Master's thesis

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

Magnus Dyre-Moe

Learning-based Collision-free Navigation for Aerial Robots

Master's thesis in Cybernetics and Robotics
Supervisor: Konstantinos Alexis
Co-supervisor: Dinh Huan Nguyen
June 2022

NTNU
Norwegian University of Science and Technology

Magnus Dyre-Moe

Learning-based Collision-free Navigation for Aerial Robots

Master's thesis in Cybernetics and Robotics
Supervisor: Konstantinos Alexis
Co-supervisor: Dinh Huan Nguyen
June 2022

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

Preface

The delivery of this thesis concludes my five-year journey as a student at NTNU. It is difficult to describe my stay at NTNU, but it is fair to say it includes a lot of joy, satisfaction, and frustration. Academically it has been fun and exciting but also challenging and, at times, monotonous. I am sure that I will look at my time at NTNU fondly, but as of now, I am excited to close this chapter of my life and open up a new one.

I want to thank my supervisor, Prof. Dr. Konstantinos Alexis, for his invaluable guidance throughout my final year at NTNU. I would also like to thank Ph.D. candidates Dinh Huan Nguyen and Mihir Kulkarni for their willingness to help me through periods of frustration and for assisting me with technical difficulties. Finally, I would like to thank myself for finishing this demanding degree on time, and my friends for all the great memories I will forever remember.

Abstract

Learning-based approaches in unmanned aerial vehicles have seen increased attention over recent years. The motivation for the increasing popularity of learning-based methods stems from traditional methods, such as SLAM, where data association is an active and challenging research topic. More efficient and lighter sensors, more onboard computation power, and better algorithms have allowed UAVs to run increasingly complex models in real-time. These advancements, facilitated by a growing open source community and complex simulation tools, have allowed the more rapid development of complex models while being cost-effective and without human supervision. Though effective concerning time and money, developing models in simulation comes at a drawback as it is not yet a perfect replica of the real world. One drawback with developing models in simulation is that depth images are more prone to noise and missing data in the real world, whereas depth in simulation is close to perfect. Various methods are used to ensure acceptable transfer between simulation and the real world in what is known as sim-to-real transfer. This thesis explores the use of a deep collision predictor network, trained through supervised learning, to predict collision probabilities of action sequences drawn from a motion primitives library to navigate real-time in cluttered environments safely. Furthermore, we attempt to improve sim-to-real transfer by utilizing representation learning on observed depth, created through semi-global matching, a stereo matching algorithm. Semi-global matching provided depth images more akin to real-world depth images. Still, the algorithm struggles to match areas where the texture was lacking. Furthermore, using a variational autoencoder for representation learning resulted in reconstructions where the depth was largely preserved but lacked more towards retaining geometric features. For more complex images, where many features were present, these results partially crumbled as the reconstructed images struggled more towards preserving depth. The results for collision-free navigation illustrate how observations are utilized to evaluate the probability of collisions along different executable trajectories, allowing the robot to safely traverse a simplistic environment with a success rate of 80%. For a more complex environment, collisions occurred more frequently, resulting in a success rate of 10%. Nonetheless, the results are encouraging but could be further improved to make navigation safer.

Sammendrag

Læringsbaserte tilnærminger i UAV-er har fått økt oppmerksomhet de siste årene. Motivasjonen for den økende populariteten stammer fra tradisjonelle metoder, som SLAM, hvor dataassosiasjon er et aktivt og utfordrende forskningstema. Mer effektive og lettere sensorer, mer innebygd beregningskraft og bedre algoritmer har gjort det mulig for UAV-er å kjøre stadig mer komplekse modeller i sanntid. Disse fremskrittene, tilrettelagt av et voksende fellesskap med åpen kildekode og komplekse simuleringsverktøy, har muliggjort raskere utvikling av komplekse modeller samtidig som de er kostnadseffektive og uten behov for menneskelig tilsyn. Selv om det er effektivt med tanke på tid og penger, har utvikling av modeller i simulering en ulempe, da simuleringer ikke er en perfekt kopi av den virkelige verden. En ulempe med å utvikle modeller i simulering er at dybdebilder er mer utsatt for støy og manglende data i den virkelige verden, mens dybden i simulering er nærmere perfekt. Ulike metoder brukes for å sikre akseptabel overføring mellom simulering og den virkelige verden i det som er kjent som simulering-til-virkelighet overføring. Denne masteroppgaven utforsker bruken av et dypt nevralt network til å predikere kollisjon, trent gjennom veiledet læring, for å forutsi kollisjonssannsynligheter for ulike sekvenser av handlinger, hentet fra et bevegelsesprimitivbibliotek for å navigere i sanntid i rotete miljøer på en sikker måte. Videre prøver vi å forbedre simulering-til-virkelighet overføring ved å bruke representasjonslæring på observert dybde, skapt gjennom stereomatching, som lignet mer på virkelige dybdebilder. Algoritmen sliter likevel med å matche områder der tekstur mangler. Videre resulterte bruk av en variasjonsautokoder for representasjonslæring i rekonstruksjoner der dybden stort sett ble bevart, men så større mangler med å beholde geometriske trekk. For mer komplekse bilder, hvor mange særtrekk var til stede, smuldret disse resultatene delvis opp ettersom de rekonstruerte bildene strevde mer mot å bevare dybden. Resultatene for kollisjonsfri navigasjon illustrerer hvordan observasjoner brukes til å evaluere sannsynligheten for kollisjoner langs forskjellige kjørbare baner, slik at roboten trygt kan krysse et relativt enkelt miljø med en suksessrate på 80%. For et mer komplekst miljø skjedde kollisjoner oftere, noe som resulterte i en suksessrate på 10%. Likevel er resultatene oppmuntrende, men de kan forbedres ytterligere for å gjøre navigasjon tryggere.

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Motivation

Safe and reliable navigation of autonomous vehicles is a challenging task traditionally solved by building a map of the surroundings and localizing the vehicle within the map[1–5]. These methods are usually referred to as *simultaneous localization and mapping*, *SLAM* [6, 7], and provide state-of-the-art methods to solving navigation tasks. However, these methods have shortcomings, where data association is still an open problem. This problem is specifically related to learning from previous information[8]. Learning-based approaches are based on machine learning and present an alternative approach to navigation tasks where a model learns directly from previous experiences [9–11].

Learning-based approaches, and machine learning in general, have increased in popularity in recent years. Quicker, cheaper, and more accessible hardware and software have made it possible to develop more complex algorithms and models. This has enabled the use of learning-based approaches in applications traditionally limited by computational complexity and weight constraints, such as aerial robotics, including control, navigation, and guidance tasks. Such tasks have widespread applications, including among others, search and rescue[12], aerial inspection[13, 14], and forestry[15].

Learning-based approaches in robotics, like SLAM, rely on sensory data and observations to properly navigate a challenging environment. Cameras provide rich contextual information about the surroundings required for different decision-making and navigation tasks. Determining what information is helpful in images can be difficult. However, convolutional neural networks (CNNs) have proved to be critical components in feature extraction on images, given their excellent capabilities on structured data[16]. Although CNNs are good at feature extraction, there is no certainty that the resulting representation is sufficiently good in union with other learning tasks. Representation learning aims to solve this issue by ensuring meaningful feature extraction to make subsequent tasks easier. This has given rise to neural network

architectures such as variational autoencoders (VAEs). Because of the capabilities of making subsequent learning easier, representation learning has forced its way into robotic applications where it has been used in tasks, ranging from navigation and path planning[17, 18] to mapping[19].

Robots inherently interact with the physical world[20]. Still, many recent proposed methods are developed in simulation. This is largely due to it being more convenient and potentially less damaging to work in simulation than in the real world. First, the physical world has the disadvantage of running in real-time, which is time-consuming and often requires human interaction. These constraints are avoidable by running in simulation, which allows for automation, removing the need for human interaction while also being able to run faster than in real-time. Learning algorithms can also be prone to unexpected and violent behavior, which can potentially be damaging to hardware, and, therefore, expensive[20]. All together, it makes learning in simulation more convenient. However, learning a model in simulation begs the question of whether the same model can be used in the real world. Different techniques are used in order to make a model trained in simulation behave reasonably in the real world. This is known as *sim-to-real* transfer. Open-source software, such as ROS[21] and Gazebo[22] has enabled the use of *digital twins* in simulation, bridging the reality gap. Still, sensor data, particularly depth imagery, tend to be close to perfect in simulation. Even though it is possible to add noise in simulation, the resulting depth images are nonetheless different from images captured in the real world, as it is not uncommon to see gaps in the depth images due to, among others, reflection. Additional measures, such as creating depth images in simulation through stereo matching, can be done to increase sim-to-real transfer further.

1.2 Scope

ORACLE[11] is a deep collision predictor neural network that predicts collision probabilities of different future action sequences for a micro aerial vehicle (MAV) in cluttered environments. The goal of ORACLE is to safely and reliably navigate through the environment by using observations of the surroundings, through images, and the robot's state. This thesis aims to further develop the ORACLE model through two further additions done in simulation. First, we attempt to further bridge the gap between simulation and the real world by adding texture to the simulation and creating depth maps more akin to actual depth maps in the real world. Creating the depth maps will be done by utilizing a stereo camera and using a stereo matching algorithm known as semi-global matching. Second, to make learning the collision probabilities easier, a VAE will be added to the deep neural network architecture, conceivably resulting in a more meaningful representation of the observed depth.

1.3 Outline

- **Chapter 1: Introduction**. This chapter contains an introduction to learning-based approaches, representation learning, and the use of simulations. Furthermore, the scope of this thesis is presented.

- **Chapter 2: Related works**. In this chapter, relevant previous work and methods are presented. The chapter is separated into two parts. The first part presents related works concerning learning-based approaches that utilize supervised learning to solve a navigation task. The second part reviews the successful use of representation learning.
- **Chapter 3: Theoretical background**. The theoretical background contains five main sections. The first two parts are used to give a brief introduction to supervised and unsupervised learning. Next, one part is dedicated to deep learning and artificial neural network, providing information about the different building blocks used to create ORACLE. The second to last part focuses on representation learning through the autoencoder (AE) and variational autoencoder (VAE) neural network architectures. Lastly, the matching algorithm of choice, semi-global matching, is presented.
- **Chapter 4: Problem Description**. This chapter describes the problem of safe navigation through a cluttered environment.
- **Chapter 5: Software Tools**. This chapter presents the necessary software tools to train a model through simulation. Moreover, with the relevant hardware, the quadrotor robot is presented before looking briefly at the external processing power used.
- **Chapter 6: Proposed approach**. This chapter will focus on the proposed approach in this thesis, which can be split into five parts. The first two parts focus on expanding the used neural network architecture and environment. The third part describes how data is collected through simulation. The second to last part expresses how the deep neural network is trained, both the VAE separate from ORACLE, as well as expanded ORACLE structure which includes the VAE. The last part outlines how the model is evaluated through evaluation studies and machine learning metrics.
- **Chapter 7: Results**. The results are three-fold, going component by component. First, the results from the stereo matching algorithm, semi-global matching, will be presented. Then, the performance of the VAE will be conferred, touching upon the training procedure and the model output, which are the input images reconstructed. Lastly, the expanded ORACLE model, ORACLE-VAE, will be evaluated through machine learning metrics and the evaluation studies presented in the method chapter.
- **Chapter 8: Discussion**. The results are discussed in this chapter, explaining why they are as they are while also proposing possible suggestions for improving the results.
- **Chapter 9: Conclusion**. The conclusion ties the project together, pointing to the main takeaways.

Chapter 2

Related Works

2.1 Learning-based Approaches For Navigation

Robotic navigation tasks have typically been solved using a paradigm of building a geometric map and localizing the robot within the built environment to solve the task at hand [8]. Such methods have enabled current state-of-the-art methods, commonly referred to as SLAM (simultaneous mapping and localization). However, these methods face some shortcomings, perhaps most importantly, learning from prior experiences[8]. Learning-based approaches challenge these methods by leveraging prior experiences to learn navigation tasks directly. Learning-based approaches encapsulate a large field of machine learning, from reinforcement learning approaches to supervised learning methods. Here, we will present different approaches to solving navigation tasks through supervised learning.

An example of a supervised learning approach to solve navigation tasks is the LaND algorithm. Recent technological advancements have seen an increased amount of autonomous robots in practical applications such as sidewalk delivery. By learning from disengagement, the LaND method seeks to improve safe navigation on sidewalks in urban environments. Although disengagements have typically been used for debugging and troubleshooting, the proposed method in [9] attempts to use disengagement directly in the learning pipeline. The premise of learning from disengagement starts with an autonomous robot navigating a sidewalk. For data collection, whenever the robot is headed in the wrong direction, the robot is disengaged through human supervision, resetting the course of the robot. This creates the data set labeled through human involvement. During training, the LaND algorithm predicts disengagement probabilities of different action sequences through a recurrent CNN neural network architecture, penalizing differences between predicted and actual disengagement in a supervised fashion. The results show that the robot can travel significantly longer distances on a sidewalk compared to other methods using reinforcement learning and behavioral cloning, am imitation learning approach[9].

A limitation of the LaND method is having a human supervisor when collecting data. Similarly, other methods have previously required human supervision to learn collision avoidance[23, 24]. The Berkeley Autonomous Driving Ground Robot or BADGR [10] is a 4-wheeled robot that collects data and learns collision avoidance without human supervision. Simultaneously, BAGDR can reason about geometric as well as non-geometric navigation. This is achieved by BADGR collection data on actions and observations in the environment before labeling the data set through self-supervision. Furthermore, contrary to predicting disengagement[9], the robot is trained to predict future events, which are driving, bumpy driving, or collision, in a supervised manner through a deep neural network. The network processes the data and uses a recurrent network structure to predict future events. Through training, BADGR can navigate to a desired goal position in the environment based on some predefined reward structure, penalizing bumpiness and collisions. The results even showed the robot's willingness to traverse areas with tall grass to increase the cumulative rewards. This is something traditional geometric approaches, such as SLAM, have generally struggled with, showing that there is no explicit mapping between geometric and physical properties[10].

ORACLE[11] is another deep neural network structure trained in simulation and used for planning and navigation. Contrary to BADGR and LaND, ORACLE is deployed on a micro aerial vehicle (MAV) in a cluttered environment. The MAV is used in simulation to collect data, labeling it self-supervised, similar to BADGR. The data collected are actions and observed depth alongside the robot's state and a collision index. A deep collision predictor network is used for training, learning collision probabilities from the collected data of different action sequences. ORACLE also adopts the idea of using a *motion primitives library*[25] where steering commands are methodically drawn within the field of view of the robot to generate action sequences. Additionally, ORACLE utilizes unscented transform on the robot's state and Monte Carlo dropout in the deep collision predictor network to account for state and model uncertainty. When deployed on the robot, ORACLE safely traverses a previously unseen cluttered environment based on a safe navigation policy, penalizing collisions along the way. This result proves valuable as it shows that the learned policy generalized well to the physical world, despite being trained in simulation, bridging the gap between simulation and the real world. However, the ORACLE model was restricted to planar movement and low velocities, indicating that further improvement is possible.

Perhaps the best results for learning-based approaches for collision-free navigation for a micro aerial vehicle in cluttered environments is the work of Loquerio et al.[26]. The method in [26] is based on simulations, utilizing imitation learning through a privileged expert with perfect knowledge of the states and the environments. The model learns to predict the three best trajectories for collision-free flight based on partial knowledge of the robot's states and depth images created through semi-global matching in simulated environments. This proved sufficient for flight in the real world as the developed model managed to traverse previously unseen environments at high speed, up to $10m/s$. The results illustrate how the method is robust to sensor noise, illumination differences, and low-texture surfaces in the real world while performing complex navigation tasks, outperforming traditional state-of-the-art obstacle avoidance pipelines. For an aerial robotic to navigate at such high velocities has enormous

potential in various applications. A significant benefit to autonomous high-speed navigation is the omission of expert human pilots, who often require years of training, making using experts expensive and a scarce recourse. Hence, the result of [26] proved a baseline for further development, showcasing the enormous potential of aerial robotics.

2.2 Representation Learning In Robotic Applications

As learning-based approaches in robotics have become more widespread, so have convolutional neural networks to capture features from structured data, namely images. This has sparked interest in representation learning, a learning-based approach to extracting valuable representations from images. Where LiDARs and depth cameras provide large amounts of redundant data that often require tailored hardware for computation, deep learning-based approaches can extract useful information on large amounts of data while removing the need for feature engineering or heuristics[17].

ANYmal is a quadrupedal robot that utilizes representation learning to extract features from a mounted camera. Using a variational autoencoder to encode the images removes redundant information, only mapping salient features relevant for collision avoidance. Furthermore, the model deployed on ANYmal[17] uses an LSTM recurrent network on the encoded features along with the camera's trajectory to estimate the environment's temporal evolution. Furthermore, a reinforcement learning algorithm is used to develop a navigation policy, considering the environment's evolution. The results in simulation saw a collision rate of 3%, demonstrating why incorporating representation learning and temporal awareness is beneficial. When traversing a real and static environment, the robot managed to navigate and mostly avoid obstacles within the field of view in a static environment, indicating that the latent representation from the VAE is very effective for sim-to-real transfer. Furthermore, this portrays that the latent representation does not differ between simulation and reality [17]. However, the collision rate increased in real dynamic environments, illustrating that there is still room for improvement. Nonetheless, the results are encouraging as they illustrate how representation learning can be used to bridge the reality gap.

Depth sensors, such as RGB-D cameras, have become a core component in many robotic tasks, including navigation. Depth cameras often come with some particular problems. Firstly, they suffer from limited range and produce images with noise and missing data. Secondly, such cameras have trouble capturing objects with bright surfaces, creating holes or noise in the resulting depth map[19]. Not surprisingly, this may cause problems in navigation tasks. Popovíc et al. [19] solves the issue of incomplete depth images by using Bayesian deep learning, a VAE, for depth completion in the context of robotic mapping. The network architecture consists of an encoder, creating a latent representation for features in the images, and two decoders, one for reconstructing the input image and one for depth uncertainty. By combining the two outputs, a probabilistic depth map is created. The network also incorporates *self-attentions*, which are skip-connections within the network relating different positions in a sequence[27]. Self-attention is mostly associated with *transformers*, a recurrent neural network structure. However, using this structure for depth completion, and creating probabilistic depth maps,

resulted in more discovered free space compared to mapping with raw depth images within the environment being mapped. Though, this was on data from the InteriorNet data set[28]. However, this translated to the real world, where probabilistic depth completion consistently yielded faster free space discovery. These results signify that the latent representation is effective in sim-to-real transfer.

Chapter 3

Theoretical Background

The landscape that is machine learning is vast but can generally be separated into three genres: supervised learning, unsupervised learning, and reinforcement learning. In this chapter, the aim is to present *supervised* and *unsupervised* learning in simple terms before elaborating on more complex general features of neural networks in the context of *deep learning*. Furthermore, we present *representation learning*, a domain of unsupervised learning, with focus on *variational autoencoders* (VAEs). Finally, *semi-global matching* (SGM), a computer vision algorithm for depth imagery, is presented.

3.1 Supervised Learning

As the name suggests, supervised learning is a machine learning technique where a *supervisor* guides the learning process. This is not to be confused by a human supervisor but instead supervised by the data at hand. In supervised learning, the data set can be summarized as a collection of labeled examples on the form $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$. Each element of $\boldsymbol{x}_i$ among N is called a feature vector corresponding to the labeled output y_i. The feature vector is a vector in which each dimension $j = 1, ..., D$ contains information that somehow describes the example. The label y_i can be either an element belonging to a finite set of classes $\{1, 2, ..., C\}$, a real number, or a more complex structure, like a vector [29].

Supervised learning can be separated into two domains: regression and classification. Regression is a problem of predicting a continuous value, a quantitative measurement, from the input. Predicting housing prices is an example of regression where the output, the predicted price, is a mapping from various input variables such as size and location. On the other hand, classification is a mapping from input to discrete outputs, a qualitative measurement [29]. An example of a classification problem could be detecting spam email, in which the goal is to detect spam from a set of emails. In this classification problem there would be a finite set of classes, namely $\{\text{spam}, \text{not\_spam}\}$. This is known as a binary classification problem.

The goal of supervised learning is to use the data set to produce a model, $f(\boldsymbol{x})$, that takes in the input feature vector $\boldsymbol{x}_i$ and outputs information that allows deducing the label for the feature vector [29]. In practice, this is commonly done through artificial neural networks, which will be discussed in more detail in section 3.3.1. The output of the model, $\hat{y} = f(\boldsymbol{x})$, is used to guide to model towards the actual label y through a loss function, also referred to as an objective function. The loss function can be plenty, depending on the problem at hand. However, formally, it is a function taking the predicted output and the actual output as inputs and returning a scalar value, measuring how far the predicted output is from the actual output[29].

$$\mathcal{L}(y, \hat{y}) \in \mathbb{R} \tag{3.1}$$

Minimizing the difference between y and $\hat{y}$ will improve the model $f(\boldsymbol{x})$, enforcing the predicted output closer to the labeled truth. Observing the loss function and whether it decreases gives a good indication of if the model is improving at the task. However, it offers little value when determining the model's performance. For a binary classification problem, accuracy gives a good measurement of the performance of the developed model. Accuracy is the total number of correct classifications over the total amount of test cases.

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}} \tag{3.2}$$

Furthermore, there are four possible outcomes when making a classification for a binary classification problem. These are the following: classified as true(1) when the label was true(1), classified as true(1) when the label was false(0), classified as false(0) when the label was true(1), or classified as false(0) when the label was false(0). These four different classifications are referred to as true positive (TP), false negative (FN), false positive (FP), and true negative (TN), respectively. These can be summarized in a confusion matrix, visualized in figure 3.1[29].

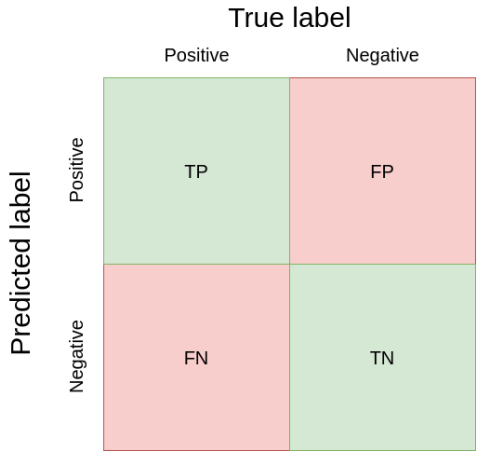


Figure 3.1: Confusion matrix

From the confusion matrix in figure 3.1 it is clear that classifications that fall into the category of true positive (TP) and true negative (TN) are correct guesses and will yield good results from an accuracy standpoint. In contrast, false positives (FP) and false negatives (FN) are incorrect guesses and will drag the overall accuracy down.

Now, lets consider the exmaple of detecting spam e-mail again. Imaging the data set contains 100000 data points where 5000 data points are positive, spam, and 95000 data points are negative, not spam. By classifying all cases as negative (not spam) the model would achieve an accuracy of 95%, which from a pure mathematical perspective would seem like a good result. However, in reality no spam mails where detected when in fact there are 5000 spam mail. This would make that particular model useless for practical use. Hence, accuracy would not be a good measurement for such a data set. This will, in fact, be the case for a wide variety of data sets where there exist a great imbalance between negative and positive labels. For such data set, precision, recall [30] and F1-score [31] are far better measurements to determine performance. The formulas for precision and recall are the following

$$\text{Precision} = \frac{TruePositives}{Truepositives + FalsePositives} = \frac{TP}{TP + FP} \tag{3.3}$$

$$\text{Recall} = \frac{TruePositives}{TruePositives + FalseNegatives} = \frac{TP}{TP + FN} \tag{3.4}$$

Precision is a measurement to illustrate how many times the model classified correctly given that the model predicted true or positive. Recall, on the other hand, tells how often the model classified positives correctly given the actual number of positive cases there are in the data [30]. Given these relationships, recall and precision will give reasonable metrics regarding performance in imbalanced data sets. Consider the example of detecting spam mail again; if all mails were classified as not spam, the accuracy would still be 95%. However, the recall and precision would be 0%, indicating that the model is not well suited to detect spam emails. Alternatively, if the model classified all emails as spam, it would yield an accuracy of 5%. Similarly, the precision would also be 5% because of all the false positives. However, the recall would be 100% because there would be no false negatives. Hence, classifying all emails as spam would not yield a good model either. Instead, accuracy, precision, and recall need to be evaluated together to determine whether the model is well suited for the task at hand. Alternatively, it is possible to evaluate the F1-score, which is the harmonic mean of precision and recall [31]. Thus, F1-score gives a weighted average for the performance of imbalanced data sets.

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.5}$$

These performance metrics show that a good model would yield a high F1-score, also resulting in high accuracy, precision, and recall. In essence, this means that the model can accurately identify which emails are spam, while still keeping the number of classified trues within a reasonable range.

3.2 Unsupervised Learning

Contrary to supervised learning is unsupervised learning, where, as the name suggests, a model is learning from data without any supervision. The data set can then be summarized as a collection of feature vectors $\{\boldsymbol{x}_i\}_{i=1}^{N}$ without corresponding labels. Similarly to supervised learning, each element of $\boldsymbol{x}_i$ among N is a feature vector in which each dimension $j = 1, ..., D$ contains information that somehow describes the example. Unsupervised learning aims to transform the feature vector into a scalar or another vector that can be used to solve a practical problem. Because of the absence of corresponding labels, it is often more problematic to use for practical problems than supervised learning. Still, unsupervised learning has several practical applications, including clustering and dimensionality reduction [29].

Perhaps the most well-known unsupervised learning problem is clustering. Clustering is a problem of learning to assign labels to examples by leveraging unlabeled data. Clustering algorithms, such as k-means clustering, use the mean-squared-error function for optimization, where data points are assigned to the closest cluster center, the centroid that minimizes the mean-squared-error. The centroids are then relocated to the mean position of the points in the cluster. Data points are then assigned to the clusters again, minimizing the loss function and repeating the process until the loss objective converges, meaning that the centroids are fixed, and all data points are assigned to a cluster. K-means is well suited to large amounts of data and is thus also used in union with big data techniques. Additionally, k-means can also be used for dimensionality reduction, and outlier detection [29].

Data sets often come with many high-dimensional examples, with up to millions of features per example. Dimensionality reduction handles the high-dimensional data and compresses it into a space of lower dimension, extracting the most important features. Principal component analysis (PCA) is an algorithm that compresses the data onto orthogonal vectors, where the vectors are in the direction where the largest variance in the data is found [29]. Autoencoders are another method used for dimensionality reduction. An autoencoder takes the high-dimensional data and encodes it into a space of lower dimension, called a latent space, before reconstructing the data from the encoded latent space. Then, through training, the reconstructed data is compared to the input data, enforcing an encoded representation to contain the most important data in the input. This is also known as representation learning, with the primary goal to make subsequent tasks easier [16].

The absence of labels in unsupervised learning makes evaluating performance harder than in supervised learning. Performance metrics, such as accuracy, precision, and recall, do not exist for unsupervised learning problems. Consequently, the performance of a model is solely guided by the loss objective, which will vary depending on the task at hand.

3.3 Neural Networks and Deep Learning

3.3.1 Artificial Neural Networks

Artificial neural networks, or ANNs for short, are computational graphs well suited for complex optimization problems where the optimal solution may lay hidden in heaps of data. ANNs are fundamental building blocks for most machine learning tasks and are sufficiently used in a majority of learning-related tasks.

Artificial neural networks are inspired by human biology, specifically the human brain, and how signals are transmitted. Signals and impulses are transmitted through connections from one neuron to another. From a computer science perspective, the connections and neurons are similar to edges and vertices in a graph. To better understand how artificial neural networks are structured, let us consider the simplest form of neural networks, the feed-forward neural network, also referred to as a multi-layer perceptron (MLP), in the literature. The feed-forward network, visualized in figure 3.2, is a directed acyclic graph consisting of vertices/neurons and edges with weights. Sequential vertical nodes in the network comprise a layer, and the network incorporates three different layers. The input layer, visualized as blue nodes, is where the input data is fed into the network. The input layer is required to be of the same dimension as the data. Then, there is an output layer, visualized in red, which is the network's output. In between, there are hidden layers, visualized in green, which contain hidden representations of the data. The input is passed from one layer to another through the edges. This forward pass from one layer to another can be formulated as the following equation:

$$\boldsymbol{h}_i = \sigma(\boldsymbol{W}_i \boldsymbol{h}_{i-1} + \boldsymbol{b}_i) \tag{3.6}$$

Here, $\boldsymbol{h}_i$ is the output of layer i, organized as a vector. $\boldsymbol{h}_{i-1}$ is the output of the previous layer $i-1$, also organized as a vector. $\boldsymbol{W}_i$ is the weights along the edges between layers $i-1$ and i, formed as a matrix, and $\boldsymbol{b}_i$ is an additional bias term, again organized as a vector. σ is an activation function, often non-linear. The use of activation functions allows the network to learn representations of non-linear data, which is essential for most problems. Typical activation functions used in practice are the *sigmoid*, *tanh*, and *ReLU* functions.

The weights and biases are trainable, meaning they are subject to change. This makes artificial neural networks well suited to solve complex optimization problems. The weights and biases are trained through backpropagation. All subsequent neural network architectures presented are trained through backpropagation. Backpropagation can be viewed as the opposite of a forward pass, namely a backward pass through the network, from output to input. Backpropagation builds open the chain rule for derivatives, is presented in algorithm 1.

Algorithm 1 Backpropagation

1: Input: data $\boldsymbol{x}$
2: Weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$ for all layers are previously initialized
3: Forward pass the data according to $\boldsymbol{h}_i = \sigma(\boldsymbol{W}_i \boldsymbol{h}_{i-1} + \boldsymbol{b}_i)$ from input to output through the hidden layers
4: Compute the loss $\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})$ where $\hat{\boldsymbol{y}}$ is the predicted output and $\boldsymbol{y}$ is the true output
5: Compute the gradients $\nabla \boldsymbol{W} = \nabla_{\boldsymbol{W}} \mathcal{L}$ for all weights
6: Compute the gradients $\nabla \boldsymbol{b} = \nabla_{\boldsymbol{b}} \mathcal{L}$ for all biases

Once the gradients are computed, the weights and biases are updated according to an update rule. There are plenty of update rules, where perhaps the most well known is the gradient descent update rule, which is on the following form:

$$\boldsymbol{W} = \boldsymbol{W} - \alpha \nabla \boldsymbol{W} \tag{3.7}$$

$$\boldsymbol{b} = \boldsymbol{b} - \alpha \nabla \boldsymbol{b} \tag{3.8}$$

Here, α is the learning rate, which determines how much the weights and biases are updated. Other update rules used in practice are the *Adam*, *Adagrad*, *RMSprop*, and many more. These three update rules are all adaptive, meaning that the magnitude of which the weights and biases updates decreases when the solution approaches a local solution to the optimization problem. In the literature, the update rules fall under an umbrella term known as *gradient descent optimization* and are also referred to as optimizers [16]. The process of updating the weights and biases after data has been passed through the network is referred to as training. It forces parameters to move towards a solution that minimizes the loss objective.

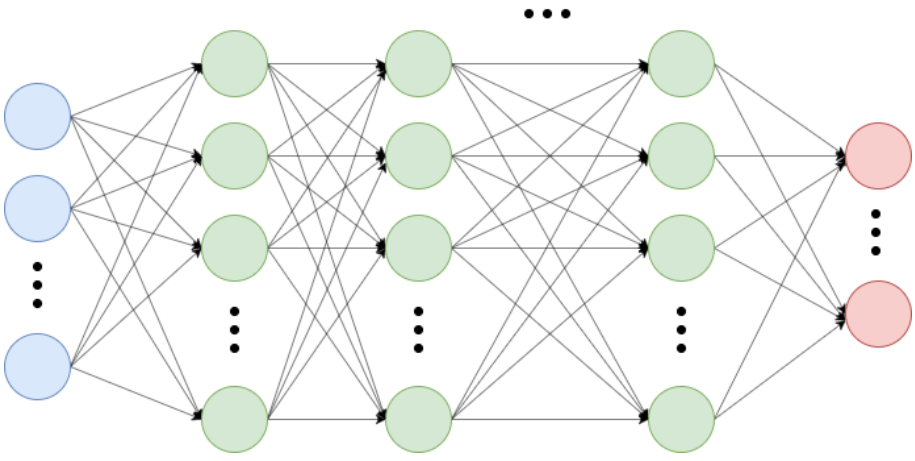


Figure 3.2: A feed-forward network. The blue nodes comprises the input layer, the green nodes comprises the hidden layers, and the red nodes comprises the output layer.

3.3.2 Monte Carlo Dropout and Batch Normalization

The main goal of training neural networks for optimization tasks is for the developed model to generalize to similar but new and unfamiliar data. Validation and test data sets facilitate this by being similar data not subject to training. Still, with the model only being trained on the training data, the model may move into a part of the optimization space where it performs well on the training data but struggles to perform equally well on unseen data. Hence, the model does not generalize well to other data. This is known as overfitting. There are multiple ways to reduce overfitting when training a model, namely regularization techniques. One form of regularization is called Monte Carlo dropout. With Monte Carlo dropout, nodes in the network are assigned a probability, known as *dropout rate*, to be switched off temporarily. This applies to input and hidden nodes but not to output nodes. There are several benefits of using dropout when training a neural network model. Firstly, by temporarily switching off nodes, there will be a collection of different neural networks that share hidden units, training on the same problem. This is known as ensemble in machine learning and has shown to be an effective regularization technique because the information is spread more evenly throughout the network, making the output less dependent on specific nodes in the network[16]. This can also be viewed as injecting noise into the model. Secondly, using dropout is computationally cheap compared to other ensemble methods, only requiring $O(n)$ computations per example[16]. An example of a neural network with Monte Carlo dropout can be seen in figure 3.3.

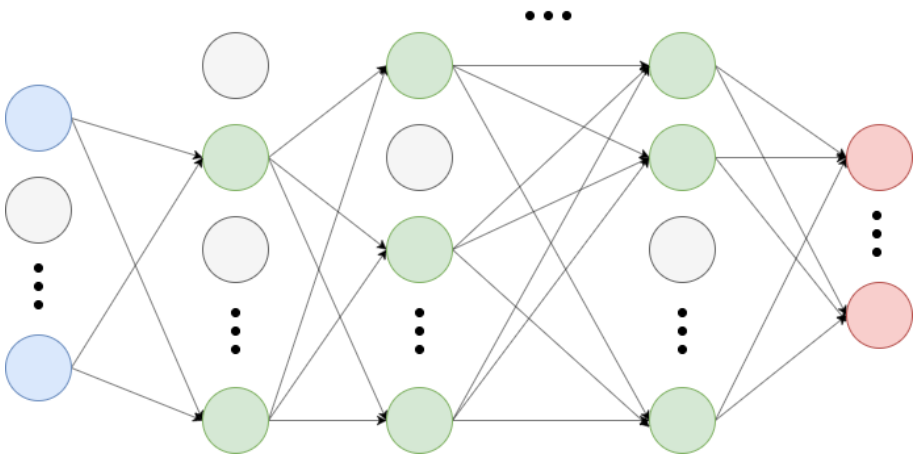


Figure 3.3: A feed-forward network where Monte Carlo dropout is used. The white nodes on input and hidden nodes represent a node being switched off.

Batch normalization is another regularization technique that adaptively reparameterizes the layers in a neural network. When updating a set of weights and biases in a network, the assumption is that the other weights and biases remain constant. Unfortunately, this is not the case as all weights and biases are updated simultaneously, which can cause the learning algorithm to forever chase a moving target. This may lead to unexpected results during optimization for specific problems. Batch normalization provides a reparameterization of the individual

layers, significantly reducing the problem of coordinating updates across the weights and biases. It is done by transforming the batch of data using the following equation [16]:

$$\boldsymbol{x}' = \frac{\boldsymbol{x} - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \tag{3.9}$$

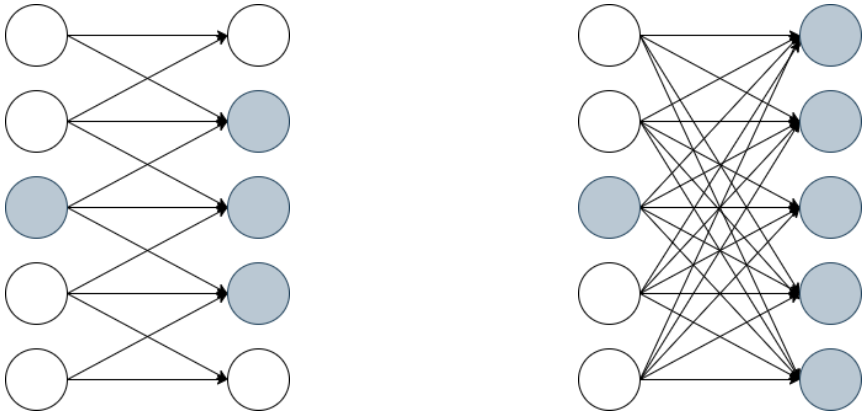
Where $\boldsymbol{x}'$ is the normalized data, $\boldsymbol{x}$ is the un-normalized data, and $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are the mean and variance of the data at a certain layer, respectively. By doing this transformation, the mean and variance are fixed for each layer speeding up the learning process. Similarly to dropout, using batch normalization on layers in a neural network is the equivalent of adding noise to the model, making it well suited as a regularization method [16].

3.3.3 Deep Learning

When the ANN has more than one hidden layer, we refer to the network as a deep neural network or a DNN for short. This is where the aspect of machine learning known as deep learning originates from. Adding layers to the networks increases the number of trainable parameters and allows the network to handle more complex data by finding more structures in the data. As mentioned previously, non-linearities are usually introduced to the hidden layers, which allows the DNN to learn representations of non-linear data, which is more complex.

3.3.4 Convolutional Neural Networks

Convolutional neural networks, or CNNs for short, are DNNs that utilize receptive fields to extract and learn features from structured data. The receptive fields in a CNN utilize shared weights, resulting in fewer weights, and sparse connections, meaning that the connections between neurons from one layer to another are one-to-some rather than one-to-all, which is the case for full connections. This is visualized in figure 3.4. Because CNNs are well suited for problems of structured data and hierarchical patterns, it is widely used for computer vision tasks such as image classification and image segmentation.



(a) Sparse connection between layers **(b)** Full connection between layers

Figure 3.4: The difference between sparse and full connections between layers in a neural network

CNNs get their name from the convolution operation, which accounts for the receptive fields. The convolution operator is often referred to as a kernel or a filter. Here the kernel or filter is overlapped and restricted to a part of the structured data, for example, an image. Sliding the receptive fields over the entire imagery allows for feature extraction of the structured data. The convolution operation is not to be confused with convolution in mathematics but is a dot product of the filter or kernel and the data. The kernel is a learnable tensor of values (weights). Like the weights in a feed-forward network, the filter contains weights from one layer to another. However, because of the sparse connections, resulting in fewer parameters, makes the model less computationally complex while still being a suitable architecture for computer vision tasks. The convolution operator from one layer to another is visualized in figure 3.5.

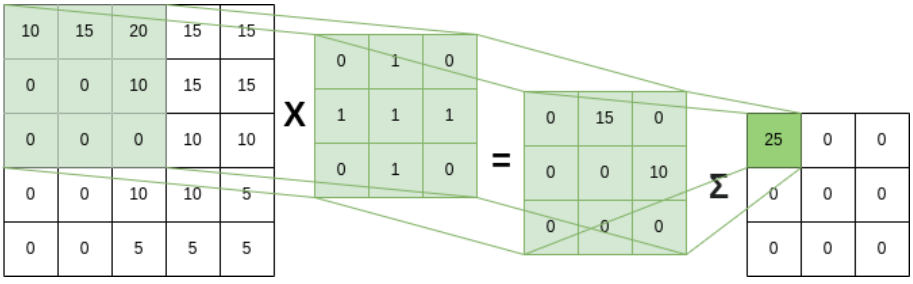


Figure 3.5: A 3×3×1 kernel applied to a 5×5×1 input

The output from a convolution operator is often referred to as a feature map or an activation map because of the feature extracting capabilities of the operator.

The filters in a CNN come with multiple tunable hyperparameters. It is usually common to use

multiple filters between layers. These filters can vary in size, where frequently used sizes are (2×2), (3×3), (5×5) and (7×7). Additionally, the movement of these filters, known as stride, is tunable. By using stride (1×1), the filter moves one step at a time, meaning that the filters will overlap across the data. Alternatively, using stride (2×2) in union with a (2×2) filter will result in no overlapping across the data.

Additionally, the convolution layer is often used along with a pooling layer, which is a form of down-sampling, also using filters. Contrary to convolution, the pooling filters are not learnable weights. The MaxPooling operator is frequently used. Here, the output from the convolution operator, the feature map, is sampled on the largest values within the pooling filter. The MaxPooling operator is visualized in figure 3.6 where a pooling filter of size (2×2) with strides (2×2) is used on a (4×4) feature map.

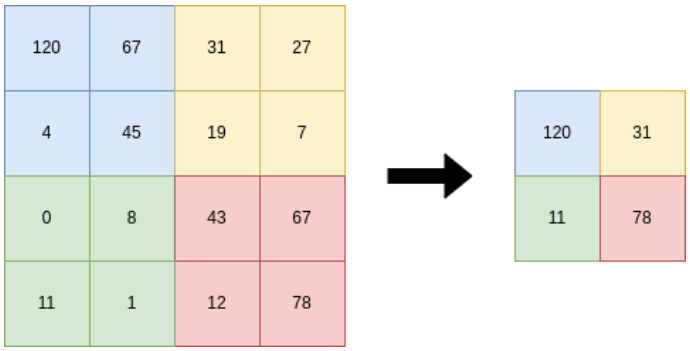


Figure 3.6: MaxPooling applied to a feature map. The colors indicate where the pooling has been applied.

Multiple convolution layers are usually stacked in CNNs, which allow more complex feature extraction. However, even though convolution operators are good at extracting features, it is rare to see CNNs without fully-connected layers following the convolution layers. This is because, however well the convolution layers are at extracting features, the features still need to be analyzed to make reasonable outputs.

3.3.5 Recurrent Neural Networks

Recurrent neural networks or RNNs are deep neural networks that consider the information of previous inputs when determining an output. The structure of an RNN is illustrated through figure 3.7 where a hidden representation at time t is passed to a hidden representation at time $t+1$. This can be viewed as having memory encoded into the network structure. Because of the recurrent architecture, RNNs are well suited to solve sequential tasks where previous inputs are perhaps as necessary as the current input to determine the current output.

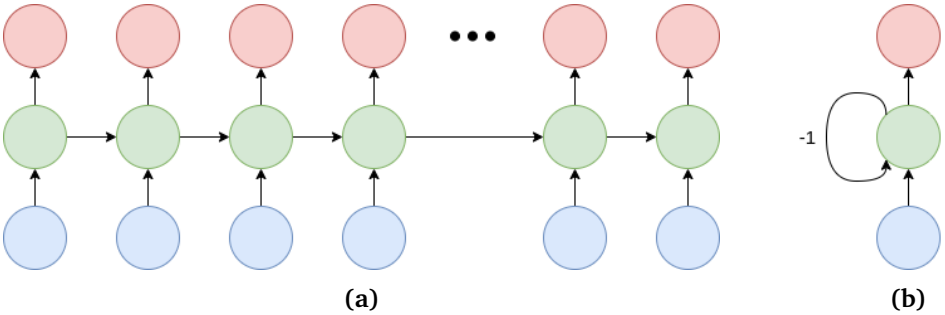


Figure 3.7: Two equal representations of a RNN. The −1 in figure 3.7b implies that the current hidden node receives information from the previous hidden node.

The RNN in figure 3.7 is often the typical visualization of an RNN where there are many inputs to many outputs. These types of networks are sequence-to-sequence, or many-to-many, RNNs that excel at natural language processing tasks such as machine translation, image captioning, and text summarization. However, recurrent nets are also well suited for other tasks such as pattern recognition and image classification. These problems require a different structure, given that only one output exists. Fortunately, RNNs can be modeled to fit such a problem by adopting a many-to-one architecture, with many inputs to only one output. Furthermore, RNNs can be adapted to various problems by alternating the RNN to fit the problem, whether that is many-to-many, many-to-one, or one-to-many [16].

The RNN architecture can be further expanded by adding more connections between the nodes in the network. Different possible connections are *teacher learning* connections, where the previous true output is fed to the hidden states; skip-connection, where a hidden node may be influenced directly by multiple other hidden nodes; and many more. There is also possible to create bi-directional RNNs where there exist multiple hidden nodes where information is passed forward through time on some nodes and backward in time through other nodes. Hence, in bi-directional RNNs, the output may depend on previous, and future inputs [16]. For simplicity and the purpose of this thesis, we will only further expand upon the already presented structure in figure 3.7.

Similarly to feed-forward networks, information is passed through the network following some formulas. These can be summarized in the following way

$$\boldsymbol{h}_i = \sigma_h(\boldsymbol{U}\boldsymbol{x}_i + \boldsymbol{W}\boldsymbol{h}_{i-1} + \boldsymbol{b}_h) \tag{3.10}$$

$$\boldsymbol{y}_i = \sigma_y(\boldsymbol{V}\boldsymbol{h}_i + \boldsymbol{b}_y) \tag{3.11}$$

Here σ_h and σ_y are activation functions, usually the *tanh* function; $\boldsymbol{x}_i$, $\boldsymbol{h}_i$ and $\boldsymbol{y}_i$ are the input, hidden state and output at time i; $\boldsymbol{U}$, $\boldsymbol{W}$ and $\boldsymbol{V}$ are weight matrices and $\boldsymbol{b}_h$ and $\boldsymbol{b}_y$ are biases.

When training an RNN, a problem can occur, which is known as the *vanishing gradient problem*. Notice how the same matrices are used repeatedly between the network layers. When backpropagating through the network, the gradient is calculated over all connections in the network. If we consider an RNN lacking non-linear activations, we can think of the recurrence relation

$$\boldsymbol{h}_i = \boldsymbol{W}\boldsymbol{h}_{i-1} \tag{3.12}$$

Which lacks the input $\boldsymbol{x}$. This recurrence relation essentially describes the power method and can simplifies as

$$\boldsymbol{h}_i = \boldsymbol{W}^i\boldsymbol{h}_0 \tag{3.13}$$

where $\boldsymbol{W}$ can be transformed to the eigencomposition

$$\boldsymbol{W} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T \tag{3.14}$$

Here, $\boldsymbol{\Lambda}$ is a diagonal matrix, and $\boldsymbol{Q}$ is orthogonal, simplifying the recurrence as:

$$\boldsymbol{h}_i = \boldsymbol{Q}\boldsymbol{\Lambda}^i\boldsymbol{Q}^T\boldsymbol{h}_0 \tag{3.15}$$

The eigenvalues are raised to the power of i, causing the eigenvalues less than one to decay while the eigenvalues larger than one to explode. Eventually, any component of $\boldsymbol{h}_0$ that is not aligned with the largest eigenvalue will be discarded [16]. This relationship will cause *gradients to either vanish or explode*, which makes learning long-term dependencies much harder. For example, it has been shown that gradients reach 0 for sequences of only length 10 or 20 [16], which puts learning longer sequences at a significant disadvantage. Various recurrent networks, such as long short-term memory (LSTM) solve the issue of the vanishing gradient problem by cleverly manipulating the way memory is preserved in the network.

3.3.6 Long Short-Term Memory

Long short-term memory, or LSTM for short, is a recurrent neural network architecture that preserves the gradient of the hidden states through gated units. The LSTM hence falls into a category of RNNs called *gated RNNs* which are widely used in practical applications [16]. The LSTM, and gated RNNs in general, are based on the idea of creating paths through the network where the gradients neither vanish nor explode. This allows the LSTM architecture to accumulate information over previous inputs and then discard the information when it is no longer of use. Two types of information are passed through the network, short-term memory, which is incorporated into the hidden state, and long-term memory, in a separate cell state. Determining what information gets passed on is done using three different gate types: a forget gate, an input gate, and an output gate.

The *forget gate* decides what information in the previous cell state $\boldsymbol{C}_{i-1}$ to discard [32]. The forget gate takes the input $\boldsymbol{x}_i$, and the hidden state $\boldsymbol{h}_{i-1}$ as input and outputs a mask between zero and one on what information to keep or discard. The sigmoid activation function is used on the forget gate to create the mask. The forget gate is given as:

$$\boldsymbol{f}_i = \sigma(\boldsymbol{W}_f \cdot [\boldsymbol{h}_{i-1}, \boldsymbol{x}_i] + \boldsymbol{b}_f) \tag{3.16}$$

Where $\boldsymbol{W}_f$ and $\boldsymbol{b}_f$ are weights and biases, respectively. $[\boldsymbol{h}_{i-1}, \boldsymbol{x}_i]$ indicates that the hidden state and the input are concatenated.

Next, the LSTM cell decides on what new information we will store in the cell state based on the two inputs, the hidden state, and the input. This has two parts. Firstly, the two inputs are passed through the *input gate*, containing the sigmoid activation function, to create an input mask $\boldsymbol{j}$. The input mask determines which parts of the cell state to update. Next, a candidate cell state $\tilde{\boldsymbol{C}}_i$ is made by passing the hidden state and input through a tanh function [32]. This is done through the following equations:

$$\boldsymbol{j}_i = \sigma(\boldsymbol{W}_j \cdot [\boldsymbol{h}_{i-1}, \boldsymbol{x}_i] + \boldsymbol{b}_j) \tag{3.17}$$

$$\tilde{\boldsymbol{C}}_i = tanh(\boldsymbol{W}_C \cdot [\boldsymbol{h}_{i-1}, \boldsymbol{x}_i] + \boldsymbol{b}_C) \tag{3.18}$$

Similarly to the forget gate, $\boldsymbol{W}_j$ and $\boldsymbol{W}_C$ are weights, and $\boldsymbol{b}_j$ and $\boldsymbol{b}_C$ are biases, with $[\boldsymbol{h}_{i-1}, \boldsymbol{x}_i]$ indicating that the hidden state and the input are concatenated.

After the input and short-term memory are used to create the forget mask, the input mask, and the candidate cell state, the old cell state is updated into a new one. The updated cell state will carry the long-term memory to the next LSTM cell. First, the new cell state is made by combining the previous cell state $\boldsymbol{C}_{i-1}$ and the forget mask $\boldsymbol{f}_i$, determining what information to keep or discard. Then, the candidate state $\tilde{\boldsymbol{C}}_i$, combined with the input mask $\boldsymbol{j}_i$, is added to the cell state, updating the long-term memory.

$$\boldsymbol{C}_i = \boldsymbol{f}_i \cdot \boldsymbol{C}_{i-1} + \boldsymbol{j}_i \cdot \tilde{\boldsymbol{C}}_i \tag{3.19}$$

Finally, with the new cell state being made, the output needs to be determined. The output is a filtered version of the new cell state combined with an output mask through the *output gate*. The filtered cell state is made by passing the state through the tanh function to push the values between -1 and 1. The output mask is made similarly to how the input and forget masks are made [32].

$$\boldsymbol{o}_i = \sigma(\boldsymbol{W}_o \cdot [\boldsymbol{h}_{i-1}, \boldsymbol{x}_i] + \boldsymbol{b}_o) \tag{3.20}$$

$$\boldsymbol{h}_i = \boldsymbol{o}_i \cdot tanh(\boldsymbol{C}_i) \tag{3.21}$$

The internal structure of the LSTM cell is visualized in figure 3.8. Similar to the recurrent network visualized in figure 3.7, it is common to use multiple LSTM cells to represent the network's hidden layer. Then, the output $\boldsymbol{h}_i$ acts as the output of the cell and the hidden state for the next LSTM cell. Multiple LSTM cells coupled in a network are visualized in figure 3.9.

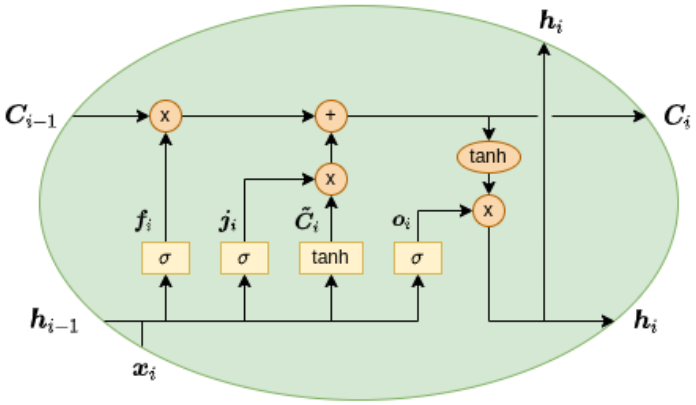


Figure 3.8: The LSTM cell

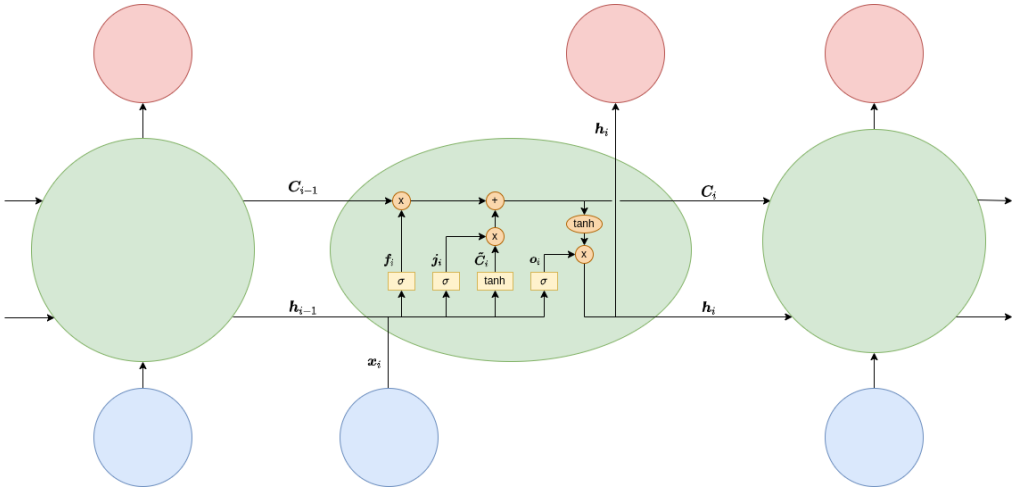


Figure 3.9: A LSTM RNN

3.4 Representation Learning

As the name suggests, representation learning aims to learn a representation of the data at hand. Learning a representation encapsulates both supervised and unsupervised learning. However, *representation learning* refers to unsupervised learning where learning is conditioned on intermediate feature [16].

Determining what a good representation is is ambiguous, but generally speaking, a good representation is a representation that makes subsequent learning easier. Autoencoders (AE) and variational autoencoders (VAE) are two models that excel at representation learning by learning a representation of the data encoded into a latent space. In particular, AEs and VAEs have successfully been applied to dimensionality reduction and information retrieval tasks, where

the learned latent variables represent a compressed version of the data [16].

3.4.1 Autoencoders

An autoencoder is a neural network that learns a representation of the input by attempting to copy its input $\boldsymbol{x}$ to its output $\tilde{\boldsymbol{x}}$. The representation is encoded into an internal latent space $\boldsymbol{z}$ and describes the input. The network architecture consist of two parts, namely an *encoder* and a *decoder*. The encoder maps the input to the latent variables $\boldsymbol{z} = f(\boldsymbol{x})$ and the decoder produces a reconstruction of the input from the latent variable $\tilde{\boldsymbol{x}} = g(\boldsymbol{z})$ [16]. The encoder and decoder can be any arbitrary neural network structure, such as a feed-forward network, or more complex networks such as a CNN or an RNN. From a machine learning perspective, the autoencoder is visualized in figure 3.10. The color scheme in the figure may seem confusing as the hidden latent space $\boldsymbol{z}$ is marked in red, indicating an output in previous visualizations. In the sense of an autoencoder, and the unsupervised learning task is optimized, the reconstructed input is the output. However, in the context of representation learning, the latent space contains the encoded representation, which is of further use in subsequent learning tasks. Hence it is marked in red to indicate the output of the model suited for other tasks.

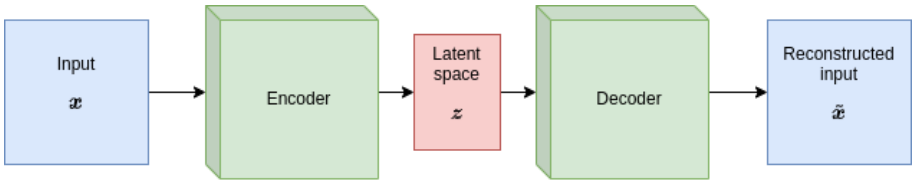
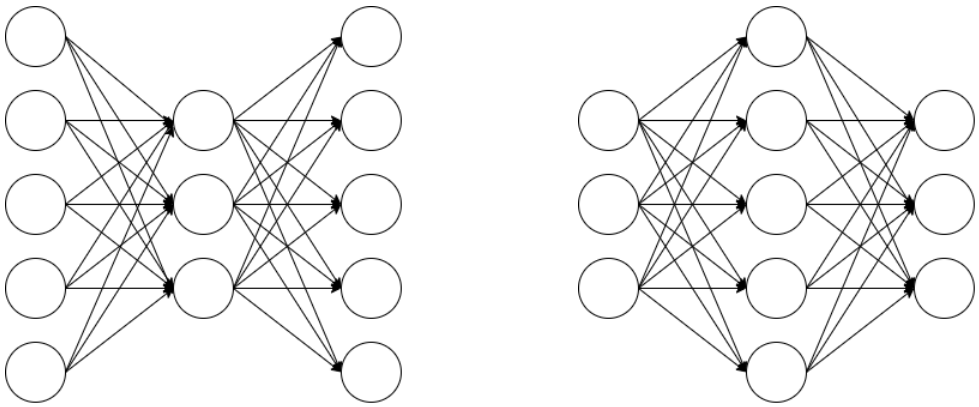


Figure 3.10: Architecture of an autoencoder

If the AE is able to perfectly reconstruct the data $g(f(\boldsymbol{x})) = \boldsymbol{x}$, then it is not especially useful as it will copy the input without extracting valuable information. This happens if the encoder and decoder are given too much capacity. Instead, autoencoders are designed not to be able to perfectly reconstruct the data by restricting the network size or the learning. If the AE is restricted by its size, it is referred to as an *undercomplete autoencoder*. In an undercomplete autoencoder, the hidden layers are smaller than the input, creating a bottleneck for information to pass through. This forces the AE to learn an undercomplete representation of the most salient features in the data. On the other hand, if the learning restricts the autoencoder, it is referred to as a *regularized autoencoder*. Regularized autoencoders can also be *overcomplete*, meaning that the hidden layers are larger than the input. The regularized autoencoders use loss functions that encourage the model to learn other properties through, as suggested by the name, regularization techniques [16]. An example of an undercomplete and an overcomplete autoencoder is visualized in figure 3.11. For this thesis, we will only further focus on the undercomplete autoencoder.



(a) An example of an ndercomplete autoencoder **(b)** An example of an overcomplete autoencoder

Figure 3.11: The difference between an undercomplete and an overcomplete autoencoder

The undercomplete autoencoder is trained by minimizing a loss function

$$\mathcal{L}(\boldsymbol{x}, g(f(\boldsymbol{x}))) = \mathcal{L}(\boldsymbol{x}, \tilde{\boldsymbol{x}}) \tag{3.22}$$

where the loss penalizes dissimilarities between the input and reconstructed input. The loss function can for example be the mean squared error or the binary cross-entropy functions.

3.4.2 Variational Autoencoders

The autoencoder is inherently deterministic, which raises the question of whether it is possible to combine representation learning in union with probabilistic models. The variational autoencoder or VAE is a neural network architecture that allows the modeling of probabilistic models in machine learning [33]. Similar to the AE, the VAE is well suited for representation learning and dimensionality reduction, accounting for stochastic data.

Similarly to other unsupervised learning tasks, let us consider a data set $\{\boldsymbol{x}_i\}_{i=1}^{N}$ consisting of N independent and identically distributed samples of a continuous and discrete variable $\boldsymbol{x}$. This data is assumed to be generated by the unobserved and continuous random latent variable $\boldsymbol{z}$, where the latent variable $\boldsymbol{z}$ is generated from a prior distribution $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ and the generated data $\boldsymbol{x}_i$ is generated from some conditional distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$. The probability for the prior and the likelihood are assumed to originate from the same parametric family $\boldsymbol{\theta}$ and are differentiable with respect to both $\boldsymbol{z}$ and $\boldsymbol{\theta}$ almost everywhere [34].

With the VAEs goal of being used for representation learning the goal is to properly being able to reconstruct the data. Hence, the goal is to find the probability distribution for the data $p(\boldsymbol{x})$. Unfortunately, the marginal likelihood $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$ is intractable, as we can not evaluate or differentiate the marginal likelihood. Alternatively, using Bayes rule to find the pos-

terior distribution $p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x}) = p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})/p_{\boldsymbol{\theta}}(\boldsymbol{x})$ unfortunately runs into the same problem of being intractable. These intractabilities are common for moderately complicated likelihood functions $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$, for example when modelled with a neural network with non-linear activations. Hence, in order to approximate the true posterior, a recognition model $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$ is used instead [34].

From a strict machine learning perspective, we have all the building blocks required. The prior distribution $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ represent the latent space $\boldsymbol{z}$, the *encoder* is the recognition model $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$, approximating the true posterior, and the *decoder* is the conditional distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$. The decoder is also referred to as a *generative model* in the literature. Finally, the reconstruction of the data is the marginal likelihood $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$.

Next, like any other machine learning optimization problem, a loss function is needed. The loss function stems from the marginal likelihood, which can be rewritten as:

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i) = D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}_i) || p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x}_i)) + \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}_i) \tag{3.23}$$

Where the marginal likelihood is decomposed into two terms. The first term is the KL divergence of the approximate from the true posterior. The second term $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}_i)$ is called the evidence lower bound or ELBO for short. The evidence lower bound is sometimes referred to as the variational lower bound in literature. The ELBO can be further rewritten as:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}_i) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})}\left[-\log q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})\right] \tag{3.24}$$

Alternatively, the evidence lower bound can be rewritten be re-arranging equation 3.23 as the following:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}_i) = -D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}_i) || p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}_i)}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i \mid \boldsymbol{z})] \tag{3.25}$$

Notice how both equation 3.24 and 3.25 include the encoder and decoder models, meaning that optimizing any of the two losses will train the VAE architecture. Still, with the two different losses, there are two different approaches to optimizing the ELBO. The first approach is to estimate the loss term in equation 3.24 through Monte Carlo estimation. This yields the first Stochastic Gradient Variational Bayes (SGVB) estimator:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}_i) \simeq \tilde{\mathcal{L}}^A(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}_i) = \frac{1}{L}\sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i, \boldsymbol{z}_{i,j}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}_{i,j} \mid \boldsymbol{x}_i) \tag{3.26}$$

Alternatively, the KL divergence can often be integrated analytically, such that only the expected reconstruction loss error $\mathbb{E}_{q_{\phi}(\boldsymbol{z}|\boldsymbol{x}_i)}[\log p_{\theta}(\boldsymbol{x}_i|\boldsymbol{z})]$ (equation 3.25) requires estimation by sampling. Because the KL divergence can be analytically computed, it serves as a regualarizing term on the encoder parameters $\boldsymbol{\phi}$. With this, it is possible to build a second SGVB estimator from equation 3.25:

$$\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi};\boldsymbol{x}_i) \simeq \tilde{\mathcal{L}}^B(\boldsymbol{\theta},\boldsymbol{\phi};\boldsymbol{x}_i) = -D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_i)||p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \frac{1}{L}\sum_{l=1}^{L}\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i,\boldsymbol{z}_{i,j}) \tag{3.27}$$

Both estimators, $\tilde{\mathcal{L}}^A(\boldsymbol{\theta},\boldsymbol{\phi};\boldsymbol{x}_i)$ and $\tilde{\mathcal{L}}^B(\boldsymbol{\theta},\boldsymbol{\phi};\boldsymbol{x}_i)$ are valid estimators for the evidence lower bound. However, $\tilde{\mathcal{L}}^B(\boldsymbol{\theta},\boldsymbol{\phi};\boldsymbol{x}_i)$ typically yields lower variance, compared to its alternative[34].

One problem we now run into is that the latent variable or latent space $\boldsymbol{z}$ is stochastic. Therefore, it is not possible to run back-propagation through random nodes as it is not guaranteed to have an expectation. The VAE solves this issue by introducing a simple reparameterization trick, shifting the stochasticity from the latent variable to an auxiliary noise variable $\boldsymbol{\epsilon}$. By introducing the noise variable $\boldsymbol{\epsilon}$, it is possible to express the latent variable $\boldsymbol{z}$ through 3 separate components: a mean, variance, and the auxiliary noise. The reparameterization trick is done through the following equation:

$$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \tag{3.28}$$

where $\boldsymbol{\mu}$ is the mean, $\boldsymbol{\sigma}$ is the variance, and $\boldsymbol{\epsilon}$ is the standard normal distribution

$$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I}) \tag{3.29}$$

The reparameterization trick is illustrated in figure 3.12 and showcases how the stochastic element of the latent space is in the auxiliary noise node $\boldsymbol{\epsilon}$. Because the latent space $\boldsymbol{z}$, the encoded mean $\boldsymbol{\mu}$ and the encoded variance $\boldsymbol{\sigma}$ are now deterministic, it is possible to perform back propagation through the network, through the gradients of the loss estimate $\tilde{\mathcal{L}}(\boldsymbol{\theta},\boldsymbol{\phi};\boldsymbol{x}_i)$.

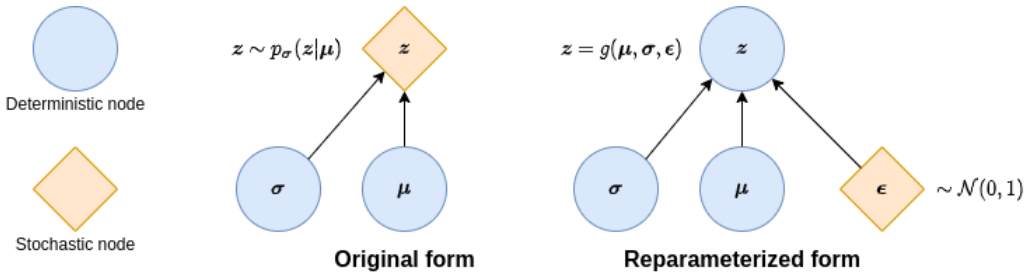


Figure 3.12: The reparameterization trick

With the latent variable now containing three different components, the autoencoder structure from figure 3.10 needs to be slight changes. The deterministic mean and variance are now the output of the encoder, while the auxiliary noise is drawn from a standard normal distribution. In addition, the mean and variance output nodes' dimensions need to be the same size as the latent variable. In practice, the encoder output is twice the size of the decoder input. The new architecture of the VAE is visualized in figure 3.13. The color scheme in figure 3.13 is the same as in the previously presented architecture for the autoencoder, showcasing that the latent representation of the input contains the representation of use for other subsequent tasks.

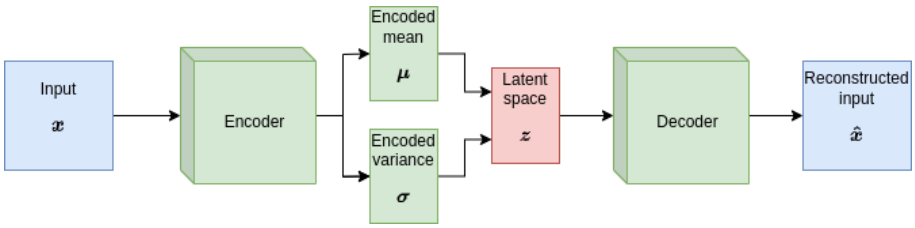


Figure 3.13: Architecture of a variational autoencoder

For practical use cases, the VAE has some advantages over the regular AE that are worth exploring. First, the VAE being a probabilistic model, allows for the modeling of probabilistic data, adding an element of uncertainty to the modeling. This is of great use in several applications, including autonomous driving[35]. Next, modeling the latent space as an isotropic Gaussian through the auxiliary noise variable $N \sim (0, \boldsymbol{I})$, forces the latent space to learn independent features. This is not the case for regular autoencoders as the latent space is fully connected to both the encoder and decoder, meaning that a single feature can be represented through multiple nodes in the latent space. Because of the independent latent nodes in the VAE, it can be used to generate data by sampling the latent space. In general, machine learning tasks require a lot of data in order to generalize well to unseen data. Thus, having the possibility to create data artificially is helpful for many tasks. The capability to generate data is also why the VAE is referred to as a *generative model* in the literature. Additionally, modeling a task as a probabilistic model allows us to reduce the cost of learning and inference[16]. Inference in machine learning refers to a forward pass of live data through a model.

3.5 Depth From Stereo Imagery and Semi-Global Matching

Depth images in simulations tend to be perfect, making the imagery significantly different from real-world images, thus creating a reality gap. Combining different images in simulation is more likely to introduce noise and uncertainties, making the differences between simulation and the real world smaller. The simplest way of obtaining multiple images is through a stereo camera, which records left and right images, also known as stereo images, at a given frequency. Using stereo images for visual perception and depth is far from new. Semi-global matching (SGM) is one algorithm that excels at finding the differences between a stereo pair, which can be used to measure depth. One particular advantage of SGM compared to several other

visual algorithms is its run time, which is faster and more predictable. More specifically, it is linear with the dimension of the images and the number of disparities, $O(WHD)$, where W and H are the width and the height of the image, and D is the disparity range which is a tunable parameter. Using a more extensive disparity range, D, will make the algorithm more robust as it increases the search area for matching. However, it will increase the run time of the algorithm[36].

Semi-global matching uses the respective images to create a disparity map, illustrating the differences between the stereo pair. The algorithm creates the output map through a four-step sequential process; matching cost computation, cost aggregation, disparity computation, and disparity refinement. The method is based on pixel-wise matching of mutual information in many different 1D constraints to approximate a global 2D smoothness constraint. [36].

Matching cost computation
The input images are assumed to have epipolar geometry, meaning that the two different cameras are planar with the possibility of distortion. It is, however, not assumed that the images are rectified (without any distortion) as it may not always be possible.

Let us consider a pixel, **p**, from an base image, I_b, with pixel intensity $I_{b\boldsymbol{p}}$. The suspected corresponding pixel, **q**, from a matching image, I_m, with similar intensity $I_{m\boldsymbol{q}}$ can be formulated as $\boldsymbol{q} = e_{bm}(\boldsymbol{p}, d)$, with the function $e_{bm}(\boldsymbol{p}, d)$ symbolizing the epipolar line distance or disparity, d, between pixel **p** and pixel **q** in the base and matching images, respectively. The disparity parameter, d, is a tunable parameter and defines max disparity allowed when matching the pixels **p** and **q**. One possible way of calculating the cost is through the minimum difference of intensities at **p** and **q** in the range of half a pixel in each direction along the epipolar line. This cost calculation is developed by Birchfield and Tomasi [37] and is used as the cost calculation for SGM by OpenCV, a computer vision library. Over all pixels **p** the cost calculations can be reformulated as[36]

$$\boldsymbol{q} = e_{bm}(\boldsymbol{p}, D_{\boldsymbol{p}}) = C(\boldsymbol{p}, D_{\boldsymbol{p}}) \tag{3.30}$$

Alternatively, the cost computation can be calculated through mutual information, which is based on calculating entropy across the images. However, this will not be presented in this project as the OpenCV implementation of semi-global matching is based on the cost computation of Birchfield and Tomasi[37].

Cost aggregation
Cost calculation is generally ambiguous, and wrong matches can easily occur when operating pixel-wise. Noise is a big contributor to wrong matches where a cost value can be lower than the correct pixel. Hence, there are added smoothness constraints to penalize changes in neighboring disparities. This is expressed through an energy function, $E(D)$

$$E(D) = \sum_{\boldsymbol{p}} \left(C(\boldsymbol{p}, D_{\boldsymbol{p}}) + \sum_{\boldsymbol{q} \in N_p} P_1 T\left[|D_{\boldsymbol{p}} - D_{\boldsymbol{q}}| = 1 \right] + \sum_{\boldsymbol{q} \in N_p} P_2 T\left[|D_{\boldsymbol{p}} - D_{\boldsymbol{q}}| > 1 \right] \right) \tag{3.31}$$

Where the $T[]$-operator is 1 if its argument is true and 0 otherwise. The first term of the equation is familiar from equation 3.30 and is the pixel matching cost for the disparities of D. The other terms adds constraints penalties P_1 and P_2 to neighboring pixels, the neighborhood $N_{\boldsymbol{p}}$ for the pixel $\boldsymbol{q}$. The second term with penalty constant P_1 is added if the disparity in the neighboring pixels change with 1. Similarly the penalty constant P_2 is added if the neighboring pixels have a larger disparity change. Using a lower penalty for small changes (P_1) permits an adaptation to slanted or curved surfaces. The penalty for all larger changes (P_2) preserves discontinuities in disparity.

SGM is a "winner takes all" type of algorithm. Thus, finding the disparity map D can be formulated as minimizing the energy function $E(D)$. Unfortunately, a global minimization in 2D is NP-complete when the image has many discontinuities. Alternatively, minimizing in 1D can be performed in linear time by using dynamic programming. However, solutions tend to suffer from streaking where undesirable inference is introduced, which can severely effect the result [38]. Hence, Hirschmüller, the creator of semi-global matching [36], introduced a new idea where the matching costs are aggregated in 1D from all direction equally. The aggregated cost $S(\boldsymbol{p}, d)$ is the minimum path that end in pixel $\boldsymbol{p}$ at disparity d. Furthermore the cost of traversing a path in direction $\boldsymbol{r}$ of the pixel $\boldsymbol{p}$ at disparity d is defined recursively

$$\begin{aligned} L_{\boldsymbol{r}} = {} & C(\boldsymbol{p}, d) \\ & + \min\Big(L_{\boldsymbol{r}}(\boldsymbol{p} - \boldsymbol{r}, d),\ L_{\boldsymbol{r}}(\boldsymbol{p} - \boldsymbol{r}, d-1) + P_1,\ L_{\boldsymbol{r}}(\boldsymbol{p} - \boldsymbol{r}, d+1) + P_1,\ \min_i L_{\boldsymbol{r}}(\boldsymbol{p} - \boldsymbol{r}, i) + P_2 \Big) \\ & - \min_k L_{\boldsymbol{r}}(\boldsymbol{p} - \boldsymbol{r}, k) \end{aligned} \tag{3.32}$$

The last term $\min_k L_{\boldsymbol{r}}(\boldsymbol{p} - \boldsymbol{r}, k)$ is subtracted to enforce numerical stability since it is constant for all values of disparity at the current pixel. The costs $L_{\boldsymbol{r}}$ are summed over paths in all direction, giving the following formula for the aggregated cost

$$S(\boldsymbol{p}, d) = \sum_{\boldsymbol{r}} L_{\boldsymbol{r}}(\boldsymbol{p}, d) \tag{3.33}$$

Disparity computation

The value of disparity at each pixel is given by $d^*(\boldsymbol{p}) = \arg\min_d S(\boldsymbol{p}, d)$. The two images in the stereo pair are not treated symmetrically in the cost calculation as one is used as a base image,

and the other is used as a matching image. Hence, a consistency check is added to enforce correct matching. This is done by creating two disparity maps, using both images as a base, and matching images, D_m, and D_b. The outliers of the disparity maps are filtered through a median filter before being compared. If a pixel in the two disparity maps differs, the pixel is invalidated. On the other hand, if the disparity at pixel $\boldsymbol{p}$ is consistent, it is set as valid:

$$D_{\boldsymbol{p}} = \begin{cases} D_{b\boldsymbol{p}} & \text{if } |D_{b\boldsymbol{p}} - D_{m\boldsymbol{q}}| \leq 1 \\ D_{inv} & \text{otherwise} \end{cases} \tag{3.34}$$

In the OpenCV implementation of SGM, D_{inv} is set to -1 to indicate an invalid match.

Disparity refinement
When computing the disparity maps, errors may occur even though a consistency check is implemented. Areas with invalid values can be recovered through post-processing. Three post-processing methods are used: *removal of peaks*, *intensity consistent disparity selection* and *discontinuity preserving interpolation*.

Peaks appear as small patches of disparity that is very different from the surrounding disparities. They usually appear in areas where noise is present; there is low to no texture or due to reflections. It is important not to confuse peaks from different disparities of valid structures. Hence, a predefined threshold is used such that smaller patches are unlikely to represent valid structures. The disparity image is segmented to identify peaks by allowing neighboring disparities within one segment to vary by one pixel, considering a 4-connected image grid. The disparities of these segments are set to invalid.

In structured indoor environments, foreground objects are often in front of low or untextured background objects, such as walls. This often results in fuzzy discontinuities around the edges of the foreground objects. Additionally, it is common to have discontinuities in the untextured background in proximity to foreground objects. In order to remove the discontinuities, semi-global matching uses fixed bandwidth Mean Shift Segmentation on the intensity image. Through surface hypothesis, the discontinuities are evaluated to either belong to a specific surface, thus inheriting the disparity intensity of that particular surface, or being independent of the hypothesis surface, thus maintaining its original value. This is performed for multiple hypotheses in what is referred to *intensity consistent disparity selection*.

The previously mentioned consistency check and peak filtering may invalidate some disparities, thus creating holes in the disparity image, which need to be interpolated for a dense resulting disparity map. The invalid disparities are classified into occlusions and mismatches and can be distinguished as part of the consistency check mentioned earlier. Mismatches are interpolated from neighboring pixels, whereas occlusions are extrapolated from the neighboring regions. This ensures correct smoothing in what is referred to as *consistency preserving interpolation*.

Depth from disparity

When the disparity map is created from the stereo pair, it is relatively straightforward to calculate the depth. Intuitively objects closer to the stereo camera will have a higher disparity than objects further away. Given this relationship, depth is proportionally inverse to the disparity. Hence, a larger pixel disparity will give a lower depth value and vice versa. Calculating the depth from disparity is dependent on the intrinsic parameters of the camera, namely the focal length, denoted f, which is the distance between the camera pinhole and the image plane; and the baseline denoted b, which is the distance between the left and right cameras in the stereo pair. The relationship between depth, the focal length, and the baseline is illustrated in figure 3.14

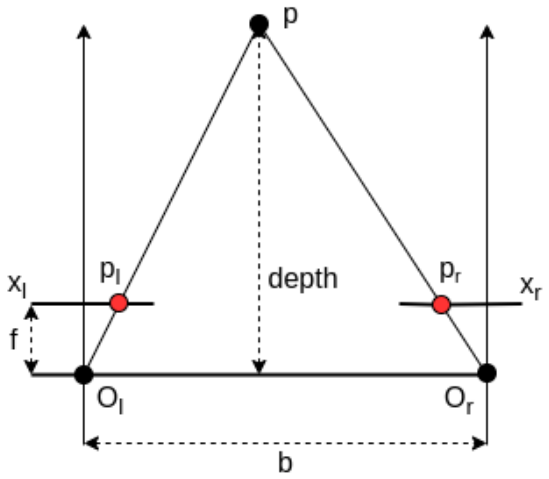


Figure 3.14: The relationship between the stereo pair and depth. The figure is inspired by lecture 12 from the course CSC420: *Introduction to Image Understanding* at The University of Toronto [39]

In figure 3.14 the optical centers are denoted O_l and O_r for the left and right cameras, respectively. Similarly, the image planes are denoted x_l and x_r, whereas the pixels p_l and p_r are the pixels in the left and right images corresponding to the pixel p. Pixel p yields the world point. The depth is hence the distance from the epipolar line of the stereo pair to the pixel p. The baseline and focal length are also illustrated in figure 3.14 as the distance between the optical centers, and the distance from the optical centers to the image planes. By comparing similar triangles we get the following relationship

$$\frac{b}{depth} = \frac{b + p_r - p_l}{depth - f} \tag{3.35}$$

By rearranging the equation we get the following relationship between depth and disparity

$$depth = \frac{f \cdot b}{disparity} \tag{3.36}$$

Where the disparity is $p_l - p_r$. The disparity is the disparity map created from semi-global matching. The focal length and disparity are measured in pixels, whereas the baseline is a distance in meters. Hence, the resulting depth map is also measured in meters.

Chapter 4

Problem Description

The main objective of this thesis is for a micro aerial vehicle to move from an initial position to a goal position through cluttered simulated environments. We assume the MAV has no access to maps of the environment but knows its position. Additionally, the MAV has partial knowledge of its surroundings through depth images captured in real-time. Let $s_t = [v_x, v_z, \omega]^T$ be the current state for forward and vertical velocity and angular velocity around the z-axis in the body frame. Additionally, let o_t be the current depth image. Furthermore, let $a_{t:t+H} = [a_t, a_{t+1}, ..., a_{t+H-1}]$ be an action sequence of length H where the action at every time step contains a forward and vertical velocity reference and a heading reference of the robot $a_{t+i} = [v^r_{x,t+i}, v^r_{z,t+i}, \psi^r_{t+i}]^T$. The goal is to find an optimized collision-free action sequence $a_{t:t+H}$ that ensures safe navigation towards a goal position based on observations about the robot's state and depth image. The goal position is assumed to be given by a global planner, which gives the robot information towards the heading direction of the goal position, ψ^g_t. The ORACLE deep neural network is designed for this particular navigation task. We intend to expand on the current architecture and simulation environment to increase sim-to-real transfer and further improve learning, enhancing the policy concerning safe navigation.

Chapter 5

Software Tools

5.1 Gazebo

Gazebo is an open-source three-dimensional dynamic robotics simulator that obeys the laws of physics[22]. By being built upon the Open Dynamics physics engine[40] and OpenGL[41] for rendering, it allows for realistically and accurately simulation of robots in complex environments. Furthermore, the use of Gazebo allows for integrating different hardware components, such as actuators, controllers, and sensors, bridging the reality gap between simulations and the physical world. Using a digital twin, in turn, allows for data collection, training, and evaluation through simulations rather than running a physical robot, which is incredibly beneficial concerning time and resources.

Furthermore, we have used an open-source Gazebo simulator named RotorS to simulate our quadrotor. RotorS is a realistic simulation framework designed for Micro Arial Vehicles (MAVs) [42]. It allows us to use a realistic replica of our real-world quadrotor in a user-designed environment while also packaging the necessary sensors, actuators, and controllers, needed in order to simulate the quadrotor robot functionally.

5.2 ROS - Robot Operating System

ROS or Robot Operating System is not an actual operating system but an open-source collection of software libraries for robot development, such as hardware abstraction, control, and message-passing. ROS allows users to build different robotic components and connect them using ROS tools called topics and messages. Topics allow different parts of the robotic system to subscribe to or publish messages, allowing for seamless and straightforward communication. Different topics may include channels for low-level controllers, typically used to publish references, and sensors, such as a stereo camera, which would be subscribed to. The messages can also be used with visualization tools, such as RViz[43], a ROS visualization library. Fur-

thermore, ROS is supported by multiple programming languages, including C++ and Python. This is beneficial because it means that different parts of the robotic systems can be written in different languages[21]. Using ROS with a wrapper, RotorS Wrapper, allows for seamless communication between different languages, namely C++, used for low-level control of the robot, and Python, used for data manipulation and machine learning.

5.3 TensorFlow

TensorFlow is an open-source software library in Python used for developing machine learning models. TensorFlow constructs computational, high-level structured graphs that allow for automatic differentiation of neural networks. Equipped in TensorFlow are also high-level application programming interfaces(APIs), such as Keras. TensorFlow and Keras offer the implementation of building blocks essential for machine learning. Among the different building blocks are all necessary neural network layers to build complex networks, such as fully connected layers, convolution, and deconvolution layers, recurrent layers, and pooling and upsampling layers. Furthermore, TensorFlow and Keras provide different activation functions, such as the ReLU and sigmoid functions, dropout, batch normalization, and different optimizers, objective functions, and regularizers. Finally, different data pre-processing tools are available, making for easy implementation of data pipelines.

5.4 OpenCV

OpenCV is a library developed by Intel to perform computer vision in real-time. The library is written in *C++*, but has a API for several different programming languages, including *Python*. Various algorithms are implemented in the library, including semi-global matching. For example, using OpenCV's implementation of SGM allows us to perform stereo matching in real-time, creating an effective pipeline for visual imagery.

5.5 Quadrotor Robot

The robot used in this thesis is a digital twin of the resilient micro flyer (RMF) [44]. The RMF falls under the category of micro aerial vehicles (MAVs). It is a light and agile robot built into a collision-tolerant frame, making it well-suited for navigation in cluttered environments. The digital twin used in simulation can be seen in figure 5.1.

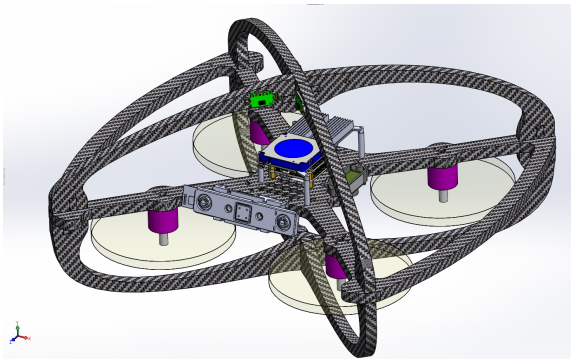


Figure 5.1: The digital twin of the RMF used in simulation. Image taken from `https://tiralonghipol.github.io/poldepetris/`

The robot is equipped with several controllers and sensors in simulation. Low-level controllers are used for roll, pitch, yaw, and altitude. The low-level controllers ensure stability, allowing reference signals to be utilized when navigating. Furthermore, the digital twin is equipped with an inertial measurement unit (IMU) consisting of a gyroscope, an accelerometer, and a stereo camera. The IMU gives three-axis measurements, providing information about the position, velocity, and angular velocity. The stereo camera provides images within its field of view, which is $86° \times 46°$. The range for the stereo camera is between 0.2 and 50 meters, and images are captured at a rate of 15 frames per second (fps), giving images of size 270×480 pixels. Furthermore, the stereo camera has a baseline of 9.5 cm with a focal length of 257.6223.

5.6 External Processing Power

Training deep machine learning models is computationally expensive. Hence, training is outsourced to a centralized computer equipped with multiple Nvidia GeForce RTX 3090 graphical processing units (GPUs). GPUs speed up training compared to central processing units (CPUs). The code and models developed are transferred between the external computer and the work computer, supplied by NTNU, through ssh pass.

Chapter 6

Proposed Approach

This chapter is split into five different sections. The first two sections focus on the expansions done to the neural network architecture and the environments in simulation. The third section presents how data is collected in a simulated environment. Furthermore, the second to last section presents the training of a stand-alone VAE and the expanded architecture outlined in the first section. Finally, the last section presents how the results are generated and evaluated.

6.1 Architectural Expansion

The ORACLE network is a deep collision predictor neural network used to predict collision probabilities of action sequences in real-time. The neural network architecture consists of multiple MLPs, a CNN, and a recurrent LSTM component. The model has three inputs: a filtered depth image, the robot's state, and action sequences generated by a motion primitives library. The filtered depth image is processed through the CNN and concatenated with the robot's state, processed through an MLP. The concatenated processed state is further processed through an MLP before being fed to the LSTM as the initial state. The action sequences processed through an MLP act as input to the LSTM as collision probabilities are eventually assigned to the different actions. The output for the LSTM is further processed, yielding the model's output, collision probabilities of the different action sequences. Furthermore, the robot's state is transformed through unscented transform, and the CNN utilized Monte Carlo dropout to account for state and model uncertainties, respectively[11] The architecture is visualized in figure 6.1.

There are currently a couple of challenges that the current architecture faces. Among them are the issue of running realistic experiments in simulations. Visual imagery, such as depth images from depth cameras, is notoriously different between simulations and the real world, creating a reality cap. In order to improve the sim-to-real transfer, the robot will create its depth images through stereo matching, using semi-global matching. The depth images are expected to be more akin to depth images generated by a depth camera under flight in the real world.

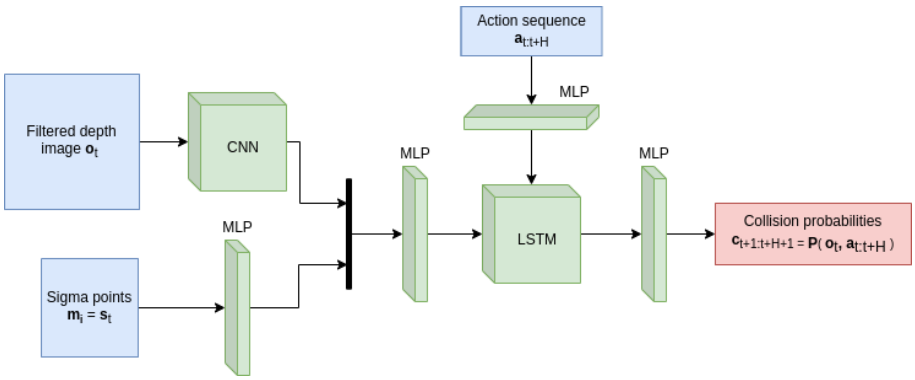


Figure 6.1: The architecture of ORACLE. MLP refers to Multi-Layer Perceptron which is identical to the feed-forward network previously discussed.

Furthermore, while CNNs are great at encoding structured data such as images, there is room for improvement. With CNNs, it is difficult to determine whether the encoded representation contains optimal information that benefits the learning process. Utilizing the concept of representation learning should conceivably create a more meaningful encoded representation while simultaneously improving sim-to-real transfer[17]. Accordingly, to obtain more meaningful encoded image representations, the architecture will be expanded to incorporate a variational autoencoder. Using a VAE instead of an AE is motivated by the reduced cost of training and inference[16], while also being more suited for autonomous driving through probabilistic modelling[35]. Presumably, it will make learning collision probabilities easier, improving safe navigation in cluttered environments.

The expanded ORACLE structure, which we will refer to as ORACLE-VAE, is visualized in figure 6.2 and can be separated into three different components: a VAE, a combiner network, and a predictor network. The VAE consists of the encoder and decoder and takes an image as input and returns the reconstructed input. The combiner network consists of the encoder and two multi-layer perceptrons (MLP) and combines the robot states and the latent representation from the visual imagery into a combined state. The last part is the predictor network which takes action sequences from a motion primitives library as input and the combined state as the initial hidden state to an LSTM recurrent net and outputs the collision probabilities of the action sequences.

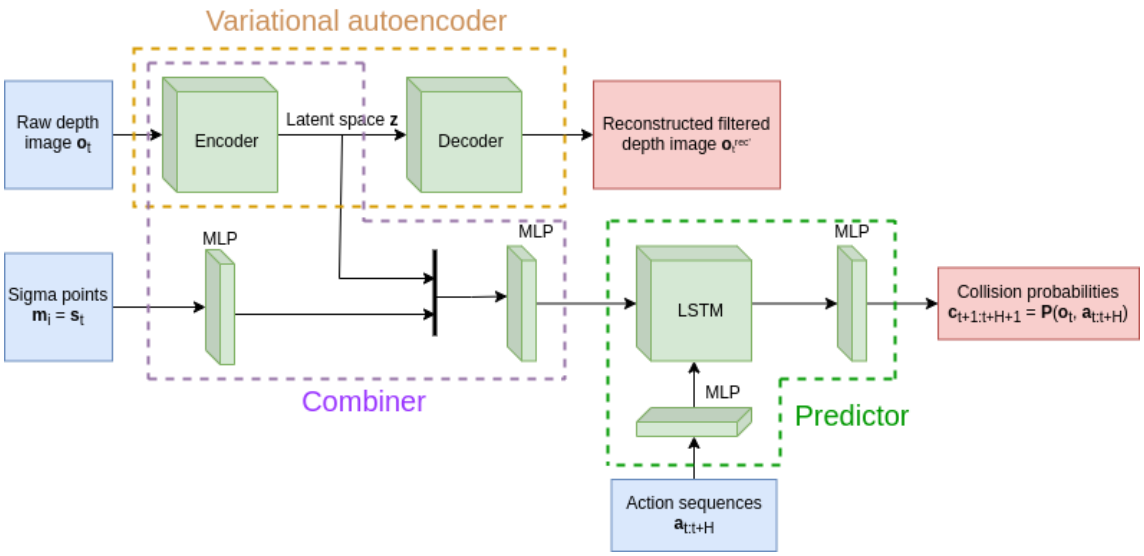


Figure 6.2: The architecture of the expanded ORACLE network. The model has 3 inputs and 2 outputs.

In an attempt to mimic the current architecture of ORACLE, the encoder and decoder of the VAE are based on ResNet8, a convolutional neural network. ResNet stands for residual network, a state-of-the-art neural network architecture for feature extraction related to image classification tasks[45]. The ResNet builds upon the residual block, visualized in figure 6.3. Here, skip-connections are used within the network, resulting in internal summations between hidden layers. The summation is described through the following equation:

$$y = F(x) + x \tag{6.1}$$

Here, y is the output of the residual block, and x is the input. The function $F(x)$ could be multiple layers of a neural network but is illustrated as two layers for simplicity in figure 6.3. The 8 in ResNet8 stands for the network's depth, containing eight convolution layers. The encoder network is visualized in figure 6.4.

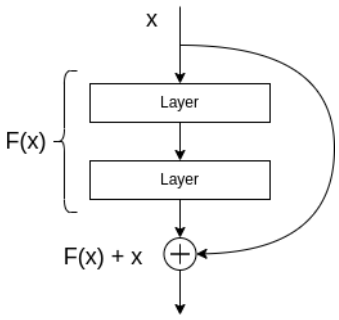


Figure 6.3: The residual block

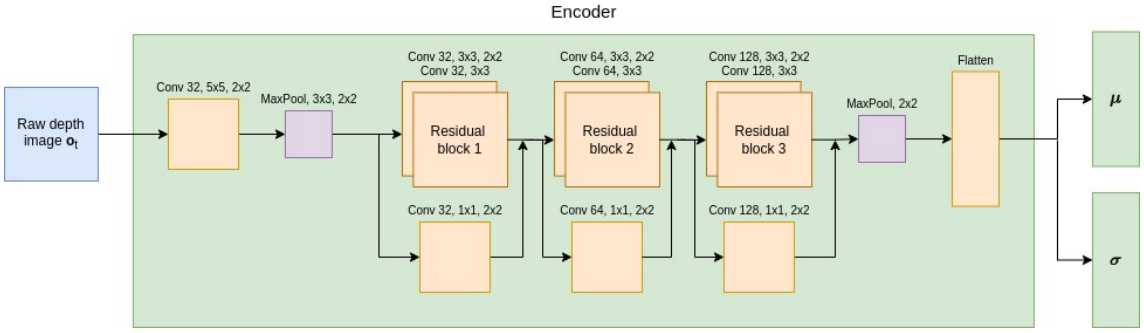


Figure 6.4: Residual encoder - ResNet8 CNN.
Every orange square represents a layer where all layers apart from the last one are convolution layers. The convolution layers are named as *Conv number_of_filters, filter_size, strides*. Additionally all the convolution layers uses batch normalization, ReLU activation and Monte Carlo dropout. For some of the layers, stride is 1×1 and not included in the graphic. The two purple layers are pooling layers and are names as *MaxPool, pooling_size, strides*. The input to the encoder is the raw depth image and the outputs are the mean and variance of the input.

The encoder in figure 6.4 does not follow the exact formula listed in equation 6.1, but rather on the form $y = F(x)+G(x)$ where $F(x)$ is two stacked convolution layers and $G(x)$ is one standalone convolution layer. The output of the encoder, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are both vectors with 64 elements. The resulting latent space, $\boldsymbol{z}$, will thus be of dimension 64 after the reparameterization trick is applied, previously discussed in section 3.4.2.

The stochastic latent space $\boldsymbol{z}$ will be the input to the decoder part of the VAE, which will architecturally mirror the ResNet encoder. The decoder is visualized in figure 6.5.

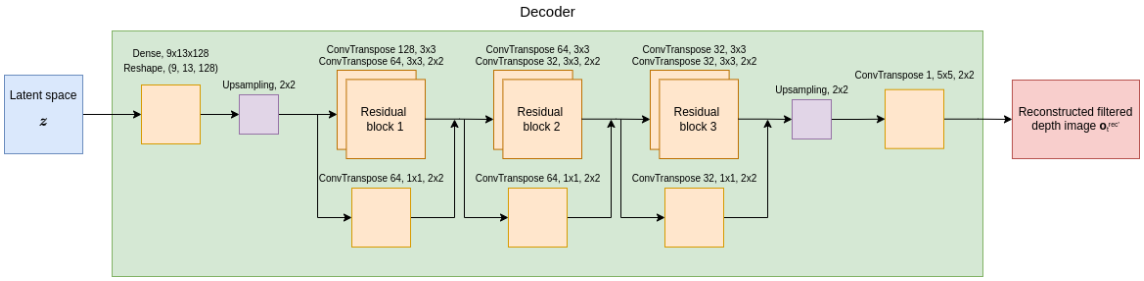
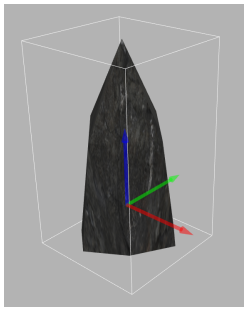


Figure 6.5: Residual decoder - A mirror image of the ResNet8 CNN.
Every orange square represents a layer where all layers apart from the first one are deconvolution layers, also known as transposed convolution in the TensorFlow Keras library. The de-convolution layers are named as *ConvTranspose number_of_filters, filter_size, strides*. Additionally all the de-convolution layers uses batch normalization, ReLU activation and Monte Carlo dropout. For some of the layers, stride is 1×1 and not included in the graphic. The two purple layers are up-sampling layers, which are the opposite of pooling, and are named as *UpSampling, pooling_size, strides*. The input to the decoder is the stochastic latent space and the output is the reconstructed image.

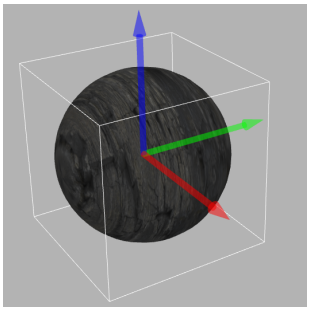
The ResNet-inspired decoder follows a similar structure to the encoder. Like the encoder, the decoder utilizes residual blocks on the form $y = F(x)+G(x)$, where $F(x)$ is two layers stacked and $G(x)$ is one stand-alone layer. The layers used in the model differ from the encoder to the decoder. Instead of convolution and pooling layers, the decoder utilizes de-convolution (ConvTranspose) and upsampling layers, which are inverse operations to convolution and pooling. The spatial dimensions are identical in the encoder and the decoder mirrored over the latent space. The resulting output of the decoder is a reconstructed image from the encoded latent space input.

6.2 Environmental Expansion

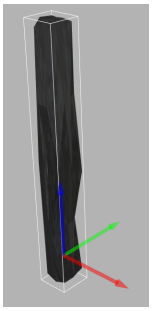
Semi-global matching is highly dependent on texture to function as a matching algorithm properly. Given that the RMF will create depth images through SGM, the environments used in simulation needs to be adapted to the task at hand. It is done by adding texture to all objects used in simulation. Three different environments will be utilized, one for data generation and two different ones for evaluation. These environments will have the same texture wrapped to walls and floor. The respective environment will later be illustrated in relevant sections. Furthermore, multiple copies of twelve different objects, varying in size, are manipulated, adding various textures to every object. These objects will be utilized when collecting data and evaluating the developed model. One example of each of the 12 objects is visualized in figure 6.6. The copies of pyramids, spheres, pillars and arches from figure 6.6a, 6.6b, 6.6c and 6.6d are wrapped with three different texture, all carrying stone-like characteristics. The $\perp$-figures, tables and chairs from figure 6.6e, 6.6f and 6.6g are wrapped with wood-like characteristics of three different colors: white, brown and dark brown. Furthermore, the three different fences in figure 6.6i, 6.6k and 6.6l are wrapped in different colors, namely, gray, beige, brown and dark brown. The wall from figure 6.6j comes with four textures: stone, brick, wood, and gray stone. Finally, the trees from figure 6.6h are as visualized.



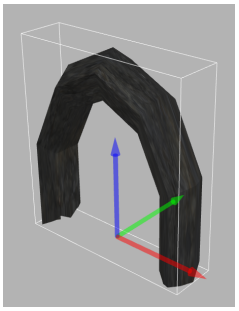
(a) Pyramid



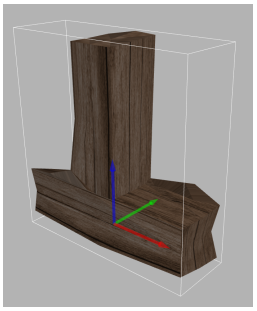
(b) Sphere



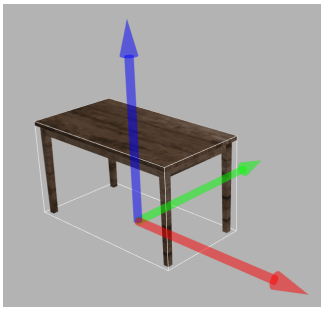
(c) Pillar



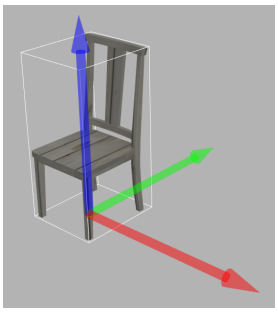
(d) Arch



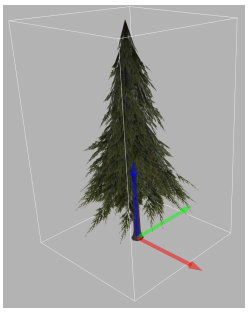
(e) ⊥ - figure



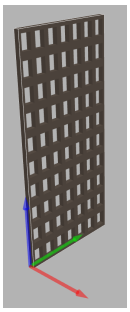
(f) Table



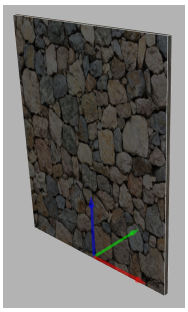
(g) Chair



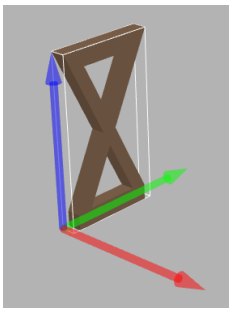
(h) Tree



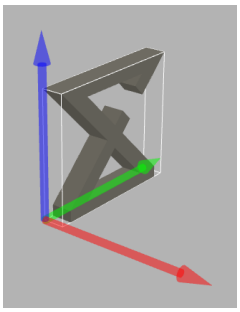
(i) Fence



(j) Wall



(k) Cross fence



(l) Cross fence with cut-out

Figure 6.6: One copy of each of the twelve different objects.

6.3 Data Collection

Many data points are required to train the ORACLE-VAE network. The data is collected in a simulated environment using the RotorS simulator[42]. In addition, low-level controllers are implemented to ensure sim-to-real transfer successfully.

The environment used for data collection is generated in a region spanning 40 by 40 meters, closed in by walls on all sides. The walls in the environment are covered by a stone texture, resembling gray stone walls. Furthermore, the floor is covered in segments resembling a toy world, adding texture to the floor (figure 6.7).

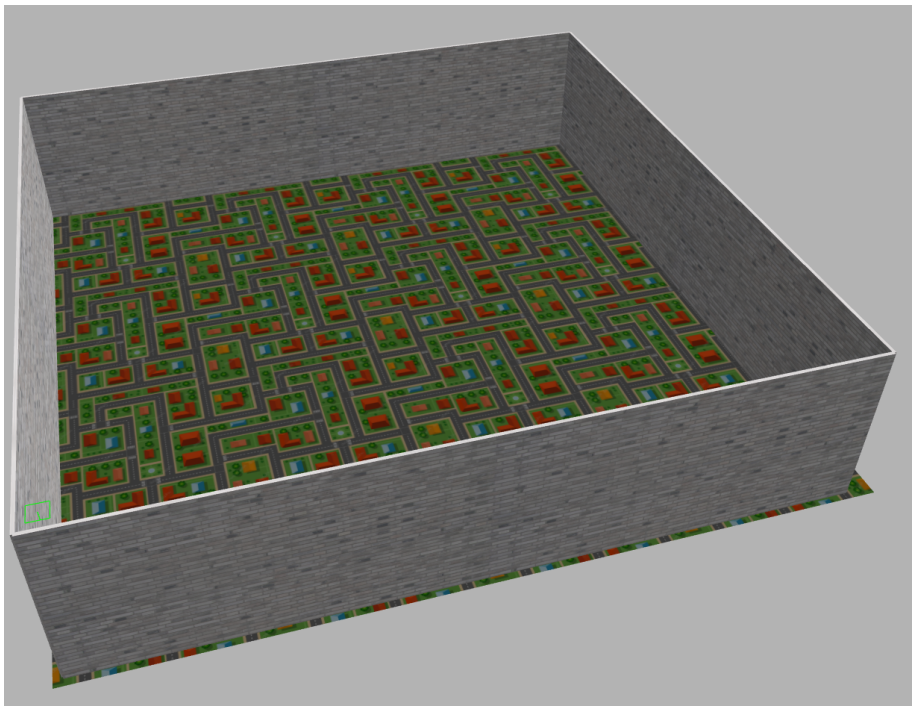


Figure 6.7: The environment used to collect data

Next, to eventually learn a safe navigation policy through collision avoidance, the model needs to learn from environments where obstacles are present. Hence, the environment for data collection is filled with the obstacles visualized in figure 6.6. The number of objects used is chosen randomly before being placed within the environment. The placement of chosen objects is all drawn from the same uniform distribution and placed correspondingly in the environment. A resulting environment is be visualized in figure 6.8.

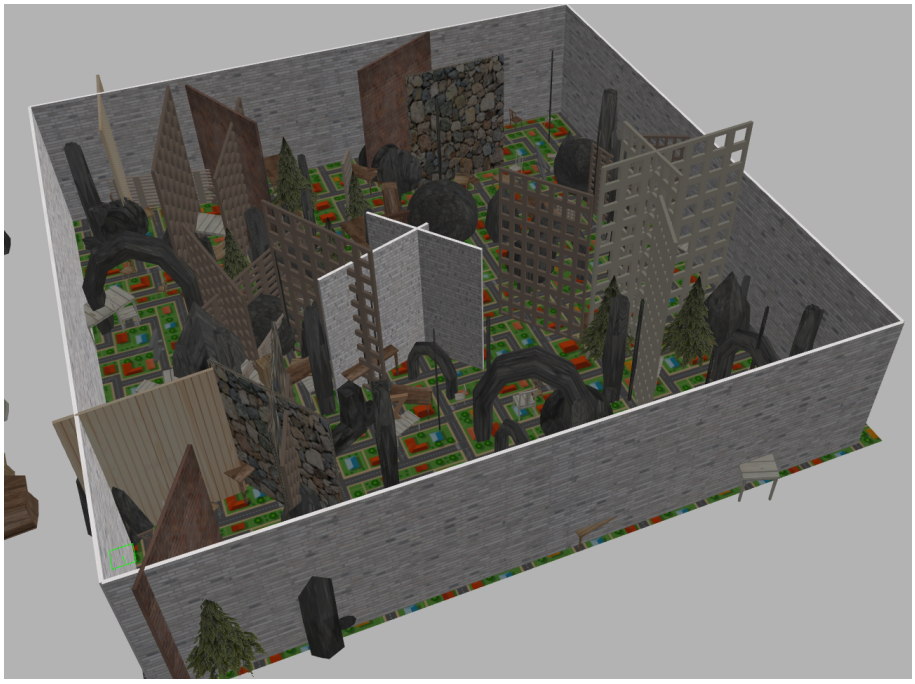


Figure 6.8: The environment used to collect data filled with obstacles

With the environment set, the robot can start collecting data. The MAV is spawned in the environment drawn from the same uniform distribution from which the objects are drawn. Then, a trajectory and an action sequence are drawn within the MAVs field of view (FOV) according to algorithm 2.

Algorithm 2 Motion Primitive Trajectory generation

Draw a relative yaw angle $\psi_{relative}$ within the horizontal field of view of the robot, $\psi_{relative} \sim U(-34°, 34°)$
Measure yaw ψ of the robot
Compute reference yaw angle $\psi_{ref} = \psi + \psi_{relative}$
Draw forward velocity from a uniform distribution $v_x \sim U(0.3, 2.0)$
Draw a reference tilt angle within the vertical field of view of the robot $\beta \sim U(-24°, 24°)$
Calculate vertical velocity $v_z = v_x \cdot tan(\beta)$ and clip it to be in the range $[-2.0, 2.0]$
Initialize acceleration, velocity and position in body frame:
$a_0 = (0, 0, 0)^T$
$v_0 = R_z(\psi_{relative}) \cdot (v_x, 0, v_z)^T$
$p_0 = (0, 0, 0)^T$
Set initial time $t = 0$
for $i \in$ number of steps before replanning **do**
    $t += \frac{1}{15}$
    $a = a_0$
    $v = at + v_0$
    $p = a\frac{t^2}{2} + vt + p_0$
    Create state estimate $state[i] = (p, v, a)$
    Create command to be executed $cmd[i] = (v_x, v_z, \psi_{ref})$
end for
Return state estimate $state$ and command cmd

The motion primitives from algorithm 2 contain a trajectory with positions, velocities, and accelerations, as well as actions to be executed, including velocities in x and z directions and a reference heading angle ψ_{ref}. If the action sequence is fully executed, new motion primitives are calculated, accounting for a new trajectory and actions. This will continue until the robot collides with an object or until a timeout occurs. In the case that a collision occurs at time step $t + k$, further actions, $a_{t+k}, k \leq H$, are randomly sampled until a timeout occurs. The timeout is set to be 30 seconds when collecting data. Executing actions until timeout accounts for an *episode* when generating data. Whenever the robot has moved more than $\Delta = 0.4$ meters a data point d is recorded, containing a raw and a filtered depth image, the robot's state, the action sequence and the collision status, $d = (o_t, o_t^{'}, s_t, a_{t:t+H}, c^{col}_{t+1:t+H+1})$. The collision index denoted the collision label for every time step $t+i, i = 1, ..., H$ and is a binary flag equal to zero for non-collision and one for collision. If the robot collides at one particular time step, $t+k$, all the remaining collision labels, $c^{col}_{t+k+1:t+H}$, are set to one. This ensures a more balanced data set where the number of action sequences with a collision label is roughly the same as collision-free data points. Furthermore, as the robot dynamics are holonomic and effectively invariant to its heading orientation, augmented data are stored, further diversifying the data set. The augmented data point is on the following form $d^{flip} = (o_t^{flip}, o_t^{flip'}, s_t^{flip}, a^{flip}_{t:t+H}, c^{col}_{t+1:t+H+1})$ where o_t^{flip} and $o_t^{flip'}$ are horizontally flipped images, the raw image and the filtered image, respectively; and s_t^{flip} and $a^{flip}_{t:t+H}$ are created by changing the sign of the angular velocity of

the robot and heading reference angle of the action sequence, ω_t and ψ^r_{t+1} respectively[11].

This procedure is repeated for a total of 20000 episodes. The data points are stored in separate files for every 40-th episode, one for the images, raw and filtered, one for the robot state, one for the action sequence, and one for the collision index. In addition, the environment is changed every 20-th episode to diversify the data set further. In total, this collects over $1,7$ million data points. The procedure of collecting data is summarized in algorithm 3.

Algorithm 3 Data collection

```
Initialize environment
for episode in number of episodes do
    Initialize robot position
    while Timeout has not occurred do
        if Robot has not collided then
            Generate trajectory and actions from motion primitives
        end if
        if Robot has collided then
            Sample actions randomly
        end if
        Execute actions
        Record data d and d^flip if the robot has travelled more than Δ meters.
    end while
    if episode % 20 = 0 then
        Reshuffle environment
    end if
    if episode % 40 = 0 then
        Store the recorded data to files
    end if
end for
```

As the micro aerial vehicle (MAV) is equipped with a stereo camera during the data generation phase, images from the stereo pair are recorded synchronously. Upon verifying that the left and right images are captured simultaneously, semi-global matching is performed through OpenCV's *StereoSGBM.create()*-function, creating a disparity map from the current stereo pair. The parameters chosen in the algorithms are inspired by [46]. After the disparity map is created, the depth map is created by combining the disparity map, the focal length, and the baseline through the relationship established in equation 3.36. Finally, the filtered depth image is created by applying a Gaussian filter to the resulting depth image.

6.4 Training

The training phase is executed in two separate steps. First, the VAE model is trained on its own. Then, the expanded neural network structure, ORACLE-VAE, from figure 6.2, is trained.

6.4.1 VAE

A current time constraint with the current ORACLE network is that image pre-processing, mainly filtering, is more time-consuming than a complete forward pass through the network. Hence, a goal of the VAE is to eliminate the need to filter the input images. The core idea is to force the encoder to act as a filter through training. This is done by passing a raw, unfiltered depth image through the encoder. The encoded latent space is fed through the decoder to reconstruct the image. This image will be trained against the filtered depth image, which has been stored during the data generation phase. Training will force the reconstructed image to look similar to the filtered image, indicating that the network structure can replace the filter currently used with ORACLE. The training procedure is illustrated in figure 6.9.

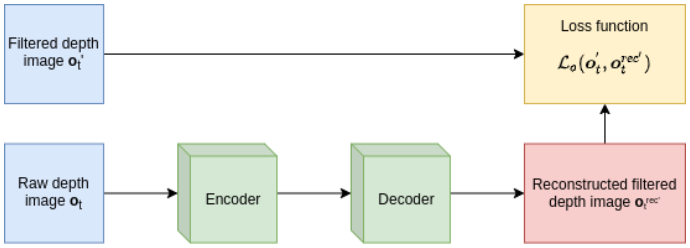


Figure 6.9: The training procedure for the VAE

When training the VAE, the objective function used will be the evidence lower bound or ELBO for short, established in section 3.4.2. In practice we optimize the single sample Monte Carlo estimate

$$\log p(x) \geq ELBO \approx \log p(x \mid z) + \log p(z) - \log q(z \mid x) \tag{6.2}$$

Here, $\log p(z)$ and $\log q(z \mid x)$ are calculated analytically. On the other hand, $\log p(x \mid z)$ is the reconstruction error. It is calculated through *binary cross entropy* for each pixel in the reconstructed output against the input. In practice, the evidence lower bound is a negative value. Therefore, maximizing the evidence lower bound will optimize the model. However, we will instead minimize the negative evidence lower bound. The change is made to obtain numerical stability when training the ORACLE-VAE model.

The VAE network is trained with the Adam optimizer with a learning rate of $1 \cdot 10^{-4}$ and decay of $1 \cdot 10^{-5}$. The network is trained through 100 epochs, 100 iterations through the data. The data used is roughly 230000 images for training and just shy of 58000 images for testing. Batch normalization is utilized, while Monte Carlo dropout is not used when training the stand-alone VAE.

6.4.2 ORACLE-VAE

Next, training the expanded ORACLE-VAE is a more complicated task as the model has two outputs, one for collision probabilities and one for the reconstructed image. The problem of predicting collision probabilities is a supervised learning task, whereas image reconstruction

through a VAE is an unsupervised learning task. Hence, in order to fit the new structure, two separate loss functions are needed. *Tensorflow* provides an excellent framework for such complex optimization problems, where multiple losses can be combined. Thus, the overall loss for ORACLE-VAE will accommodate the two learning processes by combining two loss functions through the following equation, allowing for end-to-end learning:

$$\mathcal{L}(\boldsymbol{c}, \hat{\boldsymbol{c}}, \boldsymbol{o}, \hat{\boldsymbol{o}}) = \alpha_{\boldsymbol{c}} \mathcal{L}(\boldsymbol{c}, \hat{\boldsymbol{c}}) + \alpha_{\boldsymbol{o}} \mathcal{L}(\boldsymbol{o}, \hat{\boldsymbol{o}}) \tag{6.3}$$

Here, $\boldsymbol{c}$ and $\hat{\boldsymbol{c}}$ are the collision index and probability, respectively; and $\boldsymbol{o}$ and $\hat{\boldsymbol{o}}$ are the raw depth image input and the reconstructed filtered depth image output, respectively. The loss function $\mathcal{L}(\boldsymbol{o}, \hat{\boldsymbol{o}})$ is the single sample Monte Carlo estimate of the evidence lower bound, established in equation 6.2, which gives a scalar loss value. The other loss function, $\mathcal{L}(\boldsymbol{c}, \hat{\boldsymbol{c}})$ is the *binary cross entropy* loss function which returns a scalar value for the collision predictions. The constants α_c and α_o are chosen such that the magnitude of the two losses are roughly the same. Choosing $\alpha_c = 1.0$ and $\alpha_o = 0.001$ gives the two losses roughly the same magnitude. This is due to the binary loss function used for collision, iterating over a couple hundred collision predictions. In contrast, the *ELBO* loss used for the VAE compares every pixel across the input and reconstructed images, giving it a far greater loss value.

The ORACLE-VAE model is trained with the Adam optimizer, using a learning rate of $1 \cdot 10^{-4}$ and decay of $1 \cdot 10^{-5}$. Both batch normalization and Monte Carlo dropout are utilized on the encoder and decoder during training. At the end of every epoch, the model logs the respective losses, accuracy, precision, and recall for the training and validation data. Naturally, accuracy, precision, and recall only apply to the collision probabilities. The training continues until the metrics have more or less stabilized, equating to roughly 500 epochs. The training data set contains shy of 1.4 million data points, and the validation data set contains roughly 350000 data points, totaling over 1.7 million.

6.5 Evaluation

Evaluation of the developed model will be three-fold. First, the results from SGM will be presented, highlighting the resulting depth images created. Then, the results from the VAE will be presented, focusing on the observed training history and the reconstructed images. Lastly, ORACLE-VAE will be presented from a pure machine learning perspective, and with regard to motion planning and navigation in cluttered environments. The data in the validation set is not fitted to the VAE and ORACLE-VAE models and will thus yield somewhat reasonable indications towards the general performance of the models and whether they generalize to unseen data.

The navigation policy will be evaluated in two separate environments, varying in difficulty. The environments will both be corridors with obstacles, making for cluttered environments. The corridors are 100 meters long and 13 meters wide. Both corridors have walls along the

longer sides of the environments, covered in the same stone-like texture previously seen in 6.8. Similar to the environment used to generate data, both corridors will have the floor covered in a toy-like texture. The first environment contains 6 arches (6.6d), 7 pyramids (6.6a), and 13 pillars (6.6c), a total of 26 objects. We will refer to this environment as the "easy" one. The easy environment is visualized in figure 6.10. The other corridor will be denser, including more obstacles, and henceforth be referred to as the "hard" environment. The hard environment, visualized in figure 6.11, contains 75 pillars, all placed randomly within the corridor. The increased density of obstacles in the hard environment is expected to result in a decline in performance compared to the easy environment.

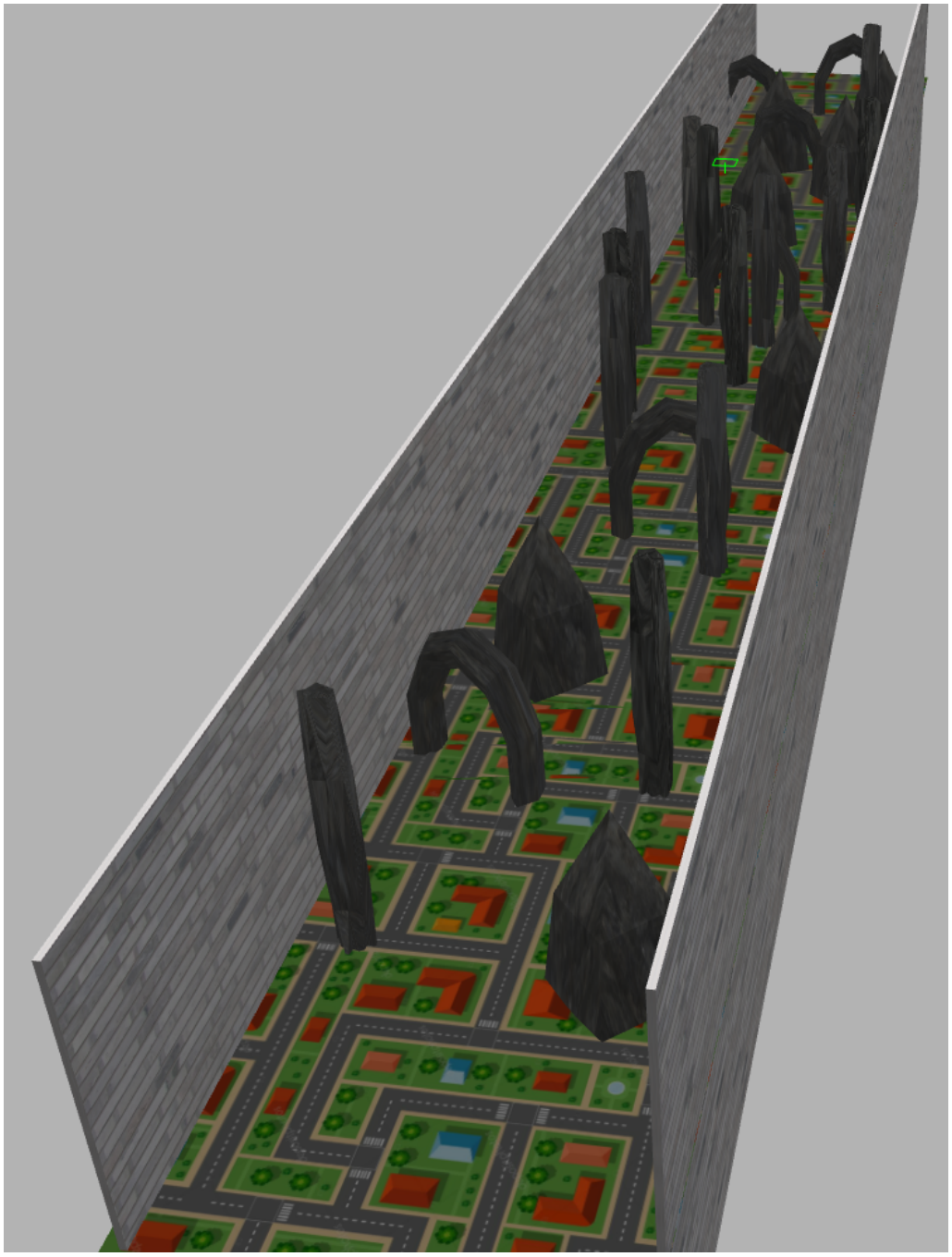


Figure 6.10: The "easy" evaluation environment.

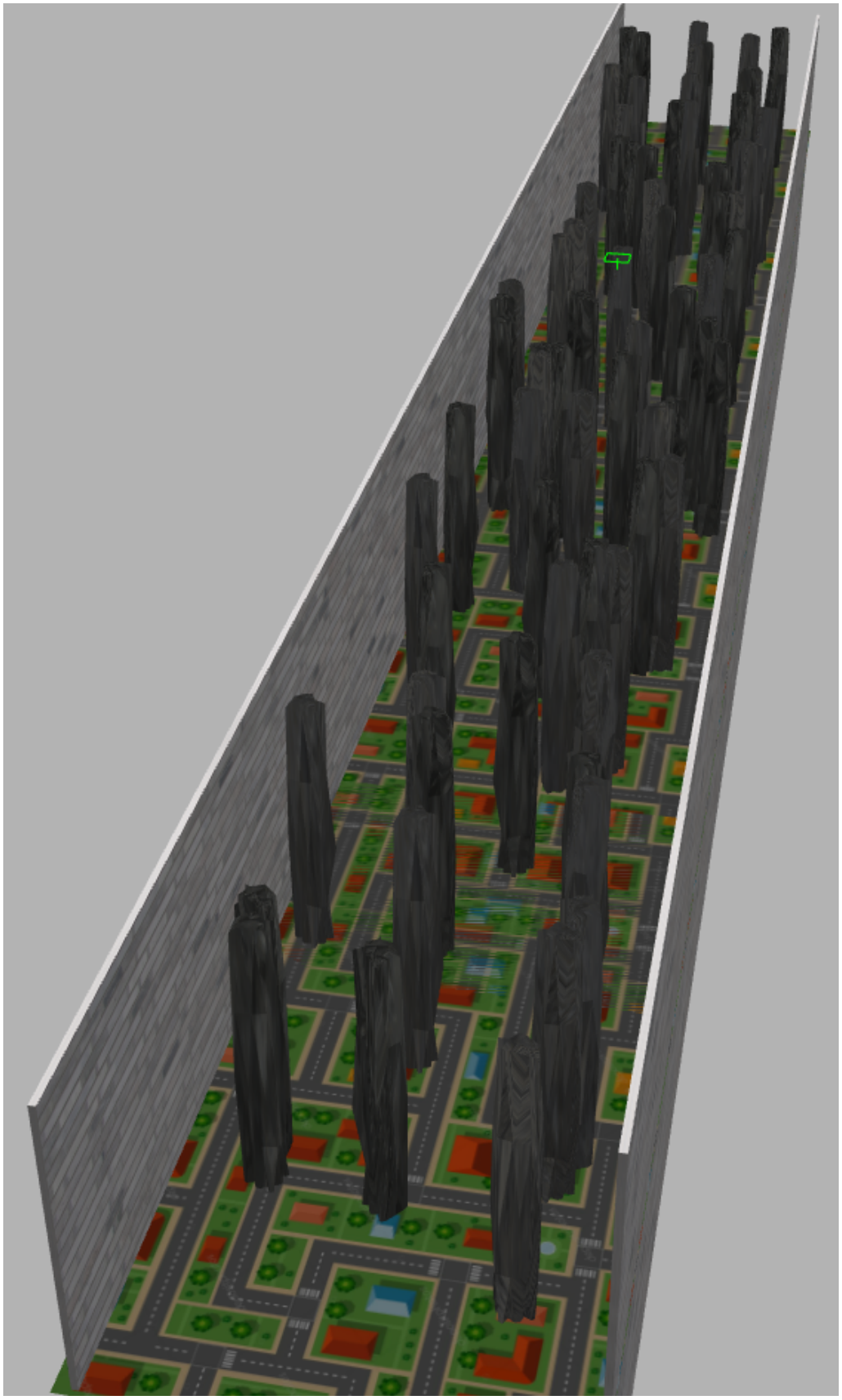


Figure 6.11: The "hard" evaluation environment.

The goal of the evaluation studies is for the RMF to traverse the corridors in a safe and timely manner. The robot will be spawned on one side of the environment, while the goal position is on the other side of the corridor. When spawned, the robot will face in the opposite direction of the goal with a random heading angle. It will encourage the drone to take different paths to reach its goal. The RMF will be evaluated based on the number of collisions and distance traveled, based on attempting to traverse the environments 20 times each.

During the flight, the robot is inclined to perform actions that yield a safe, collision-free path. The robot states and depth images are provided in real-time. Simultaneously, action sequences in the FOV are drawn from the motion primitives library. There are a total of 256 different sequences drawn, each 15 steps long, combining 32 different heading angles ψ, eight different velocities in z-direction, v_z, and a constant forward velocity, v_x. The action sequences are passed through the network along with the robot state and depth image to output collision probabilities of the action sequences. Then, the final collision cost for each sequence is calculated as the weighted sum of the collision probabilities at each step. The sooner the collision event is predicted to happen, the more it will contribute to the final collision cost[11]:

$$\hat{c}^{col} = \sum_{i=1}^{H} \hat{c}_{t+i}^{col} e^{-\lambda(i-1)} \tag{6.4}$$

The lowest weighted collision probability $\hat{c}_{min}^{col}$ is chosen as a baseline for evaluation of the other probabilities. All other weighted collision probabilities that are of a greater value than $\hat{c}_{min}^{col} + c_{th}$, where c_{th} is a positive threshold, are discarded. The remaining action sequences are checked for deviation against the goal direction ψ_t^g given by the global planner.

$$c^{goal} = |\text{wrap}(\psi^{ref} + \psi_t - \psi_t^g)| \tag{6.5}$$

Here, ψ^{ref} is the relative heading angle of the respective action sequences, ψ_t is the heading of the robot, and wrap$(\cdot)$ is the function that wraps the angle within the range $[-\pi, \pi]$. Out of the remaining action sequences, the one that minimizes c^{goal} is chosen, and the first step of the sequence is executed. Evaluating action sequences before the first step is executed is repeated in a receding horizon fashion until the goal position is reached or collision. During the flight, the robot is restricted to planar movement in the xy-plane. Its forward velocity is $0.5\frac{m}{s}$. The navigation planner is summarized in algorithm 4

Algorithm 4 ORACLE Navigation Planner [11]

while Robot has not reached goal position or not collided **do**
  Get robot's current state s_t and depth image o_t
  Get action sequences $\{a_{t:t+H}\}_{n=1}^{N}$ from motion primitives library
  for Each action sequence $\{a\}_n$, with $n \in [1, N]$ **do**
    for Each step in the action sequence, $i \in [t, t+H]$ **do**
      Calculate the collision probability, $c_{i+1}^{col} = ORACLE - VAE(s_t, o_t, a_i)$
    end for
    Calculate the weighted collision cost for action sequence $\{a\}_n$,
    $\hat{c}^{col} = \sum_{i=1}^{H} \hat{c}_{i+1}^{col} e^{-\lambda(i-1)}$
  end for
  $\hat{c}_{min}^{col} = \arg\min \hat{c}^{col}$
  Discard all action sequences with a weighted collision cost lower than $\hat{c}_{min}^{col} + c_{th}$
  Check the remaining action sequences again the deviation from the goal direction,
  $c^{goal} = |\text{wrap}(\psi^{ref} + \psi_t - \psi_t^g)|$
  Choose the action sequence, $a_{t:t+H}^*$, that minimizes c^{goal}
  Execute the first step of the action sequence a_t^*
end while

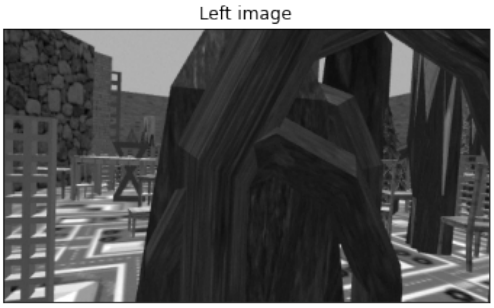
Chapter 7

Results

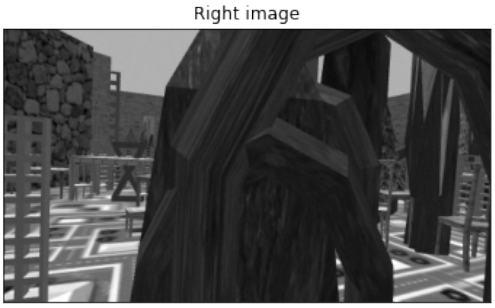
In this chapter, we show the results obtained. This chapter will be split into three separate sections. One for depth maps from disparity through semi-global matching, one for the variational autoencoder with reconstructed images, and one for ORACLE-VAE with performance regarding machine learning metrics and performance for the evaluation studies. The results will be presented by including images for the relevant sections. For the stereo matching, each set of images will contain a raw depth image and filtered depth images, along with a corresponding stereo image used for matching. Similarly, when visualizing the results for the VAE, each set of images will contain three images: the raw and filtered depth images and the reconstructed image. All image sets will also contain a color bar illustrating the depth, in meters, in the corresponding images.

7.1 Depth From Disparity

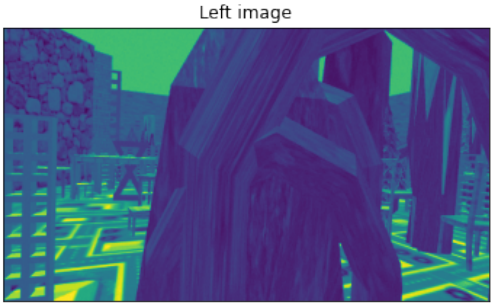
The images captured by the stereo camera are grayscale. Two images are captured simultaneously as the robot is mounted with a stereo camera. Typical images captured are visualized in figure 7.1. The same images are displayed twice, one in its original grayscale color and one subject to a change in color to *Matplotlib*'s standard color palette when displaying images. This and the next section will display Imminent images in Matplotlib's color palette. There are plenty of objects visible in the images. Firstly, the floor is covered in texture. Furthermore, there are multiple arches (figure 6.6d) and a pyramid (figure 6.6a) visible in the foreground in the center of the images. Additionally, there is a fence (figure 6.6i) visible on the left-hand side of the images in the foreground, with multiple objects further back in the images, including, amongst other things, a stone wall (figure 6.6j) and a cross fence (figure 6.6k). On the right-hand side of the image, we can see a pillar (figure 6.6c) and some chairs (figure 6.6g) with more objects further back.



(a) Left grayscale image



(b) Right grayscale image



(c) Left image with a colorful palette

(d) Right image with a colorful palette

Figure 7.1: Left and right images captured by the stereo camera in the original grayscale color palette and with *Matplotlib*'s standard color palette.

The images visualized in figure 7.1 are typical images from a stereo pair used to create disparity maps through semi-global matching before being turned into depth maps. In the resulting depth maps, the max depth is set to 10 meters as objects further away are less critical for immediate actions.

First off, from figure 7.2 we can observe how the raw and filtered depth images result from the corresponding stereo image. There are definite, separable levels of depth for the different objects in the depth images. The resulting levels of depth are more similar to how depth is captured through a depth camera. More specifically, in 7.2a, the depth between the central objects is distinguishable. Additionally, there is a visible depth difference between the image's left and right-hand sides. Similarly, in 7.2b, the same results in depth can be seen with the tabletop in the foreground and other obstacles further back in the scene, and in 7.2c between the table, the tree, and the background.

Notice how in the raw depth maps (center image in the image sets) in figure 7.2 there are visible darker patches along the edges of the objects across all three sets of images. These patches are invalid matches and are removed in the filtered depth maps using a simple Gaussian noise filter. The invalid matches are particularly visible around the arches in 7.2a, the tabletop in 7.2b, and the tree in 7.2c.

Furthermore, we can observe mismatches in regions towards the top of the images where the background is textureless. The monotonous green color in the stereo images marks the textureless background. In these regions, the objects in the foreground tend to blend in with the background, making them appear larger than their proper size in the depth images. This effect is most prominent at the top left of the pyramid in figure 7.2a, at the top of the pillar in figure 7.2b, and around the upper part of the tree in figure 7.2c. Consequently, it may infuse difficulties when training the deep collision predictor network, ORACLE-VAE, given that objects in the depth maps appear larger than the actual size. Intuitively, it is difficult to determine how this may affect the learned navigation policy. It may cause the learned policy to behave safer as the objects in the depth images appear more extensive. On the other hand, it could also lead to difficulties learning from imagery as some areas seem closer and impossible to pass through, while they may be traversable.

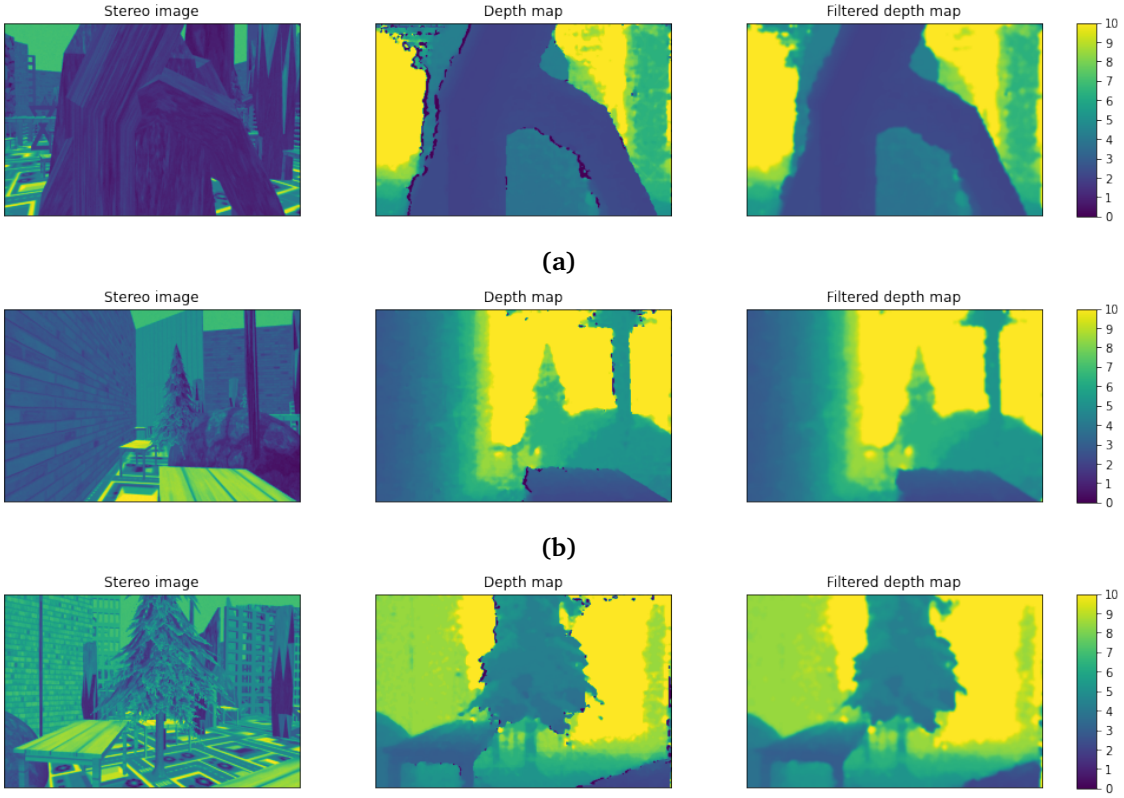
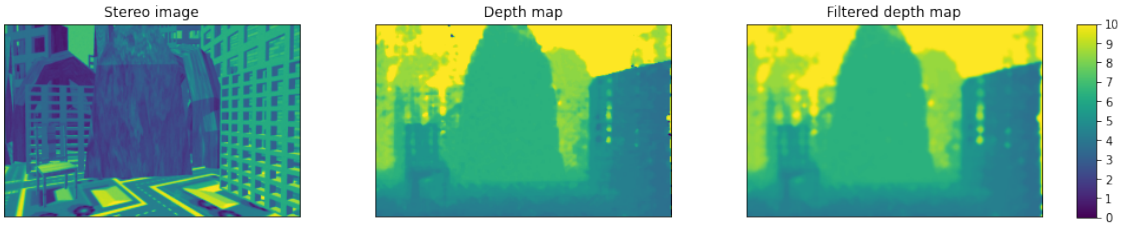


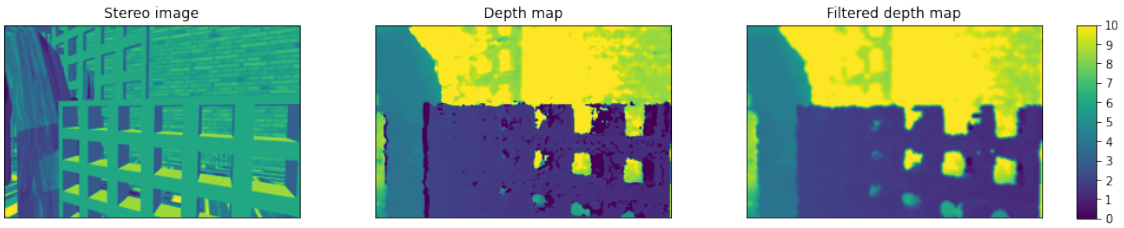
Figure 7.2: 3 random scenes

From the images in figure 7.2 we have observed how the stereo matching shows transparent, distinguishable layers of depth. Nevertheless, invalid matches appear around objects, and the texture-less background is a cause of mismatches, resulting in objects appearing larger than the actual size. Interestingly, the method struggles more with one type of structure: the fences.

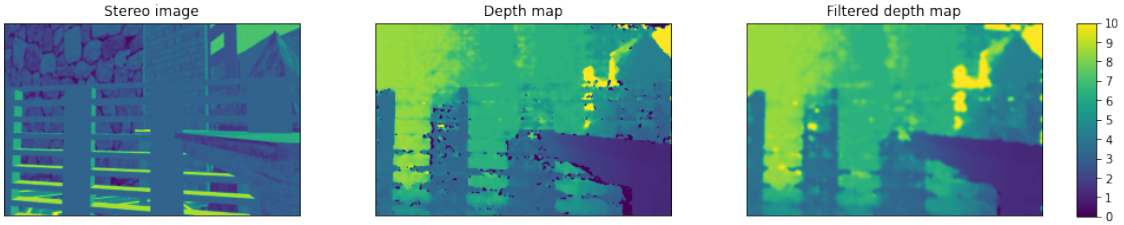
Depth maps where fences are present are visualized in figure 7.3. In 7.3a there are two fences, one to the right and one further back on the left-hand side of the image. From the right fence, we observe that the outlined depth of the fence is distinguishable from the surroundings. However, with the fence on the left, the lines defining the fence are more blurry in the depth maps. Evidently, from 7.3a, it suggests that the matching algorithm has an easier time matching the larger vertical components of the fence and is struggling more with the more minor horizontal elements. The same effect can be seen in 7.3c, where there is a fence across the captured images. Here, some horizontal components towards the top of the fence have entirely vanished, giving a false representation of the object in the resulting depth maps. Similarly, the fence's vertical lines get increasingly blurry, moving from left to right in the image. However, the result in 7.3b tells a different tale. The fence's components are more uniform in size, resulting in a clearer and more defined object in the depth maps. Nevertheless, the fence appears to be more prone to noise and mismatches compared to the section of the arch to the left of the fence in the raw depth image. Interestingly, the top of the fence is not in the depth maps. It looks like the top has completely vanished into the textured background. Based on these results, the resulting depth map around fences seems greatly influenced by the background texture. The matching of fences, or rather, the lack of, may lead to difficulties in the training of ORACLE as it may suggest an area of the environment to be traversable based on the depth image when there, in fact, is a fence present. This may give the model false safety towards where the robot can move without colliding.



(a) Example 4



(b) Example 6



(c) Example 7

Figure 7.3: 3 scenes where fences are present

The results from stereo matching highlight some significant difficulties when creating depth maps. The most obvious shortcoming is the matching of fences which tend to be blurry and, to some extent, blend with the textured background. Additionally, objects appear larger in the depth images in areas where the background is textureless. Apart from these difficulties, the matching algorithm creates depth maps where objects are separable through depth, and the shape of the objects is primarily precise. The noise that comes with the matching results is more akin to real depth images rather than perfect images that are more common in simulations. However, whether the resulting depth maps are well-suited for the subsequent learning task remains to be seen.

7.2 VAE on Depth Images From Stereo Matching

The depth maps presented in section 7.1 are the images used when training the variational autoencoder. The data set contains roughly 230000 images for training and 58000 images for validation. An epoch of training is equivalent to one iteration through the data, where the network weights are updated on the training data and remain unchanged when iterating

through the validation data. The training process proved to be relatively straightforward, only requiring minor changes to the optimization process in order to land on a suitable model. The training history is visualized in figure 7.4 with the network trained over 100 epochs. Originally, the evidence lower bound is subject to maximization as it is a negative scalar. However, as mentioned previously, the network is trained by minimizing the negative evidence lower bound for numerical stability. From the visualized history, we observe that the training and validation loss declines rapidly at the beginning of training before plateauing towards the end of the training process. Although the losses are of similar magnitude, there is still a noticeable gap between the values, where the training loss stagnates at around 52500, while the validation loss remains at around 53200.

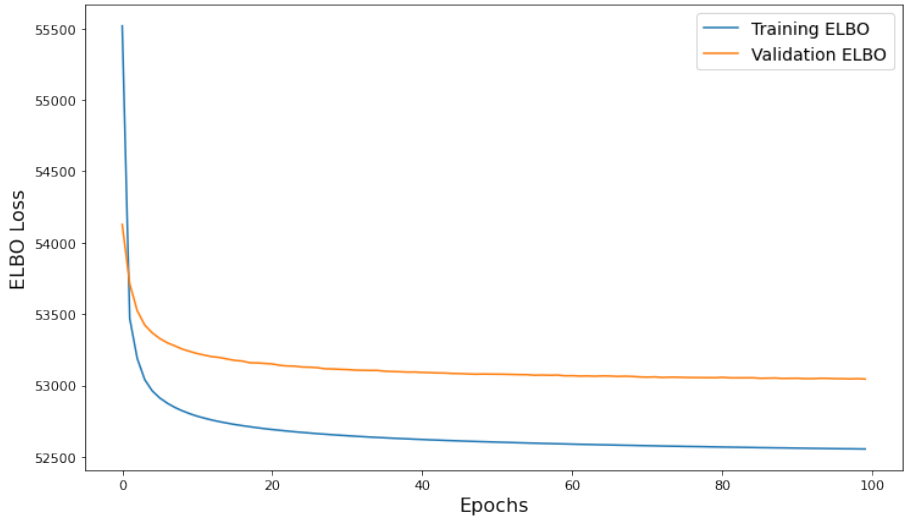


Figure 7.4: Training and validation loss for the VAE on depth images

More interestingly, with the VAE outputting reconstructed images, are the reconstructions and whether the reproduced images are comparable to the input images. Here, we will comment on the reconstructed images from the generated depth images across five different types of scenes where different objects are present. First, we will see the results for relatively simple scenes where only a couple of objects are present within the image. Then, two more different scenes will be presented, where the images increase in the density of objects. Lastly, we will examine two scenes where fences, which proved difficult to match through stereo imagery, are present in images. In order to verify the results, the VAE was also trained on perfect depth images generated from a depth camera. These results are visualized in Appendix B but will not be mentioned further.

First, the images in figure 7.5 illustrated three relatively simple scenes where only a couple of objects are present. In general, the main structures have been reconstructed quite accurately. More specifically, from 7.5a, the wall, floor, and background have been reconstructed, and there is little difference between the filtered and reconstructed images. For the table, the re-

construction is slightly more contrasting in the reconstruction. Interestingly, the tabletop, the horizontal component of the table, seems to more abruptly disappear into the wall, making the resulting depth less distinguishable from the original filtered depth map. The table leg is also noticeably different across the two images, with the reconstruction having smooth edges, thus filtering out mismatches and invalid matches. The same effect can be seen in 7.5b where the object in the background appears more even in the reconstructed image, similar to a filtering effect. The result in 7.5c are similar. Here, some details of the arch in the image's background have partially vanished in the reconstruction. Overall, the depth is well preserved across all three scenes.

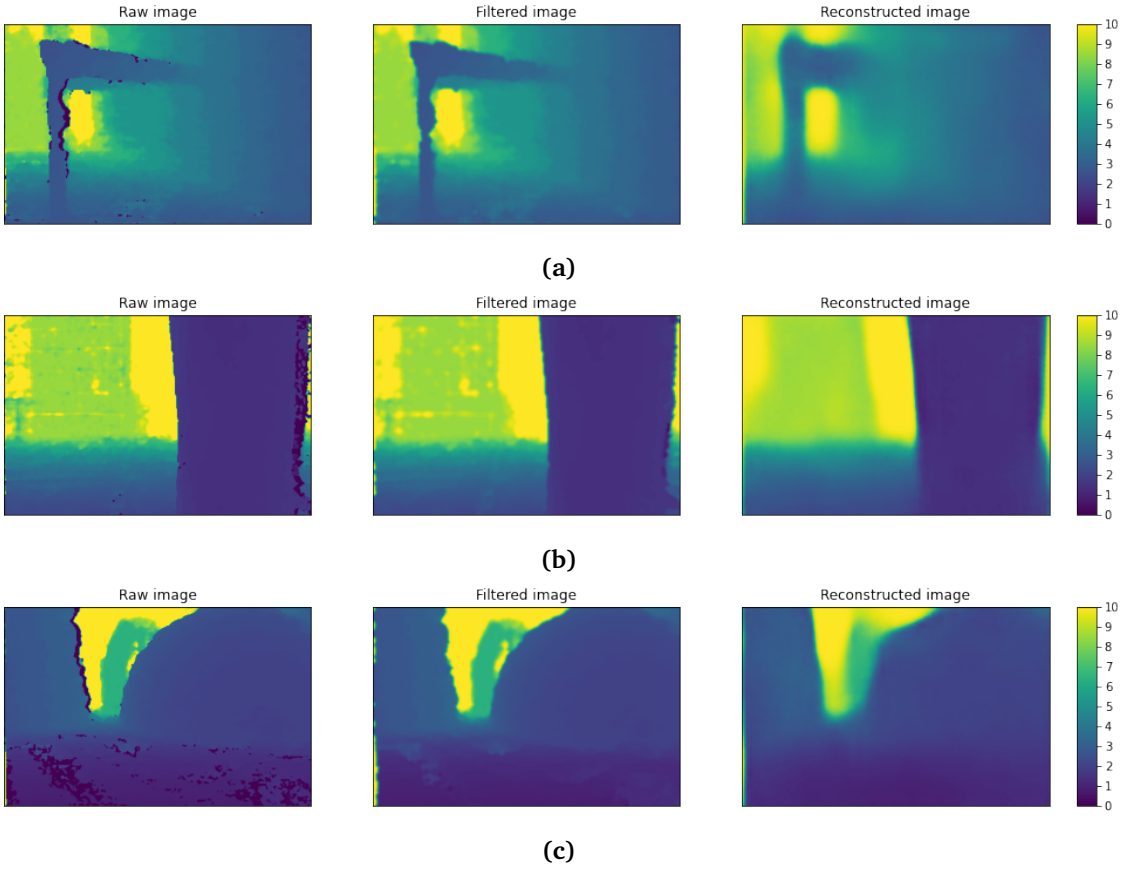


Figure 7.5: 3 sparsely cluttered scenes

In a more cluttered environment, three different scenes are captured in figure 7.6. First, in 7.6a, there are multiple objects in the scene, including a tree, a table, and objects along the sides. This makes the scene appear like a narrow, cluttered corridor. In 7.6b, a tree is to the right side of the images with another object in front of it. There is also a pyramid at roughly the same depth as the tree. Furthermore, there is a wall in the background. Lastly, in 7.6c, there is an arch in the foreground with multiple objects at a greater depth, partially covered by the arch. Across the reconstructed images, we observe, similarly to the more simple scenes, that

the reconstruction has a filtering effect. Overall, the depth is largely preserved, but geometric features seem to vanish somewhat. This is especially noticeable in 7.6a where the details of the tree have disappeared, and the table is unrecognizable in the reconstructed image. Also, the depth along the sides is smoother and more gradual, in the reconstructed image, compared to the filtered image. The same smoothing effect is visible in 7.6b where the transition across objects is smoother in the reconstruction compared to the more abrupt changes in depth in the raw and filtered depth images. Here, in 7.6b, it has resulted in the tree and the foreground object fused into what looks like a completely different object. The same effect is visible in 7.6c where the transition between the arch and the background objects is more gradual in the reconstruction and more sudden in the raw and filtered depth images. Overall, by comparing figure 7.6 to 7.5, it seems like the more cluttered a scene is, the more likely it is to lose details in the reconstructed image.

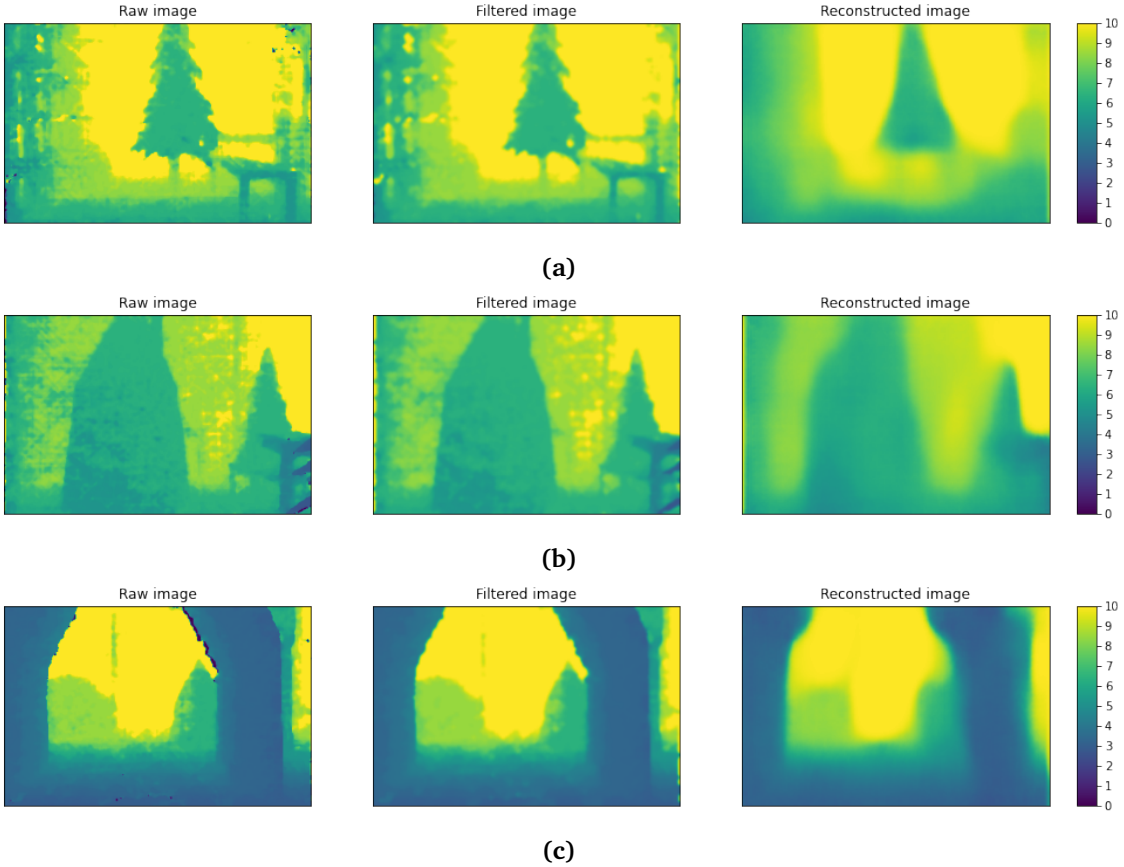


Figure 7.6: 3 scenes that are more cluttered

Following the same pattern, if the scene is even more complex, more details would be lost during reconstruction. Judging from figure 7.7, we can observe that to be the case. From 7.7a, the reconstruction shows that the depth is somewhat well preserved with imperfection in reconstructing details. Perhaps most interesting is the tree reconstruction, which resembles

a shape similar to a traffic cone. Likewise to the less cluttered environments, the reconstruction has a filtering effect, resulting in a smoother reconstruction compared to the raw and filtered depth maps. In 7.7b, depth is mostly intact similar to what can be observed in 7.7a. However, more details are lost, for example, objects in the background and the chair object to the right-hand side of the image. In the even more complex scene in 7.7c, the reconstructed depth is more lacking in preserving depth. The trees to the center-right are less defined than in previous images. In particular, the right-most branch has more or less vanished completely in the reconstructed image, giving a false perception of the depth. Likewise, the object further back in the center-left has mostly vanished where only simple lines are reconstructed, giving the impression of a different object. Altogether, in 7.7c, the objects are less defined. Overall, it appears to be a linear relationship between the number of obstacles in the depth images and the VAE's capability to preserve depth.

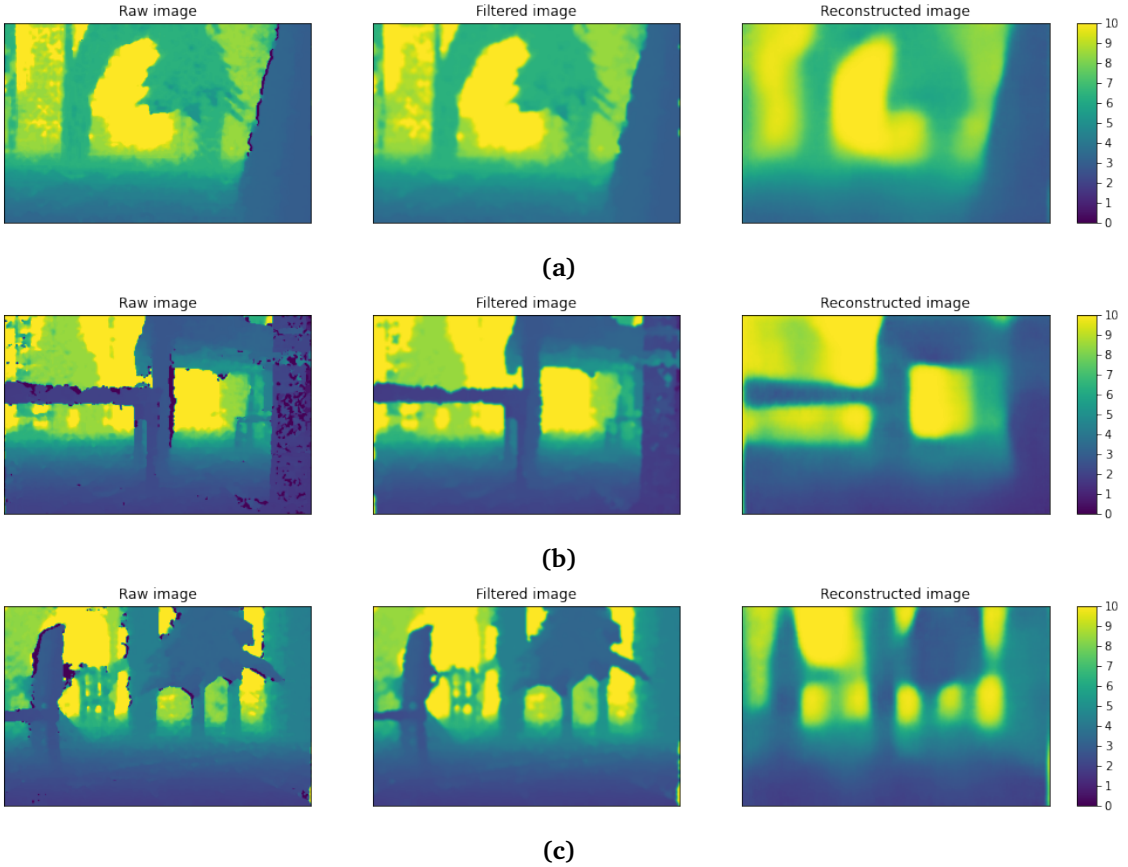
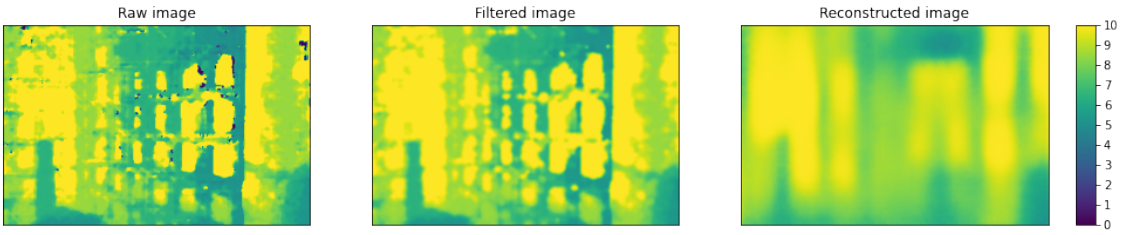


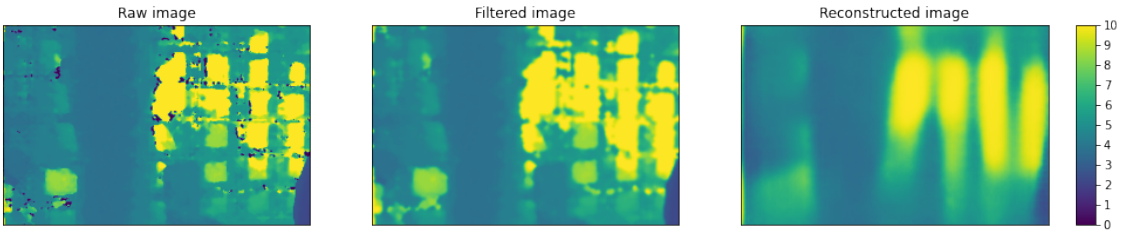
Figure 7.7: 3 very cluttered scenes

Previously, we have seen how the stereo matching algorithm struggles when there is a lack of texture. Notably, the fences proved to be difficult to match accurately. This may pose problems when reconstructing images containing fences as well, as can be observed in figure 7.8. From the two separate reconstructed images, the characteristic features of the fences have

disappeared. In 7.8a and 7.8b the reconstruction looks more like pillars than fences due to the missing reconstruction of the horizontal elements. Also, similarly to previous results, the model has a filtering effect on the reconstructions.



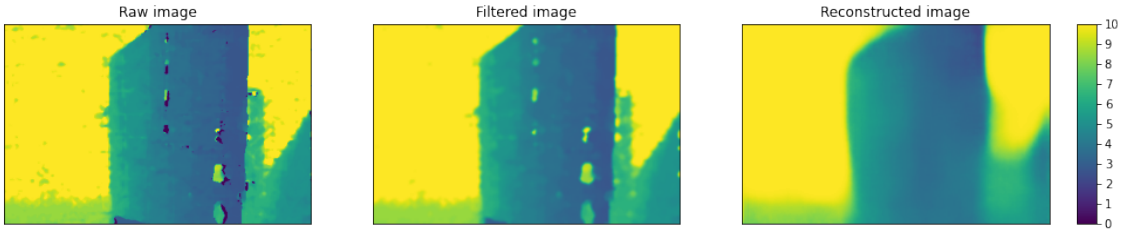
(a)



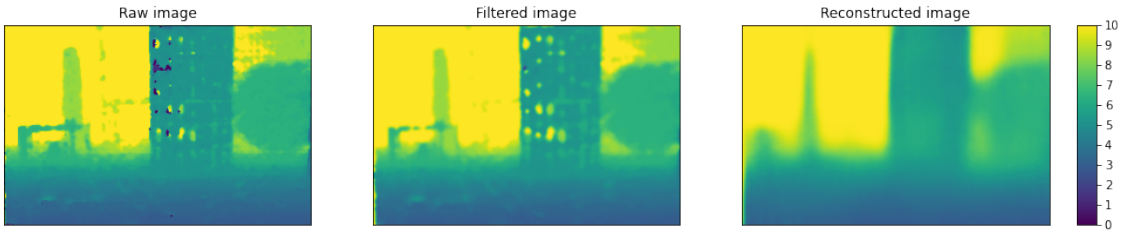
(b)

Figure 7.8: 2 scenes where fences are present closer to the robot

Lastly, we will look at scenes where fences are at a greater depth. Fortunately, when the fences are further back in the scene, the reconstruction is better (figure 7.9). In 7.9a and 7.9b, the reconstructions have, once again, a filtering effect, making the fences appear more like walls than fences.



(a)



(b)

Figure 7.9: 2 scenes where fences are present further away from the robot

From the presented results, it is evident that less complex scenes will likely result in better reconstructions. From figure 7.5, where the depth images are less cluttered, the depth is preserved, and most features are distinguishable in the reconstruction. However, when the environments in the images are more complex, we have seen how several details are left out of the reconstruction, like in figure 7.6. Nonetheless, depth is primarily intact. Differently, for even more cluttered environments, like in figure 7.7, reconstruction gets more blurry, sometimes failing to reconstruct the depth correctly.

The fences proved to cause difficulties for semi-global matching. Similarly, they proved to be challenging for the variational autoencoder. As a result, the resulting reconstructions displayed shapes more akin to pillars rather than fences, as is visualized in figure 7.8. However, as seen in figure 7.9, when the fences are at greater depth, the matching tends to be better, resulting in more accurate reconstructions.

Overall, the model developed is capable of accurately reconstructing images, mostly upholding depth. However, the two difficulties addressed; namely, complex scenes and the fences up close, are troubling and may present difficulties when training the deep collision predictor network.

7.3 ORACLE-VAE

7.3.1 From a Machine Learning Perspective

Training the ORACLE-VAE model turned out to be more difficult than just the stand-alone VAE. Different combinations of weights α_c and α_o and different dropout rates p for Monte Carlo dropout were applied to reach the best model. Additionally, different learning procedures were explored, such as pre-training different components before fine-tuning some of the layers. In the end, the model that provided the best results was trained end-to-end, without pre-training, and with $\alpha_c = 1.0$, $\alpha_o = 0.001$, $p = 0.5$.

The learning history is visualized in the following illustrations. First, the ELBO loss for reconstructed images, visualized in figure 7.10 proved to be quite similar to the stand-alone VAE. The training and validation ELBO loss decreases rapidly at the beginning of the learning process. Before 100 epochs, the curves start to plateau. After 100 epochs, the validation ELBO remains more or less constant, whereas the training ELBO sees a slight decrease. The loss values are significantly larger, here at around 56000, compared to in the stand-alone VAE (figure 7.4), where the values are in proximity of 53000. Still, the reconstructed images are similar to the previously presented reconstructions and will not be visualized here.

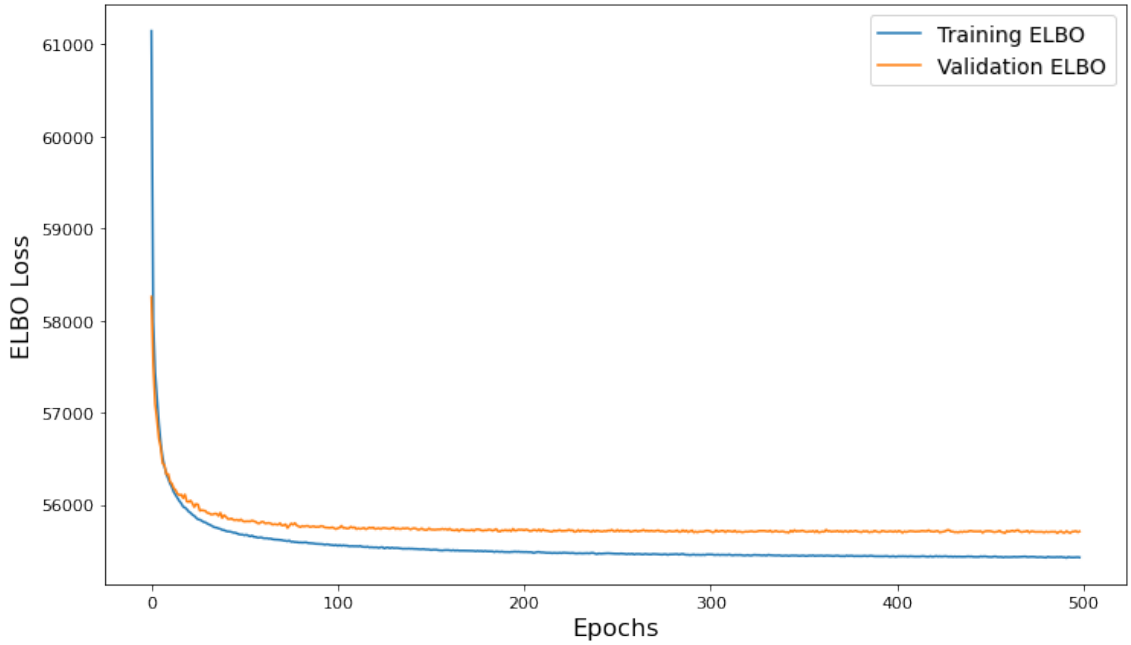


Figure 7.10: The ELBO loss on reconstructed images

Next, the binary cross-entropy loss on the collision probabilities is visualized in figure 7.11. The history curves here see different trajectories compared to the reconstruction loss. The validation loss decreases rapidly before slowly increasing after around 50 epochs. After around

400 epochs, it looks like the validation loss plateaus. On the other hand, the training loss continues to decrease further and is still slowly decreasing towards the end of the learning process. Still, the rate of change in the training loss decreases, indicating that the loss will eventually plateau. Interestingly, the two losses have a large cap in values. Towards the end of the learning process, the training loss reaches values approaching 20, whereas the validation loss reaches a value of around 50.

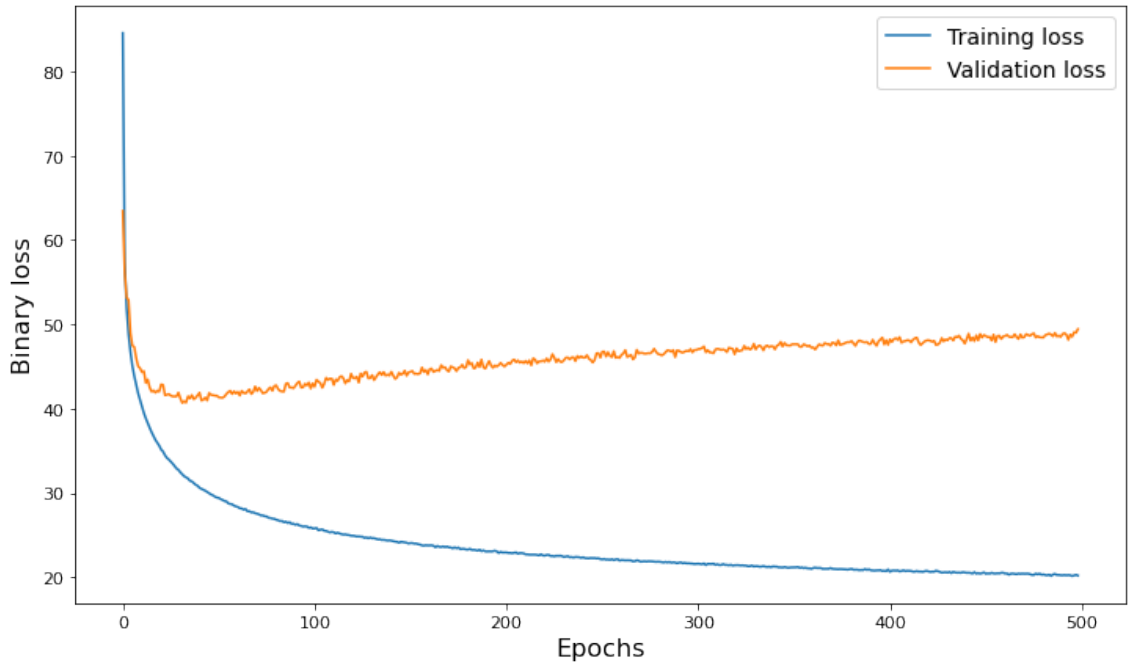


Figure 7.11: The binary loss on the collision probabilities

The accuracy, illustrated in figure 7.12, also see a significant gap in values between the training and validation. At the end of the training, the training accuracy is above 98% while the validation accuracy is around 96.5%. Early on, both training and validation accuracy see a rapid increase. While the training accuracy continues to increase, the validation accuracy plateaus at around 50 epochs. Curiously, this is around the same epoch where the validation loss reached its minimum value before it started to increase.

The history for precision (figure 7.13) is different from the other histories we have looked at. While the training precision increases, the validation precision decreases immediately. Validation precision starts at a higher value than the training precision, peaking as high as 98.5%. The rate at which validation precision decreases after several epochs is pretty significant. Eventually, the value is slightly below 96.5%. It is worth noting that the validation precision variance decreases with the number of epochs. This can be observed as the sporadic changes from epoch to epoch get less significant further down the line of training. On the contrary, the training precision keeps increasing, eventually plateauing at around 98.5%. The cap between training and precision is in the same range observed with accuracy.

Recall, visualized in figure 7.14, sees learning curves similar to accuracy. Both the training and validation recall increases rapidly in the beginning. The validation recall remains roughly the same after around 100 epochs, perhaps with a slight increase. Similar to the validation precision, the variance of the validation recall decreases with the number of epochs. The training precision continues to increase beyond 100 epochs, similar to the previous metrics. The value gap between validation and training is also similar to what was observed earlier, with the gap being around 2%.

Lastly, F1-score is the harmonic mean of precision and recall and can be observed in figure 7.15. Hence, it is expected that the learning curves see somewhat similar characteristics to precision and recall. Again, training and validation for F1-score increase rapidly at the beginning of training. The training F1-score continues to increase before arriving at a more constant value toward the end of training. On the other hand, the validation F1-score stops increasing after roughly 50 epochs, mainly remaining the same for the remaining epochs, perhaps slightly decreasing. Interestingly, the validation F1-score stays relatively constant, considering the decrease in validation precision.

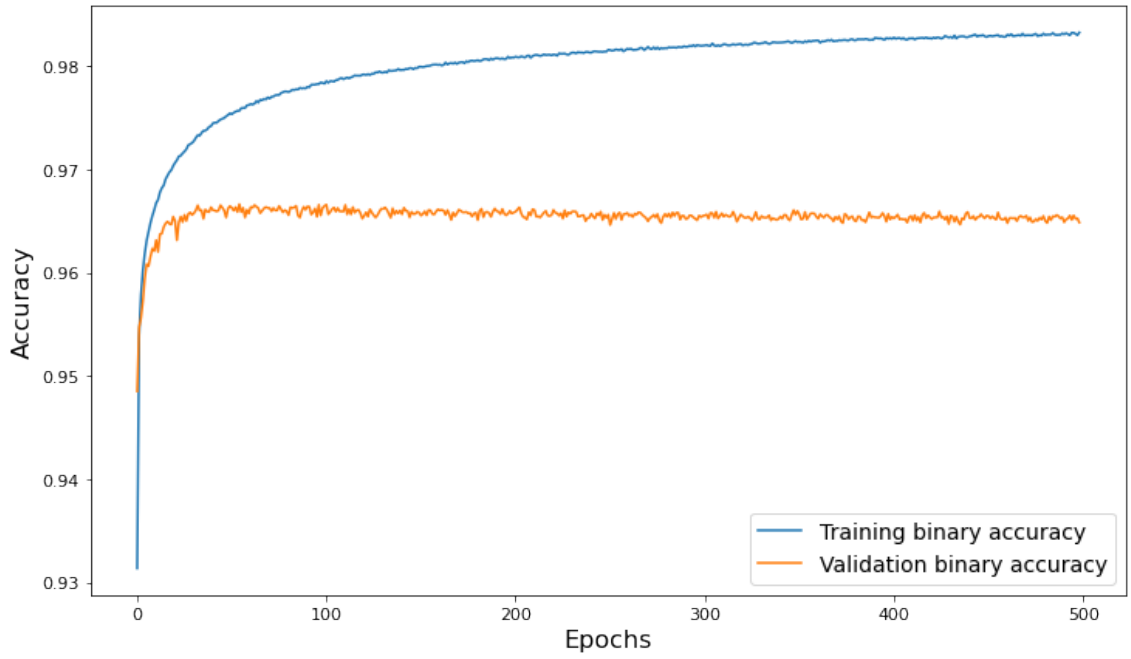


Figure 7.12: The accuracy of predicted collisions

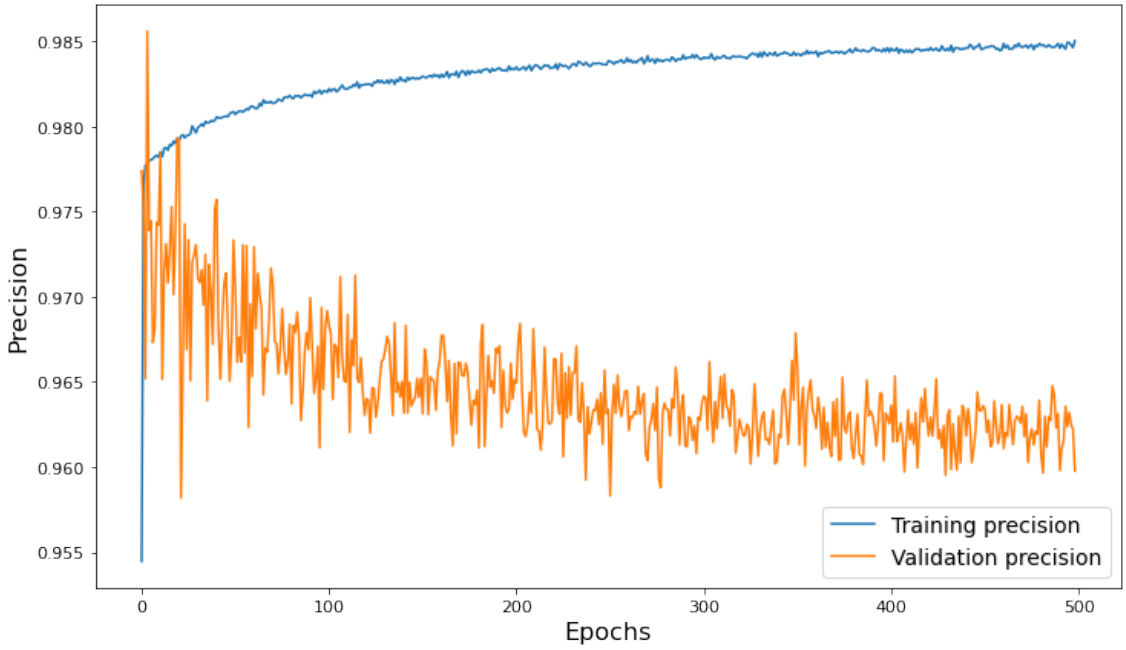


Figure 7.13: The precision of predicted collisions

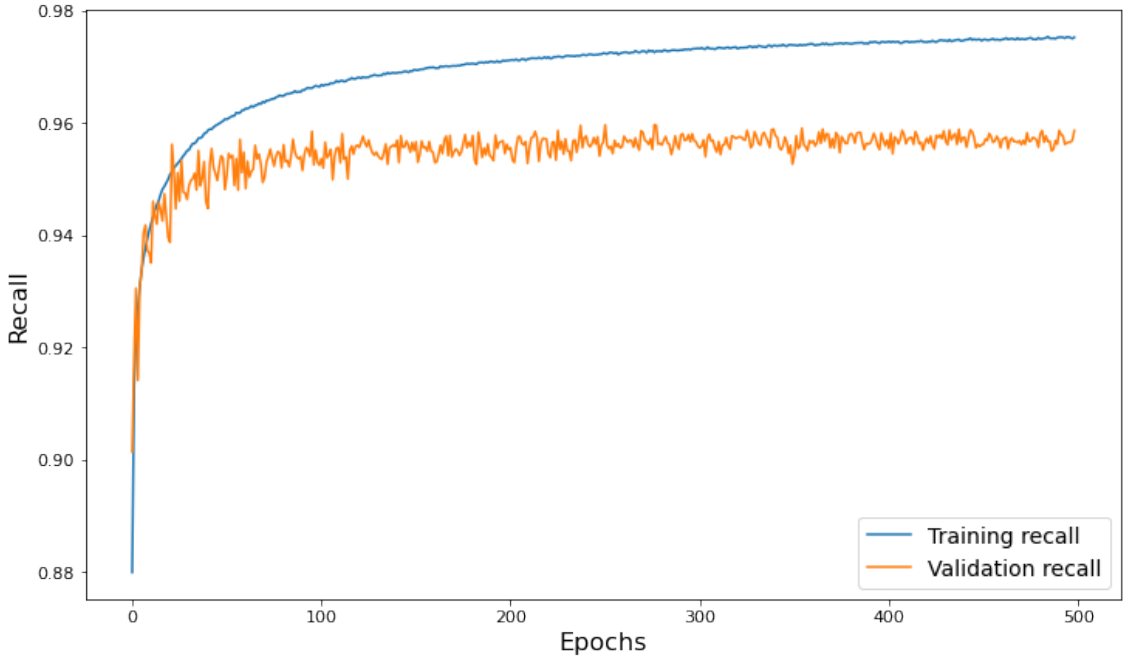


Figure 7.14: The recall of predicted collisions

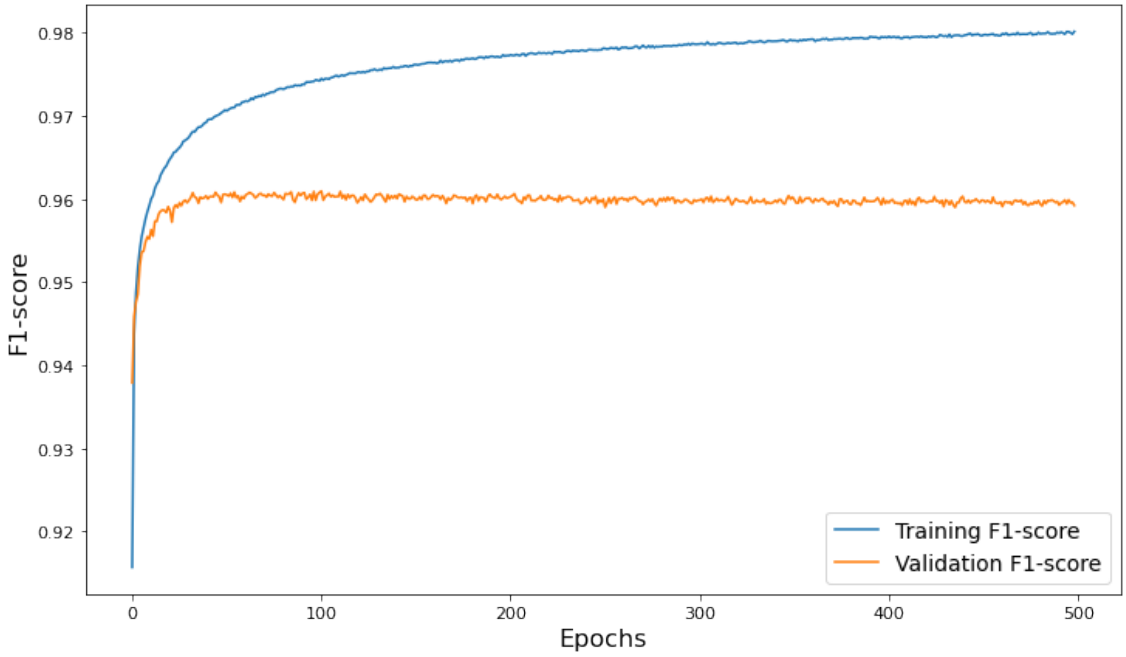


Figure 7.15: The F1-score of predicted collisions

A pattern exists for the supervised learning metrics and the binary loss on collision probabilities. Apart from validation precision which decreases, the validation accuracy, recall, and F1-score do not further increase beyond roughly 50 epochs after an initial rapid increase. Coincidentally, this is around the same time as when the validation loss starts to increase. From a pure machine learning perspective, having values for the various metrics above 95% generally indicates a well-behaved model. Whether this has translated into a safe navigation policy will soon be explored.

7.3.2 Navigating in Cluttered Environments

The RMF drone was tasked with traversing two corridors of 100×13 meters of different difficulty. During the flight, the position of the robot and the traversed distance are recorded along with a collision index. The performance of the planner in the respective environments are summarized in table 7.1.

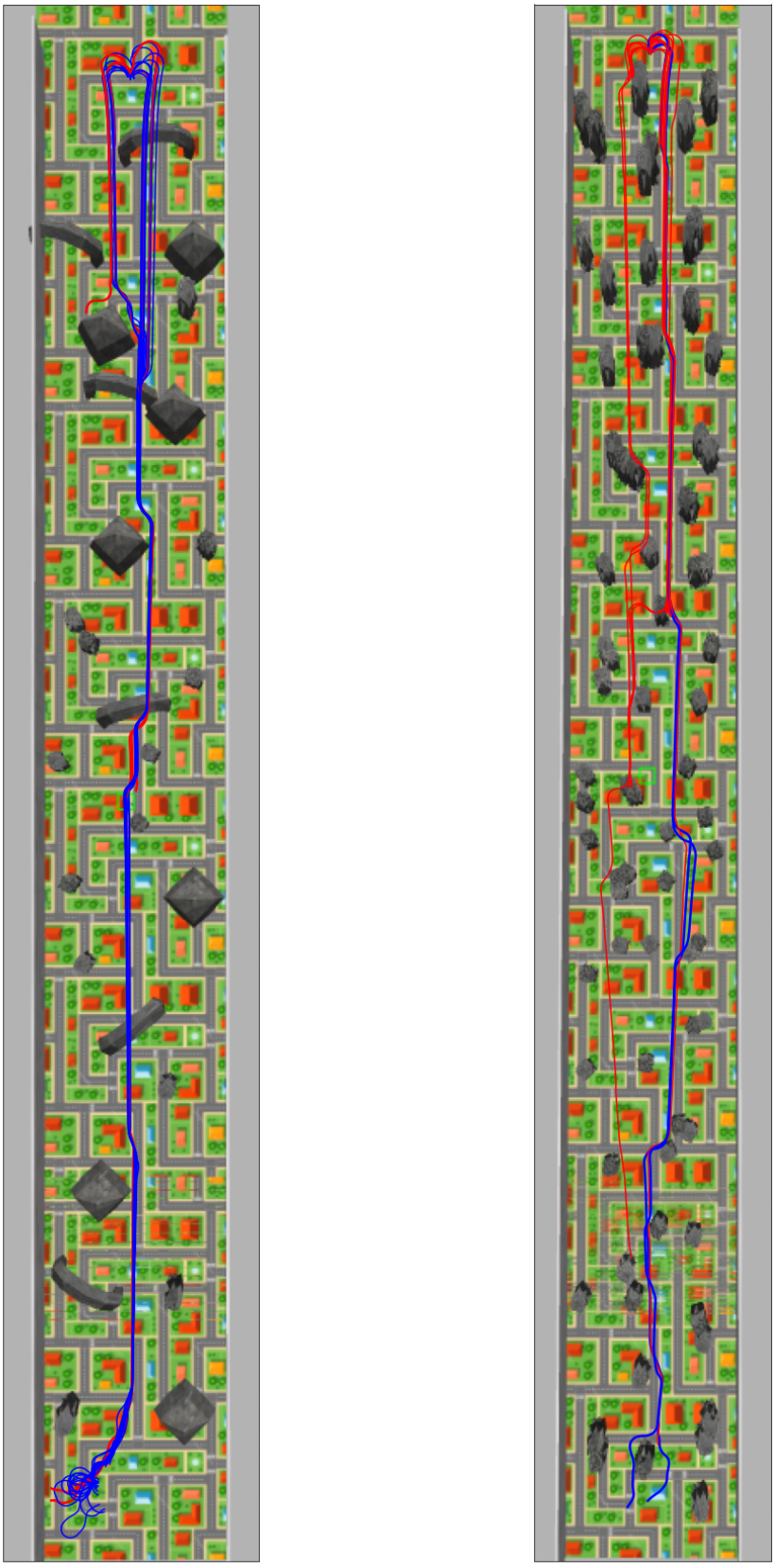
In the easy environment, the RMF reached the goal position 16 times on 20 runs, equating to a success rate of 80%. All runs are visualized in the environment in figure 7.16a where the robot starts movement at the top of the image. The blue trajectories indicate successful runs, whereas the red trajectories indicate runs where the robot collided. The average distance traveled across all 20 runs was 101.26 meters with a standard deviation of 21.86 meters. However, on the 16 successful runs, the average distance increased slightly to 108.27 meters with a standard deviation of 3.46 meters. From the visualization, it appears that the robot drifts to the right

when it has managed to avoid all the obstacles, explaining the average distance traveled on the successful runs, considering the corridor is 100 meters long. Interestingly, from the visualized trajectories, two of the runs that ended in collision managed to safely traverse the corridor, avoiding the obstacles along the way, before colliding in the wall in proximity to the goal position. For the remaining two collision runs, the robot collides earlier in the environment, one time roughly in the middle of the corridor and once closer to the spawn location. The more significant variance in the distance traveled when colliding is also noticeable in table 7.1, where the average distance traveled was 73.21 meters with a standard deviation of 36.87 meters.

The performance of the RMF sees a significant dip in performance in the hard environment, which is expected. In the hard environment, the robot reached the goal position on 2 of 20 runs, yielding a success rate of 10%. Consequently, the average distance traveled decreases, to 46.99 meters, with a standard deviation of 30.45 meters. Similarly, the average distance traveled when colliding has decreased to 40.67 meters, with a standard deviation of 25.10 meters. However, on the successful runs, the robot reaches the goal position faster, with the average distance traveled being 103.91 meters with 1.16 meters as standard deviation. The trajectories across all 20 runs in the hard environment are illustrated in figure 7.16b. Similar to visualizing the trajectories in the easy environment, blue trajectories indicate successful runs. In contrast, red indicates runs where the robot collided. On the two successful runs, the robot's trajectory appears similar, apart from a different path to avoid the last encountered obstacles, explaining the low standard deviation. The paths for collision are more sporadic, varying more in the traversed path taken and distance traveled before colliding.

Table 7.1: Performance for the RMF in the easy and hard environments

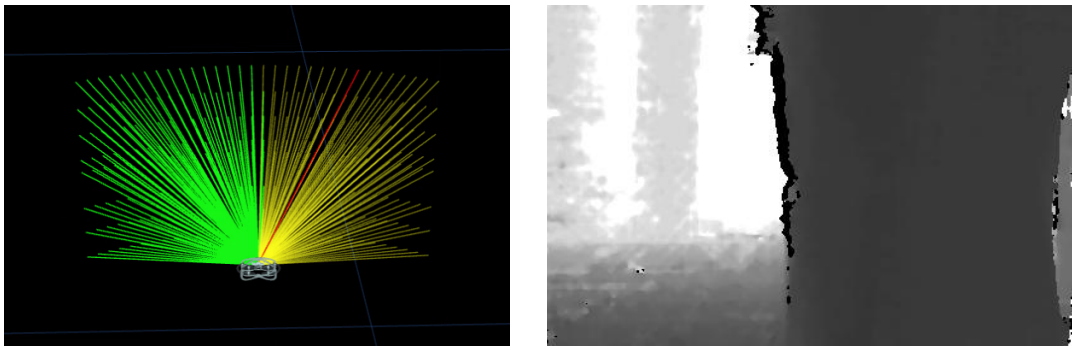
Performance		
	Easy	Hard
Success rate	16/20	2/20
Distance travelled [m]	101.26 ± 21.86	46.99 ± 30.45
Distance travelled on success [m]	108.27 ± 3.46	103.91 ± 1.16
Distance travelled on collision [m]	73.21 ± 36.87	40.67 ± 25.10



(a) Easy environment **(b) Hard** environment

Figure 7.16: Flight trajectories in the easy and hard environments. Blue paths indicate successful runs and red paths indicate runs where a collision occurred.

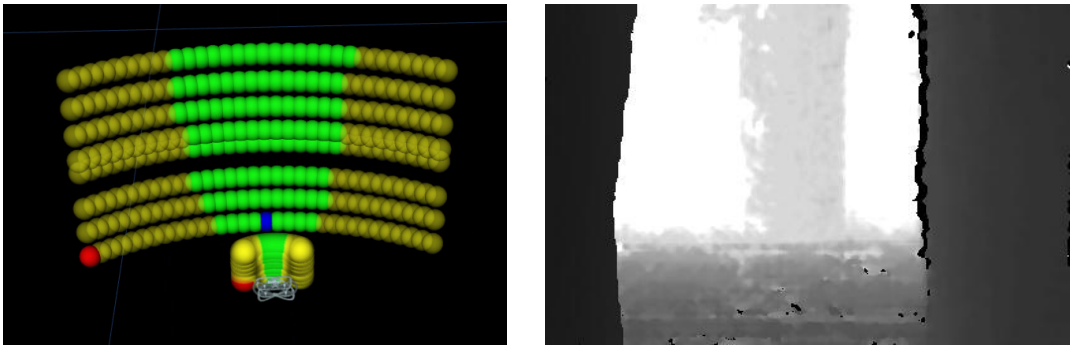
During the flight, the RMF receives information about its state and the current depth image created from stereo matching. This is used to reason about collision probabilities of the different action sequences created by the motion primitives library. The different sequences are classified as safe or unsafe, where the sequence least likely to cause a collision determines a baseline for evaluating the different sequences. Typical reasoning is visualized in figure 7.17. Here, the green sequences are safe action sequences, with lower collision probabilities than $c_{min}^{col} + c_{th}$. Furthermore, the yellow sequences are deemed unsafe, meaning the probabilities are higher than the upper threshold. For the purposes of debugging, the worst action sequence is visualized in red. The worst action sequence is the sequence that yields the highest probability of collision. Additionally, there is a blue sequence, which is the safe sequence that minimizes the deviation from the goal heading. This sequence will henceforth be referred to as the best action sequence, given the goal of minimizing the deviation in heading angle from the goal direction. However, the best action sequence is not visible in figure 7.17 due to the high density of sequences.



(a) The different action sequences **(b)** The depth image

Figure 7.17

For this particular evaluation of action sequences, the model has determined that it is safe to move to the left and unsafe to continue flight to the right, which is reasonable when considering the corresponding depth image in 7.17 given that there is an obstacle to the right-hand side of the image. An alternative visualization, at a different point in time, is visualized in figure 7.18 where only the first and last action in the different sequences are visualized. Here, the model has determined that it is safe for the RMF to execute forward motion and unsafe to turn to either the left or the right. Again, considering the depth image in 7.18, the evaluation is reasonable given that there are obstacles to the left and right sides of the robot. Here, the best action sequence is visualized in blue, indicating that the robot prefers a downward motion. The preference for moving in a downward trajectory is prevalent during the entire flight.

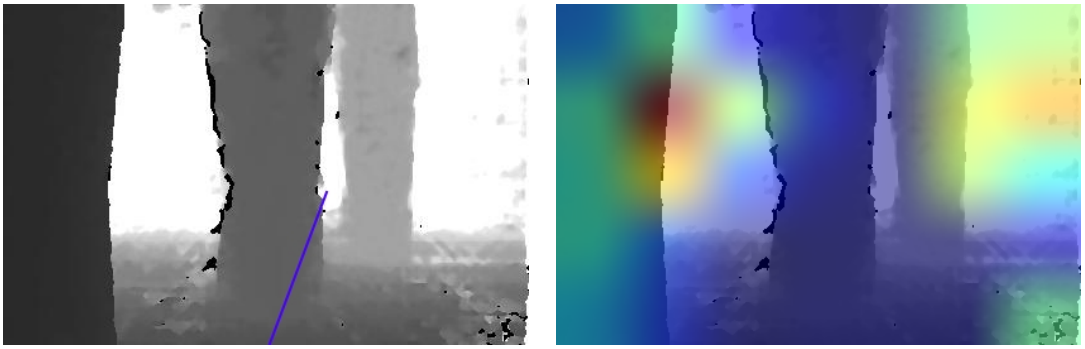


(a) The different action sequences **(b)** The depth image

Figure 7.18

Another exciting aspect when evaluating action sequences is determining which parts of the observation contribute to the different outcomes. More specifically, which parts of the images are essential to presume safe action sequences, and which parts are instrumental in determining a high probability of collision. Gradient-weighted class activation mapping (Grad-CAM) is a visualization method for CNNs that allow for "visual explanation" for decision making based on the gradients of a network, given predicted outputs. Observing the activations from the output of the last pooling layer in our encoder network enables us to produce a coarse localization map where the crucial regions in the depth image are highlighted. For the following visualizations, highlighted regions indicate activations within the encoder network. The darker blue color in the Grad-CAM visualizations indicates no activation.

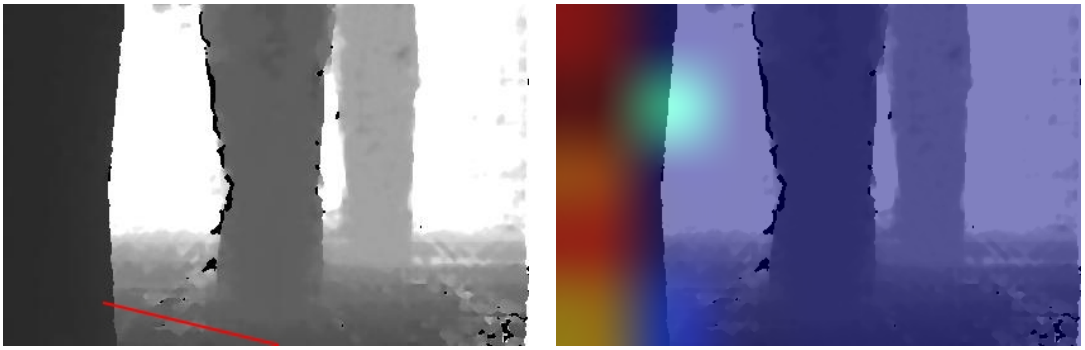
In figure 7.19 we can observe a depth image with the best action sequence, a safe sequence with the lowest deviation to the goal direction, and the corresponding Grad-CAM visualization. In the depth image, three pillars are present: one to the left, close to the robot, one in the center of the image, and one further back to the right. The Grad-CAM visualization shows highlighted regions on the right-hand side of the image where obstacles are absent. Furthermore, the pillar to the image's left and the free space behind it are highlighted. The immediate collision threat in the left pillar and the obstacle-free traversable space appear more pronounced in the Grad-CAM visualization.



(a) Depth map with the best action sequence **(b)** Grad-CAM visualization

Figure 7.19: Depth map and the corresponding GRAD-cam visualization corresponding to the best action sequence.

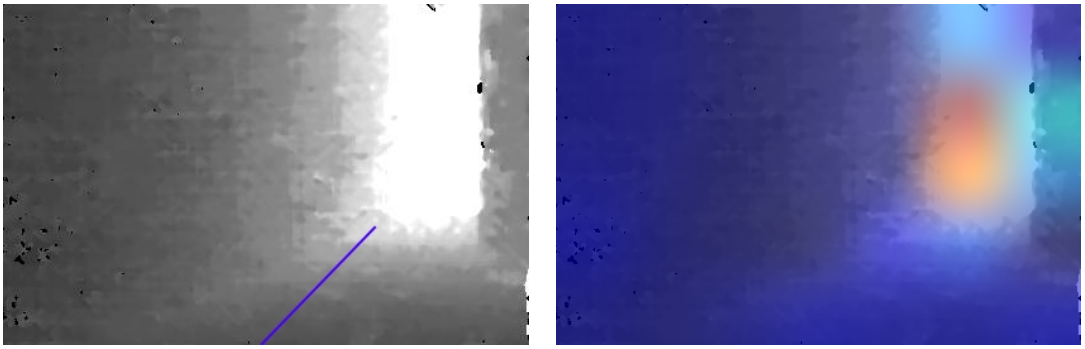
Next, we have the same scene, this time with the action sequence most likely to cause a collision. From figure 7.20, the highlighted region in the Grad-CAM visualization is almost entirely on the left pillar, which poses the most immediate threat to a collision-free path. Interestingly, the floor is not highlighted even though it could be a potential collision threat if the robot was allowed motion in the z-direction.



(a) Depth map with the least safe action sequence **(b)** Grad-CAM visualization

Figure 7.20: Depth map and the corresponding GRAD-cam visualization corresponding to the worst action sequence.

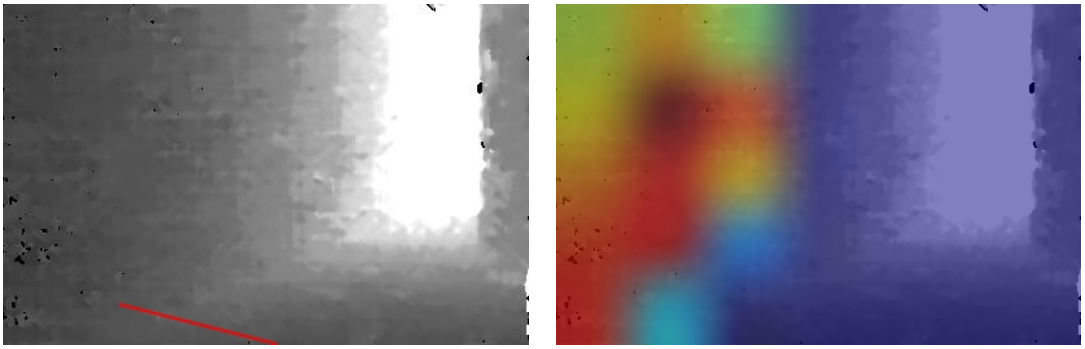
Moving on to a different scene, in figure 7.21, the RMF has moved to the left side of the corridor and is facing a wall. The action sequence that minimizes the goal objective is visualized in blue, pointing towards an obstacle-free space between the wall and a pillar on the right-hand side of the image. The localization map highlights the obstacle-free space between the wall and the pillar. Interestingly, the left-most part of the wall, along with the floor and the pillar, is barely highlighted in the Grad-CAM visualization, indicating that these areas also play a part in determining the action sequence.



(a) Depth map with the best action sequence **(b)** Grad-CAM visualization

Figure 7.21: Depth map and the corresponding GRAD-cam visualization corresponding to the best action sequence.

The wall is the most prominent feature in determining the collision for the action sequence most likely to cause a collision, as visualized in figure 7.22. This makes sense given the current trajectory of the robot, moving left, as the wall poses the most immediate threat for collision. Again, the floor is not highlighted.

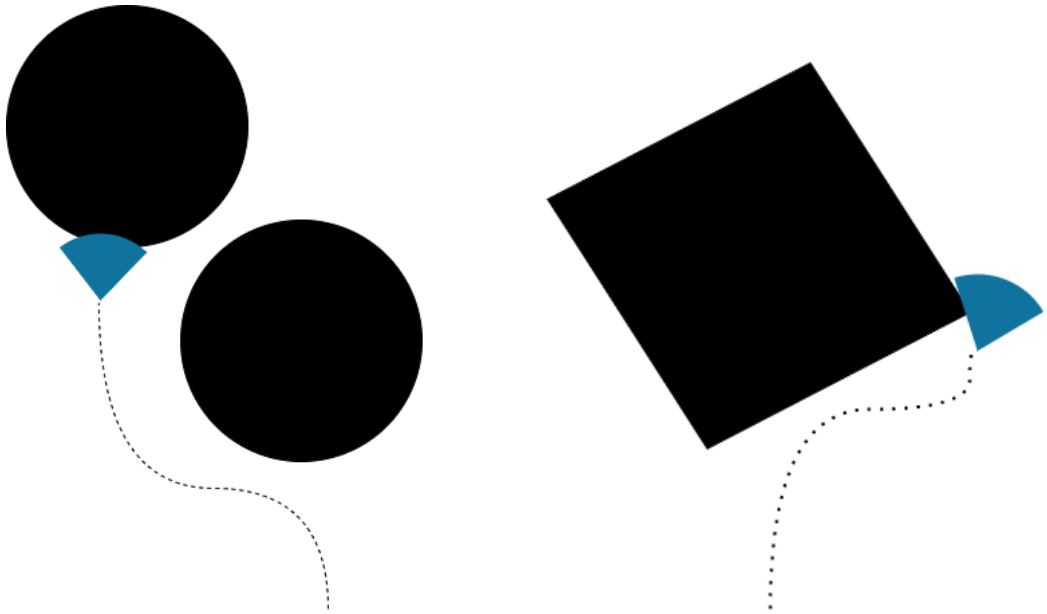


(a) Depth map with the least safe action sequence **(b)** Grad-CAM visualization of the least safe action sequence

Figure 7.22: Depth map and the corresponding GRAD-cam visualization corresponding to the worst action sequence.

Although the results presented this far mainly highlight a well-behaving navigation policy, the results in table 7.1 also tells us that the policy is not perfect and prone to collision, especially in the hard environment. The collisions can be separated into two categories: head-on collisions and pass-by collisions. Head-on collisions are when an object is well within the observed depth map. The cause of head-on collisions was, more often than not, objects close together within the environment. When two objects were close together, the drone managed to avoid the first obstacle by turning around it. After avoiding the first obstacle, the robot would turn towards the goal direction, where there would be another obstacle. With little to no time to react,

the robot would collide head-on. These collisions accounted for the majority of collisions in the hard environment. On the other hand, pass-by collision is when an obstacle is present within the FOV, located towards the edges of the observed depth. This causes the robot to clip the obstacle when attempting to traverse around it. These collisions would usually happen as a consequence of the robot turning towards the goal direction too quickly after initially attempting to avoid an obstacle. Pass-by collisions were the primary cause of collisions in the easy environment but also occurred in the hard environment. The most usual causes for head-on and pass-by collisions are visualized in figure 7.23.



(a) Head-on collision **(b)** Pass-by collision

Figure 7.23: The two different collisions

Overall, safe navigation has proved to be more difficult when there is an abundance of obstacles. The easy environment saw a success rate of 80% along with an average distance traveled above 100 meters, suggesting that the navigation policy is mainly sufficient to traverse corridors of such complexity. On the other hand, we saw a decrease in performance in the hard environment on the number of successful runs and the average distance traveled, resulting in 10% and close to 47 meters, respectively. Furthermore, the robot can reason in real-time about the environment based on the observations, as is visualized through the trajectories in figure 7.17 and 7.18. The Grad-CAM visualizations illustrate which parts of the observed images are essential to calculating collision probabilities of the best and worst action sequences. The results show that obstacle-free space is highlighted for safe action sequences, whereas immediate collision threats are more prominent in the Grad-CAM visualization for unsafe action sequences. Furthermore, collisions happen, and the type of collisions are separated into head-on and pass-by collisions. Head-on collisions accounted for most of the collisions in the hard environment, whereas pass-by collisions occurred more often in the easy environment.

Chapter 8

Discussion

As illustrated and described in chapter 7, there are some shortcomings within the different components leading up to the performance of ORACLE. Still, the overall results are encouraging. In this section, we aim to discuss the results presented by focusing on the probable causes as to why the results are as they are. Furthermore, we will suggest improvements or different approaches that, in turn, could be beneficial for further development.

8.1 Stereo Matching

Stereo matching was implemented in used in order to bridge the reality gap and increase sim-to-real transfer. We have seen how semi-global matching has been used to produce depth images.

First, through the results, it is evident that texture is essential when performing semi-global matching. Through illustrations of different scenes, the matching algorithm proved useful with enough texture as it performed well on multiple objects, like the tables and trees. However, it struggled more with less texture, as was palpable with the fences. In addition, the textureless background, often towards the images' ceiling, proved difficult when matching on the stereo pair. These results are not unanticipated from the theory established in section 3.5. An immediate improvement would be to add more texture to the objects, particularly the fences, as well as to the background. However, this would require more feature engineering, which is tedious. Unfortunately, applying sufficient texture to the fences proved difficult as the objects were impervious to details other than a color change. Instead, for training, it would be possible to remove the fences entirely, creating a slightly simpler environment for data collection.

Additionally, the OpenCV implementation of semi-global matching contains multiple tunable parameters, including penalties and smoothness. Finding the optimal combination of these parameters is complex, and in this thesis, the parameters used were inspired by [46]. The set of parameters is likely not optimal and could thus be improved. Determining the optimal

set of parameters is impossible to obtain through manual tuning, given the endless possibilities. Alternatively, finding a better set of parameters, perhaps the optimal set, could be solved through a learning-based approach. [47] used a deep neural network to learn the penalties and smoothness parameters of semi-global matching on the KITTI dataset. The KITTI dataset is a benchmark dataset for computer vision related to autonomous driving, containing real images of various traffic scenarios[48]. The results of [47] showed an improvement toward accurate matching in the dataset, highlighting how a learning-based approach can be used to improve semi-global matching. Adopting the method of [47], a learning-based approach to parameter selection could likely be beneficial for our matching purposes and potentially make subsequent learning easier.

Another issue encountered with the stereo matching was the range of depth when matching. As established in the theory section of semi-global matching 3.5, depth is inversely proportional to the disparity. Hence, more extensive disparity results in less depth. However, when initially deploying the matching algorithm and converting to depth, the depth across all images was on a scale from -1.5 and 3 meters, a range far from the actual depth as the stereo camera captured objects away as 50 meters. The negative part of the scale can be explained through invalid matches being converted to depth. However, the positive part of the scale only covering up to 3 meters is more illogical. When the range scale is inaccurate, the often logical explanation converges towards the camera setup and the images not being rectified. Despite that, the documentation of the stereo camera used in the simulation suggests that the stereo pair is rectified. Hence, the range scale is puzzling, and the origin of the range scale is uncertain. Given that the scale was off, it required engineering to adapt the scale to fit the images in question. Given that this scale was not perfectly linear, some distances in the resulting depth images are slightly off concerning actual depth. However, this is not expected to be a decisive factor when running the planner in the evaluation environments as the images received in real-time are identical in range to the images used for training.

8.2 Representation Learning and VAE

8.2.1 How The Results Affect Navigation

First, we will discuss the training history for the VAE, visualized in figure 7.4. The history decreases rapidly at the beginning, which is typical for machine learning tasks. This is due to the solution being far from an optimum solution, resulting in more significant gradients and thus more extensive updates. The gradient will eventually be more diminutive as the model approaches a local or global optimum. Simultaneously, as the model moves towards a solution in the optimization space, the adaptive optimizer Adam makes the rate at which the parameters in the network change even smaller. This effect is visible in the training history, as the respective losses converge towards more or less constant values. At the end of the training, there is a visible value gap between the training and validation losses. This is known as a generalization or training gap and is impossible to remove entirely. However, it could perhaps be reduced by adopting additional regularization methods, such as Monte Carlo dropout. Still, the validation loss does not increase during the training process, indicating that the model

does not overfit the training data. Given the training history in figure 7.4, the model seems to generalize well to the validation data. Naturally, this would most likely indicate that the model would generalize well to similar but previously unseen data, namely data received in real-time when traversing the two evaluation environments.

An interesting observation about the reconstructed images is that there is a trend for larger objects, for example, pyramids and arches, to be reconstructed. In comparison, smaller objects, like the chairs or the cross-fences, are more likely to disappear or at least lack coherent features. Intuitively, this is reasonable given the utilized loss function of the evidence lower bound. Previously established, the reconstruction term of the loss contributes the most to the overall loss. Hence, the model is more likely to properly reconstruct larger objects to optimize the loss, as they cover a more significant section of the images. This means that the more prominent objects are more likely to be represented in the latent space. However, this may not be optimal concerning motion planning as objects closer to the robot are more critical to detect to execute safe actions. Hence, a possible extension to the VAE to make feature extraction more suitable for motion planning could be to add a weight term to the loss function. Adding more constraints on objects at lesser depth makes objects further away less critical to minimizing the negative evidence lower bound.

The results in chapter 7.2 show how the VAE is used for representation learning, thus learning the most salient features in the depth images. The reconstruction of images is usually satisfactory when it comes to preserving depth but is more lacking in retaining geometric features. For more complex scenes, these results partially crumble, as the results have shown an increased difficulty in reconstructing the depth in scenes where obstacles/objects are abundant or where fences are close to the camera. In other words, the reconstructions are lacking when many features are present within the images. The reconstructions could potentially be a direct consequence of the results observed from stereo matching. However, from the part of the results where fences are closer to the robot, the reconstructions are still missing components that are somewhat present in the filtered depth maps. More likely, these results are explainable by evaluating the latent space. For the results obtained, the latent dimension of choice was 64. However, this does not necessarily mean that the latent space could represent 64 different obstacles. Realistically, when describing an object in the latent space representation, at least a couple of variables are utilized to describe a feature in the input, for example, the position and size of the object. In practice, this means that, realistically, a latent space dimension of 64 can only describe a couple of handfuls of different features. Hence, for cluttered scenes, where there are more objects, it is realistic that there would be gaps in the reconstruction as the size of the latent space is insufficient to capture all the features in the input image. This could be the case for the fences, as well as the dense environments.

8.2.2 Improving Representation Learning

Perhaps the most obvious solution to having a latent space incapable of representing all features in the input is to increase the size of the latent space. Doing so would allow more features in the input to be encoded into the latent space, also making reconstruction easier as there are

more latent variables to draw from. However, increasing the latent space comes with some disadvantages. First, increasing the latent size would increase the capacity of the VAE structure, possibly allowing shortcuts where information is copied rather than learned. For our particular problem, one could likely increase the latent space quite a lot without running into this particular problem. However, it is worth considering to what extent the size of the latent space could be expanded while still learning the most salient features. Second, although the model's performance is likely to increase by having a larger latent space, the model would also require more training. Naturally, this would increase the time required to reach a satisfactory model. Additionally, a model with higher capacity is more likely to overfit on the data[49, 50]. This, in turn, either put more constraints on the learning process, requiring further regularization, or on the data to have a higher variance. With our data set containing roughly 230000 images for training, it is reasonable to assume sufficient variance to support a higher latent dimension. Lastly, and perhaps most important, is the consequences of increasing the capacity, namely the model complexity. Increasing the complexity of the model is equal to increasing the number of weights or parameters in the network. Besides making training more cumbersome as more gradients need to be computed during backpropagation, the forward pass will also be more cumbersome. This will increase inference time when running the planner in real-time. Given that our developed model is implemented on a robot to traverse a cluttered environment, computational time is a severely important constraint. Therefore, it should at least be considered before increasing the complexity of the model.

Another potential solution related to the latent space would be that of *disentanglement*. The VAE we have used in learning safe and resilient navigation has an entangled latent space. A latent space is said to be entangled if one node, also known as a *generative factor*, contributes to two or more features in the reconstructed image. On the contrary, in a disentangled latent space, one generative factor is only mapped to one feature in the output [51]. Independent nodes in the latent space are not sufficient to ensure disentanglement. Having a disentangled latent space has proved useful as fewer nodes in the latent space are required to learn a represented feature[51]. An extension to the VAE[34] that utilizes a disentangled latent space is the β-VAE [52]. The β-VAE is created through a slight modification in the loss objective:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}, \boldsymbol{z}, \beta) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}\,|\,\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}\,|\,\boldsymbol{z})] - \beta\, D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}\,|\,\boldsymbol{x}) || p_{\boldsymbol{\theta}}(\boldsymbol{z})) \tag{8.1}$$

Having $\beta = 1$ leads to the original VAE objective. Utilizing $\beta > 1$, however, puts a stronger constraint on the capacity of the encoder network, constituting a more considerable bottleneck for the latent distribution[52]. Alternatively, this β can be viewed as forcing an upper bound on the information transmitted through the network. In other words, the bottleneck on $\boldsymbol{z}$ encourages the model to learn the most efficient representation of the input data, thus creating a disentangled latent space. However, this usually comes at the cost of having a weaker reconstruction due to the loss of high-frequent details. Still, using a β-VAE has proved beneficial for sim-to-real transfer in robotics[53]. Furthermore, for our purposes of safe navigation, adopting the β-VAE approach could make representation learning more effective without expanding the latent space and increasing inference time.

8.3 ORACLE-VAE and Safe Navigation

8.3.1 From a Machine Learning Perspective

We will start the discussion about ORACLE-VAE by analyzing the performance from a pure machine learning standpoint. First, the binary loss in figure 7.11 on the collision probabilities sees quite a large generalization gap. Furthermore, it seems to overfit the training data. This can be seen from the increase in the validation loss compared to the training loss. The learning curves for the successive metrics, accuracy, precision, recall, and F1-score, from figure 7.12, 7.13, 7.14 and 7.15 respectively, are likely as observed due to the overfitting observed from the loss. More notably is the validation precision, which decreases. Unfortunately, precision and recall are often in tension, meaning that precision is likely to decrease if recall increases. Despite the dip in validation precision, the F1-score remains mostly the same, showcasing how the overall performance is stable. Additionally, from the validation F1-score, the value increases until around 50 epochs. Coincidentally, this is also when the validation loss is at its lowest value and when the validation accuracy stabilizes. This tells us that training beyond this point is mostly a "tug-of-war" between precision and validation. Interestingly, the history of the VAE loss in ORACLE-VAE is remarkably similar to the stand-alone VAE in that it is not overfitting the training data. Concurrently, the gap between the values of the training and validation loss is smaller. Regularization is utilized to reduce the generalization gap. The reduced gap observed can likely be attributed to the use of Monte Carlo dropout in addition to batch normalization.

Furthermore, when optimizing on two losses, the optimization space in which the model attempts to find an optimum becomes more complex. In turn, this creates a more complex optimization problem. It is not entirely unlikely that the loss for the reconstructions and the loss for the collision probabilities pull the model in different directions in the optimization space, as the reconstruction loss values larger objects that cover a larger portion of the images. In contrast, the collision probabilities would likely emphasize objects close by. With the chosen loss weights $\alpha_c = 1.0$ and $\alpha_o = 0.001$, the different losses, after the initial decrease in values, will contribute with values at roughly 25 and 56 ($56000 \cdot 0.001$), respectively to the total loss. Consequently, it will yield larger gradients for the reconstruction loss, resulting in larger parameter updates for the weights that infer the reconstructed output. If the two losses pull in different directions in the optimization space, it would explain the overfitting on collision probabilities and not on the reconstructed images. Furthermore, it could also explain the gap in values in the ELBO loss when trained in union with collision probabilities compared to the stand-alone VAE. Different approaches to optimizing the total loss of ORACLE-VAE, such as pre-training and fine-tuning of various components and freezing different layers, could result in a learning process where less overfitting occurs. Even so, the best model for safe navigation in cluttered environments was the model trained end-to-end. An interesting addition to the training of the model could be to adaptively assign weights to the different losses, α_o and α_c, based on the magnitude of values in the respective losses.

8.3.2 Navigation in Cluttered Environments

The results when navigating cluttered environments are greatly affected by the difficulty. The easy environment yielded a substantially higher success rate than the hard environment, at 80% and 10% respectively.

The results in the easy environment are encouraging as they demonstrated that it is possible to execute mostly safe navigation in relatively simple environments. Through the 20 runs, 16 runs were successful, and another two runs managed to avoid all obstacles safely but crashed close to the goal position, as is visualized in 7.16a. After clearing the obstacles, the RMF had a bias toward moving to the right. A possible explanation for this behavior can be traced back to stereo matching. The environment was defined by walls on two sides, while the two remaining sides, at the spawn and goal location, were not encapsulated by walls, as illustrated in figure 6.10. In the simulated environments, when the robot has managed to avoid all obstacles along the corridor, the observed depth turns entirely to noise due to lack of texture. As the model approaches the goal, this seemingly creates confusion towards determining safe action sequences, resulting in the RMF colliding on two separate runs. These results also illustrated how the learned navigation policy has a bias for a motion to the right when it is impossible to interpret depth. Presumably, a quick fix to this particular response would be to alter the environment to be squared off by walls on all sides of the environment.

When observations are entirely noise, the bias towards the movement to the right is less pronounced in the hard environment. As the environment is denser with more obstacles, a pillar is very close to the goal position, as visualized in figure 7.16b. When the robot avoids the last obstacle, it is just meters away from the goal, thus potentially giving it less time to move to the right. Still, from the two trajectories, it is possible to observe a slight bias towards rightward movement. Again, this is probably attributed to the open-ended environment, as illustrated in figure 6.11.

Furthermore, from the results in chapter 7.3, the robot is capable of making reasonable predictions towards safe actions given the observed depth, as is visualized in figure 7.17 and 7.18. Furthermore, the Grad-CAM visualization illustrates how the different components in observed depth influence decision-making. Here, the least safe action sequences are more influenced by obstacles that are more likely to be an immediate cause of collisions. In contrast, safe actions are more affected by the obstacle-free space visible in the observations. Therefore, the Grad-CAM visualization and the predicted action sequences indicate a well-behaving navigation policy. Interestingly, the robot often preferred downward motion, as can be observed with the trajectories and partially from the Grad-CAM visualization, somewhat highlighting the floor in the case of a safe action sequence. Intuitively, this is illogical as the floor is a surface it is possible to collide with. Also, remember from stereo matching how textureless background would make objects appear larger. As the textureless background only occurred towards the top of objects, this could have implications for learning, resulting in the learned policy preferring downward motion. However, running a short test, allowing the robot to have non-zero velocity in the z-direction, v_z, resulted in the robot colliding with the floor. Hence, a

more probable explanation for the preferred downward movement is an error when labeling collision with the floor through self-supervision. If collisions with the floor are not labeled in the data set, it would explain the desire to move towards the ground, as such movement is "safe" according to the data used to train the policy.

Collisions occurred more frequently in the hard environment than in the easy one. The two environments also saw different collisions, with head-on collisions being the primary cause of collisions in the hard environment, while pass-by collisions were more frequent in the easy environment. Both types of collisions can largely be attributed to the desire to move towards the goal position. After initially avoiding an obstacle, it is common for the object to move out of the field of view due to a sharp turn. Because the robot only evaluates actions based on the current depth image and state, most subsequent action sequences would be evaluated as safe. Given the desire to move in the goal direction, the robot will likely choose an action with a significant change in heading angle, inducing another sharp turn. For the easy environment, this would occasionally result in a pass-by collision as the robot turned too quickly. While for the hard environment, the initial movement to avoid an obstacle would usually be sufficient to clear the obstacle, avoiding pass-by collisions. However, when a second object appears quickly after clearing the first obstacle, it infuses confusion, making subsequent evaluation more difficult, resulting in head-on collisions. The underlying controller for heading was tuned in an attempt to solve this issue as it influences the rate at which the robot turns. However, forcing a slower rate of change for heading would often result in the robot being unable to initially avoid the obstacle, ending flight with a head-on collision. Changing the parameter when calculating the weighted collision cost in equation 6.4 to increase the influence of future actions was also experimented with. However, this would often result in more sporadic movement as too little weight was put into immediate actions. On the other hand, decreasing the weight on future actions and prioritizing more immediate actions allowed the robot to get too close to obstacles before turning. Thus, increasing and decreasing λ resulted in more frequent collisions.

8.3.3 Possible Improvements and Further Work

Several extensions to our method were mentioned through discussion about stereo matching and image reconstruction, which could result in an improved navigation policy. For example, it could be beneficial to adopt the ideas of learning-based semi-global matching and β-VAE. However, it is possible to further extend the model by allowing the robot to choose its velocity freely, incorporate memory about previous observations, and perhaps utilize reinforcement learning for fine-tuning.

During data collection, the actions performed by the robot were sampled when creating motion primitive trajectories. Hence, different actions across different epochs were assigned different velocities in forward and vertical directions, v_x and v_z. As a result, the actions stored in the data set used for training contain action sequences varying in velocities. In theory, this would indicate that the robot implicitly learns to adjust its velocity to avoid obstacles. However, because the motion primitives library produces actions with a constant forward velocity, the robot's ability to adjust its speed to avoid obstacles is not tested when traversing the obstacle-filled

corridors. Allowing the robot to determine its velocity would be an exciting addition to the ORACLE framework. In practice, this could be implemented by sampling the forward velocity from a uniform distribution in the motion primitives library, adding an element of reasoning when evaluating the action sequences. If the robot could adapt its velocity through reasoning, it would be expected to observe higher velocities when navigating in obstacle-free space and slower movement in proximity to obstacles. Hence, variations in velocity could perhaps lead to an improved navigation policy.

The causes of collisions emphasize that the robot only acts on the current observations and thus does not consider previous observations. Introducing temporal awareness into the model could be beneficial as previous observations would be considered upon evaluating action sequences at a current point in time. Similar to the ANYmal robot[17], which can reason about movement in a dynamic environment, it would be possible to adopt a recurrent structure to the observations, thus introducing a memory of previous observations. More specifically, during training, one could utilize an LSTM network to predict future observations, optimizing the difference between predicted and actual observations. By considering previous observations when evaluating action sequences, it would be reasonable to suggest that the frequency of which the collisions we have encountered would be reduced, as the robot would possess knowledge of the location of obstacles even though no obstacles are present within the field of view after executing a collision avoiding maneuver.

More far-fetched, combining the learned navigation policy, learned through supervised learning, with reinforcement learning could yield interesting results. Reinforcement learning has also proved useful for aerial robotic applications, allowing for precise and collision-free navigation[54–56]. However, these methods are trained end-to-end on reinforcement learning algorithms. For our model, being trained through supervised learning, it would perhaps be more interesting to adopt some elements of RL without having to train the model end-to-end. Encouragingly, supervised learning and RL have been utilized in union for continuous control[57], increasing performance in simulation. By imposing the learned policy onto an RL framework, transferring the behavior[58] could thus potentially be used in order to fine-tune the navigation policy to increase performance.

Chapter 9

Conclusion

More effective and lighter sensors, more on-board computational power, better software and hardware, and difficulties related to data association in traditional SLAM methods have resulted in the increasing popularity of learning-based approaches towards solving complex navigation tasks. Learning-based approaches for ground and aerial robots [9–11, 26] have shown remarkable results in collision-free navigation. The goal of this thesis was to further improve upon the ORACLE[11] neural network architecture, which has proved successful in safe navigation in cluttered environments. Attempting to improve the navigation policy and increase sim-to-real transfer was done by adding depth images created through semi-global matching, a stereo matching algorithm, and representation learning through probabilistic modeling using a VAE. In order to conduct experiments, the relevant theory was presented, focusing on the various neural network architectures used in the ORACLE model.

The depth images created through semi-global matching illustrate how the texture is critical to obtaining meaningful depth images. Consequently, the matching algorithm struggled with fences, which lacked essential characteristics, and the untextured background, making surrounding objects appear larger than their actual size. Furthermore, the VAE used was well-behaved, primarily preserving depth, but struggled more with retaining geometric features. For denser environments, these results partially deteriorated as preserved depth in the reconstructed images was more inadequate. The results when navigating cluttered environments are two-fold. On the one hand, it is encouraging that the learned policy can frequently traverse a relatively simplistic environment, avoiding obstacles to reach a goal position. On the other hand, deploying the learned policy in a denser and more complex environment illustrates that there is still room for improvement as the robot collides more often. Still, the result shows how the RMF can reason about its observations to evaluate safe actions. Here, safe actions are more influenced by free space in the environments, whereas unsafe actions depend more on immediate threats to collision in close obstacles. Furthermore, when the robot collided, it was usually due to sharp turns to minimize the deviation in heading of the goal direction supplied by a global planner. The sharp turns can be seen as premature movement and are influenced

by the lack of temporal awareness.

Further improvements can be made to improve collision-free navigation. First, semi-global matching could potentially be improved through a learning-based approach[47]. Furthermore, additional work towards the simulation can be done by adding more texture, thus making semi-global matching more robust. Second, it can possibly further improve representation learning by altering the loss objective to emphasize closer objects. Furthermore, representation learning could be improved by either increasing the dimension of the latent space in the VAE or through disentanglement of the latent space. Adopting the idea of β-VAE[52] could, in turn, result in an increased amount of represented features in the latent space, making subsequent learning for collision probabilities easier. Lastly, further work can be done on ORACLE-VAE. As velocities are sampled when collecting data, the model should implicitly learn to adjust its velocity to navigate collision-free cluttered environments. However, when evaluating the model in cluttered environments, the robot is limited to constant velocities, thus not exploring the full potential of the navigation policy. Extending the evaluation procedure to allow the robot to adjust its velocity could result in improved performance. Furthermore, adopting temporal awareness through a recurrent neural network would add a sense of memory towards made observations, which could further improve the performance of the policy. More far-fetched, the training of ORACLE-VAE could potentially be used in union with RL as a means of fine-tuning, further improving the learned policy.

Bibliography

[1] S. Shen, N. Michael and V. Kumar, 'Autonomous multi-floor indoor navigation with a computationally constrained mav,' in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 20–25. DOI: 10.1109/ICRA.2011.5980357.

[2] S. Weiss, D. Scaramuzza and R. Siegwart, 'Monocular-slam-based navigation for autonomous micro helicopters in gps-denied environments,' *J. Field Robotics*, vol. 28, pp. 854–874, Nov. 2011. DOI: 10.1002/rob.20412.

[3] P. Murray and M. Schukat, 'Mav based slam and autonomous navigation: A view towards efficient on-board systems,' in *2017 28th Irish Signals and Systems Conference (ISSC)*, 2017, pp. 1–6. DOI: 10.1109/ISSC.2017.7983597.

[4] M. Blösch, S. Weiss, D. Scaramuzza and R. Siegwart, 'Vision based mav navigation in unknown and unstructured environments,' in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 21–28. DOI: 10.1109/ROBOT.2010.5509920.

[5] F. Fraundorfer *et al.*, 'Vision-based autonomous mapping and exploration using a quadrotor mav,' in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 4557–4564. DOI: 10.1109/IROS.2012.6385934.

[6] H. Durrant-Whyte and T. Bailey, 'Simultaneous localization and mapping: Part i,' *IEEE Robotics Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006. DOI: 10.1109/MRA.2006.1638022.

[7] T. Bailey and H. Durrant-Whyte, 'Simultaneous localization and mapping (slam): Part ii,' *IEEE Robotics Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006. DOI: 10.1109/MRA.2006.1678144.

[8] J. Fuentes-Pacheco, J. Ascencio and J. Rendon-Mancha, 'Visual simultaneous localization and mapping: A survey,' *Artificial Intelligence Review*, vol. 43, Nov. 2015. DOI: 10.1007/s10462-012-9365-8.

[9] G. Kahn, P. Abbeel and S. Levine, *Land: Learning to navigate from disengagements*, 2020. DOI: 10.48550/ARXIV.2010.04689. [Online]. Available: https://arxiv.org/abs/2010.04689.

[10] G. Kahn, P. Abbeel and S. Levine, *Badgr: An autonomous self-supervised learning-based navigation system*, 2020. DOI: 10.48550/ARXIV.2002.05700. [Online]. Available: https://arxiv.org/abs/2002.05700.

[11] H. Nguyen, S. H. Fyhn, P. D. Petris and K. Alexis, *Motion primitives-based navigation planning using deep collision prediction*, 2022. arXiv: 2201.03254 [cs.RO].

[12] C. Sampedro Pérez, A. Rodríguez Ramos, H. Bavle, A. Carrio, P. de la Puente and P. Campoy, 'A fully-autonomous aerial robot for search and rescue applications in indoor environments using learning-based techniques,' *Journal of Intelligent Robotic Systems*, vol. 95, pp. 1–27, Aug. 2019. DOI: 10.1007/s10846-018-0898-1.

[13] S. Hossain and D. Lee, 'Deep learning-based real-time multiple-object detection and tracking from aerial imagery via a flying robot with gpu-based embedded devices,' *Sensors*, vol. 19, p. 3371, Jul. 2019. DOI: 10.3390/s19153371.

[14] A. Giusti *et al.*, 'A machine learning approach to visual perception of forest trails for mobile robots,' *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2016. DOI: 10.1109/LRA.2015.2509024.

[15] S. Ross *et al.*, 'Learning monocular reactive UAV control in cluttered natural environments,' *CoRR*, vol. abs/1211.1690, 2012. arXiv: 1211.1690. [Online]. Available: http://arxiv.org/abs/1211.1690.

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

[17] D. Hoeller, L. Wellhausen, F. Farshidian and M. Hutter, 'Learning a state representation and navigation in cluttered and dynamic environments,' 2021. DOI: 10.48550/ARXIV.2103.04351. [Online]. Available: https://arxiv.org/abs/2103.04351.

[18] C. Richter and N. Roy, 'Safe visual navigation via deep learning and novelty detection,' Jul. 2017. DOI: 10.15607/RSS.2017.XIII.064.

[19] M. Popovic, F. Thomas, S. Papatheodorou, N. Funk, T. Vidal-Calleja and S. Leutenegger, *Volumetric occupancy mapping with probabilistic depth completion for robotic navigation*, 2020. DOI: 10.48550/ARXIV.2012.03023. [Online]. Available: https://arxiv.org/abs/2012.03023.

[20] J. Kober, J. Bagnell and J. Peters, 'Reinforcement learning in robotics: A survey,' *The International Journal of Robotics Research*, vol. 32, pp. 1238–1274, Sep. 2013. DOI: 10.1177/0278364913495721.

[21] M. Quigley *et al.*, 'Ros: An open-source robot operating system,' vol. 3, Jan. 2009.

[22] N. Koenig and A. Howard, 'Design and use paradigms for gazebo, an open-source multi-robot simulator,' in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[23] G. Kahn, A. Villaflor, P. Abbeel and S. Levine, *Composable action-conditioned predictors: Flexible off-policy learning for robot navigation*, 2018. DOI: 10.48550/ARXIV.1810.07167. [Online]. Available: https://arxiv.org/abs/1810.07167.

[24] G. Kahn, A. Villaflor, B. Ding, P. Abbeel and S. Levine, *Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation*, 2017. DOI: 10.48550/ARXIV.1709.10489. [Online]. Available: https://arxiv.org/abs/1709.10489.

[25] M. W. Mueller, M. Hehn and R. D'Andrea, 'A computationally efficient motion primitive for quadrocopter trajectory generation,' *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, 2015. DOI: 10.1109/TRO.2015.2479878.

[26] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun and D. Scaramuzza, 'Learning high-speed flight in the wild,' *Science Robotics*, vol. 6, no. 59, Oct. 2021. DOI: 10.1126/scirobotics.abg5810. [Online]. Available: https://doi.org/10.1126%2Fscirobotics.abg5810.

[27] A. Vaswani *et al.*, *Attention is all you need*, 2017. DOI: 10.48550/ARXIV.1706.03762. [Online]. Available: https://arxiv.org/abs/1706.03762.

[28] W. Li *et al.*, *Interiornet: Mega-scale multi-sensor photo-realistic indoor scenes dataset*, 2018. DOI: 10.48550/ARXIV.1809.00716. [Online]. Available: https://arxiv.org/abs/1809.00716.

[29] A. Burkov, *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019, ISBN: 9781999579517. [Online]. Available: https://books.google.no/books?id=0jbxwQEACAAJ.

[30] D. M. W. Powers, 'Evaluation: From precision, recall and f-measure to roc, informedness, markedness and correlation,' 2020. DOI: 10.48550/ARXIV.2010.16061. [Online]. Available: https://arxiv.org/abs/2010.16061.

[31] Y. Sasaki, 'The truth of the f-measure,' *Teach Tutor Mater*, Jan. 2007.

[32] S. Hochreiter and J. Schmidhuber, 'Long short-term memory,' *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[33] D. J. Rezende, S. Mohamed and D. Wierstra, *Stochastic backpropagation and approximate inference in deep generative models*, 2014. DOI: 10.48550/ARXIV.1401.4082. [Online]. Available: https://arxiv.org/abs/1401.4082.

[34] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, 2013. DOI: 10.48550/ARXIV.1312.6114. [Online]. Available: https://arxiv.org/abs/1312.6114.

[35] C. Hubmann, J. Schulz, M. Becker, D. Althoff and C. Stiller, 'Automated driving in uncertain environments: Planning with interaction and uncertain maneuver prediction,' *IEEE Transactions on Intelligent Vehicles*, vol. PP, pp. 1–1, Jan. 2018. DOI: 10.1109/TIV.2017.2788208.

[36] H. Hirschmuller, 'Stereo processing by semiglobal matching and mutual information,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 328–341, 2008. DOI: 10.1109/TPAMI.2007.1166.

[37] S. Birchfield and C. Tomasi, 'Depth discontinuities by pixel-to-pixel stereo,' in *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, 1998, pp. 1073–1080. DOI: 10.1109/ICCV.1998.710850.

[38] D. Scharstein, R. Szeliski and R. Zabih, 'A taxonomy and evaluation of dense two-frame stereo correspondence algorithms,' in *Proceedings IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV 2001)*, 2001, pp. 131–140. DOI: 10.1109/SMBV.2001.988771.

[39] S. Fidler, 'Depth from stereo,' p. 15, Mar. 2021. [Online]. Available: http://www.cs.toronto.edu/~fidler/slides/2015/CSC420/lecture12_hres.pdf.

[40] R. Smith, *Open dynamics engine*, http://www.ode.org/, 2008. [Online]. Available: http://www.ode.org/.

[41] M. Woo, J. Neider, T. Davis and D. Shreiner, *OpenGL programming guide: the official guide to learning OpenGL, version 1.2*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[42] F. Furrer, M. Burri, M. Achtelik and R. Siegwart, 'Rotors – a modular gazebo mav simulator framework,' in Jan. 2016, vol. 625, pp. 595–625, ISBN: 978-3-319-26054-9. DOI: 10.1007/978-3-319-26054-9_23.

[43] H. R. Kam, S.-H. Lee, T. Park and C.-H. Kim, 'Rviz: A toolkit for real domain data visualization,' *Telecommun. Syst.*, vol. 60, no. 2, pp. 337–345, Oct. 2015, ISSN: 1018-4864. DOI: 10.1007/s11235-015-0034-5. [Online]. Available: https://doi.org/10.1007/s11235-015-0034-5.

[44] P. De Petris, H. Nguyen, T. Dang, F. Mascarich and K. Alexis, 'Collision-tolerant autonomous navigation through manhole-sized confined environments,' in *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2020, pp. 84–89. DOI: 10.1109/SSRR50563.2020.9292583.

[45] K. He, X. Zhang, S. Ren and J. Sun, 'Deep residual learning for image recognition,' *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[46] P. De Petris, $T265_depth$, https://github.com/tiralonghipol/t265_depth/tree/master, 2021.

[47] A. Seki and M. Pollefeys, 'Sgm-nets: Semi-global matching with neural networks,' in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6640–6649. DOI: 10.1109/CVPR.2017.703.

[48] A. Geiger, P. Lenz and R. Urtasun, 'Are we ready for autonomous driving? the kitti vision benchmark suite,' in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361. DOI: 10.1109/CVPR.2012.6248074.

[49] K. Huesmann, S. Klemm, L. Linsen and B. Risse, *Exploiting the full capacity of deep neural networks while avoiding overfitting by targeted sparsity regularization*, 2020. DOI: 10.48550/ARXIV.2002.09237. [Online]. Available: https://arxiv.org/abs/2002.09237.

[50] R. Caruana, S. Lawrence and C. Giles, 'Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping.,' vol. 13, Jan. 2000, pp. 402–408.

[51] C. P. Burgess *et al.*, *Understanding disentangling in -vae*, 2018. DOI: 10.48550/ARXIV.1804.03599. [Online]. Available: https://arxiv.org/abs/1804.03599.

[52] I. Higgins *et al.*, 'Beta-vae: Learning basic visual concepts with a constrained variational framework,' in *ICLR*, 2017.

[53] I. Higgins *et al.*, *Darla: Improving zero-shot transfer in reinforcement learning*, 2017. DOI: 10.48550/ARXIV.1707.08475. [Online]. Available: https://arxiv.org/abs/1707.08475.

[54] Y. Song, M. Steinweg, E. Kaufmann and D. Scaramuzza, 'Autonomous drone racing with deep reinforcement learning,' *CoRR*, vol. abs/2103.08624, 2021. arXiv: 2103.08624. [Online]. Available: https://arxiv.org/abs/2103.08624.

[55] A. Singla, S. Padakandla and S. Bhatnagar, 'Memory-based deep reinforcement learning for obstacle avoidance in UAV with limited environment knowledge,' *CoRR*, vol. abs/1811.03307, 2018. arXiv: 1811.03307. [Online]. Available: http://arxiv.org/abs/1811.03307.

[56] O. Bouhamed, H. Ghazzai, H. Besbes and Y. Massoud, 'Autonomous uav navigation: A ddpg-based deep reinforcement learning approach,' in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5. DOI: 10.1109/ISCAS45731.2020.9181245.

[57] D. Kangin and N. Pugeault, 'Continuous control with a combination of supervised and reinforcement learning,' in *2018 International Joint Conference on Neural Networks (IJCNN)*, 2018, pp. 1–8. DOI: 10.1109/IJCNN.2018.8489702.

[58] V. Campos *et al.*, *Beyond fine-tuning: Transferring behavior in reinforcement learning*, 2021. DOI: 10.48550/ARXIV.2102.13515. [Online]. Available: https://arxiv.org/abs/2102.13515.

Appendix A

Acronyms

MAV - Micro aerial vehicle

RMF - Resilient micro flyer

SLAM - Simultaneous localization an mapping

NN - Neural networks

ANN - Artificial neural networks

DNN - Deep neural network

CNN - Convolutional neural network

RNN - Recurrent neural network

LSTM - Long short-term memory

GRU - Gated recurrent unit

MLP - Multi-layer perceptron

EA - Autoencoder

VAE - Variational autoencoder

ELBO - Evidence lower bound

PCA - Principal component analysis

TP - True positive

TN - True negative

FP - False positive

FN - false negative

ReLU - Rectified linear unit

Adam - Adaptive moment estimation

Adagrad - Adaptive gradient descent

RMSprop - Root mean square propagation

SGM - Semi-global matching

ROS - Robot operating system

API - Application programming interface

IMU - Inertial measurement unit

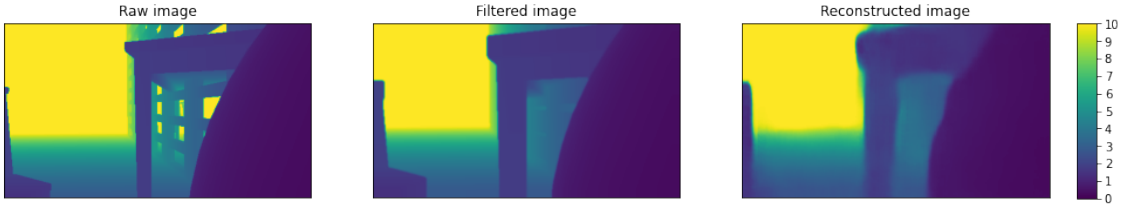
FPS - Frames per second

FOV - Field of view

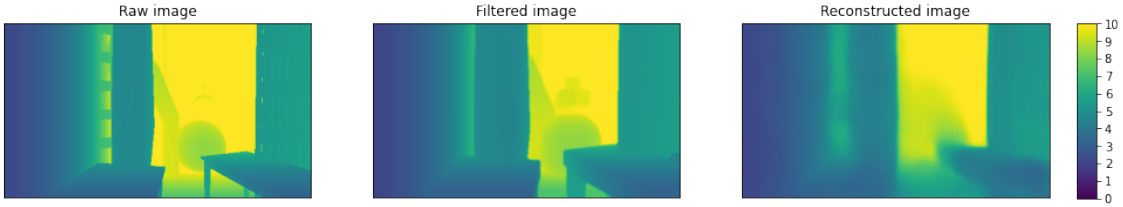
Grad-CAM - Gradient-weighted class activation mapping

Appendix B

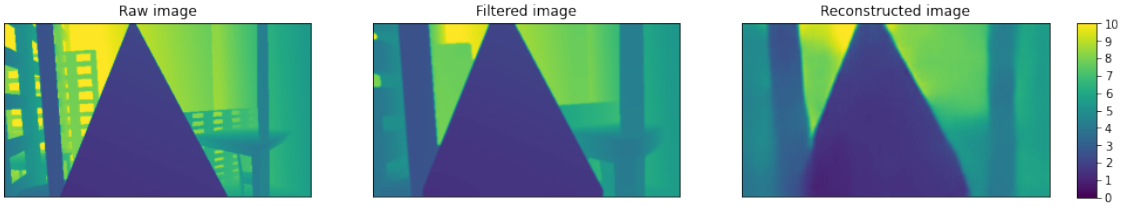
Image Reconstruction Of Depth Images From A Depth Camera



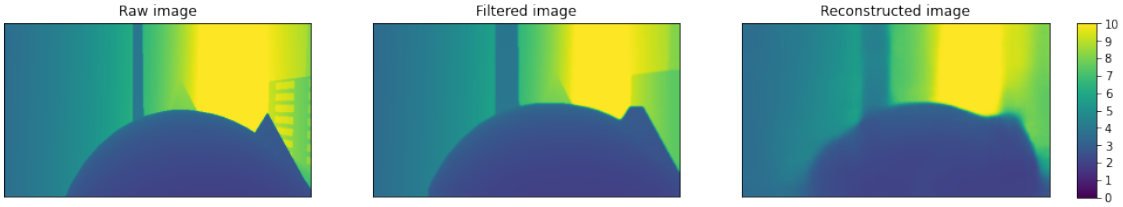
(a)



(b)

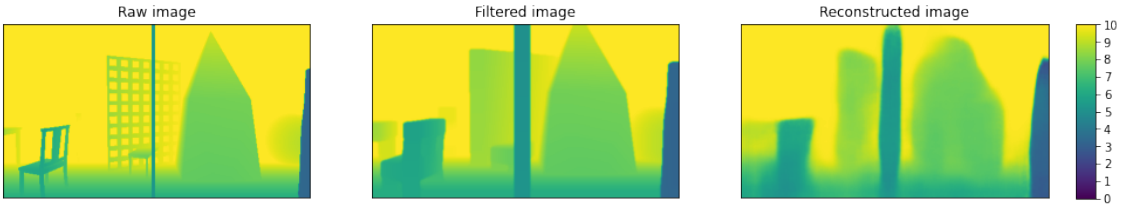


(c)

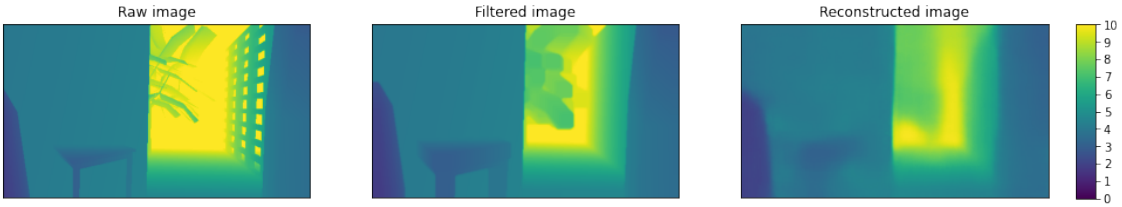


(d)

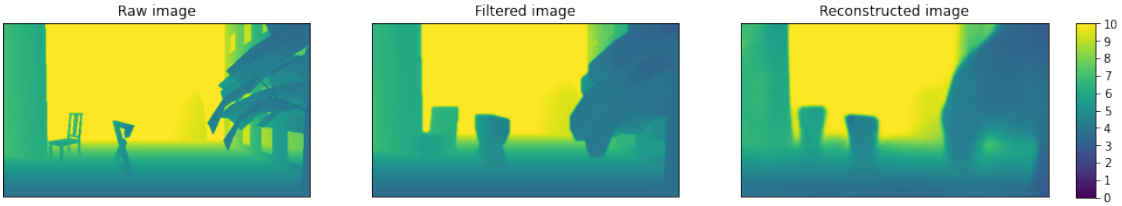
Figure B.1



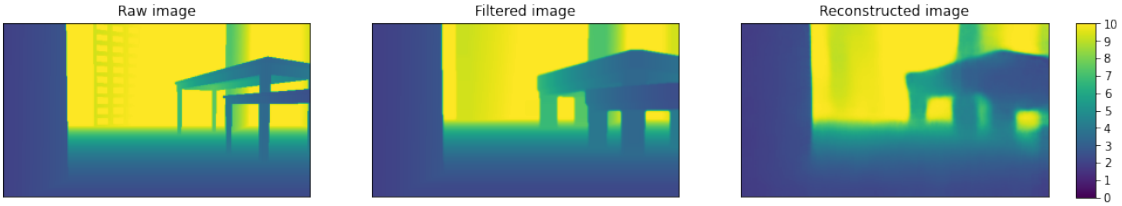
(a)



(b)

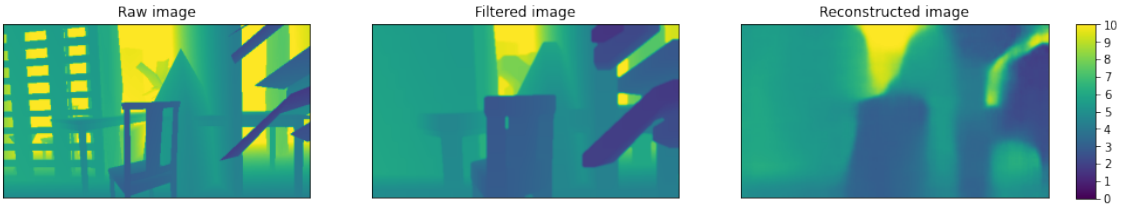


(c)

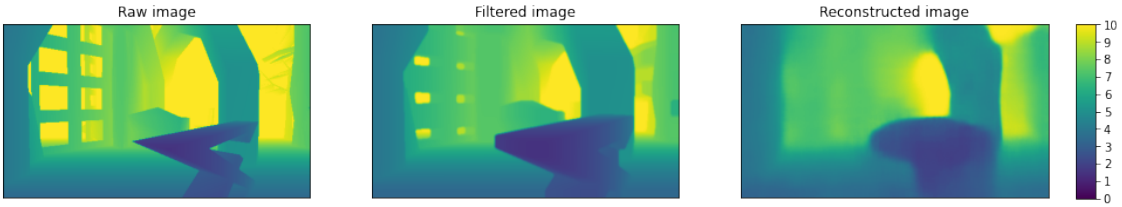


(d)

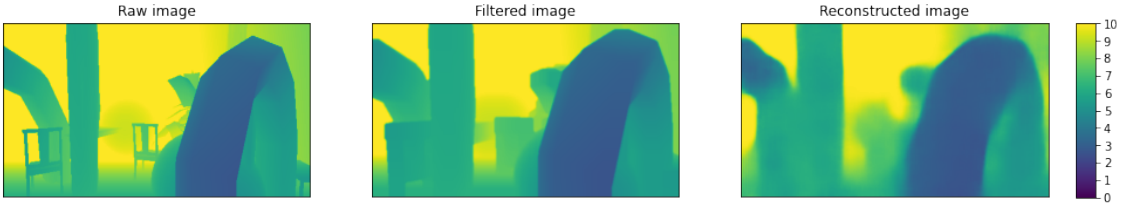
Figure B.2



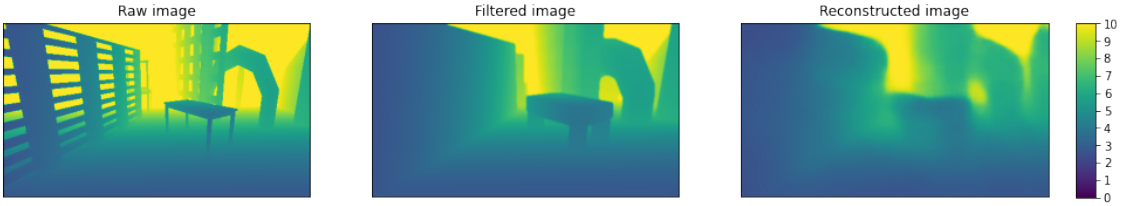
(a)



(b)



(c)



(d)

Figure B.3

Magnus Dyre-Moe

Learning-based Collision-free Navigation for Aerial Robots

NTNU
Norwegian University of
Science and Technology