CHALMERS
UNIVERSITY OF TECHNOLOGY

Automated Simulation Environment for Enhanced Radar Technology Learning

Creating Scenarios using Deep Machine Learning and Stochastic Target Trajectories for Future Radar Simulation

Master's thesis in Systems, control and mechatronics

LOUISE OLSSON
JOHANNA WAGNÉ

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's thesis 2024

Automated simulation environment for enhanced radar technology learning

Creating scenarios using deep machine learning and stochastic target trajectories for future radar simulation

Louise Olsson
Johanna Wagné

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2024

Automated simulation environment for enhanced radar technology learning
Creating scenarios using deep machine learning and stochastic target trajectories for future radar simulation
LOUISE OLSSON & JOHANNA WAGNÉ

Supervisor: Erik Forsberg, Saab Surveillance
Examiner: Tomas McKelvey, Electrical Engineering

Master's Thesis 2024
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: AI generated image of a radar simulation

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

Automated simulation environment for enhanced radar technology learning
Creating scenarios using deep machine learning and stochastic target trajectories for future radar simulation
LOUISE OLSSON & JOHANNA WAGNÉ
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Simulations contribute to reduced resource requirements, faster testing of new equipment, and an often overlooked advantage: enhanced learning through interactive scenarios, immediate feedback, and practical experience in a safe environment. While the conservation of resources contributes to sustainability, simulations also provide a quicker and more visual way to learn, which is particularly beneficial in complex fields such as radar technology. This accelerated learning process can expedite new employee onboarding, enabling more time to be dedicated to critical tasks where most needed. An automated simulation environment that generates new scenarios each use would further enhance these benefits. This study explores the development of such a system and how to implement it in the radar field.

The work consisted of three main parts: creating a 3D environment based on real-world locations using deep machine learning, implementing automated target spawning within the 3D environment, and designing a simulation to demonstrate radar concepts for educational purposes. The creation of the environment requires a terrain map to predict height map and terrain type, achieved using semantic segmentation and a U-Net structure. The data from the network includes information about the terrain's elevation and the corresponding surface color based on the terrain type. This data serves as the basis for creating a 3D environment using a Python 3D plotting library. Targets are created based on the environment, considering constraints such as maximum velocity, turn rate, and collision avoidance. Combining all the parts was a substantial aspect of the work. Before the system can function solely by inputting a terrain map, further improvements are necessary in the form of automatic data transfer and further dynamics implemented for the targets. Currently, the transfers between the main parts still need to be done manually. The goal is for the system to operate as an automated chain where one part feeds the next part with the necessary information.

In summary, the idea of developing a fully automated simulation has been demonstrated to be effective through the tests conducted in this thesis. While the results of the project can support future research in the field, still many challenges await, such as ensuring that all parts of the simulation can automatically transfer data.

Keywords: radar, simulation, semantic segmentation, 3D environment, trajectories.

Automated simulation environment for enhanced radar technology learning
Creating scenarios using deep machine learning and stochastic target trajectories for future radar simulation
Louise Olsson & Johanna Wagné

Acknowledgements

First and foremost, we extend our heartfelt gratitude to our supervisor and examiner, Tomas McKevley, for his guidance throughout this project. We also wish to thank Erik and Miriam at Saab for their ideas and knowledge, which have greatly supported our work. A special thanks goes to Ingmar Bergman for crafting the memorable scene in "The Seventh Seal." Reciting the lines "Vem är du?" and "Jag är döden" has been a source of inspiration and resilience during the most challenging moments of this thesis. Finally, we'd like to thank each other. These past five years would have been far less enjoyable without one another's support and camaraderie. So thank you Chalmers, for bringing us together!

Louise Olsson & Johanna Wagné, Gothenburg, May 2024

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

Adam	Adaptive Moment Estimation
ANN	Artificial Neural Network
BCE	Binary Cross-Entropy
CCE	Categorical Cross-Entropy
CNN	Convolutional Neural Network
MAE	Mean Absolute Error
MSE	Mean Squared Error
NaN	Not a number
Radar	Radio Detection and Ranging
ReLU	Rectified Linear Unit

Contents

List of Figures

List of Tables

1
Introduction

Recently, the modern battlefield has evolved in many aspects, and with the development of new technology, the need to test equipment has grown significantly. Cutting-edge technology, such as airborne radars, is expensive to develop, and conducting physical tests for every modification would be both time-consuming and cost-inefficient. Simulations enable faster development of radar technology without the need for extensive physical resources. Additionally, simulations serve as valuable tools for education and for testing concepts during the development stage. In this context, the introduction will present the background, aim, and limitations of the master thesis.

1.1 Background

Innovation and efficiency are pivotal in engineering. Engineers require tools to assist them during the development and testing phases to achieve optimal performance. One such tool that has recently gotten much attention is simulation [1]. Simulation is a broad subject that can be applied in various fields. Some examples of flows that can be simulated are weather forecasting, virtual prototyping, and factory operations. It is often used in the automotive and aerospace industries, where it contributes to fewer risks in terms of safety and finances. When using a simulation instead of physical tests there is less need for resources and therefore better from a sustainability perspective. Other advantages of simulation are safety, the ability to test something before investing time into building it, and the exploration of new concepts.[2]

In all simulations, scenarios are needed to describe the problem to be simulated. A scenario can be described as a series of actions over a time period. The scenario should describe situations that could occur in reality. If the scenarios succeed in representing reality, products tested in such a simulation environment are more likely to perform well in reality. However, most scenarios do not succeed very well in this task since there are often parts that are excluded [3]. Where simulations are used as a tool for learning, for example in a flight simulator, the value of the simulation corresponds to the variety of the scenarios. If the same scenarios are replayed, the learning curve will stagnate. It is therefore a challenge to make the scenario generation automatic while still preserving diversity and realism.

In the case of simulating a radar, an environment, targets, and a model of the radar are needed. With these three blocks, it is possible to capture the essential components of a radar simulation and their interactions. They form the foundation

for simulating radar systems and evaluating their performance in various scenarios. The scenarios in radar simulations typically consist of a radar being placed in a setting and the targets moving around in the setting for the radar to detect.

1.2 Purpose and objectives

The purpose of the thesis is to explore the use of simulation in the radar field, focusing on using it as a tool to educate and demonstrate different concepts. It can also be developed further in order to be used as a system for testing different modules in the development of the radar. Conducted in collaboration with Saab, this study aims to simplify the on-boarding process for new employees, given the complexity of radar technology. To be able to achieve this, the following research objectives were formulated:

- Develop a 3D simulation environment inspired by real-world settings and generated through machine learning.
- Implement automated and stochastic target spawning and navigation within the 3D environment.
- Design the simulation to effectively visualize radar concepts for educational use, while laying the groundwork for a comprehensive simulation system in the future.

1.3 Limitations

There were two limitations encountered when working with this thesis. First, due to security precautions related to Saab Surveillance and their military operations, data from their products was not utilized. The data used was instead open source to alleviate the issue with the publication of the data and results. Second, the access to computational resources was limited, which restricted the time and power for the training of the networks.

1.4 Related work

Radar technology is well-researched, and much has happened since its development took off in the 1930s. This project is divided into three main parts, which are explained in Section 1.2. The work done in the first two parts is somewhat similar to research of other kinds.

The first part of the project involves using deep machine learning, specifically semantic segmentation, to create a realistic 3D environment. A study published in 2022 performed a similar task. The study researched a method to estimate relative height and perform semantic segmentation simultaneously using airborne RGB images as input. The study uses transformer-based learning and developed dynamic weights to learn both tasks in parallel. The model uses the relationship between height and segmentation to achieve higher accuracy, knowing an object's class in

the image can aid in estimating the height [4]. The multi-task learning approach is an interesting take that differs from the approach in this project since the tasks are similar but in this master thesis, the tasks are performed separately.

Generating trajectories that move realistically and consider the surroundings is the second part of the project. Generating stochastic movement can be done in various ways. A paper published in 1987, explains an approach to generate trajectories that has been used in many other applications since then. It investigates how to simulate group movements realistically, such as a flock of birds. The algorithm simulates every bird as an individual who acts upon certain rules to navigate an environment. The paper discusses different ways to implement collision avoidance, for example, a method called steer-to-avoid. [5]

2
Theory

In the following section, the technical tools and methods used in the thesis will be covered in order to ensure understanding of the work described later in the report.

2.1 Radar fundamentals

Radar stands for Radio Detection and Ranging. A simple radar system starts with a transmitter generating an electromagnetic pulse that is transferred to an antenna through a transmit-receive switch. The pulse is then sent out in the direction of the antenna at the speed of light. When the pulse collides with an object it scatters, leading to the radar receiving some of the scattered signal back. This signal is captured by the antenna and through the transmit-receive switch it reaches the receiver. The signal then goes through a signal processing block in order to enhance it and reduce noise.[6] After the signal processing block, it goes into the tracker, which associates detections with targets and creates tracks. These tracks are then presented on a display for the operators to see.

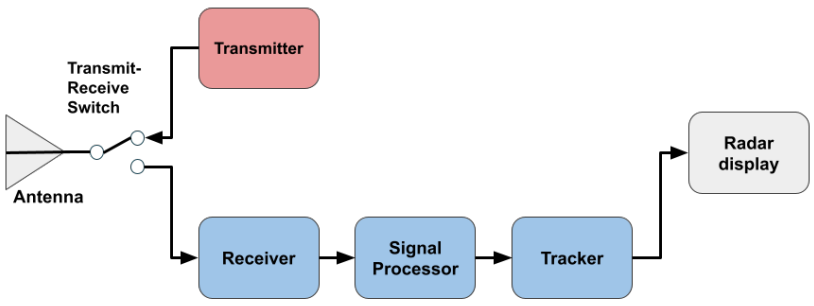


Figure 2.1: A simplified radar system depicted in a block diagram.

2.1.1 Tracker

The tracker is an important part of the radar system, associating radar detections with distinct tracks. This is especially useful when the radar detects multiple targets, allowing it to differentiate and follow them, providing essential data for antenna positioning [6].

The tracker will identify which target each radar detection belongs to at each time instance while also dismissing detections believed to be false alarms. By creating tracks for the targets, it is possible to estimate the current position and velocity, while also predicting future states. It associates new detections with existing tracks, creates new tracks for unassociated detections, and eliminates outdated tracks. Updated tracks refine target trajectories, with predictions including position, velocity, and potential acceleration. Detections within a specified target tracking window, a bounding box around the predicted target's position with specified dimensions, are associated with the target.

Detections unassociated with existing tracks will trigger the initialization of new target tracks. These new tracks will remain unconfirmed until subsequent radar measurements are associated with the same track, preventing the creation of false tracks. A common approach is to confirm the new track if a set number (M) of detections in the last number (N) of updates are successfully associated. Similarly, tracks with no recent detections are eliminated when the target is out of scope or uncertainties are too high although, the tracks are not eliminated instantly when no detection is done, to account for possible missed detections.

Filters are used to estimate the next state of the tracks and smooth the trajectories. The most commonly used are the Kalman filter and alpha-beta filter [6]. The alpha-beta filter is simple but with its use of constant covariance error, it can not handle maneuvering targets. Contrarily, the Kalman filter will model the measurement errors and dynamics of the target, better managing noise and uncertainties.

2.1.2 Search patterns

A radar uses different search patterns to scan the desired area. Sometimes it is necessary to make a wide sweep to search a large area, but when a target is detected, it becomes more important to fix the radar on that point. It is also possible to combine these two modes which is called a search-while-track mode. Different types of radars can behave in various ways and have limitations in how they change the direction of their search lobe. The scan can be done manually by altering the orientation of the antenna or electronically, depending on the radar type.

Scanning differs from tracking because tracking focuses on the range, direction, and velocity of the target to predict its future position, whereas scanning focuses on detecting a target's position and feeding this information to the tracker, as explained in section 2.1.1. A wide radar beam width can be used to maximize coverage, but this results in poor angle resolution. Conversely, a narrow beam width offers better angle resolution and can provide a more precise position estimation of the target, but the area it covers is smaller. Therefore, it is necessary to alter the direction of the beam to cover more area while still maintaining a good estimation of the detected target's position.

2.2 Introduction to target flight simulation

The theory presented in this section describes the concepts and relationships essential for generating the target trajectories in the method of the project. Not all dynamics for flight are mentioned due to simplification being done further on.

2.2.1 Turning dynamics

The **radius of a turn** refers to the distance from the center of a turn to the path of an airplane, indicating its size. Pilots need to know the radius of a turn to ensure safe maneuvering of an aircraft [7]. This information helps to maintain control and avoid the stress of the airplane during flight. The radius of a turn is defined as:

$$r = \frac{v^2}{11.26 \tan(b)} \tag{2.1}$$

Where v is the true airspeed of the airplane (knots) and b is the bank angle (degrees).

The **angular velocity** for an object moving along a circular path, is the rate of change of the angle. It is mathematically expressed as:

$$\omega = \frac{\Delta\theta}{\Delta t} \tag{2.2}$$

where ω is the angular velocity, $\Delta\theta$ is the angular displacement and Δt denotes the time interval for the displacement. There is a direct relationship between the linear velocity and the angular velocity for the object moving in a circular path, expressed as:

$$\omega = \frac{v}{r} \tag{2.3}$$

Where r is the radius and v is the linear velocity. The linear velocity is therefore proportional to the angular velocity and the distance the turn center. Consequently, the velocity of the object will be influenced by the distance from the rotational center. [8]

An object is moving in a **uniform circular motion** when traveling around in a circular path with constant speed. Even though the speed is constant, the direction of its velocity vector is constantly changing. The change of angle for the uniform circular motion corresponds to the change of heading for the moving object in the same time interval. Figure 2.2 visualizes this relationship and how the parameters are updated for the motion. [9]

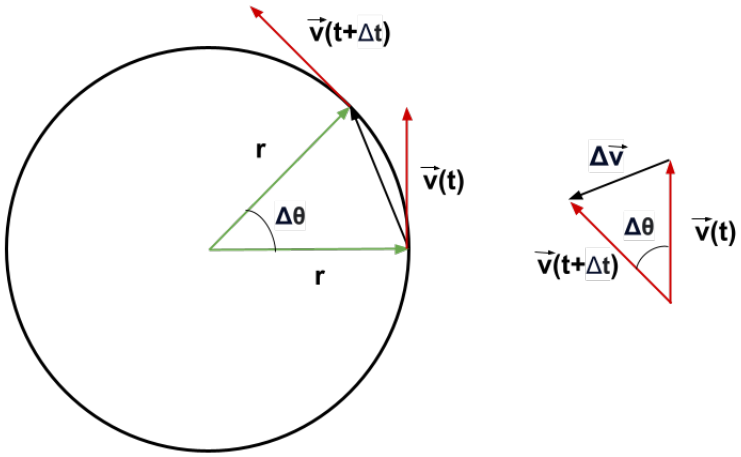


Figure 2.2: Visualization of uniform circular motion, showcasing the relationship between the change in position, angle, velocity, and time.

2.3 Core principles of deep machine learning

A machine learning algorithm is designed to acquire knowledge from data through learning. These machine learning algorithms can be divided into two broad categories; supervised and unsupervised. The categories differs on what the algorithms have access to during the learning process. A **supervised** learning algorithm is when a dataset containing labels is used. The dataset includes features, as well as the associated label for every example. An **unsupervised** learning algorithm only uses a dataset that contains features. Deep machine learning is a subclass of machine learning. It exploits neural networks with many layers to extract feature representations [10].

2.3.1 Artificial neural network

Artificial Neural Networks (ANN) are systems inspired by the functionality of human brains. Learning is a big part of the human brain's dynamic development, the same goes for ANNs. [11] These networks are utilized for problem-solving and machine learning, imitating the brain's recognition ability, and making them popular tools in various situations. Unlike traditional, non-machine learning systems where importance is explicitly defined in the code, in neural networks, significance is ingrained within the network itself through the learning process.

The connections between neurons represent knowledge, and the strength of these connections is determined by weights tuned during the training process. Through exposure to different situations during training, the network learns and adapts. After training, the network yields responses based on the given input. [12]

The architecture of a neural network can differ depending on its purpose, but generally, it consists of layers, neurons and weights. The structure of the network includes an input layer, hidden layers and an output layer. The input layer is where

we feed the network with information and it is in the hidden layer the computations are performed. Patterns are found in the hidden layers which are used to make a decision in the output layer. [13]

2.3.2 Activation functions

Activation functions are applied to the output of every neuron in the hidden layers and also to those in the output layer. These functions introduce non-linearity, allowing the network to grasp complex patterns within the data. A network without an activation function will just be able to learn how to do linear transformations, limiting the performance of the network [14]. The most commonly used activation functions in the hidden layers for deep learning tasks are the Sigmoid, ReLU, Tanh, and leaky ReLU functions.

Commonly, a single activation function is applied to the neurons of all hidden layers. However, another activation function can be used for the output layer. The function chosen for the output layer depends on the task at hand, as it will transform the output from the network to a suitable output for the specific task. For the classification tasks, the most common activation functions in the output layer are Sigmoid and Softmax. [15]

The **Sigmoid** activation function, also called the logistic function, is an activation function that will map the output from a neuron into the range of 0 and 1[15]. The mathematical expression for the function is

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

With the interval of the function, the output can be seen as a probability and works well for binary classification. However, the Sigmoid function has some limitations. The S-curve and squishing nature of the function leads to extremely small gradients for values close to 0 and 1. This might cause a case of vanishing gradients, where the gradients are exceedingly small, hindering the network's learning process [10].

The **Tanh** activation function is similar to that of the Sigmoid function but is a shifted Sigmoidal function. It is centered in 0 with values ranging between -1 and 1. The function is represented as follows:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.5}$$

Having the mean of the function at 0 is advantageous for the Tanh function compared to the Sigmoid function, as it assists the network in the backpropagation and has been shown to yield better results [16]. However, the Tanh function does have the same problem with vanishing gradient as the Sigmoid function [15].

The **Rectified linear unit (ReLU)** activation function is a simple yet effective activation function. As shown in its mathematical representation;

$$f(x) = \max(0, x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \tag{2.6}$$

the function will take the max value of 0 and x. Therefore, all negative values of x will be set to zero and the positive part will resemble a linear function. Although its simple nature, it has been shown to outperform the Sigmoid and Tanh functions in some areas due to them being easier to optimize and faster to compute [17]. Since the gradient of ReLU is either 0 or 1, the problem with exceedingly small gradients is eliminated [15]. The function does however have its limitations. With many input values in the negative range, the output from some neurons will become consistently zero. This will lead to the neurons "dying" and limiting the learning of the network. To deal with this issue, a variant of the ReLU function was created, called The **Leaky ReLU** activation function. A small slope is introduced on the negative part of the ReLU, as visible in its mathematical expression:

$$f(x) = \alpha x + x = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases} \tag{2.7}$$

This will guarantee that the gradient never becomes zero, thus ensuring that the neurons are kept active [15].

The **Softmax** activation function is an activation function that will take a vector as input and display it in a probability distribution. The probability distribution is defined as:

$$f\left(x_i\right) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.8}$$

The Softmax function yields a probability for each class, which is why it's often used as the final layer in a network. The sum of all probabilities will always equal 1 from a Softmax function.

2.3.3 Backpropagation

Using a feedforward network to transform an input x into an output $\hat{y}$ is typically referred to as forward propagation, as the information progresses forward. The input contributes information that is utilized in each layer of the network, ultimately producing the output. This process continues during training until it culminates in a scalar cost. Backpropagation reverses the flow compared to forward propagation. It starts with information from the cost and moves backward through the network to calculate the gradient [10].

2.3.4 Loss functions

Loss functions are a critical component of neural networks as they not only assess how well the model fits the training data, but also guide the optimization process. Loss functions compare the predicted output to the actual target values, providing a measure of the network's performance on the training data. Throughout the training loop, the objective is to minimize this loss by adjusting the model's parameters, such as weights and biases, to achieve the lowest possible average loss. In Equation (2.9), the average loss J depends on the weights w and the biases b. By tuning

these parameters, the loss can be minimized. L in the equation represents the loss function and N is the total number of samples.[18]

$$J\left(w^T, b\right) = \frac{1}{N}\sum_{i=1}^{N} L\left(\hat{y}^{(i)}, y^{(i)}\right) \tag{2.9}$$

In supervised learning, there are two categories of loss functions: one corresponding to regression neural networks and one to classification networks. Mean Squared Error (MSE) and Mean Absolute Error (MAE) belong to the regression loss functions. Examples of classification loss functions include Binary Cross-Entropy (BCE) and Categorical Cross-Entropy (CCE).[18]

One of the most common loss functions is the MSE, which is the average of the squared differences between the predicted output and the target output. The formula for MSE is shown in Equation (2.10).

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \hat{y}^{(i)}\right)^2 \tag{2.10}$$

Because of its squared nature, the loss function guarantees non-negative values. The MSE function particularly penalizes significant errors by squaring them, thereby assigning more weight to larger deviations compared to smaller ones, even in cases where there's only one outlier prediction with a large error.[19]

The MAE is a loss function very similar to MSE. The MAE is calculated by taking the absolute value of the difference between the predicted value and the ground truth, and then averaging it over the dataset. The equation for MAE is shown in Equation (2.11).

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{2.11}$$

Like MSE, MAE only yields positive values. The major difference is that MAE assigns constant weight to both small and large errors in a linear fashion, unlike MSE.[19]

The BCE loss function is used for classification problems with two classes. The equation for BCE is shown in Equation (2.12).

$$\text{BCE} \ = \frac{1}{N}\sum_{i=1}^{N} -\left(y_i \cdot \log\left(p_i\right) + \left(1 - y_i\right) \cdot \log\left(1 - p_i\right)\right) \tag{2.12}$$

There are only two possible outcomes in binary classification: 0 or 1. In Equation (2.12), p_i is the probability that the sample belongs to class 1 and $1 - p_i$ is the probability that it belongs to class 0.[18]

For multi-class classification, the CCE loss function can be used, as shown in Equation (2.13), where M is the number of classes.

$$\text{CCE} \ = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} \cdot \log\left(p_{ij}\right) \tag{2.13}$$

The BCE is a version of the CCE where M is set to 2 since there are only 2 classes.[20][18]

2.3.5 Optimizer

The optimizer is a crucial part of a neural network, responsible for adjusting the weights and biases during the training process. Its role is to minimize the error between the network's predictions and the ground truth values. The choice of optimizer influences the efficiency and accuracy of the network's predictions, making it a crucial aspect of the neural network architecture design. [10]

Gradient Descent is a foundational optimizer used in many neural network training processes. It operates by minimizing the loss function by updating the model parameters in the opposite direction of the gradient of the function. The size of each update step is determined by the learning rate α. Essentially, this process involves moving in the direction of the steepest descent on the surface defined by the objective function until a local minimum is reached [21]. The basic update rule for gradient descent is:

$$\theta_{t-1} \leftarrow \theta_t - \alpha \cdot \nabla \mathrm{J}(\theta_t) \tag{2.14}$$

.

In this equation, θ_t represents the model parameters at iteration t, α is the learning rate, and $\nabla \mathrm{J}(\theta_t)$ denotes the gradient of the loss function J concerning θ_t. Many advanced optimizers are extensions or variations of the basic gradient descent algorithm.

Adaptive Moment Estimation (Adam) is one of the most used optimizers in deep learning networks. For each weight of the network, Adam will compute adaptive learning rates, being separately adapted through the training process. This method will store an exponentially decaying average of the past gradients m_t,

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_\theta J_t(\theta_{t-1}) \tag{2.15}$$

and an exponentially decaying average of the past squared gradients v_t,

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_\theta J_t(\theta_{t-1}))^2 \tag{2.16}$$

comparable to momentum. In these functions, m_t is the estimate of the mean, v_t is the estimate of the uncentered variance of the gradients. The parameters β_1 and β_2 are exponential decay rates for the estimates. By adjusting learning rates based on these averages, Adam makes the learning process more effective and often yields a faster convergence compared to other optimizers [22]. There is an observed bias in the terms for the initial time steps. This is counteracted by the corrected estimate $\hat{m}_t$ and $\hat{v}_t$,

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.17}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}. \tag{2.18}$$

The parameters are then updated as,

$$\theta_t = \theta_{t-1} - \gamma \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{2.19}$$

where γ is the learning rate and ϵ is a small constant to prevent division with zero in the calculation.

2.3.6 Dropout

The problem of overfitting is common in machine learning. It occurs when the model learns the details of the training set too well, leading to bad performance on unseen data in the test set. An easily applicable regularization to combat overfitting is dropout. By dropping out some of the neurons of a layer during training, it prevents the model from becoming too reliant on specific neurons, thereby promoting more robust and generalizable learning [23]. Each neuron will have a dropout probability, which is the chance of the neuron being omitted during training. For each instance of the training, a new subset of neurons will be omitted. An example of dropout in a fully connected network is shown in Figure 2.3. The left side of the figure depicts a network without dropout, where all neurons are active. The right side shows the network with dropout applied, where some neurons are randomly omitted during training.

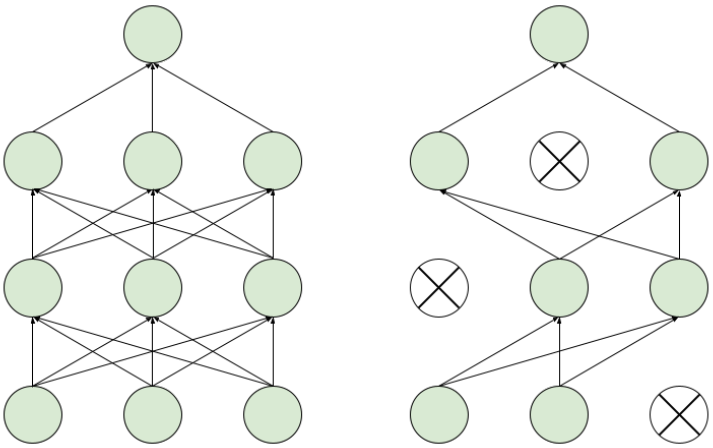


Figure 2.3: An illustration of a fully connected neural network, without and with dropout applied.

2.3.7 Weight decay

Another regularization method often used to handle overfitting is weight decay, also called L2 regularization. Weight decay is applied by introducing a penalty to the loss function,

$$L(\omega) = L(\omega) + \frac{1}{2}\lambda \sum_i \omega_i^2 \tag{2.20}$$

encouraging the model to have smaller weights. The principle for the implementation of weight decay is that models with smaller weights are simpler and less likely to overfit. [24] The parameter λ is a tuning parameter that controls how much the

larger weights are penalized and ω are the weights of the model. With the added sum of the squared weights, the updated formula for the weights will be:

$$\omega_i \leftarrow \omega_i - \eta \frac{\delta L(\omega)}{\delta \omega_i} - \eta \lambda \omega_i \tag{2.21}$$

where the last term $-\eta\lambda\omega_i$ comes from the weight decay term in Equation (2.20). This term will shrink the weights in each update, preventing them from growing too large and therefore helping to alleviate overfitting.

2.3.8 Semantic segmentation

Semantic segmentation is a method where every pixel of an image is classified to one of the predefined classes. This kind of task is often performed in computer vision and robotics. It can for example be used to determine the condition of the road ahead of a vehicle. The goal of semantic segmentation is to take an image and output a matrix with the same height and width as the input image. Every cell in the matrix is the predicted class ID of the corresponding pixel in the input image. In Figure 2.4 the concept of semantic segmentation is depicted, showing the class ID over the original image.

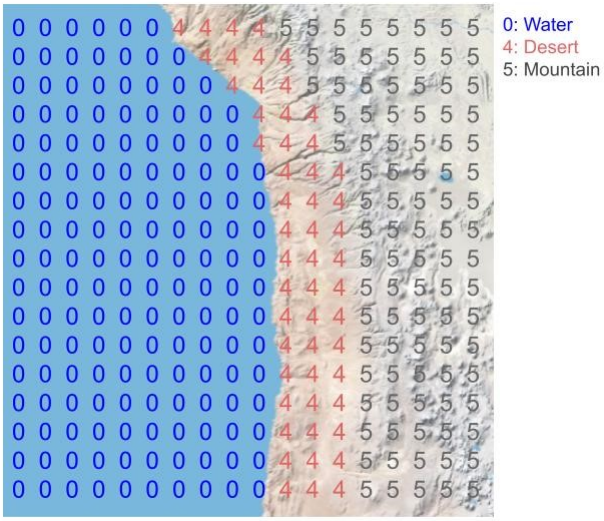


Figure 2.4: A terrain image with class labels on top. Note that the labels have a lower resolution than the image, which is not the case in reality.

The most common loss function in image segmentation is the pixel-wise cross entropy loss function. This loss function compares every pixel to its one-hot encoded target vector. One-hot encoded vector is often used to represent the target in deep machine learning. It is a binary vector where only one bit is set to 1, which is the "hot" bit, and the others to zero. Where the "hot" bit is positioned describes which class the sample belongs to. [25]

2.4 Convolutional neural network

A Convolutional Neural Network (CNN) is a versatile neural network architecture that excels in image classification and recognition tasks. It can also be applied in numerous other fields, such as Natural Language Processing. Due to its ability to process data in a grid-like manner, the network is well-suited for analyzing two-dimensional data [10]. The hidden layers of the CNN are built out of stacked convolutional layers, pooling layers, and fully connected layers. In **the convolutional layers**, a filter, also called kernel, is applied to the image. This filter is often a small matrix, that will slide over the input data, and at each instance, the filter will compute the scalar product. This operation will yield new matrices, so-called feature maps. The stride of the convolution is the number of pixels the filter moves along the surface. The stride will accept the size of the output feature map; a larger stride reduces the spatial dimension, while a small stride maintains the larger dimensions.

Padding is a method used to control the dimensions of the output feature map. It is done by adding zeros around the border of the input image. By applying padding, the dimension of the input can be preserved after convolution, ensuring that the output feature map maintains the desired size [26].

An example with these two hyperparameters applied, both padding and stride set to 1, the output feature map will retain the same dimensions as the input, as visualized in Figure 2.5.

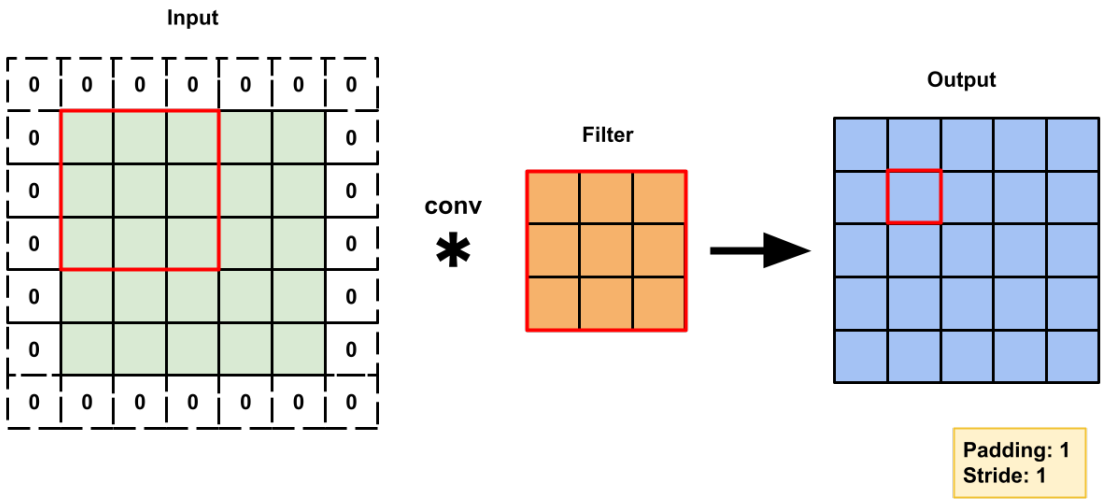


Figure 2.5: A simple representation of the convolutional operation with a 3x3 filter matrix.

The most common **pooling layer** used in a CNN is max pooling. The pooling layer will further downsample the feature maps, reducing complexity. The max pooling operation will choose the maximum element from the sub-region covered by the filter, thus producing a feature map with the most prominent features.[10]

The task of the **fully connected layer** is to take the output from the pooling

layer and produce class scores for the final classification task [27]. It will be connected to every neuron in the preceding and the succeeding layer. By integrating features from the previous layer, it makes the final prediction.

2.5 U-Net

In recent years, CNNs have been extensively used for image classification where the network's objective is to assign a single label to the whole image. As explained in Section 2.3.8, for semantic segmentation, each pixel of the image is supposed to be classified according to its semantic properties. An architecture more suitable for that complex task, requiring fewer training data, is the U-Net [28]. It was initially developed for image segmentation in the medical field, where the availability of training data can be limited. The network has a structure of a "U", consisting of a contracting path and expansive path, also called encoder and decoder. The general architecture for a U-Net is shown in Figure 2.6. The responsibility of the encoder is to extract features while downsampling from the input image, whereas the decoder upsamples the intermediate features to generate the segmented output.

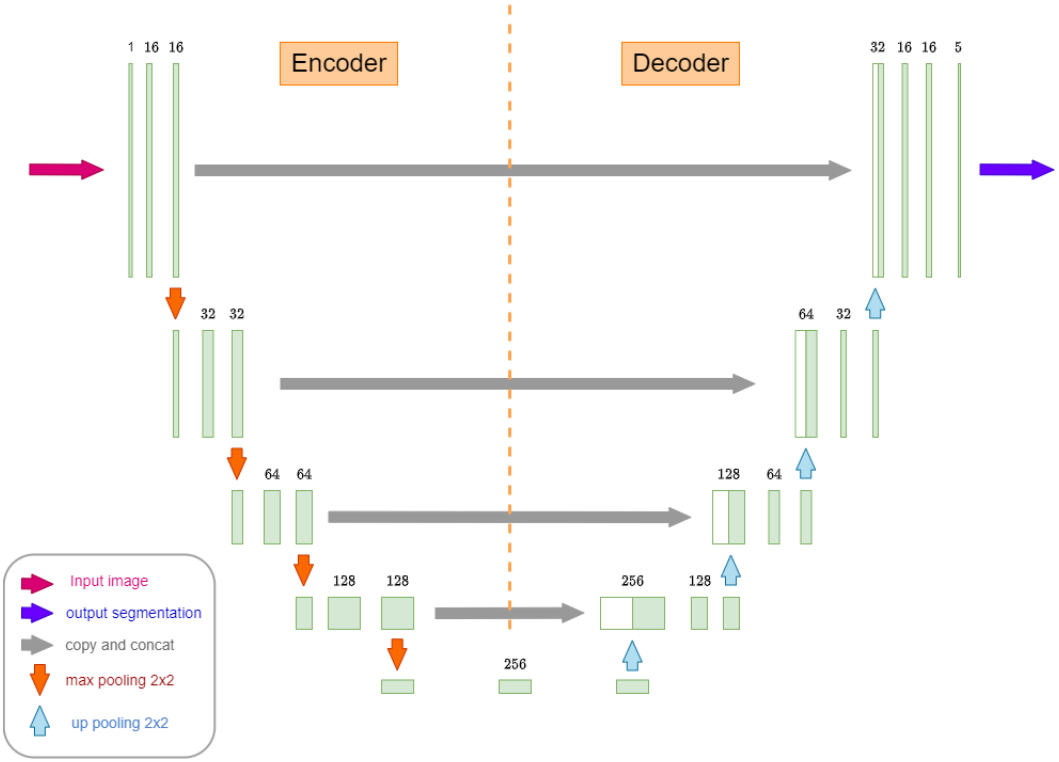


Figure 2.6: The architecture of the U-Net.

The contracting part will look like a regular CNN. Its building blocks consist of two convolutional layers, each followed by the ReLU function and max pooling operation, reducing the spatial information and increasing the feature information. The expanding part will instead use an up-pooling operation, combining the feature information with the spatial, and increasing the resolution. For each convolution

done, the border pixels of the image will be lost. To avoid this, each block of the expanding path is concatenated with the corresponding cropped feature map from the contracting path, which can be seen as the gray arrows in the middle of the U-shape in Figure 2.6. [28]

3
Methods

The following chapter will specify the process of the project. It will explain how the semantic segmentation, generation of targets, and visualization were performed.

3.1 Tracker Matlab

The project utilized a simplified tracker in Matlab created by Saab. The input format of the tracker was modified to ensure compatibility with other project components. How to visualize the tracks was researched and for this, a basic radar model was developed to detect objects based on angle, range, and noise to simulate a more realistic system. A simplified radar display was created in Matlab for experimental tests. The radar display presents the detections within the radar's range in green and those outside of the range in red.

3.2 Neural network

When simulating how the radar works in different environments, it is important to choose specific areas that should be simulated. By utilizing deep machine learning, two networks were created to enable this possibility. By using an open-source dataset from Kaggle, which consists of terrain maps, segmented maps, and height maps, the networks could learn how to create height maps from terrain images [29]. Examples of what the dataset includes are presented in Figure 3.1. These height maps can then be used to create the 3D simulation.

(a) Terrain map **(b)** Segmented map **(c)** Height map

Figure 3.1: The three types of images included in the dataset.

Two networks were created, one to learn the different terrain types, and one to learn the height values for the input. The U-Net structure was used in both

networks for efficient and accurate prediction.

3.2.1 Terrain type prediction

The segmented maps within the dataset comprised seven distinct colors, each corresponding to a specific class. The seven classes and their corresponding RGB values are illustrated in Figure 3.2. Figure 3.1b shows how these colors indicate the terrain type of each section of the map.

Figure 3.2: The 7 classes used for terrain type prediction.

To achieve our objective, we utilized the segmented map as the target. The aim was for the network to generate a segmented map based on a given terrain map.

3.2.1.1 Data preprocessing

Before feeding the images into the training loop, preprocessing was essential. The terrain map underwent resizing for easier handling and conversion to tensors using the PyTorch library. However, additional preprocessing was required for the segmented map to create a mask from it. The images were loaded, and a function was implemented to iterate through all pixels. This function assigns each pixel in an image to a class based on its RGB value. The implementation works by merging the three color channels of the image, so that each pixel position contains the complete RGB value. The function then iterates over every pixel in the image, matching each RGB value to the corresponding class index. The output is a mask with the same dimensions as the input image, where each cell contains the class index corresponding to the RGB value of the pixel in the input image.

Afterward, the mask was resized and converted into a tensor. Subsequently, datasets comprising training and test images along with their corresponding ground truth masks were constructed. After these datasets were done they were made into dataloaders with a batch size of 16.

3.2.1.2 Network for terrain segmentation

The network was designed to take the terrain map as input and output the segmented map. Thus, there are 3 channels in and 7 channels out, one for each class. A U-Net structure was used for the network in this project. The network consists of

an encoder, a bottleneck and a decoder, the structure visualized in Figure 2.6. A dropout layer was added as a final layer that could be utilized. The Optimizer used was the Adam Optimizer and the loss function was Cross-entropy.

3.2.1.3 Training and accuracy

By iterating over all batches in all epochs, the model could be trained. During each epoch, the training loss was computed, and the test loss was evaluated when the model was in evaluation mode. The training terminated automatically if the test loss fell below 0.1.

When the model had been trained, it could be evaluated. This was done by calculating the accuracy of the model. A function to calculate the accuracy was developed. The model was used to predict the terrain type of every image using the test data set. Comparing the correctly predicted pixels with all predicted pixels yielded a result on how well the model performs. The equation used to calculate the accuracy can be seen in Equation (3.1).

$$\texttt{accuracy} = \frac{\texttt{correct predicted pixels}}{\texttt{total predicted pixels}} \tag{3.1}$$

After the training was finished, the plots of both training and test loss were evaluated. With the plots and accuracy, any fitting improvements were made to get a higher accuracy.

3.2.2 Height prediction

The height maps included in the dataset were grayscale images where black corresponded to sea level and white to the highest value. The network in this part is similar to the one in Section 3.2.1, the input is still the terrain map, but here the ground truth and goal output is the height map.

3.2.2.1 Data preprocessing

Before using the input data in the network, it was preprocessed. The data used from the dataset were the terrain map images and their corresponding height maps, example shown in Figure 3.1(b)-(c). The height maps were grayscale images that contained the altitude information of the terrain, with 0, visualized as black, being the sea level and the highest value, visualized as white, the highest possible elevation. The original datatype of the images, an unsigned integer number stored with 16 bits, the pixels could take a significantly large number of values. For easier handling and visualization, it was changed to the more standard 8-bit unsigned integer, yielding possible values from 0 to 255.

Multi-class classification is computationally complex. Having 256 classes to classify each pixel of an image was deemed not doable due to computational limits. Therefore, the interval 0-255 was further minimized. By taking out each number in increments of 5, the number of classes to classify was then reduced to 52. Even though some information was lost with the decrease, it was deemed to contain enough information for the task of creating the 3D world.

Due to quite a large imbalance in the dataset, some adjustments were made. The classes representing low altitudes were far more represented. Some classes were removed since they did not contain any pixels from the data set, and the nine highest classes were merged into one due to the imbalance. This led to 40 classes remaining for the classification task, with class 40 representing the highest elevation of approximately 7500 meters. A mask was created for the height maps, assigning every pixel to the matching class, from 0 to 39.

3.2.2.2 Network for height prediction

In the network for the height prediction, the in-channels were still 3 since the input is the same as in the network for the segmented map. The difference between the height and terrain segmentation networks is that the out-channel now consists of the number of classes in height i.e., 40 classes. The optimizer used for the network was the Adam Optimizer and the loss was Cross-entropy, some test were conducted with a weighted version of the Cross-entropy loss function.

3.2.2.3 Training and accuracy

To reach the final model, the network was trained 14 times. The results from some of these tests will be presented in Section 4. When the model did not achieve the desired accuracy, parameters were changed and layers were added to try to improve it.

As explained previously, the chosen loss function was Cross-entropy. In order to compensate for the imbalanced dataset a weighted cross-entropy was tested. A manual re-scaling weight was calculated for every class in the dataset and applied to the Cross-entropy loss function. These weights tells the network that the classes that are not so represented in the dataset are more important to classify correctly.

To evaluate the result, two types of accuracy were calculated, one working exactly in the same way as the accuracy function explained in Equation (3.1) and one more customized to the height prediction. Since there are 40 classes to predict and it is better to classify to a neighbor class than one further away, e.g., class 30 to class 31 than class 1, an accuracy that considered this was calculated. A penalty matrix was constructed, and its structure can be seen in Figure 3.3. The penalty is 0 if the class is classified correctly, then 0.25 for the closest neighbor. The further the classification is from the true label, the higher the penalty.

$$\begin{bmatrix} 0.00 & 0.25 & 0.50 & 0.75 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 \ldots & 1.00 \\ 0.25 & 0.00 & 0.25 & 0.50 & 0.75 & 1.00 & 1.00 & 1.00 & 1.00 \ldots & 1.00 \\ 0.50 & 0.25 & 0.00 & 0.25 & 0.50 & 0.75 & 1.00 & 1.00 & 1.00 \ldots & 1.00 \\ 0.75 & 0.50 & 0.25 & 0.00 & 0.25 & 0.50 & 0.75 & 1.00 & 1.00 \ldots & 1.00 \\ 1.00 & 0.75 & 0.50 & 0.25 & 0.00 & 0.25 & 0.50 & 0.75 & 1.00 \ldots & 1.00 \\ \ldots & & & & & & & & & \\ 1.00 & 1.00 & 1.00 & 1.00 & 1.00 \ldots & 1.00 & 0.75 & 0.50 & 0.25 & 0.00 \end{bmatrix}$$

Figure 3.3: The structure of the penalty matrix with size 40 x 40.

3.3 Simulated target aircraft trajectories

The airplane trajectories were implemented in Python. To simulate stochastic scenarios, randomness was introduced into the initial position, velocity, and probability of turning. The modeling was done in discrete time with the position and velocity of the aircraft being updated at each time step, in order to simplify the simulation.

Given the plan for an educational and illustrative simulation of the radar, a simplified model of the targeted aircraft was employed. The primary objective was to demonstrate radar principles, e.g., the ability to track and detect the aircraft, and not how complex aerodynamic forces on an aircraft are simulated. Therefore, external forces such as lift, drag, and gravitational forces were disregarded, as they were not essential for generating basic flight trajectories. The randomness introduced in the velocity and turning of the aircraft facilitated the creation of diverse and random flight paths, effectively mimicking the real world without the need for detailed aerodynamic modeling.

As a result of the simplification, the aircraft could be simulated as a point in space. This meant that the simulation did not consider the aircraft's physical dimensions or dynamics. The aircraft was solely represented by its coordinates in space and its velocity vector.

The trajectories were created using the predicted height maps. These maps were also used to create the global coordinate system of the simulation. The height and width of the image represented the X and Y axes, while the pixel values of the image represented the Z axis. All axes were scaled to true values in meters.

A control function was developed to ensure that the aircraft was generated above the ground level. This function compared the elevation of the ground with the aircraft's altitude, adjusting the initial position if it fell below a specified threshold. This adjustment ensured that the aircraft started at a safe altitude above the terrain. The maximum altitude of the flying aircraft could also be set in the function.

Due to no external forces acting on the simulated point in space, the aircraft experiences no acceleration. Consequently, the velocity vector remains constant. This simplified the velocity and position update accordingly,

$$\begin{aligned} v_t &= v_{t-1} + a\Delta t & &\rightarrow v_t = v_{t-1} \\ p_t &= p_{t-1} + v_t\Delta t + \tfrac{1}{2}at^2 & &\rightarrow p_t = p_{t-1} + v_1\Delta t. \end{aligned} \tag{3.2}$$

A stochastic function was developed for the turning motion of the aircraft, as well as for collision detection and avoidance. The methods behind these are explained more in-depth in Sections 3.3.1-3.3.2. The function was further developed to generate multiple aircraft in the scenario. Since the state vectors of each aircraft are generated using stochastic variables, they will spawn and maneuver with variability. Given the extensive airspace covered, as detailed in Section 3.4, the risk of collision between aircraft is considered so low that collision avoidance measures between them are unnecessary. The number of aircraft that should be generated can be specified; if not specified, the function will generate a random number of aircrafts.

3.3.1 Stochastic turn function

To achieve realistic and smooth maneuvering of the aircraft, an algorithm was created to achieve the turning dynamics. A turn of the aircraft can be achieved by changing the heading for each time step and correcting the velocity and position vectors accordingly. The algorithm for the implementation is shown in Algorithm 1.

At each time step, the trajectory will have a chance to either stay in a straight trajectory or go into a turn. When a turn is initiated, the duration of the turn is determined by an exponential relationship with the bank angle of the aircraft. This captures a more realistic flight pattern, where a steep bank angle will result in a shorter turn and vice versa.

The turning motion was assumed to be done on the XY plane, with constant velocity in the Z axis. Therefore, the aircraft would maintain the same velocity in the Z direction throughout the turn. The direction of the turn was chosen with a randomly generated variable, with a 50/50 chance of either turning left or right. The heading of the aircraft is then corrected accordingly, increased for a right turn and decreased for a left turn.

To achieve realistic turning motion the function calculates the radius of the turn with the help of Equation (3.3). The equation determines how sharply a turn can be done without losing control or applying stress on the aircraft, therefore yielding realistic scenarios. It will calculate the turn radius based on the aircraft's true airspeed and the bank angle.

$$r_{turn} = \frac{v^2}{11.26 \tan(b)} \tag{3.3}$$

The equation for the rate of change of an angle, i.e., the angular velocity, is given in Equation (2.2). For the discretized model, the time interval Δt translates to the chosen time step.

The relationship between the angular velocity ω, the radius r, and the tangential velocity v is given in Equation (2.3). By combining Equations (2.2)-(2.3), the following relationship is attained:

$$\frac{\Delta\theta}{\Delta t} = \frac{v}{r} \implies \Delta\theta = \frac{v}{r}\Delta t \tag{3.4}$$

This equation allows for the usage of the calculated turn radius for the airplane, the airplane's velocity, and the time step to compute the angular change in the circular path. For an object moving in a uniform circular motion, i.e., moving along a circular path with constant speed, the angular change is equivalent to the change of heading for the object, as explained in Section 2.2.1. Therefore, $\Delta\theta$ can be seen as the change of heading of the aircraft.

With the updated heading of the aircraft, the velocity vector could be updated. While remaining at a constant speed through the turning motion, the direction will be changed accordingly. Finally, the position vector was computed. The algorithm used for the turning mechanism can be seen in Algorithm 1.

Algorithm 1 Simulate Aircraft Turn

Function simulate_aircraft_turn(p, dt, v, direction, b)
    // Function to simulate the turn of an aircraft

    // Calculate the aircraft's speed from its velocity components
    speed $\leftarrow \sqrt{\mathrm{v}[0]^2 + \mathrm{v}[1]^2}$

    // Calculate the current heading of the velocity vector
    $\theta \leftarrow \mathrm{arctan2}(\mathrm{v}[1], \mathrm{v}[0])$

    // Calculate the turn radius based on the bank angle and speed
    r $\leftarrow$ calculate_turn_radius(b, speed)

    // Calculate the change in heading during the time step
    $d\theta \leftarrow \left(\frac{\text{speed}}{\mathrm{r}}\right) \times \mathrm{dt}$

    // Update the heading based on the turn direction
if direction == 1 **then**
        $\theta \leftarrow \theta + d\theta$
else
        $\theta \leftarrow \theta - d\theta$
end if

    // Update the velocity vector based on the new heading
    $\mathrm{v}[0] \leftarrow \text{speed} \times \cos(\theta)$
    $\mathrm{v}[1] \leftarrow \text{speed} \times \sin(\theta)$

    // Update the position based on the velocity and time step
    p $\leftarrow$ p + v $\times$ dt

    // Return the updated velocity and position
    return (v, p)

3.3.2 Height map collision detection and avoidance

For enhanced realism for the trajectories created, they can not intersect with the terrain. For this purpose, a method for collision detection and avoidance was developed. This method utilizes the height map, enabling comparison between the aircraft's altitude and the terrain elevation. By ensuring that the aircraft's altitude is above ground, the collision can be preemptively avoided.

The collision detection algorithm was designed to predict and prevent potential collisions by analyzing the aircraft's future positions based on its current trajectory. Instead of comparing the current position of the aircraft with the terrain elevation, which results in detecting collisions too late for avoidance, the method will assess the potential future positions of the aircraft. This proactive approach allows for the

early detection of possible collisions, providing an opportunity to alter the course before it is too late.

The future predicted position of the aircraft was computed by extrapolating the current position with the current velocity vector and the time step. This calculation was repeated over multiple steps to provide a margin for safe avoidance. The algorithm evaluated the aircraft's altitude against the terrain elevation during each step. If the height distance is below a specified threshold at any future predicted position, the system will alert of the collision risk. Figure 3.4 illustrates the prediction and detection method.

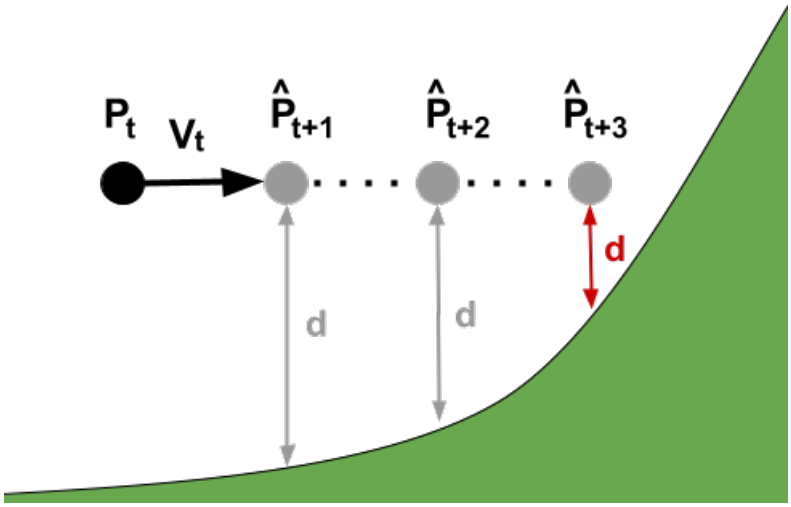


Figure 3.4: The method for collision detection. P_t and V_t are the current target state, $\hat{P}_{t+n}$ are the predicted future position and d the distance between the aircraft and the terrain. With d being smaller than the threshold, it will alert of the collision risk, demonstrated here in the color red.

When a potential collision is detected, the system will initiate a turning maneuver to avoid it. The bank angle determines the steepness of the turn. The collision distance and the bank angle are therefore designed to exhibit a linear relationship. As the object approaches the collision point, the bank angle increases, leading to a steeper turn to avoid the terrain.

Determining the direction of the avoidance turn is essential to achieve effective avoidance maneuvers for the aircraft. A method was developed to decide whether the aircraft should turn left or right to avoid the collision. This method involves checking the elevation of the terrain coordinates to the left and right of the predicted position. The left and right coordinates were retrieved by utilizing the velocity vector. The left and right directions could be obtained by calculating the direction vector perpendicular to the unit velocity vector. The left direction vector was determined by utilizing a simple rotation rule: with a 90° counterclockwise rotation, the x and y coordinates change as follows:

$$(x, y) \rightarrow (y, -x). \tag{3.5}$$

The right direction is then determined by taking the negative of the left direction vector. With the correct directions determined, we get the coordinates next to the predicted position and can retrieve their height values. If the terrain elevation to the right is higher, the aircraft will turn left, and if it is higher on the left, it will turn right.

A visualization of the collision avoidance for one aircraft is shown in Figure 3.5. The blue line indicates predicted positions where there was a risk of collision due to the terrain height being too close to the aircraft's path.

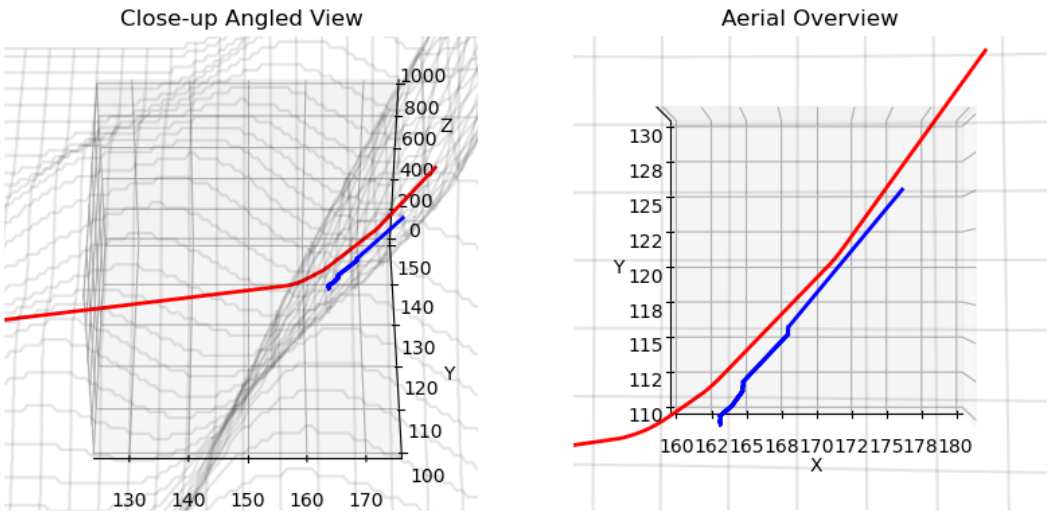


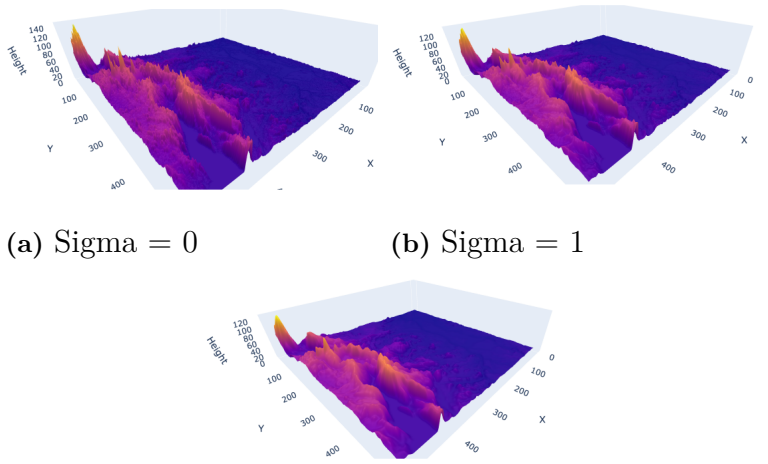
Figure 3.5: Plots showing the collision avoidance trajectory of an aircraft.

This method was implemented in the script and Algorithm 1 was used to go into the avoiding maneuver. An aircraft already in a turn when a collision is detected, will continue the turn. If needed, the steepness of the turn will be corrected for avoidance. With the use of the predicted height maps, collision detection and avoidance, and the turning algorithm, the aircraft's trajectory could be created.

3.4 Environment generator

With the height maps predicted by the network, the goal was to build the environment of the simulation. To enable parallel work with all parts of the project, ground truth images of the height maps and segmented maps were used when developing the environment. The terrain's depth data was used to build a surface in Python, utilizing the open-source library Plotly. The terrain was then scaled to represent the true values in meters. Originally every pixel of the terrain represented 400 meters in real life, during training the images were resized to 256x256 instead of 512x512 which resulted in a pixel representing 800 meters. By scaling the x and y values by 800, the axis spreads from 0 to 204800 meters. The height values were scaled by a factor of 29. This factor was calculated from the highest pixel value in the dataset which corresponded to Mount Everest. The predicted height map was spiky and therefore a Gaussian filter was applied in order to make the terrain smoother and

more realistic. The difference between the original terrain and processed terrain is shown in Figure 3.6. These figures are before any scaling was performed to any dimensions, with the only purpose of showing the spiky terrain and the smoothing filter that was used.



(a) Sigma = 0 **(b)** Sigma = 1

(c) Sigma = 2

Figure 3.6: The terrain with different sigmas for the Gaussian filter, where sigma is the standard deviation that controls how strong smoothing effect the filter has.

A point and a cone were added to represent the radar and its lobe. The library enables interactions with the 3D plot so that the scene can be rotated and moved, but at this point, the lobe was fixed in a certain direction. In order to add animations in Plotly, frames were created. Every frame represented a time step in the simulation and can be started by pressing a play button.

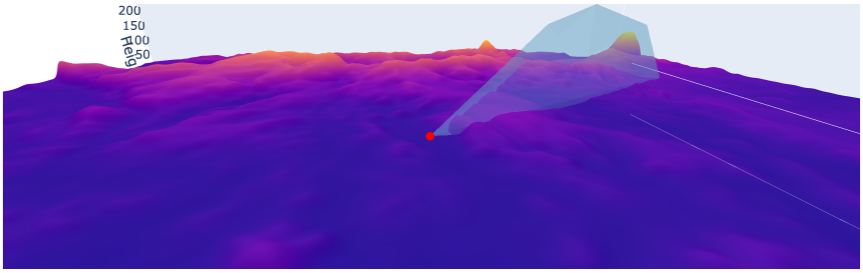


Figure 3.7: The environment with a radar and its lobe.

To represent the different terrain types in the environment, the segmented mask was implemented as a color map on the surface. Every point in the surface corresponds to a pixel in the height map which has a value representing the height. The segmented map was used to decide the color of every point, depending on the classification of that point's terrain type. In Figure 3.8 the environment with the terrain color implemented is shown, as well as the re-scaled axis. The height is scaled with

the appropriate factor to represent the meter value. The x and y axis are also scaled to represent the true meter values.

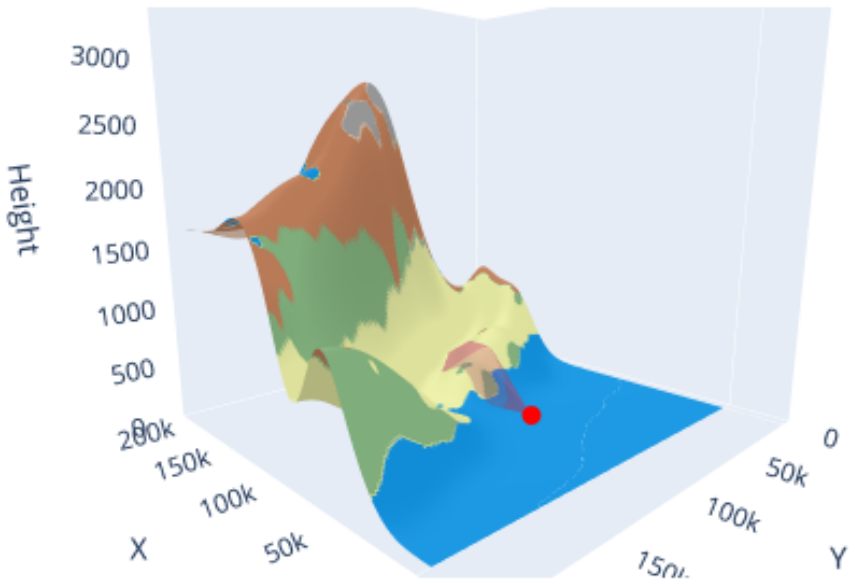


Figure 3.8: The environment with different surface color depending on the terrain type.

4
Results and discussion

The experimental work can be divided into three main parts: map segmentation, target trajectory generation and surroundings generation. Consequently, the following chapter will be divided into these three sections and one final section for combining all parts. In the chapter results obtained during the project will presented and discussed.

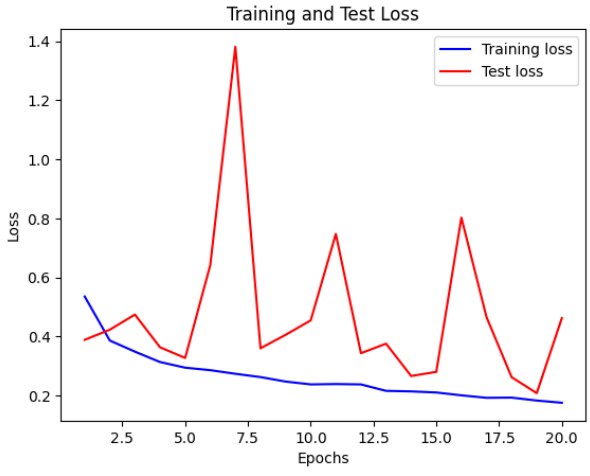
4.1 Map segmentation

The results from the two networks will be presented and discussed in two separate parts, one for each network.

4.1.1 Terrain segmentation

The network was trained and evaluated several times to achieve the desired result. Only those plots that were deemed important for the result will be presented in this chapter, other plots can be found in appendix. The tests and their specification are presented in Table 4.1.

Table 4.1: Test 1 specification, before any changes were made.

Test 1		Loss plot
Number of epochs	20	
Learning rate	0.001	
Accuracy	85.4%	

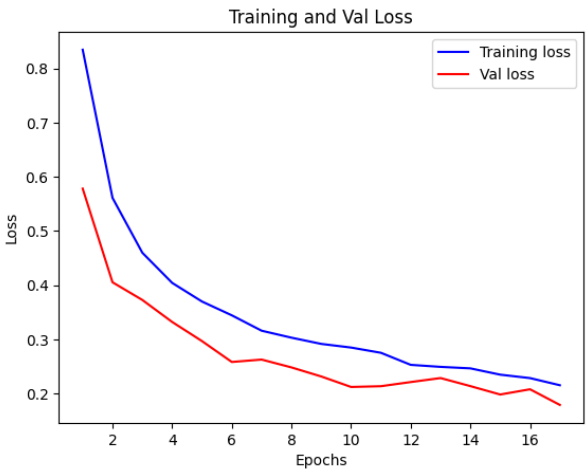
The problem after Test 1 was that the test loss was oscillating over the training loss, not decreasing in the desired manner. The conclusion drawn from the look of the loss plot was that the model was overfitted and therefore a dropout layer was implemented, with a dropout rate of 0.2, and the learning rate was lowered by a factor of 10. These changes yielded the result shown in Table 4.2.

Table 4.2: Test 2 specification with dropout and lower learning rate.

Test 2		Loss plot
Number of epochs	17	
Learning rate	0.0001	
Dropout rate	0.2	
Accuracy	92.67%	

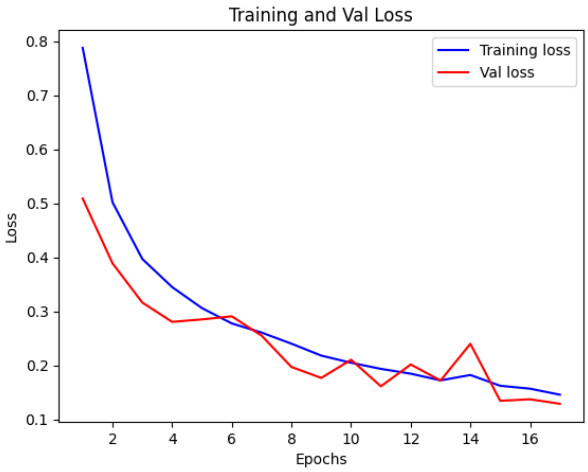
The changes led to a much higher accuracy, 92.67% compared to the 85.4% in Test 1. As can be seen in Table 4.2 the train loss is higher than the test loss in every epoch, which can be a sign of data leakage. Data leakage is when information not included in the training dataset is somehow used or accessed when creating the model. To avoid data leakage the data was split into 3: train, test, and validation before any preprocessing. However, this did not affect the result. In the third test the dropout rate was lowered to investigate if the model was too regularized. As can be seen in Table 4.3, the results improved with this change, yielding an accuracy of 93.63%.

Table 4.3: Test 3 specification with lower dropout rate and the same learning rate.

Test 3		Loss plot
Number of epochs	17	
Learning rate	0.0001	
Dropout rate	0.1	
Accuracy	93.63%	

Lastly, the network was trained with no dropout and a learning rate of 0.0001 which gave the best performance of all tests executed in the project. The result of test 4 can be seen in Table 4.4.

Table 4.4: Test 4 specification with no dropout and the same learning rate.

Test 4		Loss plot
Number of epochs	17	
Learning rate	0.0001	
Dropout rate	0.0	
Accuracy	95.30%	

Why the validation loss remains lower than the training loss could be due to several factors. It could be because the regularization is only applied during the training and not validation. When introducing a regularization method such as dropout we choose to sacrifice a possibly higher training loss in order to get a higher accuracy when testing our finished model. Another reason could be that the training

accuracy is measured during the training loop in each epoch, and the validation loss is measured after each epoch. This means that the validation loss is calculated when the network is more trained and can sometimes give a misleading result since it is measured later than the training loss. [30] What exactly causes the network to behave in this way must be further researched to draw certain conclusions.

In Figure 4.1 the result of predicting the terrain type of a terrain image with the best-performing model is shown, i.e. the model from Table 4.4. As can be seen, the model predicts the terrain similar to the ground truth.

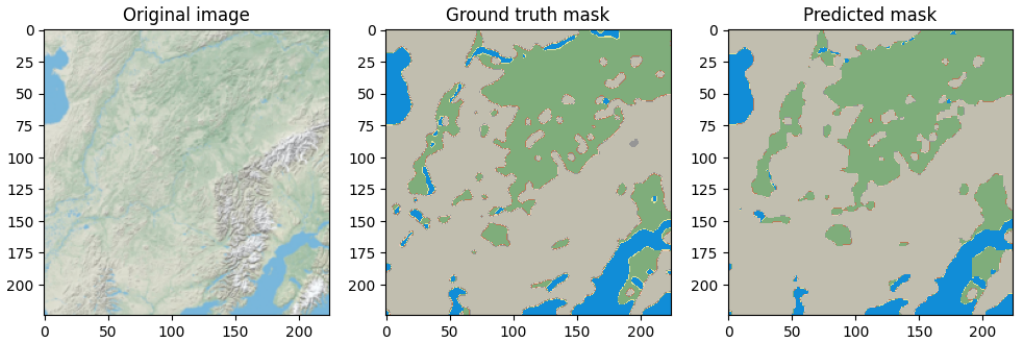


Figure 4.1: A terrain image with its ground truth segmentation map and its prediction.

4.1.2 Height predictions

The results from the first test can be seen in Table 4.5. As can be seen, the batch size was 8 instead of 16 which was used in the segment network. The training failed when trying to train it for 17 epochs with batch size 16, which most likely was due to limited GPU memory.

Table 4.5: Test 1 specification for height prediction.

Test 1	
Number of epochs	17
Number of labels	40
Learning rate	0.001
Dropout rate	0.1
Weights in loss function	YES
Batch size	8
Accuracy	42.36 %
Accuracy nearest neighbor	65.27 %

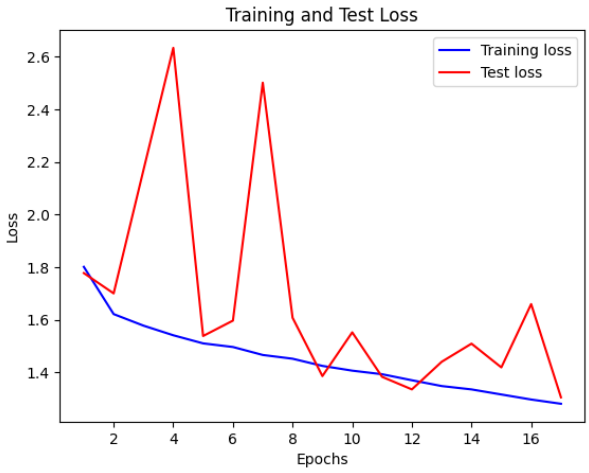
Loss plot

Redoing the training without the weighted loss function yielded an improved result. The result from this can be seen in Table 4.6.

Table 4.6: Test 2 specification for height prediction.

Test 2	
Number of epochs	17
Number of labels	40
Learning rate	0.001
Dropout rate	0.1
Weights in loss function	NO
Batch size	8
Accuracy	51.12 %
Accuracy nearest neighbor	76.31 %

Loss plot



A third test was done where the dropout rate was increased to 0.3 since the test loss oscillated above the training loss which could indicate an overfitted model. The third test resulted in a lower accuracy than the second test, which can be seen in Table 4.7, but better than the first.

Table 4.7: Test 3 specification for height prediction.

Test 3	
Number of epochs	17
Number of labels	40
Learning rate	0.001
Dropout rate	0.3
Weights in loss function	NO
Batch size	8
Accuracy	43.68 %
Accuracy nearest neighbor	66.31 %

Loss plot

More tests were conducted and the results from those can be seen in the Ap-

pendix. Test 2 gave the best result with an accuracy of 76.31 % calculated with less penalty for miss-classification with the nearest neighbors.

In Figure 4.2 the result is shown when the model from Test 2 was used to predict the height of a terrain image from the test data set. As can be seen, the predicted height map and the ground truth height map have some resemblance but there are visual differences, which shows that the model still needs some improvements until it works as intended. The result is however deemed good enough for the purpose it is meant for.

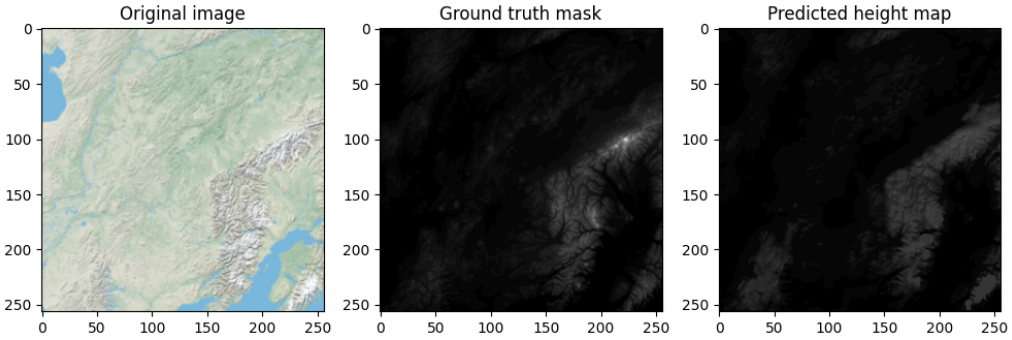


Figure 4.2: The original terrain image, the ground truth height map, and the predicted heightmap predicted by the best-performing model.

4.2 Target trajectories

The final method for generating the target trajectory produces a scenario with a specified or random number of aircraft. Figure 4.3 shows an example of $N = 4$ generated trajectories in a terrain scenario. With the predicted height map used to create boundaries for the trajectories, the aircraft will stochastically fly around in the scenario, avoiding collision with the terrain. The plot function used in Python to visualize the trajectories had some limitations, making it difficult to see the terrain. The trajectories' movement around the terrain is better visualized in the environment generation.

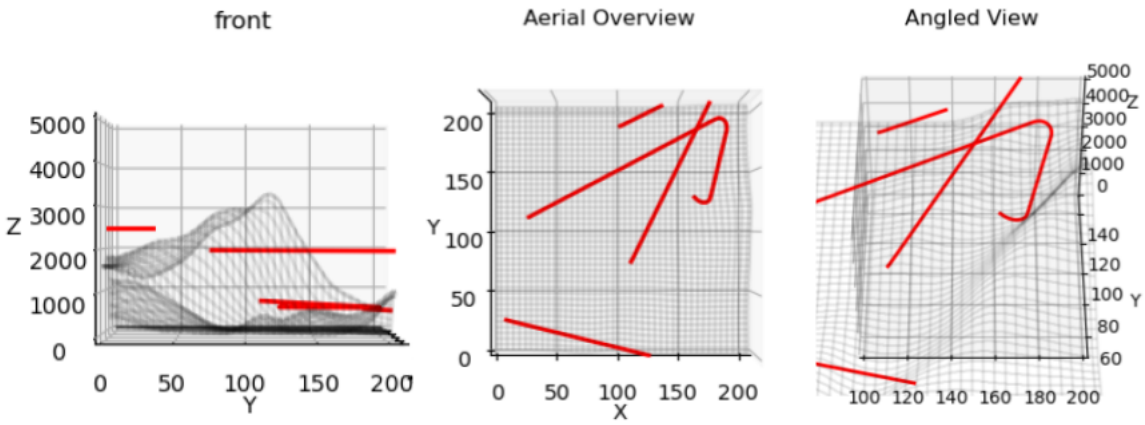


Figure 4.3: N=4 generated trajectories in a simulated scenario. The X and Y axes are in 10^3 meters.

Figure 3.5 shows a zoomed-in visualization of the collision avoidance. As the figure shows, the aircraft avoided a collision with the terrain by taking a left turn. The blue line in the graphs indicates the predicted positions where it detected a risk for collision. For most scenarios the generated collision avoidance worked well, avoiding the terrain. However, there are some cases where it does not work as effectively. As the generated trajectory in Figure 4.4 shows, if a trajectory is spawned too close to the terrain horizontally, it will exhibit some unusual behavior. It will immediately detect a future collision in a nearby time step, initiating the avoidance maneuver. Due to it being so close, it will not have enough time to avoid the terrain, resulting in a collision. After the collision happens, it will continue the turning motion inside the terrain because it detects a collision at each subsequent time step.

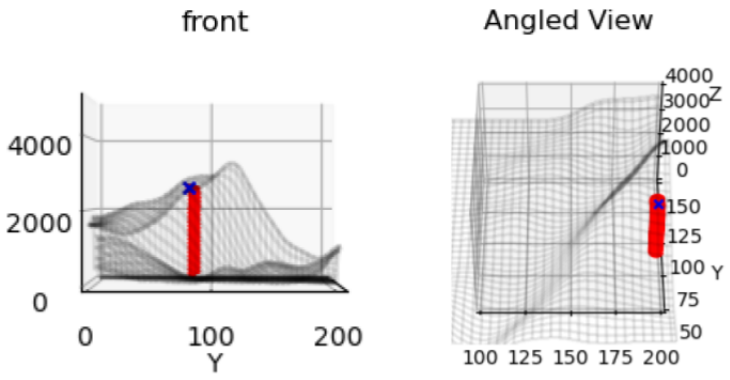


Figure 4.4: Visual representation of a faulty simulated trajectory. The airplane will fly in a turning spiral inside the mountain due to spawning to close. The blue cross indicates the initial position.

4.3 Environment generation

The final version of the environment is shown in Figure 4.5. It is built upon the actual height maps that were predicted in the network explained in Section 3.2.2. The colors in the environment are directly linked to the segmentation map that was segmented in the network depending on the terrain type of the input map. A radar and its main lobe are depicted in the plot, where the antenna moves at every time step of the simulation. Targets generated as described in Section 3.3 were added to the environment and their positions are also updated in every simulation step. Since the targets are moving in the environment Figure 4.5 shows the concept of the target moving and the target's position is not directly taken from the simulation. The black dot represents the current position of the target and the dotted line is its trajectory, where the light grey circles represent the targets future position.

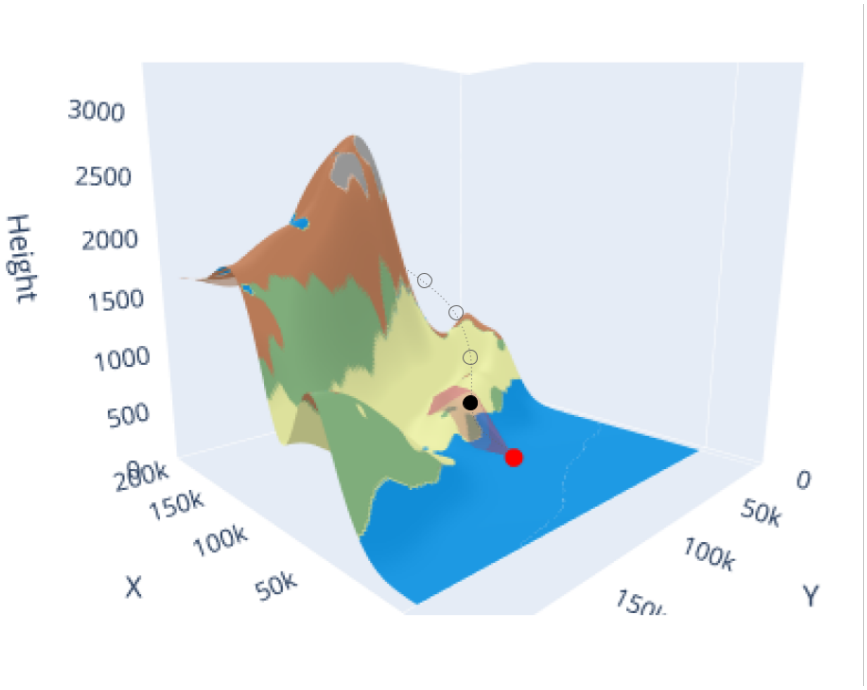


Figure 4.5: A concept image of the finished simulation with the target depicted as a black dot.

Currently, the radar searches in a circular motion without following a specific pattern. Extending the simulation to include more complicated search patterns is deemed simple to implement. A radar display connected to the simulation, similar to the one developed in Matlab, would aid in understanding radar functionality. Both of these features were planned for implementation if time permitted, but unfortunately, it did not.

4.4 Combined simulation

Using the predicted height maps, segmented maps and the generated trajectories resulted in the following simulation design shown in Figure 4.6.

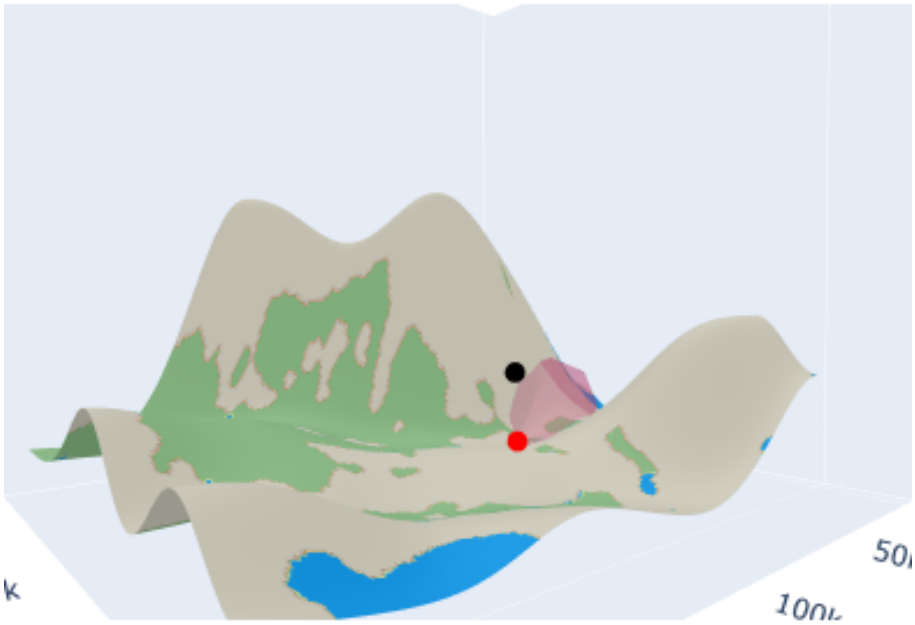


Figure 4.6: The finished simulation, built on the combination of previous parts.

The environment can be rotated and the simulation can be started to display the trajectory moving around the terrain and the radar lobe sweeping around the area.

5

Conclusion

There is limited research on constructing fully automated simulation environments in the radar field. By utilizing more simulations for both development and educational purposes, the need for resources can be reduced, leading to a more cost-effective and sustainable process. The work conducted in this project will contribute to further development in this field.

The work was divided into three parts: developing a realistic 3D simulation, implementing automated targets navigating in this environment, and designing the simulation environment to visualize radar concepts for educational purposes and lay the groundwork for a more comprehensive system in the future.

The vision of using a single terrain map as input to accelerate the creation of an entire simulation has been proven to work through the experiments conducted in this thesis. By utilizing deep machine learning, the data used to create the foundation of the environment is generated. This approach allows for the selection of the simulation setting, enabling simulations in environments based on the actual locations where the product will be used. A challenge still remaining is the impact the data set has on the simulation environment. The type of maps used during training limits what maps can be used to generate the foundation. It also requires terrain maps of the specific area to be simulated in.

The implementation of simulated target aircraft trajectories in Python successfully illustrated how target aircraft can fly in the simulation, enabling the radar to detect and track them effectively. By introducing randomness into initial positions, velocities, and turning probabilities, the simulation was able to create stochastic and realistic flight paths without the need for complex aerodynamic modeling. The simplification of representing aircraft as points in space, while disregarding external forces like lift, drag, and gravity, proved effective for the intended educational and illustrative purposes.

The incorporation of a stochastic turn function allowed for realistic maneuvering of aircraft, simulating turns by adjusting headings and velocities. This function, along with the collision detection algorithm, enabled the simulation to preemptively avoid potential collisions by predicting future positions. However, for some scenarios the avoidance function did not work, producing a faulty trajectory.

Combining the environment and the trajectories sets the foundation for an educational simulation where the radar can be visualized. The simulation enables the user to look around and change the position of the radar, but in order to work as a useful tool a lot of work remains.

5.1 Future work

The networks that create the terrain for the simulation can be further developed to get more accurate results. One approach that would be interesting to explore is the one discussed in Section 1.4 where they predict both height and terrain type simultaneously. This could reduce training time and improve the performance of the network. As the goal is to have a fully automated chain for the simulation, it would simplify the process if only one model were needed to predict the terrain type and height.

There is still much to be done to enhance the realism of the automated targets generated in the scenario. Due to time constraints, the method for creating the trajectories was not fully developed. For visualization in this project, it was deemed unnecessary to implement all the physical dynamics of aircraft; however, incorporating these dynamics in future developments could be beneficial. With accurate dimensions and parameters for the aircraft, it becomes possible to test the radar against parameters such as radar cross-section and other relevant metrics.

Collision detection and avoidance could be researched more. By incorporating machine learning algorithms to predict collisions and adjust target trajectories in real-time, it could improve the performance of the simulation. Additionally, looking into techniques like reinforcement learning, which could help the targets adapt to their environment, would ensure safe navigation and effective collision avoidance.

In order to use the simulation tool for radar design it could also be interesting to incorporate different types of search patterns and enable change of radar platform, i.e., being able to change to a flying platform. This, and more extensions to the simulation, could give valuable information when developing or improving a radar design.

The terrain images used in this project are all of the same type, which constrains the network to these specific images. To enhance the network's learning capabilities in the future, and enable the incorporation of terrain images from sources such as Google Earth or similar programs, the variety of terrain images needs to be expanded. Broadening the types of terrain images will improve the network's robustness and adaptability to different environments, leading to more accurate and flexible performance.

It would be interesting to further research how a more game-like environment could be developed to enhance the visual aspect of the simulation. This could facilitate greater interaction with the simulation, potentially making it easier to understand various concepts presented within the environment. Using Pygame to develop such an environment was researched briefly, but the idea was dismissed due to a lack of experience in game development and time constraints.

Bibliography

[1] A. Massobrio, *What is simulation-driven design? main benefits explained*, Neural concept, 2024. [Online]. Available: https://www.neuralconcept.com/post/what-is-simulation-driven-design-main-benefits-explained.

[2] BBC, *Advantages and disadvantages*, BBC, Accessed: 2024-06-03. [Online]. Available: https://www.bbc.co.uk/bitesize/guides/zyqfr82/revision/3.

[3] M. Wen, J. Park, and K. Cho, "A scenario generation pipeline for autonomous vehicle simulators," *Human-centric Computing and Information Sciences*, vol. 10, 2020. DOI: 10.1186/s13673-020-00231-z.

[4] M. Lu, J. Liu, F. Wang, and Y. Xiang, "Multi-task learning of relative height estimation and semantic segmentation from single airborne rgb images," *Remote Sensing*, vol. 14, 2022. [Online]. Available: https://www.mdpi.com/2072-4292/14/14/3450.

[5] C. W. Reynolds, "Flocks, herds, and schools: A distributed behavioral model," *Seminal graphics: pioneering efforts that shaped the field*, 1987. [Online]. Available: https://api.semanticscholar.org/CorpusID:546350.

[6] J. C. Toomay, *Radar Principles for the Non-Specialist*. Springer Dordrecht, 2012.

[7] P. Institute, *What is turn radius (aviation)?* Accessed: 2024-05-14. [Online]. Available: https://pilotinstitute.com/radius-of-turn/.

[8] L. Learning, *Rotation angle and angular velocity*, Accessed: 2024-05-21, n.d. [Online]. Available: https://courses.lumenlearning.com/suny-physics/chapter/6-1-rotation-angle-and-angular-velocity/.

[9] OpenStax, *University Physics Volume 1*. OpenStax, 2016, Accessed: 2024-05-23. [Online]. Available: https://openstax.org/books/university-physics-volume-1/pages/4-4-uniform-circular-motion.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.

[11] A. Jain, J. Mao, and K. Mohiuddin, "Artificial neural networks: A tutorial," *Computer*, vol. 29, no. 3, pp. 31–44, 1996. DOI: 10.1109/2.485891.

[12] V. Zwass, *Neural network*, Encyclopedia Britannica, Accessed: 2024-05-16. [Online]. Available: https://www.britannica.com/technology/neural-network.

[13] upGrad, *Neural network: Architecture, components top algorithms*, upGrad, Accessed: 2024-06-25, 2022. [Online]. Available: https://www.upgrad.com/blog/neural-network-architecture-components-algorithms/.

[14] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *International Journal of Engineering Applied Sciences and Technology*, vol. 04, pp. 310–316, May 2020. DOI: 10.33564/IJEAST.2020.v04i12.054.

[15] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," 2018. arXiv: 1811.03378 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1811.03378.

[16] A Olgac and B. Karlik, "Performance analysis of various activation functions in generalized mlp architectures of neural networks," *International Journal of Artificial Intelligence And Expert Systems*, vol. 1, pp. 111–122, 2011.

[17] M. Zeiler, M. Ranzato, R. Monga, *et al.*, "On rectified linear units for speech processing," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 3517–3521. DOI: 10.1109/ICASSP.2013.6638312.

[18] V. Yathish, *Loss functions and their use in neural networks*, Medium, Accessed: 2024-05-09, 2022. [Online]. Available: https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9.

[19] P. AI, *Understanding the 3 most common loss functions for machine learning regression*, Towards Data Science, Accessed: 2024-05-09, 2019. [Online]. Available: https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d3.

[20] R. Alake, *Loss functions in machine learning explained*, DataCamp, Accessed: 2024-05-09, 2023. [Online]. Available: https://www.datacamp.com/tutorial/loss-function-in-machine-learning.

[21] S. Ruder, *An overview of gradient descent optimization algorithms*, 2017. arXiv: 1609.04747 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1609.04747.

[22] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1412.6980.

[23] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, *Improving neural networks by preventing co-adaptation of feature detectors*, 2012. arXiv: 1207.0580 [cs.NE]. [Online]. Available: https://arxiv.org/abs/1207.0580.

[24] A. Kumar, *Weight decay in machine learning: Concepts*, Vitalflux, Accessed: 2024-06-27, 2023. [Online]. Available: https://vitalflux.com/weight-decay-in-machine-learning-concepts/.

[25] J. Jordan, *An overview of semantic image segmentation.* Accessed: 2024-05-09, 2018. [Online]. Available: https://www.jeremyjordan.me/semantic-segmentation/.

[26] K. O'Shea and R. Nash, *An introduction to convolutional neural networks*, 2015. arXiv: 1511.08458 [cs.NE]. [Online]. Available: https://arxiv.org/abs/1511.08458.

[27] A. Saxena, "An introduction to convolutional neural networks," *Int. J. Res. Appl. Sci. Eng. Technol*, vol. 10, no. 12, pp. 943–947, 2022.

[28] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, 2015. arXiv: 1505.04597 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1505.04597.

[29] T. Pappas, *Earth terrain, height, and segmentation map images*, Accessed: 2024-03-06. [Online]. Available: https://www.kaggle.com/datasets/tpapp157/earth-terrain-height-and-segmentation-map-images.

[30] A. Rosebrock, *Why is my validation loss lower than my training loss?* pyimagesearch, Accessed: 2024-06-3, 2019. [Online]. Available: https://pyimagesearch.com/2019/10/14/why-is-my-validation-loss-lower-than-my-training-loss/.

A

Appendix 1

Table A.1: Test with torch.inteference_mode() instead of to torch.no_grad.

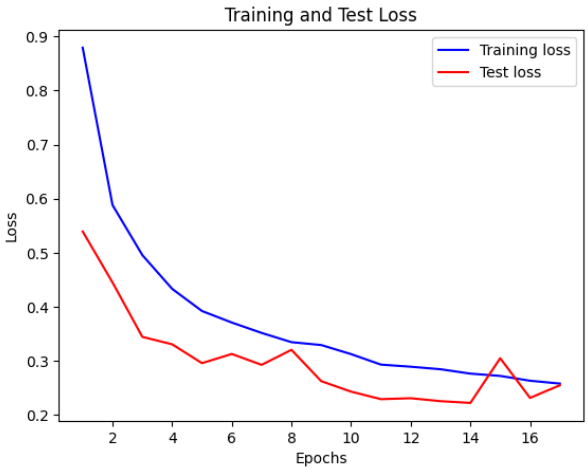
Test for segmentation of terrain		Loss plot
Number of epochs	17	
Learning rate	0.0001	
Dropout rate	0.2	
Batch size	16	
Accuracy	90.49 %	

Table A.2: Test with torch.no_grad and learning rate 0.001.

Test for segmentation of terrain		Loss plot
Number of epochs	17	
Learning rate	0.001	
Dropout rate	0.0	
Batch size	16	
Accuracy	69.55 %	

Table A.3: Test with torch.no_grad and learning rate 0.0005.

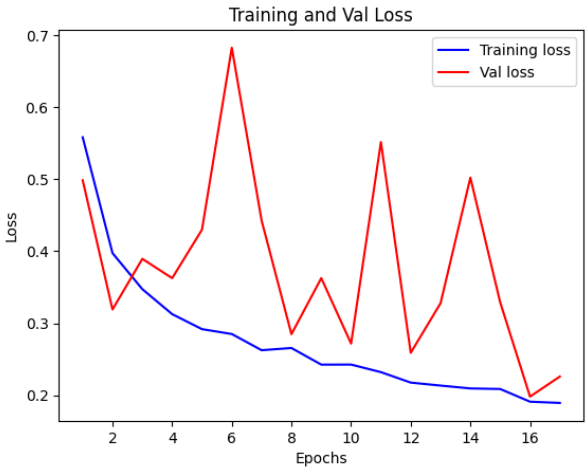
Test for segmentation of terrain		Loss plot
Number of epochs	17	
Learning rate	0.0005	
Dropout rate	0.0	
Batch size	16	
Accuracy	91.38 %	

Table A.4: Test with a different random state when creating the dataloaders and learning rate 0.0001.

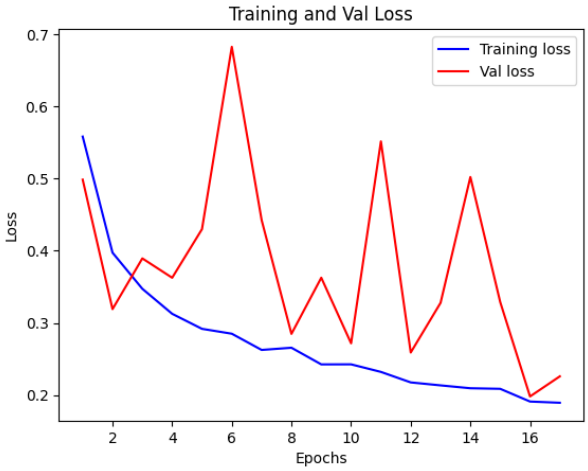
Test for segmentation of terrain		Loss plot
Number of epochs	17	
Learning rate	0.0001	
Dropout rate	0.0	
Batch size	16	
Accuracy	95.13 %	

Table A.5: Test where the best model was trained for 3 extra epochs.

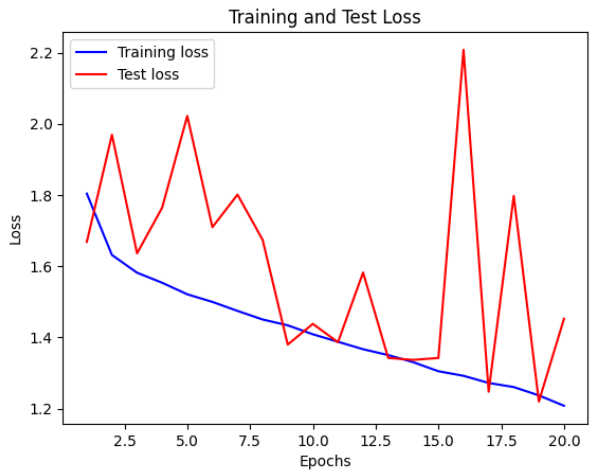
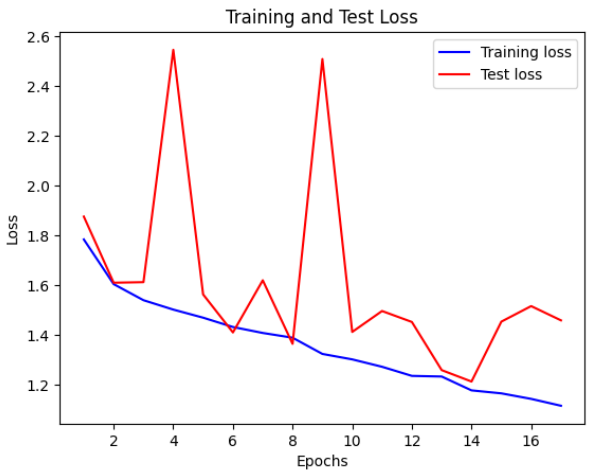
Test for height prediction		Loss plot
Number of epochs	20	
Learning rate	0.001	
Dropout rate	0.1	
Number of classes	40	
Weight in loss function	NO	
Batch size	8	
Accuracy	48.61 %	
Accuracy nearest neighbor	72.82 %	

Table A.6: Test with higher dropout rate.

Test for height prediction		Loss plot
Number of epochs	20	
Learning rate	0.001	
Dropout rate	0.2	
Number of classes	40	
Weight in loss function	NO	
Batch size	8	
Accuracy	42.75 %	
Accuracy nearest neighbor	67.76 %	

Table A.7: Test with no dropout.

Test for height prediction		Loss plot
Number of epochs	17	
Learning rate	0.001	
Dropout rate	0.0	
Number of classes	40	
Weight in loss function	NO	
Batch size	8	
Accuracy	49.01 %	
Accuracy nearest neighbor	72.60 %	

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY