss the positive aspects and drawbacks of the solutions, and compare the effectiveness of the best solutions. More specifically we compare the amount of sheep found with the total amount of sheep in the images. The results show that detecting non-white sheep is difficult, especially concerning the high amounts of false positives. By combining one of the most successful solutions with a thermal camera, one could achieve satisfactory results when detecting sheep.

Sammendrag

Bønder bruker mye tid hver høst på å hente inn sauene sine fra fjellet. Denne prosessen bør effektiviseres med moderne informasjonsteknologi, da eksisterende løsninger ikke er tidsmessig tilfredsstillende. I denne artikkelen presenterer vi et system som analyserer bilder tatt med en Unmanned Aerial Vehicle(UAV) for å detektere saueliknende objekter, og dette systemet kan brukes med hvilken som helst UAV. Vi legger fram forskjellige løsninger for objektdeteksjon ved å bruke OpenCV sitt bibliotek med tradisjonelle datasynmetoder og bildeprosessering, som kan være nyttige til å finne sau på en mer effektiv måte. I disse løsningene studerer vi utfallet av å bruke ulike filtre og tersklingsmetoder på bildene, i tillegg til detektere farger og konturer. Vi diskuterer også de positive aspektene og ulempene med de tre løsningene, og sammenligner effektiviteten til de beste løsningene. Mer spesifikt sammenligner vi antallet sauer metodene finner med det totale antallet sauer i bildene. Resultatene viser at deteksjon av ikke-hvite sauer er vanskelig, spesielt på grunn av store mengder falske positive. Ved å kombinere en av de mest suksessfulle metodene med et infrarødt kamera, vil man kunne oppnå et tilfredsstillende resultat når man detekterer sau.

Problem Description

Finding sheep at the end of summer that have been grazing freely in the mountains, is a lengthy manual process. It often takes the farmers several weeks to find them, and even then not all are necessarily found. Previous research has indicated that it is possible to use traditional computer vision methods in combination with a UAV (Unmanned Aerial Vehicle) to search for such sheep. Firstly, computer vision methods have to be developed and tested on images with sheep, to find out if they produce satisfactory results.

Preface

This paper has been a master thesis at NTNU as part of the M.Sc. programme in Computer Science during spring 2019, and is a continuation of a Project Assignment [30] written by us during fall 2018. It has been supervised by Professor Svein-Olaf Hvasshovd.

We would direct a significant thank you to farmer Steingrim Horvli who let us come to his farm in Oppdal to acquire our dataset, to Frank Lindseth who lent us his UAV, and last, but not least, a huge thank you to our supervisor for inspiration and guidance throughout this project.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

In Norway, sheep farmers let their sheep graze freely in the mountains during summer. They release them some time in late May and collect them again during the fall. During the retrieval 5-6% of the sheep never return [1], and only one of many reasons for this is predators. This means that one of the other reasons sheep never come back is because the farmers cannot find them.

In other words thousands of sheep are never found in time for winter. Sheep with less than 10 cm of fleece does not tolerate the wet and cold winters in Norway [2]. This is not only a financial problem for the farmers, but also an ethical issue. In addition the farmers get attached to their animals, and consequently do not want to lose them.

Farmers have therefore expressed their need for a system that can help them with the retrieval, ideally using modern information technology in an effective and affordable way. Search in the mountains with quadcopter drones is a possible solution which we will study in this paper.

We will use different computer vision methods to analyze the images to detect sheep. The idea is that we will develop software farmers can use with an off-the-shelf drone, which will make this an accessible solution to their problem. That way the farmers can buy their own drone, with a regular color camera, and apply it with our software.

1.2 Objective

As we developed solutions for finding white sheep in our Project Assignment [30], and the results were relatively good, we now want to move on to the other colors. Therefore, based on what we discussed in the previous section, and based on what we did in our

Project Assignment [30] we have decided on one primary research question that we will try to answer in this paper. We also added two secondary research questions that we will have in mind throughout our research;

Primary research question: How can computer vision methods and image processing be used to detect black sheep-like objects in images captured with unmanned aerial vehicles?

Secondary research question 1: How can computer vision methods and image processing be used to detect brown sheep-like objects in images captured with unmanned aerial vehicles?

Secondary research question 2: How can the solutions from our Project Assignment [30] be improved to get better results when searching for white sheep?

These research questions was decided upon because of our work in our Project Assignment [30] and the fact that far from all sheep are white. Therefore we want to develop solutions to detect other colored sheep as well. We prefer not to use a thermal camera because we want it to be easy and accessible for farmers.

We will focus mainly on the black sheep as our earlier research indicates that they are somewhat easier to detect than brown sheep. However we will also test all the researched solutions on brown sheep as well as black sheep. In addition we want try to improve the solutions for finding white sheep as our previous solutions are not optimal. For that reason, we will also test if any new solutions can be adapted to detect white sheep with better results than in our earlier research.

1.3 Outline

In **Chapter 2, *State of the art*,** the previous work in this field of study is presented. It includes other similar research projects, as well as existing solutions solving this problem.

Chapter 3, *Implementation*, describes how the different solutions were implemented and justifies the choices made along the way. We experiment with Thresholding, Preprocessing Methods, Object Dimensions and Color Detection.

In **Chapter 4, *Results and discussion*,** here the results are presented and discussed, including a section where we compare and summarize the researched solutions.

Chapter 5, *Future work*, suggests the next step and potential future research opportunities.

Finally, in **Chapter 6, *Conclusion*,** we summarize our study and answer the research questions.

Chapter 2

State of the Art

There have been attempts to solve this problem before, and one example is the Norwegian company **Telespor** [6]. They have made a solution called **Radiobjella** [7] that uses GPS tracking and has a two-way communication between the farmer and the sheep by the use of regular mobile telecommunications technology. The issue with this kind of technology is the fact that there are a lot of mountain areas and other remote areas in Norway that have low or no cellular coverage, as we can see in both **Figure 2.1** and **Figure 2.2**. The different colors illustrate the different coverage levels. In both maps, no color means no coverage. Although most of Norway has cellular coverage, there are some significant areas that are lacking, and these are usually areas where sheep wander. This means that the existing solutions that use GPS tracking have the potential to be excellent solutions for the sheep farmers, but due to the lack cellular coverage there are some farmers that cannot take advantage of the technology.

Some sheep farmers in this situation wanted to make a solution which they could use, that did not require cellular coverage. They developed their own tracking system similar to the ones mentioned previously, called **Findmy** [11]. It uses satellite technology instead of GPS tracking and can be used for other animals as well as sheep. However it is significantly more expensive than the previously mentioned solution. It is a bell, much like the traditional ones, except it uses global Low Earth Orbit satellite technology [12] to send satellite signals to the farmer who can monitor their animals using a PC or tablet.

Also the company **Nortrace** [3] in cooperation with **Telia** [4], have made a tracking sensor for sheep. They were first to launch a new technology called Narrowband Internet of Things (NB-IoT) [5] that is especially developed for sensors and computers so that all kinds of items can communicate over the telecommunications network. They used this technology in a pilot project in the summer of 2017 to track 1000 sheep in Rogaland, Norway [8]. The sheep got sensors which send GPS signals back to the farmer, who can track them on a computer or tablet.

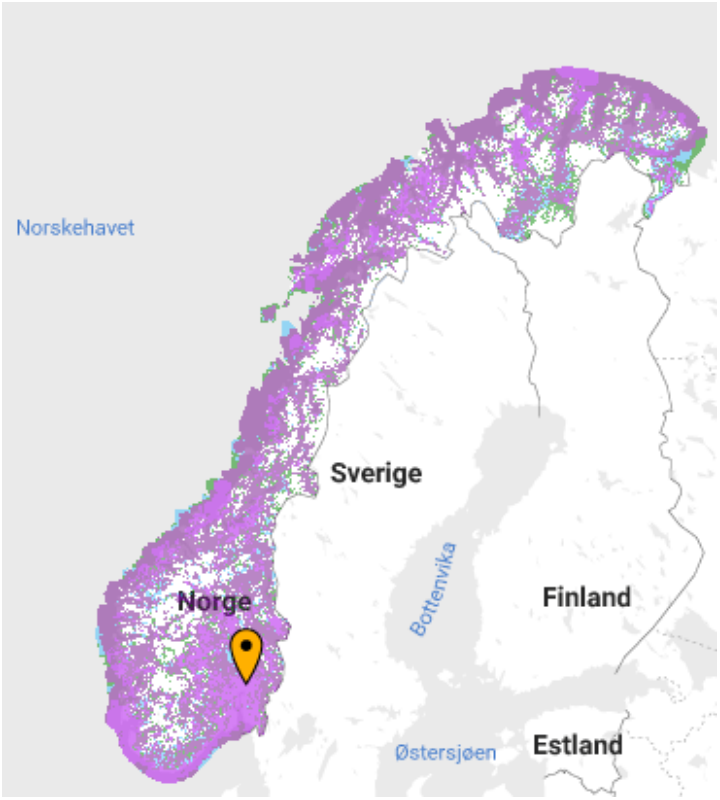


Figure 2.1: Cellular coverage in Norway provided by Telia [9]. The non-colored areas have no cellular coverage.

There have been numerous studies trying to count and/or find animals using Unmanned Aircraft Systems/Unmanned Aerial Vehicles (For the rest of the paper we will use the term UAV). In one of the studies a UAV with a thermal camera was used to count seals, and this was compared to manually counting seals [19]. They photographed two different locations and set a temperature threshold to detect the seals. Their results were good; in the first location they detected 91% of the manually counted seals, and 96% in the other location. When using thermal imagery one also has to take into account the illuminating conditions, meaning that the surroundings are not giving out any warmth and thus will look like an animal in the images. If these are not optimal, it will be reflected in the results.

Martin Israel studied how thermal imagery and UAVs can be used to detect roe deer fawns from being killed in pastures during mowing [21]. He found that this approach involved false positives, which again raises the issue of having to identify what or which animal you have found. For example, as we can see in **Figure 2.3**, it could be a challenge to separate the fawns from other animals, such as rabbits and foxes. Also, when the conditions were not optimal, only one fawn was found.

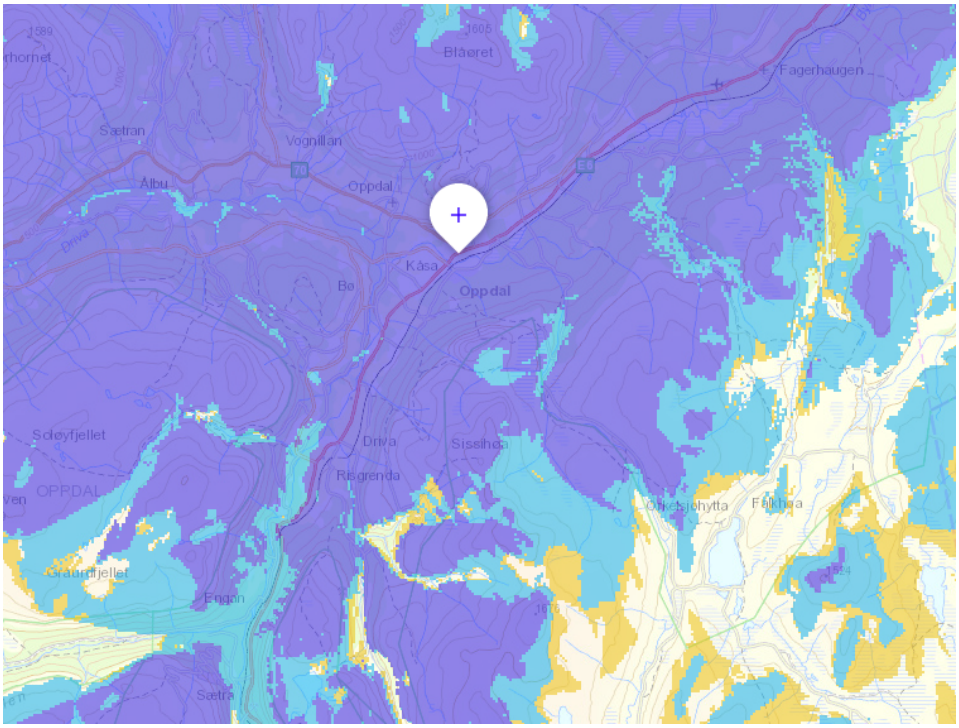


Figure 2.2: Cellular coverage in Oppdal, Norway provided by Telenor [10]. The non-colored areas have no cellular coverage.

This implies that thermal imagery presents some challenges, as one would have to develop a way to separate the sheep from other animals in addition to using the camera's thermal function. Thermal imagery would have been a possible solution for us, but for the reasons stated above, and the fact that thermal cameras are especially expensive, we chose to focus on image analysis and detecting the sheep through their color and shape.

2.1 Detecting Black Sheep

Detecting black sheep is a complex problem. For inspiration we look to an article about a methodology for potato defects detection using computer vision methods [28]. Their problem has similarities to ours in that the defects are usually dark gray or black. They propose two new methods, fixed and adaptive intensity interception method. They did this by simulating the characteristics of the human vision system. When we as humans focus on a particular object the other parts of it is out of focus. In their case, the objects in focus would be the dark spots and the rest of the potato would be out of focus. In **Figure 2.4**

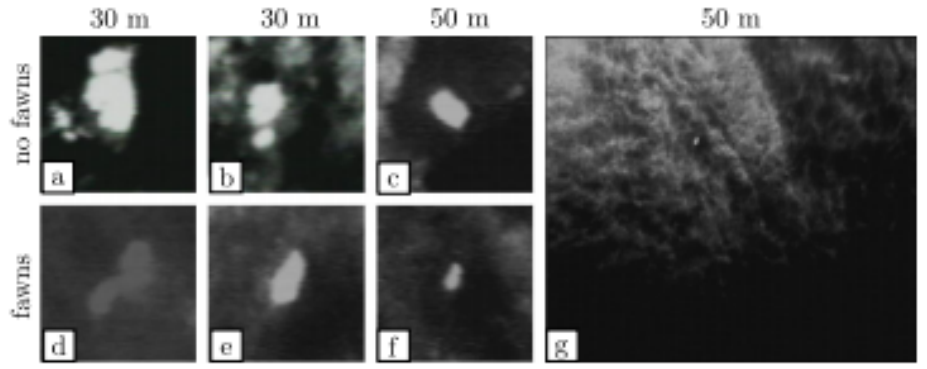


Figure 2.3: Results from M. Israels study [21] on thermal imaging with UAVs. It shows the images taken with thermal camera, and the similarity in the images when there are fawns in the images (image d, e, f) and when there are not (image a, b, c).

are the results from their experiment. The first column are the original images, the second column shows the images with fixed intensity interception applied, the third column shows the images with Otsu Segmentation applied and the fourth and last column are the images with the dark spots marked in red. To apply the Fixed Intensity Interception (FII)

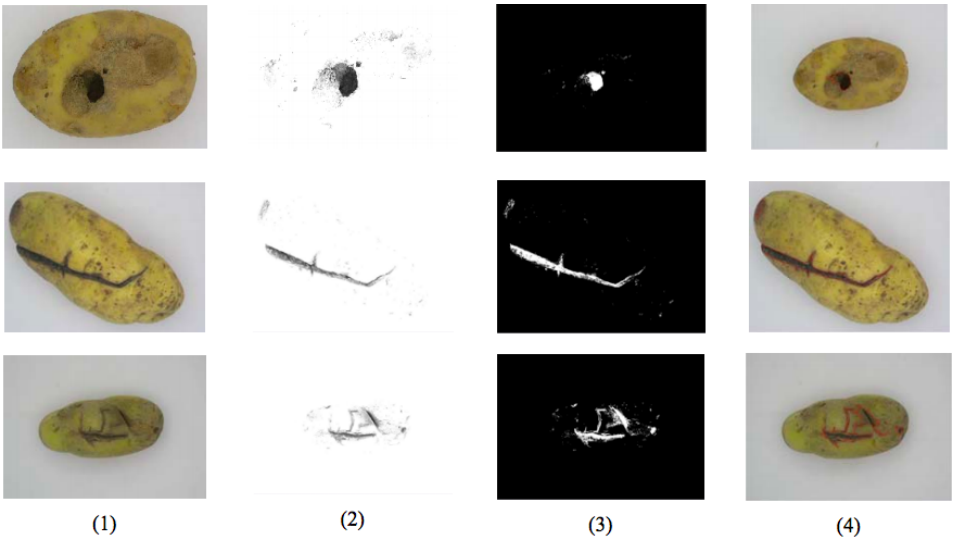


Figure 2.4: The columns are as follows: (1) are the original images, (2) is fixed intensity interception, (3) Otsu Segmentation and (4) are the images with the dark spots marked in red. These images show the different steps in finding the dark spots on the potatoes.

they used a lighting chamber and a Charged-Couple Device camera in automatic shooting mode. This will be not be possible in our case as our images are captured in the mountains

with a UAV. However the Otsu segmentation in the third column can be applied to our images. This is something we will study in **Chapter 3**.

Another option for finding sheep could be to create a 3D reconstruction of the sheep. This is something we would be able to do using OpenCV [29]. In this example they have used four images in **Figure 2.5** of La Sagrada Familia to create the 3D reconstruction in **Figure 2.6**. As we can see, the reconstruction is not a good 3D model of the church, so the

Figure 2.5: Image sequence of La Sagrada Familia. This was used to compute the 3D reconstruction.

first step would probably be to acquire more images from different angles. In addition, the camera's focal length and the center projection coordinates need to be specified and optimized for the reconstruction. The challenging factor in our case will be to collect enough images to get a good reconstruction of the sheep. We could have been able to get four different angles, but more than that will be unrealistic to acquire. Especially as the sheep are moving. We aim to find a relatively simple solution, so because of this and the factors mentioned above, we will not pursue 3D modelling of the sheep as a possible solution as it will be too time consuming and complex.

Because of the fact that the solutions using GPS cannot be used by all farmers as not all areas in Norway have sufficient cellular coverage, there is a market for a simpler solution that requires only a UAV and a software program. As mentioned, there are also a solution using satellite technology, but this is expensive and we want to develop a more affordable solution. A thermal camera would have been a possible solution, but again this is expensive compared to a regular color camera. In addition, as mentioned previously, we would probably have to use some sort of solution that can distinguish sheep from other animals when using a thermal camera. Regarding the labeling of dark spots on potatoes, the Fixed Intensity Interception would be impossibly for us to apply, but the Otsu Segmentation is a method we will experiment with. Lastly, a 3D reconstruction of the sheep would be too complicated and time consuming for it to be a realistic possibility.

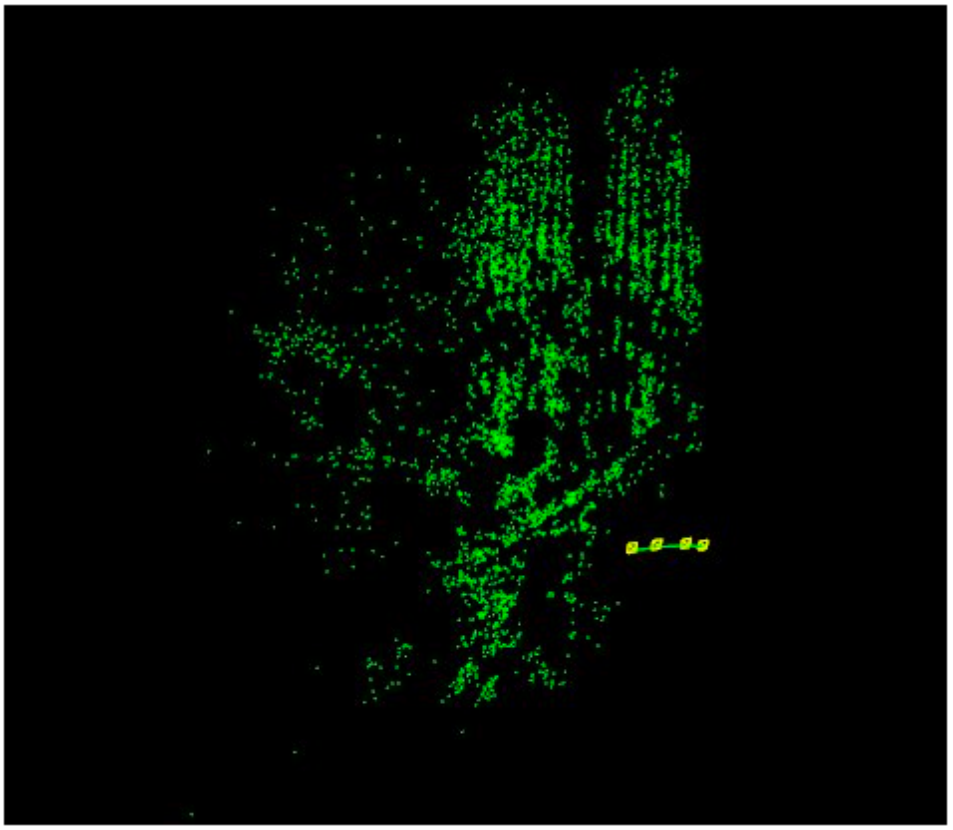


Figure 2.6: The 3D reconstruction of La Sagrada Familia. It is hard to make out that it is the church, which means they should have used more images to make the reconstruction.

Chapter 3

Implementation

To develop our solutions we chose Python 3.7 as our development language and OpenCV 2 as our library. We used Visual Studio Code for editing and Gitlab for version control. We experimented with basic computer vision methods and image processing to find the sheep.

Finding black sheep is a challenging task because the black sheep are often very similar to other objects in nature and blends into the background more than the white sheep. There are more terrains and objects in nature that are black than white. The same goes for brown sheep. Because of this we tried a lot of different methods and solutions within the field of image analysis in order to be able to find the sheep.

We have divided this chapter into four sections. Each section covers different solutions within a certain topic. In each section we explain why we chose to implement these methods and solutions. We show how each solution is applied to the images, using the image in **Figure 3.1** as an example. We will not discuss the results of the applications in this chapter, as they will be presented in **Chapter 4**. However, some observations are introduced when considering whether to continue developing a solution or not.

We have included numerous images and illustrations in this chapter. The reason is simply to show how the different solutions and methods separate from one another, and also to show how effective they are. The illustrations are also useful to understand how the methods work and why we use them.

3.1 Dataset

Using a DJI Mavic Pro drone, we captured **824** pictures on a farm in Oppdal, Sør-Trøndelag in October of 2018. The pictures include white, brown, black and gray sheep. The weather conditions were optimal, meaning that is was sun from clear sky. We also got an assorted dataset when it comes to terrains: Forest, pasture, buildings, rocks and mountains. In ad-

	White sheep	Non-white sheep
Amount in dataset	4343	1125
Percentage in dataset	79.4%	20.6%

Table 3.1: This table shows the composition of sheep in the images in our dataset. There is a total of **5468** sheep in the images, where **4343** of them are white. This means that **79.4%** of the sheep in our dataset are white, and **20.6%** are of other colors.

dition, we captured other animals such as cows, to test if we could make a system that only finds sheep. We experimented with different heights when capturing the pictures, but decided beforehand to capture most of them at about 50 metres above ground level. This was based on a previous study using a similar type of UAV concerning the same problem [22], where they found this to be most efficient.

The colors of the sheep in the captured images are summarized in **Table 3.1**. We have a dataset containing **5468** sheep in total, distributed across **361** images. This gives us **463** images not containing any sheep. Of these sheep, **4343** of them are white, while **1125** are of other colors. Put differently, we have **79.4%** white sheep and **20.6%** non-white sheep in our dataset.

For the initial testing of the solutions covered in this chapter, we chose the image in **Figure 3.1** from the dataset which includes white, brown and black sheep. The image is also captured quite close to the ground, a factor which makes it generally easier to find sheep. We chose this image because it is good for checking if the solution works well enough to run on the entire dataset. If the results are poor with this image, most likely it will generate poor results for the rest of the dataset as well.

3.2 Thresholding

We decided to experiment with different methods for thresholding an image. That means that if the pixel value is greater than the threshold value, it is assigned a certain value, in our case white, and else it is regarded as black [31]. The reason for doing thresholding operations on images is to segment different colors and highlight the colors we are looking for. That way we can say that if the pixels are lighter than a certain RGB (Red, Green, Blue) value it is considered white. Therefore we do this to hopefully make the black sheep more distinct and easier to find in the images. If the thresholding operations are successful we will get an image showing just the black sheep.

3.2.1 Preprocessing

To make it easier to perform computer vision methods on the images we preprocessed them. We did this in the same way as described in detail in our Project Assignment [30]. All the images are converted to grayscale in order to apply thresholding methods.

Resizing

To resize the image we used the method `cv2.resize(input image, output image, interpolation method)` [13], computed the ratio we wanted in the new image and ended up with the image in **Figure 3.1**.

Figure 3.1: Resized image used for the initial testing of our solutions.

Filtering

There is much noise in the images, which means that there are many elements in nature that are disturbing when detecting sheep. Such disturbance and noise will lead to false positives. We therefore apply filters as described in our Project Assignment [30]. We decided to apply the same two filters here, Bilateral [15] and Gaussian [15], with the same parameters as they seemed to work quite well for their purpose.

3.2.2 Finding Edges and Contours

To find edges in the image, we used Canny Edge Detection [18], as described in our Project Assignment [30] with the same parameters. After this we applied the method `cv2.findContours()` to find the contours [23] in the image. We used the same parameters as described in our Project Assignment [30] and sorted them, only this time

around we kept most of the contours to see what results the thresholding gave us. After that we used `cv2.drawContours()`, as also described in our Project Assignment [30] to draw the contours on the resized image to see if we found any sheep.

3.2.3 Binary Thresholding [32]

We apply the Binary Thresholding to the image with the method `cv2.threshold(input image, threshold, maxVal, threshold type)` [32]. In our case the input image is our preprocessed image. We chose the threshold value to be 60 to minimize the false positives, the `maxVal` is 255, and the threshold type is *THRESH_BINARY*. The Binary Thresholding can be described with **Equation 3.1** [32].

$$dst(x,y) = \begin{cases} maxVal & if\ src(x,y) > thresh \\ 0 & otherwise \end{cases} \tag{3.1}$$

If the intensity of the pixel is higher than the given threshold value, then the pixel intensity is set to `maxVal`. Otherwise the pixel intensity is set to 0. In **Figure 3.2** the Binary Thresholding is plotted. The blue line represents the threshold value, the x-axis represents the different pixels in the image and the y-axis is the intensity of the pixel.

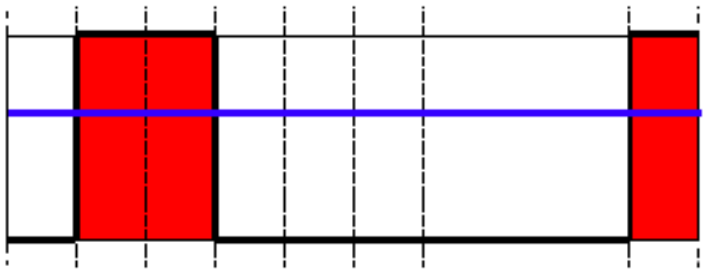


Figure 3.2: Plot of Binary Thresholding. The blue line represents the threshold value, the x-axis represents the different pixels in the image and the y-axis is the intensity of the pixel. If the intensity of the pixel is higher than the given threshold value, it is set to `maxVal`.

No Filter

First we applied the Binary Thresholding to the resized image in grayscale with no filter, the result is in **Figure 3.5a**. After this we applied `cv2.Canny()` to find possible edges in the image. The result is in **Figure 3.5b**. There are loads of noise in the image, meaning that the solutions finds too many edges which again generates many false positives. This means that if we would try to find contours, we would end up with a lot more contours than just the contours of the sheep. For this reason we stopped implementing this solution.

Gaussian Filter

We then applied the Binary Thresholding to a Gaussian filtered grayscale image, the result is in **Figure 3.6a**. After this we applied `cv2.Canny()` to find possible edges in the

image. The result is in **Figure 3.6b**. As we see in the images, there are a lot of noise which leads to false positives when finding contours. However there is less false positives than for the solution with no filter. Many false positives is undesirable and therefore we stopped the implementation after this step.

Bilateral Filter

Then we tried applying a Bilateral Filter to the grayscale image, the result is in **Figure 3.7a**. The next step was applying `cv2.Canny()` to find possible edges in the image. The result is in **Figure 3.7b**. This combination seemed promising, because there are significantly less noise in the image after finding edges, which again will lead to less false positives. Therefore we continued with finding and drawing contours in the image, as described in **Section 3.2.2**. The result can be seen in **Figure 3.8**.

3.2.4 Truncated Thresholding [32]

We apply the Truncated Thresholding to the image with the method `cv2.threshold(input image, threshold, maxVal, threshold type)` [33]. In this case the input image is our preprocessed image, we chose the threshold value to be 80 to minimize the false positives, `maxVal` is naturally 255 and the threshold type is *THRESH_TRUNC*. The binary thresholding can be described with **Equation 3.2** [32].

$$dst(x,y) = \begin{cases} threshold & if\ src(x,y) > thresh \\ src(x,y) & otherwise \end{cases} \tag{3.2}$$

The maximum intensity value for the pixels is `thresh` and if the pixel value is greater than this threshold, the value is truncated. In **Figure 3.3** the Truncated Thresholding is plotted. The blue line represents the threshold value, the x-axis represents the different pixels in the image and the y-axis is the intensity of the pixel.

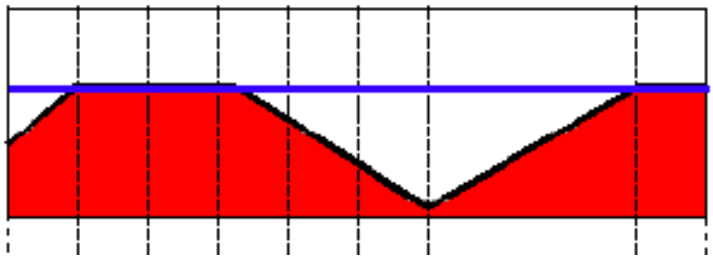


Figure 3.3: Plot of Truncated Thresholding. The blue line represents the threshold value, the x-axis represents the different pixels in the image and the y-axis is the intensity of the pixel. If the pixel value is greater than the threshold value, it is truncated.

No Filter

We applied the Truncated Thresholding to the resized image in grayscale with no filter, the result is in **Figure 3.9a**. After this we applied `cv2.Canny()` to find possible edges

in the image. The result is in **Figure 3.9b**. As we can see from this image, the amount of noise in the image is high. This will lead to finding more contours than there are sheep. As that will lead to many false positives, we chose not to go forth with this combination.

Gaussian Filter

The next thing we did was apply a Gaussian Filter to the grayscale image, before applying the Truncated Thresholding. The result can be seen in **Figure 3.10a**. After this we applied `cv2.Canny()` to find possible edges in the image. The result is in **Figure 3.10b**. Here we can see that there are somewhat less noise than in **Figure 3.9b**, but still a high amount. It will lead to finding a lot of false positives, which we want to avoid. Because of this we stopped the implementation after this step.

Bilateral Filter

Then we applied a Bilateral Filter to the grayscale image, before applying the Truncated Thresholding. The result is in **Figure 3.11a**. After this we applied `cv2.Canny()` to find possible edges in the image, and the result is in **Figure 3.11b**. Here we can see there is almost no noise in the image, which means that there will probably be few false positives. In addition the solution has found the edges of many of the sheep. Therefore we went on with finding and drawing contours in the image, as described in **Section 3.2.2**. The result is in **Figure 3.12**.

3.2.5 Threshold To Zero [32]

We apply the Threshold To Zero method [32] to the image with the method `cv2.threshold(input image, threshold, maxVal, threshold type)` [33]. In this case the input image is our preprocessed image, we chose the threshold value to be 60 to minimize the false positives, `maxVal` is naturally 255 and the threshold type is *THRESH_TOZERO*. This method can be described with **Equation 3.3** [32].

$$dst(x,y) = \begin{cases} src(x,y) & if\ src(x,y) > thresh \\ 0 & otherwise \end{cases} \tag{3.3}$$

If `src(x,y)` is lower than `thresh`, the pixel value will be set to 0, else it is not changed. In **Figure 3.4** the Threshold to Zero method is plotted. The blue line represents the threshold value, the x-axis represents the different pixels in the image and the y-axis is the intensity of the pixel.

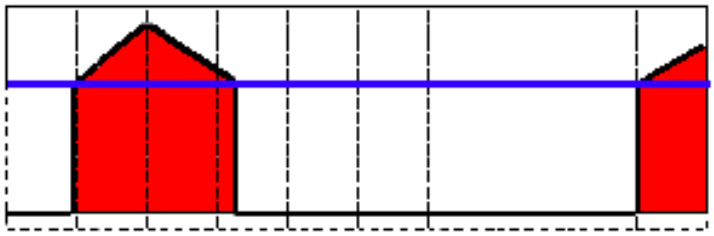


Figure 3.4: Plot of Threshold To Zero method. The blue line represents the threshold value, the x-axis represents the different pixels in the image and the y-axis is the intensity of the pixel. If the pixel value is lower than the threshold value, it is set to 0, else it is not changed.

No Filter

We applied the Threshold To Zero method to the resized image in grayscale with no filter, the result is in **Figure 3.13a**. After this we applied `cv2.Canny()` to find possible edges in the image, and the result is in **Figure 3.13b**. We can see from the image that the solution finds many edges that are not sheep. This will lead to many false positives when finding contours in the image, which we want to avoid. Therefore we stop the implementation of this solution.

Gaussian Filter

The next thing we did was adding a Gaussian Filter to the image, before we applied the Threshold To Zero method. The result is in **Figure 3.14a**. After this we applied `cv2.Canny()` to find possible edges in the image. The result is in **Figure 3.14b**. As can be seen from the image, there are somewhat less noise than the for the solution with no filter, however there is still a high amount. For this reason we stopped the implementation here.

Bilateral Filter

The last thing we tried was applying a Bilateral Filter to the image, before we applied the Threshold To Zero method. The result is in **Figure 3.15a**. After this we applied `cv2.Canny()` to find possible edges in the image, and the result is in **Figure 3.15b**. The solution find significantly less edges in the image than for the Gaussian Filter, and the edges it does find look much like the sheep. It seemed like this would not generate many false positives, therefore we decided to find and draw contours in the image, as described in **Section 3.2.2**. The result is in **Figure 3.16**.

3.2.6 Otsu Segmentation [31]

We apply the Otsu Segmentation to the image with the method `cv2.threshold(input image, threshold, maxVal, threshold type)` [32]. In this case the input image is our preprocessed image, the threshold value is 0, `maxVal` is naturally 255 and

(a) Binary Thresholding applied to grayscale image.

(b) Edges found in image with Binary Thresholding applied to grayscale image.

Figure 3.5: Binary Thresholding on grayscale image. Here we can see too much noise in the images.

the threshold type is *THRESH_BINARY+THRESH_OTSU*. The Otsu Segmentation calculates a threshold value from the image histogram of a bimodal image [31]. A bimodal image is an image whose histogram has two peaks. The histograms for our image is in **Figure 3.17**. Because the method calculates its own threshold value, we simply pass in 0 as threshold value so the algorithm finds the optimal value. The algorithm tries to find a threshold value which minimizes the weighted within-class variance given by **Equation 3.4** where the parameters are as in **Equation 3.5**, **3.6** and **3.7**. It finds a value of *t* that lies between two peaks so that the variance in both of them are minimal.

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t) \tag{3.4}$$

$$q_1(t) = \sum_{i=1}^{t} P(i) \quad \& \quad q_2(t) = \sum_{i=t+1}^{I} P(i) \tag{3.5}$$

$$\sigma_1^2(t) = \sum_{i=1}^{t} [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)} \quad \& \quad \sigma_2^2(t) = \sum_{i=t+1}^{I} [i - \mu_2(t)]^2 \frac{P(i)}{q_2(t)} \tag{3.6}$$

$$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)} \quad \& \quad \mu_2(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{q_2(t)} \tag{3.7}$$

No Filter

We applied the Otsu Segmentation [31] to the resized image in grayscale with no filter, the result is in **Figure 3.18a**. Next we applied `cv2.Canny()` to find potential edges in the image, and the result is in **Figure 3.18b**. There are too much noise in the image, and the solution has detected too many edges that are not sheep. This will lead to false positives when finding contours. Therefore we stopped the implementation of this solution.

Gaussian Filter

Then we applied a Gaussian Filter to the image, before we applied the Otsu Segmentation. The result is in **Figure 3.19a**. Next we applied `cv2.Canny()` to find potential edges in the image, and the result is in **Figure 3.19b**. Here we can see that there are less edges found than for the solution with no filter, but there are still many edges detected that are not sheep. Again, it will lead to many false positives when finding contours. We are trying to minimize the number of false positives, and because of this we did not develop this solution further.

Bilateral Filter

We tried applying a Bilateral Filter to the grayscale image, before we applied the Otsu Segmentation. The result is in **Figure 3.20a**. Next we applied `cv2.Canny()` to find potential edges in the image, and the result is in **Figure 3.20b**. With this solution, there are far less edges found than with the two previous solutions. This again leads to less false positives, and therefore we went on with finding and drawing contours in the image, as described in **Section 3.2.2**. The result is in **Figure 3.21**.

3.2.7 Adaptive Mean Thresholding [31]

We apply the Adaptive Mean Thresholding to the image with the method `cv2.adaptiveThreshold(input image, maxVal, adaptive method, threshold type, blockSize, constant subtracted from the mean)` [33]. For the `maxVal` we again naturally chose 255, and *ADAPTIVE_THRESH_MEAN* as the adaptive method, *THRESH_BINARY* as the threshold type to keep it simple, 11 as block size and 2 as the constant. What the Adaptive Mean Thresholding does is calculate the threshold for small regions of the image based on the mean value of the neighborhood area, which means we get different thresholds for different regions. This will give us a better result for images with varying illuminations [31].

No Filter

We apply the Adaptive Mean Thresholding to the resized image in grayscale with no filter, and the result is in **Figure 3.22a**. After this we applied `cv2.Canny()` to find possible edges in the image. The result is in **Figure 3.22b**. If we look at all the edges in this image, it is hard to make out the edges that form the sheep. That will not only lead to many false positives, but we may not find any sheep at all. That is why we stopped implementing this solution after this step.

Gaussian Filter

Next we applied a Gaussian Filter to the grayscale image, before we applied the Adaptive Mean Thresholding. The result is in **Figure 3.23a**. Next we applied `cv2.Canny()` to find potential edges in the image, and the result is in **Figure 3.23b**. As in the previous solution with no filter, there are many edges and hard to make out the sheep's edges from the rest. Although it is somewhat easier to see the sheep's edges here than for the solution with no filter. Nevertheless, it will lead to many false positives, and because of that we do not continue with the implementation.

Bilateral Filter

The last thing we tried was applying a Bilateral Filter to the image, before we applied the Adaptive Mean Thresholding. The result is in **Figure 3.24a**. Next we applied `cv2.Canny()` to find potential edges in the image, and the result is in **Figure 3.24b**. This solution found less edges in the image, and it is easier to see the sheep's edges. Although there are quite a lot of noise in the image which will lead to false positives, we still go on with finding contours, as described in **Section 3.2.2**, in the image because it is the combination with Adaptive Mean Thresholding that has shown the most potential. The result is in **Figure 3.25**.

3.2.8 Adaptive Gaussian Thresholding [31]

We apply the Adaptive Gaussian Thresholding to the image with the method `cv2.adaptiveThreshold(input image, maxVal, adaptive method, threshold type, blockSize, constant subtracted from the mean)`

[33]. For the `maxVal` we again naturally chose 255, and *ADAPTIVE_THRESH_GAUSSIAN* as the adaptive method, *THRESH_BINARY* as the threshold type to keep it simple, 11 as block size and 2 as the constant. What the Adaptive Gaussian Thresholding does is calculate the threshold for small regions of the image based on the weighted sum of the neighborhood values where the weights are a Gaussian window, which means we get different thresholds for different regions. This will give us a better result for images with varying illuminations [31].

No Filter

We applied the Adaptive Gaussian Thresholding to the resized image in grayscale with no filter, the result is in **Figure 3.26a**. After this we applied `cv2.Canny()` to find possible edges in the image. The result is in **Figure 3.26b**. The image is full of edges, and it is impossible to make out the sheep's edges. That will lead to many false positives and the solution will probably not find any of the sheep. Therefore we do not continue implementing this solution.

Gaussian Filter

Then we applied a Gaussian Filter to the grayscale image, before applying the Adaptive Gaussian Thresholding. The result is in **Figure 3.27a**. Next we applied `cv2.Canny()` to find potential edges in the image, and the result is in **Figure 3.27b**. Here we can barely make out the sheep's edges, but similar to the previous solution with no filter, there is too much noise. It will generate too many false positives and probably not find the sheep. Because of this we stop the implementation of this solution.

Bilateral Filter

We tried applying a Bilateral Filter to the image, before we applied the Adaptive Gaussian Thresholding. The result is in **Figure 3.28a**. Next we applied `cv2.Canny()` to find potential edges in the image, and the result is in **Figure 3.28b**. This solution generates less edges than the two previous solutions, and we can see the edges that make up the sheep. There are still a lot of noise in the image which will give us many false positives, however it is the most promising solution using Adaptive Gaussian Thresholding and therefore we find contours, as described in **Section 3.2.2**. The result is in **Figure 3.29**.

3.3 Edge Detection On Preprocessed Images

One of the main problems with finding brown or black sheep is that they are more similar to the background than white sheep are, so we decided that it could be useful to look at different filters and preprocessing methods to see if we could make the objects in an image appear more clearly. In our Project Assignment [30] we had already been experimenting with the Gaussian Filter and the Bilateral Filter, but we wanted to test if applying other filters could provide better results, both on their own and in combination with the others.

For comparison, all methods mentioned below were performed on the image in **Figure**

3.1 in the same way as with the *Solution Using Bilateral Filter* in [30]. Meaning that we used Canny Edge Detection and OpenCV's `cv2.findContours()` method to find the contours of objects, but we applied other preprocessing methods and/or filters than we did in the previous solutions.

As we had good experiences with the Bilateral Filter from our earlier work, we tested all the methods listed in this section on both the original image and on a Bilateral filtered image. This was done because we noticed that the Bilateral Filter removed a lot of small and unwanted disturbances in the image while keeping the important features of the objects.

3.3.1 Dilation [46]

The Dilation method in OpenCV dilates an object in an image by using a specific structuring element. This means that the method makes the objects in the image bigger by using a matrix of the neighboring values of each pixel to calculate the new value of the pixels. The maximum value of the neighboring pixels is chosen, which makes the objects become larger, and removes small holes in the image which again makes it smoother. This method can be applied iteratively, that is to say that we specify the amount of times we want to perform the Dilation in the method's parameters.

The reason why we chose to try the Dilation, was that it is going to make objects appear bigger while removing disturbances by evening out the image. We thought this could be good for our purpose, as we wanted the objects to stand out more from the background so that they are easier to detect.

In OpenCV, the method used for Dilation is the `cv2.dilate(image, structuring_element, iterations)` [46] where the `image` is the image on which we want to perform the Dilation, the `structuring_element` is given by `cv2.getStructuringElement(type, size)` and is used to expand the objects with a similar shape and size in the image, and the `iterations` specify the number of times the method is run on the image.

The `type` parameter in the `cv2.getStructuringElement()` method defines the shape of the structuring element, for example rectangular, and the chosen shape helps differentiating an object from the others. The `size` parameter defines the size of the structuring element.

For our objects, we found that a 3x3 rectangular structuring element was preferable. When we applied the Dilation method with our chosen parameters and with one iteration, we obtained the dilated image in **Figure 3.30a**. If combining the Dilation with the Bilateral Filter [15], we get the image as seen in **Figure 3.30b**.

When applying the Canny Edge Detection [18], we obtained the edges seen in **Figure 3.31a and 3.31b**, and after applying the `cv2.findContours()` method and filtering out the unwanted contours, the detected objects was as seen in **Figure 3.32a** and **3.32b**.

3.3.2 Meanshift Filter [34]

Applying the Meanshift Filter to our images works by replacing each pixel in the image with the mean pixel value within a given neighborhood of pixels. This leads to an image with less noise, and make the objects stand out more from the background, which is what we want to achieve.

The Meanshift Filter function `cv2.pyrMeanShiftFiltering(image, spatial_window_radius, color_window_radius)` [34] in OpenCV, implements the filtering stage of the Meanshift segmentation. This means that it returns an image with less colors and less texture than the original image, as it at every pixel in the original image executes meanshift iterations as explained below. The output image is more blurry with fewer details, depending on the parameters.

The Meanshift iterations replace each pixel in the image with the mean of the pixels within a neighborhood of a given size (spatial window radius (`sp`)) that are within a given color range (color window radius(`sr`)). A higher spatial window radius gives an image that looks smoother, and a higher color window radius gives an image with fewer colors.

In other words, for each pixel `p` in the image, the pixels around `p` are considered if they are located closer to the original pixel than `sp`, and differ less in color from the original pixel than `sr`. The considered neighboring pixels of the pixel `(X,Y)` can also be described as in **Equation 3.8**

$$(x,y) : X-sp \leq x \leq X+sp, Y-sp \leq y \leq Y+sp, \parallel (R,G,B)-(r,g,b) \leq sr \parallel \quad (3.8)$$

where `(R,G,B)` and `(r,g,b)` are the vectors of color components at `(X,Y)` and `(x,y)` respectively. By experimentation, we found that the parameters that smoothed the disturbances in the image while keeping the important details of the objects best in our case was `sp = 15` and `sr = 60`. The Meanshift Filter with these parameters applied to the original image can be seen in **Figure 3.33a**, while the filter applied to a Bilateral filtered image can be seen in **Figure 3.33b**.

The effect of applying the Canny Edge Detection to the images in **Figure 3.33** can be seen in **Figure 3.34**, and the contours found can be seen in **Figure 3.35**.

3.3.3 Shadow Removal

As some of the sheep in our images are hard to distinguish from the background because of their shadows, we considered the possibility to remove these shadows. To do so, we tried to use a method that can remove shadows on scanned pages of text. [35]

We started by splitting the image into the three different color planes in the RGB color space(Red, Green and Blue) by using `cv2.split(image)`. Then, we iterated over the planes and performed actions on each of them. First, we dilated the image to get rid of the small details, using `cv2.dilate(plane, structuring_element, iterations)` as in **Section 3.3.1**. Secondly, we performed a Median Blur on the image to further remove

unwanted details from the image. This was done using the method `cv2.medianBlur (dilated_plane, kernel_size)` [36] which replaced every pixel in the plane with the median of the pixel values under the kernel area.

Then, we calculated the difference between the new, blurred plane and the original plane to keep only the details. This was done using `cv2.absdiff(plane, new_plane)`, and subtracting it from 255 as identical bits are shown as black as default, while we prefer them to be white.

After iterating through the color planes, they were merged back to one image. We experimented with using different structuring elements (as explained in **Section 3.3.1**) and different kernel sizes to obtain different results. One example of the Shadow Removal applied to our image without and with the Bilateral Filter can be seen in **Figure 3.38a** and **Figure 3.38b** respectively. When applying Canny Edge Detection, we get the edges seen in **Figure 3.37**, which again gives us the contours seen in **Figure 3.38**.

3.3.4 Histogram Equalization [37]

One way to create bigger differences between light and dark objects in an image is to use Histogram Equalization [37]. As we wanted to increase the differences between the sheep and the background, it could be an idea to increase the contrast. To do so, we applied the `cv2.equalizeHist(image)` function to the image. The image histogram is a graphical representation of the intensity values in an image, and it presents the number of pixels of each of the intensity values. Performing Histogram Equalization means to redistribute these intensity values more evenly across the whole range.

The `cv2.equalizeHist(image)` can be performed using different color spaces, and one of the preferred ones for Histogram Equalization is the YUV color space as it keeps a separate intensity channel.

The YUV color space is commonly used for TV broadcasting and other applications where compression is important [38]. Compared to RGB, the YUV color space separates the color information into one luminance(intensity) channel, Y, and two chrominance(color) channels, U and V. The positive aspect of this is that we can do operations on the intensity channel without impacting the colors in the image. That is ideal when performing Histogram Equalization, as we want to even out differences in the luminance in the image.

It is not possible to perform Histogram Equalization in a good way in the RGB color space, as we have to consider the intensity in the whole image, and that is not reflected in each of the three color channels [39]. So to not disturb the color balance in the image we have to perform Histogram Equalization in a color space with a separate intensity channel.

We tried to perform Histogram Equalization on our images both using the YUV color space and on a grayscale version of the image. This can be seen in **Figure 3.39a** and **Figure 3.39b** respectively. If we apply the Bilateral Filter to the image, we get the result seen in **Figure 3.40** with the edges in **Figure 3.41** and the contours found in **Figure 3.42**.

3.4 Bounding Rectangle [41]

When applying some of these blurring filters mentioned in the previous section, we had an increased number of false positives. Therefore, to try to limit the amount of contours, we wanted to apply some kind of limitation on the size of the detected objects. One idea was to use the height above ground to estimate the real size of the object. It is possible to extract data from the UAV images containing the height above ground, but the problem is that this height is just at the exact point where the UAV take off. As we were operating in the terrain, there were big height differences from one image to another as we flew the UAV over large areas, so the indicated height above ground is very imprecise.

It is also possible to extract the GPS coordinates from the images. These can be used to calculate the height above ground if we know the meters above sea level(from now on referred to as MASL) at both the starting position and the position of the UAV when the image was taken. We tried using the Open-Elevation API [40] to get the MASL, but it lacks data at our location, and we had problems finding good alternative APIs to get the information needed to do the calculations.

As an alternative we tried to use the rotated Bounding Rectangle method `cv2.minAreaReact(contour)` [41]. This makes the smallest possible rectangle around an object found using the `cv2.findContours()` method as described in [30]. Then we calculated the relationship between the width and the height of the Bounding Rectangle to check if the object's dimensions could resemble the ones of a sheep.

From studying the average size of sheep, we found that they usually range in length from 120-180cm and in width from 65-127cm [42]. By experimenting with these values, we found that it was good to exclude all contours where the relationship between the width and height was over **2,0** and less than **1,4**. The relationship *R* is given by **Equation 3.9**, and the contour was kept if *R* was between **1,4** and **2,0**. As we had seen that the *Solution using Dilation with Bilateral Filter* seemed promising, we added the Bounding Rectangle to that solution, and compared the results.

$$R = \begin{cases} \frac{width}{height} & \text{if } width > height \\ \frac{height}{width} & \text{if } width \leq height \\ 0 & \text{if } width = 0 \text{ or } height = 0 \end{cases} \tag{3.9}$$

3.5 Color Detection

Detecting certain colors in the images is naturally a possible solution for us as we are looking for sheep-like objects in a specific color. In earlier research, we experienced that looking for white pixels in the images gave good results when looking for white sheep [30]. Therefore, we wanted to try if this was possible to extend to other colors. It was probably easier to do with white pixels as there are not too many white objects in nature except sheep, but with black and especially brown there are a lot of other objects with

similar colors. Hence, the challenge is to pick the right colors and methods that minimize the number of false positives.

3.5.1 Searching For Black Areas

We tried the most basic solution, namely searching for black areas in the same way we searched for white areas in our earlier research [30]. We used the exact same solution as before, where we checked the color of each pixel and the neighboring pixels to see if they were within a given color interval. Here, we just changed the color interval from white to black. We chose to search for the color values between `RGB(0,0,0)` and `RGB(60,60,60)` as it goes from completely black to dark gray. That is an advantage because the black sheep vary in color, and some are mostly dark gray instead of completely black.

3.5.2 Removing Irrelevant Colors

Other than just iterating through the image to find pixels of a given color, we explored a method that removes all other colors than the one we are looking for. In other words, only the pixels with a color value inside a given RGB-interval were kept, while the others were set to black.

We started by using the `cv2.inRange()` method which expects three parameters: the input image, the lower boundary and the upper boundary of the color that we want to detect. The output is a binary mask, meaning that the image is split into black or white. The white pixels represent the pixels within the boundaries, and the black pixels represent the ones outside the limits.

To get the image showing only the given color, we made a call to `cv2.bitwise_and()` which returns an image showing only the pixels in the image corresponding to the white pixels in the binary mask obtained above.

Then, we used the Canny Edge Detection method [18] to obtain the edges, found the contours using `cv2.findContours()` and drew the detected edges on the original image using `cv2.drawContours()` [23]. We wanted to try this for brown, black and white sheep separately, but found that it was difficult for black and brown sheep.

For black sheep, the problem is that there are a lot of other dark objects and shadows in the image, and also that the background in the result image is black. As the sheep are rather very dark gray than completely black, we chose the interval to go from `RGB(0,0,0)`, to a more grayish black, `RGB(60,60,60)`. The result when applying this to the image in **Figure 3.1** can be seen in **Figure 3.43a**.

When it comes to brown sheep, it is nearly impossible to set an interval for the color brown in the RGB color space. We experimented with a couple of different intervals, but this was very difficult as every interval we tried that detected the brown sheep also included grass and trees.

It worked better on white sheep, although it was quite hard to find a good interval for the white values, because the sheep are not always completely white. After spending the summer in the mountains many of them get quite dirty, and changes in lighting can also impact how the colors appear. Then again, if we make the interval too big, we detect more irrelevant objects. Therefore, by experimenting, we ended up with an interval going from completely white, `RGB(255,255,255)`, to a color that is a bit more gray, `RGB(190,190,190)`. This seemed to detect a decent amount of sheep while avoiding too many false positives.

Applying the above explained procedure to the image in **Figure 3.1**, gives us the images in **Figure 3.43**. The image in **Figure 3.43a** shows the result of removing every color except black to dark gray, while the image in **Figure 3.43b** shows the result of removing all other colors than white.

As the image in **Figure 3.43a** is very dark, we had problems detecting edges. Therefore we have chosen to only present the results of applying Canny Edge Detection to the image in **Figure 3.43b**, as well as the contours found. This can be seen in **Figure 3.44**.

3.5.3 Negative Colors

As most of the objects we detected where white, we considered the possibility that the color of the object was taken into account when looking for edges. Therefore, we wanted to try if it made any difference if we did Canny Edge Detection followed by finding contours using `cv2.findContours()` as described in [30] on the negative of the image. The negative of an image implies that every color in the image is set to its opposite. Meaning that for the RGB color space, we get that `R = 255-R`, `G = 255-G` and `B = 255-B`. The negative of the image in **Figure 3.1** can be seen in **Figure 3.45.**

To compare the contours found in the negative image with the contours found in the original image, we ran it with the same parameters and filters as in **Section 3.3.1**, meaning that we applied the Bilateral Filter and Dilation, and used Canny Edge Detection. This comparison can be seen in **Figure 3.46**.

3.5.4 Hue Saturation Value Color Space [43]

We exerimented with the Hue Saturation Value (HSV) color space to make it easier to find the black and brown sheep. The HSV color space model represents the color spectrum similar to the RGB model [43]. In HSV color space it is the hue channel that represents the color space, contrary to RGB where every channel represents a color. That means it can be useful to segment objects in images based on color. The saturation channel goes from unsaturated shades of gray to fully saturated and the value channel represents the intensity of the color. In **Figure 3.47** we can see the HSV color space model compared to the RGB color space model [43].

To test this implementation we used the same resizing as described in **Subsection**

3.2.1. Then we applied the function `cv2.cvtColor()` which converts an image from one color space to another [33], in our case from RGB to HSV. It takes two input parameters, input image and a color space conversion code, in this case it is `COLOR_BGR2HSV`. Then we define the lower and upper range of the color we are looking for. For black we chose `lower_black = numpy.array([0,0,0])` and `upper_black = numpy.array([0,0,48])` which corresponds to the colors in **Figure 3.48**, and for brown we chose `lower_brown = numpy.array([21,43,63])` and `upper_brown = numpy.array([21,58,42])` which corresponds to the colors in **Figure 3.49**.

Next we create the mask image [44], which means that it is a binary image consisting of only black and white. To create the mask image we use the `cv2.inRange()` function [45]. It takes in three parameters: `input array`, `lower boundary` (array or scalar) and `upper boundary` (array or scalar). In our case the input array is: the output of `cv2.cvtColor()`, `lower_brown` or `lower_black`, and `upper_brown` or `upper_black`. The function checks if the input array, namely the colors in the image, lies between the two boundaries given. The function checks the range for every element of a single-channel array according to **Equation 3.10** and checks the range for two-channel arrays according to **Equation 3.11**.

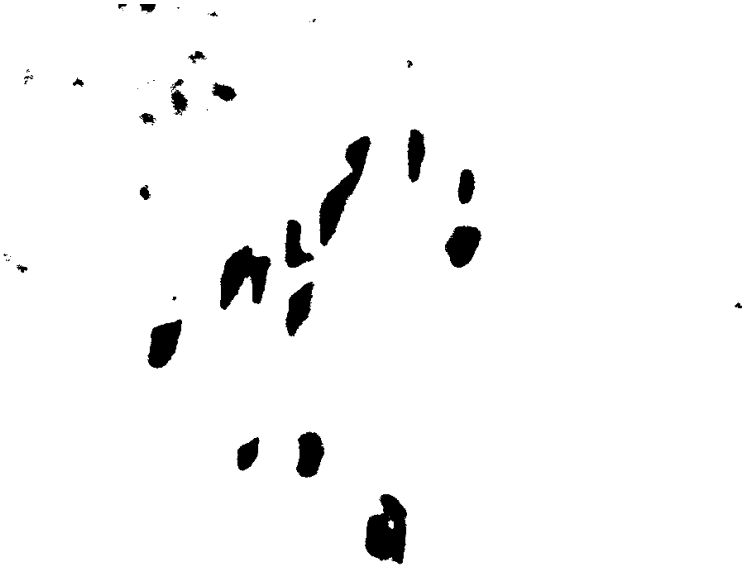
$$dst(I) = lowerb(I)_0 \leq src(I)_0 \leq upperb(I)_0 \tag{3.10}$$

$$dst(I) = lowerb(I)_0 \leq src(I)_0 \leq upperb(I)_0 \ \wedge \ lowerb(I)_1 \leq src(I)_1 \leq upperb(I)_1 \tag{3.11}$$

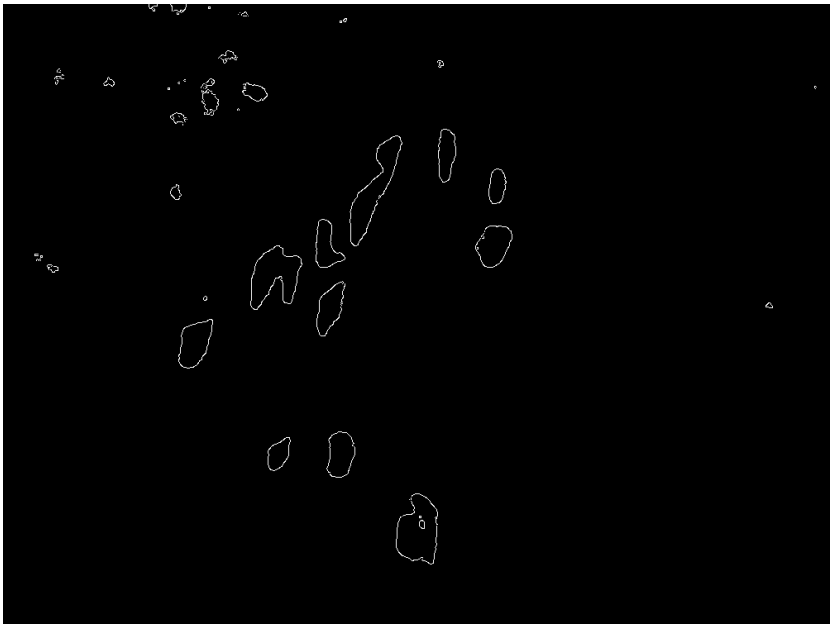
(a) Binary Thresholding applied to Gaussian filtered grayscale image.

(b) Edges found in image with Binary Thresholding applied to Gaussian filtered grayscale image.

Figure 3.6: Binary Thresholding on Gaussian filtered grayscale image. There is too much noise in the image for it to give good results when finding contours.



(a) Binary Thresholding applied to Bilateral filtered grayscale image.



(b) Edges found in image with Binary Thresholding applied to Bilateral filtered grayscale image.

Figure 3.7: Binary Thresholding in Bilateral filtered grayscale image. The noise in these images is significantly reduced, which makes it eligible for finding contours.

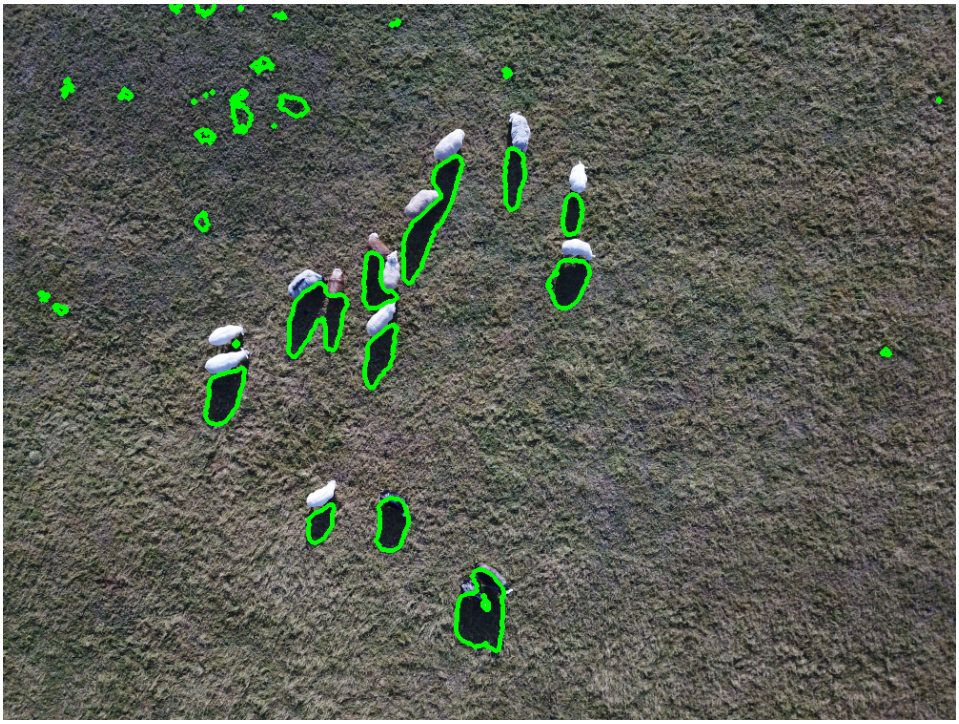


Figure 3.8: Contours found in image with Binary Thresholding applied to Bilateral filtered grayscale image. Here we find all the sheep's shadows and also some false positives.

(a) Truncated Thresholding applied to grayscale image.

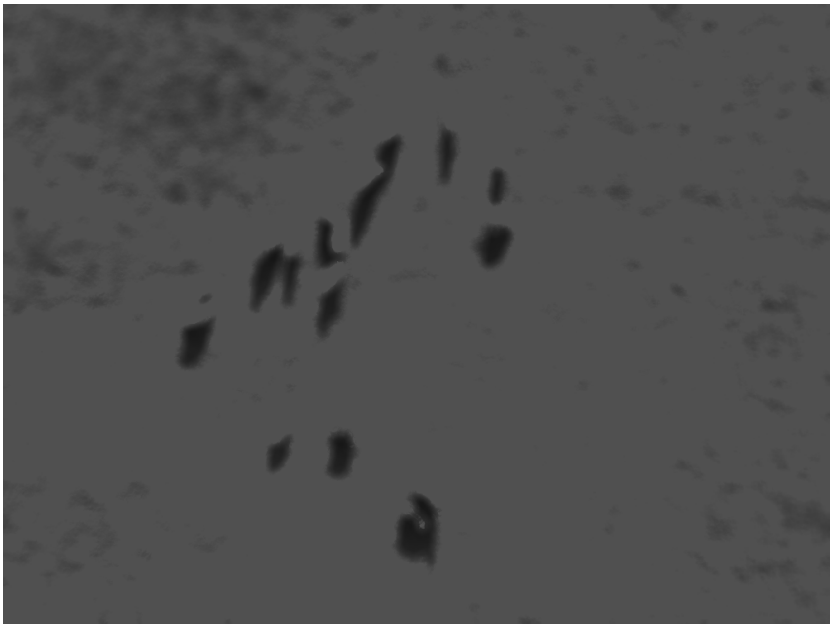
(b) Edges found in image with Truncated Thresholding applied to grayscale image.

Figure 3.9: Truncated Thresholding on grayscale image. There is too much noise in the images, and thus we do not go forward with this solution.

(a) Truncated Thresholding applied to Gaussian filtered grayscale image.

(b) Edges found in image with Truncated Thresholding applied to Gaussian filtered grayscale image.

Figure 3.10: Truncated Thresholding on Gaussian filtered grayscale image. There is too much noise here, because of this we do not find contours.



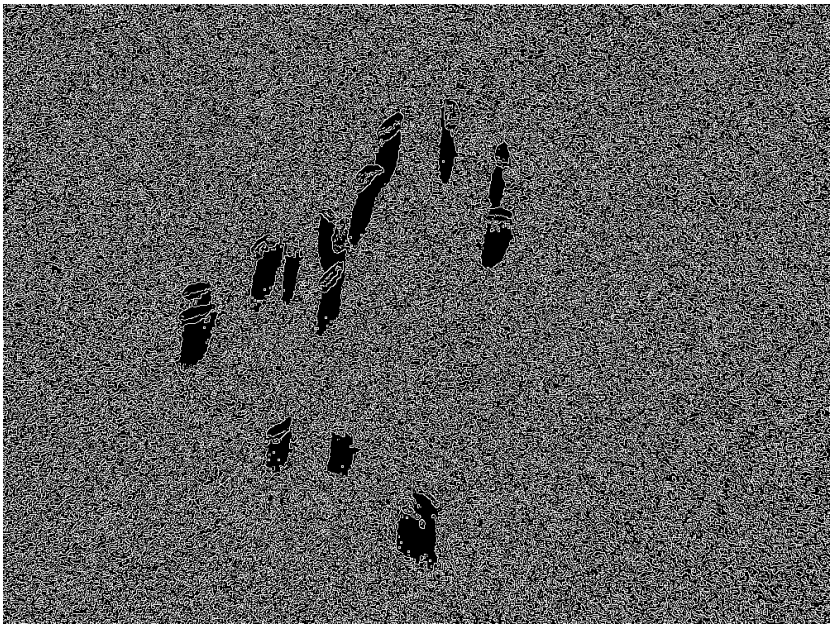
(a) Truncated Thresholding applied to Bilateral filtered grayscale image.

(b) Edges found in image with Truncated Thresholding applied to Bilateral filtered grayscale image.

Figure 3.11: Truncated Thresholding on Bilateral filtered grayscale image. The results here are promising, there are no noise.

Figure 3.12: Contours found in image with Truncated Thresholding applied to Bilateral filtered grayscale image. Good results after finding contours, the solution found all the black sheep and one brown sheep.

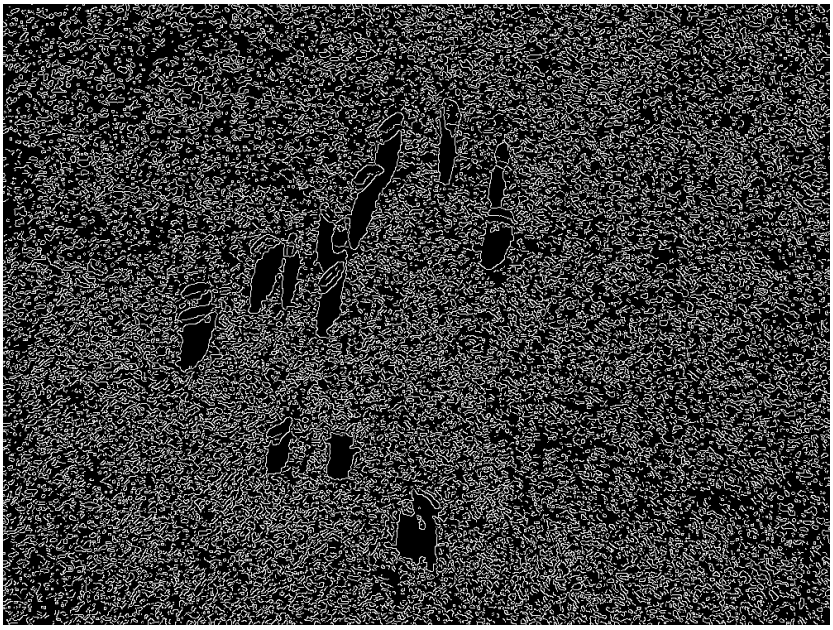
(a) Threshold To Zero method applied to grayscale image.



(b) Edges found in image with Threshold To Zero method applied to grayscale image.

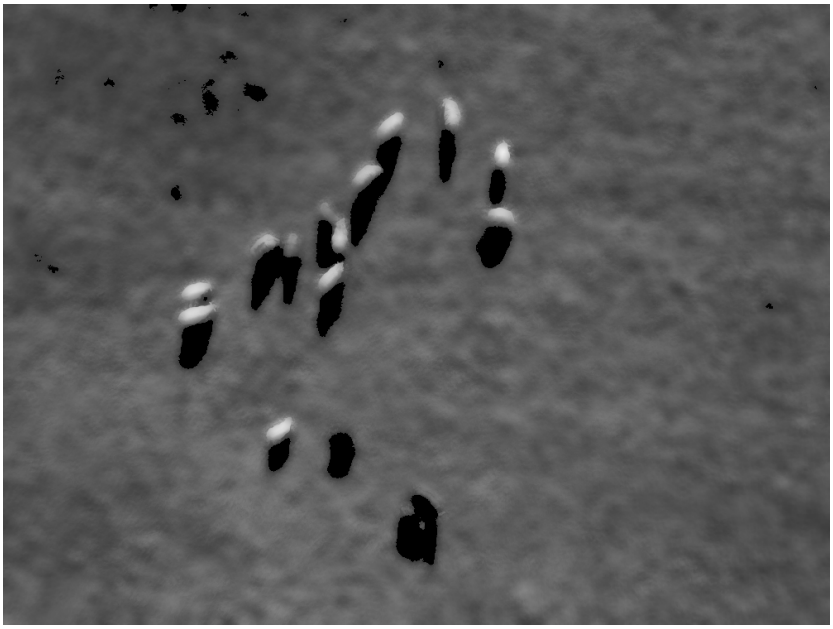
Figure 3.13: Threshold To Zero method on grayscale image. There is too much noise with this solution.

(a) Threshold To Zero applied to Gaussian filtered grayscale image.

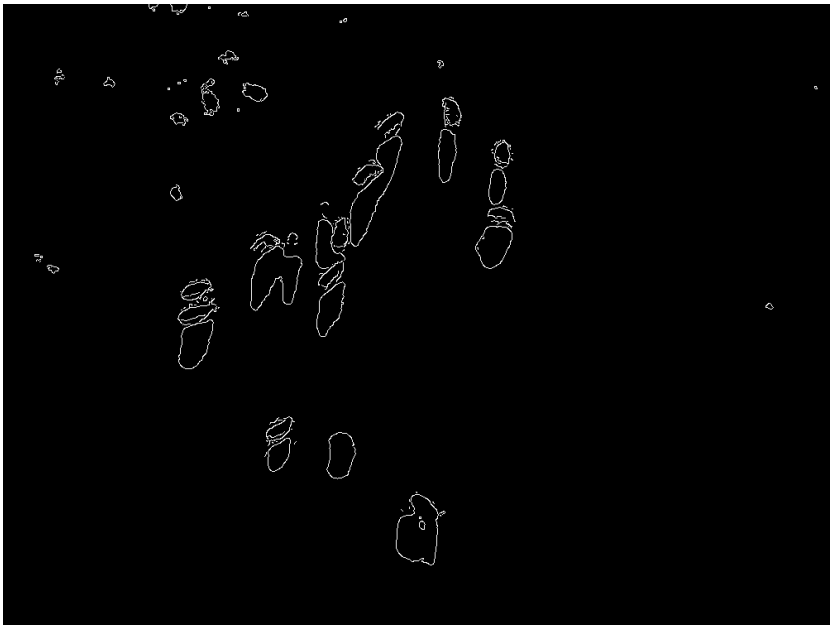


(b) Edges found in image with Threshold To Zero method applied to Gaussian filtered grayscale image.

Figure 3.14: Threshold To Zero method on Gaussian filtered grayscale image. There is too much noise in the image.



(a) Threshold To Zero method applied to Bilateral filtered grayscale image.



(b) Edges found in image with Threshold To Zero method applied to Bilateral filtered grayscale image.

Figure 3.15: Threshold To Zero method on Bilateral filtered grayscale image. Almost no noise, a good basis for finding contours.

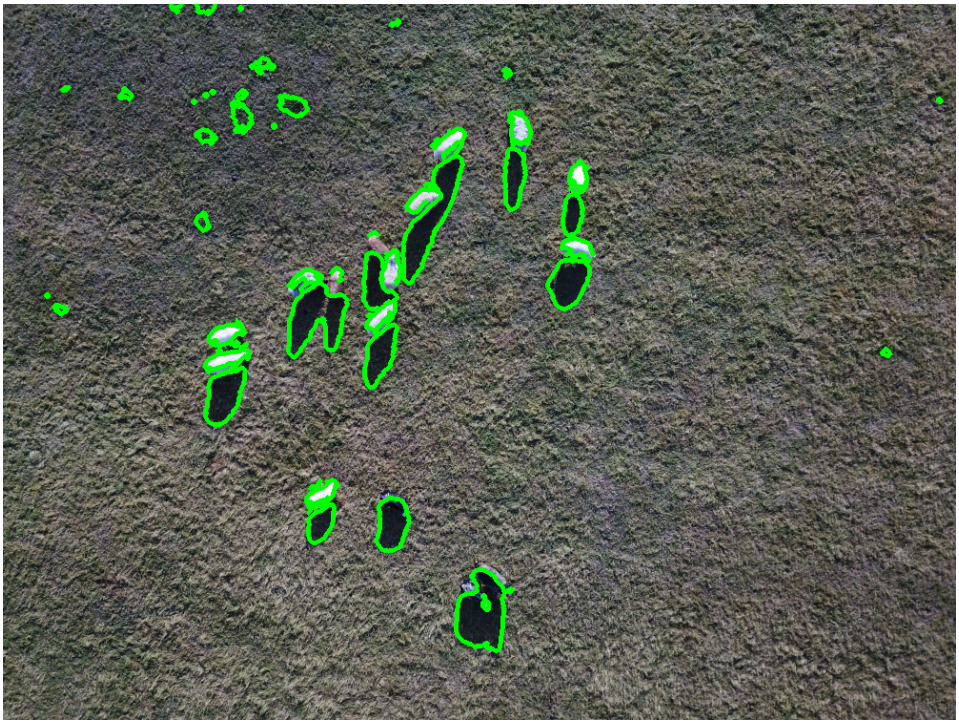
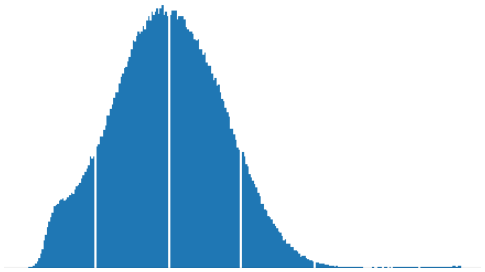
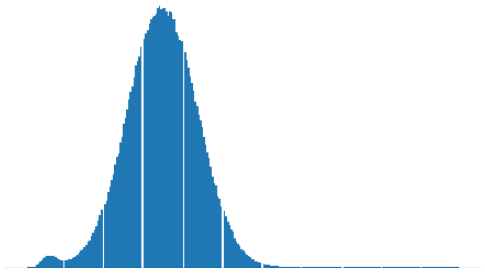


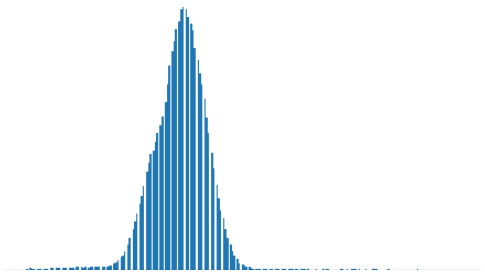
Figure 3.16: Contours found in image with Threshold To Zero method applied to Bilateral filtered grayscale image. It finds all the sheep's shadows and some false positives.



(a) Histogram for grayscale image.



(b) Histogram for Gaussian filtered grayscale image.



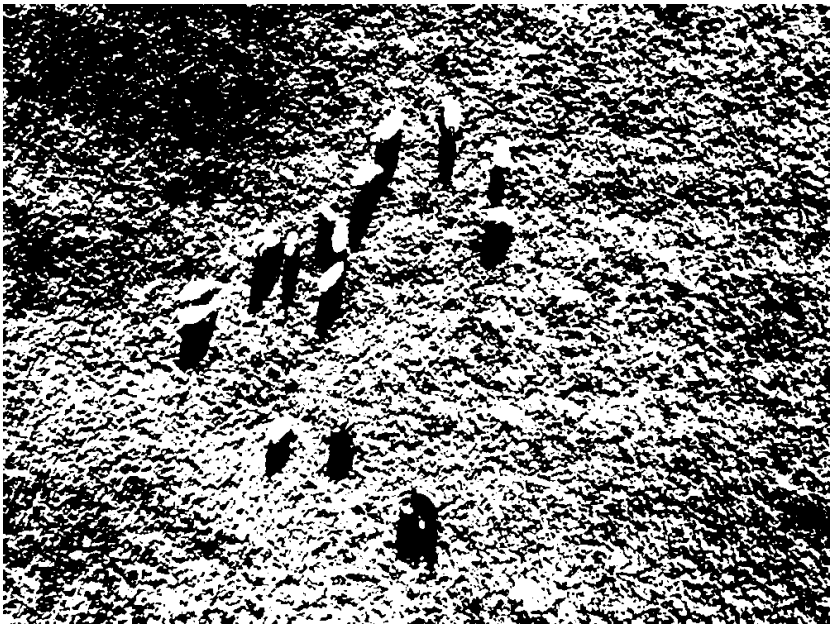
(c) Histogram for Bilateral filtered grayscale image.

Figure 3.17: Histograms of the image in **Figure 3.1** with different filters applied. The Otsu Segmentation calculates a threshold value from these histograms, and the ideal histogram for this segmentation is a histogram with two peaks.

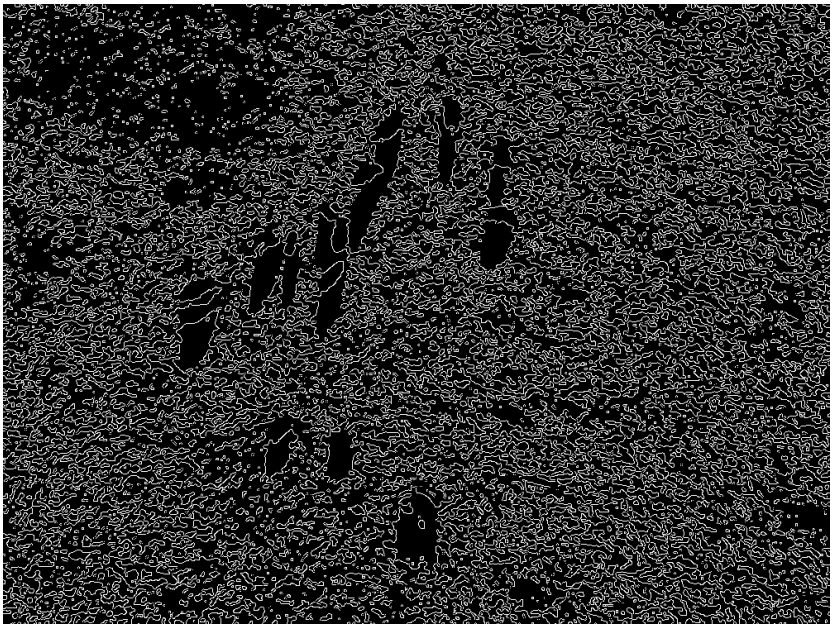
(a) Otsu Segmentation applied to grayscale image.

(b) Edges found in image with Otsu Segmentation applied to grayscale image.

Figure 3.18: Otsu Segmentation on grayscale image. There is too much noise in the image.

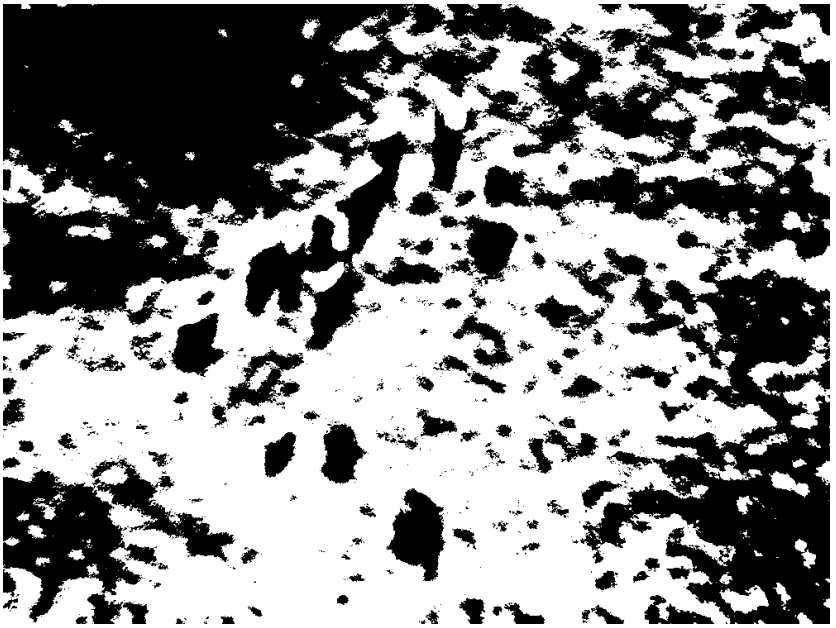


(a) Otsu Segmentation applied to Gaussian filtered grayscale image.

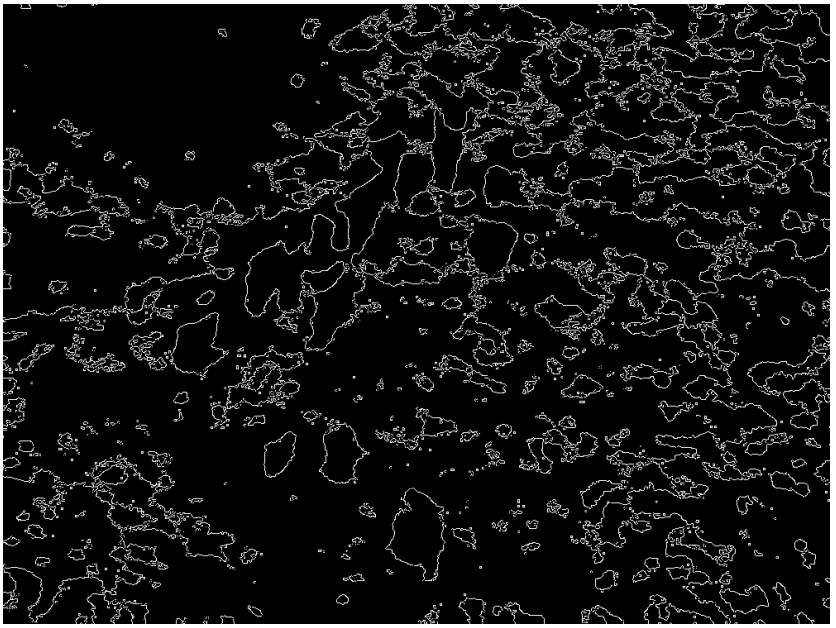


(b) Edges found in image with Otsu Segmentation applied to Gaussian filtered grayscale image.

Figure 3.19: Otsu Segmentation on Gaussian filtered grayscale image. There is too much noise in the image.



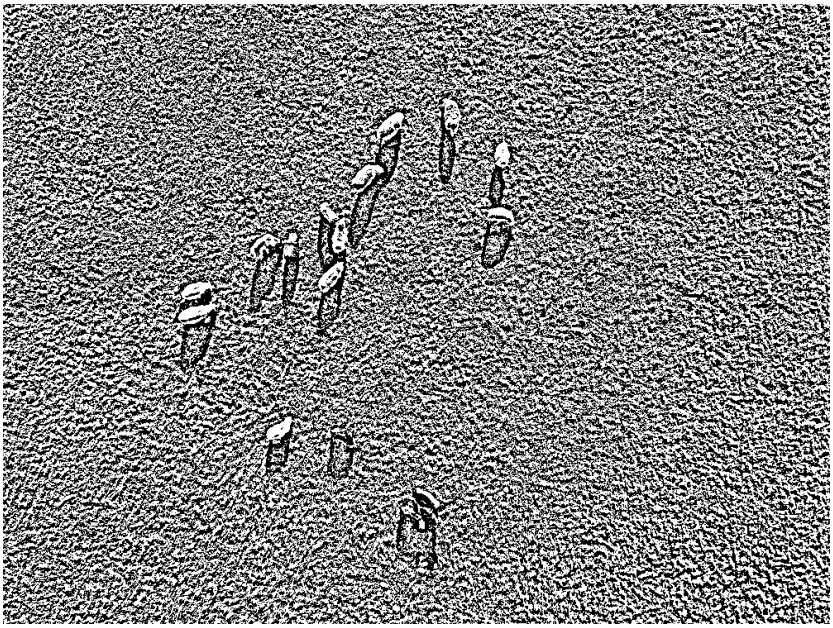
(a) Otsu Segmentation applied to Bilateral filtered grayscale image.



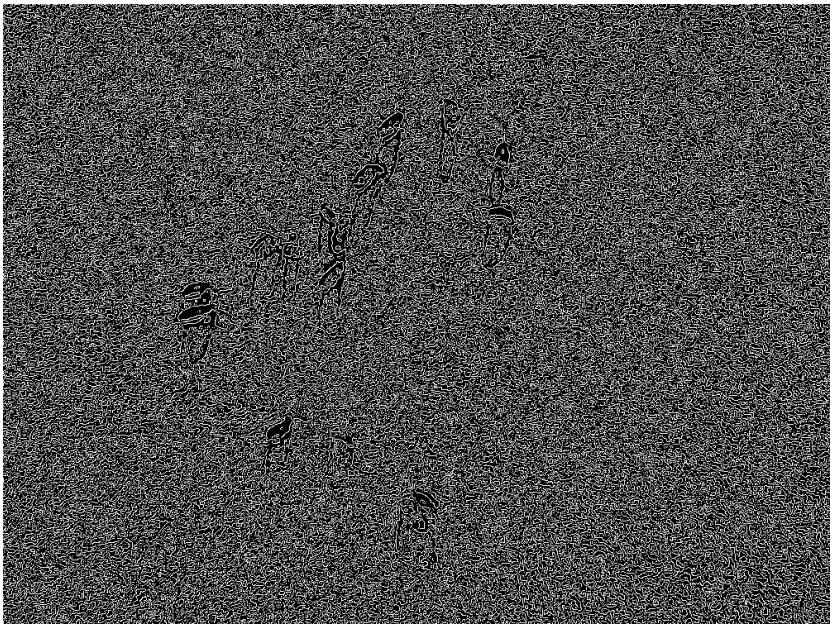
(b) Edges found in image with Otsu segmentation applied to Bilateral filtered grayscale image.

Figure 3.20: Otsu Segmentation on Bilateral filtered grayscale image. There is quite a lot of noise in the image.

Figure 3.21: Contours found in image with Otsu Segmentation applied to Bilateral filtered grayscale image. It does not find any black sheep, and there are many false positives.

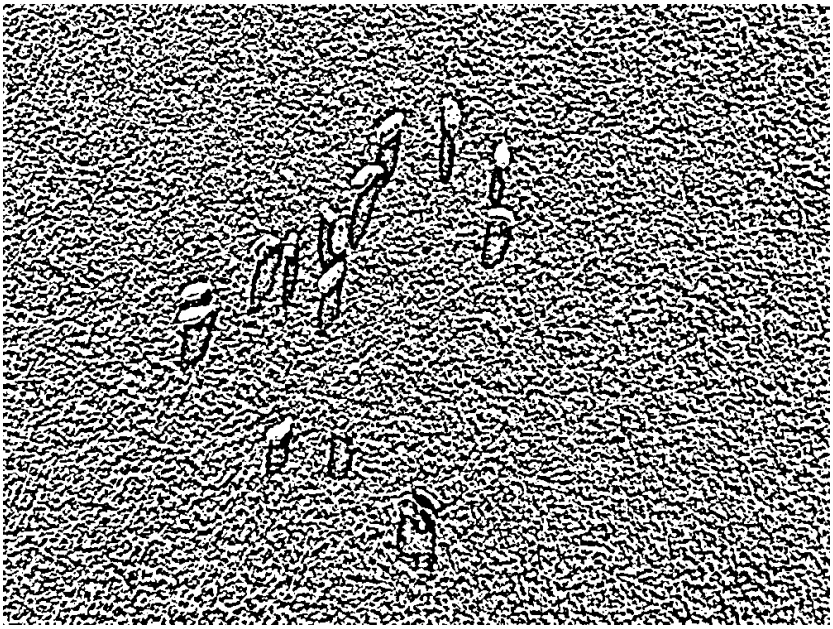


(a) Adaptive Mean Thresholding applied to grayscale image.



(b) Edges found in image with Adaptive Mean Thresholding applied to grayscale image.

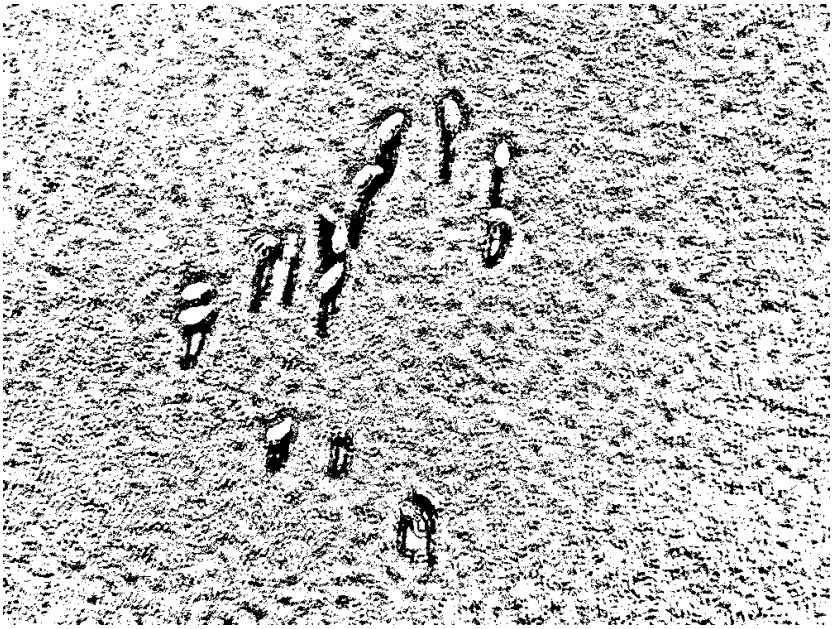
Figure 3.22: Adaptive Mean Thresholding on grayscale image. There is too much noise in the image.



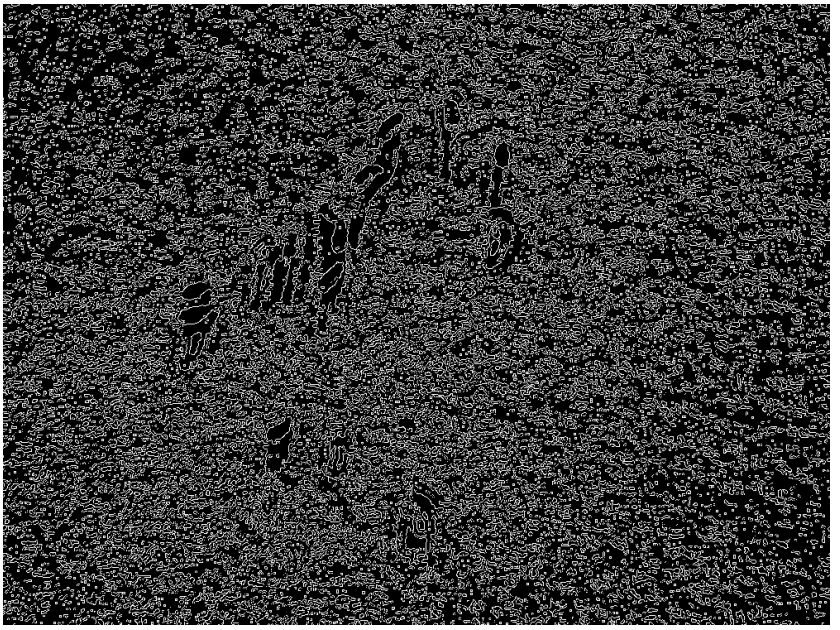
(a) Adaptive Mean Thresholding applied to Gaussian filtered grayscale image.

(b) Edges found in image with Adaptive Mean Thresholding applied to Gaussian filtered grayscale image.

Figure 3.23: Adaptive Mean Thresholding on Gaussian filtered grayscale image. There is too much noise in the image.



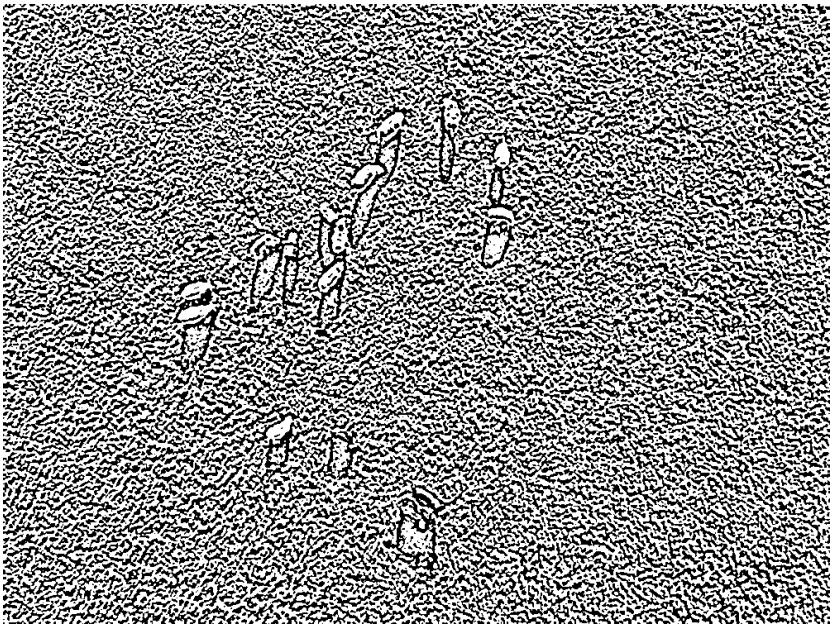
(a) Adaptive Mean Thresholding applied to Bilateral filtered grayscale image.



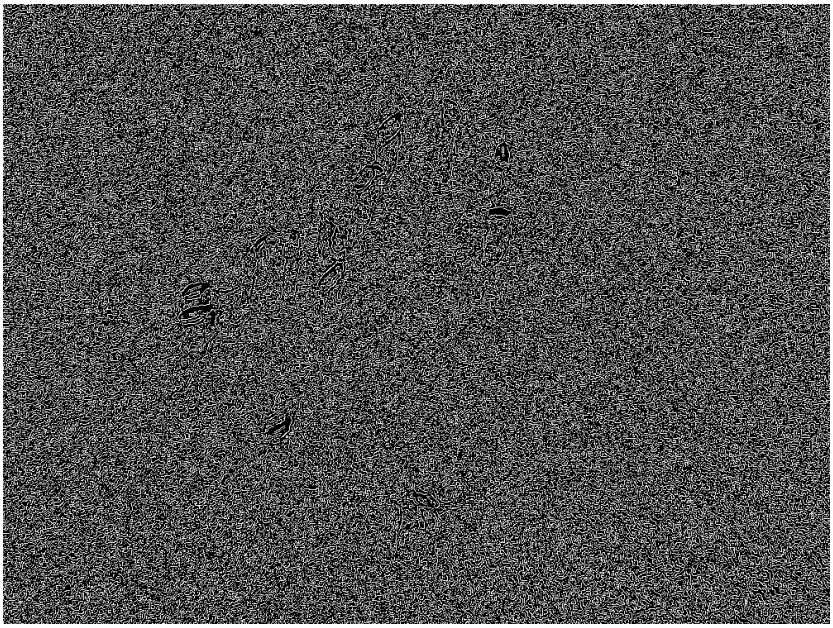
(b) Edges found in image with Adaptive Mean Thresholding applied to Bilateral filtered grayscale image.

Figure 3.24: Adaptive Mean Thresholding on Bilateral filtered grayscale image. There is too much noise in the image.

Figure 3.25: Contours found in image with Adaptive Mean Thresholding applied to Bilateral filtered grayscale image. It does find one black sheep, but also many false positives.

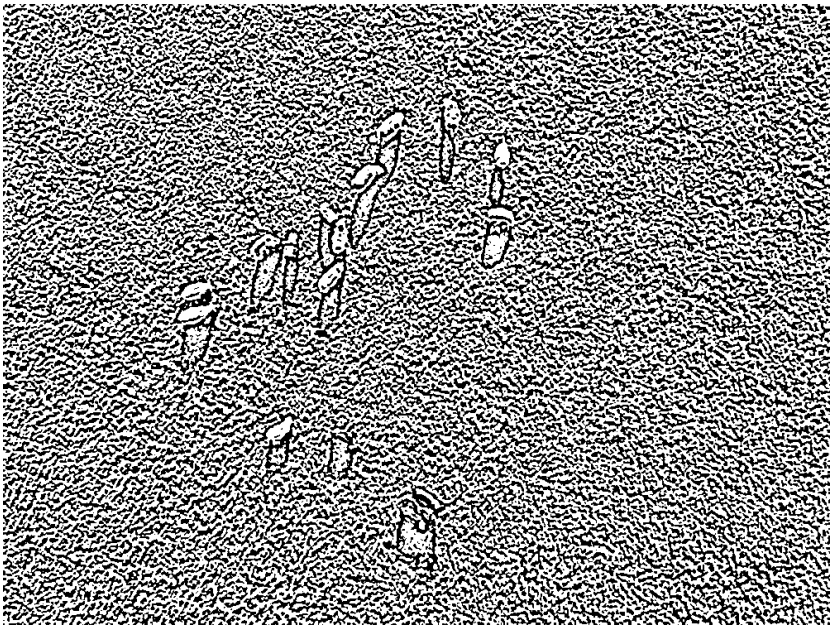


(a) Adaptive Gaussian Thresholding applied to grayscale image.

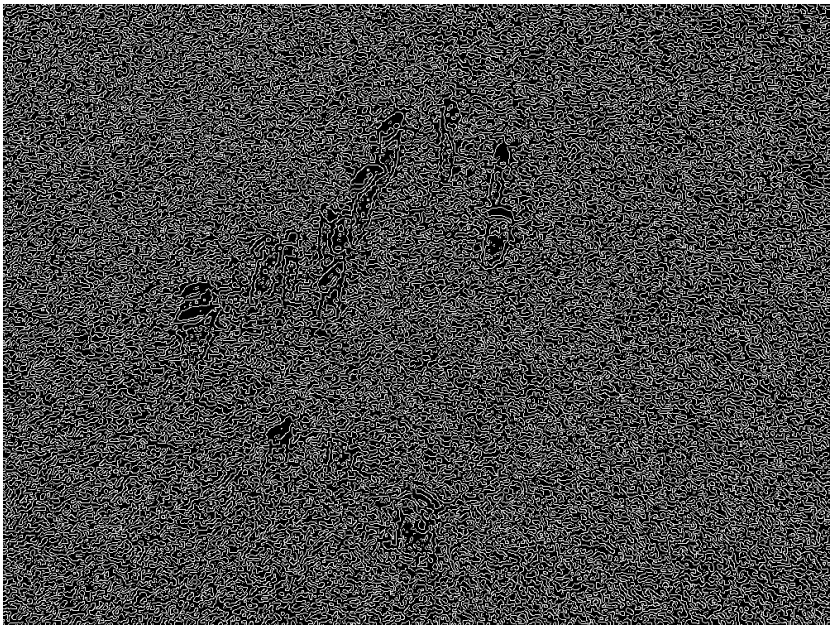


(b) Edges found in image with Adaptive Gaussian Thresholding applied to grayscale image.

Figure 3.26: Adaptive Gaussian Thresholding on grayscale image. There is too much noise in the image.

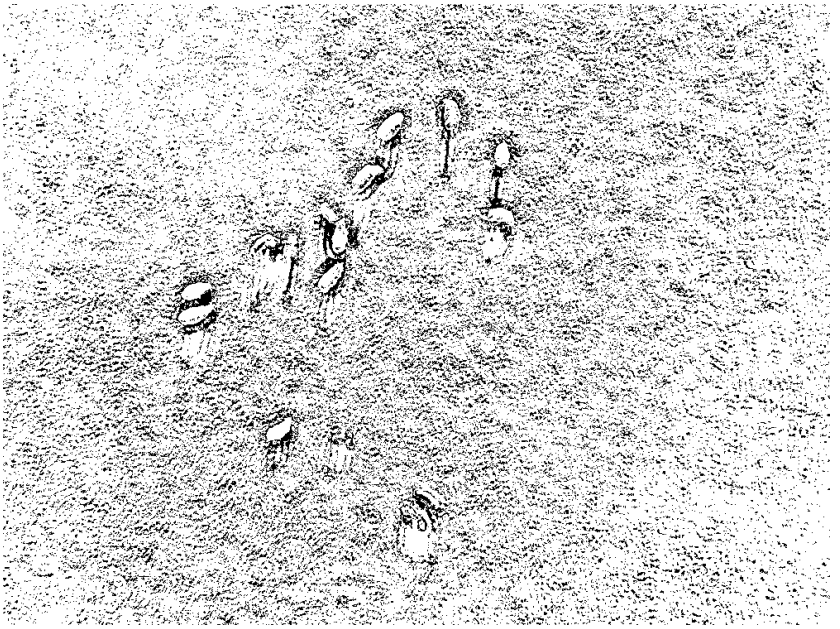


(a) Adaptive Gaussian Thresholding applied to Gaussian filtered grayscale image.

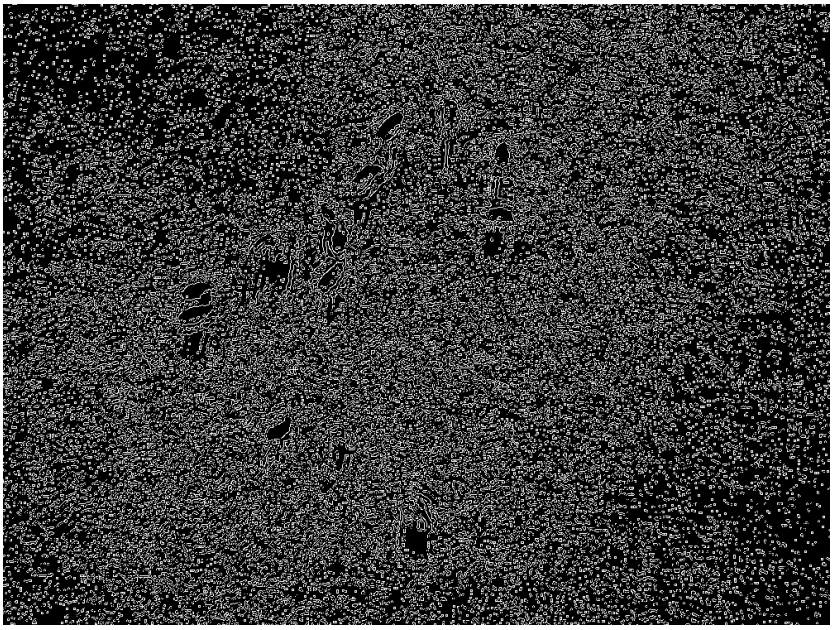


(b) Edges found in image with Adaptive Gaussian Thresholding applied to Gaussian filtered grayscale image.

Figure 3.27: Adaptive Gaussian Thresholding on Gaussian filtered grayscale image. There is too much noise in the image.



(a) Adaptive Gaussian Thresholding applied to Bilateral filtered grayscale image.



(b) Edges found in image with Adaptive Gaussian Thresholding applied to Bilateral filtered grayscale image.

Figure 3.28: Adaptive Gaussian Thresholding on Bilateral filtered grayscale image. There is too much noise in the image and the sheep's edges are hard to make out in **Figure 3.27b**.

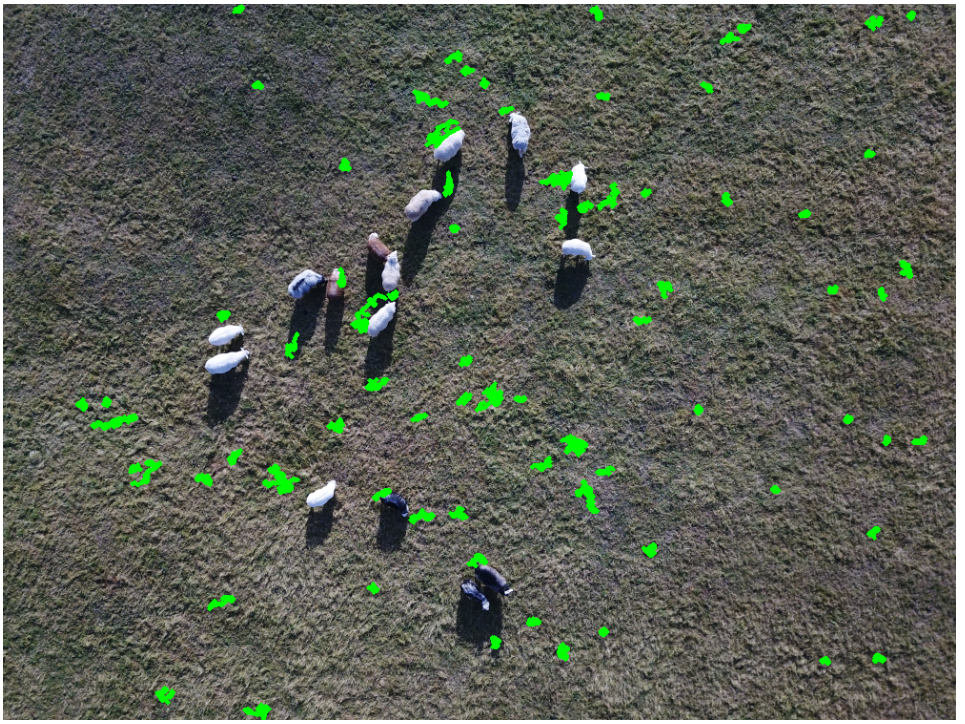
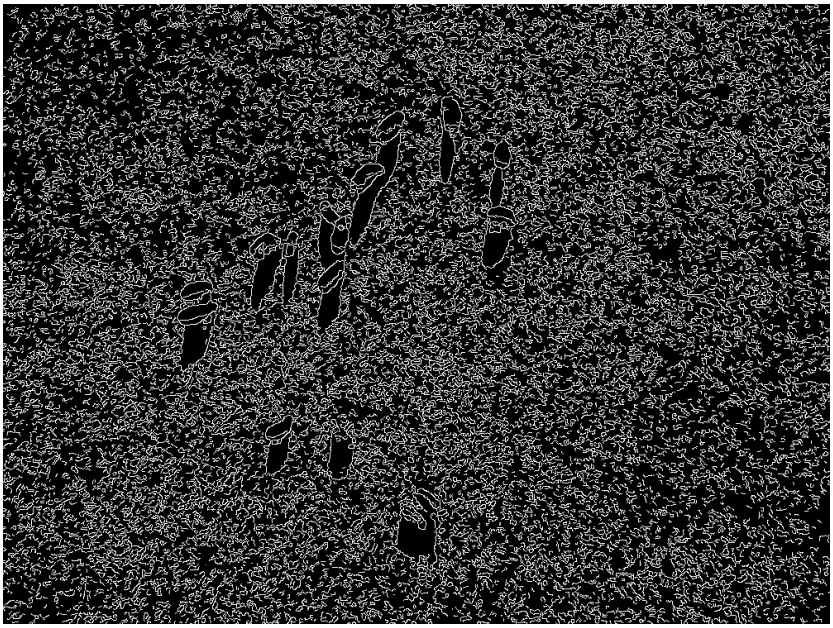


Figure 3.29: Contours found in image with Adaptive Mean Thresholding applied to Bilateral filtered grayscale image. It finds some of the black sheep, but also many false positives.

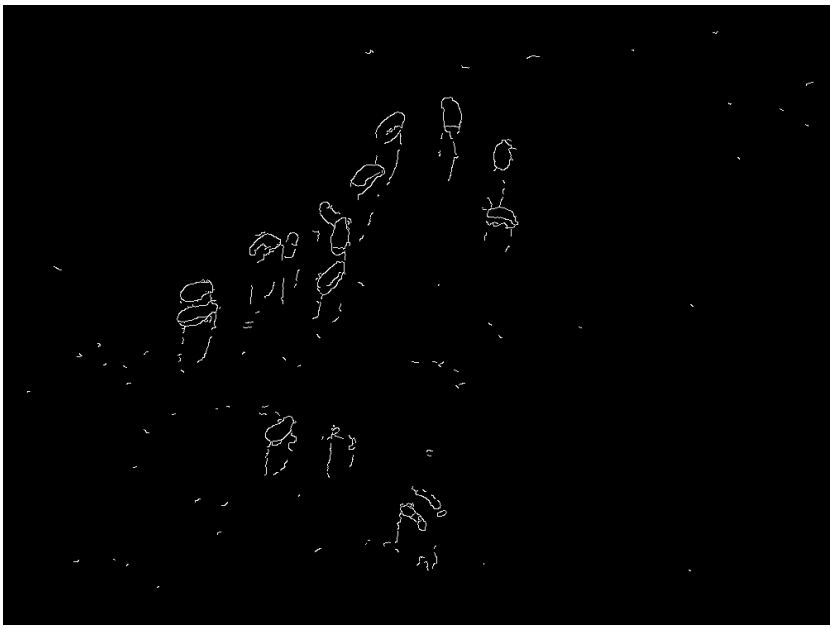
(a) The result of applying Dilation to the original image in **Figure 3.1**. It makes the objects appear bigger.

(b) The result of applying both Dilation and Bilateral Filter to the original image. It makes the objects appear bigger as well as smoothening the background.

Figure 3.30: Dilation applied to image without and with Bilateral Filter.



(a) Edges detected when applying Canny Edge Detection to the Dilated image in **Figure 3.30a**. We can see that there are a lot of disturbances caused by the uneven background.



(b) Edges detected when applying Canny Edge Detection to the Dilated and Bilateral filtered image in **Figure 3.30b**.

Figure 3.31: Edges detected in Dilated images without and with Bilateral Filter. As seen in the images above, the result is much better when applying the Bilateral Filter.

(a) Contours found in the Dilated image.

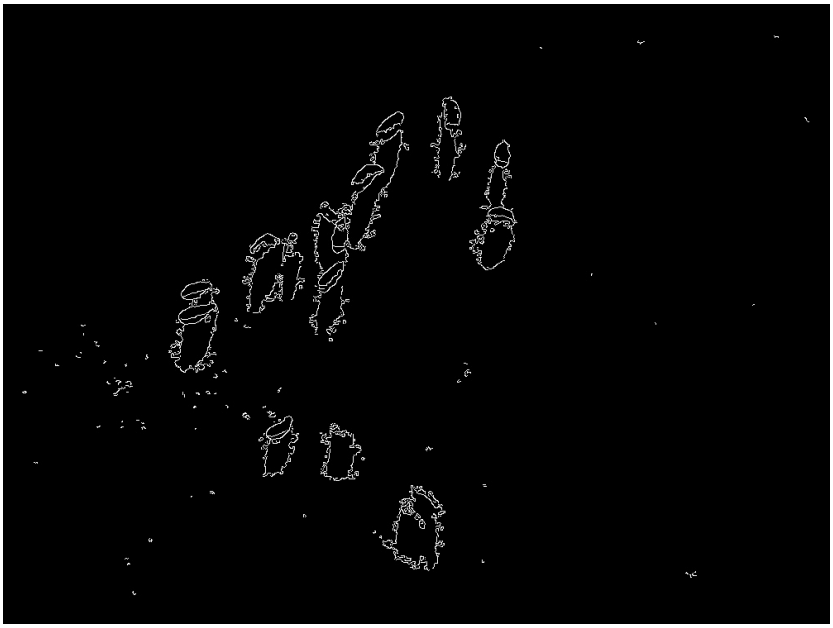
(b) Contours found in the Dilated and Bilateral filtered image.

Figure 3.32: Contours found in the Dilated image, with and without Bilateral Filter, after filtering out the discontinuous edges. We can see that there are quite a few irrelevant edges found in **Figure 3.32a** without the Bilateral filter, while in **Figure 3.32b** all the found edges are relevant, but not all sheep are detected.

(a) Meanshift Filter applied to the image in **Figure 3.1**.

(b) Meanshift Filter and Bilateral Filter applied to image in in **Figure 3.1**.

Figure 3.33: Meanshift Filter applied to the original image without and with Bilateral Filter.



(a) Edges Found when applying Canny Edge Detection to the Meanshift filtered image in **Figure 3.33a**.

(b) Edges found when applying Canny Edge Detection to the Meanshift- and Bilateral filtered image in **Figure 3.33b**.

Figure 3.34: Edges found in Meanshift filtered images, without and with Bilateral Filter. We can see that applying the Bilateral Filter reduces the noise.

(a) Contours found in the image after applying the Meanshift Filter to the image in **Figure 3.1**.

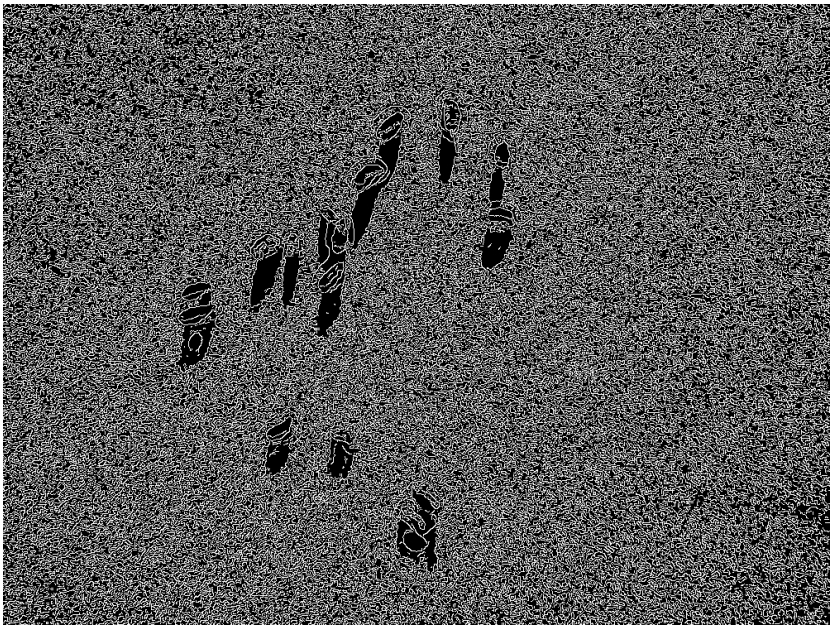
(b) Contours found in the image after applying the Meanshift Filter and the Bilateral Filter to the image in **Figure 3.1**.

Figure 3.35: Contours found in image with Meanshift Filter, without and with the Bilateral Filter. From these images we can see that we have problems detecting some of the sheep even after applying the Meanshift Filter.

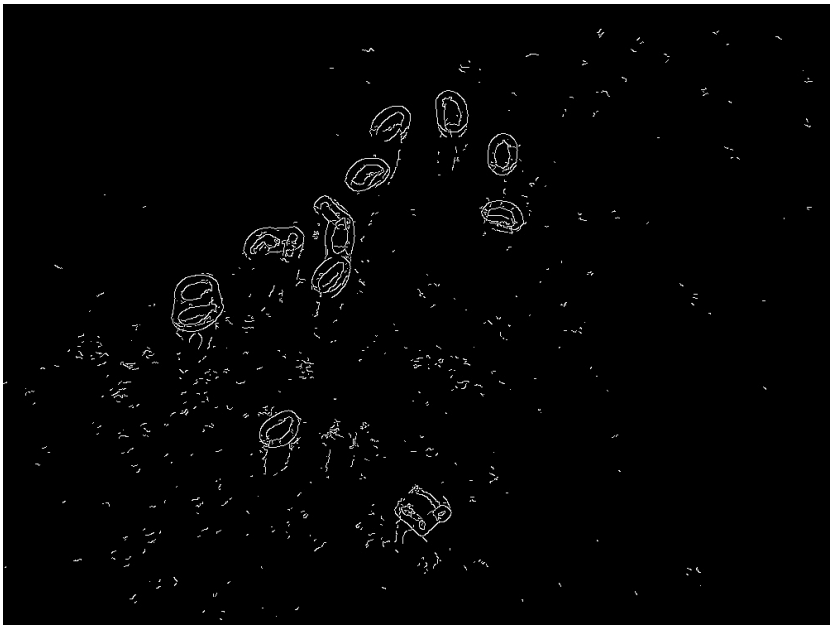
(a) Shadow Removal as explained in **Section 3.3.3** applied to the image in **Figure 3.1**.

(b) Shadow Removal as explained in **Section 3.3.3** applied to the image in **Figure 3.1** after applying the Bilateral Filter.

Figure 3.36: Images with shadows removed using the method explained in **Section 3.3.3**, without and with the Bilateral Filter.

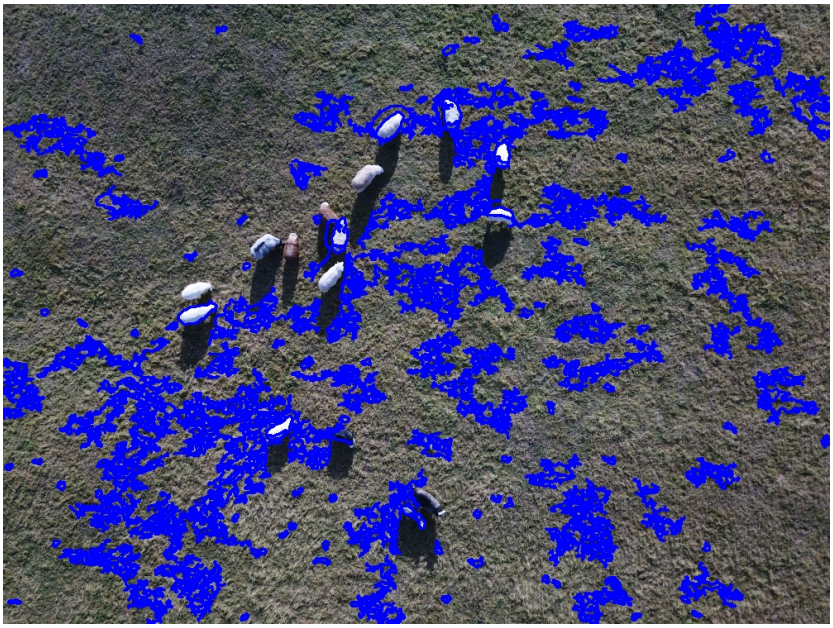


(a) Edges detected using Canny Edge Detection in image from **Figure 3.38a** with shadows removed.



(b) Edges detected using Canny Edge Detection on the Bilateral filtered image from **Figure 3.38b** with shadows removed.

Figure 3.37: Edges detected in images with shadows removed. We can see that applying the Bilateral Filter removes a lot of noise from the background while detecting most of the important edges.

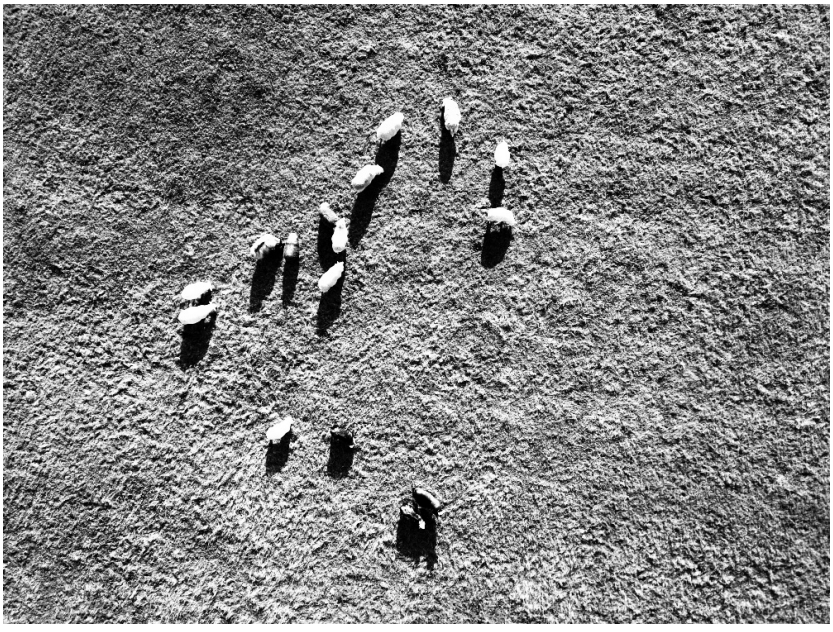


(a) Contours found in the image in **Figure 3.1** after removing the shadows. We can see that the noise from the background is so prominent that we achieve bad results.

(b) Contours found in the image in **Figure 3.1** after applying the Bilateral Filter and removing the shadows.

Figure 3.38: Contours found after removing the shadows, without and with the Bilateral Filter. We can see that the method detects most of the sheep, and even two of the black ones.

(a) Histogram Equalization applied to the image in **Figure 3.1**, done over the intensity channel in the YUV color space.



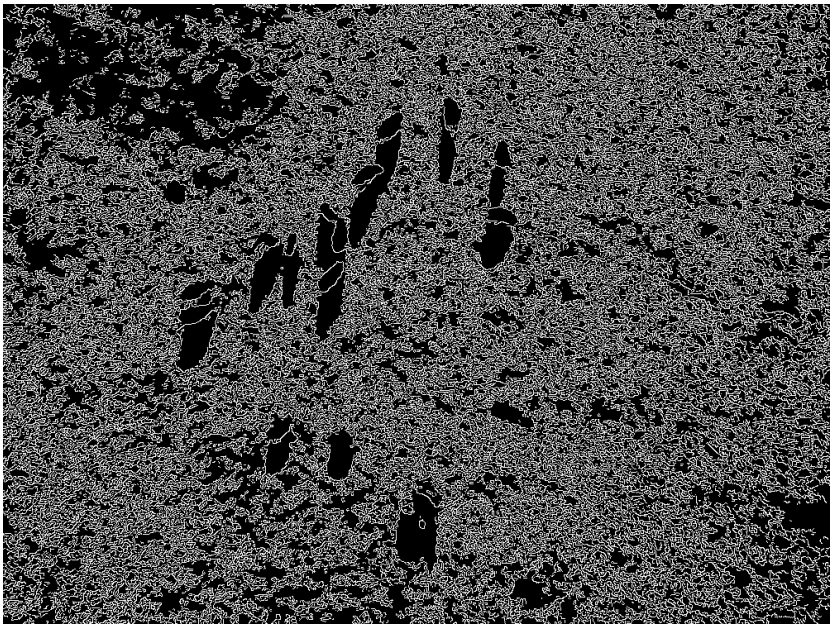
(b) Histogram Equalization applied to the image in **Figure 3.1** after converting it to grayscale.

Figure 3.39: Histogram Equalization applied to image after converting it into two different color spaces.

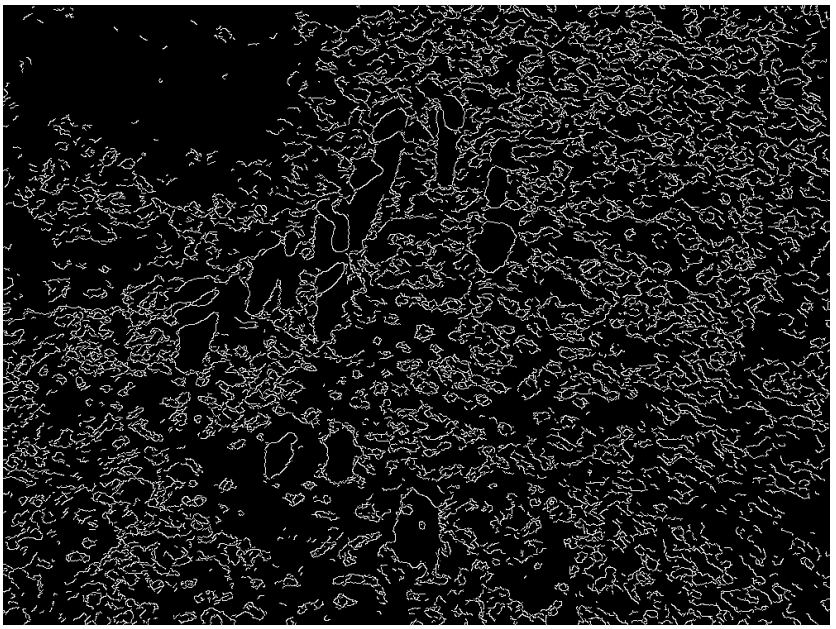
(a) Histogram Equalization applied to the image in **Figure 3.1**, done over the intensity channel in the YUV color space, after applying the Bilateral Filter.

(b) Histogram Equalization applied to the image in **Figure 3.1** after converting it to grayscale and applying the Bilateral Filter.

Figure 3.40: Histogram Equalization applied to the image as in **Figure 3.39**, but after applying the Bilateral Filter first.

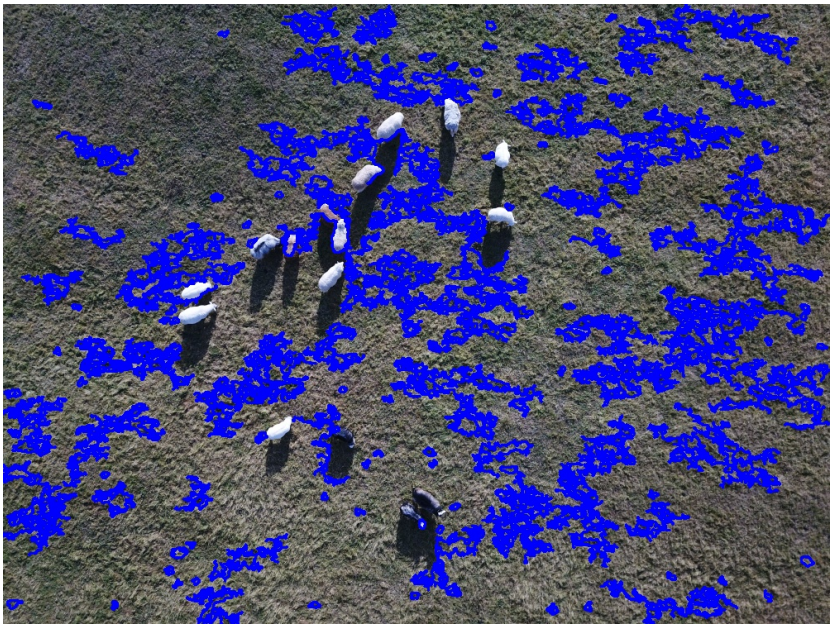


(a) Edges detected in the Bilateral filtered image from **Figure 3.40a**, with Histogram Equalization over the intensity channel in the YUV color space.



(b) Edges detected in Histogram equalized grayscale image from **Figure 3.40b** with Bilateral Filter.

Figure 3.41: Edges detected in Histogram equalized and Bilateral filtered images. Here, we have only shown the Edge Detection after applying the Bilateral Filter, as Edge Detection on the images in **Figure 3.39** contains even more noise, and leads to no good results.



(a) Contours found in the image in **Figure 3.1** after applying the Bilateral Filter and Histogram Equalization over the intensity channel in the YUV color space.

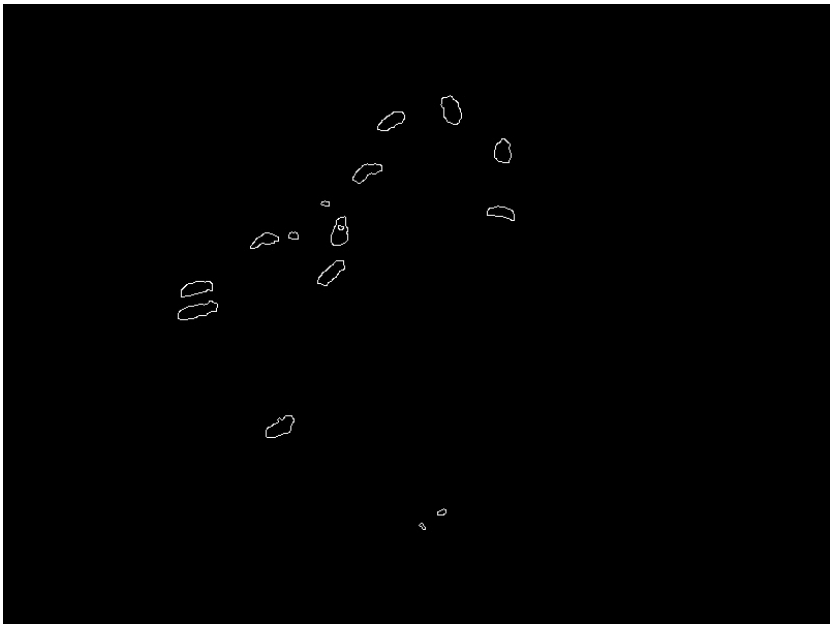
(b) Contours found in the image in **Figure 3.1** after converting it to grayscale and applying the Bilateral Filter and Histogram Equalization.

Figure 3.42: Contours found in Histogram equalized and Bilateral filtered images. As seen in these images, Histogram Equalization seem to create more noise without detecting more sheep.

(a) Black and dark gray in the image after removing all other colors. As seen can only the shadows be recognized.

(b) White areas in the image after removing all other colors. Most of the white sheep are still in the image.

Figure 3.43: Result after removing all other colors than black or white using a binary mask.



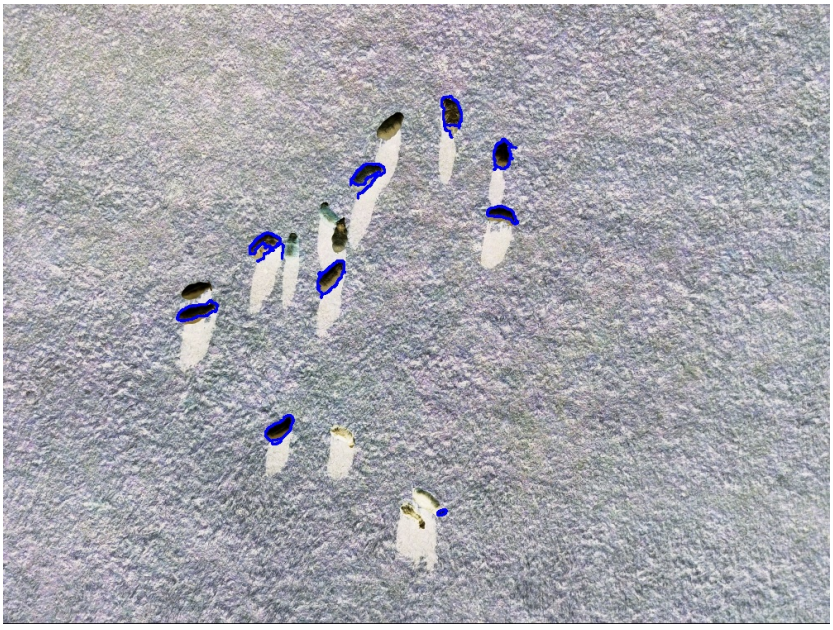
(a) Edges detected in the image in **Figure 3.43b** using Canny Edge Detection.

(b) Contours found using the method of removing all other colors than the desired one, drawn on the original image. As we can see are all the white sheep detected in this image.

Figure 3.44: Edges detected and contours found in the image in **Figure 3.43b**.

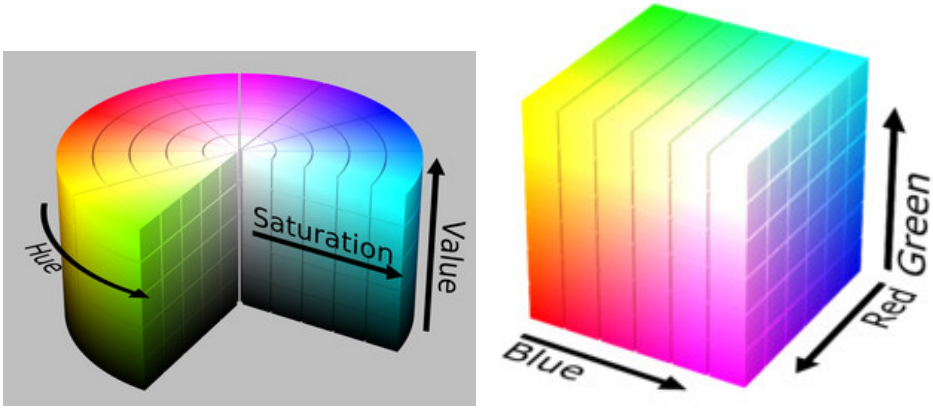
Figure 3.45: Negative of the image in **Figure 3.1**. Every pixel in this image has the opposite color value compared to the original image.

(a) Contours found in the image in **Figure 3.1** after applying the Bilateral Filter and Dilation.



(b) Contours found in the image in **Figure 3.1** after applying the Bilateral Filter and Dilation, and setting every pixel value to its opposite value, `(R,G,B) = (255-R,255-G,255-B)`.

Figure 3.46: Comparision on contours found in image after applying Bilateral Filter and Dilation with the same parameters, with original color values and negative color values respectively. We can see that the contours are identical in both images.



(a) HSV color space model. The hue channel is the color channel, the saturation channel shows how saturated the color is and the value channel represents the color intensity.

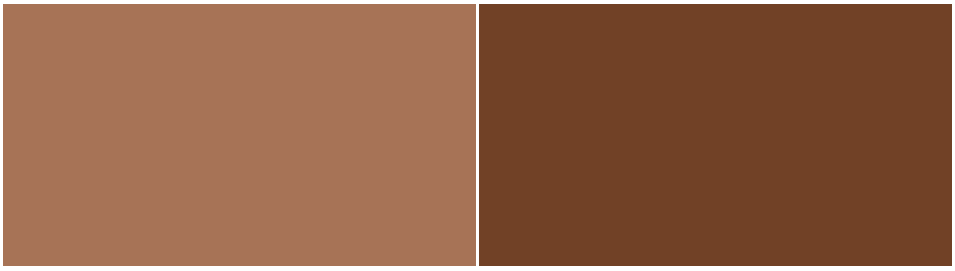
(b) RGB color space model. One channel represents the level of blue in the color, the other the level of red in the color and the last the level of green in the color.

Figure 3.47: HSV and RGB color spaces.

(a) HSV color 0°,0%,0%.

(b) HSV color 0°,0%,48%.

Figure 3.48: Lower and upper black boundary colors used to find sheep with the HSV colorspace.



(a) HSV color 21°,43%,63%.

(b) HSV color 21°,58%,42%.

Figure 3.49: Lower and upper brown boundary colors used to find sheep with the HSV colorspace.

Chapter 4

Results and Discussion

In this chapter, we present the results from the solutions researched in **Chapter 3**. Furthermore, we discuss the positive and negative aspects of each solution, and the reasons why the solutions are effective or not. We have presented numerical results for only the solutions that showed promising results when we applied them to a couple or more different images from the dataset. In other words, if the solution detected more than **50%** of the sheep in the test image in **Figure 3.1** and detected objects in less than **75%** of our images, we have presented the results numerically. These results are presented in **Table 4.5**, in addition to in each of the subsections.

In the following sections, as well as in **Chapter 3**, we have included a number of images. This is to show the results in more detail, and to better explain why the solutions work well, work or do not work.

4.1 Dataset

Before we collected the dataset we had decided on a flight height for the UAV of 50 meters above ground for capturing of the majority of our images. This turned out to be a challenge, as the terrain was more varying than we anticipated, and for this reason, the images are taken at many different heights. A lot of the images were actually captured much higher than our preferred height, as we were standing at the top of a hill most of the time. This made it a challenge to implement the solutions, as we had to consider images taken at different heights and consequently of sheep at different scales when adjusting the methods and parameters.

It is important to take the varying altitude into account, as a good number of the sheep in Norway are wandering in mountain terrains which are very varying in altitude. With the UAV you can adjust the desired height, but when you fly it out of sight it is hard to know if it flies over a hill or mountain. A solution to this problem could be to find a way the program could adjust the height automatically if it flies over mountains or down hill-

(a) **(b)**

Figure 4.1: Color difference in images taken only seconds apart, due to change in lighting.

sides etc. That way the farmer could know that all the images are taken under consistent conditions. If the images would have been captured at more similar heights, our solutions would possibly have given better results.

Another solution is to take the altitude into account when choosing the range at which we want to capture the contours. We could do this by using the GPS coordinates from the captured image to extract the height above sea level for that exact position. This height could in turn be compared with the height above sea level of the UAV, to know the height above ground at that exact point. One way to get the height above sea level from GPS coordinates is to use the Open-Elevation API [27], but in our case, the API is not able to extract the desired height data at our exact location. We have not yet found a way to do this, but both of these solutions could be interesting to look at in future work.

When we collected the images, the autofocus feature was activated. This made the colors appear very different in many images, as shown in the two images in **Figure 4.1** which were taken two seconds apart. This could probably be solved by either using manual focus instead of autofocus, or by preprocessing the images in some way. We believe that by making the images more similar in color, it would be easier to develop a solution with good results.

Another source of error is a problem with the quality of the UAV camera we used to capture the images in the dataset. The problem is a slight blur in the lower right corner, as seen in the cropped image in **Figure 4.2**. This is not a general problem, but might have affected our results in a negative way.

4.2 Method Evaluation

When we evaluate how good the solutions we have implemented are, we look at a few different aspects. The most important factor is how many black sheep we have found using

Figure 4.2: Illustration of defect on the camera lens with a blurry lower right corner.

the solution. This can be measured in a few different ways. One way is to just count how many sheep that are found in each image and then sum it up and compare the total number of sheep with the number of sheep found.

A second way is to count distinct sheep, namely if a sheep is found once in the dataset we do not have to look for that specific sheep in the rest of the images because we have already located the sheep. In other words, if we had found Sheep001 in one image, it would be labeled as ”found” for the entire dataset, so we would only have to look for Sheep002, Sheep003, Sheep004 and so on, in the rest of the images. The challenge with this method is to identify the different sheep in all the images, which we have found almost impossible and very time consuming. Because of this, we did not count the sheep in this way.

A third way to count how many sheep have been found, is to decide that if one sheep is found in the image, then the rest of the sheep in the image is also marked as found. Meaning that it will be sufficient to detect one sheep in each image. This tactic can be used because naturally if you find one sheep you will find the one next to it as well when manually bringing the sheep home. As sheep are flock animals and travel in herds, they will most likely stay together when they move. We will refer to this method of counting as **Any sheep found**.

	Percentage of non-white sheep found	Any sheep found
Bilateral Filter	2,2%	86,7%
White Areas	N/A	96,9%

Table 4.1: The left column shows the percentage of non-white sheep found with the previous solutions implemented in our Project Assignment [30]. The right column shows how many sheep were found with our previous solutions when assuming that we find all the sheep in the image if we find at least one.

Both the first and third way of counting have been included in **Table 4.1** for the solutions where running it on the entire dataset was expedient.

The last aspect we consider when evaluating our solutions is the number of false positives. A solution may find almost all the black sheep, but if it finds every tree in the forest as well, it might not be a good solution after all. A false positive is the same as a falsely detected object, namely anything that is found that is not a sheep. Some false positives are of course not a problem. A high number is however a problem.

We evaluate the false positives by counting the number of images in the dataset where at least one object is detected and compare it to the total number of images. This will later be referred to as **Objects found**. This is not the best way to evaluate the number of false positives, as one image is likely to contain more than one falsely detected object, but it gives us an indication. Counting every detected object would be too big of a job as some of our images contained more than 50 objects. We therefore decided not to do this.

Unlike with the other evaluated aspects, having a high number of objects found is bad because it indicates many false positives, but so is a low one because that indicates few sheep detected. The dataset consists of **43.8%** images containing sheep, so that would be the ideal value of objects found. In general this problem could have been avoided with a thermal camera. We could have eliminated the false positives by using a combination of that and the solutions we describe in this chapter.

4.3 Previous Solutions

In our Project Assignment [30] we developed a solution using Bilateral Filter and ran this solution on the dataset. We calculated the results based on our goal of finding the white sheep. If we change the focus and calculate the results based on black and brown sheep, we can see how effective our previous solutions are for our current problem. As we can see from **Table 4.1** only **2.2%** of the non-white sheep were found with the solution using Bilateral Filter. This is of course unfortunately a very poor result. It means that out of the **1128** non-white sheep, we found **25**. The reason we divide the sheep into white and non-white sheep is that the black, brown and gray sheep can be hard to separate and classify.

Another way of viewing the results, as discussed in **Section 4.2**, is to consider the fact

that sheep live in herds. We can therefore look at the results from the two previous solutions, and decide that if we managed to find at least one sheep in an image, we have found all the sheep in that image. In **Table 4.1** we can see the results using this strategy. As we then can see, the percentage of **Any sheep found** have increased to **86.7%** with the solution with Bilateral Filter and **96.9%** with the **solution searching for white areas**. This means that of all of the **5472** sheep in the images, we have been able to find **5302** of them with the best solution. This is a very good result considering the brown and gray ones are really hard to find. This way of analysing the result will probably be the most realistic way of finding the sheep, since you just need to find one in a herd. Then you have found them all.

4.4 Thresholding

As the results in **Section 3.2** show, the images with no filter and Gaussian Filter applied, have too much noise. For this reason and previous experience with filtering in our Project Assignment [30], we decided not to find contours in these images. On the contrary, the results in **Section 3.2** where we applied Bilateral Filter to the images gave promising results. Therefore we chose to find contours in images where we applied the Bilateral Filter. Hence, these are the results we address in this section.

To make the solutions more effective we could have chosen the parameters more carefully. We experimented with different options, but more trying and failing would most likely have given a more optimal solution. In addition the color difference discussed in **Section 4.1** will affect these solutions more than others, because we look for a certain color value. If we had a consistent color, it would be easier to adjust the parameters according to the background and optimize the solutions.

4.4.1 Binary Thresholding

In **Figure 3.7** we see there is a lot less noise which makes it easier to find contours. As we can see from **Figure 3.8** the solution finds all the black sheep in the image, but it also finds the sheep's shadows. Because of this we not only found the black sheep, but the brown and the white as well. In this particular terrain and weather it worked out really well. However, if it would have been cloudy or in a forest area, the shadows wouldn't be that visible. And there are many other elements in nature that create shadows, for example other animals, trees, bushes etc. Removing the shadows from the image could have been a solution, but as discussed in **Section 4.5.3**, this is complex. In addition, in this image the pasture has a very light color, meaning that it is easy to use thresholding. Also, even this image has some false positives, that are just dark spots in the pasture.

We ran the solution on the entire dataset, and found that it returned **99.9%** detected objects. This means that it detects objects in **823** of the **824** images. Therefore we did not count the number of sheep detected. As we can see in **Figure 4.3a** we find black sheep in the pasture, which is good, but it also finds several other contours and treetops. In **Figure 4.3b** we see another result with more trees in the image. It would be a significant problem

with using this solution. To prevent too many false positives it would have been an advantage to have a thermal camera we could capture images with as well, in order to only find the sheep.

4.4.2 Truncated Thresholding

As we see in **Figure 3.11** the thresholding works quite well, because we do not find any other edges than the sheep's. In addition in **Figure 3.12** we see by the contours that it actually finds all the black sheep, which is our primary goal. It also finds one of the brown sheep, which is a plus. When we ran the solution on the entire dataset, the results showed that there are a lot of false negatives. False negatives mean that there are sheep in the image that the solution does not detect as sheep. **Figure 4.4a** is a good example. Here it does not find any black sheep at all, but one white. And in most of the images, it does not find any sheep at all. **Figure 4.4b** is an image taken at the same height above ground, as **Figure 3.12** where it finds all the black sheep, but in this case it only finds one of the three black sheep. This means that although the solutions seemed to work well on these images, it might also be coincidental.

4.4.3 Threshold To Zero

From **Figure 3.15** we see that the thresholding works quite well, but it still gives us some noise in the image. It finds all the edges of the sheep, but also many other objects that are not sheep. And from **Figure 3.16** we see that all the black sheep are found. But it still detects a lot of the noise as well. This image has quite a light colored background and no trees and other elements that can be disturbing. We ran the solution on the entire dataset and there are **99.9%** detected objects. This means that the solution detects objects in **823** og the **824** images. They are mostly detected in the areas where there are trees. **Figure 4.5a** is an example of just that. Because of this we did not find it expedient to count how many sheep were detected with this solution.

As mentioned earlier, if we had images captured with a thermal camera as well we could have eliminated most of the false positives. In **Figure 4.5b** all the black sheep are found, which is very good. It also detects a lot of noise here, but it would not be a big problem in this particular image as there are sheep that are found as well. In **Figure 4.6** however, there are no sheep, and still there are many contours. This means that even though the solution actually finds quite a lot of the black sheep, considering all the false positives we get, it is not an optimal solution.

4.4.4 Otsu Segmentation

From **Figure 3.20** we see that there is still a lot of noise in the image after applying the filtering and segmentation. In **Figure 3.21** this is more clear, since there are a lot of contours in the pasture which are not non-white sheep. The main problem here is that it does not find any of the non-white sheep which is why we did not run the solution on the entire dataset. One potential reason for this solution not working all that well, is probably because of the image's histogram as seen in **Figure 3.17c**. The Otsu Segmentation is

specifically developed for bimodal images, images with histograms that have two peaks. This means that there are a high number of pixels for two different pixel values. Our test image only has one peak in its histogram. Considering the fact that the solution both does not detect any non-white sheep and generates many false positives it is not a good solution to our problem.

4.4.5 Adaptive Mean Thresholding

From **Figure 3.24** we see that the Adaptive Mean Thresholding gives us too much noise. It find some of the edges of the sheep, but also a lot of other edges. We see this more clearly in **Figure 3.25** where it only finds one of the black sheep, and a great number of contours in the pasture. Also we notice that the contours are found quite randomly in the image, which leads us to believe that the one non-white sheep it found was by accident. For this reason we did not see it expedient to go further and test it on the rest of our images.

4.4.6 Adaptive Gaussian Thresholding

For this last method we found the same as for the Adaptive Mean Thresholding. It produces too much noise in the image, as seen in **Figure 3.28**. Here it is very hard to make out the edges of the sheep in **Figure 3.28b**. In **Figure 3.29** we see that it actually finds two out of three black sheep. The contours found quite randomly in the image, as in **Subsection 4.4.5** leads us to believe that it might be a coincidence that the solution found the non-white sheep. In addition the amount of noise it detects is too great for it to be beneficial for us to run it on the rest of the images in our dataset.

4.5 Edge Detection on Preprocessed Images

As seen in **Section 3.3**, the tested methods for preprocessing had varying results. Because of promising results in earlier research [30], we chose to apply the methods to be tested on a Bilateral filtered image in addition to the original one. This seemed to give a smoother image and more precise object detection in most cases.

4.5.1 Dilation

Applying dilation to the image seemed to have a positive effect, especially in combination with the Bilateral filter. Without the Bilateral filter we had too many false positives due to disturbances from the background, as can be seen in the image with detected edges in **Figure 3.30a**.

When running the method on our entire dataset, with both the Bilateral filter and dilation and using canny edge detection, we obtained the result presented in **Table 4.2**. We were able to find **79.0%** of the white sheep in the dataset, meaning that we found **3432** sheep out of **4343** white sheep. Of all the sheep, regardless of color, we were able to find **65.4%** of the sheep in the dataset, which is **3584** sheep out of **5468**. An example of sheep detected in a different environment can be seen in **Figure 4.7a**.

	Non-white sheep	White sheep	Any sheep	Any object
Number of objects found	152	3432	353	765
Percentage of objects found	13.5%	79.0%	97.8%	89.2%

Table 4.2: The amount and percentage of sheep detected using Dilation together with Bilateral Filter. The column presenting "**Any sheep**" shows the number of images in which at least one sheep was found, and the percentage of these images out of all the images containing sheep. The "**Any object**" presents the number of images where at least one object was detected, and the percentages of images form the dataset where at least one object was detected. This counts both the sheep and all other objects found, and gives an indication on the amount of false positives in the image.

As explained earlier, we have assumed that if we find one sheep in the image, we believe that we in practice would be able to find all of them. For this method, we were able to find at least one sheep in **354** of the images, which gives us **97.8%** of the sheep.

There is still a problem with finding sheep with other colors than white. We have **1125** colored sheep in our collection, and using this method, we were able to detect **152** of these, giving us just a hit rate of only **13.5%**.

Nevertheless, we have the problem with false positives also in this solution. As can be seen in **Figure 4.7b**, other objects with a similar shape are detected as well, for example gray rocks. We believe that these will always be a problem as long as we avoid using thermal cameras as they can resemble sheep both in shape and color, but different methods will provide different amounts of false positives. Using the dilation method, we find at least one object in **765** of the **824** images, while there are sheep in only **361** of them.

4.5.2 Meanshift Filter

The effect of applying the Meanshift Filter to our example image seems promising, both with and without the Bilateral Filter, as it reduces a lot of noise and evens out the colors. In spite of that, we can see that if we apply the filter to other images with the same parameters, we have problems detecting the sheep, as seen in **Figure 4.8a**. Even if no sheep are detected, we still have the problem with false positives, and as can be seen in **Figure 4.8b**.

Because of these bad signs, we decided not to evaluate this method by counting the found sheep, as it seems to perform so much worse than the previous one. Adjusting the parameters might help us find more of the sheep in **Figure 4.8a**, but then we recieve even more false positives. The main problem with this method might be that all the images are taken at different heights above ground, and the amount of details we would like to keep in each image would therefore be different. So if we decide to smoothen the image less to find the sheep in this exact image, we would keep other disturbing details that keeps us from finding sheep in other images or lead to more false positives.

4.5.3 Shadow Removal

The method we used to try remove shadows was originally meant for removing shadows on scanned images of text. It was not working the way we intended it to.

The result we got when applying our method is the image seen in **Figure 3.36**, where everything except for the objects appear foggy. It seems to detect the white sheep, but has more problems detecting the brown and gray/black ones as they are more similar to the background in color. It seems to detect the sheep quite well, but it also detects every other object, so it gives a tremendous amount of false positives. Both an example of the detected sheep and the false positives can be seen in **Figure 4.9** where the shadow removal method is applied together with the Bilateral Filter.

From researching shadow removal in photographs, we found that this is much more complex than we first thought. What is said to be Simple Shadow Removal [48] consists of multiple mathematical calculations, and as this is supposed to be a preprocessing step before actually doing the object recognition, it was not our top priority. Our main problem is that we get too many false positives, so if we could reduce the amount of falsely detected objects, removing the shadows would most likely not be necessary. Although if the images are captured on a cloudy day, there would be no shadows and we would avoid the problem completely.

4.5.4 Histogram Equalization

Applying Histogram Equalization is supposed to increase the global contrast in the image by spreading out the most frequent intensity values [37]. We decided that this sounded like a good idea for our purpose, to make the objects in our images appear more clearly. Unfortunately, the method makes the background more disturbing than earlier and gives us very poor results as can be seen in **Figure 3.42**. Based on these observations, we decided not to proceed further with this solution.

4.6 Bounding Rectangle

As mentioned earlier, one of our biggest problems when detecting sheep in the images is the amount of false positives. In all the methods with decent results, we have high amounts of false positives. By this we mean that even if we are able to detect a high amount of sheep in the data set, we also detect a lot of other objects. We could always accept a few false positives, as it is better for the farmer to find many sheep and a couple of rocks, than to not find the sheep. But if we find objects in every image, the solution is useless.

The solution with the Bounding Rectangle was an idea to try to limit the amount of contours found. In our earlier solutions, we had filtered the contours on length and area, but we wanted to see if it made a difference if we also filtered the contours on the relationship between the width and the length of the object. As the preprocessing method giving the best results was the *Dilation with Bilateral Filter* as seen in **Section 4.5.1**, we chose to

	Non-white sheep	Any sheep	Any object
Number of objects found	47	339	698
Percentage of objects found	4.2%	93.9%	84.7%

Table 4.3: The amount and percentage of sheep found using the Bounding Rectangle together with the *Solution using Dilation with Bilateral Filter*. The column showing "**Any sheep**" presents the number of images in which at least one sheep was found, and the percentage of these images out of all the images containing sheep. The "**Any object**" column presents the number of images where at least one object was detected, sheep or not, and the percentage of images from the dataset where at least one object was detected. This last percentage indicates the amount of false positives found using this solution.

combine this with the Bounding Rectangle solution to try to limit the huge amount of false positives.

The results of adding the Bounding Rectangle are presented in **Table 4.3**. We can see that the amount of images where at least one object is found is a bit lower, with **84.7%** compared to **89.2%**. However, this is not a very precise measure for the false positives, as one image can contain more than one detected object. An example is presented in **Figure 4.10**, where we can see that most of the false positives in that image are removed after applying the limitations using the Bounding Rectangle. There are **0** sheep in this image, but some of the white rocks has a sheep-like shape, and are therefore hard to avoid detecting.

Nevertheless, adding the Bounding Rectangle limitations, we also lower the total amount of sheep found. One reason for this is that sheep are not always standing up straight. If a sheep is lying down, for example, they have a different shape seen from above, which can impact the height and width of the sheep in the image. The same yields for sheep standing together in a cluster. Also, we can see that the amount of non-white sheep found is much lower after adding the Bounding Rectangle limitations. This can be explained by the darker colored sheep somewhat being detected together with their shadows as one single object, which again cause them to appear in a different shape.

To summarize, adding the Bounding Rectangle limitations to the *Solution using Dilation with Bilateral Filter* decreases the amount of detected false positives as desired, but by doing so it also excludes a lot of sheep. Depending on what is valued higher, one has to decide whether adding these limitations is useful or not.

4.7 Color Detection

To use the colors in the image to find the sheep seemed to work quite well in our earlier research [30], but it does not seem to work well for sheep of other colors than white.

	Non-white sheep	Any sheep	Any object
Number of objects found	N/A	324	532
Percentage of objects found	N/A	89.8%	64.6%

Table 4.4: Here we can see the amount and the rate of objects found in using the *Solution Removing Irrelevant Colors*. **Any sheep** shows the amount of images where at least one sheep was found, and **Any object** presents the amount of images in which at least one object was found, sheep or not. This last rate indicates the amount of false positives found using this solution. **Non-white sheep** is not relevant here, as we studied the solution thoroughly only after removing all other colors than white.

4.7.1 Searching For Black Areas

When searching for black areas by iterating through the pixels in the image and look for regions of multiple black pixels, we saw that there are too many black areas in the image that are not sheep. The results showed way too many false positives and it found ”sheep” in all the images. In **Figure 4.11** there are two examples from the dataset.

4.7.2 Removing Irrelevant Colors

When we tried to remove everything from the image except for the colors inside a desired color interval, we found that it was difficult to filter based on black or brown sheep. There are a lot of dark objects in the images, so this works poorly for detecting objects, as when searching for black areas in **Section 4.7.1**. Regarding brown sheep, it is difficult to distinguish brown from gray when it comes to RGB values, as there is no clear interval of which the color is brown.

Although, we found that this method worked quite well for detecting white sheep, as seen in **Table 4.4**. It did not detect nearly as many sheep in total as the *Solution using Dilation with Bilateral Filter* in **Section 4.5.1**, but if we assume that detecting any sheep in an image means that we'll find all of them, we found **89.8%** of the sheep in the dataset applying this solution.

This method avoided a lot of the previously detected false positives, as they often had other colors than white. Although false positives is still a problem, as there is at least one object detected in **532** of the **824** images in our dataset, resulting in **64.6%**. There are sheep in **361** of the images, and out of the images containing sheep, this method found at least one sheep in **324** images. This means that in addition to detected objects in the images actually containing sheep, there are false positives in **208** images where no sheep are detected. This gives us false positives in **25%** of the images, which still is a lot.

4.7.3 Negative Colors

We wanted to study if changing the color values in the image to the opposite would make any difference, meaning to set the value to `(R,G,B) = (255-R,255-G,255-B)` for every pixel in the image. Applying the Canny Edge Detection method to the image with original color values compared to applying the same method to the image with negative

color values can be seen in **Figure 3.46**. By studying these two images, we can see that the contours found in the two images are identical.

This observation shows that the color has no impact on how the objects are detected, so dark objects on light background is detected as easily as light objects on dark background. The reason why the white sheep are detected more easily than black or brown ones seems to be simply because the color difference between the sheep and the background is greater.

4.7.4 Hue Saturation Value Color Space

In **Figure 4.12** we can see the mask image where the targeted color found in the image is in white and the other colors are removed. As we can see the results are quite poor, there are some spots detected in the mask image, but they are few. For the brown colors, it did not detect anything even though there are brown sheep in the image. There may be several reasons why the results were insufficient. The choosing of the lower and upper color boundaries is the factor with the biggest impact.

For the black boundaries, it is easier to choose, we simply chose the darkest color and another one which was dark gray as can be seen in **Figure 3.48**. Of course the gray color could have been chosen differently, but it is just the value channel that varies. For the brown, there are a vast number of browns in the color spectrum, and to choose the right brown for the sheep is tricky. For this we simply chose brown colors that we found similar to the sheep we had captured in the dataset as in **Figure 3.49**. To illustrate the color problem, **Figure 4.13** shows the color named ”brown” [47].

Figure 4.13: The color named ”brown”. For most people it looks red, therefore to choose a well suited brown color is hard.

It is red and would be a bad choice when looking for brown sheep. This shows that to find the brown sheep is very hard, even if we use color detection and the HSV color space. Therefore we do not proceed with this solution.

4.8 Review Of Results

In general, the *Thresholding* methods gave too many false positives, with Otsu Segmentation, Adaptive Mean Thresholding and Adaptive Gaussian Thresholding giving the worst

results. Binary Thresholding and Threshold To Zero has somewhat lower number of false positives, but they detect the shadows of the sheep rather than the actual sheep when looking for dark objects. The last method, Truncated Thresholding, has nearly no false positives. It detects very few black and brown sheep, and just a few white sheep.

When it comes to the *Edge Detection On Preprocessed Images*, we found that none of the researched methods worked well for finding neither black nor brown sheep. Shadow Removal detected a lot of sheep, but also basically every other object in the images. However, using a more complex method for removing shadows might improve the detection of sheep. This is time consuming, and therefore we did not research this further.

Applying Dilation together with the Bilateral Filter performs better than any of the previous solutions as discussed in **Section 4.3** for white sheep, but still has a high amount of false positives. Meanshift filtering with Bilateral Filter detects fewer sheep than Dilation, and detects even more false positives.

Using the *Bounding Rectangle* to limit the amount of false positives works to a certain degree, but the main problem is that we lack information about the flight height above ground. Therefore we have problems adjusting the parameters, as the sheep differ in size throughout the dataset.

Color Detection seemed to be a good solution for white sheep, but extending this to other colors turned out to be difficult. Neither when explicitly searching for black areas in the RGB color space or black and brown areas in the HSV color space, were we able to distinguish sheep from the background. Detecting edges after inverting the colors to their negatives gave the exact same results as when detecting edges with original colors.

As for the solution removing irrelevant colors, we still had problems with detecting black and brown sheep. Although, this worked reasonably well for white sheep, but as for most of the other methods it also detects a lot of other white objects.

Quantitative results from the most successful methods are summarized in **Table 4.5**. The first column presents the amount of non-white sheep found in the dataset applying the relevant methods, and we can see that the *Solution using Dilation with Bilateral Filter* finds the highest amount with **13.5%**. However, this is not a high rate of colored sheep detected.

Secondly, the table shows the percentage of images in which at least one sheep is found. This is shown in the **Any sheep found** column, and we can observe that in all the tested solutions where this is relevant, the solutions has detected at least one sheep in more than **86%** of the images containing sheep. This is a decent result, and the best results are also in this case provided by the *Solution using Dilation with Bilateral Filter*.

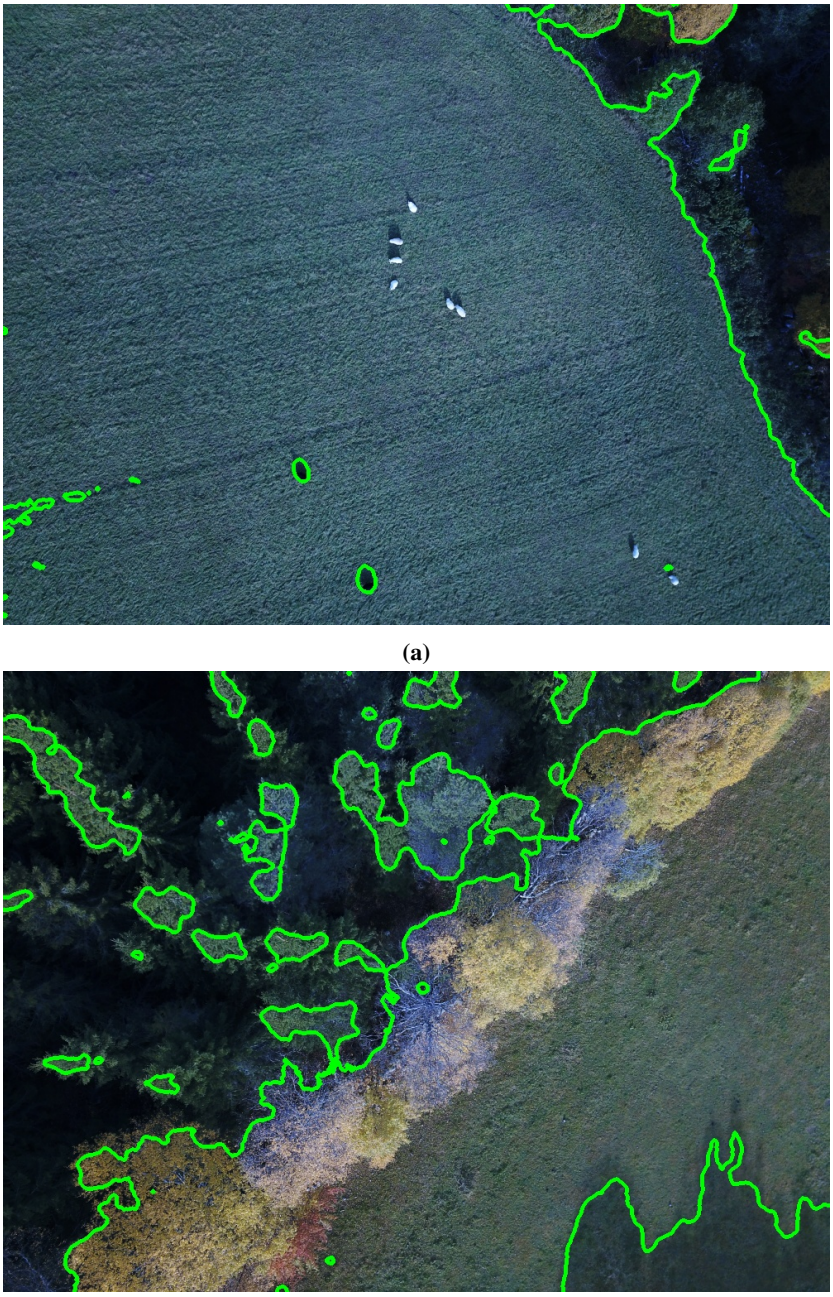
The last column presents the percentage of images in the dataset where at least one object is detected. In this case a high number, for example **99.9%**, is negative, as that shows that the solution detects objects in almost every image in the dataset. A high number indicates

	Non-white sheep found	Any sheep found	Objects found
Bilateral Filter	2.2%	86.7%	N/A
White Areas	N/A	96.9%	N/A
Binary Thresholding	N/A	N/A	99.9%
Threshold To Zero	N/A	N/A	99.9%
Dilation + Bilateral Filter	13.5%	97.8%	89.2%
Bounding Rectangle	4.2%	93.9%	84.7%
Removing Irrelevant Colors	N/A	89.8%	64.6%

Table 4.5: The results obtained when running each of the solutions on our dataset. *Non-white sheep found* shows the percentage of the amount of non-white sheep found in our dataset. *Any sheep found* shows the amount of images in which we found at least one of the sheep, assuming that if we find one sheep, we also find the other nearby sheep. *Objects found* presents the percentage of images where the solution found at least one object.

that the solution finds a lot of false positives and not only sheep. As a reference, there are sheep in **43.8%** of the images in the dataset. From these numbers, we can come to the conclusion that the *Solution Removing Irrelevant Colors* detects the highest rate of relevant objects, but it also has a lower rate of sheep found in total.

To put it concisely, the general problem with all these solutions is that they generate too many false positives. If using any of these solutions, one has to consider whether finding the highest amount of sheep or finding them as effective as possible is more important. As a possible solution to the problem of falsely detected objects, we propose the use of a thermal camera, as discussed further in **Chapter 5**.



(a)

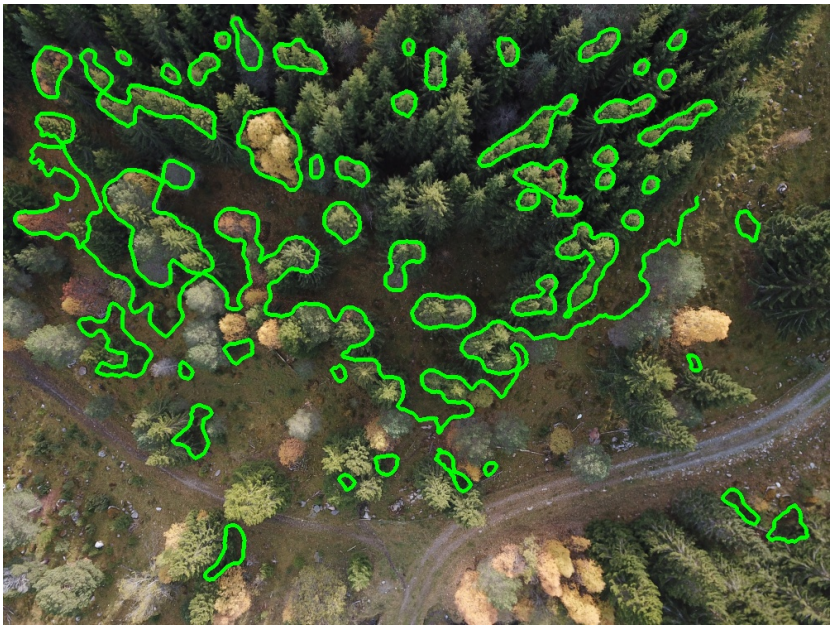
(b)

Figure 4.3: Results with Binary Thresholding on Bilateral filtered grayscale image. It finds the two black sheep in the pasture, but it also finds many false positives in both images, especially in **Figure 4.3b**.

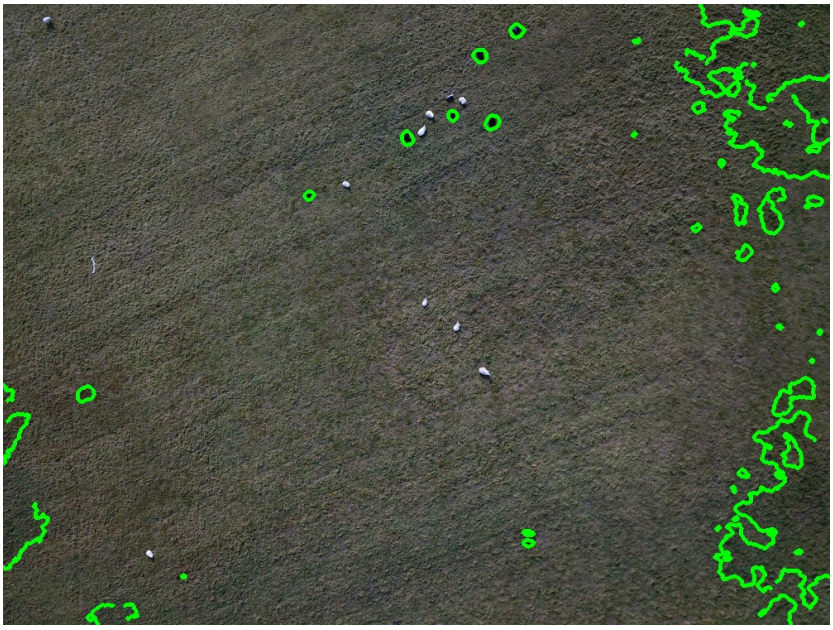
(a)

(b)

Figure 4.4: Results with Truncated Thresholding on Bilateral filtered grayscale image. It found one black sheep and some white, but in **Figure 4.4a** taken farther away, it does not find anything.



(a)



(b)

Figure 4.5: Results with the Thresholding To Zero method on Bilateral filtered grayscale image. Once again, if finds a lot of false positives in both images, but also finds all the black sheep in **Figure 4.5b**.

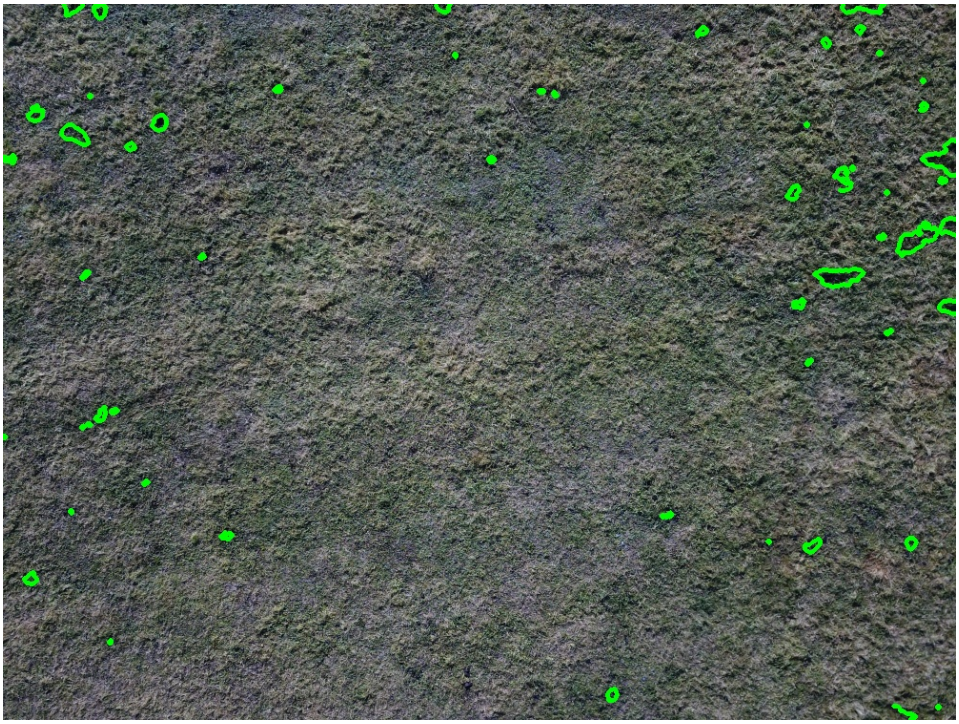
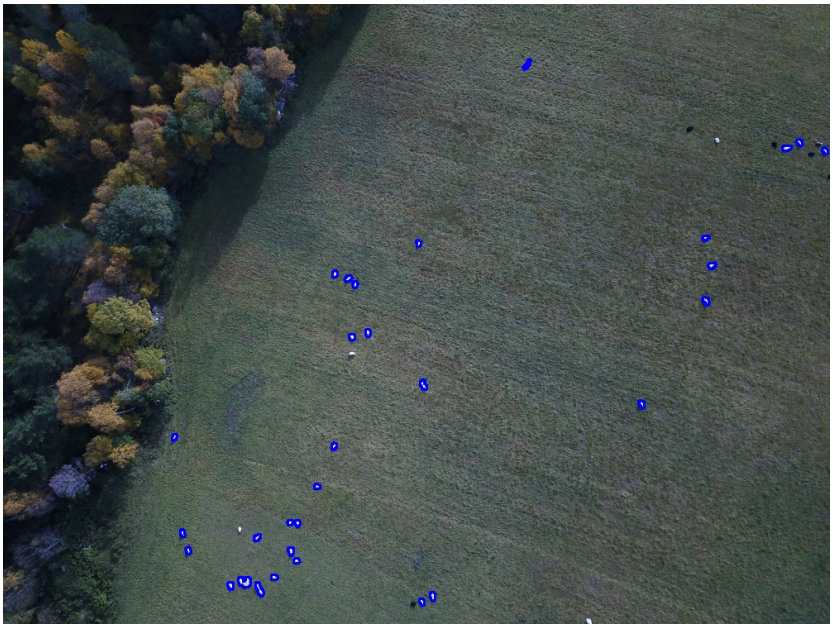
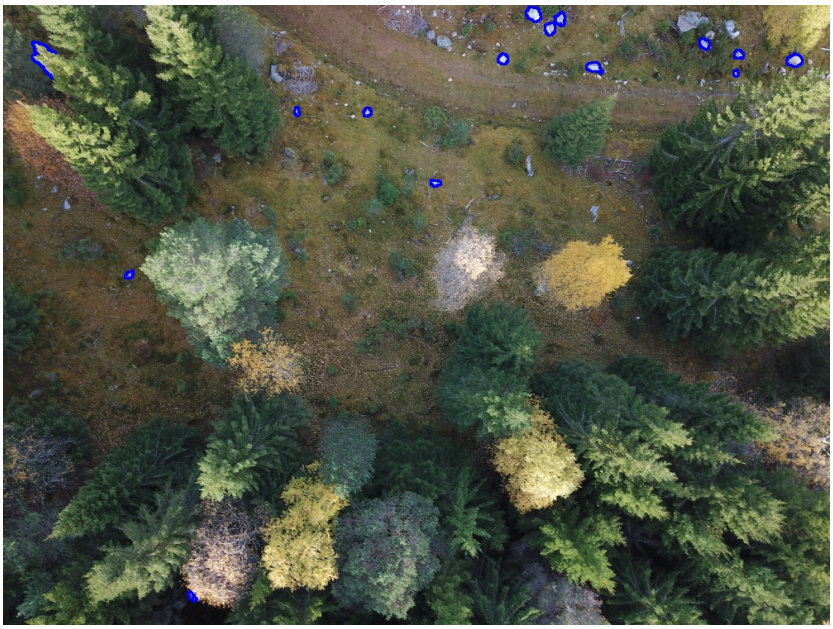


Figure 4.6: Result with the Thresholding To Zero method on Bilateral filtered image. There are no black sheep in the image, and therefore there are many false positives in this image.



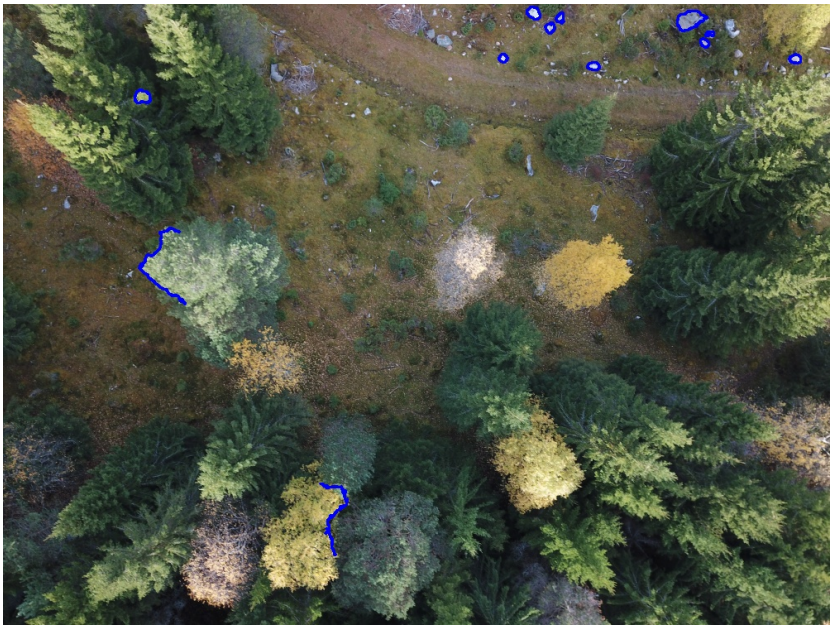
(a) Example of sheep detected when using Dilation with Bilateral Filter. As we can see, most white sheep are detected, but it has problems detecting black sheep.



(b) Example of false positives found when using Dilation with Bilateral Filter. We can see that a lot of irrelevant objects, mostly rocks, are detected.

Figure 4.7: Example of contours found after applying Dilation and Bilateral Filter to images with and without sheep.

(a) Example of poorly detected sheep using Meanshift Filter with Bilateral Filter.



(b) Example of irrelevant objects detected using Meanshift Filter with Bilateral Filter.

Figure 4.8: Example of contours found in images after applying the Meanshift Filter and the Bilateral Filter. We can see that this solution detects a lot of false positives, but has problems detecting the actual sheep.

(a) Example of detected sheep using the Shadow Removal solution.

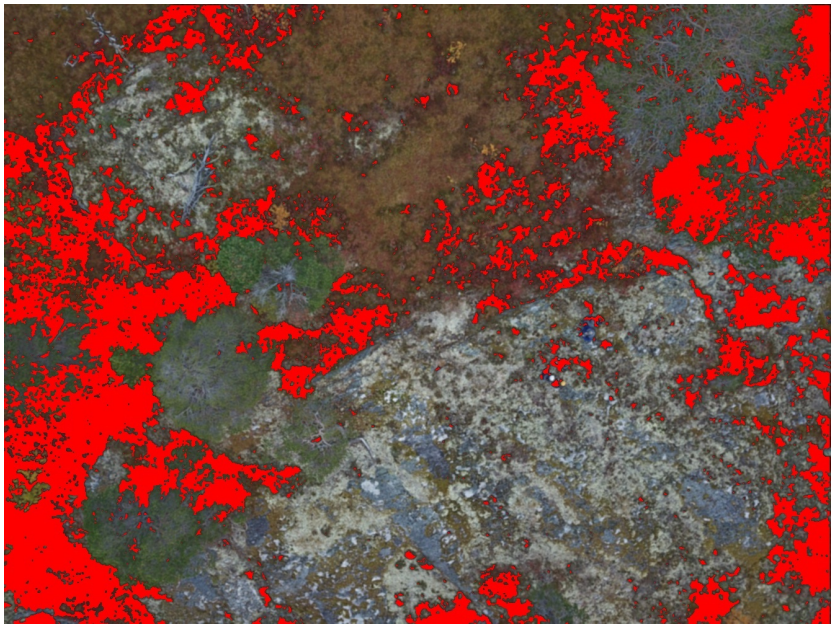
(b) Example of irrelevant objects found using the Shadow Removal solution.

Figure 4.9: Example of contours found in images after applying the Shadow Removal solution. We observe that a lot of sheep are detected, but so are a lot of other objects. Also, sheep are here sometimes detected in clusters, not always as single objects.

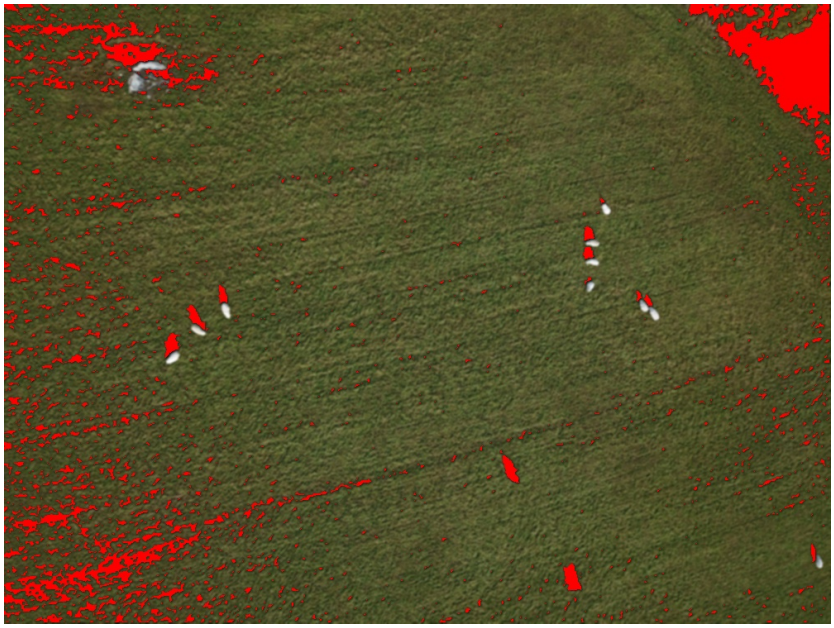
(a) Example of falsely detected objects using the *Solution using Dilation with Bilateral Filter*.

(b) Example of falsely detected objects after adding the Bounding Rectangle to the *Solution using Dilation with Bilateral Filter*.

Figure 4.10: Example of false positives found in the same image before and after adding the Bounding Rectangle limitations to the *Solution using Dilation with Bilateral Filter*, meaning that we exclude objects that has a width/height relationship over **2.0** and under **1.4**.



(a)



(b)

Figure 4.11: Results from Solution Searching For Black Areas. As we can see, there are false positives in both images and it finds the sheep's shadows in **4.11b**.

Figure 4.12: Mask image after the other HSV colors are removed, and only the color we were looking for are the small white dots.

Chapter 5

Future Work

There are still significant work to be done in this field of study, especially since we did not find any optimal solution to the researched questions.

5.1 Detecting Brown Sheep

Finding brown sheep is something we tried briefly during our research, but we did not get any good results with any of our solutions. As far as we are concerned, it is the most difficult colored sheep to find as there is so much terrain that also has a shade of brown. Therefore this is a challenge that needs to be addressed further to explore if there can be developed any good solutions to find these sheep. Although there are not many of them, they still need to be found. However if we combine our solutions with a thermal camera, as we discuss in **Section 5.2**, the task of finding them may be a lot easier.

5.2 Thermal Imagery

As mentioned in **Chapter 2** thermal imagery has been used to find animals previously. We believe that if we put our solutions to use and combine them with images taken with a thermal camera, we can use that data to eliminate most false positives. For example, all the trees and pasture spots found, will not be returned as sheep detected, if we cross check it with the thermal images. That means some of the solutions that gave a great number of false positives, but also a lot of true positives will be good solutions in practice. Of course, as also mentioned in **Chapter 2** thermal imagery can also give us false positives, but this again can be ruled out by using the solutions we have developed in this paper.

5.3 Flight Path Generation

If we are to develop a working system to run on UAVs we also need to generate a flight path so that the farmers do not have to control the UAV manually. The goal is to make a system that the farmer can start up and it would fly over an area given by the farmer and come back and land. Henceforth the farmer would transfer the images to a tablet or computer and the program will give the farmer the GPS coordinates where the sheep are located. Ideally we propose to develop something similar to what they used for fawn detection[21] as seen in **Figure 5.1**. They used the Google Maps Application Programming Interface to

Figure 5.1: Waypoint Editor[21]. With the use of Google Maps Application Programming Interface, they developed a program that allowed them to mark the area they wanted to search with the UAV.

develop a program which allows you to mark the area you want the UAV to search and it will calculate the optimal flight path relatively fast.

5.4 Developing the System

If succeeding in developing a satisfactory solution for sheep detection, a complete system, including a flight path program, would need to be developed. Ideally the UAV would send the images to a tablet or computer whilst searching, together with the location of the sheep.

Chapter 6

Conclusion

The objective of this paper was to study how traditional computer vision methods and image processing could be used to find sheep in UAV images taken in different terrains, focusing mainly on black sheep.

We experimented with many different solutions within different topics of computer vision. Firstly we looked at various ways of thresholding an image, secondly we researched *Edge Detection On Preprocessed Images*, thirdly we tried to take the size of the objects into account and lastly we experimented with different kinds of *Color Detection*.

Most of the *Thresholding solutions* had problems finding the sheep or gave too many false positives. Some of them were able to find quite a few of the black sheep, with the *Binary Thresholding* solution and the *Threshold To Zero* solution as the most effective. Although, none of the *Thresholding solutions* detected the desired amount of sheep.

Neither of the *Preprocessing solutions* seemed to make a significant difference when detecting black or brown sheep, compared to the earlier researched solutions. *Dilation* was the most successful and found many of the white sheep, but it also detected a lot of false positives.

When adding the *Bounding Rectangle* to the *Solution using Dilation with Bilateral Filter*, we were able to filter out some of the false positives by limiting the dimensions of the contours. Although, this would have worked better if we were able to calculate the true size of the sheep.

From studying the detection of colors, we learned that it is very difficult to distinguish black and brown sheep from other colors in nature. However, as there are fewer white objects in mountain areas, white sheep are easier to find using *Color Detection*. In particular we found that the *Solution Removing Irrelevant Colors* provided satisfactory results with one of the lowest amounts of false positives.

To summarize, our research shows that none of the solutions we implemented gave adequate results when looking for black or brown sheep. The highest amount of non-white sheep found was by applying the *Solution using Dilation with Bilateral Filter* to the images in the dataset, detecting **13,5%** non-white sheep.

The *Solution using Dilation with Bilateral Filter* also generated great results when searching for all colored sheep, finding at least one sheep in **97,8%** of the images containing sheep. By adding the *Bounding Rectangle*, we were able to somewhat reduce the amount of false positives. Furthermore, *Removing Irrelevant Colors* also detected a high number of sheep, with less false positives.

The biggest drawback with all the solutions is the high number of false positives. Of all the solutions we studied, the lowest number of false positives was **64,6%** in the *Solution Removing Irrelevant Colors*. These falsely detected objects consist mainly of rocks and other sheep-like objects, both in shape and color, but also of other objects like trees and man-made items. Separating the sheep-like rocks from sheep is nearly impossible without using a thermal camera.

In conclusion, we believe that detecting sheep in a satisfactory way is difficult using a regular UAV color camera alone. Although, the problem with false positives can be solved by combining the *Solution using Dilation with Bilateral Filter* with the use of a UAV with a thermal camera. Detecting non-white sheep will still be challenging, but finding most of the white sheep would be very helpful for the farmers. As thermal cameras have become more available throughout our research, we consider this to be the next step in the retrieval of sheep using UAVs.

Bibliography

[1] Dyrebeskyttelsen Norge: Fanesak tap av sau på beite
https://www.dyrebeskyttelsen.no/tap-sau-pa-beite/,
accessed 24.10.2018

[2] Effect of Environment on Nutrient Requirements of Domestic Animals
https://www.ncbi.nlm.nih.gov/books/NBK232324/,
accessed 15.05.2019

[3] Nortrace AS,
http://www.nortrace.no/, accessed 20.10.2018

[4] Telia,
https://www.telia.no/, accessed 20.10.2018

[5] Telia, NarrowBand Internet of Things
https://old.telia.no/bedriftsmagasinet/telia-forst-i
-norden-med-fremtidens-teknologi, accessed 20.10.2018

[6] Telespor,
https://telespor.no/, accessed 10.10.2018

[7] Radiobjella,
https://telespor.no/produkt/, accessed 10.10.2018

[8] Telia, Verdens største IoT-pilot: skal gjre 1000 sauer smartere
https://old.telia.no/bedriftsmagasinet/verdens-storste-
iot-pilot, accessed 10.10.2018

[9] Telia Dekningskart,
https://old.telia.no/dekningskart, accessed 10.10.2018

[10] Telenor Dekningskart,
https://www.telenor.no/privat/dekningskart//#map,
accessed 10.10.2018

[11] Findmy,
http://findmy.no/om-oss/, accessed 21.11.2018

[12] Findmy: Produkter,
http://findmy.no/produkter/bjeller/, accessed 21.11.2018

[13] OpenCV Geometric Image Transformations,
https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html, accessed 03.12.2018

[14] OpenCV Basic Operations on Images,
https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_core/py_basic_ops/py_basic_ops.html, accessed 03.12.2018

[15] OpenCV Image Filtering,
https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=bilateralfilter#bilateralfilter,
accessed 03.12.2018

[16] OpenCV Image Filtering,
https://docs.opencv.org/3.1.0/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11ac52cf1,
accessed 03.12.2018

[17] OpenCV Smoothing Images,
https://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html, accessed 03.12.2018

[18] Canny Edge Detection,
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html,
accessed 03.12.2018

[19] A. C. Seymour, J. Dale, M. Hammill, P. N. Halpin, D.W. Johnston,
Automated detection and enumeration of marine wildlife using unmanned aircraft systems (UAS) and thermal imagery, 2017

[20] John Canny,
A computational approach to edge detection, 1986

[21] Martin Israel,
A UAV-based Roe Deer Fawn Detection System, 2011

[22] Even Arneberg Rognlien, Tien Quoc Tran,
Detecting Location of Free Range Sheep, 2018

[23] OpenCV Contours: Getting Started,
https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html,
accessed 05.12.2018

[24] OpenCV Contours Hierarchy,
https://docs.opencv.org/3.4/d9/d8b/tutorial_py_contours_hierarchy.html,
accessed 05.12.2018

[25] OpenCV Contour Features,
https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html,
accessed 05.12.2018

[26] David H. Douglas, Thomas E. Peucker,
https://utpjournals.press/doi/10.3138/FM57-6770-U75U-7727,
accessed 09.12.2018

[27] Open-Elevation API,
https://open-elevation.com/,
accessed 10.12.2018

[28] J. Jin, J. Li, G. Liao, X. Yu, L. C. C. Viray,
Methodology for Potatoes Defects Detection with Computer Vision, 2009

[29] OpenCV: Scene Reconstruction,
https://docs.opencv.org/3.4.3/d4/d18/tutorial_sfm_scene_reconstruction.html,
accessed 14.03.2019

[30] Marit Gjøstøl Ytterland, Tone Kathrine Ervik Winsnes,
Retrieval of sheep using UAVs, 2018

[31] OpenCV Image Thresholding,
https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html,
accessed 27.03.2018

[32] OpenCV: Basic Thresholding Operations,
https://docs.opencv.org/2.4.13.7/doc/tutorials/imgproc/threshold/threshold.html,
accessed 28.03.2019

[33] OpenCV: Miscellaneous Image Transformations,
https://docs.opencv.org/3.4.3/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57,
accessed 29.03.2019

[34] OpenCV: Image Filtering,
https://docs.opencv.org/3.0-beta/modules/imgproc/doc/filtering.html#pyrmeanshiftfiltering,
accessed 09.04.2019

[35] Zhang Zheng,
Document Image Restoration, 2005

[36] OpenCV: Median Blur,
https://docs.opencv.org/3.1.0/d4/d86/group__imgproc__filter.html#ga564869aa33e58769b4469101aac458f9,
accessed 02.05.2019

[37] OpenCV: Histogram Equalization,
https://docs.opencv.org/2.4.13.7/doc/tutorials/imgproc/histograms/histogram_equalization/histogram_equalization.html,
accessed 09.04.2019

[38] A. Ford, A. Roberts,
Colour Space Conversions, 1998

[39] Histogram Equalization Of RGB Images,
https://prateekvjoshi.com/2013/11/22/histogram-equalization-of-rgb-images/,
accessed 24.05.2019

[40] Open-Elevation API,
https://open-elevation.com/,
accessed 10.04.2019

[41] OpenCV: Bounding Rectangle,
https://docs.opencv.org/3.1.0/d3/dc0/group__imgproc__shape.html#ga3d476a3417130ae5154aea421ca7ead9,
accessed 10.04.2019

[42] Facts About Sheep,
https://www.livescience.com/52755-sheep-facts.html
accessed 22.05.2019

[43] OpenCV: Thresholding Operations Using inRange,
https://docs.opencv.org/3.4.3/da/d97/tutorial_threshold_inRange.html,
accessed 26.04.2019

[44] OpenCV: Changing Colorspaces,
https://docs.opencv.org/3.4.3/df/d9d/tutorial_py_colorspaces.html,
accessed 26.04.2019

[45] OpenCV: Operations On Arrays,
https://docs.opencv.org/3.4.3/d2/de8/group__core__array.html#ga48af0ab51e36436c5d04340e036ce981,
accessed 29.04.2019

[46] OpenCV: Dilation,
https://docs.opencv.org/3.0-beta/modules/imgproc/doc/filtering.html#dilate,
accessed 30.04.2019

[47] Named Colors,
`https://convertingcolors.com/named-colors.html`,
accessed 30.04.2019

[48] G. Finlayson, C. Fredembach,
Simple Shadow Removal, 2006

Marit G. Ytterland, Tone K. E. Winsnes

Retrieval of Sheep Using Unmanned Aerial Vehicles

NTNU
Kunnskap for en bedre verden