

Agnar Martin Bjørnstad
Torstein Emdal Otterlei

Automatic Detection of Safe Landing Areas for Vertical Take-Off and Landing Unmanned Aerial Vehicles

Master's thesis in Computer Science
Supervisor: Gabriel Kiss
Co-supervisor: Frank Lindseth
July 2022

Agnar Martin Bjørnstad
Torstein Emdal Otterlei

Automatic Detection of Safe Landing Areas for Vertical Take-Off and Landing Unmanned Aerial Vehicles



Master's thesis in Computer Science
Supervisor: Gabriel Kiss
Co-supervisor: Frank Lindseth
July 2022

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

Agnar Martin Bjørnstad, Torstein Emdal Otterlei

Automatic Detection of Safe Landing Areas for Vertical Take-Off and Landing Unmanned Aerial Vehicles

Master Thesis, Spring 2022

Artificial Intelligence Group
Department of Computer Science
Faculty of Information Technology and Electrical Engineering



Abstract

In recent years, drones have become household objects. Due to cheaper and more advanced electronics, drone technologies which once were exclusive to militaries and other large institutions have become commercially available for smaller companies and even private use. Today, drones are being used in many applications ranging from recreation and racing to maintenance inspections and transport. One of the companies providing transportation services using drones is the Norwegian company Aviant, which is the collaboration partner of this thesis.

When performing drone operations such as transport in Beyond Visual Line Of Sight (BVLOS) scenarios, an important security aspect is to have predetermined rally points along the flight path. These rally points represent safe landing areas that the drone can land on if any issues were to occur. Today, companies such as Aviant plot these landing areas manually using digital maps. In this thesis, we introduce and compare two deep learning-based systems that can automatically detect safe landing areas in the context of vertical take-off and landing unmanned aerial vehicles using an onboard, downward-facing camera.

Detecting safe landing areas for drones is a well explored field. However, limited research exists using deep learning-based approaches, and we therefore wish to explore this field in this thesis. In addition to the systems mentioned above, we also present a PCA-based color mapping technique capable of correcting colors in images and a pixel geolocator capable of translating image pixels into GPS coordinates. Furthermore, we present a labeled drone-video dataset, in addition to a synthetic dataset containing aerial images of people. We also introduce a method for using the safe landing area detection system to validate manually chosen rally points.

Even though the SLAD systems presented in this thesis are capable of successfully detecting good landing areas, they are not considered robust enough to operate fully autonomously. However, they can operate with minimal human oversight, and with some further work, possibly fully autonomously.

Sammendrag

De siste årene har droner blitt hverdagslige. På grunn av rimeligere og mer avansert elektronikk har droneteknologi, som en gang var forbeholdt forsvarssektoren og andre store institusjoner, blitt kommersielt tilgjengelig for mindre bedrifter og privat bruk. I dag benyttes droner innen mange områder, fra fornøyelse og kappløp til inspeksjoner og transport. En av bedriftene som tilbyr leveransetjenester ved bruk av drone er det norske selskapet Aviant, som er samarbeidspartneren for denne oppgaven.

Når en utfører droneoperasjoner som er utenfor visuell synslinje, er et viktig sikkerhetstiltak å ha forhåndsbestemte landingspunkter langs flyruten. Dersom noen problemer skulle oppstå, kan dermed dronen lande på disse landingspunktene. I dag bestemmer aktører slik som Aviant disse punktene manuelt ved bruk av digitale kart. I denne oppgaven introduserer vi to dyp lærings-baserte systemer som automatisk kan detektere trygge landingsområder for ubemannede droner. For å gjøre dette benytter vi et nedovervendt kamera som viser terrenget under dronen.

Å detektere landingsområder for droner er allerede et område som har blitt forsket mye på. Likevel er det lite av denne forskningen som benytter metoder basert på dyp læring. Vi ønsker derfor å utforske dette i denne oppgaven. I tillegg til systemene nevnt tidligere, presenterer vi også en PCA-basert metode for fargegjenoppretting som kan korrigere farger i bilder og en piksel-geolokator som kan oversette piksler fra bildene til GPS koordinater. Vi presenterer også et merket datasett som består av dronevideoer, i tillegg til et syntetisk datasett som inneholder bilder av mennesker i fugleperspektiv. Vi introduserer også en metode for å bruke systemet for deteksjon av landingspunkter til å validere de manuelt utvalgte landingspunktene.

Selv om systemene for å detektere trygge landingpunkter som presenteres i denne oppgaven er i stand til å finne gode landingspunkter, er de ikke pålitelige nok til å kunne operere helt autonomt. Likevel er de egnet til å operere med litt menneskelig tilsyn, og med mer arbeid og forskning kan de muligens operere helt autonomt.

Acknowledgements

We wish to thank our supervisor Gabriel Kiss, co-supervisor Frank Lindseth and the Aviant representative Torjus Bakkene for the support during this project.

Agnar Martin Bjørnstad, Torstein Emdal Otterlei
Trondheim, July 1, 2022

Contents

1	Introduction	1
1.1	Background and Motivation	1
1.2	Goals and Research Questions	2
1.3	Research Method	4
1.4	Contributions	4
1.5	Thesis Structure	5
2	Background	6
2.1	Aviant	6
2.1.1	Drone	6
2.1.2	Mission planning	9
2.1.3	Flights	11
2.1.4	Internal setup of drone	11
2.2	Object Detection	12
2.2.1	Metrics in object detection	13
2.2.2	YOLOv5	16
2.2.3	Swin Transformer	16
2.2.4	ConvNeXt	17
2.3	Image Segmentation	18
2.3.1	Metrics in Image Segmentation	18
2.3.2	Unet	20
2.4	Geolocating Pixels	21
2.4.1	Generating Camera Rays	21
2.4.2	Rotatating the Camera Rays	22
2.4.3	Translation and Raytracing	24
2.4.4	Geolocation to pixel	25
2.5	Splitting videos into train, validation and test sets	25
2.6	Principal Component Analysis	26
2.7	Structured Literature Review Protocol	28

2.7.1	Motivation	30
2.8	Previous Work	30
2.8.1	SLAD using Artificial Landmarks	31
2.8.2	SLAD using Natural Landmarks	31
2.8.3	Color restoration	34
3	Method	36
3.1	Drone Dataset	36
3.1.1	Data Collection	36
3.1.2	Data Labeling	40
3.1.3	Dataset Partitioning	42
3.2	Pretraining Datasets	43
3.2.1	Potsdam Dataset	43
3.2.2	Unreal Engine 4 Dataset	44
3.3	Color Restoration using Principal Component Analysis	47
3.3.1	Overall Idea	47
3.3.2	Change of Principal Components Basis	48
3.3.3	Generalize to Unseen Images	51
3.4	Converting Segmentation to Rally Points	53
3.5	Pixel Geolocator	54
3.5.1	Mapping from Pixels to Coordinates	54
3.5.2	Mapping from Coordinates to Pixels	55
3.6	Segmentation	57
3.6.1	Tested Architectures	57
3.6.2	Training Setup	58
3.6.3	Segmentation-based SLAD System	60
3.7	Rally Point Verification	61
3.8	Object Detection	62
3.8.1	Tested Architectures	63
3.8.2	Training Setup	63
3.8.3	Object Detection-Based SLAD System	64
4	Results	66
4.1	Color Restoration using Principal Component Analysis	66
4.2	Pixel Geolocator	68
4.3	Segmentation	70
4.4	Rally Point Verification	72
4.5	Object Detection	75

5	Discussion	77
5.1	Drone Dataset	77
5.1.1	Data Collection	77
5.1.2	Data Labeling	78
5.1.3	Data Partitioning	78
5.2	Pretraining Datasets	79
5.2.1	Potsdam Dataset	79
5.2.2	Unreal Engine 4 Dataset	79
5.2.3	Answering Research Question 1	80
5.3	Color Restoration using Principal Component Analysis	81
5.3.1	Discussion of Results	81
5.3.2	Answering Research Question 2	81
5.4	Converting Segmentation to Rally Points	82
5.5	Pixel Geolocator	82
5.5.1	Answering Research Question 3	83
5.6	Segmentation	84
5.6.1	Pretraining	84
5.6.2	Impact of Color Restoration	86
5.6.3	Answering Research Question 1,2 and 4	87
5.7	Rally Point Verification	88
5.7.1	Components	88
5.7.2	Answering Research Question 5	89
5.8	Object Detection	89
5.8.1	Pretraining	90
5.8.2	Impact of Color Restoration	91
5.8.3	Answering Research Question 1,2 and 4	92
6	Conclusion and Future Work	95
6.1	Conclusion	95
6.2	Future Work	96
	Bibliography	98
	Appendices	105
A	Literature Review Table	106
B	Reference images for PCA	108
C	Result images from color restoration	111
D	Masks from Segmentation Models	113
E	Results from Object Detection-based System	120
F	Results from Segmentation-based System	127
G	Rally Point Verification Examples	134

List of Figures

2.1	One of Aviants drones.	7
2.2	Mission from Sandmoen to Haltdalen	8
2.3	Ground Risk Model and Air Risk Model	10
2.4	Components in a drone.	12
2.5	Example of object detection application	13
2.6	IoU scores in object detection	13
2.7	IoU formula	14
2.8	Precision-Recall curve	15
2.9	Swin-T architecture	17
2.10	Types of image segmentation	19
2.11	TN, FN, FP and TP in segmentation	20
2.12	Unet architecture	21
2.13	Roll, pitch and yaw on plane	22
2.14	PCA of a multivariate Gaussian distribution	27
2.15	Orientations of principal components	28
3.1	Sample images from <i>NT02</i> and <i>NT04</i>	37
3.2	Setup for capturing video	38
3.3	Missions in dataset.	39
3.4	Overview of the labeling process of the collected video	39
3.5	Automatic labeling of trees in snow	42
3.6	Example image from the Potsdam dataset	44
3.7	Example image of Unreal Engine dataset	46
3.8	Comparison of different axis in the PCA color restoration	49
3.9	PCA of image colors	51
3.10	Scatter plots of PCA color restoration distributions	52
3.11	Converting segmentation mask to specific rally points	53
3.12	Comparison of points from pixel geolocator	56
3.13	3D view of drone with estimated and actual locations	56

3.14	Single class prediction to multi class	60
3.15	SLAD pipeline for segmentation	61
3.16	SLAD pipeline for object detection	65
4.1	Color Mapper Results Example	67
4.2	Color Mapper Reference Image Example	67
4.3	An overview of the accuracy measuring of the pixel geolocator	69
4.4	Prediction on image of person by the segmentation-based SLAD system	72
4.5	Output examples of the rally point verification system	74
4.6	Prediction on image of person by the object detection-based SLAD system	76
1	Reference images used for PCA color restoration	110
3	Results from color restoration	112
4	Results from color restoration	113

List of Tables

2.1	Technical specifications of the Foxtech drone	7
2.2	Classification of risks	11
2.3	Search term groups for literature review	29
2.4	Inclusion and quality criteria for literature review	30
3.1	Flights in the dataset	38
3.2	Categories used when manually labelling images	43
3.3	Occurrences of obstacles in the collected dataset	43
3.4	Occurrences of obstacles in the Potsdam dataset.	44
3.5	Occurrences of obstacles in the Unreal Engine 4 dataset.	46
3.6	Converting single class semantic segmentation to multiclass	59
4.1	Accuracy of pixel geolocator	68
4.2	Ids for segmentation models	70
4.3	Results from segmentation	71
4.4	Results from the segmentation-based SLAD system	71
4.5	Recall of rally point verification	73
4.6	Ids for object detection models	75
4.7	Results from the object detection-based SLAD system	75
4.8	Overview of sizes of missed obstacles by SLAD systems	76
A	Literature Review	106
B	Search engines used in literature review	107

Acronyms

AP Average Precision

CNN Convolutional Neural Network

CVAT Computer Vision Annotation Tool

GPS Global Positioning System

BVLOS Beyond Visual Line Of Sight

IoU Intersection over Union

mAP Mean Average Precision

PCA Principal Component Analysis

SIFT Scale Invariant Feature Transform

SLAD Safe Landing Area Detection

SLAM Simultaneous Localization And Mapping

UAV Unmanned Aerial Vehicle

UTM Universal Transverse Mercator

VPN Virtual Private Network

VTOL Vertical Take-Off and Landing

QGC QGroundControl

Chapter 1

Introduction

1.1 Background and Motivation

In recent years, drones have become household objects. Due to cheaper and more advanced electronics, drone-technologies which once were exclusive to militaries and other large institutions have become commercially available for smaller companies and even private use. Today, drones are being used in many applications ranging from recreation and racing to maintenance inspections and transportation.

A challenge that arises when flying drones BVLOS, such as in long-range transportation, is to maintain safety. In these cases, drones often have to fly over inhabited areas to reach their destinations. A crash in these areas could cause damage to both property and people. Therefore, in order to maintain high safety standards, reliable technical systems and procedures are required. Governments impose strict legislation regarding drone transportation, which is an ongoing process in Norway and other countries. Actors such as Aviant [Aviant, 2022] use considerable amounts of resources to ensure that flight paths go over low-risk areas. An important part of the safety measures by Aviant is to have pre-programmed intermediate safe landing areas along the flight path, called rally points. In case of emergency, the drone may attempt to land on the nearest rally points. Evidently, their real-time state is not considered when they are chosen. An obstacle such as a car or a tree that is not visible on satellite imagery could therefore obstruct the landing zone.

In order to monitor the drone, Aviant uses an on-board, downward-facing camera that streams video to an operator. The aim of this thesis is to use this camera

to attempt to automatically propose new rally points. In addition, we will try to verify the manually determined rally points to ensure their quality. In this way, the quality of a landing area can be determined without using additional sensors, which would incur extra cost and weight. The survey [Shah Alam and Oluoch, 2021] provides a suitable framework for classifying drones and associated landing zone detection techniques. According to the terminology used in the survey, the problem we choose to explore in this thesis concerns using monocular vision techniques to find outdoor unknown static landing zones for a Vertical Take-Off and Landing (VTOL) Unmanned Aerial Vehicle (UAV).

1.2 Goals and Research Questions

A significant amount of research has been conducted in the field of Safe Landing Area Detection (SLAD) for drones, as will be explained in Section 2.8. However, there is limited research exploring SLAD using deep learning-based computer vision techniques with a monocular, downward-facing camera feed. In this thesis, we are therefore going to further explore this area, as stated below.

Goal *Explore the use of deep learning computer vision techniques in the context of safe landing area detection for VTOL UAVs with monocular aerial vision.*

When capturing video from the drones, we observed that the onboard camera on one of the drones produced videos with distorted colors. In addition, the captured videos contained few instances of some classes such as people, vehicles, buildings and power lines. When exploring the use of deep learning computer vision techniques in the context of safe landing area detection, these issues need to be addressed. We therefore introduce five components, some of which address the color and class imbalance issues, while the rest constitutes the components required to create a safe landing area detection system. The components are: (1) a synthetic dataset that can provide more examples of situations that rarely occur in the collected images, (2) a method capable of restoring the images with distorted colors, (3) a system that can geolocate pixels in the images from the video stream, (4) a system capable of detecting safe landing areas from images and (5) a system that can verify predetermined landing areas along the flight path. We then associate a research question with each of these components.

Research question 1 *Can synthetic data be beneficial in generating rare situations when trying to train a deep learning-based system for safe landing area detection?*

In this thesis, we have collected videos from Aviant’s drones, and used them to train deep learning-based systems. However, there are certain features that would be valuable to detect, but rarely occur in the videos. One of the most important of these rare features is people. It is therefore of scientific value to explore whether pretraining the systems on a synthetic dataset that contains many people could improve their ability to detect people on videos from the drones.

Research question 2 *How can images with distorted colors be restored to their original colors, and will this improve the effect of pretraining the deep learning models?*

One of the drones that captured videos had issues with the colors. There was a red hue on almost all of the images it captured. This might pose problems for deep learning networks that are initialized with weights from *regular colored* datasets, like *Imagenet* [Russakovsky et al., 2015]. Even though this was an issue on only one of the drones, videos from this drone constitute the vast majority of the collected videos. Hence, it is crucial to find out if this has a negative impact on the performance of the deep learning models, and if it is possible to restore the colors to fix the issue.

Research question 3 *How and with what accuracy can detected landing areas be converted into Global Positioning System (GPS) coordinates?*

A central part of the safe landing area detection system is to be able to convert the detected landing zones from the aerial images into GPS coordinates. In addition, we want to be able to determine where the predetermined, hand-picked rally points are located in the aerial images. In this way, the drone’s flight controller can locate the proposed safe landing areas, and the predetermined safe landing points can be visually verified.

Research question 4 *Which deep learning-based computer vision technique provides the best results in determining safe landing areas?*

The field of computer vision includes multiple sub-domains. Two of these are image segmentation and object detection. The properties of these methods, which we will explain in Section 2.3 and 2.2, make them particularly suitable for detecting safe landing areas. We will therefore explore two architectures for detecting safe landing areas based on each of these two methods.

Research question 5 *How effectively can a deep learning-based system for safe landing area detection verify predetermined landing areas?*

Aviant uses hand-picked rally points as emergency landing areas, as we will explain in Section 2.1.2. These are plotted using digital maps, and do not consider the current state of the landing area. It would therefore be of operational and scientific value to explore whether the safe landing area detection system is capable of verifying these predetermined rally points as the drone is flying above them. This research question builds upon Research question 3 and Research question 4, as both of these are necessary in order to get an effective mapping from GPS coordinates to a location in the image and evaluate the predetermined rally point, respectively.

1.3 Research Method

Because of the technical nature of the research goal, the research method will largely consist of design and experimentation. We will create a series of subsystems, that when combined creates a system that performs the desired action of detecting safe landing areas. Because we want to explore results from different system architectures when using different deep learning models and image augmentations, many combinations need to be tested and compared. To do this, we will first test the performance of each subsystem using appropriate metrics depending on their purpose. After this, we will test the systems as a whole using custom metrics that reveal details of their performance and behavior, and do this for each combination.

1.4 Contributions

In the work towards the research goal, a selection of contributions have been produced. These are listed below. In addition to these tangible contributions, we have also contributed with scientific insight into the architecture and performance of a deep learning-based system for safe landing area determination.

1. *A PCA-based color mapping technique capable of correcting colors in images*
2. *A pixel geolocator capable of translating image-pixels into GPS coordinates and vice-versa*
3. *A labeled drone-video dataset consisting of 11733 images*
4. *A synthetic dataset containing orthophotos with models of people*

5. *A segmentation-based system capable of detecting safe landing areas from downward-facing drone videos*
6. *An object detection-based system capable of detecting safe landing areas from downward-facing drone videos*

1.5 Thesis Structure

The thesis starts with a Background chapter. This chapter contains both general background knowledge and technical background theory related to the research project. It also includes a Structured Literature Review of previous work. In the next chapter, Method, we will go through the technical details of our approaches. This section uses the presented background theory to explain each of the submodules in the project. At the end of the method chapter, these submodules are combined into two complete SLAD systems and one system for Rally Point Verification. In the Results chapter, we will present the results acquired from these systems. Then, we will discuss our approach in light of the results in the Discussion chapter. Eventually, we will end the thesis with a chapter containing Conclusion and Future Work.

Chapter 2

Background

In this section we will go through some of the background theory that is required in order to explore our research goal. This includes an overview of how Aviant operates, in addition to some of the background theory about segmentation and object detection. Eventually, we will elaborate on the theory behind our pixel geolocator, present our literature review and go through previous work in the field.

2.1 Aviant

2.1.1 Drone

Aviant transports biological samples between destinations that are up to 120km apart. They use VTOL drones to transport these samples. One of these drones is depicted in Figure 2.1. The drones have two ways to fly: fixedwing and multirotor mode. In fixedwing mode, the drone flies like an airplane. It has wings and a motor that is attached to the back, which pushes it forward. This is a very efficient mode of flight. However, if the drone were to land in this mode, it would need wheels and a small runway. This is an issue, as it imposes great restrictions on the areas in which it can perform a safe landing. In order to solve this problem, the multirotor mode is used. In this mode, the drone uses the four vertical propellers to fly like a quadcopter. This means that it can take-off and land vertically, effectively reducing the required landing area. However, this mode of flight is much less energy efficient and the velocity is very limited, so this flight mode is only used for take-off, landing and hovering.

The technical specifications of the drones used by Aviant are listed in Table 2.1. By looking at the maximum cruising speed, the flight time and the max-

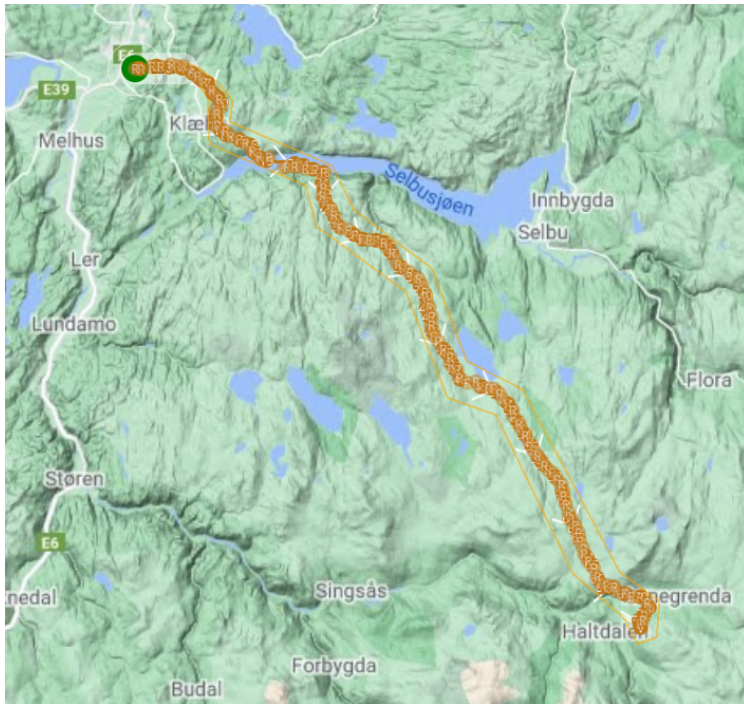
imum payload capacity, it is clear that the drones are suitable for long-range transportation of lightweight cargo. Aviant uses a cruising speed of 23m/s in fixedwing mode.



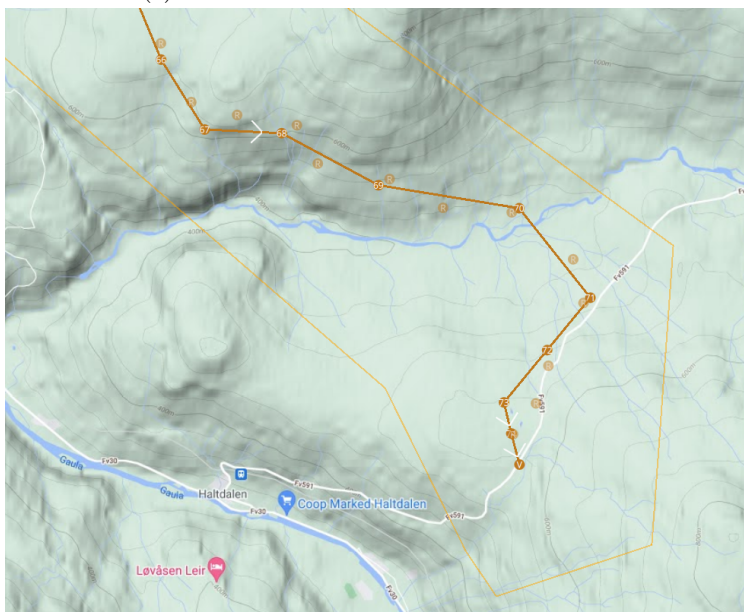
Figure 2.1: One of Aviant's drones. (Source: Tom Erik Holthe, Aviant)

Specification	Value
Max cruise speed	110km/h
Flight time	2.5h
Max payload capacity	3kg
Wingspan	250cm
Length	140cm

Table 2.1: Technical specifications of the Foxtech drone that Aviant uses.



(a) A mission from Sandmoen to Haltdalen.



(b) A closer look at the end of the mission from Figure 2.2a

Figure 2.2: The mission from Sandmoen to Haltdalen. (Maps from [Google Maps, 2005])

2.1.2 Mission planning

During flight, the drone is following a preprogrammed path. Such a path is part of a mission. A mission is composed of several components in order to safely and effectively get the drone to its destination. One of Aviant's missions is depicted in Figure 2.2a. This mission goes from Sandmoen to Haltdalen and is about 70km long. Figure 2.2b shows a close-up view of the end of the mission. Here we can see its individual components. The most conspicuous component is the flight path, the thick yellow line with points. These points are called waypoints, and each of them has a latitude, longitude and a height above sea level. When flying a mission, the drone is pursuing a path that resembles a straight line from its position to the next waypoint. The actual path it plans to fly is a bit more complex. However, since this is not relevant in our case, it will not be elaborated on.

When the pursued waypoint is within a certain range of the drone, it proceeds to pursue the next waypoint in the mission, and so on, until it reaches a landing point. Here it goes from fixedwing mode to multirotor mode and lands vertically. Along the flight path, there are rally points. These are points where the drone can make an emergency landing if anything unexpected happens during flight. The rally points have a light orange color and are marked with an *R* in Figure 2.2b. Furthermore, the entire flight path and the rally points are encapsulated in a geofence, the yellow polygon. If the drone hits the geofence during flight, it lands on the nearest rally point or lands straight down, depending on the settings.

The rules and regulations that Aviant follows are specified by the European Union Aviation Safety Agency [EASA, 2021]. The airspace that needs to be considered in missions is visualized in Figure 2.3. The Flight Geography is the area in which the drone is allowed to fly, this is where the waypoints are located. The Contingency Volume is a buffer volume around the Flight Geography. The drone is allowed to be in the Contingency Volume, but needs to initiate actions to get the drone back into the Flight Geography zone. The Operational Volume is the union of the Flight Geography and the Contingency Volume. The rally points are located on ground level in this volume. The border between the Flight Geography and the Contingency Volume is where the geofence is located. Outside the Contingency Volume there is the Ground Risk Buffer for ground risks, and the Air Risk Buffer for air risks. These are places where the drone under no circumstances, even in emergencies, should be located. As the drone would then be too close to risks. Table 2.2 classifies the different risks into Hard/Soft Ground Risks and Air Risks. Hard risks must be avoided, while soft risks should be avoided.

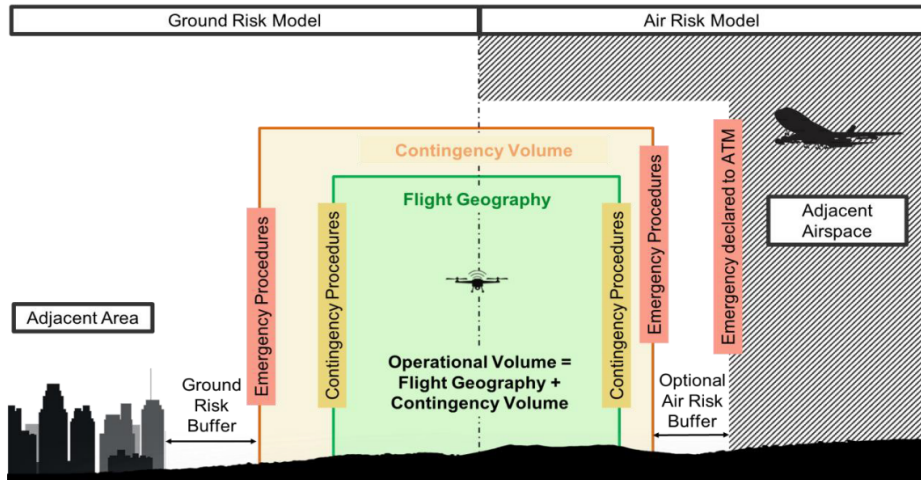


Figure 2.3: Ground Risk Model and Air Risk Model. (Source: [EASA, 2021])

Rally points are considered safe points to land during a flight. Hence, according to Aviant’s internal guidelines, a rally point must satisfy the requirements below:

1. Free of obstructions to the best of our knowledge
2. Low or no population density
3. Close enough to the flight path such that the drone can reach it
4. Located inside the operational volume

Furthermore, a good rally point is characterized by the two following traits.

1. There is a road nearby
2. The terrain is flat

The first trait reduces the work needed to retrieve the drone after an emergency landing on the rally point. The second trait mitigates the risk of damaging the drone during such a landing. Rally points have been used in almost all events where an emergency has occurred, which is why it is imperative to have rally points evenly distributed along the flight path. The alternative is for the drone to land straight down from its current position. This might be necessary if the drone for some reason cannot reach the closest rally point, for instance, if it is too low on battery.

Ground Risks		Air Risks	
Hard	Soft	Hard	Soft
Gathering areas (schools, churches, malls, etc.)	Busy roads	Urban airspace	Airstrips
Populated areas	Homes	Airspace above 500ft	Helipads
Prisons			
Nature preserves with flight prohibi- tion	Nature preserves without flight pro- hibition	Within 5km of air- ports and heliports	

Table 2.2: Classification of risks.

2.1.3 Flights

Every time a drone is flying, it is called a flight. Each flight has a flight crew and a mission. The flight crew carries out the preflight checks and monitors the drone during flight. The pilot has a touchpad with QGroundControl (QGC) [QGC, 2015] installed. This is used to send high level instructions to the drone, like *take-off*, *land at the nearest rally point*, *fly back to take-off position* and *fly in a circle*. The latter command can be used if something temporarily prevents the drone from continuing its flight, like incoming traffic ahead. The drone also has a downward-facing camera, that streams video in real-time to QGC. This is, amongst other things, used to check that the landing area is clear of obstructions.

GPS and a cellular connection such as 5G are critical components in a flight. 5G is used for communication between QGC and the drone, while GPS is used by the drone for navigation. If any of these components fail, the outcome may be critical. In the event of a sustained loss of GPS, the drone lands straight down, regardless of what is underneath. If it experiences a sustained loss of cellular connection, it lands at the nearest rally point.

2.1.4 Internal setup of drone

Figure 2.4 shows the components in the drone. The Flight Controller is a real-time system that controls the drone, this is where the autopilot is running. The companion computer is an onboard computer that, amongst other things, communicates with the Flight Controller and streams video from the camera. The MAVLink Router [MAVLink, 2017] on the companion computer is the main source of communication between the commander and the drone. MAVLink

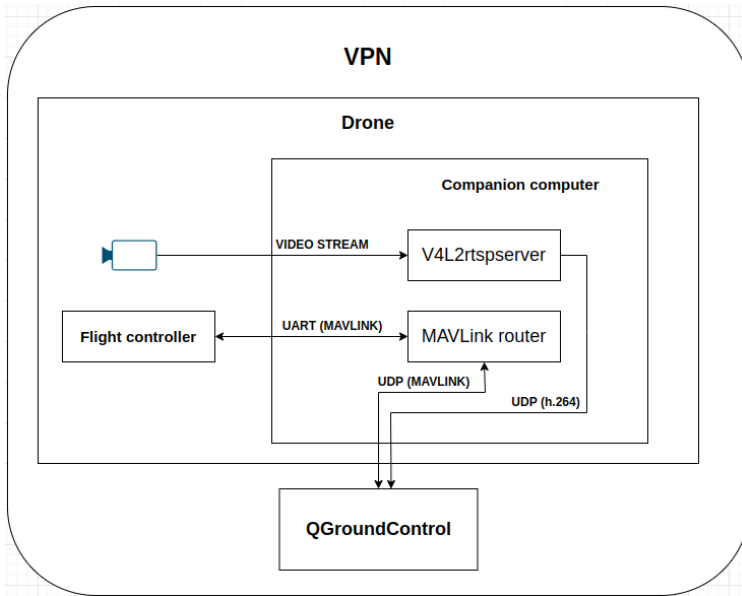


Figure 2.4: Components in a drone.

[Koubâa et al., 2019] is a protocol that is used for communication, between the Flight Controller and the MAVLink Router. It goes over a serial port cable, and between the MAVLink Router and QGC it goes over UDP [UDP, 1980]. The video is streamed from the camera to a v4l2rtsp server [v4l2rtspserver, 2015] and streamed by UDP in H.264 format [Hsiao et al., 2011] to QGC.s

2.2 Object Detection

Object detection is the task of locating and classifying objects in an image. The location of an object is captured by a bounding box. A bounding box is a rectangular area that covers the entire object of interest. The most common scenario is to have the bounding boxes parallel to the edges of the image. If the bounding boxes can take on any orientation, it is called multi-oriented object detection. Figure 2.5 shows a typical application of object detection algorithms: detecting cars. The goal of the algorithm is to predict bounding boxes that match the ground truth bounding boxes as closely as possible, in addition to determining the type of object in the bounding box. Figure 2.6 shows how the predictions of an object detection algorithm need to align with the ground truths. Notice that some of the objects in Figure 2.5 are partially occluded. This is one of the

challenges that object detection models need to handle.



Figure 2.5: Example of object detection application: detecting cars on roads. (Source: [Hackem, 2020])

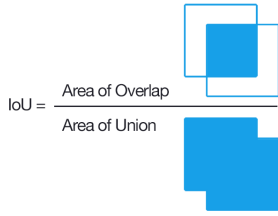


Figure 2.6: IoU scores of object detection. The better the prediction, the higher is the IoU. (Source: [Adrian Rosebrock, 2017])

2.2.1 Metrics in object detection

In order to quantitatively measure the performance of an object detection model, metrics that compare the predictions to the ground truth are needed. One of the most fundamental metrics in object detection is the Intersection over Union (IoU). The IoU measures the degree of overlap between two bounding boxes. Figure 2.7a shows how the IoU is calculated. The intersecting area is divided by the union area of the two bounding boxes, hence the name. The IoU ranges from 0 to 1,

where the former indicates no overlap and the latter indicates perfect overlap. When measuring the quality of a predicted bounding box, the formula in Figure 2.7b is used to compare it to the ground truth. As we will see, the IoU is the foundation for other metrics used in object detection. It is important to note that the IoU is not only used as a quantitative measure of a bounding box’s quality relative to the ground truth. It is also used by object detection algorithms to filter away potentially bad predictions in favor of better ones.



(a)

$$\text{IoU} = \frac{\text{area}(gt \cap pd)}{\text{area}(gt \cup pd)}$$

(b)

Figure 2.7: Formula for IoU. (Image a source: [Adrian Rosebrock, 2016])

Bounding boxes are categorized into one of three different categories, as listed below. All four possible combinations of true/false and positive/negative are listed for clarity.

- True positive (TP): number of predictions with $\text{IoU} \geq \alpha$ with a ground truth bounding box
- False positive (FP): number of predictions with $\text{IoU} < \alpha$ for all ground truth bounding boxes
- False negative (FN): number of ground truths that no prediction has $\text{IoU} \geq \alpha$
- True negative (TN): the background region, correctly not detected by the model. This metric is not used.

A prediction is either true positive or false positive as it may or may not have a sufficient IoU with a ground truth bounding box. Likewise, a ground truth bounding box is classified as false negative if no predicted bounding box has a sufficient IoU.

TP , FP and FN are the numbers needed to calculate precision and recall for a model. Equation 2.1 shows how to calculate the precision. The precision measures the portion of predictions that actually are relevant. Ie. a low precision

means that the model produces many predictions that are not correct, relative to the number of predictions that are correct. The formula for recall is stated in Equation 2.2. The recall is the degree to which the model manages to detect all ground truth bounding boxes. I.e. a low recall means that only a small portion of the ground truth bounding boxes are detected.

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{\text{all detections}} \quad (2.1)$$

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{\text{all ground truths}} \quad (2.2)$$

The output from an object detection model often contains many predictions with confidence for each prediction ranging from 0 to 1, increasing with model certainty. The question then becomes what to consider to be a positive classification. Lowering the confidence threshold will result in more positive predictions that the model has less confidence in. From the equations above, it is clear that the impact of this is an increased recall, as some less confident predictions probably are correct, at the cost of decreased precision. The proportion of true positive predictions is probably lower for less confident predictions. In the same way, increasing the confidence threshold has a tendency to lower the recall and increase the precision. This trade-off between precision and recall can be visualized as a precision-recall curve. Figure 2.8 shows a precision-recall curve.

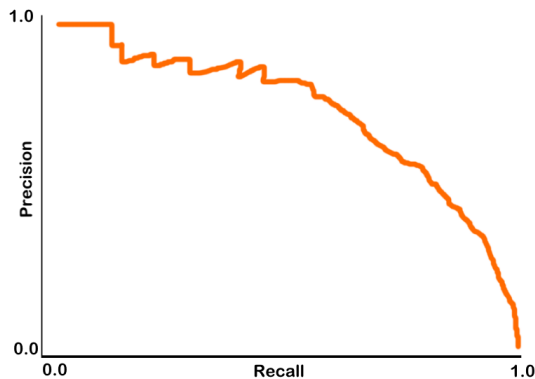


Figure 2.8: Example of a precision-recall curve. (Created by the authors)

To measure how precise an object detection algorithm is, the Average Precision (AP) is calculated. The AP is the area under the precision-recall curve. It is commonly evaluated at IoU thresholds 0.5 or 0.75. The Mean Average Precision (mAP) is the mean of all the class AP.

2.2.2 YOLOv5

YOLO [Redmon et al., 2015] is an acronym for *you only look once*, and consists of a unified model that simultaneously predicts multiple bounding boxes and class probabilities for those boxes. The model divides the image into a grid, and for each grid cell, it predicts a certain amount of bounding boxes, confidence for those boxes and class probabilities. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Since the model only consists of a single neural network, it is able to run very fast. In certain cases, YOLO has proven to be able to run at more than a hundred fps. In addition, YOLO reasons globally about the image when making predictions. This makes the system able to encode contextual information about classes as well as their appearance. YOLO has become one of the primary object detection techniques, and it has been incrementally improved upon since it was first introduced. YOLOv2 [Redmon and Farhadi, 2016] improved recall and localization by introducing batch normalization and anchor boxes. YOLOv3 [Redmon and Farhadi, 2018] uses a more complex model backbone called DarkNet-53, a 106 layer neural network complete with residual blocks and upsampling networks to improve the detection of smaller objects. YOLOv4 [Bochkovskiy et al., 2020] comes with a combination of modern improvements to the architecture. YOLOv5 [YOLOv5, 2020] does not have a paper, but is an implementation of YOLO that includes various improvements by the open-source community.

2.2.3 Swin Transformer

Transformers were initially presented by researchers at Google in the paper *Attention Is All You Need* [Vaswani et al., 2017], and were created as an alternative for Recurrent Neural Networks to handle Natural Language Processing tasks such as translating and summarizing text. However, in the paper *An Image Is Worth 16x16 Words: Transformers For Image Recognition* [Dosovitskiy et al., 2020], another team of researchers from Google proposed an image recognition technique using transformers. The unique capability of transformers is their ability to incorporate self-attention. Self-attention is a mechanism that allows the inputs of the model to interact with each other and find out which of them that should be emphasized when making a prediction.

Swin Transformer [Liu et al., 2021] from Microsoft Research Asia is a recent proposal that presents a transformer-based backbone architecture for vision tasks. The goal of a backbone architecture is to provide a proven method of effective feature extraction and has primarily consisted of Convolutional Neural Network (CNN) architectures in the past. Instead of dividing each input image into patches, it also divides them into windows. The swin transformer block then calculates the attention between the patches in each window and ignores the rest of the patches. The windows then slide across the image embedding to calculate the attention over a new group of patches. This technique has a lot in common with how CNNs work and is an attempt to circumvent the quadratic calculations required for conventional self-attention. An overview of the swin transformer architecture is shown in Figure 2.9.

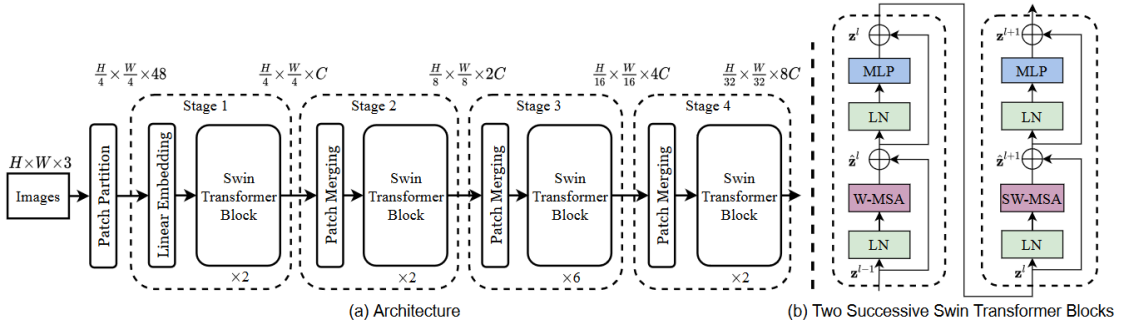


Figure 2.9: An overview of the Swin-Transformer architecture. (Source: [Liu et al., 2021])

2.2.4 ConvNeXt

ConvNeXt was introduced by researchers at Facebook in the paper *A ConvNet for the 2020s* [Liu et al., 2022]. It was developed in an attempt to create a modern CNN architecture capable of competing with transformer-based methods such as Swin-Transformer. The architecture of ConvNeXt is based on a standard ResNet architecture [He et al., 2015], but is gradually modernized to include the favorable features of a vision transformer [Dosovitskiy et al., 2020]. These improvements can be separated into changes on macro and micro level:

Macro level:

1. Changing stage compute ratio
2. Changing stem to “Patchify”

3. Introduce depthwise convolution similar to ResNeXt [Xie et al., 2016]
4. Using inverted bottlenecks
5. Using 7×7 depthwise conv in each block

Micro level:

1. Replacing ReLU with GELU
2. Fewer activation functions
3. Fewer normalization layers
4. Substituting batch normalization with layer normalization in each residual block
5. Separate downsampling layers

All these improvements are incorporated into the final ConvNeXt model.

2.3 Image Segmentation

Image segmentation is the process of partitioning an image into non-overlapping segments on the pixel level. There are different ways to segment an image. Figure 2.10 shows the difference between semantic, instance and panoptic segmentation. In semantic segmentation, each pixel is assigned a value corresponding to the class of the object that it is a part of. In instance segmentation, each semantic class is categorized along with the particular instance of that class. This means that each pixel is labeled according to what type of object it is part of in addition to which instance of that object class in the image it constitutes. Panoptic segmentation is a combination of both semantic segmentation and instance segmentation, as each pixel in the image belongs to both a class and an instance.

2.3.1 Metrics in Image Segmentation

IoU is also used in the segmentation context. However, when it comes to how True Negatives, False Negatives, False Positives and True Positives are decided, it is slightly different from object detection. They are listed below, and shown in Figure 2.11. This is then further used to calculate the AP and mAP.

- True positive (TP): The pixel is correctly categorized as part of its category
- False positive (FP): The pixel is incorrectly categorized as part of a category

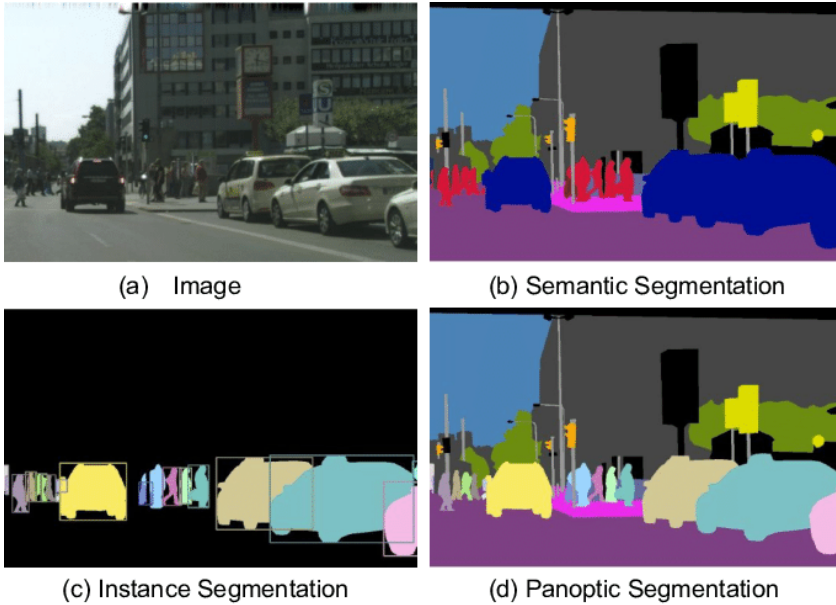


Figure 2.10: Types of image segmentation. (Source: [Kirillov et al., 2019])

- False negative (FN): The pixel is incorrectly categorized as not part of its category
- True negative (TN): The pixel is correctly categorized as not part of a category

$$F1 = \frac{2TP}{2TP + FP + FN} \quad (2.3)$$

Another metric is the F1 score, also called the Dice coefficient. The F1 score is calculated by taking $2 \times \text{Area of Overlap}$ divided by the total number of pixels in each image, see Equation 2.3. F1 score and IoU give similarity values between 0 and 1, and are always positively correlated. To a large degree, the two metrics serve the same purpose. However, when taking the average score over a set of inferences, IoU tends to penalize single instances of bad classification more than F1 score. If a model is providing overall good results, but with some bad outliers, IoU will put more emphasis on the outliers, and F1 and IoU might therefore end up favoring different models.

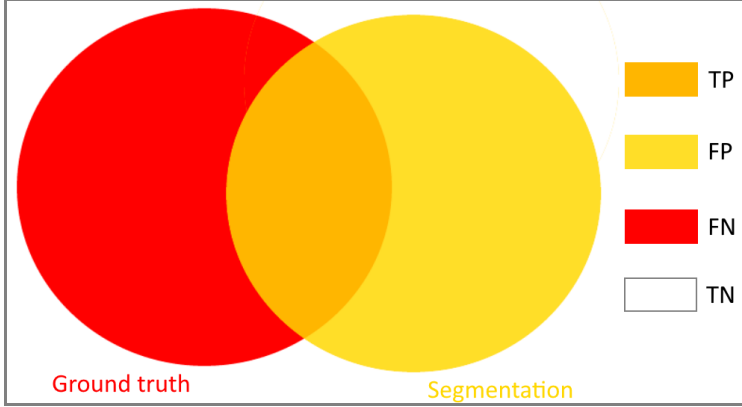


Figure 2.11: Illustration of True Negative, False Negative, False Positive and True Positive in segmentation (Created by the authors).

$$CA = \frac{TP + TN}{TP + FP + FN + TN} \quad (2.4)$$

Classification accuracy is the proportion of the number of correctly classified pixels to the total number of pixels in the segmented image. The formula for classification accuracy is shown in Equation 2.4.

2.3.2 Unet

In this thesis, we will test three segmentation models: Unet, Swin-Transformer and ConvNeXt.

Unet was originally developed by researchers from the University of Freiburg to perform biomedical image segmentation. Unet builds upon the architecture of the Fully Convolutional Neural Network [Long et al., 2014]. It includes a contracting path, an encoder, and an expansive path, a decoder. The encoder acts as a feature extractor and learns abstract representations of the input images using a series of encoder blocks with convolutional layers. The decoder part of the network generates the actual segmentation mask, and uses deconvolution to upsample the compressed image representation. Between the encoder and decoder, there are skip-connections that provide the decoder with the representations learned at each corresponding step of the encoder block. This helps the network get a more precise localization of masks. An overview of the Unet architecture is shown in

Figure 2.12. Notice that the output segmentation map is not the same resolution as the input mask in this illustration. This is because the illustration shows an example using unpadded convolutions. If padded convolutions were used instead, the input and output would be the same size.

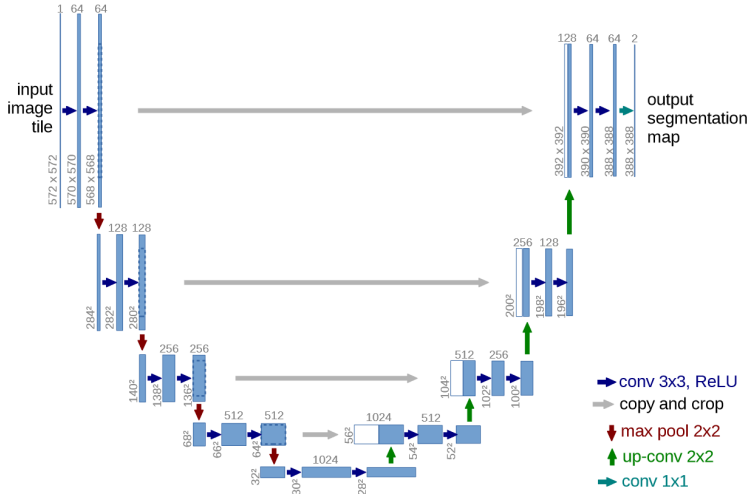


Figure 2.12: An overview of the U-Net architecture. (Source: [Ronneberger et al., 2015])

2.4 Geolocating Pixels

To be able to pinpoint the real-world location of pixels in an image from the video stream is a crucial component in a system for determining safe landing areas, as potential landing areas have to be conveyed as pairs of latitude and longitude to the flight controller. Hence, an elementary understanding of rotations in 3D space and raytracing is needed to comprehend how a pixel in an image might be mapped to a position on earth.

2.4.1 Generating Camera Rays

A camera captures the light that passes through its lens. The light is going *from* the outside world and *into* the camera. However, when we are to geolocate a pixel

on the resulting image, it is easier to imagine that the ray goes the other way: *from* the camera, through the lens and *down to the ground*. When we geolocate a pixel, we first have to find the direction of the ray relative to the camera. For this purpose, let us define a 3D-coordinate system where the camera is placed in origo, and rotated such that it is facing down the z -axis while the upper left corner of the image has positive x and y coordinates. Consider the plane that is parallel to the xy -plane and goes through $(0, 0, -1)$. This plane is defined by Equation 2.5.

$$z = -1 \quad (2.5)$$

If the camera were to take an image of size $width \times height$ of this plane. The area that we would see on the image would depend on the field of view in the direction of the $height$ and $width$ of the image, FOV_h and FOV_w , respectively. The coordinates of a pixel in row i and column j in the image are defined by Equation 2.6.

$$c_{ij} = \begin{bmatrix} FOV_h(\frac{1}{2} - \frac{i}{height}) \\ FOV_w(\frac{1}{2} - \frac{j}{width}) \\ -1 \end{bmatrix} \quad (2.6)$$

2.4.2 Rotatating the Camera Rays

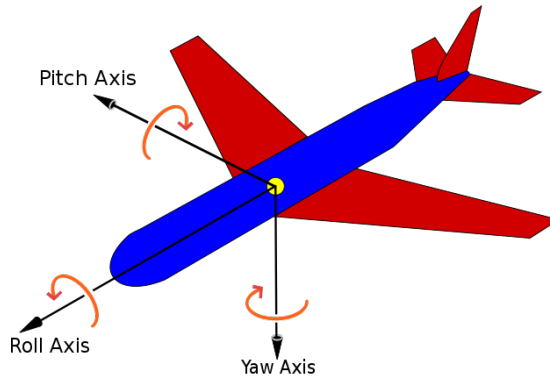


Figure 2.13: Illustration of roll, pitch and yaw on a plane. (Source: [Auawise, 2010])

All rotations on a body in 3D space can be represented by rotations along three orthogonal axes. In aviation, these rotations are called roll, pitch and yaw. These rotations are displayed in Figure 2.13. Defined by a right-hand coordinate system centered in the drone with the first axis forward on the drone, the second axis parallel to the wings oriented outwards from the right-wing of the drone, and the third axis pointing down. The rotation along each axis is called roll, pitch and yaw, respectively. Positive orientation is in the clockwise direction for all axis.

Roll, pitch and yaw are intrinsic rotations, ie. the axes of rotation are rotated along with the plane of rotation, in contrast to extrinsic rotations, where rotations happen along the originally fixed axes. The consequence of this is that the order in which the rotations are needs to be taken into account, as the other axes of rotation are dependent on the preceding rotations.

Rotation along each axis in Figure 2.13 can be described by a rotation matrix.

$$R_x(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma) & \cos(\gamma) \end{bmatrix} \quad (2.7)$$

$$R_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \quad (2.8)$$

$$R_z(\alpha) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.9)$$

Equation 2.7, 2.8 and 2.9 represents rotation along the first, second and third axis, respectively. Hence, γ , β and α corresponds to roll, pitch and yaw. The combined rotation matrix is computed in Equation 2.10.

$$R(\gamma, \beta, \alpha) = R_z(\alpha)R_y(\beta)R_x(\gamma) = \quad (2.10)$$

$$\begin{bmatrix} \cos(\alpha)\cos(\beta) & \cos(\alpha)\sin(\beta)\sin(\gamma) - \sin(\alpha)\cos(\gamma) & \cos(\alpha)\sin(\beta)\cos(\gamma) + \sin(\alpha)\sin(\gamma) \\ \sin(\alpha)\cos(\beta) & \sin(\alpha)\sin(\beta)\sin(\gamma) + \cos(\alpha)\cos(\gamma) & \sin(\alpha)\sin(\beta)\cos(\gamma) - \cos(\alpha)\sin(\gamma) \\ -\sin(\beta) & \cos(\beta)\sin(\gamma) & \cos(\beta)\cos(\gamma) \end{bmatrix}$$

To rotate the ray from the camera in Equation 2.6, it is left multiplied with

the rotation matrix in Equation 2.10. By dividing by the length of the camera ray, we end up with a rotated camera ray scaled to length 1. However, we first need to align the roll axis of the drone ray with the first axis of the world reference frame and the pitch and yaw axes with the negative second and third axis respectively. The orientation of rotations for pitch and yaw are thus flipped, and they have to be negated. Furthermore, the offset of the yaw then becomes a quarter of a rotation. This is shown in Equation 2.11.

$$r_{ij}(\gamma, \beta, \alpha) = \frac{R(\gamma, -\beta, \frac{\pi}{2} - \alpha)c_{ij}}{\|c_{ij}\|} \quad (2.11)$$

2.4.3 Translation and Raytracing

Up until now, the position of a pixel in an image has been transformed into a camera ray, scaled and then rotated according to the orientation of the drone. Thus, the next step is to translate the drone to its position in the Universal Transverse Mercator (UTM) projection [Snyder, 1987] and trace the camera ray down to the terrain.

$$p_0 = \begin{bmatrix} x_0 \\ y_0 \\ h_0 \end{bmatrix} \quad (2.12)$$

Let Equation 2.12 define the position, p_0 , of the drone in the UTM projection. The drone has x_0 , y_0 and h_0 as x , y and z coordinates, respectively. h_0 is the drones height above sea level.

$$p_t = p_0 + t * r_{ij}(\gamma, \beta, \alpha), 0 \leq t \quad (2.13)$$

Furthermore, by combining this equation with Equation 2.11 multiplied with the parameter t , we end up with Equation 2.13. This equation is a parameterization of the ray that goes from the drones position, p_0 , and in the direction of the rotated camera ray, $r_{ij}(\gamma, \beta, \alpha)$. The parameter t specifies the number of meters from the drone.

$$p_{t,z} = H(p_{t,x}, p_{t,y}) \quad (2.14)$$

Let $H(x, y)$ be the height above sea level for a point in UTM with coordinates x and y . Evidently, the point where the ray hits the ground is where the height of the ray equals the height of the terrain, as shown in Equation 2.14. Hence, the ray in Equation 2.13 needs to be traced from the drone and down to the ground, in order to georeference a pixel in the original image. Consequently, the value t_{ground} that solves Equation 2.14 provides the position, $p_{t_{ground}}$ of the ray-traced pixel in the UTM projection. Finally, $p_{t_{ground}}$ needs to be converted from

the UTM projection to geographic coordinates in the World Geodetic System 84 (WGS 84) [Dpt.Defense, 1991]. This transformation is commonly known and is therefore omitted.

2.4.4 Geolocation to pixel

The previous sections explained how to go from a pixel in an image to a geolocation. However, by some small adjustments, it is possible to go the other way around: from a geolocation to a pixel on the image. In this case, it is actually helpful to imagine the actual path of the ray, contrary to Section 2.4.1. In that case $p_{t_{ground}}$ was the unknown, here i and j are unknown, while $p_{t_{ground}}$ is known. Firstly, the point needs to be converted from WGS 84 to coordinates in the chosen UTM projection. The resulting point is shown in Equation 2.15.

$$p_{t_{ground}} = \begin{bmatrix} x_{t_{ground}} \\ y_{t_{ground}} \\ H(x_{t_{ground}}, y_{t_{ground}}) \end{bmatrix} \quad (2.15)$$

If we rearrange the terms in Equation 2.13, we get the ray from the drone to the ground, as stated in Equation 2.17.

$$t_{ground} * r_{ij}(\gamma, \beta, \alpha) = p_{t_{ground}} - p_0 \quad (2.16)$$

By left multiplying this equation with the inverse of the rotation matrix, $R(\gamma, -\beta, \frac{\pi}{2} - \alpha)^{-1}$, and rearranging the terms, we end up with the ray that is rotated back into the reference frame of the drone and scaled to go from origo to the plane in Equation 2.5. Equation 2.17 shows this.

$$\frac{c_{ij}}{\|c_{ij}\|} = \frac{R(\gamma, -\beta, \frac{\pi}{2} - \alpha)^{-1}(p_{t_{ground}} - p_0)}{t_{ground}} \quad (2.17)$$

Where $\|c_{ij}\|$ equals -1 divided by the z -component of the right hand side of the equation. We have then calculated c_{ij} , and we can go on to rearrange Equation 2.6 to solve for i and j to get the pixels in the image. If $0 \leq i < height$ and $0 \leq j < width$, then the point is inside the image.

2.5 Splitting videos into train, validation and test sets

In order to train, validate and test a model, the dataset is split into three disjoint sets, called training, validation and test. Each of these serves its own purpose. The training set is used to fit the parameters of the model. Models with a large

hypothesis space are often able to fit the training set too well to generalize beyond it. This is called overfitting, and it constrains the model's predictive power outside the training set. Because of this, a separate dataset to control the amount of overfitting is needed.

This is where the validation set comes in. The purpose of the validation set is to provide an unbiased estimate of the model fit on the training set while simultaneously tuning the hyperparameters of the model. The validation set gives a much better estimate of the generalization error than the training set, as the model is not directly fit to this dataset. However, as the hyperparameters of the model are tuned to performance on the validation set, bias is introduced. This is where the third and final dataset comes into the picture, the test set.

The purpose of the test set is to provide a final unbiased estimate of the generalization error of the model. This is used after model training on the training set and hyperparameter tuning on the validation set. No further changes to the model are supposed to happen after it has been tested on the test set, as bias would then be introduced here as well. The training set usually constitutes between 60% and 90% of the original dataset, while the remaining dataset is often split equally between the validation and test set.

2.6 Principal Component Analysis

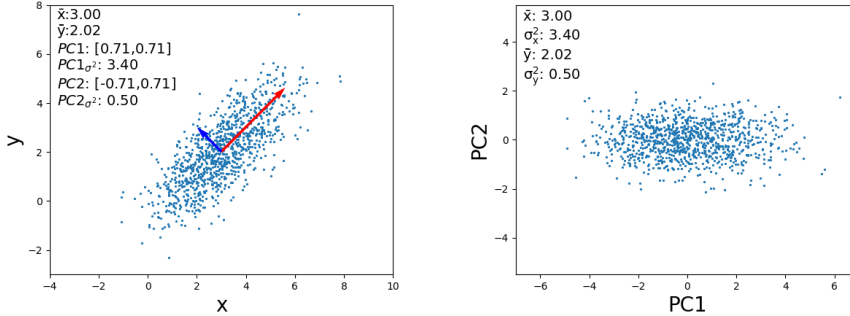
Here we will go through the background theory regarding Principal Component Analysis (PCA). We will use this method later on for color restoration.

PCA is a popular dimensionality reduction technique. PCA is an orthogonal linear transformation, where the data is transformed in such a way that the greatest variance of the data comes to lie on the first axis, the second greatest variance on the second axis and so on. It performs a change of basis on the data by computing the principal components of the data. The first principal component is the line in which the spread of the data varies the most, which is the line that minimizes the mean squared distance from the line to the points. The n -th principal component is the line that is perpendicular to the $n - 1$ first principal components and minimizes the squared distance from the line to the points.

Let $\mathbf{X} \in \mathbb{R}^{n \times p}$ be the data matrix with n data points, p features for each data point and column-wise zero empirical mean. Additionally, let $\mathbf{w}_{(k)} = (w_1, \dots, w_p)_{(k)}$ be the k -th principal component. Then, Equation 2.18 and 2.19 shows how the k -th principal component is defined.

$$\tilde{\mathbf{X}}_k = \begin{cases} \mathbf{X} & \text{if } k = 1 \\ \mathbf{X} - \sum_{s=1}^{k-1} \mathbf{X} \mathbf{w}_{(s)} \mathbf{w}_{(s)}^\top & \text{if } k \in \mathbb{N} \setminus \{1\} \end{cases} \quad (2.18)$$

$$\mathbf{w}_{(k)} = \underset{\|\mathbf{w}\|=1}{\operatorname{argmax}} \{ \|\tilde{\mathbf{X}}_k \mathbf{w}\| \} \quad (2.19)$$



(a) PCA of sampled points from a multi-variate Gaussian distribution. (b) Projection of the sampled points from 2.14a in the principal components.

Figure 2.14: PCA of a multivariate Gaussian distribution. (Created by the authors.)

$$\mathcal{N}\left(\begin{bmatrix} 3 & 2 \end{bmatrix}, \begin{bmatrix} 2 & 1.5 \\ 1.5 & 2 \end{bmatrix}\right) \quad (2.20)$$

The concept of PCA is easy to understand when carried out in two dimensions. In Figure 2.14a we have plotted the result of sampling 1000 times from the multivariate Gaussian distribution in Equation 2.20, as well as found the principal components of the sampled points. The principal components are translated to start from the empirical mean of the samples. Their lengths are scaled to 3 times the square root of the corresponding eigenvalues. These figures are projected into the principal component space and plotted in Figure 2.14b. It is clear from these two figures that the latter is only a translation and rotation of the former. More specifically, to go from Figure 2.14a to Figure 2.14b we first subtract the empirical mean of the sampled points ($\bar{x} = 3.01$ and $\bar{y} = 2.03$) and then we rotate the resulting points clockwise by about 45° , as this is the angle between the first principal component and the first axis. As we see from the two figures, the first principal component captures the most variance, and the second principal

component is perpendicular to the first, and captures the rest of the variance. One should note that the signs of principal components may be flipped. Meaning that in 2.14a there are four different combinations of signs for the principal components. All of these combinations are plotted in Figure 2.15. The reader should also note that PCA is sensitive to scaling of the variables, as the variance along each of the axis would differ. For instance, if we were to plot the length and weight of objects, the units that we choose to measure each of them in, will affect the resulting PCA. For example, if we chose kg instead of g to measure the weight, the variance in the weight would diminish compared to the variance of the length. To mitigate this effect, the variables are often standardized.

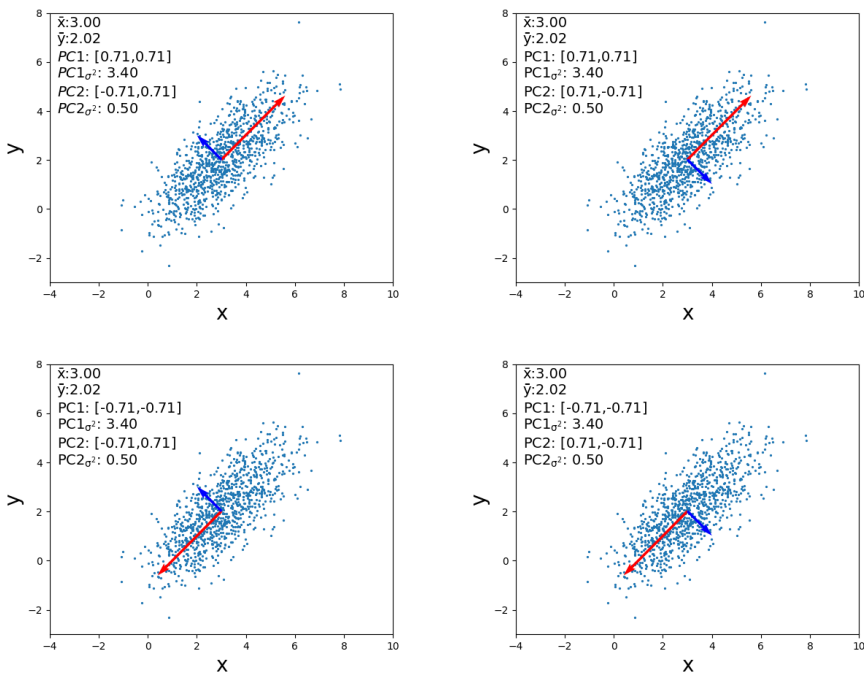


Figure 2.15: The four different combinations of orientation for the principal components. (Created by the authors)

2.7 Structured Literature Review Protocol

To find papers for the literature review, the search engines *Oria*, *IEEE Xplore*, *Google Scholar* and *Papers with Code* were utilized (Table B). The search terms

were selected to be related to multiple aspects within the field of SLAD. The full overview is listed in Table 2.3. Even though we sought papers that satisfied as many term groups as possible, studies that only partially satisfied the search criteria were also included. This was done in order to research specific fields like object detection and segmentation.

The papers that were chosen to be read were judged based on how well their abstract section satisfied the inclusion criteria in Table 2.4. A cutoff score of 2 was chosen, which means that papers that were assigned a score of 2 or higher were studied further. The motivation behind the inclusion criteria is explained in Section 2.7.1. The overview of papers in the literature review, the search engines that were used, the search terms, and their given score can be found in the table in Appendix A. The year limit indicates the time filter applied in the search. An overview of the search engines is provided in Appendix B. After the papers from the literature review had been read, those considered to be the most relevant to our research project were selected. These papers are presented in Section 2.8: Previous Work.

Group 4 in the search term overview in Table 2.3 did not contribute to any meaningful difference in the search results. This is probably due to the words in this group being covered by Groups 3 and 6. Therefore, none of the papers in the overview in Appendix A have G_4 in the search term column.

	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6
Term 1	Camera	Drone	Landing	Obstacle	On-board	Safe
Term 2	Vision	UAV	Zone	Object	Autonomous	Emergency
Term 3	Segmentation	Vehicle	Area	Trajectory	Automatic	Failure
Term 4	Object Detection	Flight	Site	Obstruction	Unknown	
Term 5	Monocular	Aerial	Ground	Clear	Static	
Term 6	Deep learning	Rotorcraft				

Table 2.3: Search term groups containing different synonyms.

Criteria ID	Criteria
IC 1	The study's main concern is finding drone landing areas using monocular vision
IC 2	The study focuses on a subject directly related to finding safe landing areas for drones
IC 3	The study uses or fully focuses on deep learning-based techniques like segmentation or object detection
IC 4	The study is a primary study presenting empirical results

Table 2.4: Inclusion and quality criteria for literature review.

2.7.1 Motivation

The motivation behind each of the inclusion criteria in Table 2.4 is presented below.

1. **IC1** was chosen because a constraint of our research task is to perform SLAD using a single camera view, often referred to as monocular vision. The research in this field would therefore be the most relevant in the context of our research goal.
2. **IC2** was chosen so that the review also would include previous research related to drone landing in general, not only systems using monocular vision.
3. **IC3** concerns deep learning-based techniques specifically, as it is necessary to explore and learn from the previous work in this area. This is important considering it is part of our research goal to develop a deep learning-based system.
4. **IC4** was chosen as it is valuable that the papers we study present empirical data that can be used as an indicator of the performance of their methods.

2.8 Previous Work

The previous research in the field of SLAD can be separated into studies using artificial and natural landmarks. Both of these scenarios will be discussed in 2.8.1 and 2.8.2 respectively. Because of the lack of universally applied benchmarks in this task, each study is evaluated on the qualitative and quantitative results presented. During the project, we found it necessary to perform color restoration on our collected data. Therefore, a selection of previous work within the field of color restoration is presented in Section 2.8.3.

2.8.1 SLAD using Artificial Landmarks

A research area that is well investigated is SLAD using an artificial landmark. An artificial landmark is a special marker that serves as a guiding position for the drone. The drone might use different sensors and techniques in order to perceive the landmark and its location relative to the drone. The task is then for the drone to land on the landmark itself.

The study *A Fast and Accurate Marker Tracker For Autonomous UAV Landing By Visible Light Camera Sensor On Drone* [Nguyen et al., 2018] introduces a modified YOLO-technique called LightDenseYolo to recognize the landmark using a single camera.

The same authors use two CNNs [LeCun et al., 1989] to perform super-resolution [Park et al., 2003] and then landmark detection in the study *Deep Learning Based Super-Resolution Reconstruction and Marker Detection For Drone Landing* [Truong et al., 2019].

Development Of An Automated Camera Based Drone Landing System [Demirhan and Premachandra, 2020] is another paper using an artificial landmark. However, this paper does not utilize deep learning for detection. Instead, the authors use an advanced edge detection algorithm to recognize the unique edges within the "H" on the landing area.

Landing Area Recognition using Deep Learning for Unmanned Aerial Vehicles [Lee et al., 2020] uses a relatively simple approach. The *faster R-CNN* network [Ren et al., 2015] is trained on a custom, hand-labeled dataset to detect a landmark marked with an "H". The accuracy of detected landmarks reaches just above 90%. The system is also able to determine whether the landmark is obstructed, and classify it as *unsafe*.

2.8.2 SLAD using Natural Landmarks

Natural landmarks are properties in natural terrain that can be recognized and used to determine safe landing areas. SLAD using natural landmarks is usually a more complex task than SLAD using artificial landmarks. One reason for this is that artificial landmarks always have the same form, making it easier to train computer vision systems that recognize them. Research in the field of safe landing area detection using natural landmarks (unknown environments) is presented below.

Monocular Vision Slam-Based UAV Autonomous Landing In Emergencies and Unknown Environments [Yang et al., 2018] is a paper using Simultaneous Localization And Mapping (SLAM) [Durrant-Whyte and Bailey, 2006] to perform SLAD. This method first uses the Oriented FAST and Rotated BRIEF feature tracker [Rublee et al., 2011] on multiple frames to generate coarse depth measurements. These depth measurements are used to create a 3D point cloud. The 3D point cloud is then converted into a 2D grid map. The grid map is then divided into different regions, and the flattest region is determined to be the best landing area. A downside of this approach is that the drone needs to fly around and scan an area to create the depth map before SLAD can be executed.

The study *A Vision-Based Guidance System For UAV Navigation and Safe Landing Using Natural Landmarks* [Cesetti et al., 2009] presents a SLAD system using *optical flow* [Beauchemin and Barron, 1995] for depth perception. Natural landmarks are detected using Scale Invariant Feature Transform (SIFT) [LoweDavid, 2004], which is invariant to image translation, scaling and rotation, and partially invariant to illumination changes. First, an image sequence of the landing area is used to find the SIFT features. These are then used to estimate the optical flow in two successive frames. This optical flow is then used to extract depth information from the image. This is done by using feature matching and calculating the distances between the features. A flatness value of the given area is then estimated from the depth information, and a threshold value is used to decide if the surface has variable depth. If it does not, the area is determined to be a safe landing area. The system is capable of estimating depth both when the drone is flying horizontally, and when it is landing vertically. The system seems to provide reasonable landing area recommendations. However, it is only capable of evaluating a whole frame at the time, not a specific location. In addition, it depends on constant movement to estimate depth, meaning the drone will have to keep descending to evaluate the area beneath it. This can be unfavorable in certain situations.

Automated Emergency Landing System For Drones: SafeEYE Project [Bektash et al., 2020] uses deep learning in order to perform SLAD. A downward-facing camera is used to capture the terrain below the drone. The image is then divided into 78 smaller frames of 180x180 pixels. Each frame is classified by a CNN called LeNet [Lecun et al., 1998] to either be a *LandingField* or *NotLandingField*. The CNN is trained to classify frames with green grass fields with low color variation as a *LandingField*. A problem with this approach is that it is tied to finding green, uniform areas, which might make the system ineffective when flying over forests. Another aspect is that at higher altitudes, the frames will cover large

areas, and the system will therefore only classify large green plains as safe landing zones.

Timely Autonomous Identification of UAV Safe Landing Zones [Patterson et al., 2014] relies on Canny edge detection [Canny, 1986] in combination with ordinance surveys to decide safe landing areas. It extracts features from images by analyzing different forms of noise and including information from the ordinance survey. It then uses a fuzzy-set classification with the categories *terrain suitability*, *roughness* and *distance to man-made features* to determine an overall safety score of a landing zone. Areas with a lot of obstacles like trees and rivers will have many edges, while open areas with a low amount of edges will be judged by their roughness. This way, the system is capable of correctly classifying suitable landing areas. The weakness of the method lies in evaluating areas with complex patterns and surfaces.

An Automatic Zone Detection System For Safe Landing of UAVs [Kaljahi et al., 2019] uses a Gabor filter [Daugman, 1988] in eight directions to analyze an image frame. The Gabor filter is used for texture analysis. Regions in the image that are homogenous and plain will receive a Gabor response of low value, close to zero. The pixels that receive a low Gabor response are regarded as a *Candidate Pixel*. To eliminate spurious pixels that have been wrongly classified as candidate pixels, Markov Chain Codes (MCC) [Puterman, 2014] are used. The MCC works based on the fact that the states of the pixels should be defined based on the high probabilities of neighboring pixels. The result is *Candidate Regions* (CR), of which the larger is selected based on a threshold value. This process is carried out on all eight of the resulting Gabor frames. The CRs in the different frames are then compared using Chi-square similarity [Gagunashvili, 2009], and those which are most similar are fused together. This results in a final, single frame that contains the proposed *safe land region*. The study compares the system's capabilities to [Patterson et al., 2014], and the Gabor filter-based system seems to give more reasonable recommendations. However, it classifies water as a safe landing zone, as water qualifies as a homogenous and plain surface. In addition, it is not able to correctly evaluate buildings, and struggles with images of higher altitudes.

Free LSD: Prior-Free Visual Landing Site Detection for Autonomous Planes [Hinzmann et al., 2018] uses a comprehensive framework for SLAD for a fixed-wing, non-VTOL drone using a downward-facing camera. Firstly, Canny edge detection [Canny, 1986] is applied to find edges in the frames. Then, distance transform is applied and a threshold is set. The result is a partitioning of the original RGB frame into the regions that are similar and separated by edges, for instance, grassy areas and fields. A binary Random Forest classifier [Breiman,

2001] is then used to find which of these regions are grass. The homogenous grass regions are then regarded as Regions of Interest (ROI). Consecutive camera frames are used to determine coarse depth measurements, and the ROIs are combined with these measurements using a Rotating Caliper algorithm [Toussaint, 2000] to associate an overall *grade* of the potential sites. These points are given to a *Fine Evaluation backend* that evaluates the points using a 3D point cloud. Finally, because this is a fixed-wing non-VTOL drone, an approach vector is calculated using the local wind field in combination with the location of local hazards. The approach seems to perform well, even though it is more constrained than our research task as the drone requires a considerably larger area to land. In addition, because of the frameworks' dependence on edge detection, it is dependent upon large rectangular areas which have a uniform color and low roughness. This might be unfavorable in Norwegian conditions where fields often are barred and rough.

2.8.3 Color restoration

To explore Research Question 2 regarding color restoration, we also need to go through some previous work in this field. Color restoration is a fairly well-explored field, and is currently dominated by neural networks.

A use-case of color restoration is in underwater photography, in which the water creates a green or blue tint on the images depending on the depth. In the paper *UWGAN: Underwater GAN for Real-world Underwater Color Restoration and Dehazing* [Wang et al., 2019] the authors present a Unet configuration that is capable of removing the color distortion created by the sea.

In the paper *FC4: Fully Convolutional Color Constancy with Confidence-weighted Pooling* [Hu et al., 2017], researchers from Microsoft Research propose a fully convolutional network architecture in which patches throughout an image can carry different confidence weights according to the value they provide for color constancy estimation. Color constancy is the problem of inferring the color of the light that illuminated a scene, usually so that the illumination color can be removed [Barron, 2015]. This network, called FC⁴, proved to be an effective method to reduce and even remove illumination distortion from images.

In the paper *An Efficient PCA-Based Color Transfer Method* [Abadpour and Kasaei, 2007], the authors propose a PCA-based color transfer method which utilizes small sample areas of a source image and a reference image to recolor the

source image.

The paper *Color correction for multi-view images combined with PCA and ICA* [Shao et al., 2007] presents another method of correcting a source image using a reference image using PCA and independent component analysis (ICA). This method is intended to correct the colors of images using multiple views of the same scene. The method is demonstrated to be capable of removing slight color distortions in images.

In the paper *Color Converting of Endoscopic Images Using Decomposition Theory and Principal Component Analysis* [Ansari et al., 2019], the authors use PCA to transfer the color space of a source image to a target image. In this use case, they try to restore RGB colors to narrow-band imagery in endoscopic medical tests. Narrow-band imaging is a technique used to improve visibility inside the human body by increasing blue and green wavelengths.

In Section 3.3, we present a new PCA-based color restoration method inspired by the PCA-based techniques mentioned above.

Chapter 3

Method

In this chapter we will explain the decisions and setup behind the individual components that are included in our segmentation and object detection-based systems. Towards the end of the chapter, we will explain how each of these components are combined into the overall systems.

3.1 Drone Dataset

3.1.1 Data Collection

In order to train, validate and test the SLAD system, collecting data is a natural first step. Hence, the video that is streamed from the drone to QGC, as described in Section 2.1.3, needs to be stored. This was carried out by creating a systemd service onboard the companion computer that listens to the video stream on system boot. The stream capture component is depicted in Figure 3.2. By having the service on the drone, we avoid congesting the Virtual Private Network (VPN). However, this means that the captured videos need to be extracted manually. Another risk of storing the videos onboard the companion computer is that it might run out of storage, as the flights are very long, and hence also the videos. The drone stores telemetry data while flying in addition to streaming video to an operator. If the computer ran out of storage it could interfere with the mission of the drone. This risk was mitigated by choosing a low resolution, of 320×240 , for the videos. This resolution was chosen in accordance with Aviant, as it was low enough to not take a lot storage space, but still enough to clearly see objects on the ground. In addition, real-time video compression was applied to the video stream, such that 1 hour of video only required about 500MB of memory. The videos were also routinely transferred from the companion com-



(a) An image from a video collected by *NT02*.
 (b) An image from a video collected by *NT04*.

Figure 3.1: Sample images from *NT02* and *NT04*.

puter in order to reduce the risk of running out of storage.

The data collection was started 17th of March 2022 and lasted until the 19th of May 2022. The stream capture service was set up on two of Aviant’s operational drones: *NT02* and *NT04*. A total of 38 videos were collected in this time period. The videos are from different routes, in varying weather conditions and captured at different times of the day. All of these three factors diversify the features of the collected data and increase the feature space. Figure 3.3 shows all the different routes that *NT02* and *NT04* flew in time period stated earlier. Table 3.1 shows when they were flown. 19th of May, during the data collection period, *NT04* became the first drone to officially cross a border between two countries in Northern Europe [Iver Waldahl Lillegjære, 2022].

There were some issues with the captured videos. As can be seen in Figure 3.1a, the video of *NT02* was detailed, but suffered from a red tint. Attempts were made to try to remove this distortion, but the issue was a broken pin connector on the camera of the drone. There were also issues with the video captured by *NT04*, as can be seen in Figure 3.1b. These videos had their color intact, but issues with the lens had created a white blur effect. In Section 3.3, we will show how we restored the colors in the red videos of *NT02* using the blurry videos from *NT04*.

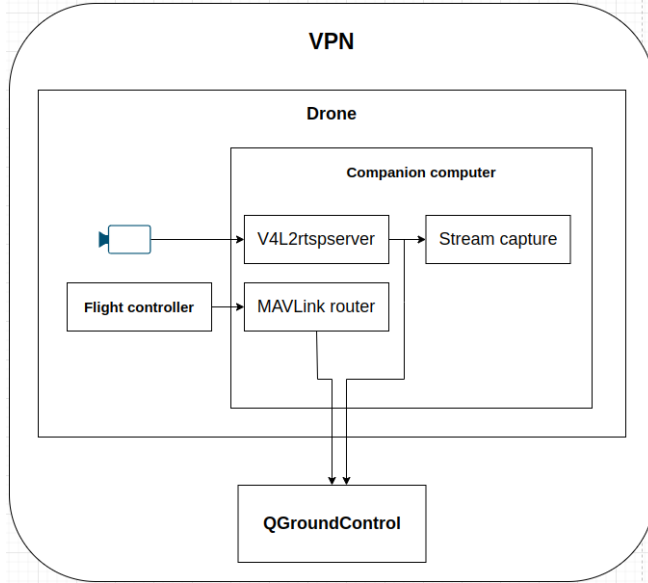


Figure 3.2: Setup showing the required components within the drone to capture video.

From	To	Mission path	Dates
Sandmoen	Haltdalen	3.3a	22.03, 22.03, 23.03, 29.03, 29.03, 01.04, 22.04, 18.05
Haltdalen	Sandmoen	3.3a reversed	22.03, 22.03, 23.03, 29.03, 29.03, 01.04, 22.04, 22.04
Kongens gruve	Sandmoen	3.3b	01.04
Sandmoen	Kongens gruve	3.3b reversed	01.04
Sandmoen	Sørungen and back	3.3c	12.04, 12.04, 13.04, 11.05
Langørjan, circles		3.3d	19.03, 19.03, 19.03 , 31.03, 31.03 , 04.03, 04.03, 04.03, 09.05, 09.05, 10.05, 10.05
Langørjan, 8° test		3.3e	21.03, 21.03 , 31.03
Funäsdalen	Røros and back	3.3f	19.05

Table 3.1: Flights captured on video in the data collection period. The dates marked in bold are the selected flights that were labeled and put in the dataset.

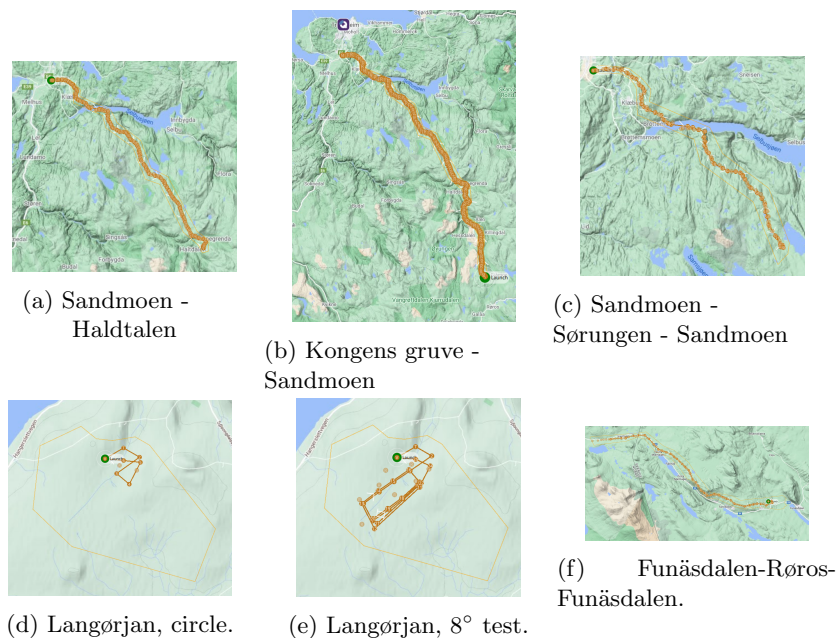


Figure 3.3: Overview of all the missions that were flown in the data collection period (Maps from [Google Maps, 2005]).

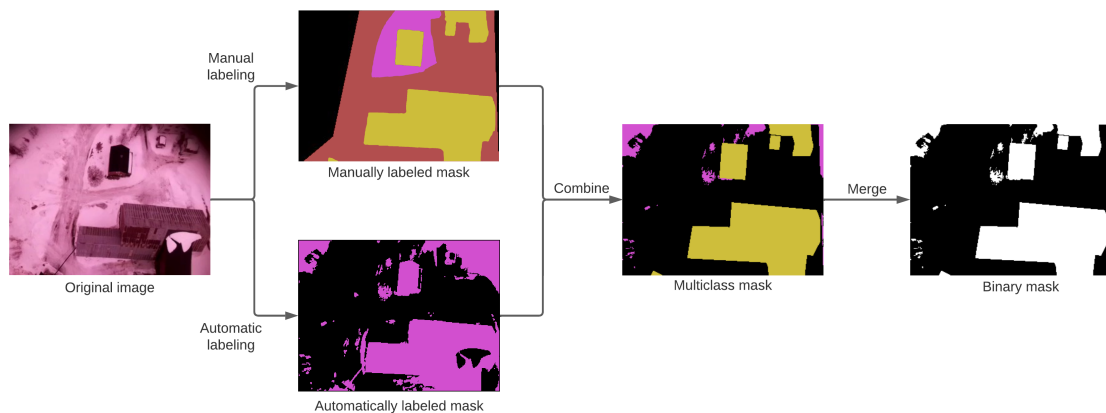


Figure 3.4: Overview of the labeling process of the collected video.

3.1.2 Data Labeling

The videos from the drone had to be labeled to be used in model training, validation and selection. The categories used are listed in Table 3.2. Only *snow* and *field* are considered safe areas to land, the rest are obstacles.

Manual Labeling

To label images manually, Computer Vision Annotation Tool (CVAT) [CVAT, 2022] was set up and hosted on a Linux server owned by the authors. The categories that were used for labeling the images are listed in Table 3.2. In total, 11733 images were manually labeled. The categories *field* and *trees* were special categories that determined what areas of images should be handled by the automatic labeling system.

Automatic Labeling of Snow and Trees

In accordance with the safety procedures outlined in Section 2.1.2, a significant portion of missions are over desolate areas to avoid risk. Consequently, most of the obstacles in each frame of the collected data are trees. Furthermore, as the videos were recorded in March and April, most of the ground was covered in snow. Hence, if we consider snow-covered areas as safe for landing, there is a stark contrast between safe landing areas and obstructed ones, as the trees are not covered in snow. This conspicuous difference in color between trees and snow enables a simple and automatic approach for labeling trees and snow, which accounts for a substantial part of the collected data.

The technical structure of the automatic labeling system is simple, yet effective on the collected data. The process is visualized in Figure 3.5. The image is first grayscaled, then all pixels with a value of 150 or more, in a range from 0 to 255, are labeled as snow, while the rest is labeled as trees. In the rightmost image in the figure, this mask is overlaid on the original image. We see that this provides a very accurate detection of the trees in the image.

Combining the Labelings

When labeling an image, we combine the manually labeled mask with the automatically labeled mask. The manually labeled mask has precedence over the automatically labeled mask for all categories, except when the manually labeled category is *tree*. Because of a limitation in CVAT which makes it impossible to create holes in masks, trees in the middle of fields had to be manually labeled. An example of this is the pink area in the *Manually labeled mask* in Figure 3.4, which ends up being labeled more accurately by the automatic labeling system

in the *Multiclass mask* image. Here, the manual labeling specifies the areas in which the trees are located, then the automatic labeling system improves upon this mask. Trees outside of this special case did not have to be manually labeled, as they would be handled by the automatic labeling system.

To see why the *Manually labeled mask* has precedence over the *Automatically labeled mask*, we consider two cases, the case where the *Manually labeled mask* says that there is an object while the *Automatically labeled mask* do not, and the opposite case, where the *Manually labeled mask* says that the area is landable while the *Automatically labeled mask* says otherwise.

In the first case, we have snowy areas that are not suitable for landing, for example if there is snow on the roof of a building. An example of this is the snow on the building in the *Original image* in Figure 3.4. We see that this area is considered to be a safe landing area in the *Automatically labeled mask*. Therefore, the manual labeling is prioritized over the automatic labeling if there is a conflict between the labeling of obstacles.

In the second case, we have dark areas that are actually safe to land on. These are wrongfully considered to be obstacles by the automatic labeling system. In these areas, a special *field* category was used in the manual labeling to signify an area that is safe to land on. The classifications from the automatic labeling system were then omitted in these areas. The brown area around the houses in the *Manually labeled mask* in Figure 3.4 is therefore considered to be a safe landing area in the *Multiclass mask*, despite the classification from the automatic labeling system in the *Automatically labeled mask*.

Binary Masks

As shown in Figure 3.4, the *multiclass mask* is eventually merged into a *binary mask* representing whether the area is safe to land on or not. The reason we use binary masks as opposed to multi-class masks is that it is not necessary for the models to encode the differences between obstacles. The actual information that is valuable in our case is whether the area is safe for landing or not. The assumption behind this decision is that it might be easier for the networks to learn our desired behavior when it only needs to regulate two classes. This way, no excess information is encoded into the network, but only the information that is valuable. Furthermore, this means that the concept of *obstacle* may generalize to objects that are not part of the dataset, but occur in the real world.

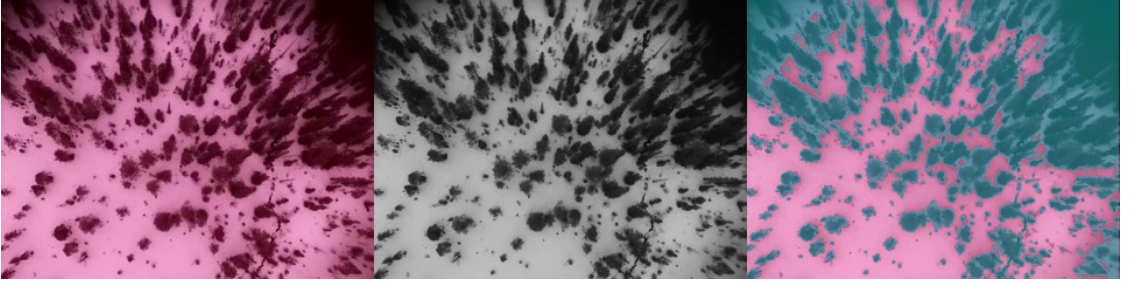


Figure 3.5: Automatic labeling of trees in snow. The leftmost image is the original, the center image is the image after being grayscaled, and the rightmost image shows the generated tree mask on top of the original.

3.1.3 Dataset Partitioning

As the collected dataset is to be used for model training, hyperparameter tuning and testing, it is split into three sets for training, validation and testing. A division of approximately 70% of the images for training, 15% for validation and 15% for testing was chosen. When deciding how to split the data, we need to consider the distribution from which the data is sampled. As we are dealing with images that originate from videos, the images have a temporal aspect. This means that features that occur in one image are likely to be in preceding and succeeding ones as well. Consequently, if the images were randomly split into three datasets, data leakage would occur and both the validation and test set would lose their purpose.

Hence, when splitting the dataset, 30 frames directly preceding and succeeding a chosen segment of consecutive images are discarded to alleviate this problem, as no frames in each of the sets would have any overlap. Due to the safety requirements for missions, as explained in Section 2.1.2, populated areas are avoided. This has a direct consequence on the features observed in the videos and their temporal placement. The majority of people, buildings, vehicles and roads are observed at the beginning and end of the videos, as the drone takes off and lands in sparsely populated areas. While the rest of the video from the flights, which constitutes the vast majority of the flight time, is mostly limited to the three features: rivers, trees and snow. Hence, the normal way of dividing temporal data into train, validation and test sets, which is to split the data into three contiguous disjunct parts, is considered unfavorable as the feature space in each of the datasets would then be very different from one another.

Table 3.3 shows the number of images containing different obstacles. We see

that there are few people, power lines, roads and vehicles in the dataset. In fact, there are only four instances of unique persons and two instances of power lines.

Category	Color(RGB)	Description
People	196, 30, 8	Red
Power Line	205, 120, 161	Light pink
Road	11, 236, 9	Green
Vehicle	192, 255, 193	Light green
Water	29, 26, 199	Blue
Building	206, 190, 59	Yellow
Trees	211, 80, 208	Pink
Other obstacle	68, 218, 116	Light green
Snow	0, 0, 0	Black
Field	0, 0, 0	Black

Table 3.2: The categories used when manually labelling images. Black indicates safe landing area.

Dataset	People	Power line	Road	Vehicle	Water	Building	Trees	Other	LA	Total
Train	207	40	85	215	2430	1042	3651	708	6668	7532
Validation	126	0	128	0	512	108	769	27	1805	1985
Test	152	90	123	30	410	103	903	70	2096	2216
Total	485	130	336	245	3352	1253	5323	805	10569	11733

Table 3.3: Number of images containing different obstacles in the collected drone dataset. Note that this is images in total, not unique instances. There are multiple images of the same instance of an obstacle.

3.2 Pretraining Datasets

3.2.1 Potsdam Dataset

The Potsdam dataset [Potsdam, 2016] is a dataset of true orthophoto taken over the german city Potsdam. It comes with labeled ground-truth masks for semantic segmentation. The labeled classes include *Impervious surfaces*, *Building*, *Low vegetation*, *Tree*, *Car* and *Clutter/background*. Because it is taken over a city, it largely consists of images of urban environments. However, since the city is not that dense, it also includes trees and park areas. The purpose of selecting this dataset for pretraining was because the drone videos contained limited examples of buildings and cars. The Potsdam dataset contains a variety of buildings and

cars, and has a visual similarity to our own real-world data.

The dataset originally comes as 38 large patches with a resolution of 6000x6000. A script was used to split these up into images with a resolution of 500x500. This results in a total of 5472 images with accompanying segmentation masks. The classes mentioned earlier were evaluated whether they could be designated as safe landing areas. Of the classes, *Impervious surfaces* and *low vegetation* were determined to be safe landing areas, and therefore given the same color (black). The other classes were given appropriate mask colors in accordance with Table 3.2. Since this dataset only will be used for pretraining, a test set is not necessary. A training/validation set split was therefore chosen at 80/20 %. An example of an image from the Potsdam dataset can be seen in Figure 3.6.

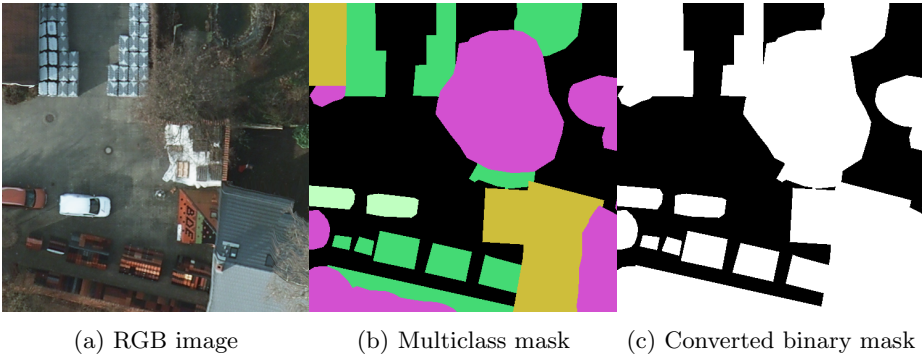


Figure 3.6: Example image from the Potsdam dataset (Source: [Potsdam, 2016]).

Dataset	People	Power line	Road	Vehicle	Water	Building	Trees	Other	LA	Total
Train	0	0	0	2281	0	3277	3557	3073	4224	4266
Validation	0	0	0	553	0	818	883	742	1048	1062
Total	0	0	0	2834	0	4095	4440	3815	5272	5328

Table 3.4: Number of occurrences of obstacles in the Potsdam dataset.

3.2.2 Unreal Engine 4 Dataset

The Unreal Engine dataset is an artificial dataset generated by the authors. It contains scenes from the 3D game engine Unreal Engine [Unreal Engine, 2022]. The drone videos (Section 3.1.1) contained few examples of people, and this synthetic dataset was generated for the purpose of helping the system detect people

in the terrain.

The dataset was generated using a plug-in to Unreal Engine called Airsim [AirSim, 2017]. AirSim allows us to simulate a drone camera taking pictures over a map. In addition to taking RGB orthophotos, AirSim also provides functionality to simultaneously generate exact masks. By specifying which objects are to be labeled with which colors, our script can then generate a pixel-perfect mask-RGB picture pair. The masks include objects of the categories *trees*, *water*, *building*, *other obstacles* and *people*. Images with accompanying masks can be seen in Figure 3.7.

The camera was programmed to move across the map in steps, and appropriate coordinates with different offsets were determined to avoid capturing overlapping images. Four maps were chosen to be included in the dataset: *Landscape Mountains* [LandscapeMountains, 2020], *Stone Pine Forest* [StonePineForest, 2021] and *City Park* [CityPark, 2021]. These were chosen because they all imitate nature, with the objective of getting a closer resemblance to the captured real-world data. In addition, the maps provide their own respective terrains. The Landscape Mountain map provides a snowy winter landscape, the Stone Pine Forest map provides a pine forest, while the City Park map provides a flat park-landscape with unique features such as buildings and playgrounds.

To further imitate the real-world data, three height intervals were set in order to get images from different heights. This was done in order to replicate the situation when the drone is taking off or landing and to simulate flight at different heights above ground. The different elevations can be seen in Figure 3.7.

To place people on the maps, a pre-made collection of people-models called People Pack [PeoplePack, 2019] was used. This collection of people was chosen because it contains models of people of different sexes and ethnicities. Having a diverse selection of people might not matter as much when the images are taken from high altitude, as they are then only represented by a few pixels. However, when taking images from a lower altitude, more details are visible, and it might therefore be beneficial to have a wider selection of people. In order to maximize the exposure of people to the neural networks, a whole crowd of people was placed all over the maps. This made it so that one frame could contain up to about 50 models of people.

To further add to the realism of the images, the time of day was also incorporated. Unreal Engine can render different lighting conditions based on the time of day. 7 timestamps were chosen, and the images were captured once for

each height at each timestamp. The timestamps were chosen to get a variety of shadows in the images. The hours at night when it was darkest were not selected, as the research in this thesis focuses on SLAD in daylight conditions. As with the Potsdam dataset, a training/validation split of this dataset was chosen at 80/20%.

The synthetic Unreal Engine dataset and its potential weaknesses will be discussed in Section 5.2.

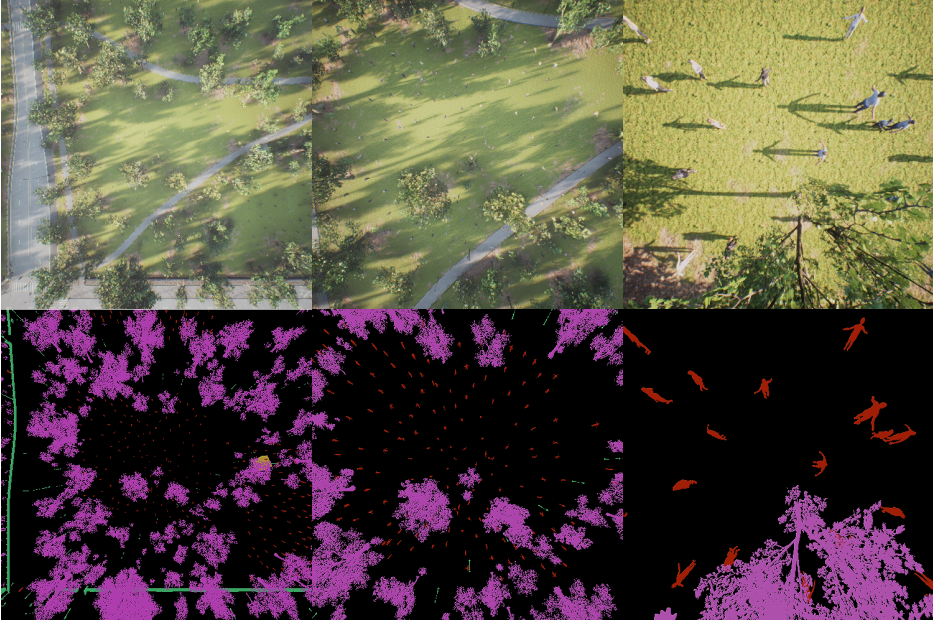


Figure 3.7: An example of images in the synthetic Unreal Engine dataset with corresponding masks. The images are taken from three different elevations.

Dataset	People	Power line	Road	Vehicle	Water	Building	Trees	Other	LA	Total
Train	2473	0	74	0	542	345	3580	1155	3705	3729
Validation	560	0	0	0	62	104	898	273	924	924
Total	3033	0	74	0	604	449	4478	1428	4629	4653

Table 3.5: Number of occurrences of obstacles in the Unreal Engine 4 dataset.

3.3 Color Restoration using Principal Component Analysis

The images from *NT02* do not have the correct color, as illustrated in Figure 3.1a. This might be an issue, as pretraining could be less effective if the data distributions are significantly different. Hence, we have to map the red-tinted images from *NT02* to the correct colors. This poses a problem, as the ground truth colors are unknown. Fortunately, *NT04* has flown some of the same missions as *NT02*. And even though these videos suffer from a white fog effect caused by the lens, it does not have the same color issue as *NT02*. It is possible to exploit this by matching similar images from flights by *NT04* and *NT02* and use the color palette from the former to color the latter.

We take inspiration from the previous work on color restoration which we explored in Section 2.8.3. We did not want to apply yet another neural network to this task, as two consecutive neural networks in the SLAD system might be slow and unpredictable. Therefore, we present a color restoration technique using PCA.

3.3.1 Overall Idea

To map the colors in one image to the colors of another, the authors propose an approach where the principal components of the colors of two similar images are calculated separately. Then the colors of the source image are transformed to its PCA space and inversely transformed by the principal components of the other. To generalize this mapping to unseen images, the process is repeated several times for other pairs of similar images from the two drones, then the mappings are averaged. In total, there are more than 16 million (256^3) colors that are possible to map from, and equally as many to map to. The colors in the images used for mapping do not span this entire spectrum. To cover more colors, a breadth-first search is employed to cascade the transitions to nearby colors.

The approach outlined above relies upon four critical assumptions:

1. The two distributions have a similar shape
2. Corresponding features in the two distributions have the same place relative to their distribution
3. The domain of the mapping is the possible colors
4. Colors that are close to each other have a similar mapping

The overall idea of this color mapping method is to rotate one color distribution to another. This is what is stated in assumptions 1 and 2. The first assumption is necessary as PCA is used to find the orientation of the color distributions. If the shape of the distributions is not similar enough, the transformation might be misaligned. The second assumption comes on top of the first one, in the sense that it is not enough that the distributions have a similar shape. Features in the images need to be placed similarly in their distributions. This means that when we rotate one distribution to the other, similar features in the two images occupy similar areas. The third assumption assumes the domain of the mapping function. For this approach to work, one color has to map to another color regardless of the colors of the surrounding pixels, the brightness of the image etc. Intuitively it makes sense that a color always maps to another color regardless of other factors than its color. However, it is trivial to come up with a counterexample, where this assumption is violated. For instance, when gray scaling an image, a 3D space of colors is mapped onto a range of grayscale by Equation 3.1, and information is lost. Thus, mapping from the grayscale domain to the color domain would violate the third assumption as a shade of gray maps to a plane of colors, not just one.

$$g(r, g, b) = 0.3 * r + 0.59 * g + 0.11 * b \quad (3.1)$$

The last assumption is necessary to increase the domain of the mapping. The colors in the training images are not enough to make a satisfactory mapping for the colors in images outside the training set. This assumption means that we can simply increase the domain of the mapping by just expanding the transformation to the nearby colors. Such an expansion would increase the domain of the mapping cubically with the radius of the expansion, as there are three color channels.

3.3.2 Change of Principal Components Basis

Let the $c_{r,g,b}$ denote a color with r , g and b as values for the amount of red, green and blue. An image can be converted to a multiset of c , where each c corresponds to the color of a pixel in the image. Let W^f and W^t be the eigenvectors of the covariance matrices for two images that fulfill the above requirements. Then one can transform the colors of one image to the other by first transforming the colors to its image PCA space, then inversely transforming to the other, as given by Equation 3.2.

$$c_{r^t, g^t, b^t} = c_{r^f, g^f, b^f} W^f (W^t)^{-1} \quad (3.2)$$

To get a good mapping, image pairs fulfilling assumptions 1 and 2 is necessary. Hence, images where *NT04* and *NT02* flew over the same area are good candidates. However, even though some of the missions are identical, features in

the images from those flights are noticeably different, as the videos from *NT04* are about a month after the videos from *NT02*. This means that the amount of snow present in the videos is significantly less in flights by *NT04* than *NT02*. Furthermore, the time of day needs to be taken into account, as shadows are a distinct feature. Consequently, only a few images from *NT02* flights share the same features as images from *NT04* flights.

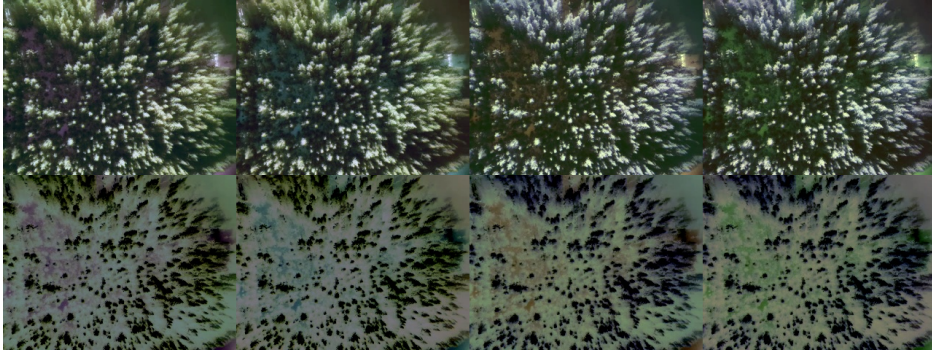


Figure 3.8: Comparison of resulting colors when changing axis in the PCA color restoration.

The signs of the principal components are arbitrary, as explained in Section 2.6 Principal Component Analysis. This means that a principal component can be flipped. When mapping between the two color distributions, this needs to be taken into account. Otherwise, the distribution will not be rotated correctly to the other. Hence, the signs of the principal components in W^f might have to be changed.

As there are 3 principal components, and each one of them can either be flipped or not, there are 8 possible transformations (2^3). Figure 3.8 shows all the different ways to flip the 3 principal components. In the second row, the first principal component is flipped, in the second and fourth column the third principal component is flipped, and in the third and fourth column, the second principal component is flipped. It is evident that the first principal component should not be flipped, as the second row of images is not having the correct colors, so we can disregard this row. Furthermore, we see that the second principal component determines the color of the tree tops. If it is flipped they are a shade of white. Otherwise, they are light green. Hence, we can disregard the former images. When comparing the two remaining images, we see that the only significant difference is the color of the snow. It either has a red or a green shade. None of them are perfect, but

we think that the latter looks more natural.

The result is that the image in the first row and the second column is the correct mapping from the red image in Figure 3.9. The order in which the images were discarded might seem natural: grouping similar images and choosing the most natural-looking. But this is linked to the variance that each principal component explains. The first principal component encompasses the most variance, hence by flipping this, we get entirely different images. The second principal component is less so, resulting in somewhat different images. And the last principal component, which explains the least amount of variance, is where we have to study the tiny amount of snow to spot the difference.

The above approach is repeated for all pairs of images with similar features. These reference images are depicted in Appendix B. The images selected, cover most of the features that are seen in the videos. The images from *NT02* are from flights carried out 29.03, 01.04, 13.04 and 22.04, and the images from *NT04* are from flights carried out 09.05, 11.05 and 18.05.

The top row in Figure 3.9 shows the original red image that is used in Figure 3.8, the target image and the resulting image. The bottom row shows the scatter plot of the colors in the images directly above. From these plots, we see that the rightmost, final color distribution is just the first one that has been translated and rotated in accordance with the distribution in the middle.

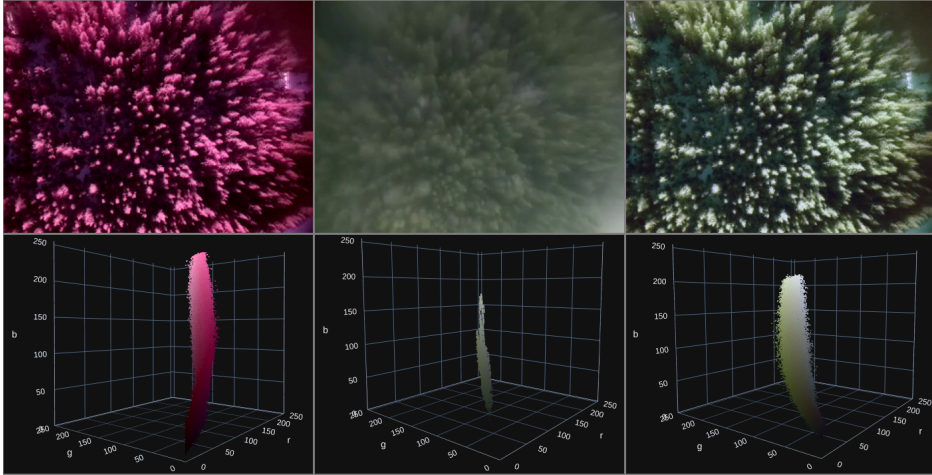


Figure 3.9: PCA of image colors. The red image is captured by the *NT02* drone. The center image is a distorted image captured by the *NT04* drone. The right-most image is the original red image after applying the colors from the center image using the color mapping technique. The bottom row shows a scatter plot of the colors in each image. More examples of color-restored images are provided in Appendix C.

3.3.3 Generalize to Unseen Images

Repeating the approach above means that we can get a good estimate of the mapping from the colors that are present in the training images. As the chosen training set is limited, and the color variation in them restricted, this mapping does not have a sufficient domain to be able to generalize to unseen images. Hence, to expand the domain, a breadth-first search starting from the elements in the domain is used. A maximum euclidean distance of 5 is searched. This expands the number of mappable pixels from 94943, the number of colors present in the images in the left column in Figure 1, to 1392893. Thereby, increasing the domain of the mapping by a factor of 14.6. The domain of the mapping then covers about 8.3% of all the possible colors. This is showed in Figure 3.10. Images in the same column display the same scatterplot from different angles, such that the 3D structures of the distributions are clearer. The leftmost column displays the colors in the training images from *NT02* and the column to the right displays the colors that they are mapped to. The rightmost and second rightmost columns display the same distributions after the breadth-first search. By comparing the rightmost columns to the leftmost columns, we see that they are identical, except

that the former has substantially more points than the latter. The rightmost column clearly outlines the boundaries of the mapping, as it is very visible which colors are part of the mapping, and thus can be restored through the restoration process. From this column, we see that shades of white, green, red, orange and yellow are the colors that can be restored.

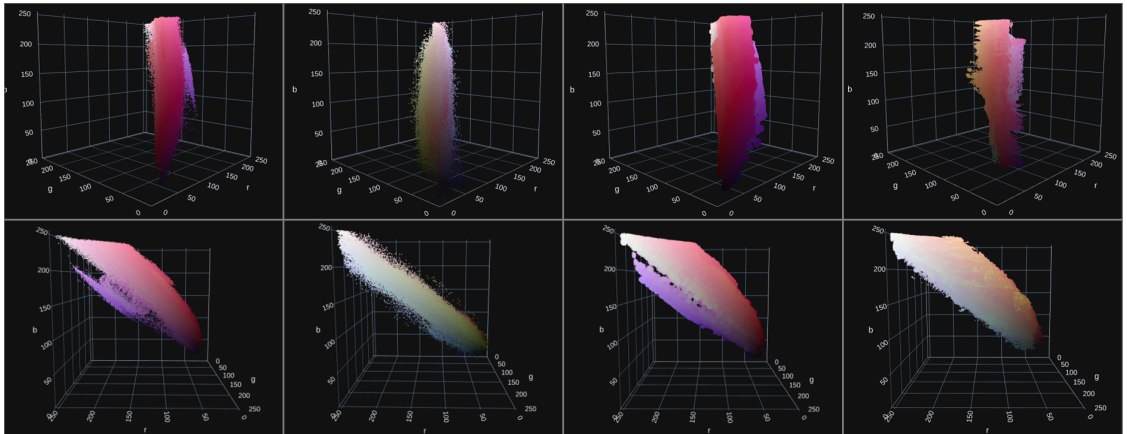


Figure 3.10: Scatter plots of PCA color restoration distributions.

3.4 Converting Segmentation to Rally Points

In Section 3.1.2, we described the process of how we labeled the images we collected from the drones. These labeled masks tell us where there are obstacles in the images. However, to find the best landing area for the drone, we need to find the areas in the masks that are as far away from obstacles as possible. Therefore, we present a method to convert a semantic segmentation mask into a set of landing areas.



Figure 3.11: Converting segmentation mask to specific rally points.

Figure 3.11 illustrates the different steps in this process. Starting from the left, we have the input image and the corresponding semantic segmentation mask, which comes as a result of the labeling process described in Section 3.1.2. From the semantic segmentation, we find the distance to the closest obstacle for each pixel by using a modified version of Dijkstra’s algorithm [Dijkstra, 1959] where the cost to a node is the distance from the closest node. The result of this is an image like the second from the right in the figure. The further away the closest obstacle is from the pixel, the brighter the color of the pixel. By looking at this image, we see that there is one global optimum, and several local optima. To find the best landing area, one simply chooses the brightest pixel. The radius of the landing area then becomes the distance from this point to the closest obstacle.

The question then becomes if other landing areas should be considered and how. One can not simply choose the second and third brightest pixel as the second and third best places to land, as these are often situated right next to the brightest pixel. In that case, these points are essentially just off-center placements in the original landing area. Hence, we choose the second-best landing area to be the brightest pixel that is at least 1.5 times the radius of the best landing area away from the brightest pixel. This allows for some degree of overlap between the landing areas, while not being too close to each other. The third best landing area then has to obey this restriction for both the best- and the second best

landing area. This process is repeated until the best potential landing spot has a distance of fewer than 10 pixels to the closest obstacle. 10 pixels is chosen as this is approximately 10m when the drone is 100m above the ground. The result of this can be seen in the rightmost image in Figure 3.11.

3.5 Pixel Geolocator

Even if we can evaluate which areas in the terrain below the drone that contains obstacles, and which are safe for landing, this information is of little use if it is in a format that cannot be used by the drone. Therefore, we need a way to convert the pixels in the images into GPS coordinates. We also need to go the other way around, from a GPS coordinate to a location on the image, in order to verify a rally point. To do this, we introduce the pixel geolocator.

We outlined the steps for geolocating pixels and the process of going from a GPS coordinate to a point in the image in Section 2.4. We implemented these steps in Python. To find the correct orientation of the drone, the roll, pitch and yaw were extracted from the drone’s gyroscope (γ, β, α) . These values were used to calculate $r_{ij}(\gamma, \beta, \alpha)$ from Equation 2.11. Right before the drone transitions from multirotor mode to fixedwing mode, it rotates to face the correct orientation for the flight path. This abrupt change in yaw was used to synchronize the metrics from the flight controller and the video stream. Furthermore, FOV_h and FOV_w is measured to be 1 and 1.4, respectively.

To find the point where the ray hits the ground, we need to approximate the elevation of the terrain, this corresponds to $H(x, y)$ in Equation 2.14. Elevation maps from *Høydedata* [Høydedata, 2017] was used for this purpose. They have collected elevation data from Norway and created national elevation maps with resolutions of 2500m^2 , 100m^2 and 1m^2 . Due to storage considerations, the resolution used for ray tracing in our case was chosen to be 100m^2 . This means that for every 100m^2 there is an estimate of the height above sea level. We call this $\tilde{H}(x, y)$ and uses this as an approximation of $H(x, y)$, as stated in Equation 3.3.

$$H(x, y) \approx \tilde{H}(x, y) \tag{3.3}$$

3.5.1 Mapping from Pixels to Coordinates

By including the approximation for H , we can proceed to trace the ray to the ground. Equation 2.14 then becomes Equation 3.4.

$$p_{t,z} = \tilde{H}(p_{t,x}, p_{t,y}) \quad (3.4)$$

To find the t that satisfies Equation 3.4, we check the elevation of the terrain and the height of the ray for every meter in the ray direction, until it either hits the ground or reaches 500m. This is stated in Equation 3.5. The drone is never more than 120m above the terrain. Meaning that the point that is straight down, will never be more than 120m away from the drone. However, when it is in the middle of a turn, the camera is no longer filming the ground directly below, as the camera is following the rotation of the drone. Hence, to account for this we chose the upper threshold for t to be 500m, as we expect that any point further away is too noisy.

$$t_{ground} = \min(\{t | \tilde{H}(p_{t,x}, p_{t,y}) \geq p_{t,z}, t \leq 500, t \in N\}) \quad (3.5)$$

Then we need to find t_{ground} that solves Equation 3.4. Raytracing is an active area of research, and there are several ways to optimize for performance. This is a necessity in scenarios with high image resolution, high frame rate and multiple reflections [Deng et al., 2017]. However, in this scenario, we only have a few rays, a frame rate of 10 frames per second, and no reflections. Hence, to solve Equation 3.4, t is linearly spaced from 0 to 500 with a spacing of 1. The first occurrence where $p_{t_n,z} \leq \tilde{H}(p_{t_n,x}, p_{t_n,y})$ is the value of t_{ground} .

3.5.2 Mapping from Coordinates to Pixels

To go from a UTM coordinate to a location in an image is simpler than the above approach, as the only *ray tracing* needed is to calculate where the camera ray hits the plane in Equation 2.5. We can see an example of this mapping by looking at the images in Figure 3.12. Figure 3.12b shows the location of the rally point in a satellite image, while Figure 3.12a shows the resulting point when mapping from the GPS coordinates onto an image from the drone. In the upper left corner, we see that the drone is currently in a roll of -19° . Figure 3.13 is a 3D view of the drone, and illustrates the deviation between the points in Figure 3.12.

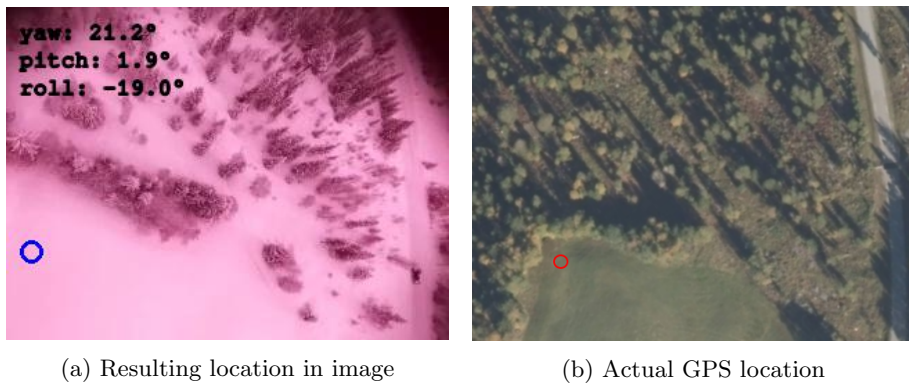


Figure 3.12: Comparison of points from pixel geolocator. (Rightmost image from [Norge i bilder, 2005])

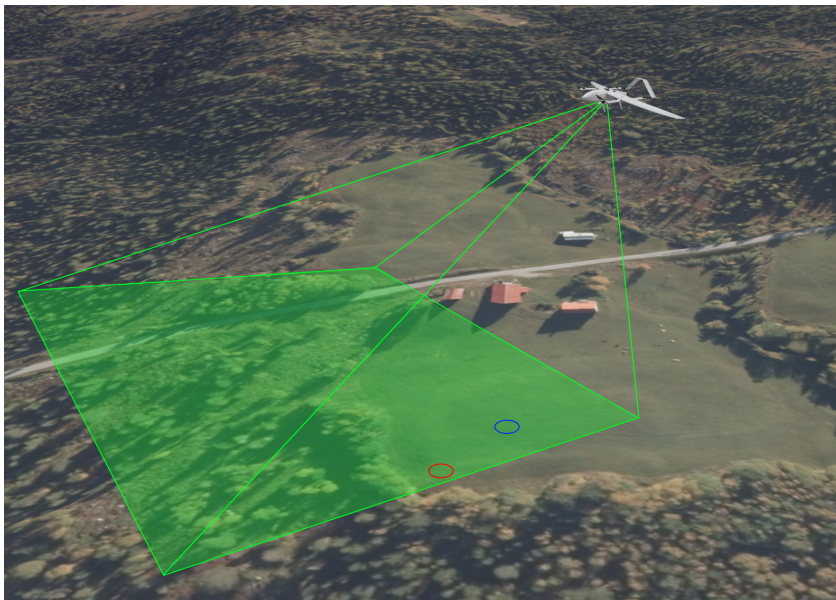


Figure 3.13: 3D view of the drone with estimated and actual locations from the pixel geolocator. (Satellite imagery from [Norge i bilder, 2005], image of drone provided by Aviant.)

3.6 Segmentation

One of the techniques we will use to extract information from the images is image segmentation. Image segmentation might be a particularly suitable technique for safe landing area detection using monocular vision. This is because segmentation classifies each pixel in the image, which again makes its output very rich in information. This information can be valuable, considering we do not have any other sensors that can evaluate the real-time state of the ground, except for a single camera. Towards the end of this section, we will bring together the systems presented in this chapter, and introduce the unified segmentation-based SLAD system.

3.6.1 Tested Architectures

The segmentation architectures tested in this thesis are Unet, Swin-Transformer and ConvNeXt.

Unet was chosen because it is one of the most implemented segmentation methods according to [Papers with code, 2018]. Even though it has been surpassed by newer models in certain benchmarks, it can in our case serve as a valuable benchmark to compare with.

Swin-T was chosen because it has achieved state-of-the-art performance in segmentation benchmarks such as COCO and ADE20K. In addition, the swin-transformer incorporates the relatively new concept of transformers. It is therefore of scientific value to evaluate whether transformers can prove beneficial in our use case.

ConvNeXt is a new CNN-based architecture from researchers at Facebook. It was created to match the performance of transformer-based methods while using a more traditional CNN architecture. This model was chosen as it is of scientific value to see how an advanced, state-of-the-art, CNN-based architecture performs in our use case.

Both Swin-Transformer and ConvNeXt are backbones used in combination with UPerNet [Xiao et al., 2018]. UPerNet is a perceptual parser that is capable of considering unique characteristics such as objects, parts, materials and textures when performing segmentation. It is used in conjunction with most state-of-the-art backbone architectures.

3.6.2 Training Setup

Computers

All training and testing were performed on the authors' personal computers. The two computers had their own discrete graphics processing units (GPU), the Nvidia [Nvidia, 1993] RTX2070 Super and the RTX3080. Each of the systems had 8 GB and 10 GB of graphics memory, respectively.

Ground Truth

The ground truth masks for the segmentation networks are binary masks created from labeled images as explained in Section 3.1.2: Data Labeling.

Frameworks

For Swin-T and ConvNeXt, we used the framework MMsegmentation [MMSegmentation, 2020], which had implementations of both architectures. To train Unet, an implementation from the Segmentation Models PyTorch (SMP) library [SMP, 2019] was used. Both MMsegmentation and SMP use PyTorch [PyTorch, 2019].

Pretraining

In order to explore the use of pretraining on synthetic data as stated in Research Question 1, we needed to train the image segmentation networks both with and without pretraining using the pretraining dataset presented in Section 3.2. However, since the images in the collected video contain a red hue, the color difference between the drone videos and the pretraining dataset might be severe enough to influence the effect of pretraining. Therefore, we also need to test the pretraining using images that have been processed by the color restoration technique presented in Section 3.3.

This leaves us with four possible training setups that have to be tested for each of the three segmentation networks:

1. Trained with pretraining and then on color restored images
2. Trained with pretraining and then on red images
3. No pretraining, but trained only on color restored images
4. No pretraining, but trained only on red images

This way, we can explore both the effect of pretraining and whether the color discrepancy between the two datasets affects the pretraining, as stated in research question 2.

Average Precision for Obstacles

It would be useful to have a metric that captures to which extent the models manages to detect the different types of obstacles. The AP is well suited for this purpose, as explained in Section 2.3.1. In order to do this, we need a mask for each type of object. However, the presented segmentation networks are trained for single-class, binary semantic segmentation. Hence, we cannot calculate the AP directly. On the other hand, we can go from single to multi-class if we assume that the model correctly predicts the underlying class.

The way in which we do this is displayed in Table 3.6. When a pixel is predicted to be landable area, this also becomes the final prediction. However, if it is predicted to constitute a part of an obstacle, the final prediction for that pixel depends on the ground truth. If the ground truth is an obstacle, the final prediction is that obstacle. Otherwise, the final prediction becomes the closest obstacle to that pixel. The closest obstacle is found by breadth-first search starting from the *logical and* of predicted obstacles and the ground truth obstacles. This means that there might be islands in the prediction of obstacles where no part of the island overlaps a ground truth obstacle. These islands are set to be forests, as this is by far the most common obstacle. The effect of this might be that the forest AP is impaired to some degree. The effect of this might be that the forest AP is impaired to some degree. Furthermore, as we assume that the model always correctly predicts the underlying class, this metric may have a tendency to overestimate the performance of the models.

	Obstacle of category X	Landable area
Obstacle	Predicted to be category X	Predicted to be the closest category
Landable area	Predicted to be landable area	Predicted to be landable area

Table 3.6: Converting single class semantic segmentation to multi class.

Figure 3.14 shows the result of this transformation. From the rightmost and second rightmost images, we see that most of the pixels fall into the upper left or lower right category in Table 3.6. The upper part of the building is not a part of the ground truth obstacle, but it is predicted as such, hence it falls into the upper right category in the table. In the ground truth image, there are also some trees in the lower-left corner that are missing in the prediction. These areas go into the lower-left category in the table. There are no islands in the image.



Figure 3.14: Converting a single class prediction to multi class.

3.6.3 Segmentation-based SLAD System

We will now combine the systems presented in this chapter to form the segmentation-based SLAD system. The system follows the pipeline illustrated in Figure 3.15.

Firstly, the images might be processed by the color restoration module presented in Section 3.3. However, this step is optional. The image is then given to one of the segmentation networks presented in Section 3.6.1. This network is trained to recognize many types of obstacles, and will produce a binary mask showing the regions of the image that it considers to be suitable (black) and unsuitable (white) for landing. However, these regions need to be translated into specific points. Therefore, this mask is given to the rally point converter presented in Section 3.4. The converter finds the points which are the furthest away from obstacles, and therefore the most suitable landing areas for the given image. The pixel coordinates of the safe landing areas are then given to the pixel geolocator presented in Section 3.5, which converts these into GPS coordinates. The GPS coordinates can then be given to the flight controller of the drone, so that the safe landing areas can be used as rally points.

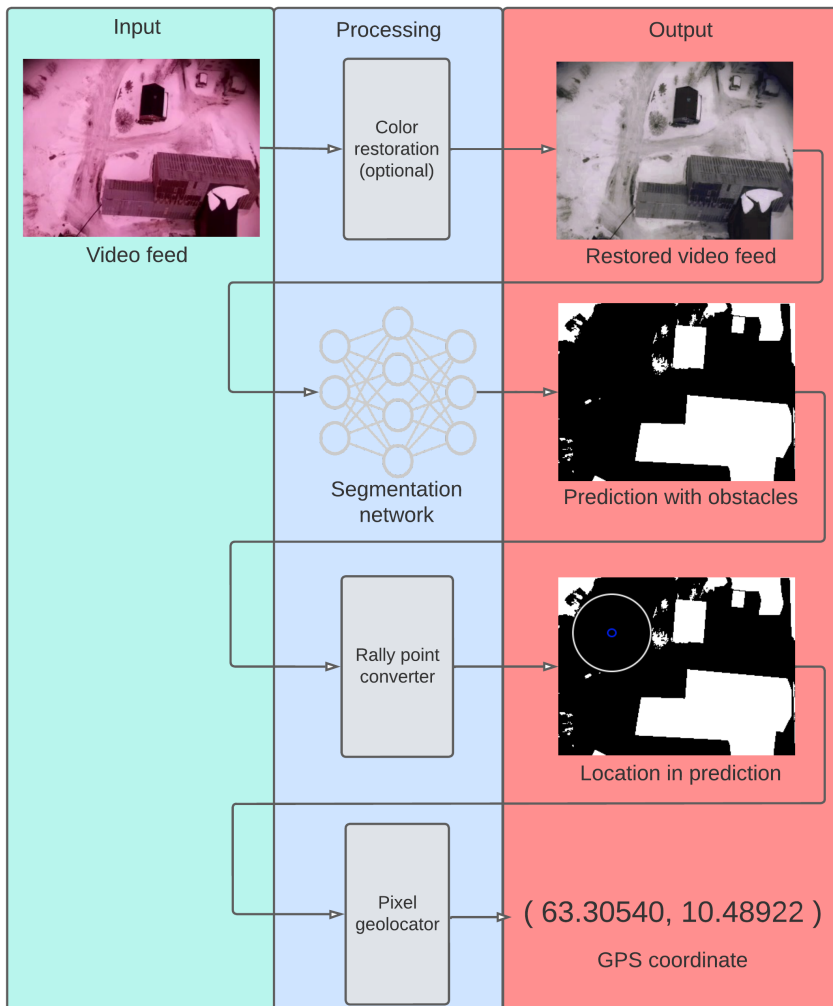


Figure 3.15: An overview of the SLAD pipeline using the segmentation-based approach.

3.7 Rally Point Verification

By combining the two first modules in Figure 3.15, color restoration and the segmentation network, with the coordinate to pixel mapping from Section 3.5.1, we

can automatically verify or refute rally points in the video feed when the drone flies over them.

This is achieved by mapping several points that are in the vicinity of the rally point to their respective locations in the image. We can then proceed to take a *logical and* of this mask and the prediction from the segmentation network, then take the sum of the result. If the sum is greater than zero, the Segmentation network has predicted that there is at least one obstacle in the immediate vicinity of the rally point. Consequently, it is classified as unsafe.

As the drone flies over the rally point, we get several images of the same rally point from different angles. Naturally, when the drone is right above the rally point, it would have a better view of its surrounding area than if it were far away. This means that we need a way of combining classifications of the same rally point from different images for a final classification. We choose the simplest way of combining these results, which is to say that if the rally point is verified in at least one image, then the final classification is that it is verified. This method is undoubtedly prone to bad segmentation predictions on just a single image. More sophisticated methods that do not have this weakness could be considered. However, this weakness can also be mitigated by increasing the required radius around the rally point.

A rally point is classified as either verified, refuted, partially spotted or out of frame. Given a clearance radius, r , a rally point is said to be verified if the entire clearance area is present in an image and simultaneously predicted safe by the segmentation network. On the other hand, if the entire clearance area is present in at least one image, but the segmentation network never predicts the entire area to be safe, the rally point is refuted. If the rally point is present in at least one image, but the entire clearance area is not, the point is considered to only be partially spotted. No prediction from the segmentation network is used, as the view that the drone has over the area is too bad to reliably verify or refute the rally point. The remaining rally points, that are not present in any image, are classified as out of frame.

3.8 Object Detection

The other technique we will use to extract information from the images is object detection. Object detection might be a suitable technique for safe landing area detection because these networks are created to recognize specific areas in images, typically certain types of objects. This resembles the aim of a safe landing area detection system, in which we want to pinpoint specific areas in an aerial image

that are considered to be safe. This differs from image segmentation, which typically outputs more general knowledge about the overall contents of an image.

3.8.1 Tested Architectures

Since Swin-Transformer and ConvNeXt are considered to be state-of-the-art methods also within object detection, we choose to also use these for our object detection-based SLAD system. This way, we get a more direct comparison without any major architectural differences. However, since Unet is an architecture solely used for image segmentation, we also included the well-known object detection architecture called YOLO. This way, we can compare the segmentation-based and object detection-based systems using two similar architectures (Swin-T, ConvNeXt), and one architecture that is unique to each of the techniques (Unet, YOLO).

YOLO [Redmon et al., 2015] was chosen as it is one of the primary methods within the field of object detection. Since its introduction in 2015, YOLO has been one of the most implemented object detection methods according to Papers With Code [Papers with code, 2018]. According to Google Scholar [Google Scholar, 2004], the original YOLO paper has been cited over 24000 times.

3.8.2 Training Setup

Computers

The training and testing of the object detection architectures were performed on the same computers stated in Section 3.6.2.

Ground Truth

The ground truth target points for the object detection networks were generated using the method outlined in Section 3.4: Converting segmentation to rally points. This method utilizes the labeled frames from Section 3.1.2: Data Labeling to find the best landing points. Essentially, we are here trying to train the task performed by the rally point converter in the segmentation-based SLAD system into the object detection network.

Frameworks

For the object detection versions of Swin-T and ConvNeXt, we used the framework MMDetection [MMDetection, 2018]. As explained in Section 2.2, many versions of YOLO exists. In this thesis, we used the YOLOv5 [YOLOv5, 2020] framework. This framework uses the base YOLOv4 [Bochkovskiy et al., 2020]

architecture with some enhancements provided by the open-source community. Both MMDetection and YOLOv5 use PyTorch [PyTorch, 2019].

Pretraining

When it comes to exploring whether pretraining on our pretraining dataset is beneficial, the same situation as explained in Section 3.6.2 applies. We therefore need to train four combinations for each of our object detection architectures, similarly to our training setup for the image segmentation networks.

Avoidance Score

To measure how well different models avoid the different types of obstacles in the dataset, we implemented a custom metric. The metric captures to which extent the models tend to avoid an object type when making predictions. It goes from 0 up to 100, where lower is better. To calculate the score for a certain type of object, we loop through the images where that type of object is present. For each image, we count the number of bad predictions that are caused by this object type. A prediction is considered to be bad if more than half of an instance of that object type is inside the predicted landing area, or if the object is closer than half the radius of the predicted landing area to the center of the predicted landing area. The number of bad predictions is then divided by the total number of predictions in those images. The Avoidance score is therefore the portion of predictions that are considered bad due to a type of object.

3.8.3 Object Detection-Based SLAD System

We will now combine some of the systems presented earlier in this chapter to form the object detection-based SLAD system. The system follows the pipeline illustrated in Figure 3.16.

Similarly to the segmentation-based system in Section 3.6.3, the images might firstly be processed by the color restoration system presented in Section 3.3. However, this is optional. The image is then given to the object detection network, which is trained to detect suitable landing areas in the image. The point coordinates of the detected landing areas are then given to the pixel geolocator explained in Section 3.5, which converts these into GPS coordinates. The GPS coordinates can then be given to the flight controller of the drone, so that the safe landing areas can be used as rally points.

In contrast to the segmentation-based system in Section 3.6, the approach using object detection requires fewer parts. This is because the neural network itself

returns a position in the video frame, which then directly can be converted into a GPS coordinate.

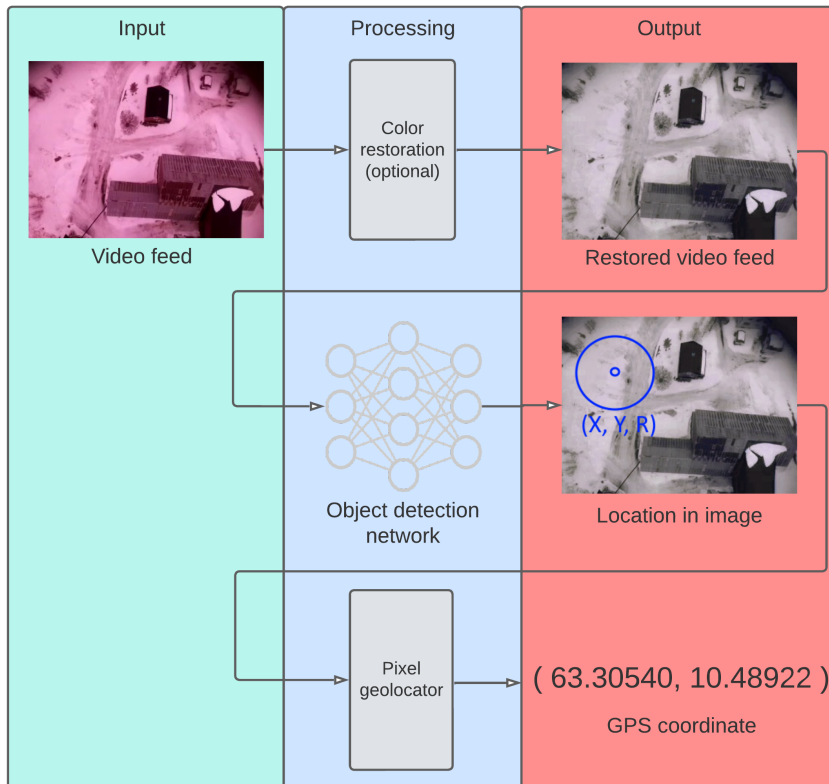


Figure 3.16: An overview of the SLAD pipeline using the object detection-based approach.

Chapter 4

Results

In this chapter, we will go through the results from each of the subsystems of the SLAD-system architectures. We will only state the results here, and provide discussions in Chapter 5.

4.1 Color Restoration using Principal Component Analysis

It is difficult to find a quantitative measure of the performance of the PCA color mapping as there are no ground truth images to compare with. Hence, the results of the color mapper is evaluated using a qualitative comparison. To save space in this thesis, we show a few examples in Figure 4.1, and provide a selection of 32 images in Appendix C.

Notice the artifacts in Figure 4.1c). Of the images in Appendix C, Figure 2a) and Figure 3f), g), h), i), k) and n) contain some artifacts of various degree.

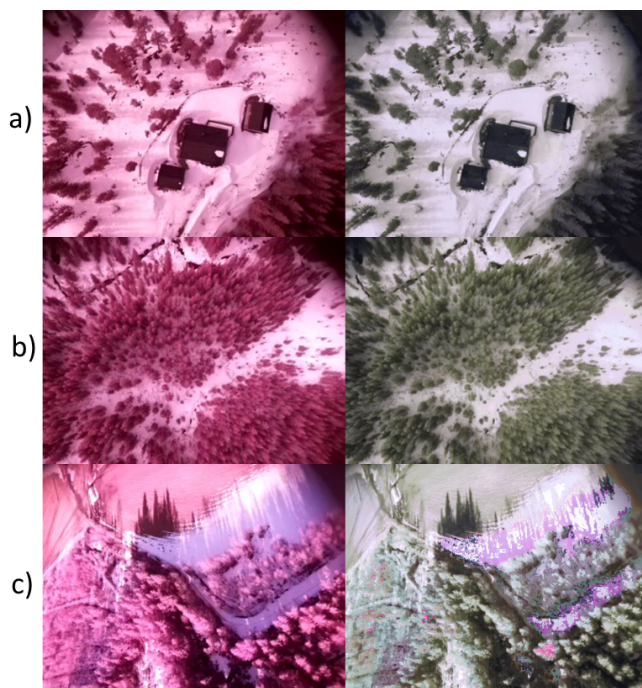


Figure 4.1: The left column shows some of the collected images, while the right column shows the images after the colors have been restored using the color mapper. More examples are provided in Appendix C.

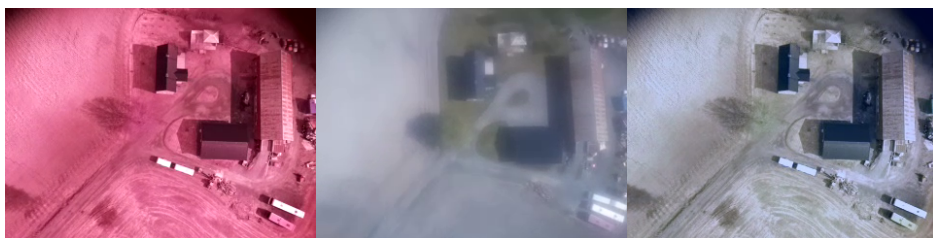


Figure 4.2: Example of one of the images used to calibrate the color mapper. The leftmost image shows the captured video with color distortion, the center image shows the captured color video with fog, and the rightmost image shows the first image after the PCA color mapper was calibrated. All of the reference images used for PCA calibration are provided in Appendix B.

4.2 Pixel Geocator

The accuracy of the pixel geocator was measured by comparing the in-video output of the system to the actual GPS location. Firstly, the GPS-locations of 65 predetermined rally points were plotted into a video using the pixel geocator. Then, the GPS positions were plotted in *Google Maps* [Google Maps, 2005] and *Norge i bilder* [Norge i bilder, 2005], and compared to the resulting output in the video. The difference between the points was then measured using the built-in distance tool of the two map services, in addition to using an online coordinate distance calculator [Boulter, 2004]. An overview of the deviation between the two points is shown in Table 4.1. The average of these deviations ended up being $9.95m$. Figure 4.3 shows the points from the table with the distance between them.

Num	Actual point	Output point	Deviation (m)
1	62.93188, 11.19731	62.93187, 11.19733	2.0
2	62.94357, 11.21108	62.94356, 11.21120	6.0
3	62.94745, 11.20906	62.94730, 11.20890	18.5
4	62.96144, 11.13444	62.96149, 11.13445	5.3
5	62.97301, 11.12270	62.97286, 11.12287	18.5
6	62.98753, 11.10556	62.98756, 11.10580	12.5
7	62.99246, 11.10019	62.99248, 11.10034	7.5
8	63.02604, 11.07394	63.02602, 11.07402	5.0
9	63.02973, 11.07134	63.02966, 11.07162	16.2
10	63.03814, 11.06302	63.03810, 11.06314	7.0
11	63.04822, 11.04285	63.04832, 11.04297	11.0
12	63.07014, 11.01214	63.07014, 11.01238	11.0
13	63.09375, 10.98386	63.09369, 10.98400	10.0
14	63.09688, 10.97554	63.09685, 10.97571	8.0
15	63.10473, 10.95232	63.10464, 10.95252	13.5
16	63.11237, 10.90525	63.11238, 10.90538	6.0
17	63.13348, 10.87976	63.13341, 10.87989	10.0
18	63.23129, 10.70259	63.23124, 10.70260	5.0
19	63.32198, 10.50547	63.32185, 10.50566	16.0
20	63.33749, 10.38170	63.33742, 10.38177	10.0
Avg			9.95
SD			4.74

Table 4.1: Overview of the accuracy of the pixel geocator with average accuracy and standard deviation. The points are shown in Figure 4.3.



Figure 4.3: An overview of the accuracy measuring of the pixel geolocator. The point coordinate written in each image is the location the system claims the blue point is in. The red point is the actual location of the coordinate. The points are listed in Table 4.1.

4.3 Segmentation

Here we will provide the results from the segmentation networks. An overview of all the models with associated information is provided in Table 4.2. Table 4.3 provides the results from the generated masks from the segmentation networks. This table therefore provides information about the general mask predictions of each model. Note that this is before specific landing areas are extracted from them. To view the mask predictions from each model, see Appedix D.

Table 4.4 shows the results for the actual landing areas after being extracted from the aforementioned masks. It is therefore the contents of this table that is directly comparable to the results of the object detection-based system presented in Section 4.7. To view examples of the outputs of the segmentation-based system, see Appendix F.

Notice that the AP LA scores in Table 4.3 and 4.4 are not the same. The AP LA scores in Table 4.3 consider the average precision of the whole mask. The AP LA scores in Table 4.4 consider the average precision of the circular landing areas.

ID	Backbone	Framework	Pretrained	Image color
Seg_Unet_Res_P	mobilenet_v2	Unet	Y	Restored
Seg_Unet_Red_P	mobilenet_v2	Unet	Y	Red
Seg_Unet_Res	mobilenet_v2	Unet		Restored
Seg_Unet_Red	mobilenet_v2	Unet		Red
Seg_SwinT_Res_P	Swin-T	UperNet	Y	Restored
Seg_SwinT_Red_P	Swin-T	UperNet	Y	Red
Seg_SwinT_Res	Swin-T	UperNet		Restored
Seg_SwinT_Red	Swin-T	UperNet		Red
Seg_CNX_Res_P	ConvNeXt-T	UperNet	Y	Restored
Seg_CNX_Red_P	ConvNeXt-T	UperNet	Y	Red
Seg_CNX_Res	ConvNeXt-T	UperNet		Restored
Seg_CNX_Red	ConvNeXt-T	UperNet		Red

Table 4.2: Ids for segmentation models.

ID	F1	mAP	AP People	AP Power Line	AP Road	AP Vehicle	AP Water	AP Building	AP Tree	AP Other	AP LA
Seg_Unet_Res_P	91.8	65.6	0.035	24.2	62.6	94.5	86.3	94.6	87.1	43.8	97.6
Seg_Unet_Red_P	93.7	60.0	0.035	20.0	59.2	86.4	76.6	92.5	68.7	41.0	95.9
Seg_Unet_Res	96.0	62.6	0.035	22.3	59.8	92.0	81.3	88.5	84.1	37.5	98.3
Seg_Unet_Red	92.7	63.6	0.035	23.2	64.5	92.9	81.4	89.8	87.6	35.1	98.2
Seg_SwinT_Res_P	92.6	62.8	0.054	22.3	61.9	90.4	87.3	85.1	82.8	39.0	96.1
Seg_SwinT_Red_P	89.3	62.9	0.035	22.5	58.8	94.1	88.3	86.1	83.0	37.4	96.3
Seg_SwinT_Res	93.5	61.9	0.035	22.6	56.7	92.3	88.1	86.3	83.0	32.1	96.4
Seg_SwinT_Red	92.0	60.6	0.035	21.7	57.2	86.2	86.3	85.2	82.1	30.7	95.8
Seg_CNX_Res_P	92.3	64.8	0.035	22.5	62.1	94.3	86.2	86.9	84.0	50.6	96.6
Seg_CNX_Red_P	93.2	63.8	0.054	22.2	60.0	92.4	86.1	86.4	83.5	47.7	96.1
Seg_CNX_Res	93.6	64.5	0.035	22.0	61.9	91.7	86.1	88.7	83.7	50.7	96.1
Seg_CNX_Red	91.3	63.7	0.334	22.1	59.9	92.2	84.6	87.7	83.1	47.3	96.0

Table 4.3: Results from the mask output of the segmentation networks. The listed results indicate Average Precision (AP).

ID	AP LA	mAS	AS People	AS Power Line	AS Road	AS Vehicle	AS Water	AS Building	AS Tree	AS Other
Seg_Unet_Res_P	82.2	17.0	71.1	26.0	2.2	0	5.5	0	30.8	0
Seg_Unet_Red_P	79.6	16.2	65.5	32.2	2.0	0	0.2	0	29.3	0
Seg_Unet_Res	81.6	18.1	71.1	30.6	0	0	3.6	0	39.4	0
Seg_Unet_Red	81.8	16.6	69.4	32.0	0	0	0.9	0	30.2	0
Seg_SwinT_Res_P	85.6	19.4	71.1	36.3	0	0	4.2	0	43.8	0
Seg_SwinT_Red_P	81.2	20.2	71.1	36.0	0	0	0	0	54.6	0
Seg_SwinT_Res	84.1	20.7	71.1	36.4	0	0	3.0	0	54.9	0
Seg_SwinT_Red	81.3	20.0	71.1	36.0	0	0	0	0	52.8	0
Seg_CNX_Res_P	87.2	21.1	71.1	37.9	6.5	0	0	0	53.6	0
Seg_CNX_Red_P	81.0	21.6	71.1	38.9	6.5	0	0.1	0	56.5	0
Seg_CNX_Res	86.4	20.4	71.1	35.9	0.3	0	0.5	0	55.0	0
Seg_CNX_Red	78.9	21.6	71.1	36.2	2.5	0	0.1	0	63.0	0

Table 4.4: Results from the landing area outputs of the segmentation-based SLAD system. The results listed are Avoidance Scores (AS) and Average Precision of the Landing Areas (AP LA).

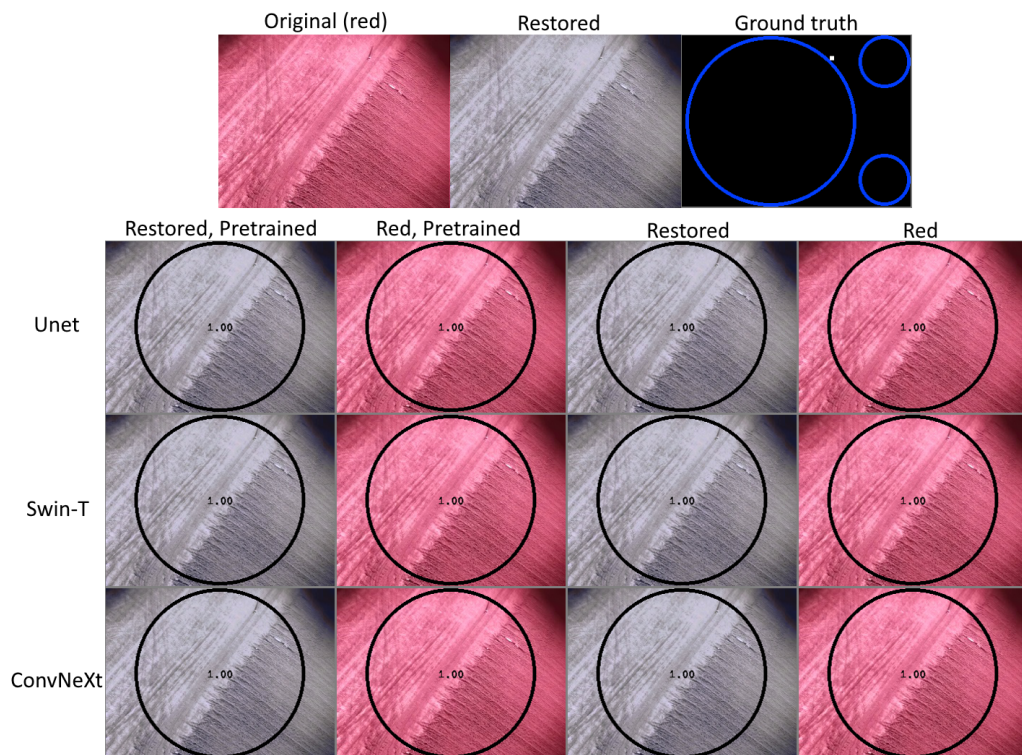


Figure 4.4: Single-point predictions from the segmentation-based SLAD system. Notice that there is a person in the upper part of the image, which is marked by a white mask in the ground truth. None of the models manages to recognize the person. More predictions from this system are provided in Appendix F.

4.4 Rally Point Verification

The rally point verification was tested on a flight by *NT02* from Haltdalen to Sandmoen on April 1st. In total, there were 114 rally points in the mission. Table 4.5 lists the results from the rally point verification system. Out of all the rally points in the mission, 50 were not present in any image when mapped by the pixel geolocator. Furthermore, 6 and 14 points were only partially spotted when the circle of interest was 10m and 12m, respectively. We see that out of the remaining rally points that could be verified, the 10m threshold lead to a recall of 67.2%, while 20m gave a recall of 36%.

From Figure 4.5 below we see a selected number of verifications and refutations of

Clearance Radius	Recall	Verified	Refuted	Partially Spotted	Out of Frame	Total
10m	67.2	39	19	6	50	114
20m	36	18	32	14	50	114

Table 4.5: Recall for rally point verification.

rally points by using **Seg_Unet_Res_P**. The figure shows the different combinations of classifications that are present for the video used for verification. When a rally point is verified or partially spotted, the earliest verification or partial spotting is displayed. If the rally point is refuted, the last frame of refutation is shown. In 4.5a), both thresholds verify the rally point, however the 10m radius manages to verify the rally point earlier as the entire area of the circle is inside the frame before the 20m. We can see this as the rally point is closer to the top in the 10m case than in the 20m case. For 4.5b), the 20m radius prevents the entire area to be inside the image, but for the 10m radius it is inside and can therefore be verified.

In 4.5c), circles for both distances are completely inside the frame, and the 10m radius requirement verifies the point. However, the 20m radius refutes it because of the trees within this threshold. In 4.5d) neither distance is ever sufficiently within the image frame to evaluate the area. For 4.5e), the obstacles within the 10m radius leads to the point being refuted. In the 20m case, the entire area is never completely inside the image. In 4.5f), both radii refutes the rally point. Note that there are no instances of partially spotted-verified, partially spotted-refuted and refuted-verified for 10m-20m as this is impossible. Additional rally points that are either verified, refuted or partially spotted are depicted in Appendix G.



Figure 4.5: An overview of predictions from the rally point verification system. The leftmost column shows the hand-picked rally points. The center column shows the points covered by a circle with a radius of 10m. The rightmost column has circles with a radius of 20m. The points are evaluated based on the area of the circle. If the circle is green, the rally point is considered to be safe. If it is red, it means the point is considered unsafe. Yellow means that the entire circle is never completely inside any images in the video, and can therefore not be evaluated.

4.5 Object Detection

Here we will provide the results from the object detection-based SLAD system. An overview of all the models with associated information is provided in Table 4.6. Table 4.7 contains the avoidance scores for different categories, in addition to the average precision of the predicted landing areas (AP LA). Table 4.8 shows some of the best performing segmentation and object detection models, and the sizes of the objects they fail to detect. To see the predictions from the object detection models, see Appendix E.

ID	Backbone	Framework	Pretrained	Image color
OD_Yolo_Res_P	New CSP-Darknet53	YOLOv5s	Y	Restored
OD_Yolo_Red_P	New CSP-Darknet53	YOLOv5s	Y	Red
OD_Yolo_Res	New CSP-Darknet53	YOLOv5s		Restored
OD_Yolo_Red	New CSP-Darknet53	YOLOv5s		Red
OD_SwinT_Res_P	Swin-T	Mask R-CNN	Y	Restored
OD_SwinT_Red_P	Swin-T	Mask R-CNN	Y	Red
OD_SwinT_Res	Swin-T	Mask R-CNN		Restored
OD_SwinT_Red	Swin-T	Mask R-CNN		Red
OD_CNX_Res_P	ConvNeXt-T	Cascade Mask R-CNN	Y	Restored
OD_CNX_Red_P	ConvNeXt-T	Cascade Mask R-CNN	Y	Red
OD_CNX_Res	ConvNeXt-T	Cascade Mask R-CNN		Restored
OD_CNX_Red	ConvNeXt-T	Cascade Mask R-CNN		Red

Table 4.6: Ids for object detection models.

ID	AP LA	mAS	AS People	AS Power Line	AS Road	AS Vehicle	AS Water	AS Building	AS Trees	AS Other
OD_Yolo_Res_P	82.7	22.5	73.7	45.0	0	0	2.6	0	58.6	0
OD_Yolo_Red_P	77.8	22.1	67.1	46.6	0	0	8.4	0	49.5	0
OD_Yolo_Res	80.7	20.5	66.4	37.5	0	0	4.3	0	55.9	0
OD_Yolo_Red	79.5	20.7	69.7	38.9	0	0	3.1	0	53.7	0
OD_SwinT_Res_P	82.4	16.6	80.2	37.7	0	0	5.0	0	10.0	0
OD_SwinT_Red_P	82.0	16.3	74.0	36.9	0	0	6.0	0	13.8	0
OD_SwinT_Res	70.3	15.0	55.1	44.7	0	0	5.5	0	14.5	0
OD_SwinT_Red	69.3	15.2	55.1	36.0	0	0	10.3	0	20.4	0
OD_CNX_Res_P	70.9	15.2	37.5	42.3	4.5	0	13.1	0	24.5	0
OD_CNX_Red_P	72.2	16.6	48.1	42.4	4.1	0	12.4	0	25.7	0
OD_CNX_Res	70.6	16.9	63.6	38.6	0	0	10.4	0	22.9	0
OD_CNX_Red	72.4	12.6	21.1	35.6	7.2	0	12.6	0	24.0	0

Table 4.7: Results from the landing area outputs of the object detection-based SLAD system. The results listed are Avoidance Scores (AS) and Average Precision of the Landing Areas (AP LA).

ID	1 – 4	5 – 16	17 – 64	65 – 256	257 – 1024	1025 – 4096	4097 – 16384	≥ 16385
Seg_Unet_Res_P	79.1	6.9	2.9	1.8	0.7	0.6	5.7	2.3
Seg_CNX_Res_P	89.7	3.9	1.1	0.0	0.8	0.0	3.1	1.5
OD_Yolo_Res_P	76.4	13.2	5.7	0.9	0.4	0.0	2.6	0.7
OD_SwinT_Res_P	75.4	9.8	4.1	1.3	0.5	0.1	5.3	3.6

Table 4.8: Sizes of missed obstacles.

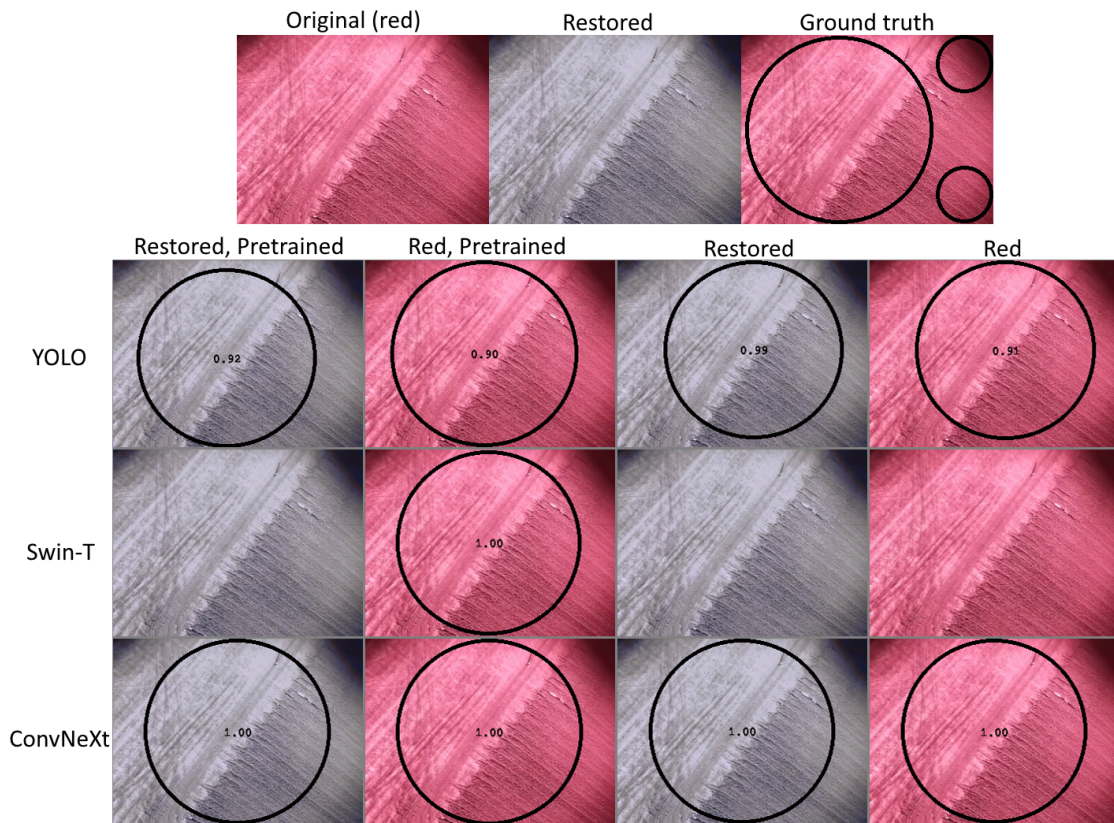


Figure 4.6: Predictions from the object detection-based SLAD system. Notice that there is a person in the upper part of the image, which is causing the ground truth areas to be split into three distinct points. None of the models manages to recognize the person. More examples of predictions from this system are provided in Appendix E.

Chapter 5

Discussion

In this chapter, we will discuss the results presented in Chapter 4 and relate them to the research questions presented in Section 1.2. In addition, we will address strengths and weaknesses in the technical systems and the performed research.

5.1 Drone Dataset

5.1.1 Data Collection

The data collection setup described in Section 3.1.1 proved to be a reliable and efficient way to collect videos from the drones without any noticeable performance overhead during flight.

Since we were well within the limit of taking up too much storage space, the resolution of the videos could have been increased. In this thesis, the data collection system was connected to a v4l2rtsp server. However, it is possible to set up GStreamer [GStreamer, 2004] instead. GStreamer is able to have multiple streams with different resolutions. The result of this would have been that we could have had a higher resolution on the collected data, while the video stream to QGC had the same resolution. This would have enabled us to capture more detailed videos. However, it could have interrupted the stream to Aviant’s operators. This is because the stream with the lower quality would have to be compressed in real-time, and the delay to QGC could increase. Still, this is something that could have been explored.

The red hue in videos from *NT02* was believed to be due to a bad pin connector. This is unfortunate, as a large portion of the collected videos was from this drone. Attempts to check if this problem was caused by software issues were unsuccessful. No comprehensive hardware debugging was attempted. However, measures to alleviate this issue on videos that were already captured were prioritized, as elaborated on in Section 3.3.

5.1.2 Data Labeling

By merging manual and automatic labeling, the strengths of both approaches were combined to create accurate ground truths. Manual labeling is the classical way to generate ground truth labels. It was time-consuming, but yielded accurate masks on objects that could not be labeled automatically. The automatic labeling approach was a very efficient way to label trees. The quality and quantity of the masks generated by this approach are impossible to achieve with manual labeling due to time limitations. Consequently, the combinations of these two approaches enabled the creation of such a large dataset.

The automatic labeling system is very simple, yet effective. However, the consequence of this is that in its simplicity, minor details are ignored in favor of the general concept that *darker areas are obstacles*. An example of this is shown in Figure 6a) in Appendix D. We see that, as the tree tops are so bright due to the illumination from the sun, they are considered to be safe areas to land, which is clearly wrong. Furthermore, shadows are sometimes considered as obstacles due to their darker appearance. This is evidently not as bad as the former weakness, as it is better to miss a safe landing area than it is to land on an unsafe area. Still, this is a weakness in the labeling that needs to be improved.

5.1.3 Data Partitioning

The collected drone dataset contains representative features and is of adequate size to train deep learning models. Furthermore, it is successfully split into three disjoint sets in a way that prevents data leakage. However, when we look at the dataset as a whole, it is unfortunately dominated by the winter season. This is because of the timing of the data collection period. This may be a limiting factor to the generalizability of the results from this dataset.

5.2 Pretraining Datasets

Here we will discuss the dataset used for pretraining. We will also lay the groundwork for addressing Research Question 1: *Can synthetic data be beneficial in generating rare situations when trying to train a deep learning-based system for safe landing area detection?*

5.2.1 Potsdam Dataset

The intention behind the inclusion of the Potsdam dataset was to get a more diverse selection of buildings, in addition to more examples of cars. It also provides some more examples of *clutter*, which are regarded as other obstacles in our case.

There is a weakness regarding the Potsdam dataset. This is, that the area in the dataset that is considered to be landable might contain obstacles. This is because the original dataset had *low vegetation* labeled as its own class. As stated in Section 3.2, the area comprising low vegetation was designated as safe. This was done because this label mainly covered flat, green areas. However, this label also covered a small amount of bushes and low vegetation that would not be suitable for landing. This presumably did not have a significant impact on training, as there were not many instances of this. Taller vegetation was covered by the tree label, which further constrained the issue.

Other missing labels in the Potsdam dataset are roads, people and power lines. The absence of these masks is not considered to be a big issue, as the intention behind the inclusion of this dataset was to get more training data of cars and buildings. However, manual labeling of these missing features could have been beneficial to get extra training samples, especially of power lines, as there are so few examples of this in the drone dataset.

5.2.2 Unreal Engine 4 Dataset

The main purpose of the Unreal Engine 4 Dataset is to compensate for the lack of people in the Drone Dataset and the Potsdam Dataset. It also gives a more nuanced labeling of trees than what is provided by the automatic labeling in the Drone Dataset, as the labeling is pixel perfect and not dependent on brightness, but rather the actual placement of the object in the scene.

However, there are certain weaknesses regarding the people in this synthetic dataset, for instance with the selection of people-models. Even though the models constitute a diverse selection of people, the breadth of people can always be better. Furthermore, there is also an issue with the pose of the people. All the

models have an unnatural pose, standing with their hands out. This reduces the resemblance between people in the synthetic dataset and people in real-life situations, which have all sorts of poses when they are outside.

There is also an issue with the roads in the dataset. They could not be added to the mask due to a limitation in Unreal Engine 4 where they are a part of the ground texture. They could therefore not be individually isolated. In Table 3.5, the 74 frames in the dataset that contains roads are therefore depicting intersections and bridges, that are not a part of the ground texture.

Another issue could be with the height at which the images were taken. Some images were taken at very high altitudes, making the people in the frame very small. In certain cases, each person only constituted a few pixels of the image. Even though three height levels were set to try to mitigate this, as stated in Section 3.2, the images that were taken at the highest altitudes still suffer from this. Therefore, it might have been beneficial to generate images only at low to medium heights. This way, the dataset would be more specialized towards its original purpose of helping the systems detect people. In addition, it would alleviate the risk of confusing the neural networks, as it can be hard for them to differentiate between people and features of the terrain in images taken at high altitudes.

5.2.3 Answering Research Question 1

The Potsdam Dataset and the Unreal Engine 4 Dataset lay the foundations for answering Research Question 1. The quality and diversity in these datasets are essential in order to successfully learn concepts that also occur in the Drone Dataset, but to a significantly less extent. Furthermore, this combined pretraining dataset is relatively balanced. People, vehicles and buildings appear almost just as frequently. This is a desirable property. However, to fully answer the Research Question, we first need to discuss the results from the segmentation and object detection-based SLAD systems. We will therefore revisit this question in Section 5.8.3.

5.3 Color Restoration using Principal Component Analysis

Here we will discuss Research Question 2: *How can images with distorted colors be restored to their original colors, and will this improve the effect of pretraining the deep learning models?*

5.3.1 Discussion of Results

It is hard to create a suitable qualitative measure of the quality of the color mapping technique, as stated in Section 4.1. Therefore, a quantitative analysis is performed. As we see from Figure 4.1, the color mapping has certainly been effective, and the restored images contain more realistic colors than the raw images with red hue. However, there are still slight differences between colors in the blurry images and the restored images. An example of this can be seen in Figure 4.2, which depicts one of the images used to calibrate the color mapper. Here we can see the color differences of the grass and gravel in the blurry color image and the restored image, even though this is a reference image. This is because there is information loss in the red images, as stated by assumption 3 in Section 3.3. Therefore, it is not possible to create a perfect mapping. Considering this, we view this difference to be acceptable, and the results have exceeded the authors' expectations.

Ideally, one would not require techniques like color restoration as the collected data would provide video of adequate color accuracy to give a good mapping between collected and already existing data. In addition, any technique that alters the colors or quality of imagery has the potential of creating defects. An example of this is the artifacts created by our color-mapping technique, shown in Figure 4.1c). The artifacts are not that serious, and do not make the images useless as features in the images are still clearly visible. However, they are significant enough to potentially disturb deep-learning networks.

5.3.2 Answering Research Question 2

The color restoration technique proposed in this thesis constitutes the first step in Research Question 2. The images with restored colors are conspicuously better than their red counterparts. Despite this, the restoration process also introduces artifacts, in addition to not covering the entire domain of all the colors in the Drone Dataset. The effect this has on the neural networks is discussed in 5.6.2

and 5.8.2 for Segmentation and Object Detection, respectively. We will provide the final conclusion of RQ2 in Section 5.8.3, after going through these discussion sections.

5.4 Converting Segmentation to Rally Points

Using Dijkstra’s algorithm to go from segmentation to object detection proved to be very fast for an image of size 320×240 . Hence, when comparing the performance of segmentation-based approaches to object detection-based approaches, the performance overhead of converting the segmentation predictions can almost be neglected.

When finding the best rally points from a segmentation mask, the area that each pixel represents is assumed to be equal for all pixels in the image. If the drone has a flat surface underneath and a roll and pitch of both 0° , the outermost pixels of the image would cover more area than the central ones. So even in the simplest case, this assumption proves false. The shortcomings of this assumption are especially apparent when the terrain is rugged or the drone is in the middle of a turn. The second image in Figure 4.3 depicts the latter case. However, this is an assumption that simplifies and speeds up the calculations. The alternative is to use the pixel geolocator, which would likely yield a result closer to the real-world distances between obstacles. However, as the average error for geolocation is about 10m, the resulting depth map would not be perfect.

This module does not distinguish between different obstacles in the image, it considers an obstacle to be just that, an obstacle. This might be a fair assumption as we do not want to collide into anything. However, some obstacles are especially important to avoid. For instance, it is much worse to injure a person than it is to hit a tree. Additionally, persons and vehicles are obstacles that might move when the drone tries to land, while obstacles like trees, houses and roads do not. This means that one could consider weighing distances to obstacles when determining the best places to land, as some obstacles might move or are less desirable to collide into.

5.5 Pixel Geolocator

Here we will discuss Research Question 3: *How and with what accuracy can detected landing areas be converted into geolocation coordinates?* and the ability of the associated system: the pixel geolocator.

For SLAD systems using monocular video, converting pixels to geolocation coordinates with high accuracy is especially important. This is because the whole SLAD system is dependent on a correct and accurate conversion. If the system finds the best landing area in the image, it is of no use if the location of this area is altered by a subsequent system.

When locating a point from a pixel, there are errors in every sensor and measurement. The intrinsic rotations, the GPS coordinates from the flight controller, the field of view of the camera and synchronization between the video stream and the clock of the flight controller are the sources of error that are present when locating a point from a pixel or vice versa. As we can see in Table 4.1, the result of this is that the pixel geolocator has an average deviation of 9.95m and an empirical standard deviation of 4.74m. This might sound like a lot, but this deviation is largely due to the drone being about 100m up in the air, traveling at speeds of about 23m/s, combined with the previously mentioned sources of error. When the drone attempts to land, the height above ground shrinks, and consequently the error should too. This is because a divergence of a certain amount of pixels translates into a shorter distance when the drone is closer to the ground. Furthermore, this makes the argument as of why the landing area needs to be of a certain size, as the estimation of its location is on average 9.95m off.

One could argue that the pixel geolocator not necessarily needs to be that accurate at high altitudes. This is because as the drone lands, the output of the pixel geolocator should become more accurate, and the chosen rally point should therefore correct itself to a more accurate position. The drone could therefore alter its landing area as it descends.

One potential improvement can be seen in Figure 4.3, in which the point from the pixel geolocator (blue) tends to generally be further down and to the right from the actual location (red). Tuning the pixel geolocator by moving the output points closer to the actual points could reduce the average deviation of 9.95m.

5.5.1 Answering Research Question 3

When it comes to Research Question 3, we have found a competent method for converting landing areas to geolocation coordinates. Even though there is room for improvement, the pixel geolocator has been more than sufficient for us to explore our research goal. However, before being ready for real-world use, its accuracy would have to be improved. The potential sources of error mentioned earlier should be tested and addressed to increase the accuracy. Still, the improvements lie in sensor accuracy and tuning of constants. The overall setup and

logic behind the system would still remain the same, and it is this that constitutes the contribution of this thesis.

5.6 Segmentation

Here we will discuss the Segmentation-based SLAD-system presented in Section 3.6.3, and how it performs related to Research Question 1: *Which deep learning-based computer vision technique provides the best results in determining safe landing areas?*. We will also explore whether the synthetic dataset had any effect, as mentioned in Research Question 4: *Can synthetic data be beneficial in generating rare situations when trying to train a deep learning-based system for safe landing area detection?*

5.6.1 Pretraining

The aim of the pretraining was to improve performance on the classes vehicle, building and person. To determine whether pretraining benefited these categories, we can consider their average precision in Table 4.3.

Vehicles

When it comes to detecting vehicles, the models seem to generally benefit from pretraining. For each of the three model-architectures, the best performing model on vehicles is always a pre-trained model:

- Seg_Unet_Res_P: 94.5%
- Seg_SwinT_Red_P: 94.1%
- Seg_CNX_Res_P: 94.3%

The Potsdam dataset, which is included in the pretraining dataset, contains many examples of vehicles, as can be seen in Table 3.4. Therefore, this result is a strong indicator that the models can benefit from pretraining.

Buildings

The observed benefit of pretraining when it comes to detecting vehicles does not apply when it comes to detecting buildings. For each architecture, the best model at detecting buildings are:

- Seg_Unet_Res_P: 94.6%

- Seg_SwinT_Res: 86.3%
- Seg_CNX_Res: 88.7%

Notice that Seg_SwinT_Res and Seg_CNX_Res do not contain a P, meaning they are not pretrained. The reason why pretraining does not unequivocally improve building detection might be because of a mapping problem. The Potsdam dataset, which constitutes the majority of buildings in the pretraining dataset (3277/3622 occurrences, see Table 3.4 and 3.5), contains many buildings such as apartment complexes and dense neighborhoods. These buildings might not be similar enough to the rural houses and huts in the drone dataset. Therefore, the mapping between the buildings in the pretraining dataset and the drone dataset is too divergent for the networks to benefit from pretraining. This theory is strengthened by the fact that pretraining increases the precision of vehicle detection, as vehicles appear the same in the Potsdam dataset and the drone dataset.

However, it must also be stated that the pretrained Unet models, Seg_Unet_Res_P and Seg_Unet_Red_P seems to benefit considerably from pretraining when it comes to detecting buildings. This could indicate that the primary problem lies not in the similarity between the datasets, but rather in the model architectures. Unet contains fewer trainable parameters than Swin-T and ConvNeXt, which might make it more robust to the variations of the buildings in the Potsdam dataset and the drone dataset.

People

The special Unreal Engine dataset was included in the pretraining dataset in order to increase the amount of training data containing people. According to Table 4.3 the average precision for the people category is very low, below 0.334% for every model. The reason for this is that people detection is flawed. The primary issue is that from high altitudes, people are so small that it is hard to recognize that it is a person. Another issue is that because of the limited amount of people in the drone dataset, the test set only includes a single person. As can be seen in Table 3.3, only 152 frames in the test set contain people, and this is the same person in all of those frames. The person is not particularly visible, and it is therefore understandable that the networks do not manage to detect him/her. A single image from the test-set containing the person is shown in Figure 4.4. Even though this image is one of those in which the person is most visible, one can barely see the person. In addition to these issues are those of the Unreal Engine dataset mentioned in Section 5.2.2.

Considering these issues with the detection of people, and the fact that many

of the average precisions in Table 4.3 are repeating, these values are most likely not a result of the network managing to detect the person. They are more likely to be a result of coincidence or randomness.

Specific class trade-off

An interesting observation when studying the results from the segmentation networks is that the models with the best average precision at overall landing area, the *AP LA* column in Table 4.3, tends to be models that are not pretrained. The best models when it comes to average precision of landing area are:

- Seg_Unet_Res: 98.3%
- Seg_SwinT_Res: 96.4%
- Seg_CNX_Res_P: 96.6%

This might indicate that there is a trade-off occurring when applying pretraining. Average precision of the other classes could be sacrificed for increased precision in the specific classes of vehicles and buildings, as these are prevailing in the pretraining dataset. However, this theory is weakened by Seg_CNX_Res_P, as this model is pretrained, and still has the highest average precision when it comes to landing area, in addition to having the best AP score on vehicles among the ConvNeXt models.

5.6.2 Impact of Color Restoration

When it comes to evaluating whether color restoration has an effect on the similarity between the pretraining set and the drone dataset in the context of the segmentation networks, we can look at the difference between the pretrained models. As stated earlier, the pretraining dataset mostly consists of images containing vehicles, buildings and people. Therefore, these are the primary classes that should be considered when evaluating the similarity between the pretraining dataset and the red/restored drone dataset.

The pretrained Unet models seem to benefit considerably from color-restored images. Seg_Unet_Red_P manages an AP of just 86.4% on vehicles, while Seg_Unet_Res_P achieves 94.5%. The AP score of buildings is also improved when using restored images. The same applies to ConvNeXt. Seg_CNX_Red_P achieves an AP of 92.4% on vehicles, while Seg_CNX_Res_P manages to get 94.3%. Similar to Unet, ConvNeXt also benefits from restored images when it comes to the AP score of buildings.

As stated in Section 5.6.1, the best models for detecting vehicles for each architecture are all pretrained. However, one of these models is not trained on restored images. Seg_SwinT_Red_P has an AP of 94.1% on vehicles, which is the highest of all the Swin-T models. Still, this model is trained on the drone dataset with red images. The pretrained Swin-T model trained on restored images (Seg_SwinT_Res_P) achieves an AP of just 90.4% on vehicles. Seg_SwinT_Red_P also has a better AP score on buildings.

If we take the provided examples into consideration, they do not unconditionally indicate that color restoration leads to a better mapping between the pretraining dataset and the drone dataset. However, considering that Swin-T is a unique architecture that includes advanced features such as self-attention, it might be possible that color restoration is generally beneficial, just not in the case of Swin-T. This theory is strengthened by the fact that even though Unet and ConvNeXt have architectural differences, they both benefit from color restoration. We would therefore consider it plausible that the color restoration leads to a better mapping between the pretraining dataset and the drone dataset when it comes to segmentation.

5.6.3 Answering Research Question 1,2 and 4

In Section 5.6.1, we discussed the effect of pretraining the segmentation networks. In this discussion, we explained that there is a weak indication that pretraining might be beneficial for segmentation, especially when it comes to vehicles. To also consider the pretraining results of the object detection models, we will address Research Question 1: *Can synthetic data be beneficial in generating rare situations when trying to train a deep learning-based system for safe landing area detection?* in Section 5.8.3.

In Section 5.6.2, we discussed the impact of color restoration when pretraining the segmentation models. We explained that there is a plausible link between color restoration and the improved effect of pretraining. We therefore take this observation into account, and address Research Question 2: *How can images with distorted colors be restored to their original colors, and will this improve the effect of pretraining the deep learning models?* in Section 5.8.3. This way we can address the research question after having discussed the impact of color restoration when training the object detection networks.

When it comes to choosing which segmentation model that performs best, there is not any clear winner. If we look at the average precision results in Table 4.3, the model which seems to perform the best is Seg_Unet_Res_P. The AP results

in this table tell us about the quality of the segmentations generated by the networks. If we want to look at the rally point outputs of the whole SLAD system, we need to look at Table 4.4. These values tell us about the Avoidance Score, which represents how often there are obstacles in the predicted landing areas. According to this table, Seg_CNX_Res_P seems to be the best model, as this has the best average precision when it comes to landing area (87.2). It has a bit higher mean avoidance score (mAS) than most of the others. This is probably because it predicts landing areas that are a bit larger than they should be. This is also reflected in the relatively high avoidance score in the tree category. However, landing area predictions that are a bit too large do not matter as long as they are in the correct place. The drone will attempt to land in the center of the area, so obstacles in the periphery of the area are not harmful. Since this model has such a high average precision of the landing areas, this indicates that the areas are in fact placed in favorable locations. We will take the observations regarding this model into account, and compare them to the best object detection model in Section 5.8.3. This way, we can answer Research Question 4: *Which deep learning-based computer vision technique provides the best results in determining safe landing areas?*

5.7 Rally Point Verification

Here we are going to discuss the Rally Point Verification system presented in Section 3.7 and how it performs related to Research Question 3: *How effectively can a deep learning-based system for safe landing area detection verify predetermined landing areas?*

5.7.1 Components

The rally point verification system is a combination of the segmentation system and the pixel geolocator. Consequently, strengths and weaknesses in these systems affect the overall performance of the combined system, the rally point verification system.

The tested segmentation architectures perform well and are sufficient to detect most stationary obstacles that are filmed during flight, as discussed in Section 5.6. This insight needs to be taken into account when evaluating the performance of the rally point verification system, as it is trivial to come up with a case where the recall of the rally point verification system is close to 100% due to the segmentation system wrongfully predicting almost anywhere to be landable. This is not the case here.

The area that is needed to be clear of obstacles increases quadratically with the uncertainty of the pixel geolocator, as the exact whereabouts of the rally point is unknown, and needs to be added into the radius that needs to be clear of obstacles. The 20m radius represents just that, the uncertainty in the pixel geolocator in addition to the clearance radius around the rally point. As we see from Table 4.5, the number of verified rally points are more than halved due to the increase from 10m to 20m. Thus, even though the pixel geolocator has an ok average deviation, decreasing the uncertainty in the pixel geolocator is something that should be prioritized in order to improve the overall performance of the rally point verification system.

Furthermore, we approximated the elevation of the terrain by using data from *Høydedata* with a resolution of 100m². We can actually see the consequence of this by looking at the 20m radius in Figure 4.5 a) and b). We see that a small piece of those circles is separated from the rest of the circle. This is due to the height difference in the discretization in the respective areas. More conspicuous examples of this can be seen in Appendix G. To test the effect the resolution of the discretization has on the overall performance of the rally point verification, one could have run the above experiment with different grid sizes. As rally points are mostly located in flat areas, in accordance with the second desirable property of rally points listed in Section 2.1.2, this was not believed to have a substantial negative impact on the performance of the system. Hence, this was not tested.

5.7.2 Answering Research Question 5

Regarding Research Question 5, we see that the proposed deep learning-based system reliably manages to verify predetermined landing areas that are directly underneath the drone. Even though not all rally points are verified in the video, it is possible that subsequent flights will be able to verify the rally points that were outside the camera frame. If repeated flights do not manage to verify a specific rally point and rather refute it, a module to notify about this behavior could be implemented. Thus, rally points that in reality are not safe could be automatically detected and then manually removed from the missions.

5.8 Object Detection

Here we will discuss the Object detection-based SLAD-system presented in Section 3.8.3, and how it performs related to Research Question 1: *Which deep learning-based computer vision technique provides the best results in determining safe landing areas?* We will also explore whether the synthetic dataset had any effect, as mentioned in Research Question 4: *Can synthetic data be beneficial in*

generating rare situations when trying to train a deep learning-based system for safe landing area detection?

5.8.1 Pretraining

The aim of the pretraining was to improve performance on the classes vehicle, building and person. To determine whether pretraining benefited these categories, we can consider the avoidance scores in Table 4.7.

Vehicles and Buildings

Because none of the predicted landing areas ever contained any vehicles or buildings, it cannot be said with certainty whether pretraining provided any benefit for detecting these classes. This can be seen in Table 4.7, where the columns *AS Vehicle* and *AS Building* only contain zeroes. This is in principle a good result, as this means that all models have learned to predict landing areas that avoid these classes. However, it allows little discussion to be made regarding pretraining.

People

The people class provided some more interesting results. Since lower is better when it comes to Avoidance Score, we can see that none of the best models for detecting people are pretrained. These are:

- OD_Yolo_Res: 66.4%
- OD_SwinT_Res/Red (tie): 55.1%
- OD_CNX_Red: 96.6%

This provides strong evidence that the object detection system has not benefited from pretraining on our pretraining dataset when it comes to detecting people. The reasons for this are most likely the same as outlined in Section 5.2.2 regarding the weaknesses of the Unreal Engine dataset and the issues of the people class mentioned in Section 5.6.1. In addition, one can see in Figure 4.6 that the object detection-based SLAD system is not recognizing the person in the test set. The models which do not have any predictions in this figure most likely consider the terrain to be unsuitable for landing, as they do not consider the area next to the person to be safe either.

Average Precision Landing Area

An interesting observation is that the average precision of the predicted landing area compared to the ground truth seems to be positively impacted by pretrain-

ing. This can be seen in the *AP LA* column in Table 4.7. The best models at *AP LA* for each architecture were:

- OD_Yolo_Res_P: 82.7%
- OD_SwinT_Res_P: 82.4%
- OD_CNX_Red: 72.4%

Especially the Swin-T models seem to benefit considerably by pretraining. Both the pretrained Swin-T models manage to get over 12 percentage points better average precision on landing area than their non-pretrained counterparts. The same applies to OD_Yolo_Res_P, which has an improvement in accuracy of over 2 percentage points from the YOLO model in second place (OD_Yolo_Res). On the other hand, pretraining seems to not provide any benefits to ConvNeXt, as the best of these models is OD_CNX_Red.

Because of the elements discussed in the paragraphs above, it is hard to evaluate whether pretraining actually provide any benefit when it comes to the object detection-based SLAD system.

5.8.2 Impact of Color Restoration

To evaluate whether color restoration has an effect on the similarity between the pretraining set and the drone dataset in the context of the object detection networks, we can look at the difference between the pretrained models. Also in this case it is the classes of vehicle, building and people that should be evaluated.

As stated earlier, since the classes vehicle and building both have zero as their avoidance score for all models, these classes cannot be considered in evaluating whether color restoration has had any impact. We therefore need to look at the people class. In this case, there seems to be no improvement when training on color-restored images. The Res_P network performs worse than its Red_P counterpart in the case of Yolo and Swin-T. ConvNeXt gained somewhat of an improvement. However, it is important to consider the issues with the people class, as some of these results could be random.

If we look at the average precision of landing area (*AP LA*), Yolo and Swin-T seem to gain slight improvements from retraining on color-restored images. OD_Yolo_Res_P has over 5 percentage points better average precision than OD_Yolo_Red_P. On the other hand, ConvNeXt achieves worse performance when trained on color-restored images.

Considering the elements discussed above, it is hard to evaluate whether the color restoration provides a better mapping between the drone dataset and the pretraining dataset. We do not have any indicators that can tell us whether color restoration has helped when it comes to detecting vehicles or buildings. Still, the average precision of landing area seems to be positively impacted in two out of three models. Therefore, we can consider it plausible that the color restoration causes better similarity, which again makes the models benefit more from pretraining.

5.8.3 Answering Research Question 1,2 and 4

We now return to the Research Questions mentioned in Section 5.6.3.

Research Question 1: *Can synthetic data be beneficial in generating rare situations when trying to train a deep learning-based system for safe landing area detection?*

Now that we have discussed both segmentation and object detection, we can take the results from both of these sections into account when addressing this research question. The conclusion from Section 5.6.3 was that in case of the segmentation networks, there is a weak indication that pretraining might be beneficial. In Section 5.8.1 regarding pretraining the object detection models, we concluded that it is hard to evaluate. However, there is some observed benefit from pretraining when it comes to the average precision of the landing areas. Therefore, we conclude that the networks in general benefit slightly from pretraining on our pretraining dataset.

However, the research question specifically focuses on the synthetic Unreal Engine dataset. Since we mixed the Potsdam and Unreal Engine datasets into a combined pretraining dataset, we do not have a satisfactory method to directly answer this research question. The purpose of the Unreal Engine dataset was to provide more training examples of people. From the discussions in Sections 5.6.1 and 5.8.1 about pretraining and people, neither of the methods seems to have benefited from pretraining when it comes to detecting people. Also, the problems associated with the detection of people also need to be taken into account. Therefore, we cannot conclude whether synthetic data can be beneficial in this context or not. This needs to be researched further. The main improvement one can do is to have a pretraining dataset that solely consists of synthetic data. In addition, it would be beneficial to incorporate measures to create synthetic data that more closely resembles the real-world dataset. It is very much possible that the application of synthetic data can provide substantial benefit, as large actors such as Waymo and Tesla utilize synthetic data for their self-driving systems

[Waymo, 2021] [Anyverse, 2021].

Research Question 2: *How can images with distorted colors be restored to their original colors, and will this improve the effect of pretraining the deep learning models?*

We can now answer Research Question 2, as we have discussed the impact of the color restoration technique on training the segmentation models in Section 5.6.2 and the object detection models in Section 5.8.2.

As stated earlier, we consider the color restoration method to be a success. The colors of the restored images are more realistic and natural than those with a red hue. However, we also need to answer whether the color restoration provides a better resemblance with the pretraining dataset. The anticipation was that the models might benefit more from the pretraining if the colors in the datasets were more similar. The conclusion in Section 5.6.2 was that in the case of the segmentation models, there were clear indicators that certain models benefited from retraining on color-restored images. The conclusion in Section 5.8.2 was that since two out of three pretrained models achieved better results when retraining on color restored images, we also regard the color restoration to be beneficial in the case of the object detection models.

Because both architectures seem to have benefited from retraining on the color-restored images, we conclude that this has been beneficial. The cases in which the pretrained models performed worse than the non-pretrained ones might be because of weaknesses in the pretraining dataset itself, rather than the color similarity.

Research Question 4: *Which deep learning-based computer vision technique provides the best results in determining safe landing areas?*

We can now address Research Question 4, as we have discussed the results from each of the two SLAD systems.

The best performing object-detection model seems to be OD_SwinT_Res_P. This model has neither the best *AP LA* score, nor the best mAS. Still, it has a nice trade-off between the two, achieving practically the same *AP LA* score as the best OD model (OD_Yolo_Res_P), with a substantially better mAS score. In addition,

it is exceptionally effective at avoiding trees, with an avoidance score of just 10 in this category.

From Section 5.6.3, we have already established that `Seg_CNX_Res_P` is the best of the segmentation models. When we compare the best OD model with the best segmentation model, we discover that it is not easy to determine which is best. Even though the two SLAD systems depend on different architectures, their results are surprisingly hard to evaluate. If we look at their stats in Table 4.4 and 4.7, we see that while `Seg_CNX_Res_P` has a higher AP LA score of 87.2 vs 82.4, `OD_SwinT_Res_P` has a lower mean avoidance score of 16.6 vs 21.1. This is primarily because `OD_SwinT_Res_P` is so good at avoiding trees. This means that the OD-based SLAD system is slightly more careful when selecting points than the segmentation-based system, most likely at the expense of landing area size. Therefore, both of the SLAD system architectures presented in this thesis have proven their feasibility. Further research is required into both of these architectures to determine which is the most suitable. However, we would like to emphasize the convenience of the segmentation-based method, as this also can be used as part of the system to validate predetermined rally points. This is something that is especially useful in our use case.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The overall goal of this thesis was to explore the use of deep learning computer vision techniques in the context of safe landing area detection using monocular vision. In the pursuit of this goal, we have explored many different areas, including color restoration and generating synthetic data. We have also created and tested a series of subsystems: color restoration system, rally point converter, pixel geolocator, rally point verifier, segmentation networks and object detection networks. When combined, these subsystems constitute two functioning SLAD systems: one dependent on image segmentation and one dependent on object detection. Both of these SLAD systems are capable of successfully detecting landing areas that are free of obstacles and safe for the drone to land on.

However, when it comes to the feasibility of a deep learning-based safe landing area detection system using monocular vision, we have identified certain challenges:

- By solely using a single camera to interpret the terrain below the drone, the elevation of the terrain is not taken into account. This could cause the drone to land on steep slopes.
- The SLAD systems in this thesis were largely trained on images that contained snow, this limits their generalizability.
- The ability of the deep learning systems to detect small features heavily

depends on the resolution of the images. In our case, the resolution was not sufficient for the system to detect people.

- The black box nature of neural networks makes their behavior unpredictable. Therefore, it is not known what predictions the SLAD systems will generate if they encounter objects they have not been trained on.

Considering these challenges, we would not consider any of the SLAD systems presented in this thesis to be robust enough to select new rally points completely autonomously. On the other hand, one could argue that the rally points provided by the SLAD systems are better than some of the current, hand-picked rally points, considering some of these are not in a safe area. Still, the safest application of the systems presented in this thesis is probably to recommend new rally points that then can be confirmed by a human. In addition, using the rally point verification system, the SLAD systems can also be used to notify human operators if a rally point seems to be obstructed, for example by new building activity or vegetation. The systems provided in this thesis are more than capable enough to fulfill these use-cases. For complete autonomous operation, some further work is required.

6.2 Future Work

To further contribute to the research goal of this thesis, there exists many opportunities for improvement in future work. This includes measures to address the challenges outlined in the conclusion above, but also in other areas.

Data from multiple sources might be beneficial when determining landing areas. Therefore, one could explore including more cameras and sensors such as LIDAR. With multiple cameras, one can also use techniques to extract depth from the images. One of the strengths of the pixel geolocator is that it also allows for geolocation conversion in the case of multiple cameras. A combined system that utilizes both downward and forward-facing cameras could therefore be devised. However, one of the main benefits of a monocular SLAD system is that a single camera adds little extra weight to the drone.

More diverse data from different parts of the year so that the dataset includes more diverse terrain would be beneficial for generalizability. Another way of achieving this could be to augment the data we already have collected, for example by replacing the snow with fields and similar.

Increasing the resolution of the video stream can be beneficial in order to detect smaller obstacles more reliably. In this thesis, there were limitations from

Aviant that restricted what resolution we could choose. If we were to increase the resolution, we would have to re-evaluate these restrictions in cooperation with Aviant.

Automatic labeling using height data is an interesting aspect one could explore to label even more data automatically. By including a Digital Elevation Model (DEM) in the automatic labeling system, stationary obstacles could be detected. There exist two different kinds of DEMs. A digital surface model includes the earth's surface in addition to all objects on it, such as trees and buildings. A digital terrain model includes only the surface of the earth, without the objects on top of it [Li et al., 2005]. By looking at the difference between these two models, one could determine where there are obstacles on top of the ground. Since the masks we use are binary, obstacle or no obstacle, this information could be used to automatically generate binary masks. In combination with drone footage or satellite imagery and the pixel geolocator, this could be used to train the SLAD system. Furthermore, one could impose restrictions on the labeling such that only flat areas are regarded as safe for landing. To achieve this kind of automatic labeling, the performance of the pixel geolocator would have to be improved.

Bibliography

- Abadpour, A. and Kasaei, S. (2007). An efficient pca-based color transfer method. *Journal of Visual Communication and Image Representation*, 18(1):15–34.
- Adrian Rosebrock (2016). https://commons.wikimedia.org/wiki/File:Intersection_over_Union_-_visual_equation.png Image by Adrian Rosebrock, via Wikimedia Commons. Creative Commons License CC BY-SA 4.0. Not edited.
- Adrian Rosebrock (2017). https://commons.wikimedia.org/wiki/File:Intersection_over_Union_-_poor,_good_and_excellent_score.png Image by Adrian Rosebrock, via Wikimedia Commons. Creative Commons License CC BY-SA 4.0. Not edited.
- AirSim (2017). <https://github.com/microsoft/AirSim> AirSim simulator for drones, cars and more. Open source project by Microsoft.
- Ansari, K., Krebs, A., Benezeth, Y., and Marzani, F. (2019). Color converting of endoscopic images using decomposition theory and principal component analysis. pages 151–159.
- Anyverse (2021). <https://anyverse.ai/news/tesla-ai-day-physically-correct-synthetic-data/> Tesla bets on the physically correct synthetic data at its AI Day.
- Auawise (2010). https://commons.wikimedia.org/wiki/File:Yaw_Axis_Corrected.svg Yaw_Axis_Corrected.svg Image by Auawise, via Wikimedia Commons. Creative Commons License CC BY-SA 3.0. Not edited.
- Aviant (2022). <https://www.aviant.no> Drone transport company.
- Barron, J. T. (2015). Convolutional color constancy. *CoRR*, abs/1507.00410.
- Beauchemin, S. and Barron, J. (1995). The computation of optical flow. *ACM Computing Surveys (CSUR)*, 27:433–466.

- Bektash, O., Pedersen, J. N., Ramirez Gomez, A., and Cour-Harbo, A. I. (2020). Automated emergency landing system for drones: Safeeye project. In *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1056–1064.
- Bochkovskiy, A., Wang, C., and Liao, H. M. (2020). Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934.
- Boulter (2004). <https://boulter.com/gps/distance/> Coordinate Distance Calculator.
- Breiman, L. (2001). Random forests. *Machine Learning*, 45:5–32.
- Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698.
- Cesetti, A., Frontoni, E., Mancini, A., Zingaretti, P., and Longhi, S. (2009). A vision-based guidance system for uav navigation and safe landing using natural landmarks. *Journal of intelligent & robotic systems*, 57(1-4):233–257.
- CityPark (2021). <https://www.unrealengine.com/marketplace/en-US/product/city-park-environment-collection-lite> City Park map for Unreal Engine by SilverTim.
- CVAT (2022). <https://openvinotoolkit.github.io/cvat/about/> Computer Vision Annotation Tool.
- Daugman, J. (1988). Complete discrete 2-d gabor transforms by neural networks for image analysis and compression. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(7):1169–1179.
- Demirhan, M. and Premachandra, C. (2020). Development of an automated camera-based drone landing system. *IEEE Access*, 8:202111–202121.
- Deng, Y., Ni, Y., Li, Z., Mu, S., and Zhang, W. (2017). Toward real-time ray tracing: A survey on hardware acceleration and microarchitecture techniques. *ACM Computing Surveys*, 50:1–41.
- Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.
- Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929.

- Dpt.Defense (1991). United states department of defense, world geodetic system 1984: Its definition and relationships with local geodetic systems. *Tech. Rep., TR8350.2*.
- Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110.
- EASA (2021). <https://www.easa.europa.eu/document-library/easy-access-rules/easy-access-rules-unmanned-aircraft-systems-regulation-eu#group-publications> European Union Aviation Safety Agency. Easy Access Rules for Unmanned Aircraft Systems (Regulation (EU) 2019/947 and Regulation (EU) 2019/945).
- Gagunashvili, N. (2009). Chi-square tests for comparing weighted histograms. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 614:287–296.
- Google Maps (2005). <https://maps.google.com>.
- Google Scholar (2004). <https://scholar.google.com/> A simple way to broadly search for scholarly literature.
- Gstreamer (2004). <https://gstreamer.freedesktop.org/> Open source multimedia framework.
- Hackem (2020). https://commons.wikimedia.org/wiki/File:Computer_vision_sample_in_SimImage by Comunidad de Software Libre Hackem [Research Group], via Wikimedia Commons. Creative Commons License CC BY-SA 3.0. Not edited.
- He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.
- Hinzmann, T., Stastny, T., Cadena, C., Siegwart, R., and Gilitschenski, I. (2018). Free lsd: Prior-free visual landing site detection for autonomous planes. *IEEE Robotics and Automation Letters*, 3(3):2545–2552.
- Hsiao, Y.-M., Lee, J.-F., Chen, J.-S., and Chu, Y.-S. (2011). H.264 video transmissions over wireless networks: Challenges and solutions. *Computer Communications*, 34(14):1661–1672.
- Hu, Y., Wang, B., and Lin, S. (2017). Fc4: Fully convolutional color constancy with confidence-weighted pooling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Høydedata (2017). <https://hoydedata.no/> Terrain height data. Norwegian Mapping and Cadastre Authority.

- Iver Waldahl Lillegjære (2022). <https://nearradio.no/sjekket-av-tollere/19.29023>
Unik grensekryssing: Sjekket av tollerne.
- Kaljahi, M. A., Shivakumara, P., Idris, M. Y. I., Anisi, M. H., Lu, T., Blumenstein, M., and Noor, N. M. (2019). An automatic zone detection system for safe landing of uavs. *Expert Systems with Applications*, 122:319–333.
- Kirillov, A., He, K., Girshick, R., Rother, C., and Dollar, P. (2019). Panoptic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Koubâa, A., Allouch, A., Alajlan, M., Javed, Y., Belghith, A., and Khalgui, M. (2019). Micro air vehicle link (mavlink) in a nutshell: A survey. *IEEE Access*, 7:87658–87680.
- LandscapeMountains (2020). <https://www.unrealengine.com/marketplace/en-US/product/landscape-mountains> Landscape Mountains map for Unreal Engine by Epic Games.
- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.
- Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- Lee, M.-F. R., Nugroho, A., Le, T.-T., Bahrudin, and Bastida, S. N. (2020). Landing area recognition using deep learning for unmanned aerial vehicles. In *2020 International Conference on Advanced Robotics and Intelligent Systems (ARIS)*, pages 1–6.
- Li, Z., Zhu, Q., and Gold, C. (2005). *Digital Terrain Modeling: Principles and Methodology*.
- Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. *CoRR*, abs/2103.14030.
- Liu, Z., Mao, H., Wu, C., Feichtenhofer, C., Darrell, T., and Xie, S. (2022). A convnet for the 2020s. *CoRR*, abs/2201.03545.
- Long, J., Shelhamer, E., and Darrell, T. (2014). Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038.

- LoweDavid, G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*.
- MAVLink (2017). <https://github.com/mavlink-router/mavlink-router> Application to distribute MAVLink messages between multiple endpoints (connections).
- MMDetection (2018). <https://github.com/open-mmlab/mmdetection> MMDetection open source object detection toolbox based on PyTorch.
- MMSegmentation (2020). <https://github.com/open-mmlab/mms Segmentation> MMSegmentation open source segmentation toolbox based on PyTorch.
- Nguyen, P. H., Arsalan, M., Koo, J. H., Naqvi, R. A., Truong, N. Q., and Park, K. R. (2018). LightDenseYOLO: A fast and accurate marker tracker for autonomous uav landing by visible light camera sensor on drone. *Sensors*, 18(6).
- Norge i bilder (2005). <https://norgebilder.no/> Statens vegvesen, Norsk institutt for Bioøkonomi (NIBIO) og Statens kartverk.
- Nvidia (1993). <https://www.nvidia.com/> GPUs and APIs for data science and high-performance computing.
- Papers with code (2018). <https://paperswithcode.com/> Free and open resource with Machine Learning papers, code, datasets, methods and evaluation tables.
- Park, S. C., Park, M. K., and Kang, M. G. (2003). Super-resolution image reconstruction: a technical overview. *IEEE Signal Processing Magazine*, 20(3):21–36.
- Patterson, T., McClean, S., Morrow, P., Parr, G., and Luo, C. (2014). Timely autonomous identification of uav safe landing zones. *Image and Vision Computing*, 32.
- PeoplePack (2019). <https://www.unrealengine.com/marketplace/en-US/product/9c3fab270dfe468a9a920da0c10fa2ad> Scanned 3D People Pack for Unreal Engine by Renderpeople.
- Potsdam (2016). <https://www.isprs.org/education/benchmarks/UrbanSemLab/2d-sem-label-potsdam.aspx> Potsdam 2D Semantic Labeling Contest. The ISPRS Foundation.
- Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

- PyTorch (2019). <https://github.com/pytorch/pytorch> Tensors and Dynamic neural networks in Python with strong GPU acceleration.
- QGC (2015). <https://qgroundcontrol.com/> QGroundControl Ground Control Station. Intuitive and powerful ground control station (GCS) for UAVs.
- Redmon, J., Divvala, S. K., Girshick, R. B., and Farhadi, A. (2015). You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640.
- Redmon, J. and Farhadi, A. (2016). YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242.
- Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement. *CoRR*, abs/1804.02767.
- Ren, S., He, K., Girshick, R. B., and Sun, J. (2015). Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497.
- Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597.
- Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.
- Shah Alam, M. and Oluoch, J. (2021). A survey of safe landing zone detection techniques for autonomous unmanned aerial vehicles (uavs). *Expert Systems with Applications*, 179:115091.
- Shao, F., Jiang, G., and Yu, M. (2007). Color correction for multi-view images combined with pca and ica.
- SMP (2019). https://github.com/qubvel/segmentation_models.pytorch Python library with Neural Networks for Image Segmentation based on PyTorch.
- Snyder, J. P. (1987). *Map projections—A working manual*, volume 1395. US Government Printing Office.
- StonePineForest (2021). <https://www.unrealengine.com/marketplace/en-US/product/stone-pine-forest> Stone Pine Forest map for Unreal Engine by Mythra Tech.

- Toussaint, G. (2000). Solving geometric problems with the rotating calipers. *In Proceedings of IEEE MELECON'83*, 83.
- Truong, N. Q., Nguyen, P. H., Nam, S. H., and Park, K. R. (2019). Deep learning-based super-resolution reconstruction and marker detection for drone landing. *IEEE Access*, 7:61639–61655.
- UDP (1980). User Datagram Protocol. RFC 768.
- Unreal Engine (2022). <https://www.unrealengine.com> Unreal Engine: Real-time 3D creation tool.
- v4l2rtspserver (2015). <https://github.com/mpromonet/v4l2rtspserver> RTSP Server for V4L2 device capture supporting HEVC/H264/JPEG/VP8/VP9 .
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.
- Wang, N., Zhou, Y., Han, F., Zhu, H., and Yao, J. (2019). Uwgan: Underwater gan for real-world underwater color restoration and dehazing.
- Waymo (2021). <https://blog.waymo.com/2021/06/SimulationCity.html> Simulation City: Introducing Waymo’s most advanced simulation system yet for autonomous driving.
- Xiao, T., Liu, Y., Zhou, B., Jiang, Y., and Sun, J. (2018). Unified perceptual parsing for scene understanding. *CoRR*, abs/1807.10221.
- Xie, S., Girshick, R. B., Dollár, P., Tu, Z., and He, K. (2016). Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431.
- Yang, T., Li, P., Zhang, H., Li, J., and Li, Z. (2018). Monocular vision slam-based uav autonomous landing in emergencies and unknown environments. *Electronics*, 7(5).
- YOLOv5 (2020). <https://github.com/ultralytics/yolov5> YOLOv5 Pytorch framework by Ultralytics.

Appendices

A Literature Review Table

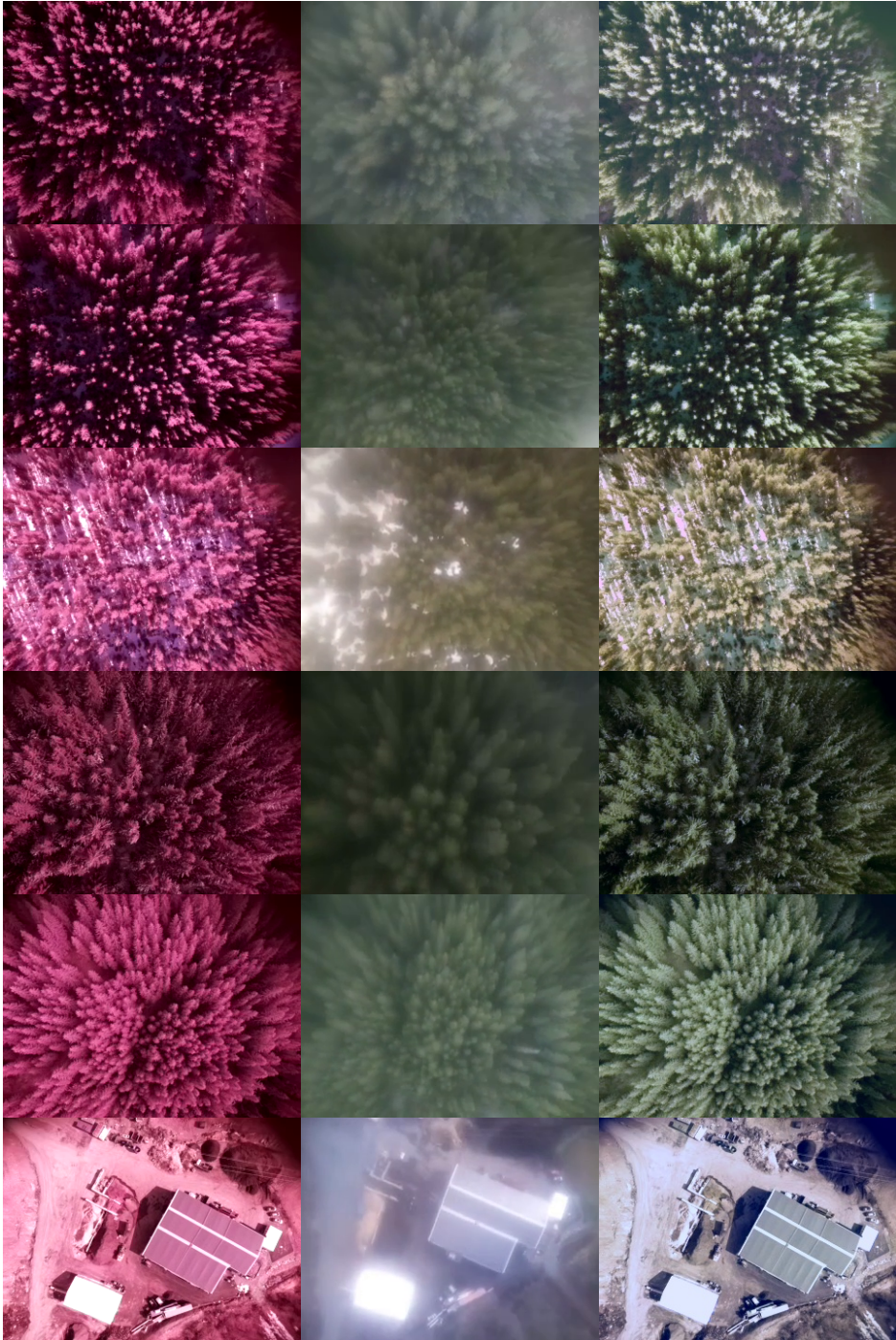
Paper	Search Engine	Year Limit	Search term	Criteria	Score
A survey of safe landing zone detection techniques for autonomous unmanned aerial vehicles (UAVs)	NTNU	2021	G2^G3^G5^G6	IC: 2,3	2
On-board vision autonomous landing techniques for quadrotor: A survey	IX	2016	G1^G3^G5	IC: 2,3	2
Evaluation of Safe Landing Area Determination Algorithms for Autonomous Rotorcraft Using Site Benchmarking	NTNU	2011	G2^G3^G6	IC: 2,4	2
Evaluation of Runtime Monitoring for UAV Emergency Landing	NTNU	2022	G2^G3^G6	IC: 1,2,3,4	4
Automated Emergency Landing System for Drones: SafeEYE Project	IX	2020	G2^G3^G6	IC: 1,2,3,4	4
Surface-Condition Detection System of Drone-Landing Space using Ultrasonic Waves and Deep Learning	IX	2020	G1^G2^G3	IC: 2,3,4	3
Visual-based Safe Landing for UAVs in Populated Areas: Real-time Validation in Virtual Environments	NTNU	2022	G2^G3^G6	IC: 1,2,4	3
Computer Vision for Autonomous UAV Flight Safety: An Overview and a Vision-based Safe Landing Pipeline Example	NTNU	2021	G1^G2^G3^G5^G6	IC: 1,2,3,4	4
Autonomous Detection of Safe Landing Areas for an UAV from Monocular Images	IX	2006	G1^G2^G3^G5^G6	IC: 1,2,4	3
A Vision-Based Guidance System for UAV Navigation and Safe Landing Using Natural Landmarks	NTNU	2010	G1^G2^G3^G6	IC: 1,2,4	3
Vision-based UAV Safe Landing exploiting Lightweight Deep Neural Networks	NTNU	2021	G1^G2^G3^G6	IC: 1,2,3	3
Timely autonomous identification of UAV safe landing zones	NTNU	2014	G2^G3^G5^G6	IC: 1,2,3,4	4
Survey on Computer Vision for UAVs: Current Developments and Trends	NTNU	2016	G1^G2	IC: 2,3	2
Vision based autonomous landing system for UAV: A survey	IX	2014	G1^G2^G3^G5	IC: 1,2	2
An automatic zone detection system for safe landing of UAVs	NTNU	2019	G2^G3^G5^G6	IC: 1,2,3,4	4
Landing Area Recognition using Deep Learning for Unmanned Aerial Vehicles	IX	2020	G1^G2^G3	IC: 1,2,3,4	4
Monocular vision SLAM-based UAV Autonomous Landing in Emergencies and Unknown Environments	NTNU	2018	G1^G2^G3^G5^G6	IC: 1,2,4	3
Aerial image segmentation by use of textural features	IX	2016	G1^G2	IC: 3,4	2
LUAI Challenge 2021 on Learning to Understand Aerial Images	PWC	2021	G1^G2	IC: 3,4	3
PointFlow: Flowing Semantics Through Points for Aerial Image Segmentation	PWC	2021	G1^G2	IC: 2,3,4	3
Semantic Segmentation Of Aerial Images With An Ensemble Of CNNs	NTNU	2016	G1^G2	IC: 3,4	2
Semantic Labeling of Large-Area Geographic Regions Using Multi-View and Multi-Date Satellite Images and Noisy OSM Training Labels	NTNU	2020	G1^G2	IC: 3,4	2
UVID-Net: Enhanced Semantic Segmentation of UAV Aerial Videos by Embedding Temporal Information	IX	2020	G1^G2	IC: 2,3,4	3
SPIN Road Mapper: Extracting Roads from Aerial Images via Spatial and Interaction Space Graph Reasoning for Autonomous Driving	GS	2021	G1^G2	IC: 3,4	2
Image Segmentation using deep learning: A survey	GS	2021	G1	IC: 2,3	2
Multitask Learning of Height and Semantics From Aerial Images	IX	2019	G1^G2	IC: 4	1
Map Creation from Semantic Segmentation of Aerial Images Using Deep Convolutional Neural Networks	NTNU	2018	G1^G2	IC: 2,3,4	3
LightDenseYOLO: A Fast and Accurate Marker Tracker for Autonomous UAV Landing by Visible Light Camera Sensor on Drone	GS	2018	G1^G2^G3	IC: 1,2,3,4	4
Autonomous Vision-based Target Detection and Safe Landing for UAV	NTNU	2018	G1^G2^G3^G5^G6	IC: 1,2,4	3
Multi-Model Estimation Based Moving Object Detection for Aerial Video	GS	2015	G1^G2	IC: 3,4	2
Development of an Automated Camera-Based Drone Landing System	GS	2020	G1^G2^G3^G5	IC: 1,2,4	3
Unmanned Aerial Vehicle Emergency Landing Site Identification System using Machine Vision	GS	2015	G1^G2^G3^G6	IC: 1,2,4	3
Free LSD: Prior-Free Visual Landing Site Detection for Autonomous Planes	GS	2015	G2^G3^G5	IC: 1,2,4	3
Computer vision in autonomous unmanned aerial vehicles-A systematic mapping study	GS	2019	G1^G2^G5	IC: 2	1

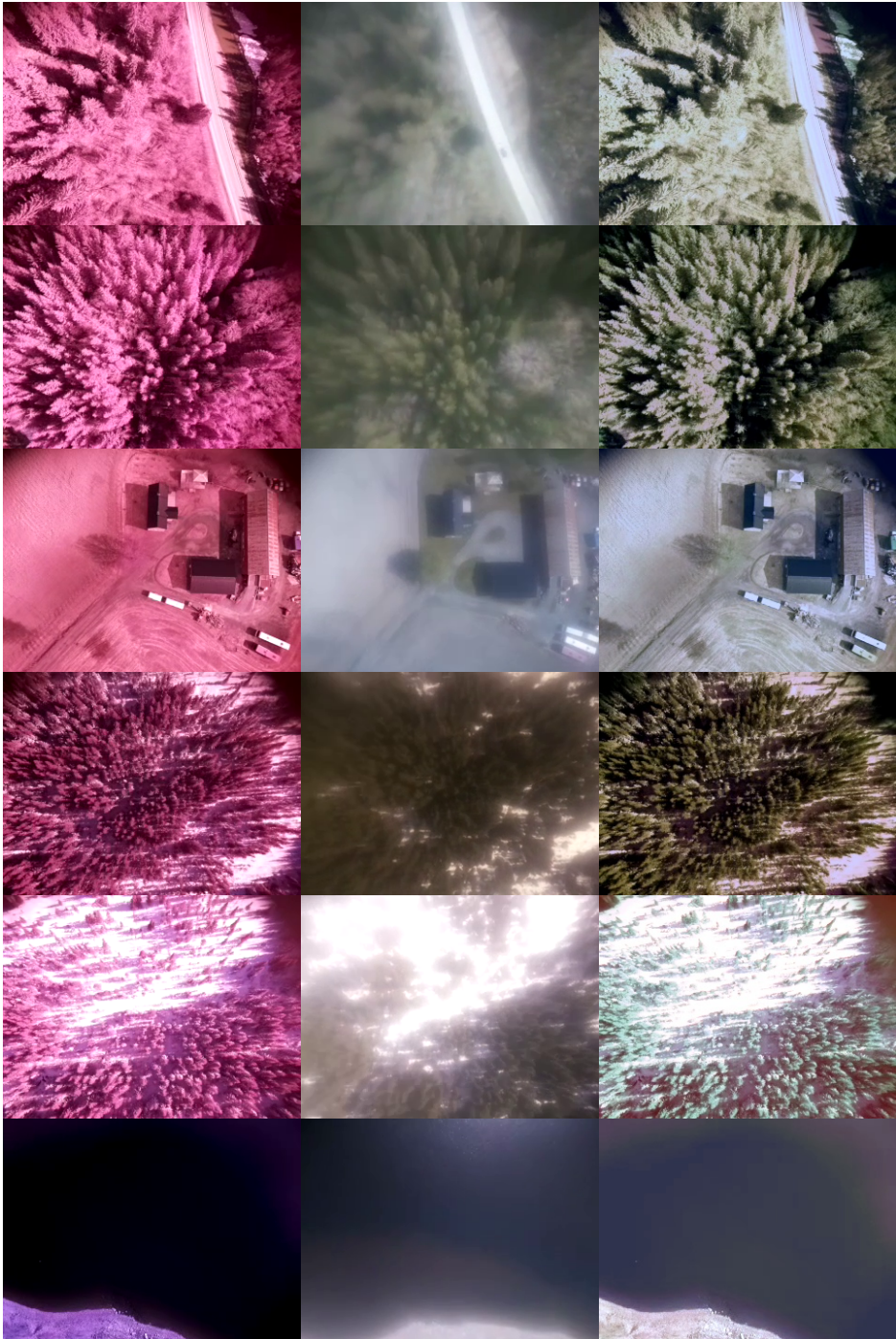
Table A: Evaluation of papers in Literature Review

Search Engine	ID	URL
Oria	NTNU	ntnu.oria.no
IEEE Xplore	IX	ieeexplore.ieee.org
Google Scholar	GS	scholar.google.com
Papers With Code	PWC	paperswithcode.com

Table B: The search engines used in the literature review

B Reference images for PCA





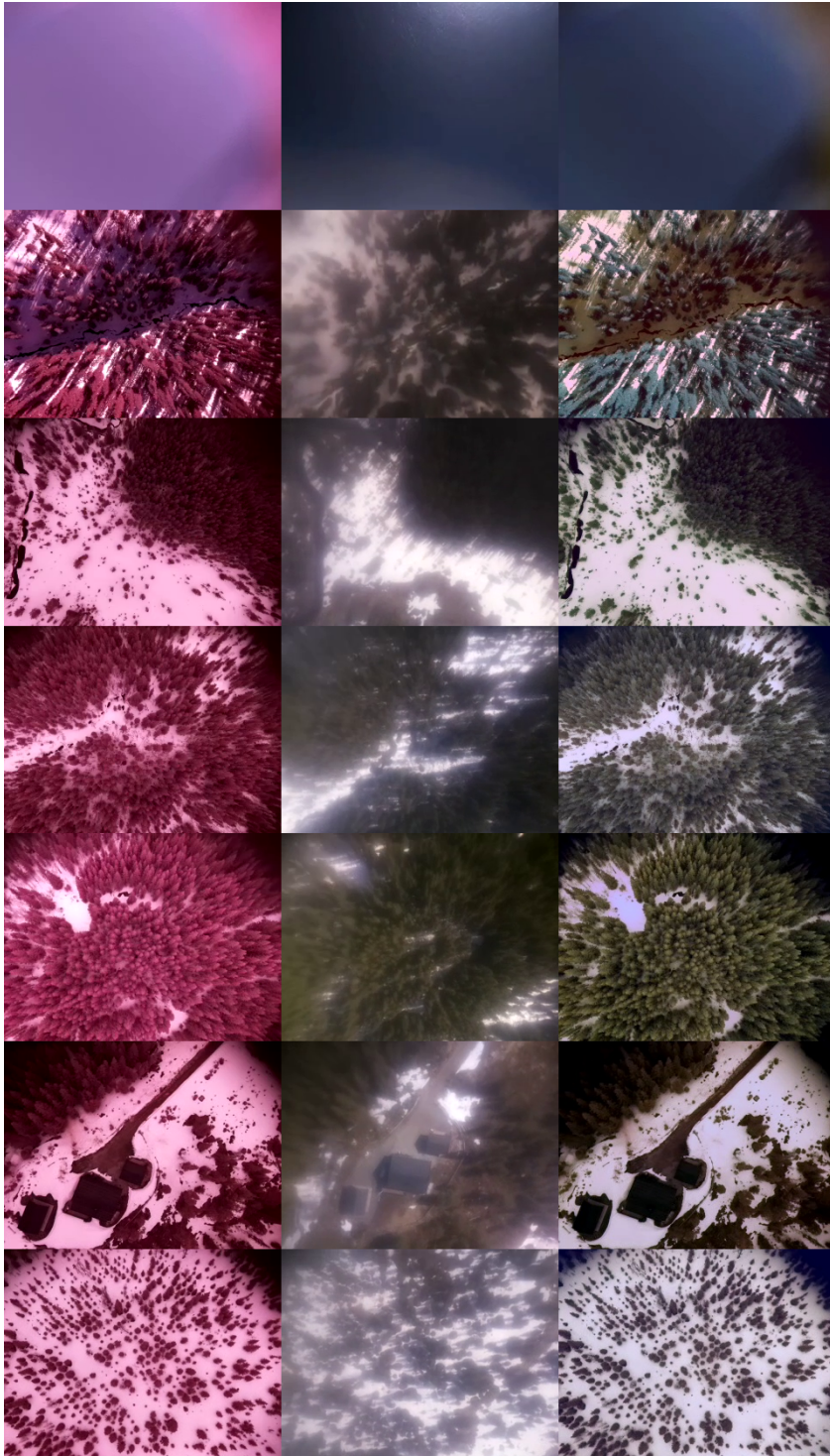


Figure 1: Reference images used for PCA color restoration. The leftmost column are the original images. The center column shows the most similar colored image, which is used to create the target distribution. The rightmost column shows the images with restored colors.

C Result images from color restoration

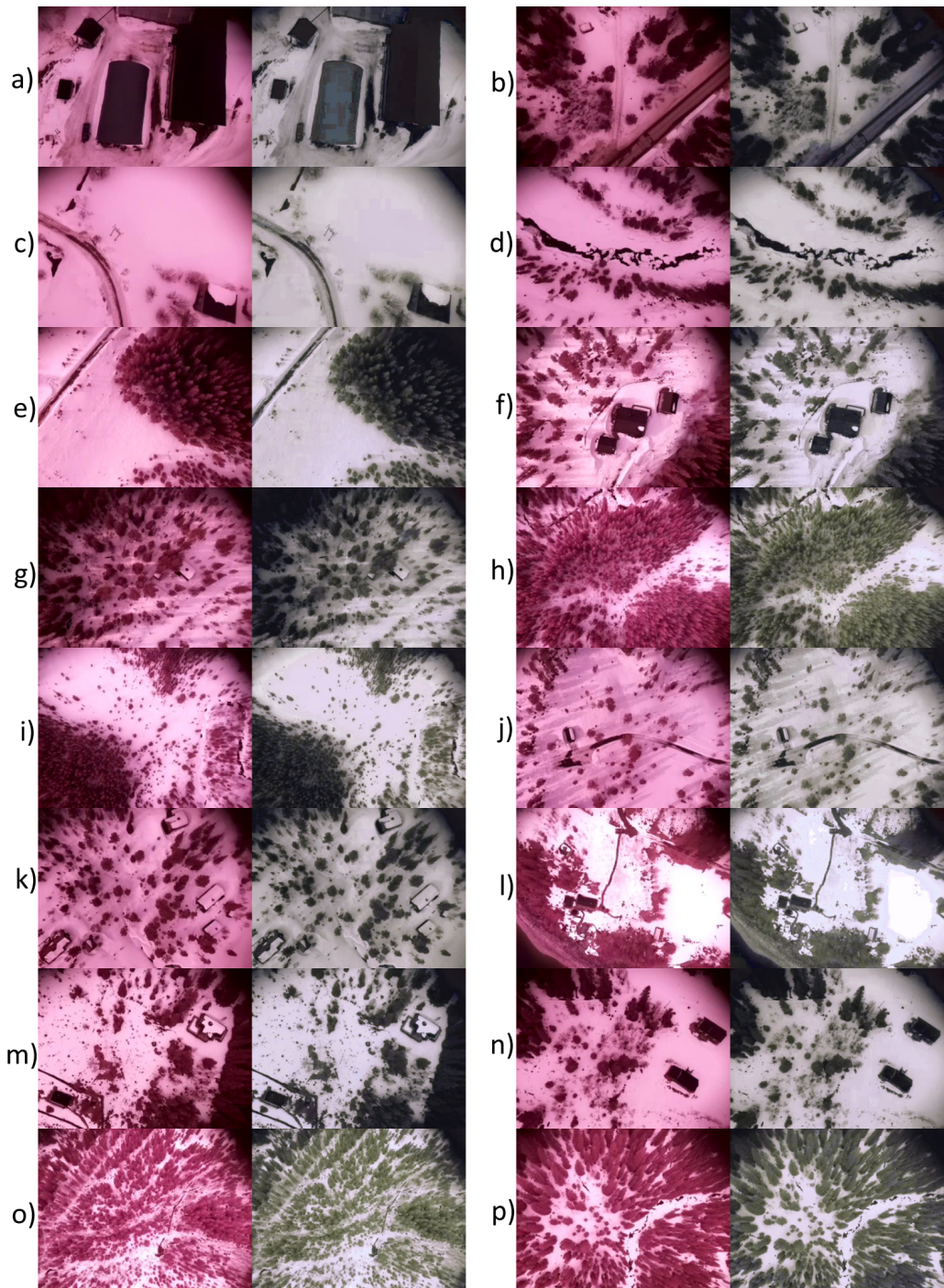


Figure 2



Figure 3: A selection of images showing the results from the color mapper. Unlike the images presented in Appendix B, the color mapper has not been calibrated on these images.

D Masks from Segmentation Models

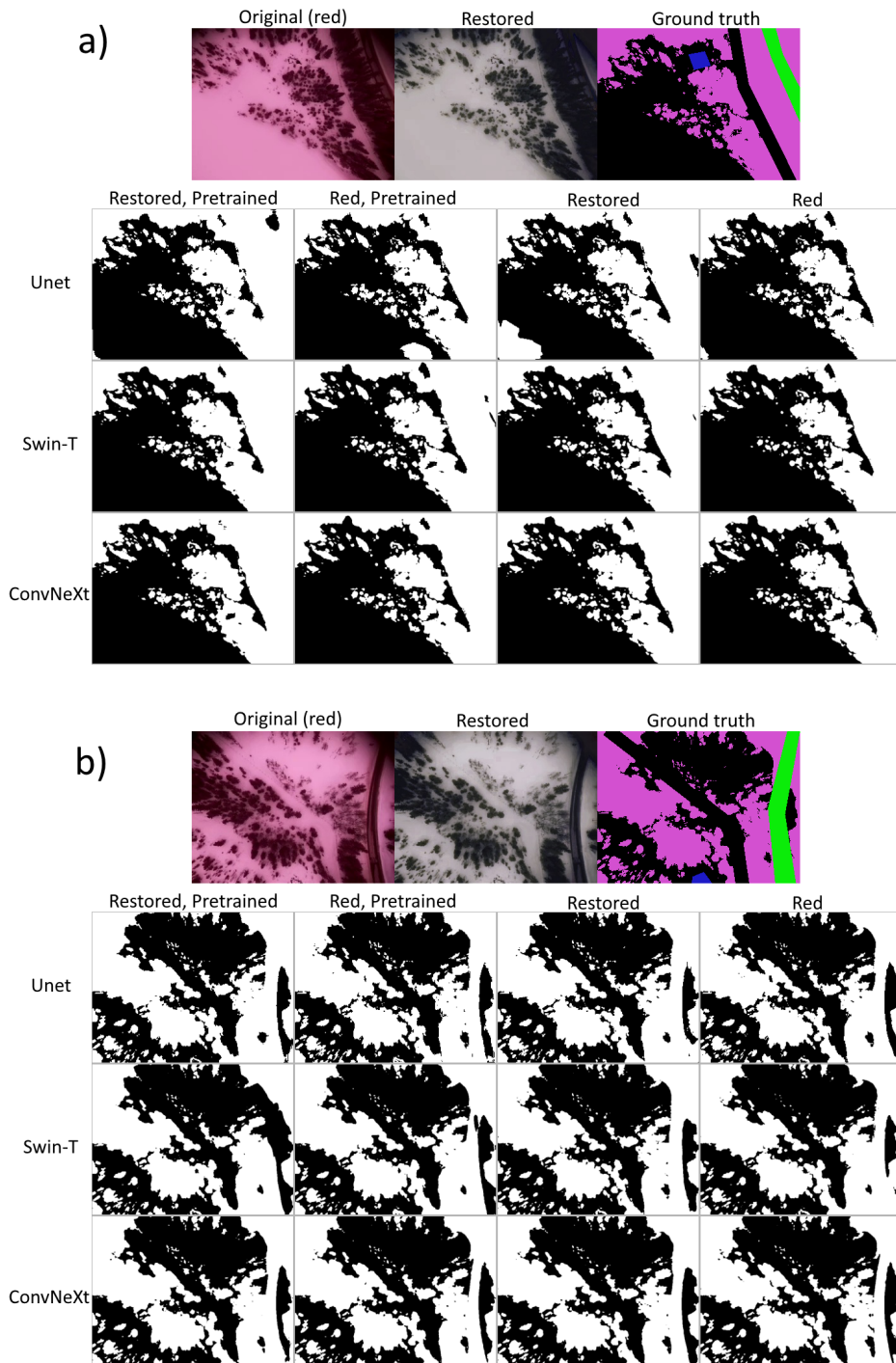


Figure 4

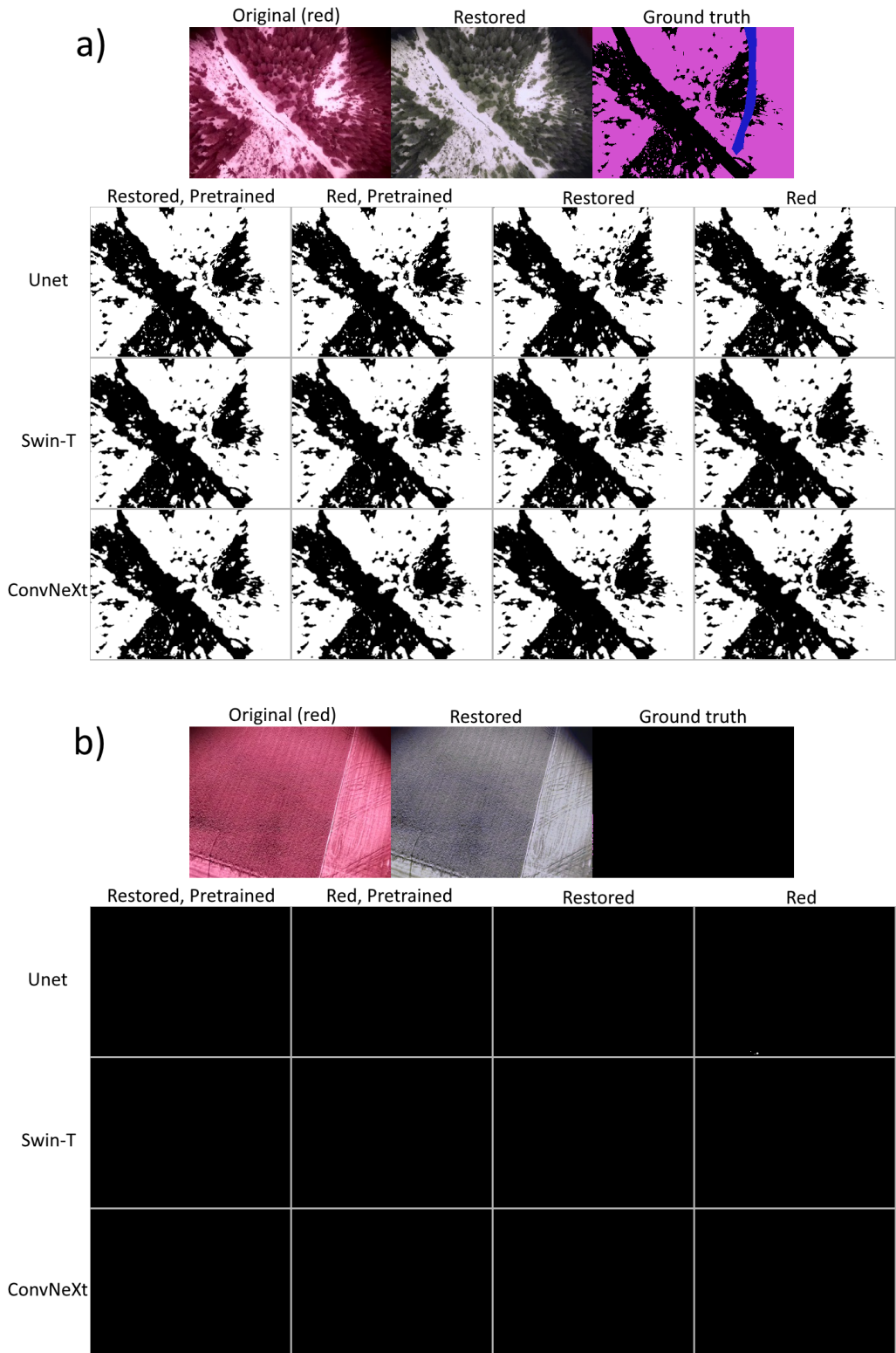


Figure 5

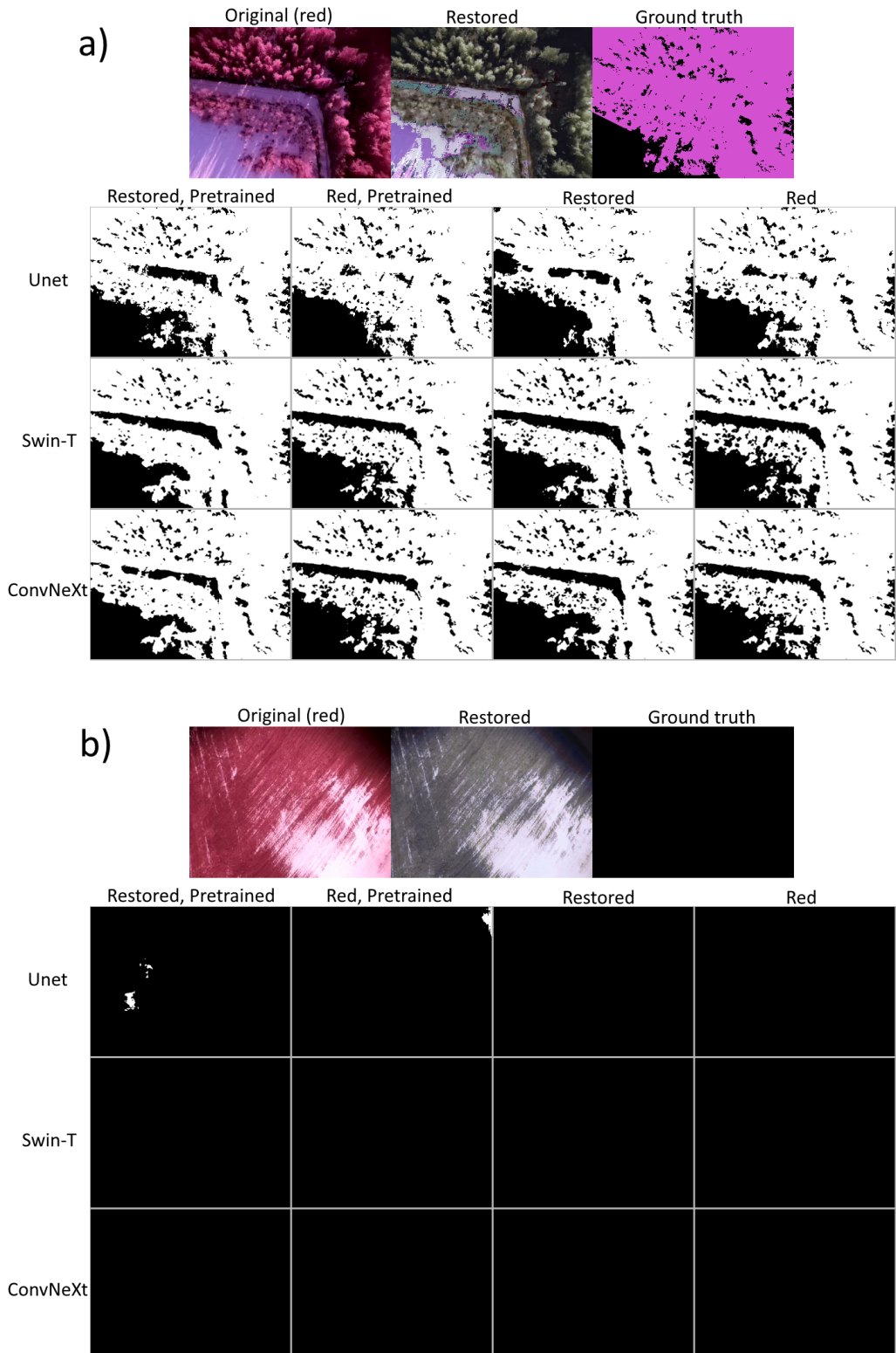


Figure 6

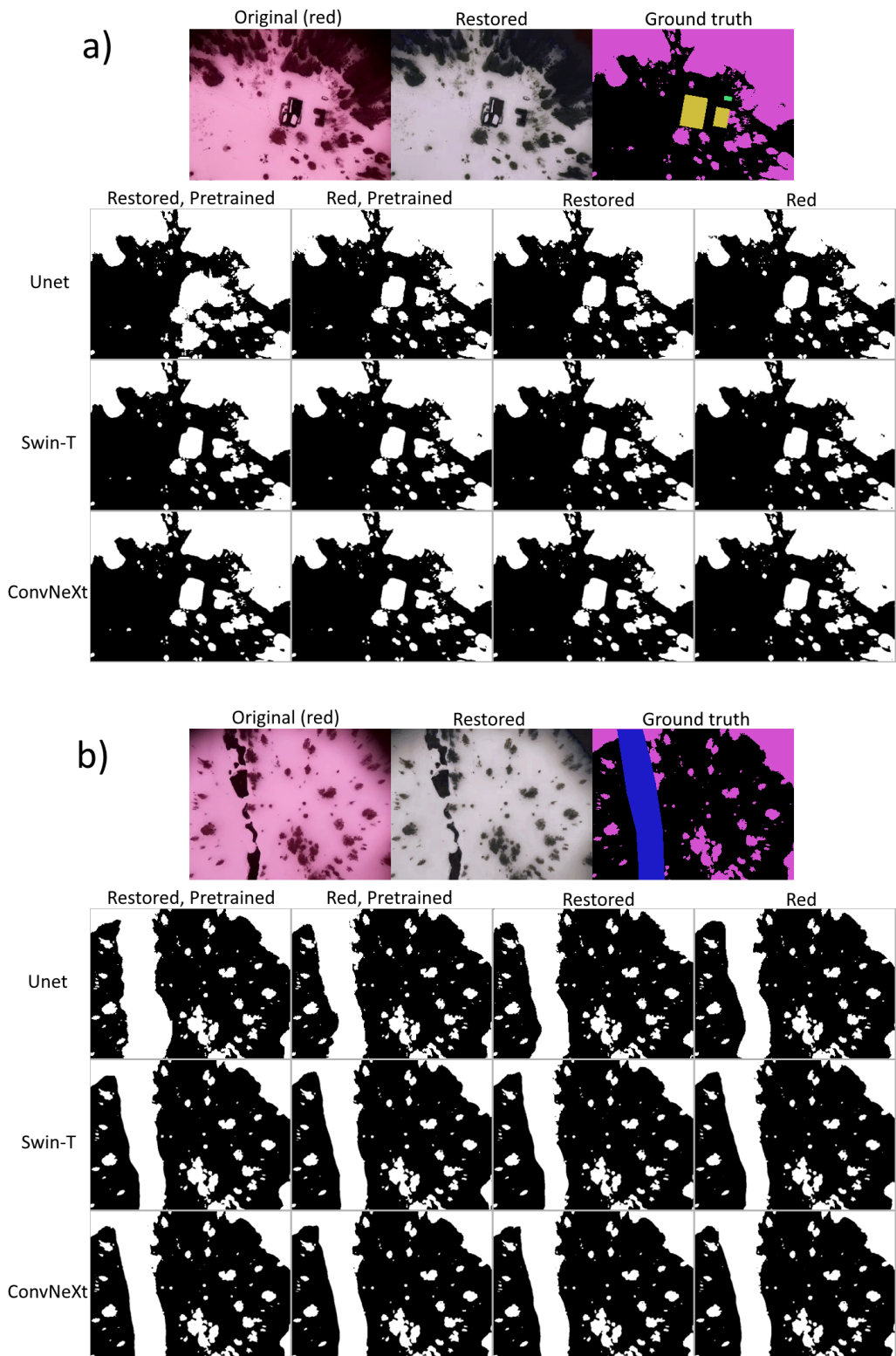


Figure 7

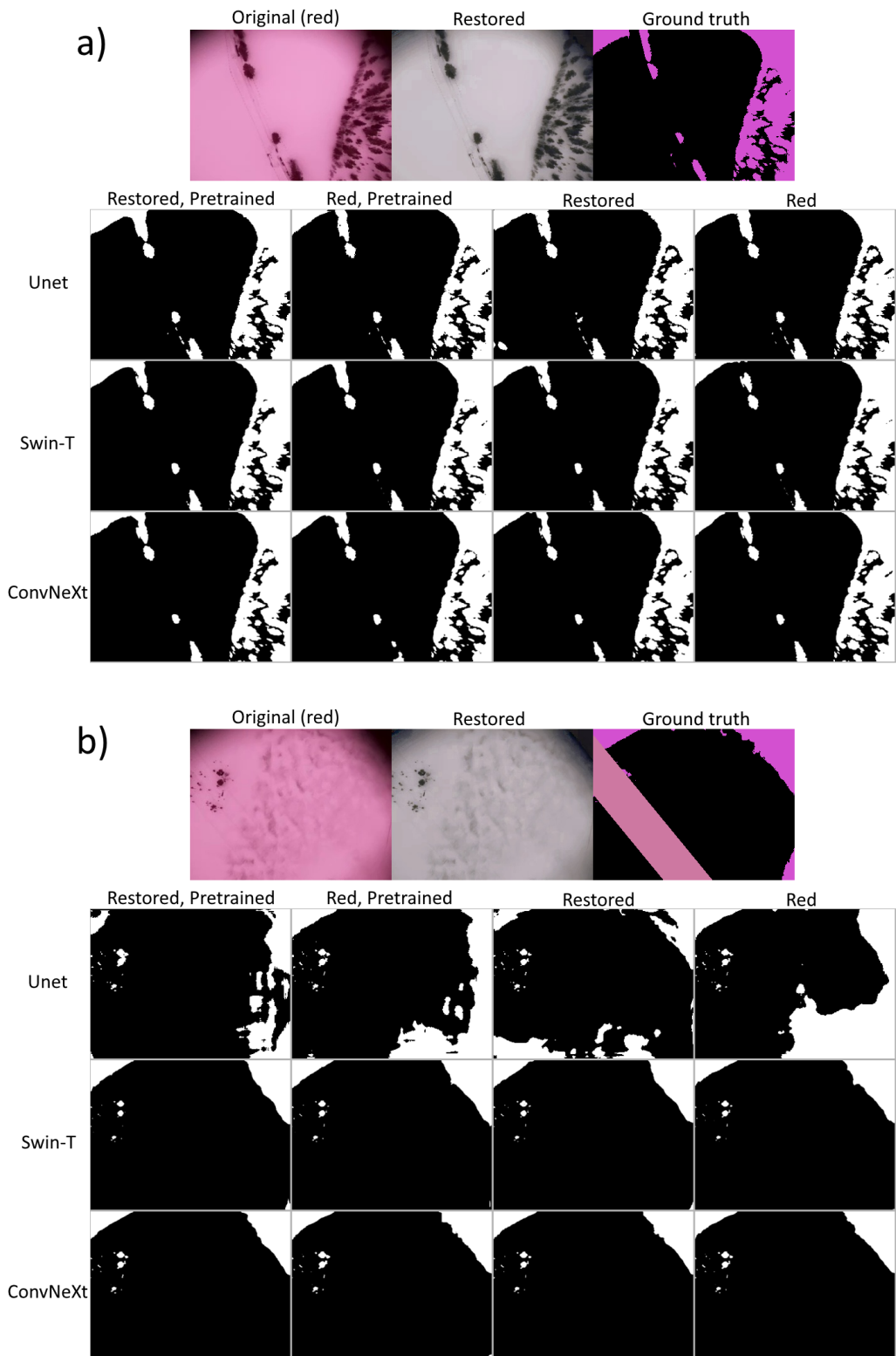


Figure 8

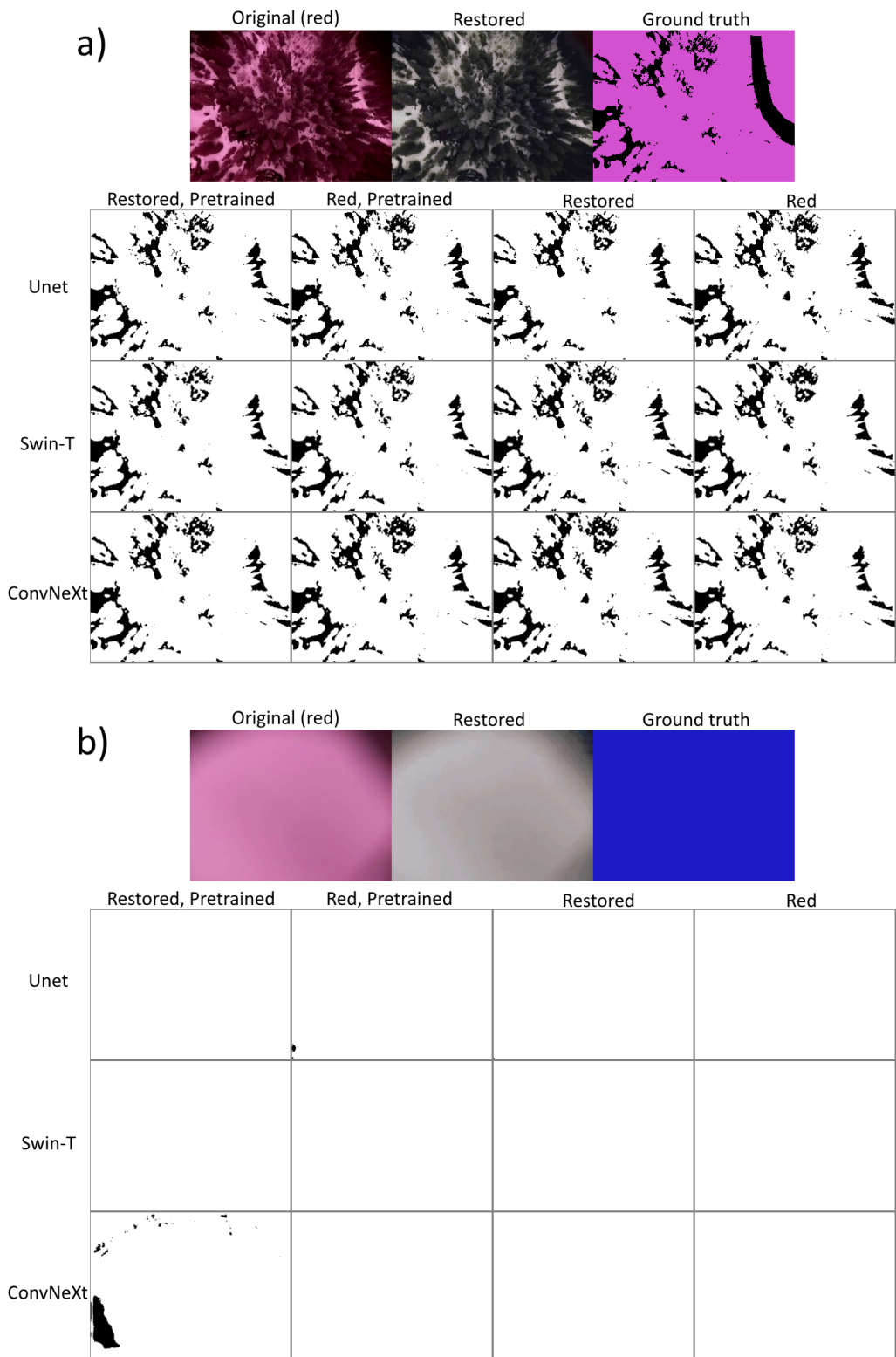


Figure 9

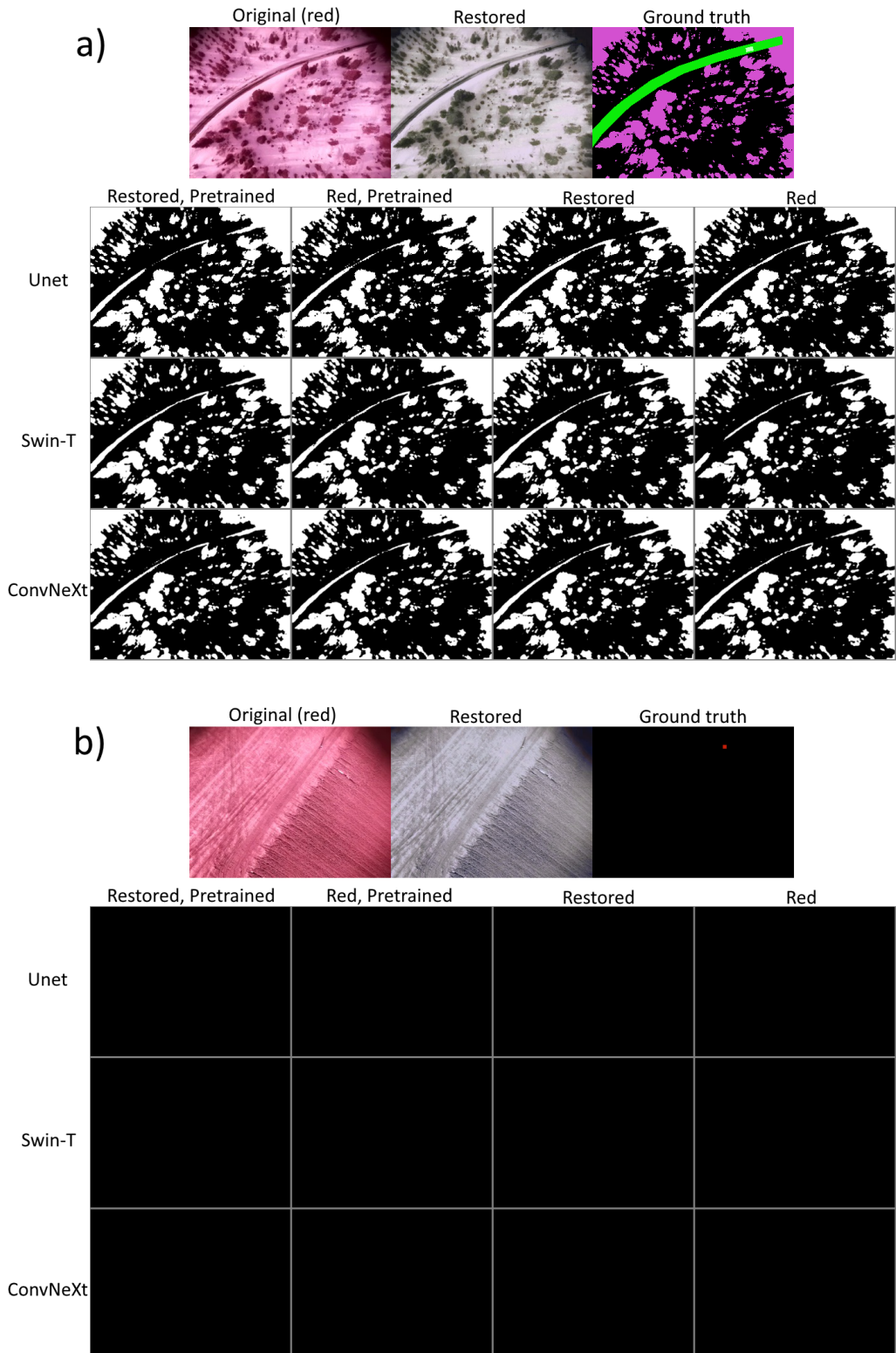


Figure 10

E Results from Object Detection-based System

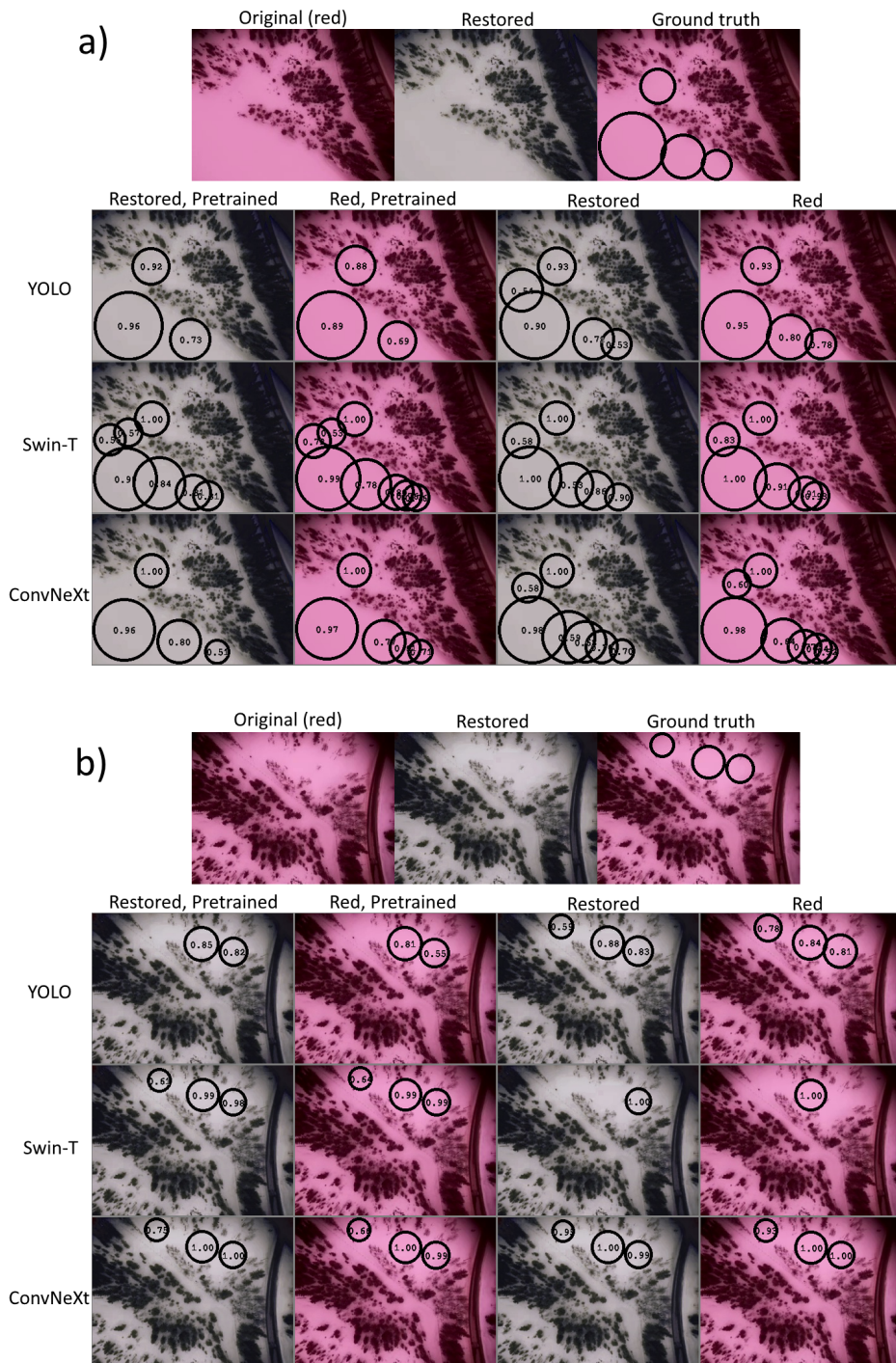


Figure 11

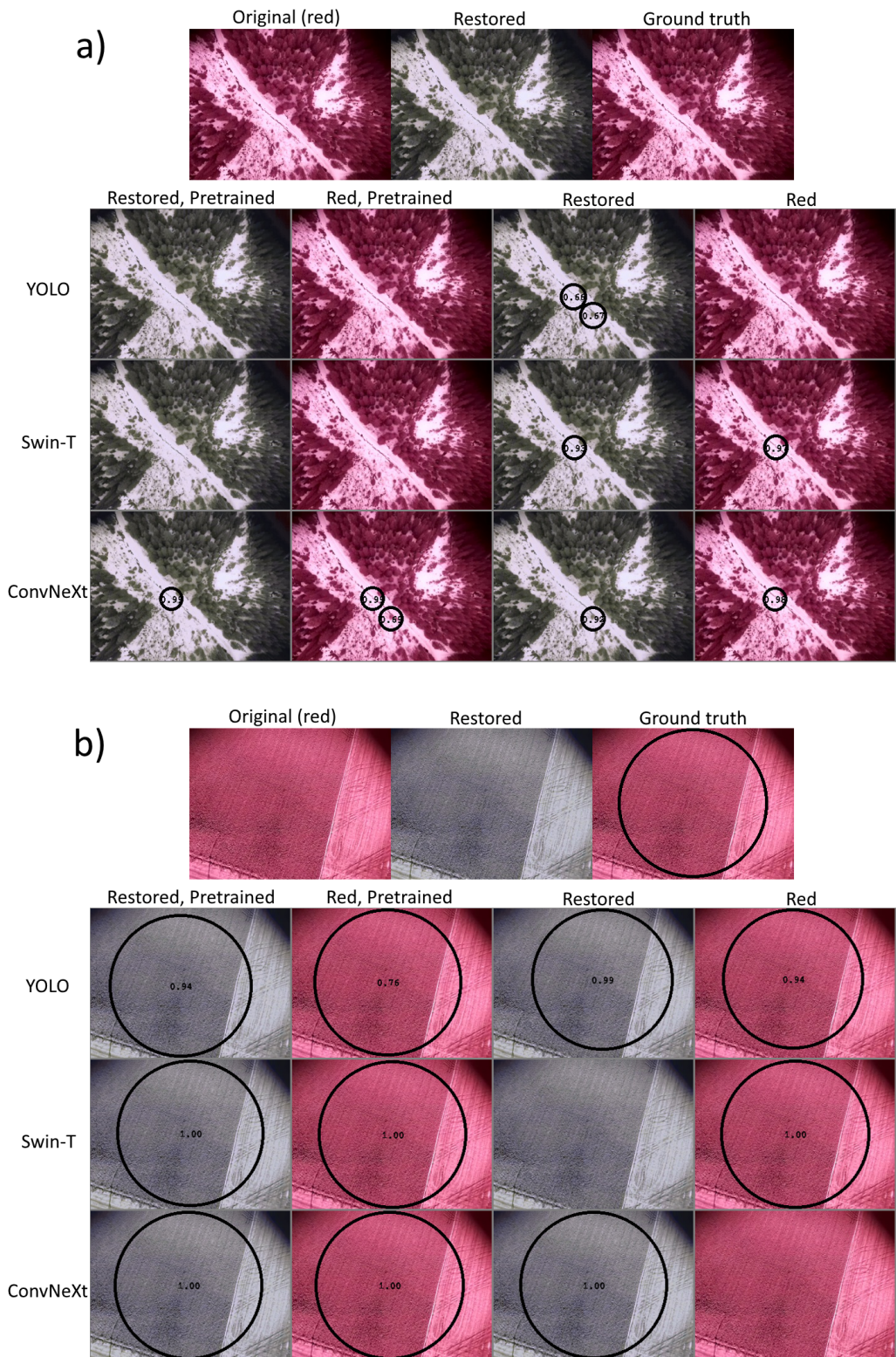


Figure 12

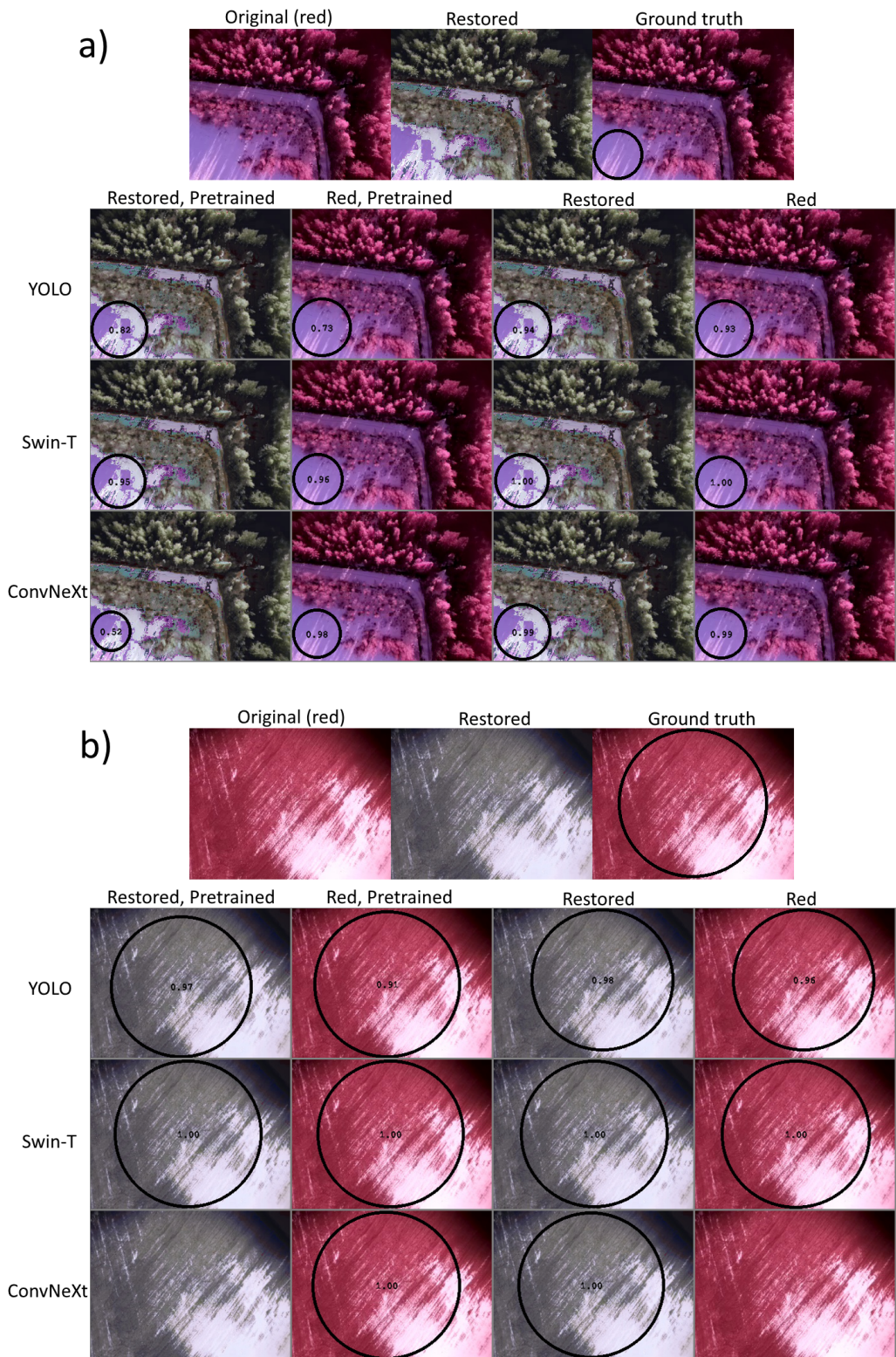


Figure 13

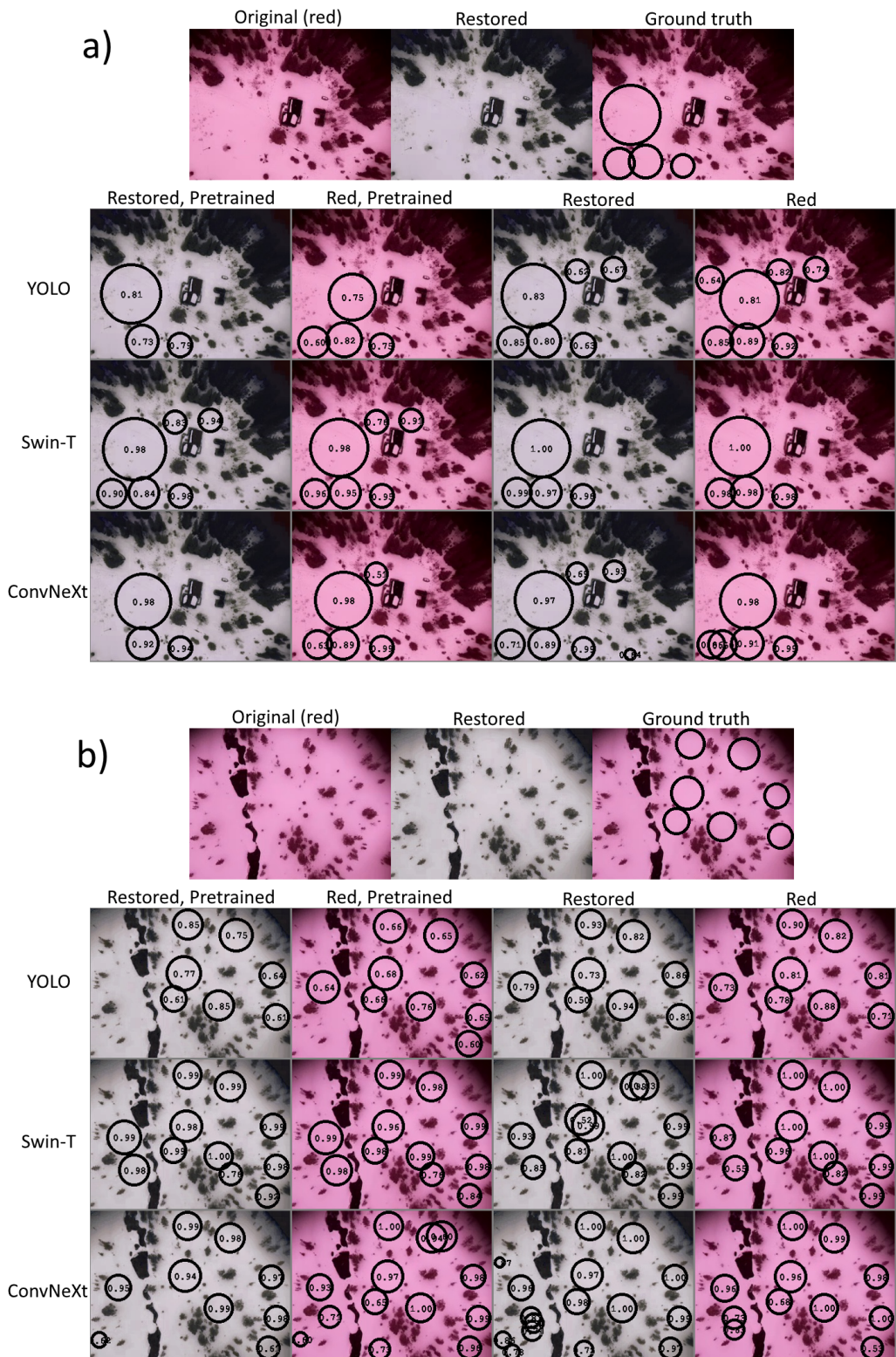


Figure 14

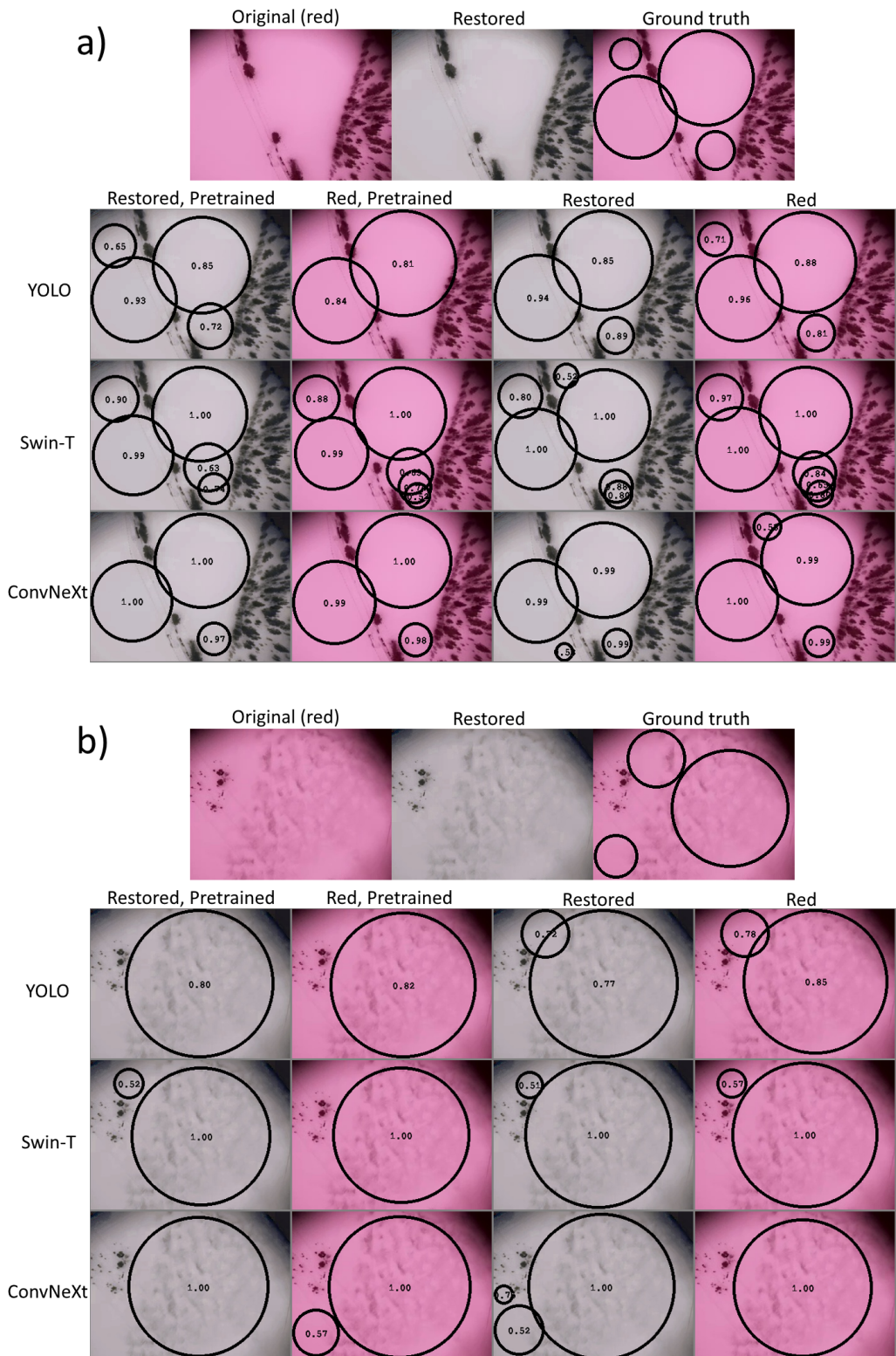


Figure 15

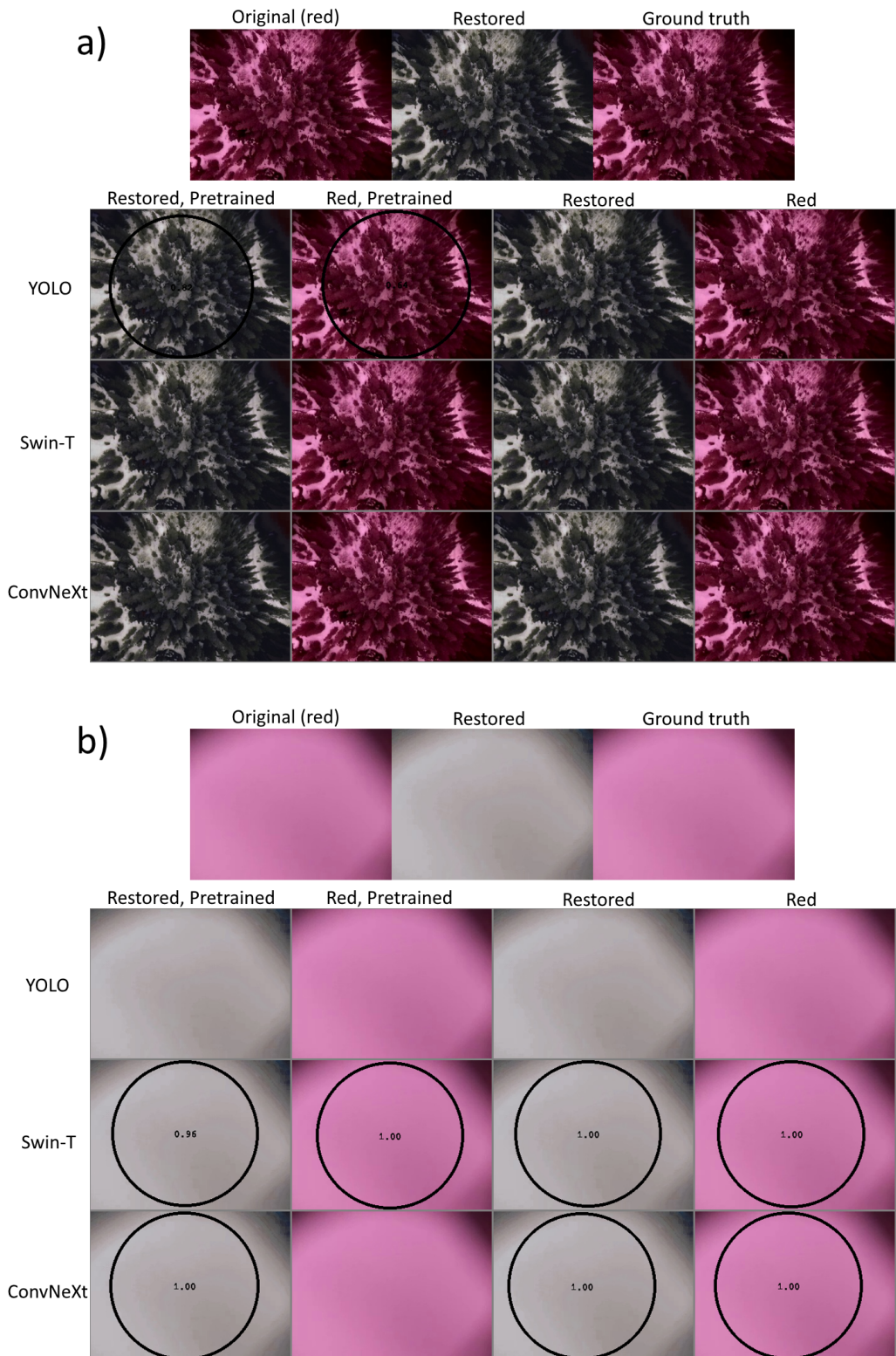


Figure 16

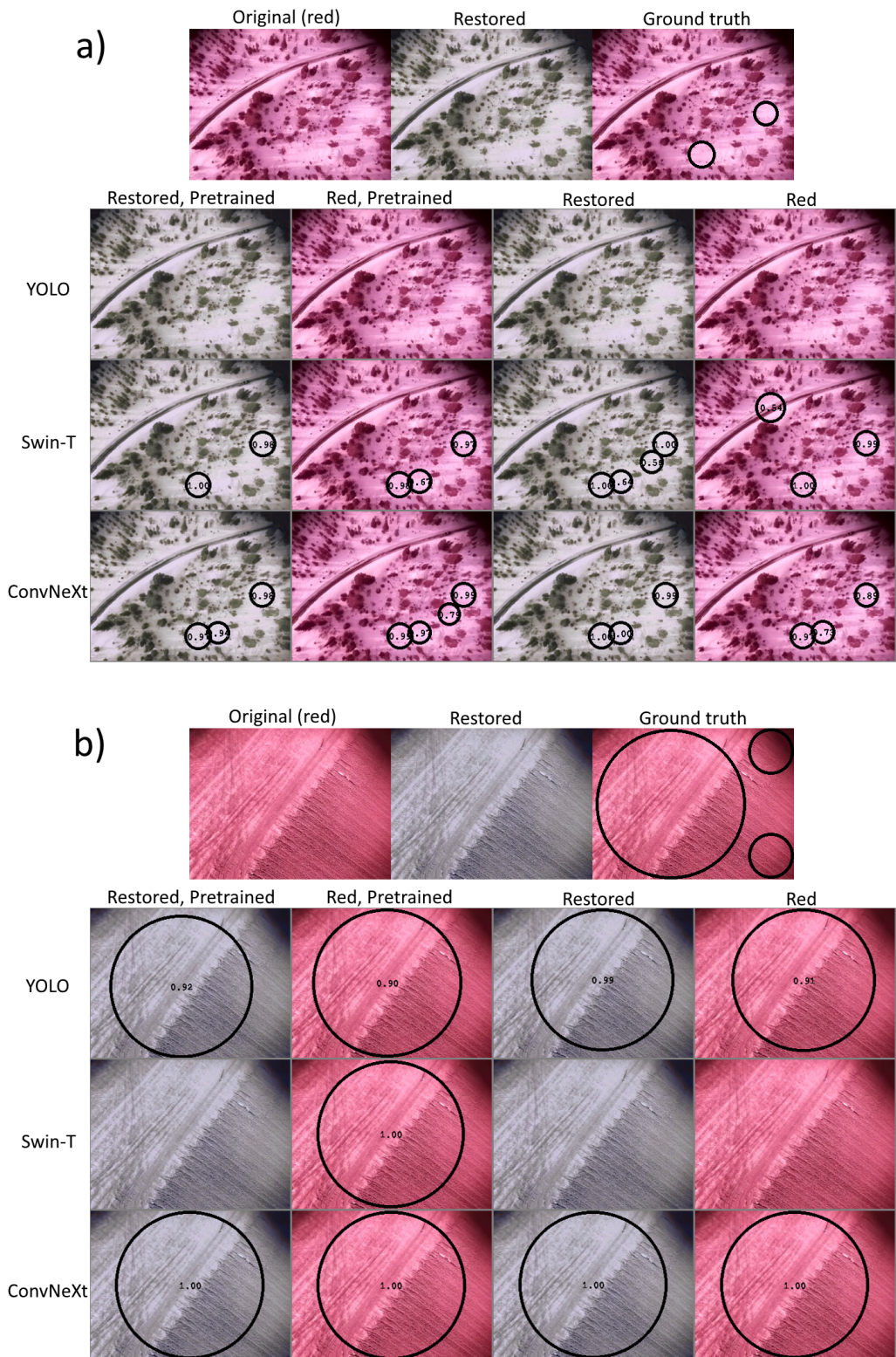


Figure 17

F Results from Segmentation-based System

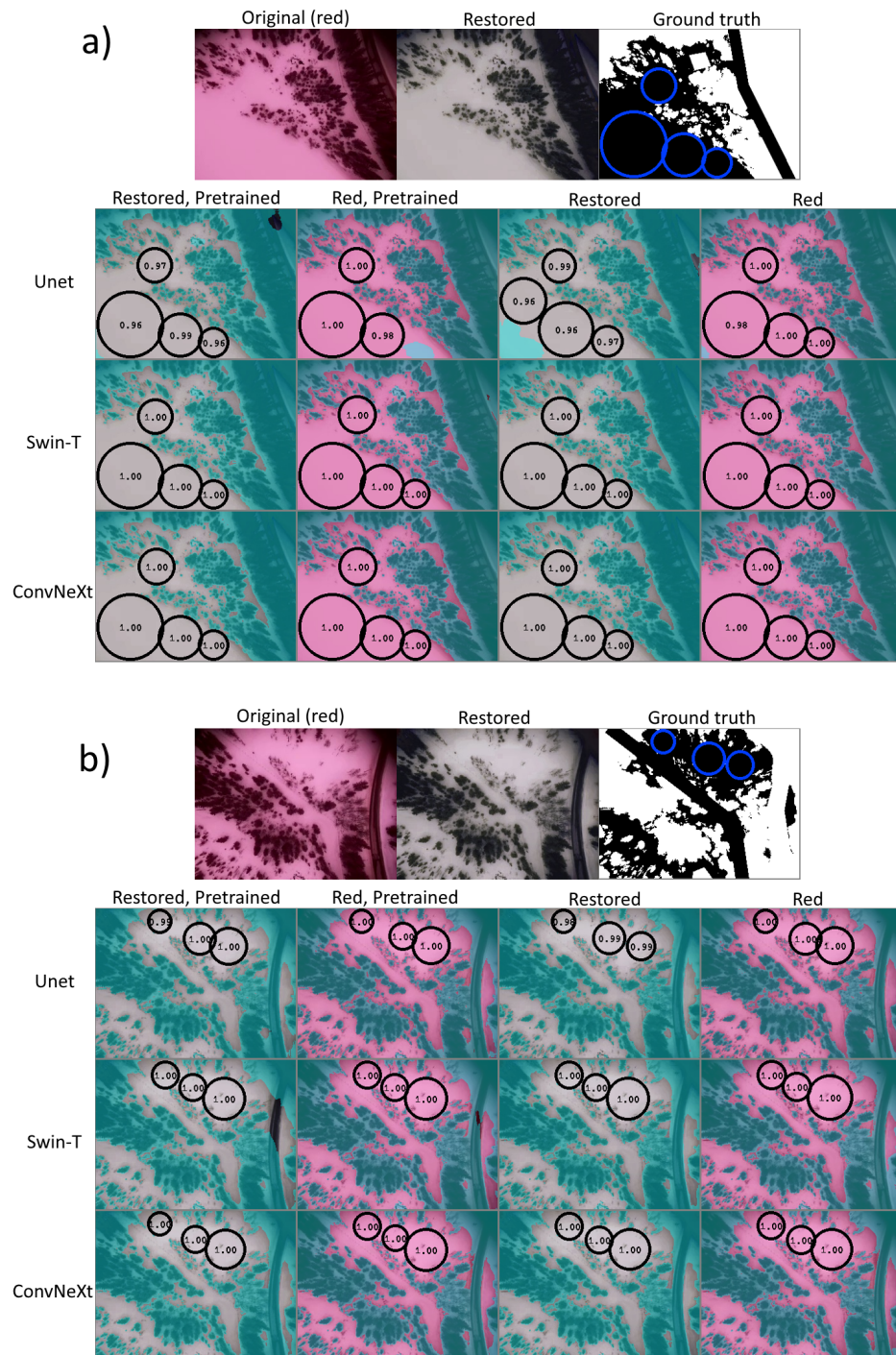


Figure 18

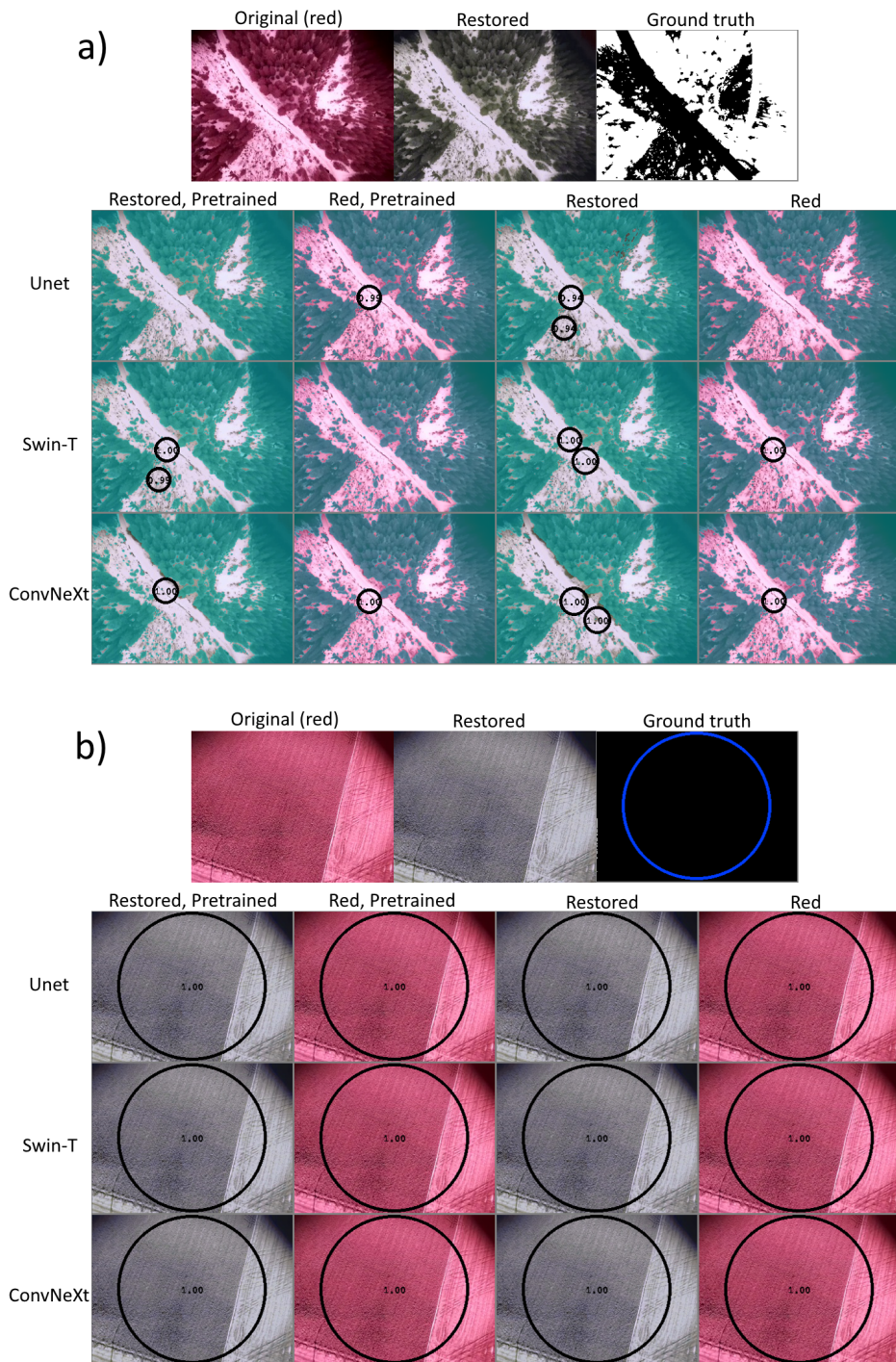


Figure 19

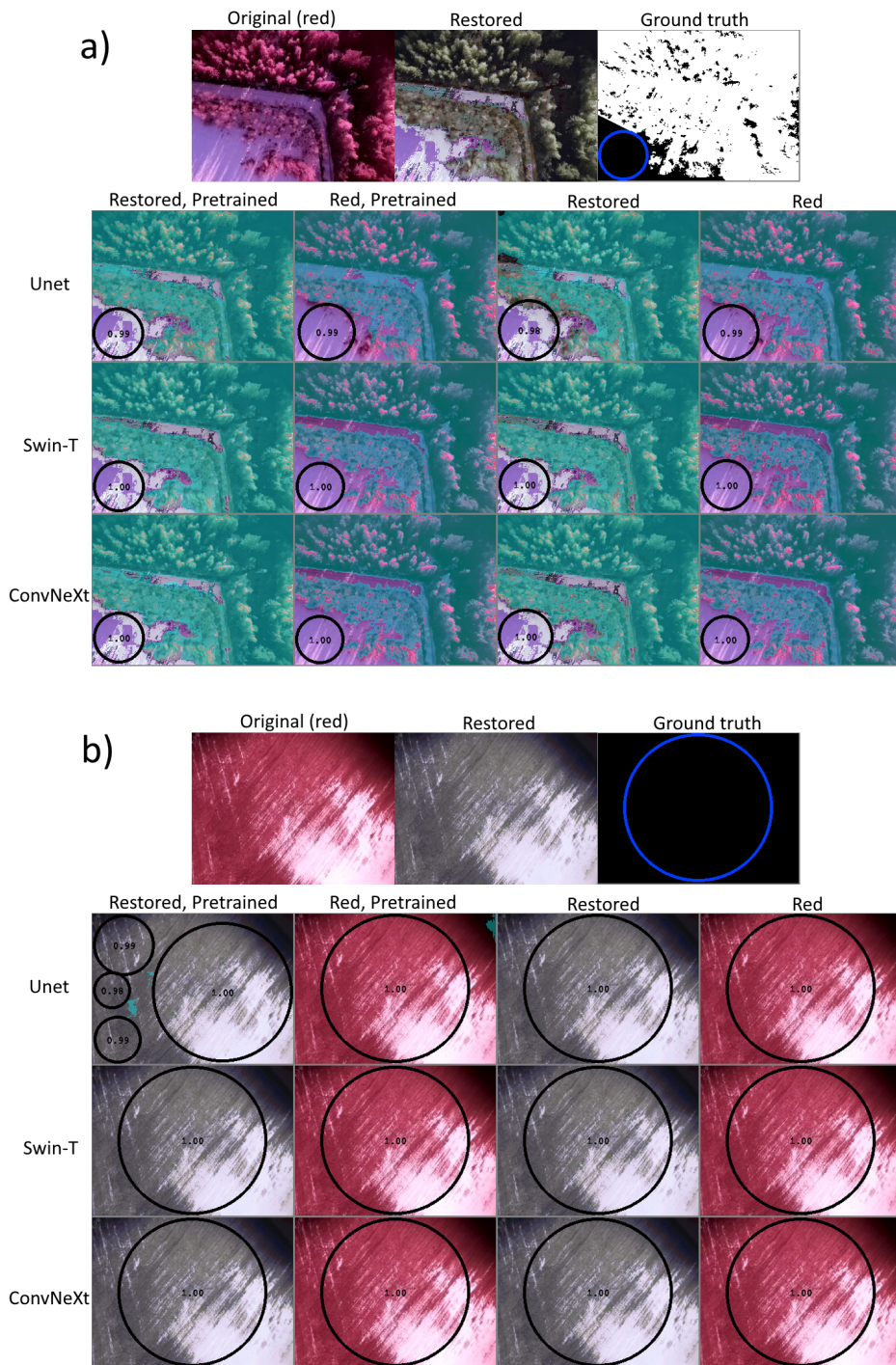


Figure 20

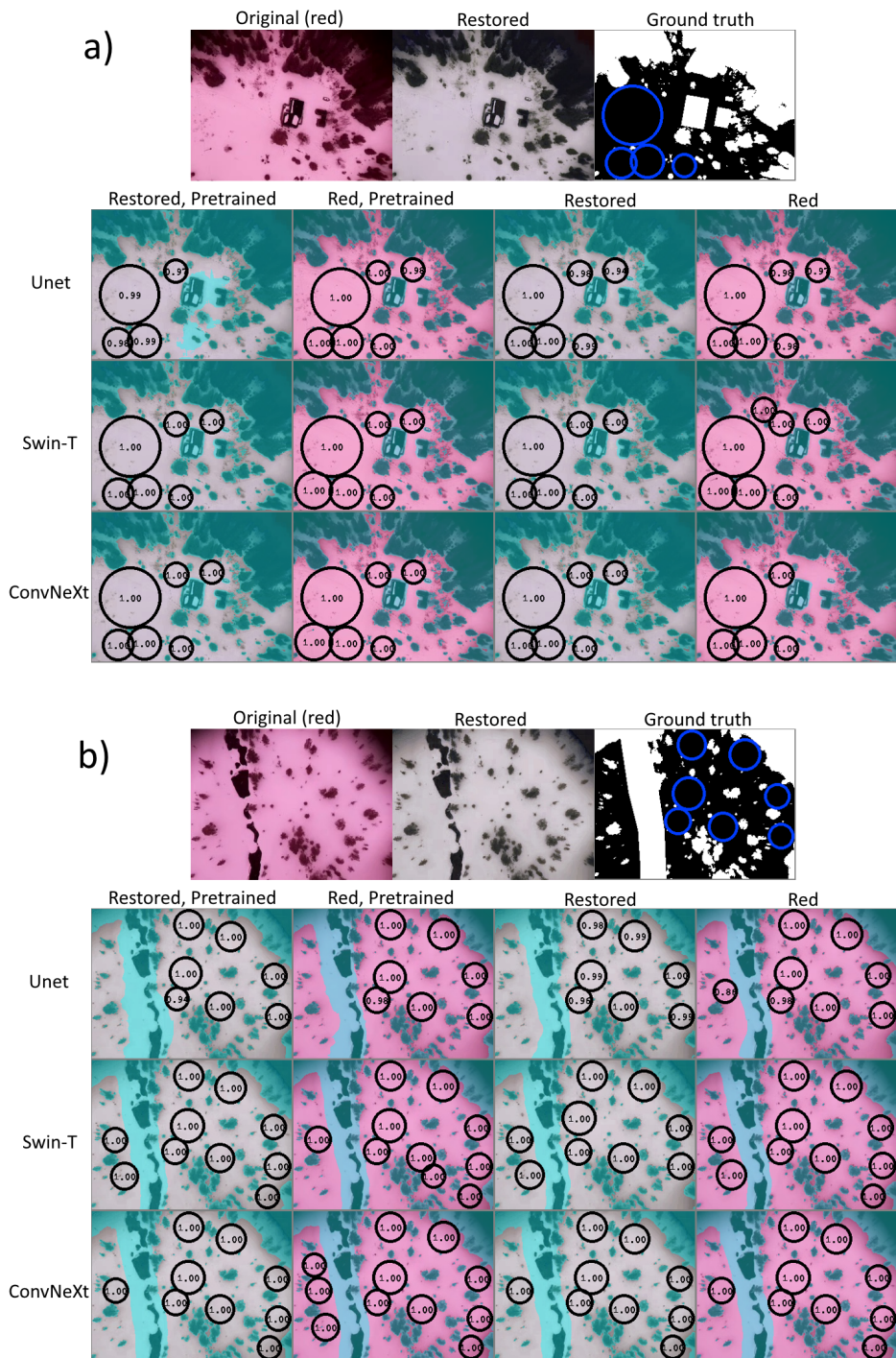


Figure 21

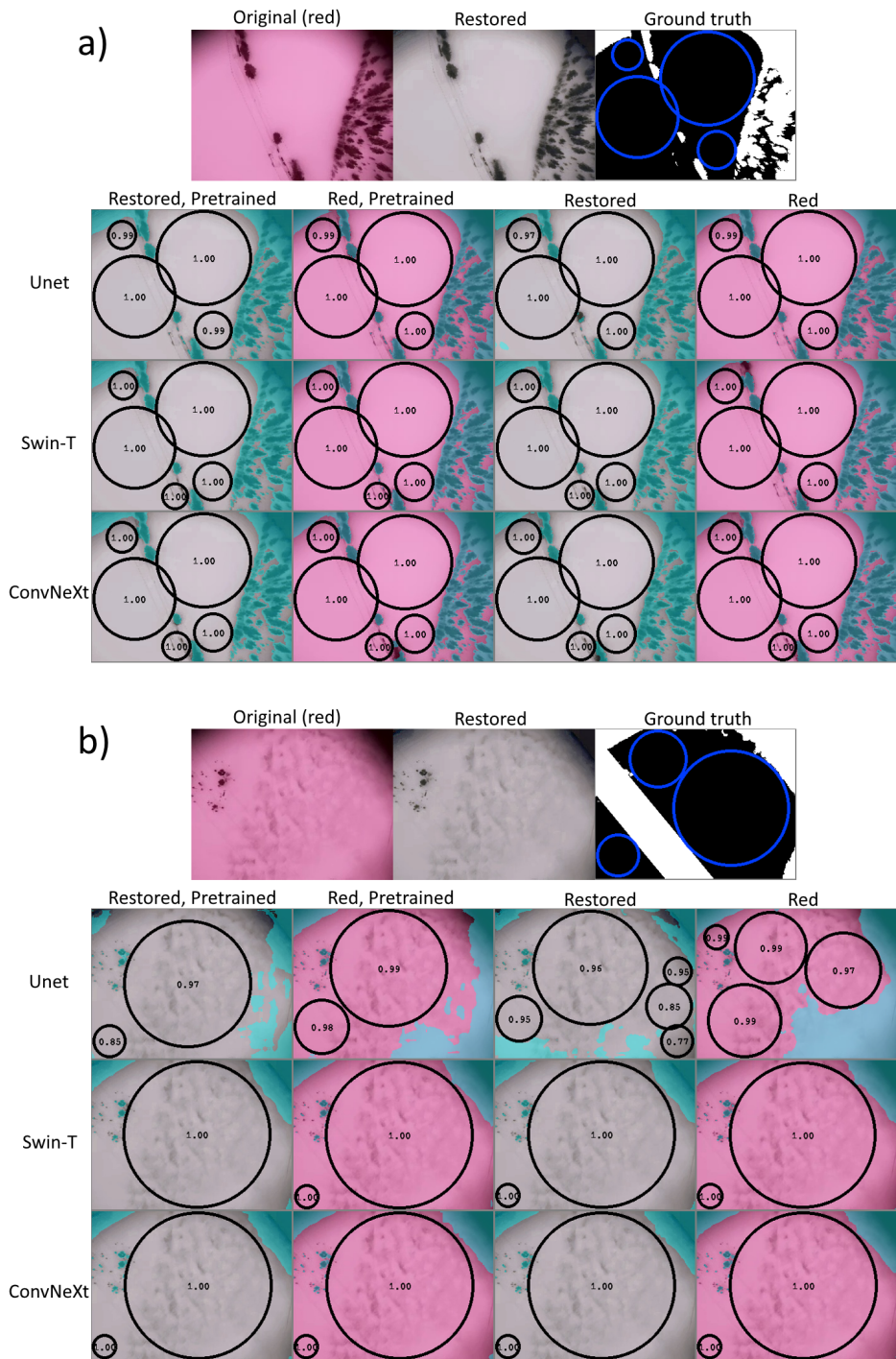


Figure 22

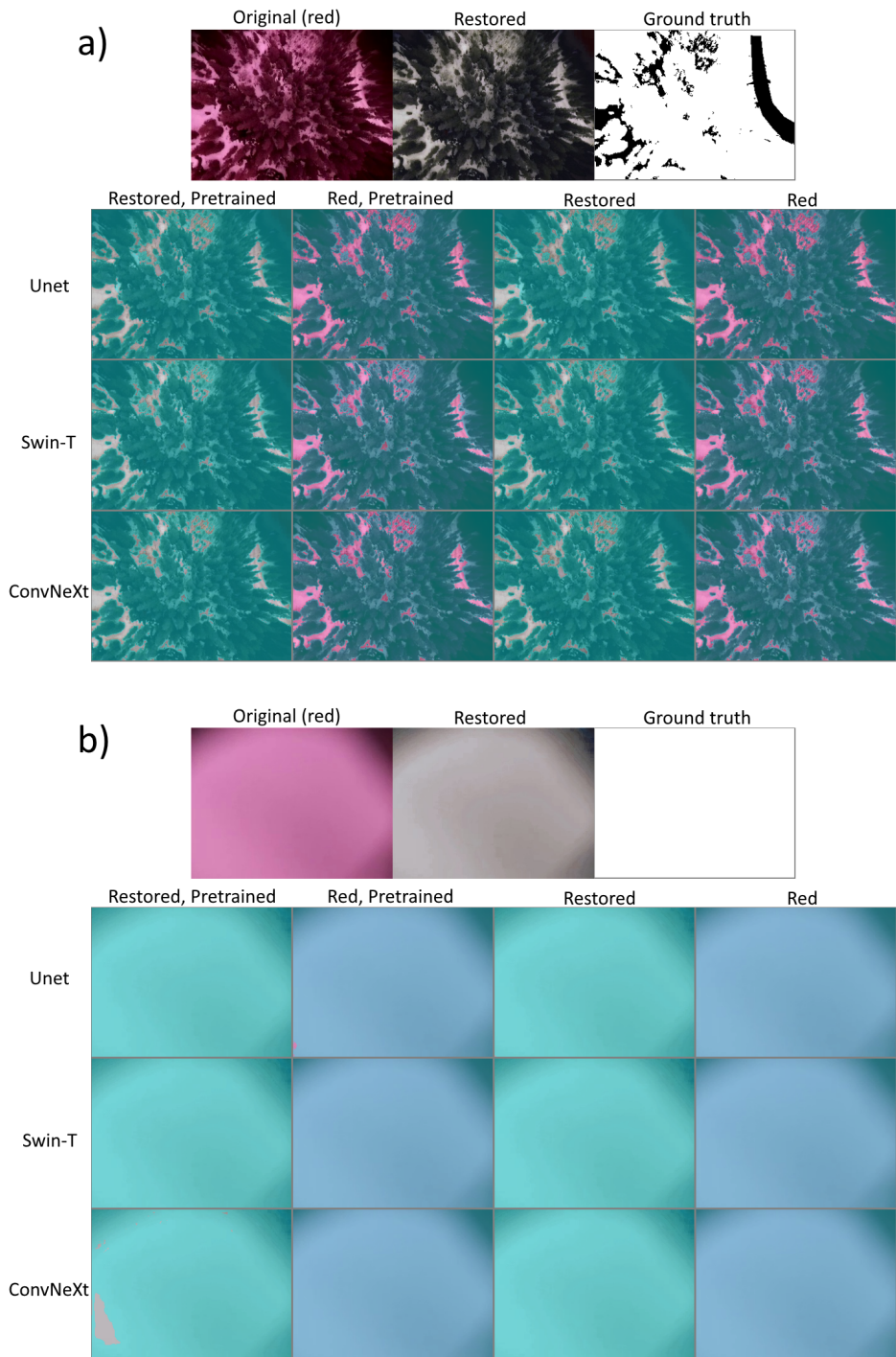


Figure 23

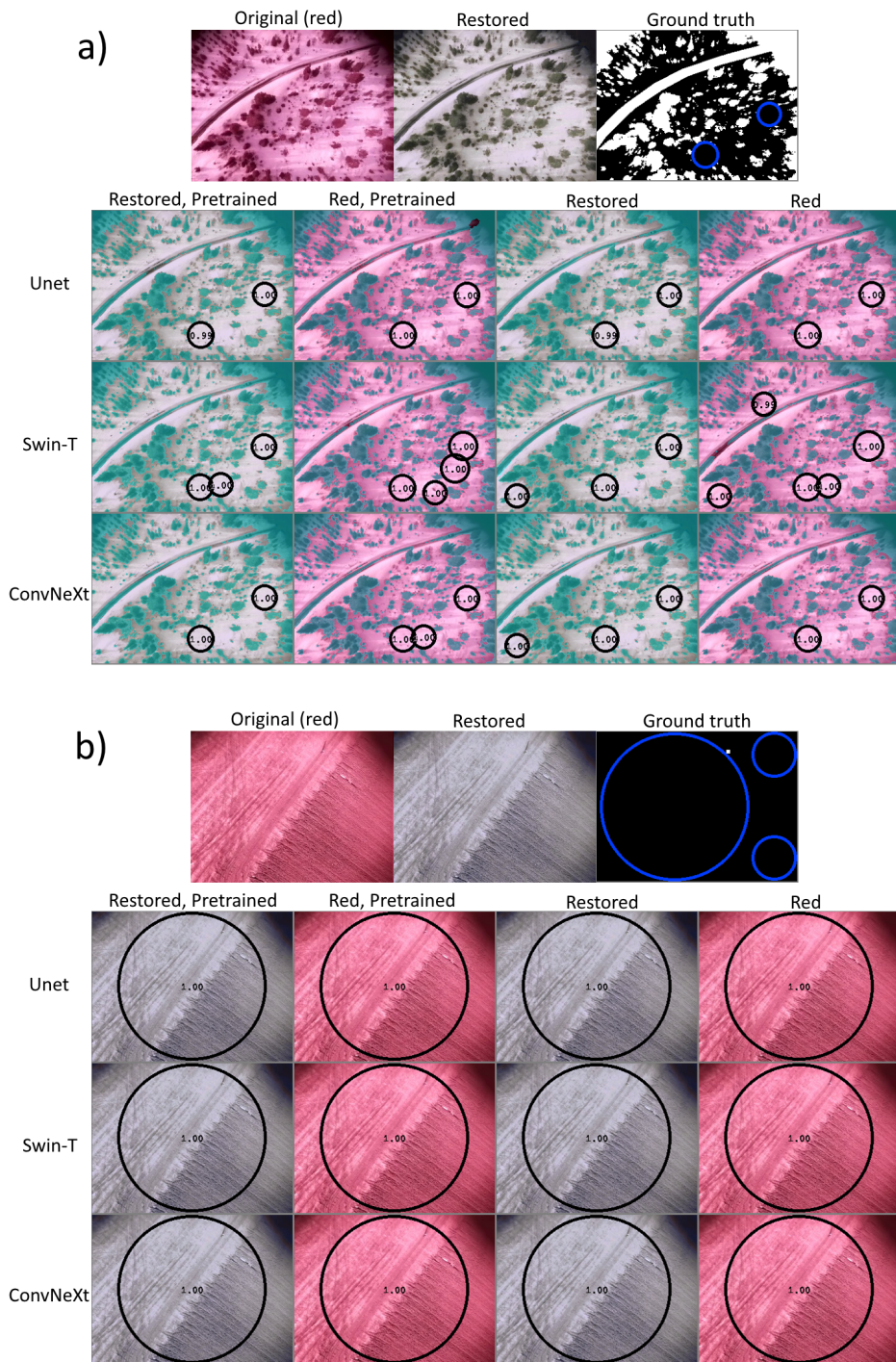


Figure 24

G Rally Point Verification Examples

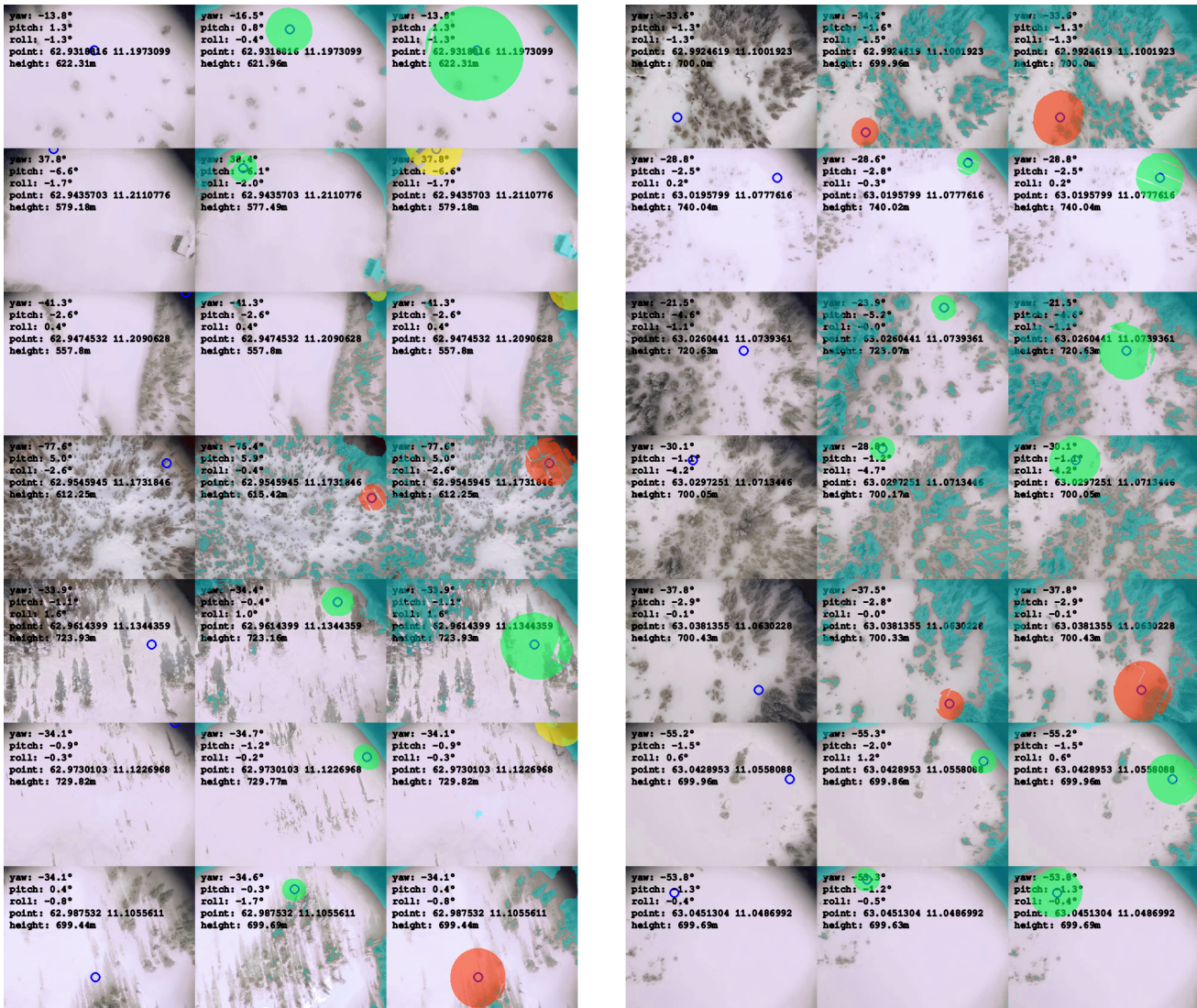


Figure 25

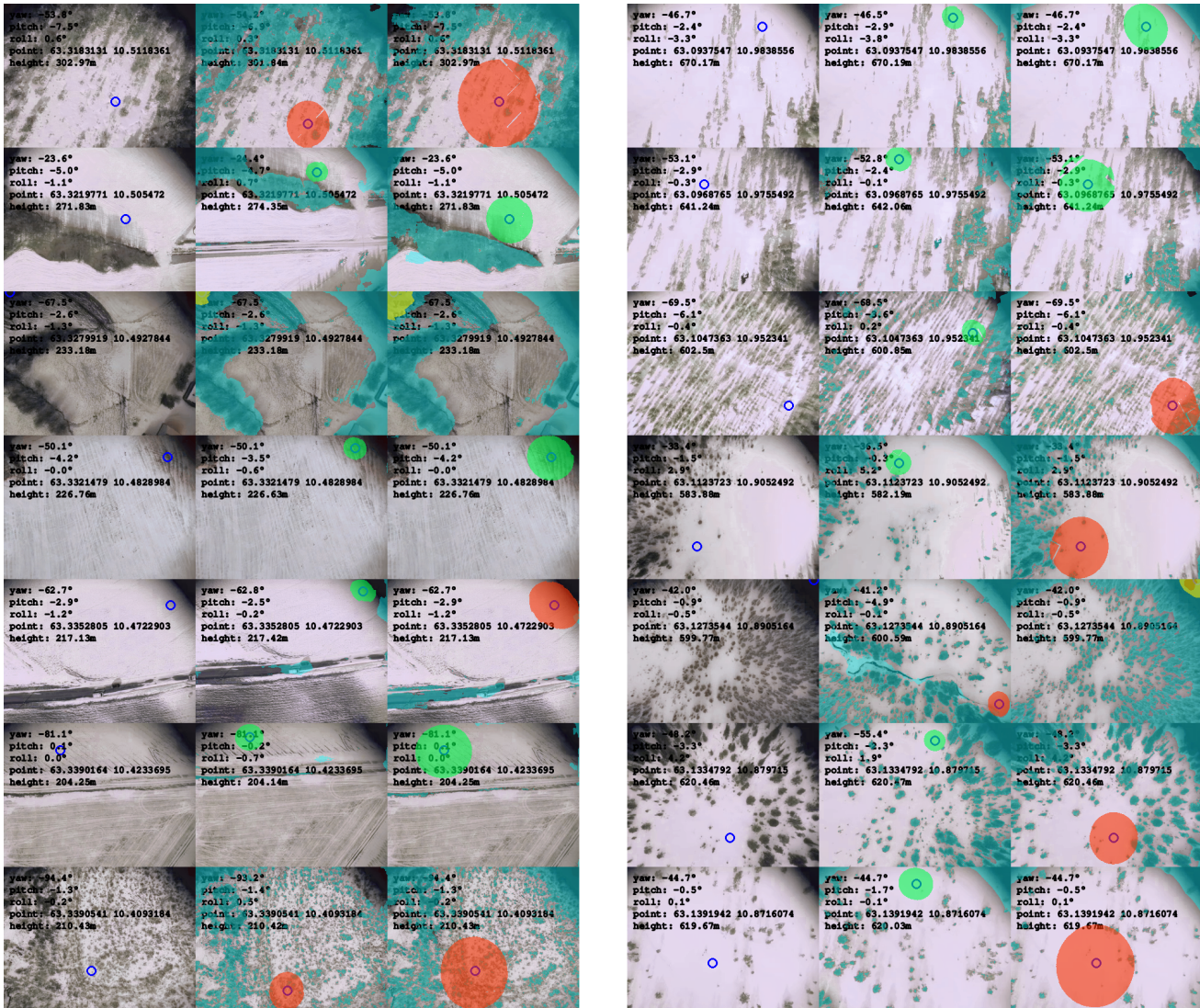


Figure 26

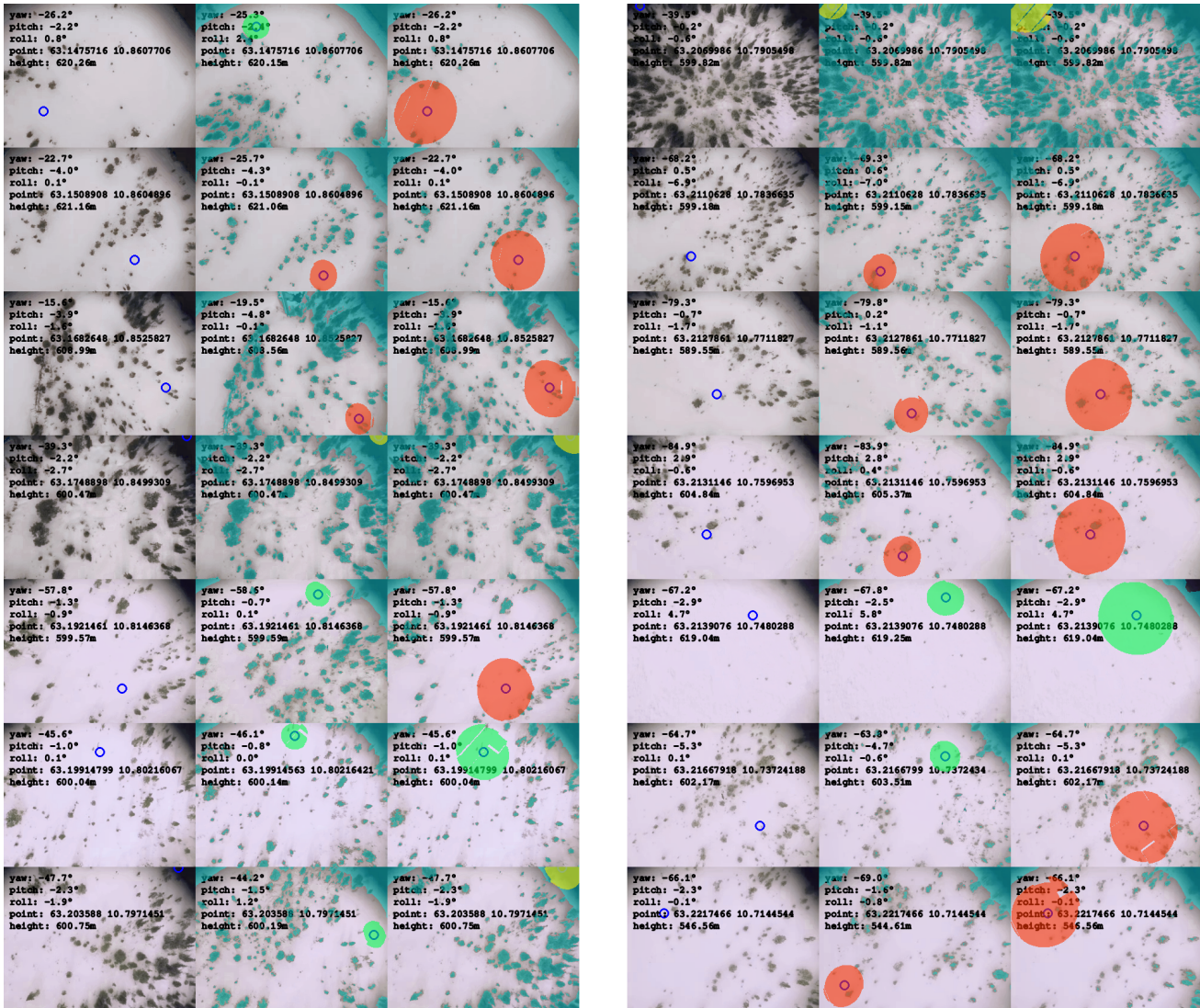


Figure 27

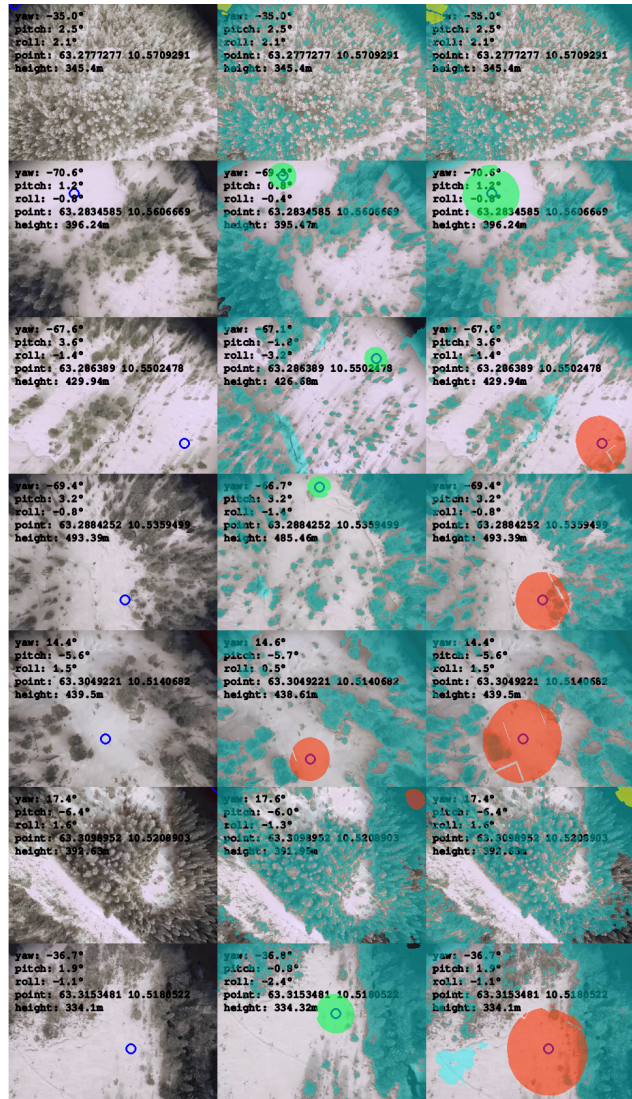
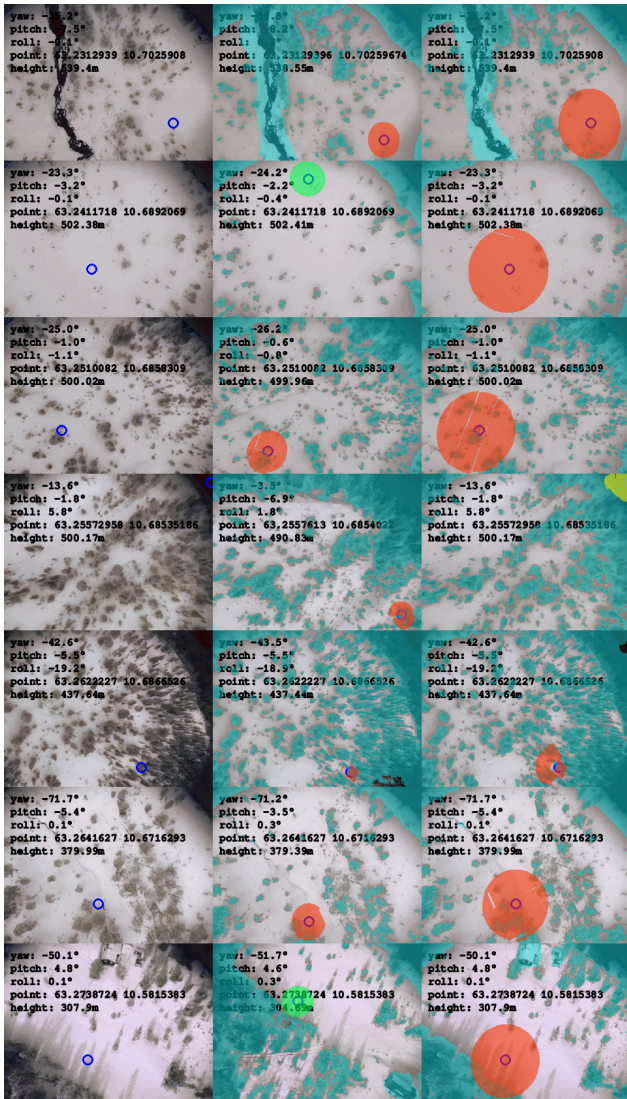


Figure 28

