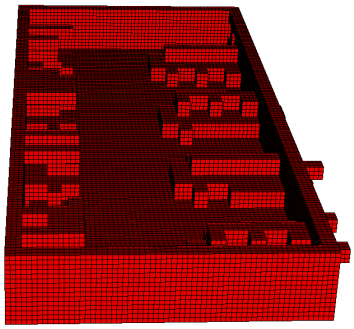
CHALMERS
UNIVERSITY OF TECHNOLOGY



Autonomous Mapping of Unknown Environments Using a UAV

Using Deep Reinforcement Learning to Achieve Collision-Free Navigation and Exploration, Together With SIFT-Based Object Search

Master's thesis in Engineering Mathematics and Computational Science, and Complex Adaptive Systems

ERIK PERSSON, FILIP HEIKKILÄ

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

Autonomous Mapping of Unknown Environments Using a UAV

Using Deep Reinforcement Learning to Achieve Collision-Free Navigation and Exploration, Together With SIFT-Based Object Search

ERIK PERSSON, FILIP HEIKKILÄ

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2020

Autonomous Mapping of Unknown Environments Using a UAV
Using Deep Reinforcement Learning to Achieve Collision-Free Navigation and Exploration, Together With SIFT-Based Object Search
ERIK PERSSON, FILIP HEIKKILÄ

Supervisor: Cristofer Englund, RISE Viktoria
Examiner: Klas Modin, Department of Mathematical Sciences

Master's Thesis 2020
Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Screenshot from the simulated environment together with illustration of the obstacles detected by the system.

Typeset in LaTeX
Gothenburg, Sweden 2020

Autonomous Mapping of Unknown Environments Using a UAV
Using Deep Reinforcement Learning to Achieve Collision-Free Navigation and Exploration, Together With SIFT-Based Object Search
ERIK PERSSON, FILIP HEIKKILÄ
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Automatic object search in a bounded area can be accomplished using camera-carrying autonomous aerial robots. The system requires several functionalities to solve the task in a safe and efficient way, including finding a navigation and exploration strategy, creating a representation of the surrounding environment and detecting objects visually.
Here we create a modular framework and provide solutions to the different subproblems in a simulated environment. The navigation and exploration subproblems are tackled using deep reinforcement learning (DRL). Object and obstacle detection is approached using methods based on the scale-invariant feature transform and the pinhole camera model. Information gathered by the system is used to build a 3D voxel map. We further show that the object detection system is capable of detecting certain target objects with high recall. The DRL approach is able to achieve navigation that avoids collisions to a high degree, but the performance of the exploration policy is suboptimal.
Due to the modular character of the solution further improvements of each subsystems can easily be developed independently.

Keywords: Deep reinforcement learning, autonomous exploration and navigation, feature extraction, object detection, voxel map, UAV, modular framework.

Acknowledgements

We would first like to thank our thesis advisor Klas Modin of the Department of Mathematical Sciences at Chalmers University of Technology. He showed great interest in our work and the door to his office was always open when we needed help.

We would also like to thank the experts at RISE who were involved in the project and supported us throughout: Cristofer Englund, Boris Durán and Martin Torstensson.

Erik Persson, Filip Heikkilä, Gothenburg, June 2020

Contents

1

Introduction

Long-distance transportation of cars and other vehicles is often done using RoRo (roll-on/roll-off) ships. The cargo is densely arranged on the ships in order to save space. It is important to keep track of the location of individual vehicles. Mainly, it should be possible to distinguish between electric and nonelectric vehicles for safety reasons.

The vehicles all have a sticker with an identification code (vehicle identification number - VIN). Currently, the only way to identify the cars is through the VINs, which makes it a demanding task to manually search for and find a particular car. In the case of fire it would be close to impossible to determine which cars are electric [1].

As a possible solution, RISE is interested in investigating whether small autonomous aerial drones can be used to explore the cargo spaces and then detect, localize and identify all vehicles on board. In order to keep costs down, the drones would only carry a single or a few sensors, including a camera, to help them navigate, avoid obstacles and then scan the VINs. The investigation is part of a H2020 project, LASH FIRE [1], coordinated by RISE with more than 20 international partners from both industry and academy.

1.1 Aim

The aim is to develop an autonomous unmanned aerial vehicle (UAV) control system, that is capable of safely exploring unseen environments and detecting specific target objects. The UAV should be able to search the surroundings without causing any collisions and it should cover as much of the area as possible. The results can be used to evaluate the feasibility of a UAV-based approach to solve the problem described above.

1.2 Limitations

To ensure that it is possible to finish the project within the given time frame some aspects must be excluded from the project. First, experiments will be carried out

[1]LASH FIRE website: `https://risefr.com/services/research-and-assessments/lash-fire`

in a simulated environment. It will make development and testing easier, since the risks associated with a flying robot are eliminated. Another benefit is the availability of ground truth information that can be useful when verifying and evaluating the solutions.

Second, virtual environments closely resembling the real cargo decks will not be built. Focus will lie on developing a solution that works in other already available environments. At a later stage the solution can be adapted and applied to more realistic environments.

In order to simplify the problem the target objects will not be restricted to cars or other vehicles. They may be objects of simple shapes and colors that are easily distinguished from the surroundings. There will not be any restrictions on the sensors of the UAV or its computational capacity. The last aspect that will be omitted is a system for approaching the target objects and reading the VINs.

1.3 Issue Specification

With the specified limitations in mind, several requirements on the system can be defined that must be met in order to accomplish the aim.

- The UAV explores an unseen environment and avoids collisions while moving.
- The system builds a representation of the environment, e.g. a map.
- The system identifies obstacles, i.e. areas of the environment where it is not possible to fly, and adds them to the map.
- The system recognizes objects of a specified type, determines their positions in the environment and adds the objects to the map.

1.4 Previous Work

Related challenges have previously been studied in a number of different papers.

Autonomous exploration of unknown environments and simultaneous object search using a UAV is investigated, both in simulated and real-world scenarios, by Dang, Papachristos and Alexis [2]. The environment is assumed to be GPS-denied. For that reason, visual-inertial localization and mapping is used to keep track of the position of the drone. They propose a sampling-based path planning strategy, based on rapidly exploring random trees (RRTs), which maximizes an objective gain that consists of an exploration term and an object detection term. The objects are detected using a YOLO detector, which is a neural network based object detector [3]. The UAV agent builds an occupancy map of its surroundings using i.a. depth information from a stereo camera. The map is used to keep track of detected objects and to create collision-free paths.

In tunnel-like i.e. narrow and multibranched environments Dang, Khattak, Mascarich and Alexis [4] separate the path planning problem into a local and a global

planning problem. The local task consists of exploring the immediate surroundings of the UAV agent, whereas a graph-based global planner is responsible for redirecting the robot to new frontiers, e.g. crossroads with unexplored tunnels, when reaching dead ends.

Wijmans et al. [5] solve a navigation task in an indoor environment using deep reinforcement learning (DRL). After extensive training, they manage to achieve near perfect results, demonstrating the ability to reach a target with no prior knowledge of the environment, using RGB-D (RGB and depth) images and compass directions as input information. Furthermore, they show that similar tasks, such as exploration or flight (see [6]), can be solved efficiently using transfer learning from a trained navigation agent.

UAV navigation in heterogeneous outdoor environments with varying weather conditions is investigated by Maciel-Pearson et al. [7]. Further, they break down the global navigation problem into fixed size local tasks. Instead of trying to fly directly to the target, they set a sequence of local waypoints to reach. A DRL agent controlling a simulated UAV is trained in the flight simulator AirSim. While exploring its surroundings the UAV marks the positions of detected obstacles on a map. The map, together with an RGB image from the UAV point of view, serve as input to the agent. The agent is trained with different versions of the Deep Q-Network (DQN) algorithm and they demonstrate that the agent is able to solve the task in different environments and weather conditions [7, 8].

In a similar fashion Walker, Vanegas, Gonzalez and Koenig also make a distinction between a global and a local planning task when tackling the problem of searching an indoor environment using a UAV, simulated in Gazebo [9]. The environment is divided into subsections and the global planner can take high level actions, deciding which subsection to search. After a decision has been made, the local planner takes over and controls heading and velocity of the UAV. The modules are trained separately using a DRL algorithm called Trust Region Policy Optimization (TRPO) [10].

Madaan, Saxena, Bonatti, Mukherjee and Scherer [11] look into UAV collision avoidance in a monocular camera scenario. The environment is a forest of column shaped trees simulated in Gazebo. The task of the agent is to output yaw rate commands to avoid collisions while flying at constant speed and altitude through the forest. The agent is trained with DQN, with a stack of four consecutive images as input. They show that their trained policy performs significantly better than the initial random policy.

2

Theory

This chapter introduces important concepts relevant to the thesis. Section 2.1 covers necessary background to understand the deep reinforcement learning algorithms, on which the local navigation and global planning solutions rely. Section 2.2 describes the SIFT algorithm that is used for object detection. Lastly, concepts from computer vision such as a camera model, that describes how 3D points are projected to the image plane, are explained in Section 2.3.

2.1 Reinforcement Learning

Reinforcement learning (RL) is an approach to solve planning and decision-making problems, which can be summarized as the process of repeatedly choosing between several *actions* in some setting that is affected by those choices. Formally the decision maker is called the *agent* and the abstract space that the agent interacts with is called the *environment*. When the agent performs an action it will receive a signal, usually called a *reward*, that indicates whether the action was good or bad. The content of Sections 2.1.1-2.1.3 is largely based on [12, 13], which are recommended for the interested reader.

2.1.1 Markov Decision Processes

RL builds on the mathematical framework provided by Markov decision processes (MDPs). An MDP is a model of sequential decision-making. It is a discrete time stochastic process defined by a state space $\mathcal{S}$, a set of actions $\mathcal{A}$, a transition function p and a reward function r. At each time step t the environment is represented by a state $s_t \in \mathcal{S}$. The agent can interact with the environment by taking an action $a_t \in \mathcal{A}$. The environment is updated according to the dynamics described by the probability distribution $p(s') = P(s_{t+1} = s' \mid s_t, a_t)\ \forall s' \in \mathcal{S}$ resulting in a new state s_{t+1} and a reward $r_{t+1} = r(s_t, s_{t+1}, a_t)$. The Markov property must hold for the transition function — the next state may only depend on the previous state-action pair, or $P(s_{t+1} \mid s_t, s_{t-1}, ..., s_0, a_t, a_{t-1}, ..., a_0) = P(s_{t+1} \mid s_t, a_t)$.

The definition of MDPs can be generalized to include cases where the true state s is not always known. Let an observation be a representation of a state. In an MDP the observation is equivalent to the true state. If the observation does not contain all the information about the true state, the environment is said to be partially observable.

An MDP where this is the case is called a partially observable MDP (POMDP). A POMDP is characterized by an observation space Ω and a conditional probability function O, in addition to the underlying MDP. Every time step, the next observation ω_{t+1} is governed by $O(\omega') = P(\omega_{t+1} = \omega' \mid s_{t+1}, a_t)\ \forall \omega' \in \Omega$. Consequently, the agent has to make decisions based on uncertain information.

Let the belief $b(s \mid \omega)$ be the probability of s being the true state when observing ω. By interpreting b as the the state instead of s, along with other modifications, POMDPs can be formulated as MDPs with infinite state space. Therefore, mainly MDPs will be considered throughout this section.

The implicit goal regarding problems formulated as MDPs is to collect rewards. Let the *return* $R_t = R_t((r_k)_{k=t+1}^{T})$ be a function of all future rewards, where T is the final time step, possibly $T = \infty$. The return can be defined as a discounted sum

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+1+k}, \tag{2.1}$$

where $\gamma \in [0, 1)$ is the discount factor. Due to this definition, distant rewards are considered less valuable and convergence of the infinite sum is ensured (assuming that the sequence of rewards is bounded).

2.1.2 Policy and Value Functions

The behavior of an agent is called a *policy*, which can be thought of as a mapping $\pi : \mathcal{S} \to \mathcal{A}$ that chooses an action depending on the state. A probabilistic policy can be defined as a probability distribution over all actions given a state,

$$\pi(s_t, a_t) = P(a_t \mid s_t), \quad s_t \in \mathcal{S},\ \forall a_t \in \mathcal{A}, \tag{2.2}$$

from which actions can be sampled. RL can now be defined as finding a policy which maximizes the expected return $\mathbb{E}[R_t]$.

Assuming that an agent is following a policy, a value function can be introduced, which describes how good, or valuable, a certain state is.

$$V_\pi(s_t) = \mathbb{E}_\pi[R_t \mid s_t] \tag{2.3}$$

is called the *state-value function* and it is the expected discounted sum of future rewards if the agent follows policy π starting from state s_t. Furthermore, the *action-value function*

$$Q_\pi(s_t, a_t) = \mathbb{E}_\pi[R_t \mid s_t, a_t], \tag{2.4}$$

is defined as the expected return of taking action a_t in state s_t and from there on following policy π. The difference between the action value and the state value is called *advantage*,

$$A(s_t, a_t) = Q_\pi(s_t, a_t) - V_\pi(s_t), \tag{2.5}$$

and it describes the value of an action in relation to the expected value of the current state. This definition can be useful e.g. in a poor state where every action will result

in a large negative reward. By subtracting the negative expectation, some actions may lead to a positive advantage and it becomes more clear that an action can be favorable despite resulting in a negative reward.

If either of the value functions can be calculated easily, the optimal policy π^* from state s_t can be found e.g. by

$$\pi^* = \arg\max_{\pi} V_\pi(s_t). \tag{2.6}$$

There are methods of finding the exact action-value function [14], but since calculations have to be done for the whole state-action space they are only suitable for small problems. Solving environments with large state and/or action spaces require other methods.

2.1.3 Deep Reinforcement Learning

A popular approach to deal with complex environments is to search for approximate solutions instead of finding the exact policy or value functions [15]. A class of function approximators that work well with high dimensional data is deep neural networks (NN) . The use of NN in RL methods is referred to as deep reinforcement learning (DRL). DRL agents are often trained by first interacting with the environment, following a stochastic policy and collecting minibatches of experience. The data is then used to update the NN and the process repeats.

There are several different types of DRL algorithms i.a. policy gradient methods which primarily focus on optimizing a parameterized policy function using some form of gradient descent [13]. Actor-critic methods are a form of policy gradient methods that also learn a value function [16, 17]. In an actor-critic setting, the two functions π_θ and V_θ are the parameterized policy and value functions, which are referred to as the actor and the critic. The two functions can be expressed by a single multiheaded NN, using the shared parameters θ.

2.1.4 Proximal Policy Optimization

One of the shortcomings of policy gradient methods is a bad sample efficiency. Proximal Policy Optimization (PPO) [18] improves the actor-critic algorithm by allowing multiple training updates on the same minibatch of experience. It also uses ideas from Trust Region Policy Optimization (TRPO) [10] to improve the robustness of the algorithm. The algorithm has proven to work on a wide variety of tasks, with little hyperparameter tuning.

The main idea with PPO is to avoid too large updates of the parameterized policy, known as the actor, which can lead to performance drops [19]. For this reason a probability ratio is introduced,

$$\phi_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_\theta^{\text{old}}(a_t|s_t)}, \tag{2.7}$$

which is a measure of how much a new policy differs from the previous (old) one. To ensure conservative policy updates a special policy objective function, see Figure

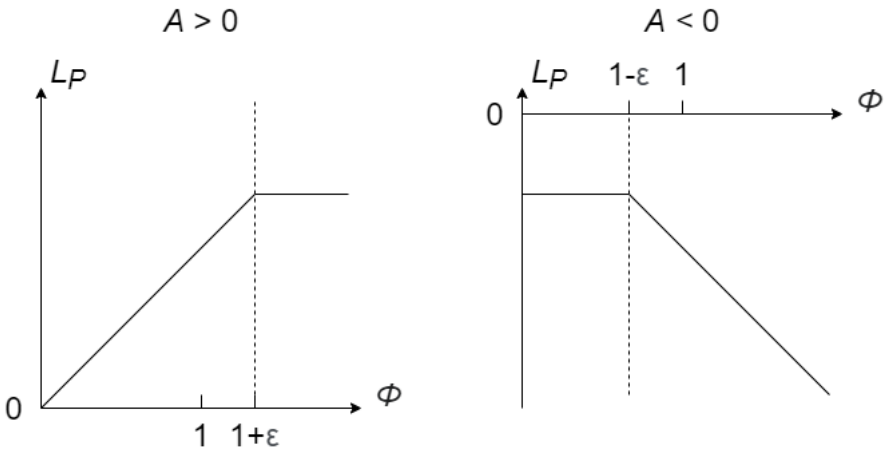


Figure 2.1: Illustration of the policy loss as a function of the ratio $\phi_t(\theta)$ for positive advantage (left) and negative advantage (right).

2.1, is defined

$$L_P(\theta) = \hat{\mathbb{E}}_t \left[\min \left(\phi_t(\theta)\hat{A}_t,\ \text{clip}[\phi_t(\theta), 1-\epsilon, 1+\epsilon]\hat{A}_t \right) \right], \tag{2.8}$$

where ϵ is a small parameter and $\hat{A}_t$ is an estimate of the advantage, $\hat{A}_t \approx R_t - V_\theta(s_t)$.

The partial derivative $\frac{\partial L_p}{\partial \phi_t}$ will be zero whenever the objective value is clipped, and due to the chain rule the same is true for the gradient $\nabla_\theta L_P$. Consequently, the parameters of the NN will remain unchanged during the backpropagation update, with respect to the policy loss. As a result, training can be repeated multiple times each epoch using the same experience — if the new policy starts to deviate from the old one, updates that otherwise would push the policy too far in some direction will have no effect.

In order to also train the critic part of the agent a value loss is defined as the mean squared error (MSE) of the predicted and the observed state-values

$$L_V(\theta) = \left(V_\theta(s_t) - V(s_t) \right)^2. \tag{2.9}$$

An entropy term, which for a categorical output distribution looks like

$$L_S(\theta) = -\sum_{a \in \mathbf{A}} \pi_\theta(a_t|s_t) \log(\pi_\theta(a_t|s_t)), \tag{2.10}$$

is added in order to discourage exploitation of suboptimal policies, thereby avoiding premature convergence [17].

The NN can then be trained using stochastic gradient descent (SGD) with the total objective function

$$L(\theta) = -L_P(\theta) + \alpha L_V(\theta) - \beta L_S(\theta), \tag{2.11}$$

where α and β are hyperparameters.

2.2 Scale-Invariant Feature Transform

Lowe [20] introduced Scale-invariant feature transform (SIFT) as an algorithm to extract distinctive features from images. He also proposed that those features could be used for object recognition tasks. The advantage of SIFT is that the features are invariant to scaling and rotation and robust to other transformations: such as change of viewpoint, illumination and noise. In a comparison SIFT is showed to outperform other feature extraction algorithms, but some of them run faster [21].

SIFT works in multiple steps to extract the features and the algorithm can be divided in four main stages, described below.

2.2.1 Scale-Space Extrema Detection

In order to find potential keypoints the first step is to determine locations that can be identified in different views of the same object, which could mean that the relative size of the object varies [20]. Locations that are invariant to scale changes can be found by searching for stable features across all possible scales. This is done by analyzing the scale space — a continuous function L that applies a smoothing kernel to the image [22]. Under certain conditions it can be shown that the Gaussian kernel is the only suitable kernel [23]. For SIFT the scale space function is defined as:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y), \tag{2.12}$$

where $G(x, y, \sigma)$ is a Gaussian of scale σ and $I(x, y)$ is the input image [20]. Specifically the Gaussian is defined as

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/(2\sigma^2)}. \tag{2.13}$$

It can be noted that this is related to the heat equation where L can be seen as its solution. This means that the original image can be interpreted as the initial temperature distribution and the smoothed images of increasing scale as the result of heat diffusion over time.

To find the locations of keypoints that are stable in scale space, the extrema of a difference-of-Gaussian (DoG) function $D(x, y, \sigma)$ are calculated. $D(x, y, \sigma)$ is defined as the difference between two scales separated by a factor k:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma). \tag{2.14}$$

In the original SIFT paper it is shown that the DoG is a good approximator of the Laplacian of Gaussian (LoG) [20]. The maxima and minima of the scale-normalized LoG has in turn been shown to produce the most stable image features [24].

DoGs are created as illustrated in Figure 2.2. Each doubling of the scale σ is called an octave and each octave is divided into s blurred images, resulting in $k = 2^{1/s}$. The input image is convoluted with the Gaussian kernel repeatedly to produce each

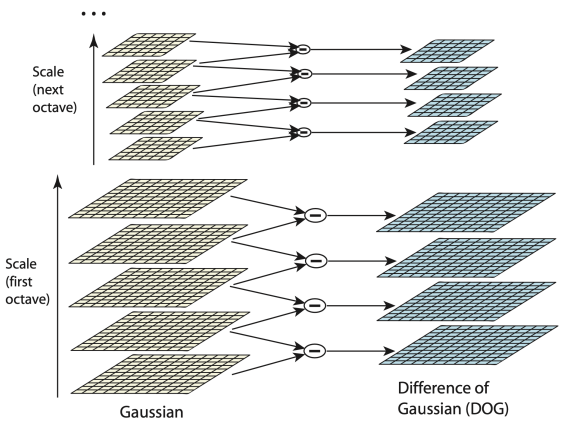


Figure 2.2: Gaussian blur of increasing scale is applied to the image. After each octave the image is down-sampled. DoGs are created by subtracting adjacent blurred images. From [20]. Reproduced with permission.

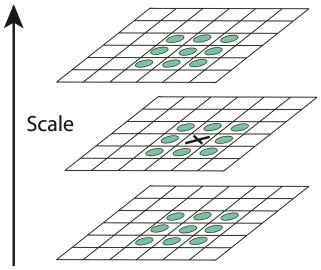


Figure 2.3: DoG extremas are found by comparing the sample point, marked as X, with its neighbors in a 3x3 area at the current and adjacent scales. From [20]. Reproduced with permission.

blurred image in the octave. When an octave has been finished, the final blurred image is down-sampled by a factor two and the process is repeated for the next octave. Adjacent images are then subtracted to form a so called DoG pyramid.

The local extrema of the DoGs in the pyramid is found by comparing each sample point to its eight immediate neighbors in the same scale as well as the nine neighbors in the scale above and the nine below, illustrated in Figure 2.3. The sample point is selected if it is smaller than all or larger than all neighbors it is compared to.

2.2.2 Keypoint Localization

After potential keypoints have been found Taylor expansions are used to obtain more accurate estimates of the locations of the keypoints. The Taylor expansion is done on the DoG function in Equation (2.14) up to the quadratic term and then shifted so that the potential keypoint is at the origin:

$$D(\mathbf{x}) = D + \frac{\partial D}{\partial \mathbf{x}}^T \mathbf{x} + \frac{1}{2}\mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}, \tag{2.15}$$

where D is evaluated at the sample point and $\mathbf{x}$ is the offset from that point. The position of the keypoint, $\hat{\mathbf{x}}$, is calculated by taking the derivative of the function in

Equation (2.15) and equating it to zero, which should give a stationary point of the function.

$$\hat{\mathbf{x}} = -\left(\frac{\partial^2 D}{\partial \mathbf{x}^2}\right)^{-1} \frac{\partial D}{\partial \mathbf{x}}. \tag{2.16}$$

It can be noted that this is the same thing as taking one step of Newton's method for the function $D'(\mathbf{x})$.

Once the position has been calculated it is also used to reject suggested keypoints originating from unstable extrema of low contrast. Specifically the rejection is performed by substituting Equation (2.16) into Equation (2.15) which gives:

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2}\frac{\partial D^T}{\partial \mathbf{x}}\,\hat{\mathbf{x}}. \tag{2.17}$$

If the magnitude of this function is smaller than a threshold the corresponding keypoint is rejected.

It is also a problem that the DoG-function will give large responses for points along edges, even when their locations are poorly known. To combat this, SIFT includes measures aimed at eliminating these responses [20].

2.2.3 Orientation Assignment

In order to achieve invariance to image rotations each keypoint is assigned an orientation based on the local image area. Computation of local orientation is performed on the smoothed image, L, that is closest in scale to where the keypoint was found. This is done to ensure scale invariance of the assigned orientation. The gradient magnitude, $m(x, y)$, and orientation, $\theta(x, y)$, are computed for each pixel in L according to:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}, \tag{2.18}$$

$$\theta(x,y) = \arctan\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right). \tag{2.19}$$

For each keypoint an orientation histogram is created for a neighborhood around that point. Each sample is weighted with its gradient magnitude multiplied with a Gaussian window and added to one of 36 bins, covering a total of 360°. The bin with the highest magnitude is selected as the orientation of the keypoint. If there exist additional peaks with magnitudes of at least 80 % of the dominant peak, additional keypoints with these orientations are also created.

2.2.4 Keypoint Descriptor

The last step of the SIFT algorithm is to create a descriptor of the local region around each keypoint. The descriptor is constructed by sampling the gradient magnitudes and orientations in a 16x16 neighborhood around the keypoint. The magnitudes are multiplied with a Gaussian window to give more importance to the gradients

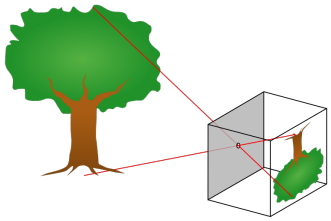


Figure 2.4: Diagram of a pinhole camera. The light enters through a single point and the object is projected upside down at the back of the box. From [26]. CC0.

close to the keypoint. To ensure orientation invariance, the coordinate system of the neighborhood is aligned with the orientation of the keypoint.

The gradients are then used to create orientation histograms, with eight bins, over 4x4 sample regions. This allows for positional shift of the gradients, which should make it more robust to viewpoint changes and non-rigid deformations.

The orientation bins from all sample regions are arranged as a vector which forms the descriptor. Finally, a few additional tricks are done to make the descriptor robust against illumination changes.

2.2.5 Keypoint Matching

After keypoints have been identified in an image they can be matched against a database of keypoints from one or multiple reference images. The best potential match is the reference descriptor with the closest Euclidean distance – the nearest neighbor.

Many keypoints will originate from background clutter or will not have been detected in the reference image. For this reason a method for rejecting bad matches is needed. A suitable metric for the quality of a match is the ratio between the distance to the nearest and the second nearest neighbor. This is motivated by the notion that for false matches it is likely that there will exist multiple other matches at a similar distance, given the high dimensionality of the descriptor, which would put the ratio close to 1. By rejecting matches with a ratio above 0.8 it is claimed that 90 % of the false matches, but only 5 % of the correct ones, are rejected [20].

2.3 Camera Model

A pinhole camera, illustrated in Figure 2.4, is a closed box with a single point, the aperture, where light can enter to hit a planar surface called the image plane. The pinhole camera model describes the projection of 3D points to the image plane in a pinhole camera. Even though the pinhole camera model omits many aspects of a true camera, such as finite size of aperture and radial distortion caused by lenses, the model is still suitable for many cameras and applications [25].

Figure 2.5 shows the geometry of the pinhole camera model. To set up the algebraic relations for the model a few notations must be defined. The focal length, f, is

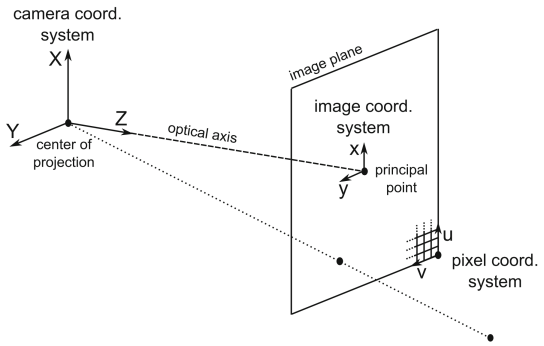


Figure 2.5: Pinhole camera model. From [25]. Reproduced with permission.

the distance between the pinhole and image plane. The *optical axis* is the line that is orthogonal to the image plane and passes through the pinhole. The point where the optical axis intersects the image plane is called the *principal point*. The digital image is expressed in its own coordinate system, the pixel coordinate system. Relative to the principal point, the origin of the pixel coordinate system is translated by $(-x_0, -y_0)$ corresponding to the corner of the image. [25]. From the geometry of the figure it is possible to derive expressions for where a 3D point will be projected in the image plane. For a 3D point with coordinates (X, Y, Z) the point in the image plane is

$$x = f\frac{X}{Z}, \quad y = f\frac{Y}{Z}. \tag{2.20}$$

To shift it to the pixel coordinate system translation and scaling must be applied:

$$u = k_u(x + x_0) = k_u\left(f\frac{X}{Z} + x_0\right), \tag{2.21}$$

$$v = k_v\left(f\frac{Y}{Z} + y_0\right), \tag{2.22}$$

where k_u and k_v are the pixel densities in the u and v directions, respectively. From these equations it is easy to obtain expressions for how 3D points are reconstructed from the image, provided that the depth, Z, is known.

3

Methods

To meet the overall aim of the project and the issue specification, presented in Sections 1.1 and 1.3, the control system must solve multiple sub-problems. Each sub-problem was solved individually and the modular solutions (modules) were then combined. A benefit of a modular approach is that the individual modules can easily be modified or replaced with better solutions in the future. The project includes the implementation of five different modules. A local navigation module controls the flight of the UAV and steers toward a target position, while avoiding collisions with obstacles along the way. Two detection systems are responsible for detecting obstacles and identifying target objects and calculating their positions. The obtained information is stored in an internal map. Lastly, a global planning module responsible for exploring the surroundings of the UAV guides the local navigation system by determining its waypoints. An overview of the full system and its components is presented in Figure 3.1.

The division of a local navigation and a global planning module is in line with previous work [4, 9] and is similar to thoughts expressed in [27], where the problem solving process of a car driver is divided into three different task levels; a strategical, a tactical and an operational. In this setting the strategical planning is performed by the global navigation module. The tactical problems, such as moving between two points are solved by the local navigation module. Lastly, low-level operational tasks, such as position control, are handled automatically by the UAV flight simulator.

Two modules necessary for having a fully working autonomous system in a more realistic setting are missing from the scope; a localization system that determines the position and orientation of the UAV in a GPS-denied environment, and a solution for approaching the objects of interest and scanning their identification codes.

3.1 UAV Simulation

In this project the flight simulator AirSim was used to simulate the UAV and its sensors. AirSim is built on Unreal Engine and gives the ability to control a UAV in environments that are physically and visually realistic [28]. Sensors such as cameras, GPS, IMUs, LIDARS are supported. AirSim comes with a Python API that can be used to interact with the simulator, both to control the flight of the UAV and to obtain sensor readings and ground truth information, such as position and velocity of the drone.

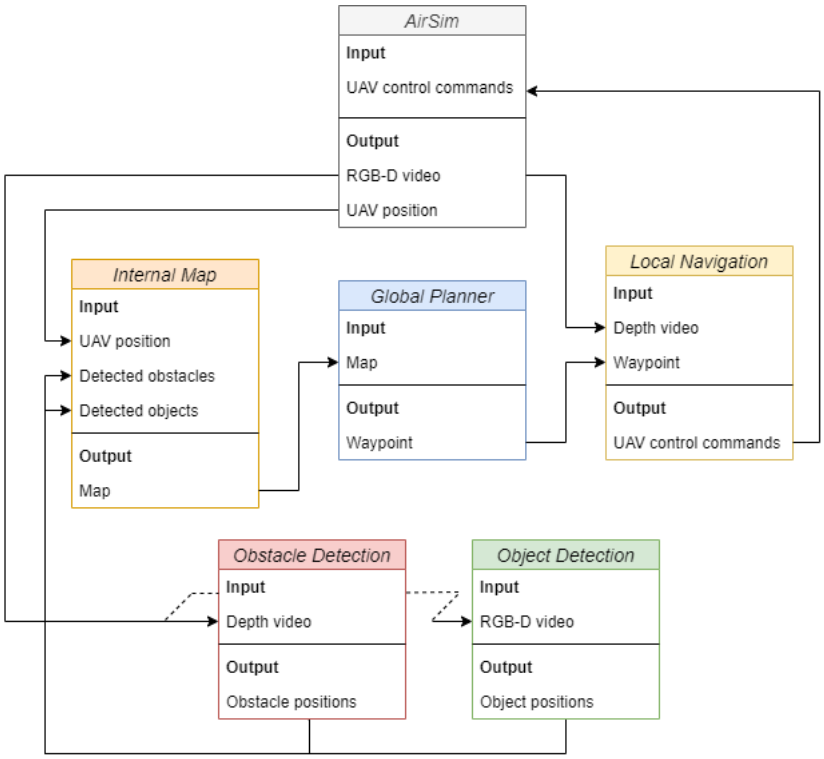


Figure 3.1: Schematic overview of the proposed UAV control system, which shows the different modules and how they are connected.

Two simulated sensors, an RGB camera and a depth camera, were used to provide state information of the environment. In addition, the position and the orientation of the UAV were obtained from AirSim, which can be compared to having GPS and IMU units.

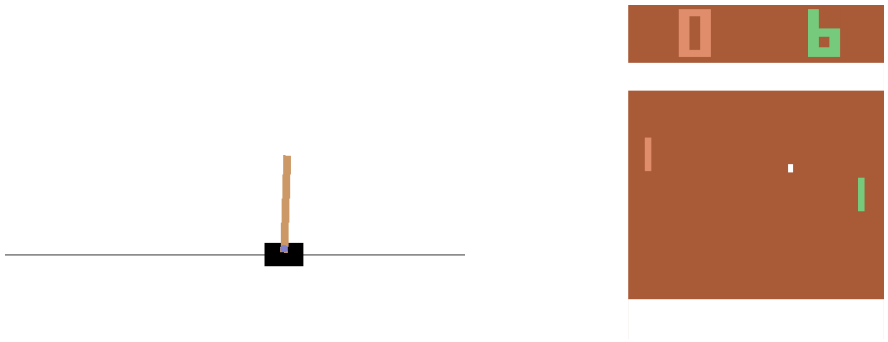
AirSim can be run in different scenes that can be created using the Unreal Editor. There are a few ready-made scenes included, for instance a city scene and a suburban neighborhood, that are uneditable. As a result, their use for this project was limited. Instead, a model based on the RISE office at Lindholmen was used as the main scene, see Figure 3.2. The scene, called *Viktoria*, provides an indoor setting with multiple obstacles and the ability to add new objects when needed.

3.2 Deep Reinforcement Learning Algorithm

The local navigation and global planning problems were modeled as MDPs and approached using DRL. The NNs were trained using PPO. The same implementation was used in both cases, meaning that it had to be environment agnostic. Inspiration for the implementation was mainly drawn from the paper presenting PPO [18], as well as [5, 29].

The solution was implemented in Python using the Pytorch library and it uses mini-batch gradient descent with the Adam optimizer[30], entropy regularization [5] and early stopping based on KL-divergence [29].

Figure 3.2: Screenshot of the UAV simulation in AirSim, set in the *Viktoria* scene.



(a) **(b)**

Figure 3.3: Screenshots of two RL environments: **(a)** *CartPole*. From [31]. MIT License. **(b)** *Pong*.

3.2.1 OpenAI Gym

OpenAI Gym [31] is a toolkit for training and benchmarking DRL algorithms. It contains a library of environments of different difficulty with a standardized interface. Two examples are the *CartPole* and *Pong* environments, see Figure 3.3. The task in *CartPole* is to keep an inverted pendulum balanced by pushing a cart that is supporting its base left or right. The observations from the environment consist of four values: cart position, cart velocity, pole angle and velocity of the tip of the pole. The *Pong* environment is an old Atari game where the goal is to beat the computer player in a game of pong. The paddle/racket can be moved up and down. The observations comprise the 210×160 pixel values representing the RGB image that would be used to show the game on a screen.

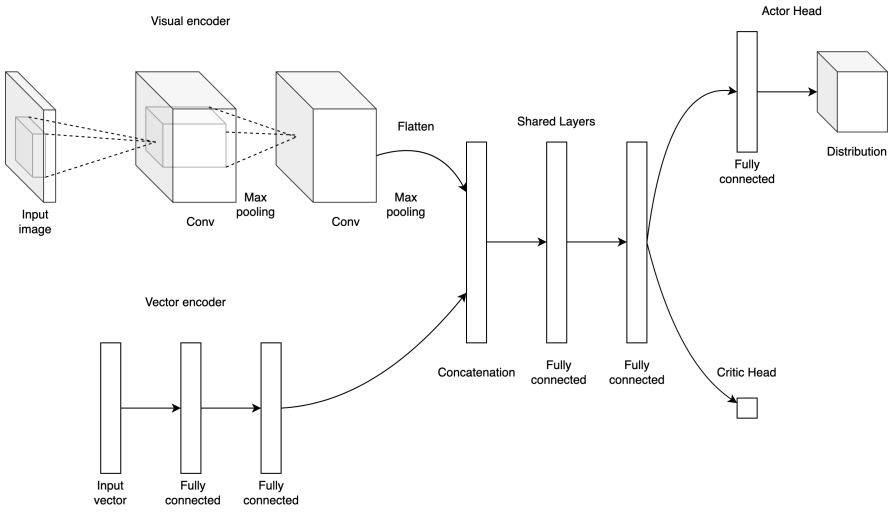


Figure 3.4: Overview of the general neural network structure used for DRL.

3.2.2 Neural Network Structure

In order to avoid having to implement new NN agents for each environment a single general design was used. This implementation allows for controlling input and output shapes, number of hidden layers etc. using a parameter file instead of modifying the code. An overview of the general design is presented in Figure 3.4. The network structure starts with two encoders, a vector encoder for 1D inputs and a visual encoder for image inputs. The vector encoder only consists of fully connected layers, while the visual encoder is a stack of convolutional layers. Each convolutional layer is optionally followed by a max pooling layer. The encoders run in parallel and their outputs are then concatenated. Inspiration for using encoders in this fashion comes from [5]. After concatenation, the output is passed through a number of shared fully connected hidden layers. The network is then split into two heads, an actor head that predicts the next action and a critic head that estimates the value of the current state. The critic head is simply a single neuron, without activation function, that outputs the estimate.

The actor head can be of two types, a categorical actor or a continuous actor depending on the the type of action space in the environment. The categorical actor interprets the last hidden layer as the logarithmic probabilities of each discrete action and creates a categorical distribution based on these values. The continuous actor was designed in a similar fashion, however the last hidden layer instead predicts the elements of the mean vector and the diagonal elements of the covariance matrix of a multivariate normal distribution, similar to [17].

3.2.3 Performance Verification

To verify the correctness of the PPO implementation, training was first done on the *CartPole* and *Pong* environments, as *CartPole* is a simple problem that does not require extensive training and since Atari games are often used for benchmarking DRL algorithms [15, 17, 18]. For the *CartPole* experiment the vector encoder

was used whereas the visual encoder was used for *Pong*. In both cases the action space was discrete and therefore the categorical actor head was used. The exact hyperparameters can be found in Appendix A.

3.3 Local Navigation

The module responsible for controlling the UAV is called local navigation. More specifically, the functional requirements of the system are to safely steer the UAV to a waypoint, avoiding collisions with any obstacles on the way and to terminate execution either when the waypoint has been reached or if the waypoint is deemed unreachable. The local navigation module has access to a video feed from the depth camera of the UAV, the position and orientation of the UAV and the position of the waypoint. The positional information is obtained directly from AirSim.

One possible approach to solving the navigation problem is to use RRT-based path planning as in [2, 4]. However, this approach is dependent on information about the surroundings in the form of a 3D map, the correctness of which could be hard to guarantee. Instead the system was built using a DRL approach, similar to how a navigation task was solved in [5]. This approach should be more robust and straightforward since it is not using a generated map of the environment, but rather the direct input from the depth sensor.

Despite the drone being able to move in three dimensions, the waypoints were defined using two-dimensional coordinates (x, y). This is motivated by the fact that the distance between the floor and the ceiling in the *Viktoria* scene is limited, and the camera on the UAV has a good overview regardless of the height. The z-value of the waypoint can therefore be considered arbitrary.

3.3.1 Reinforcement Learning Environment

A wrapper class that conforms to the OpenAI Gym framework was created on top of the AirSim Python API. After opening the *Viktoria* scene, the wrapper class spawns the agent (the UAV) at a random obstacle-free location in the scene. A waypoint is also generated somewhere in the scene. Observations consisting of a depth map of the environment, and the distance and direction to the waypoint, are provided to the agent. The agent can perform six different actions: fly forward 0.25 m, rotate 10° left or right, ascend or descend by 0.25 m and terminate the trajectory.

During movement, AirSim checks for collisions. If a collision with an obstacle in the scene occurs, the environment will reset. When the agent terminates, the waypoint is replaced by a new one. 90 % of the generated waypoints are valid, meaning that they can be reached by the agent. The rest will be invalid, located e.g. inside obstacles or beyond walls. The invalid waypoints were added in order to create a robust system, that is capable of collision-free flight independent of the quality of the waypoints.

The intended goal of the agent is to reach waypoints, safely and efficiently. The agent should be able to terminate correctly — at the location of a valid waypoint,

Table 3.1: The reward function in the local navigation RL environment.

Event/action	Reward
Correct termination	+5
False termination	-2
Collision	-5
Move towards waypoint	+0.1
Move away from waypoint	-0.1
Rotate or ascend/descend	-0.05

Table 3.2: Network layout of local navigation agent.

Visual Encoder		**Vector Encoder**	
Filters (per layer)	[16, 32, 64]	Hidden layers	[32, 32]
Kernel size	3-by-3		
Shared layers		**Categorical Head**	
Hidden layers	[32, 32]	Hidden layers	[32]

or when it is confident that a waypoint is invalid. Colliding with an obstacle is penalized. This is reflected in the reward function, which is presented in Table 3.1.

In order to speed up training, a small reward or penalty was added whenever the agent moves towards or away from the waypoint, respectively, with the argument that the agent is more likely to discover how to terminate correctly if it first learns to approach the waypoints. Finally, a penalty was added whenever the agent rotates, since early tests produced agents that only were capable of turning in one direction.

3.3.2 Neural Network Agent

The NN agent for local navigation was based on the design described in Subsection 3.2.2. The number of layers and their sizes were chosen to resemble previous solutions to tasks of similar nature [7, 15]. For this module both encoders were used, since the input consists of both the depth map and distance and angle to the target. Due to a discrete action space the categorical head was used. More details are provided in Table 3.2. The NN agent was trained using PPO, with the hyperparameters seen in Table 3.3.

The reward function was designed to encourage a specific behavior. The DRL algorithm will however only seek to maximize the expected return. There could exist policies that fail to achieve the intended goals, while still generating high rewards [1]. In addition to the return, other metrics were therefore used to evaluate the agents; the number of collisions, the number of terminations and the fraction of correct/incorrect terminations.

[1]The phenomenon is referred to as *reward hacking*.

Table 3.3: PPO hyperparameters used in local navigation experiments.

Parameter	Value
Discount (γ)	0.99
GAE parameter (λ)	0.95
Clipping parameter (ϵ)	0.2
Learning rate	1e-5
Epochs	2700
Steps per epoch	2048
Minibatch size	128
Backprop. repetitions	8
KL divergence threshold	0.03
Value coefficient (α)	0.5
Entropy coefficient (β)	0.01

3.4 Internal Map

A 3D voxel map forming an *x-y-z* grid was built for several reasons. It was mainly needed to serve as input to the global planning module, but it can also be useful during evaluation of other modules, since it helps visualizing the movements and detections of the agent. Similar approaches have been done in [2, 4, 7]

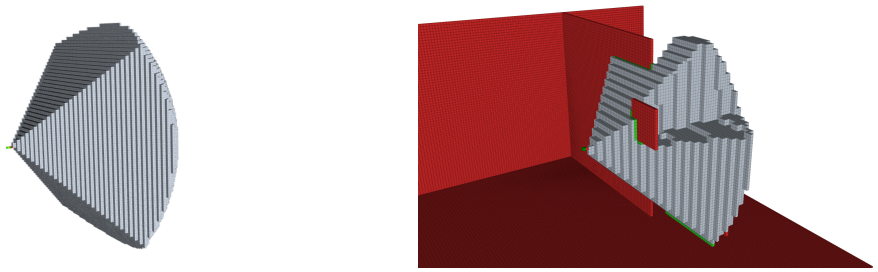
The size of the map can be increased during runtime, since the size of an unexplored environment is likely unknown. Functionality was added to be able to retrieve a fixed size local map, in order to ensure compatibility with the global planner, which requires an input with predefined size.

Each cell of the map can represent one of several states:

- *unknown* - the cell has not yet been detected,
- *free* - the cell has been detected and is empty,
- *visited* - the cell has been visited by the agent,
- *occupied* - the cell is occupied and contains an obstacle,
- *target* - the cell contains a target object,
- *agent* - the cell contains the UAV.

The map supports addition of new information, such as changing the cell states from *unknown* to *free*. Positional information regarding the agent is obtained from the AirSim environment and different detections are reported from the obstacle and object detection systems.

In order to track the cells that have been observed, the vision of the agent was modeled. Given the position and orientation of the agent and the field of view of its camera, a pyramid-shaped volume can be calculated, see figure 3.5a. A manually defined vision range was used to limit the size of the volume. The cells within the modeled vision volume can then be marked as *free*, if they were previously *unknown*.



(a) **(b)**

Figure 3.5: Model of the vision of the agent in empty surroundings (a) and with obstacles present (b).

Since obstacles, including the ceiling and walls, are opaque, the vision model was adjusted to not detect cells within the vision volume if there exists an obstacle in between the cell and the agent, see Figure 3.5b.

There is no guarantee that the information added to the map is at all times correct. To reduce the effect of noise, thresholds were introduced so that multiple detections are required before a cell is updated.

3.5 Global Planning

The global planning system is focused on the exploration aspects of the project. The module is responsible for guiding the trained local navigation system by generating its waypoints. The internal map is used as input to highlight detected obstacles and previously visited or observed locations and areas. The core problem is to develop a waypoint generating policy, which maximizes the exploration of a scene - defined as the fraction of cells that have been observed. Similar to the local navigation solution, the global planning was implemented using DRL, inspired by [5, 17].

3.5.1 Reinforcement Learning Environment

The internal map system was used as a base to build an RL environment suitable for an exploration task. The observation provided to the global planning agent is (a part of) the current map of the scene. An action is then defined as providing the next waypoint, described by a direction $(dx, dy) \in \mathbb{R}^2$ from the current position of the UAV. The RL environment requires a local navigation agent that is responsible for moving the UAV between its current position and the generated waypoint. Two versions were implemented; first a naive navigator which moves straight towards the target was used, then the AirSim wrapper and a navigation policy trained in the *Viktoria* scene were integrated with the RL environment. The naive navigator was implemented in order to investigate how an RL approach would perform on the exploration task. Several maze-like toy maps were created for training purposes, see Figure 3.6. When exploring these maps the obstacle detection module was not used, for two reasons: first, no visual simulation of them were available. Secondly, since the maps were created manually, ground truth information about the obstacles

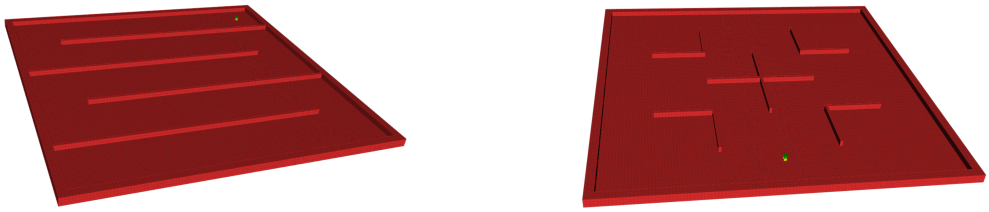


Figure 3.6: Examples of maze scenes used as training maps for the global planning agent.

Table 3.4: Neural network architecture of the global planning agent.

Visual Encoder		**Vector Encoder**	
Filters (per layer)	[16, 32, 64]	———	——
Kernel size	3-by-3		
Shared layers		**Continuous Head**	
Hidden layers	[32, 32]	Hidden layers	[32]

were available. When using these maps, obstacle information about the immediate surrounding of the UAV, as it moved around, was instead provided directly.

Since the navigation agent used in the first approach is unable to avoid obstacles, the global planner would presumably have to micromanage the local navigation rather than acting on a higher level. The RL environment that was combined with the *Viktoria* scene therefore utilizes a pre-trained navigation policy (described in Section 3.3) to reach the waypoints. After receiving an action (waypoint) from the global planning agent, the navigation policy controls the drone in the scene until it terminates. During execution, the internal map is updated continuously, recording e.g. obstacle detections.

To develop a behavior of the agent that explores a scene, the reward function depends on the detection of previously unseen cells in the internal map. The reward for each action (waypoint) is equal to the number of *unknown* cells that become detected, inversely scaled with the size of the vision volume and the number of navigation steps taken to reach the waypoint. If the local navigation policy collides with an obstacle, the reward is set to -10.

3.5.2 Neural Network Agent

The general neural network design established in Section 3.2.2 was used as a starting point. The spatial nature of the internal map serving as the input motivated the use of a visual encoder. In order to not lose any important information, downsampling in the convolutional stack was avoided, meaning max pooling was not used. Due to the continuous action space, a continuous action head was chosen. The NN architecture is summarized in Table 3.4.

The global planning agent was trained using similar parameters in both configura-

Table 3.5: PPO hyperparameters used in the two global planning experiments: trained on toy maps using a naive local navigator and trained in *Viktoria* with a pre-trained local navigation policy.

Parameter	**Naive**	**Viktoria**
Discount (γ)	0.99	0.99
GAE parameter (λ)	0.95	0.95
Clipping parameter (ϵ)	0.2	0.2
Learning rate	2e-6	1e-8
Epochs	2200	600
Steps per epoch	512	512
Minibatch size	64	128
Backprop. repetitions	16	8
KL divergence threshold	Not used	0.1
Value coefficient (α)	0.5	0.05
Entropy coefficient (β)	0.01	0.5

tions, which are presented in Table 3.5. The performance of the agent was measured by the number of detected cells as a function of the number of actions taken (number of waypoints) and the results were compared to a random walk baseline. In the case of the naive navigator, that lacks collision avoidance capabilities, the waypoint was resampled if it was not reachable in order to make it comparable to a trained policy.

3.6 Obstacle Detection

The obstacle detection module was designed to find obstacles such as walls, the ceiling or other large objects, with which the UAV could collide. The solution resembles the approach in [32].

First, the algorithm checks the depth at a number of predefined gridpoints in the depth map. The 3D coordinates for all points smaller than a distance threshold are calculated using a camera model and the intrinsic camera parameters specified in AirSim. Using the position and orientation of the UAV the points are then transformed to the global coordinate system and reported to the internal map. Section 2.3 gives more details on how 3D points are reconstructed from the depth map. Specifically, from Equations (2.21) and (2.22) expressions for the 3D point can be derived.

To ensure the performance of the module verification was performed. No complete information of the occupied volume of the scene was available, but the planes of the walls, ceiling and floor were easy to obtain. The verification was performed by checking that the detected walls, ceiling and floor are in the correct planes. Additional obstacles were only checked with a qualitative comparison. To build a map of the obstacles the UAV was set to follow a predefined route while running the obstacle detection module. After the route was completed the obstacle positions were extracted and compared as described above.

3.7 Object Detection

To complete the task of finding specific target objects in the scene a module that can find the objects in the RGB images from the simulation was designed. The module is also able to calculate the positions of the objects. Scale-invariant feature transform (SIFT) was used to find the objects in the RGB images. A benefit of the algorithm is that it does not not have to be trained, contrary to a NN solution. Instead it only needs one or multiple reference images of the target objects, from which it can extract keypoints.

Since multiple target objects can be present in a single image, the mean shift algorithm, a mode seeking algorithm, was used to group the keypoints into multiple clusters [33]. Matching was then performed per cluster, meaning that potentially one target object can be found per cluster. If enough matches are found a homography is estimated, while using RANSAC to mitigate the problem with outliers [34]. Using this homography the bounding box of the reference object can be projected to the input image to give the location of the object in the image, which is defined as the center of the projected bounding box. Using the same method as for obstacle detection, see Section 3.6, the point can be reprojected to 3D space. These 3D points serve as input to the map module that keeps track of the explored world.

Since it is easy to switch the type of target object by simply replacing the reference images, the exact type of target object to be searching for does not matter much. It was therefore decided search for computer monitors, as they are already present in the *Viktoria* scene.

To ensure that the object detection module is able to recognize the target objects and calculate their positions, the system was tested by letting the UAV follow a predefined route, that should allow the UAV to see all monitors, while running the object detection script. Positions of all found targets objects were then extracted and compared to the ground truth positions in order to calculate precision and recall.

3.8 Full System Evaluation

After all modules were implemented the full system was put together and evaluated in order to see how well it performs. The system was set to control the UAV in the *Viktoria* scene and the metrics used to determine the performance were precision and recall as function of steps in the environment. In order to obtain more reliable results these metrics were averaged over multiple runs in the scene.

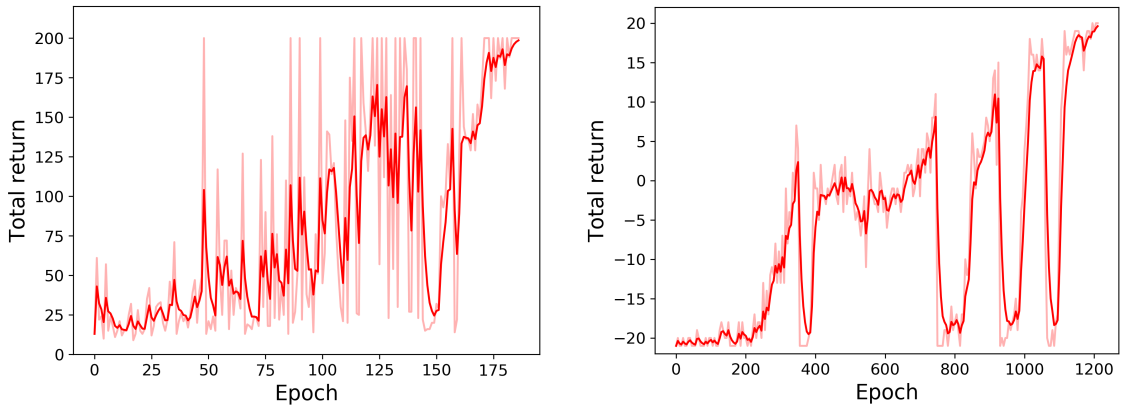
4

Results

This chapter presents results from the experiments described in chapter 3. First, the DRL algorithm was tested on standard RL problems. Then the different subsystems were evaluated individually using different metrics. Lastly, the combined subsystems were treated and tested as a full system.

4.1 PPO verification

The PPO algorithm was evaluated on *CartPole* and *Pong*, two common RL problems. Here, an episode is defined as the sequence of agent-environment interactions from start to failure or victory. In *CartPole* an episode ends when the pole falls over, the cart moves too far away from the center or when the pole has been balanced for 200 steps, which is defined as victory. An episode in *Pong* comes to an end when either player reaches 21 points. The total episode return (sum of rewards collected in one episode) as a function of number of training epochs is presented in Figure 4.1. The maximum possible returns for both environments were obtained during training. The collapses in performance that occur could indicate a poor choice of hyperparameters, but since the purpose of these tests only were to verify the algorithm, other parameters were not investigated.



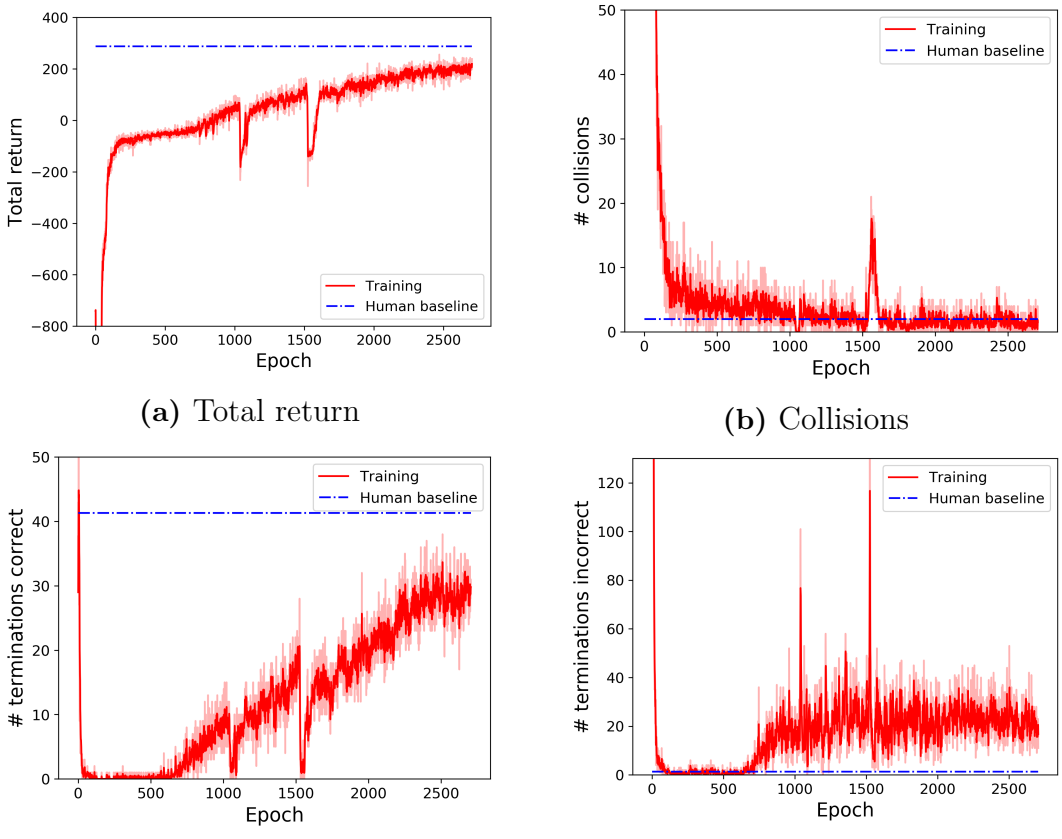
(a) **(b)**

Figure 4.1: Evaluation of the probabilistic policies showing the total return per episode for the **(a)** *CartPole* and **(b)** *Pong* environments.

4.2 Local Navigation

The local navigation agent was trained for 2700 epochs. For each epoch the total return, number of collisions, number of correct and incorrect terminations were measured, see Figure 4.2. A baseline showing the results of a human expert is included as a comparison. During the initial phase, approximately the first 500 epochs, the total return increases rapidly, while the total number of terminations quickly approaches zero. The number of collision also decreases rapidly during this phase. Thereafter, the agent starts to learn to terminate to increase its reward. Both the correct and incorrect terminations increase up to around epoch 1000, after which the number of incorrect terminations plateaus while the number of correct terminations keep increasing. It should also be noted that the training progress collapses twice, around epoch 1000 and 1500 when the number of incorrect terminations spikes. The second collapse is followed by a short period with many collisions. However, in both cases the training recovers quickly.

During training the agent was not able to reach the same level of performance as the human expert. The number of collisions is on the same level as the human expert, albeit noisy. For the other metrics the agent is performing clearly worse.



(a) Total return

(b) Collisions

(c) Terminations - correct

(d) Terminations - incorrect

Figure 4.2: Training results for the local navigation agent in the *Viktoria* scene. The dashed blue line shows the performance of a human.

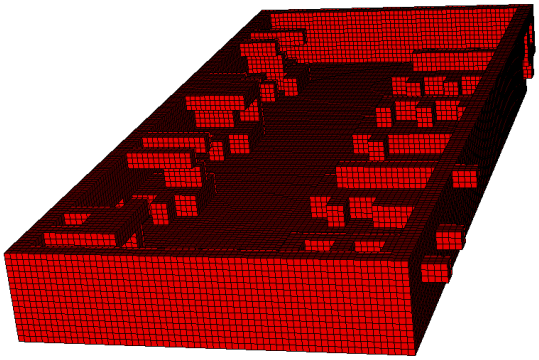


Figure 4.3: Visualization of the 3D occupancy map obtained during a verification flight for the obstacle detection script. Each voxel in the map is represented by a 2x2x2 cube in the figure.

4.3 Obstacle Detection

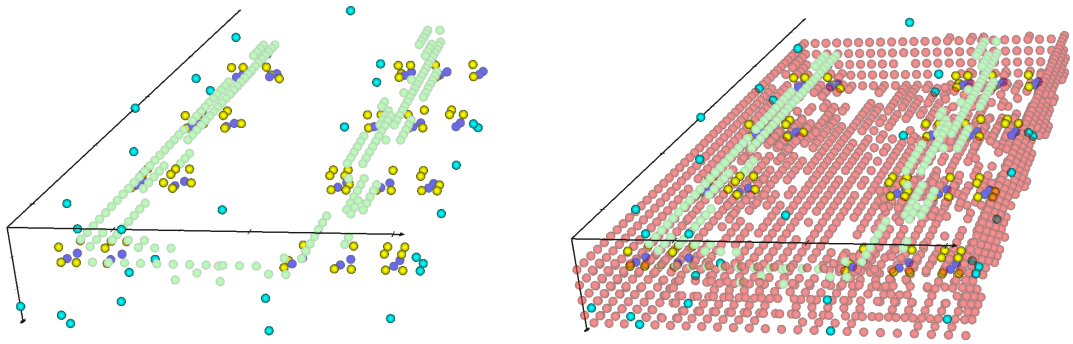
The map of obstacles identified during the verification flight is presented in Figure 4.3. In order to highlight the detected chairs, tables etc, the ceiling was removed from the figure. From the figure it can be seen that the obstacle detection module is able to detect the walls and floor to a high degree. The estimated positions of the floor, ceiling and walls were confirmed to be correct. Some irregularities exist, including a hole in the wall and several obstacles detected behind the walls. It is clear that the furniture inside the room are picked up as obstacles as well. The coarseness of the voxel map can make it hard to make out the shapes, but when comparing to the screenshot of the scene in Figure 3.2 the tables, chairs, lights and monitors can be recognized. It should be noted that these smaller obstacles are not detected as consistently as the walls and floor.

4.4 Object Detection

When verifying the object detection module a precision of 72 % and a recall of 100 % was obtained. A visualization of the map generated during the verification flight is presented in Figure 4.4. As expected most detected objects are located in close proximity to the actual objects. The figure also shows that multiple detections occur around most ground truth objects, which indicates that the system has some issues when trying to determine the position of an identified object. The false positives are more spread out and mostly located along the walls of the scene, likely due to spurious keypoint matchings in the SIFT-algorithm.

4.5 Global Planning

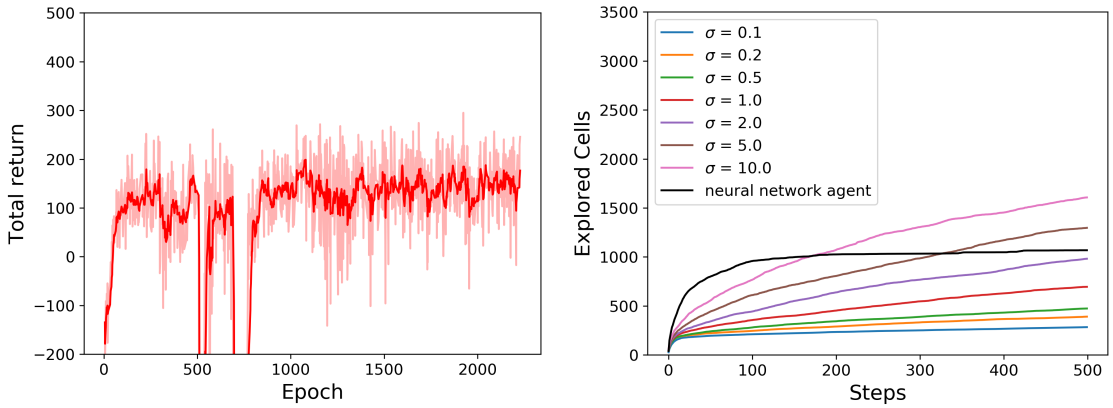
The first training results of the global planning agent trained with the naive navigator on toy maps are presented in Figure 4.5a, showing the total return per training epoch. During the first epochs the agent learns to avoid collisions by avoiding



(a) **(b)**

Figure 4.4: Visualization of the result from the verification flight for the object detection module. **(a)** Blue indicates true position of the objects, yellow the true positives, cyan the false positives and green is the trace of the flight. **(b)** On the right the detected obstacles are also included, indicated by red.

generating waypoints too close to obstacles and the total return increases rapidly. Between epoch 500 and 800 the policy once again starts to cause collisions. After recovering to previous levels of total return the agent fails to learn a better policy and the performance stagnates. After 2200 epochs of training, its performance was compared with several random walks, generated by sampling waypoints from zero-mean multivariate Gaussian distributions with varying covariance matrices $\Sigma = \sigma I$, with different values of σ. The number of explored cells as a function of the number of issued waypoints for both algorithms, averaged over multiple maps can be seen in Figure 4.5b.



(a) **(b)**

Figure 4.5: Evaluation of the global planning agent trained on toy maps with a naive navigator. **(a)** Moving average of the total return for each training epoch. **(b)** A comparison between the exploration rate of a trained agent and several random policies with varying spherical covariance matrices $\Sigma = \sigma I$, averaged over multiple maps.

The neural network based approach shows a higher exploration rate in the early

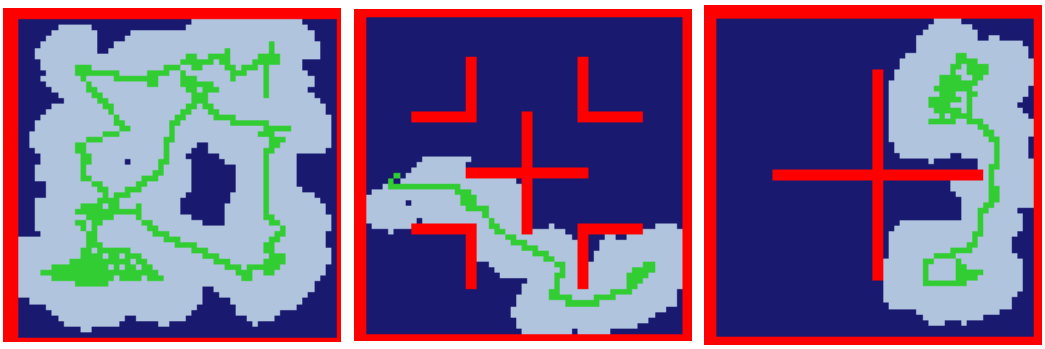


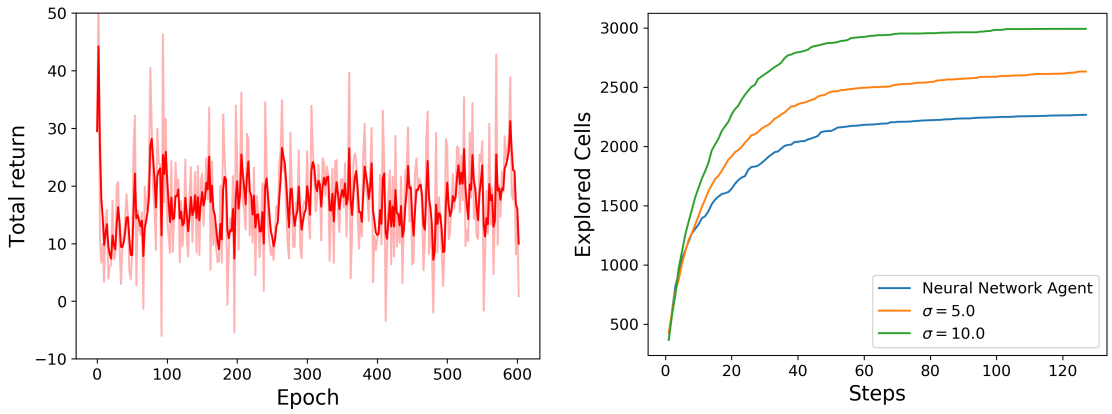
Figure 4.6: Visualization of the exploration policy trained on toy maps with a naive navigator. Dark blue cells are *unknown*, red cells are *occupied*, *free* cells that have been detected at some point are light blue and *visited* cells are marked with green.

phase of exploration. The agent is able to explore a little bit over 1000 cells on average, which is approximately a third of a map. Visual examples of the policy can be seen in Figure 4.6. The random policies do not explore the maps as quickly as the trained policy but they manage to reach a higher amount of explored cells in the end, given a sufficient number of steps. The random walk performs better with higher values of σ.

The global planner trained in the *Viktoria* scene with a pre-trained local navigator produced weak results. Figure 4.7 shows the training results in the left pane; during training no clear improvements of the return can be seen. The fact that the trained global planner fails to generate suitable waypoints is further supported by the results shown in the right pane of Figure 4.7, where it can be seen that a random walk global planner manages to explore significantly more of the scene. The total number of cells in the *Viktoria* scene is approximately 4000, meaning that on average a quarter of the scene is left unexplored by the random walk as well.

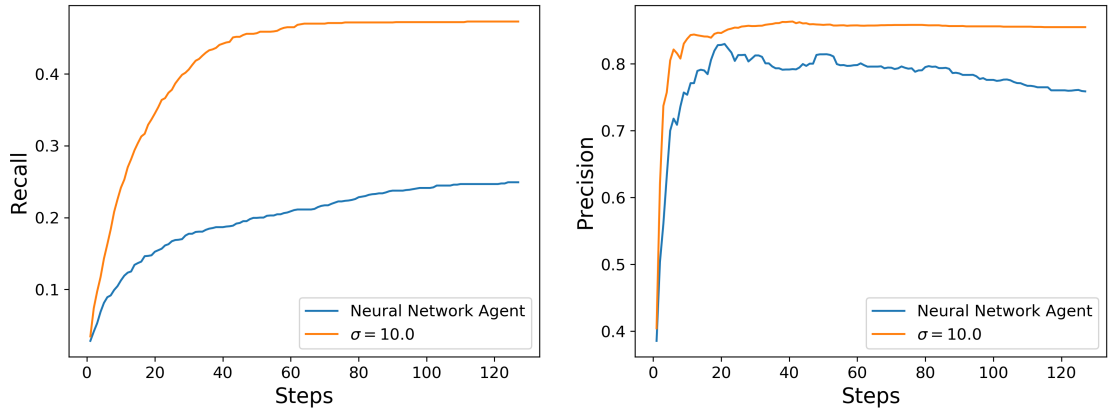
4.6 Full System

The evaluation of the full system is presented in Figure 4.8 and shows precision and recall, in terms of objects found, as functions of the number of generated waypoints. As can be seen in the figure, almost three times as many target objects are found when using a random walk as the global navigation policy compared to the trained agent. When using the random walk policy the full system finds approximately 50 % of the target objects with a precision of about 85 %. In contrast to the recall, the precision function does not increase monotonically. This is due to false positives beeing detected along the way which decreases the recall.



(a) **(b)**

Figure 4.7: Evaluation of the global planning agent trained in Viktoria. **(a)** Moving average of the total return for each training epoch. **(b)** A comparison between the exploration rate of a trained agent and several random policies with varying spherical covariance matrices $\Sigma = \sigma I$, averaged over multiple runs.



(a) **(b)**

Figure 4.8: Evaluation of the full system in Viktoria. Comparison when using the trained policy for global planning compared to random walk. The metrics are in terms of the number of objects found and averaged over multiple runs. **(a)** Recall as a function of number of steps. **(b)** Precision as a function of number of steps.

5

Discussion

This chapter will comment on the results presented in chapter 4. Many of the findings were expected and in line with current research, with the exception of the DRL approach to the global planning task which produced weaker results than expected.

5.1 PPO

The verification results of the DRL algorithm were in line with the expectations. Both RL environments were solved in reasonable time; solving the *Pong* problem required approximately six times more steps in the environment than the simpler *CartPole* and the number of steps is of the same order as reported in the original benchmark [18]. The results suggest that the algorithm was implemented correctly.

5.2 Local Navigation

The trained local navigation policy manages to reach its waypoint in a majority of the cases. Since reaching almost perfect performance seems to require a NN — with a large visual encoder and recurrent layers — trained for an extensive amount of time [5], it was expected that suboptimal, but still sufficiently good, collision-free navigation could be achieved. As can be seen in Figure 4.2, collisions occur rarely, and are comparable to the human baseline. The severity of occasional collisions depends on multiple factors, for instance the size and speed of the UAV and the fragileness of the cargo in the expected setting. It is left to be discussed to which degree collisions may be allowed.

As noted, the performance of the trained agent is not fully on a par with a human expert. The ability to terminate correctly is not as well-developed as the ability to avoid collisions, meaning that the UAV does not always manage to reach the given waypoints. Both the total return and the number of correct terminations follow an upwards trend for much of the training, but seem to plateau in the last few hundred epochs. It is possible that additional training would improve the performance further.

Due to long training times it was not possible to do a thorough search of the hyperparameter space. By investigating more choices for hyperparameters and network

designs it should be possible to achieve a performance increase. To mitigate the need for a lot of computing power, transfer learning could be used to fine-tune pre-trained NN models instead of learning the navigation policy from scratch, reducing the required training time significantly [5].

The reinforcement learning environment is assumed to be an MDP. It is however not true since the environment is not fully observable. The agent is able to estimate its height through the depth map, but it is not possible to know if there is an object directly above or beneath the UAV. This can lead to a problematic behavior if a policy starts avoiding flying over obstacles, including the tables in *Viktoria*, due to unexpected collisions. Setting a waypoint in one of these locations could prompt the agent to terminate incorrectly since the associated penalty is smaller than the penalty for a collision. On the same topic, a shortcoming of the agent is that it lacks the ability to remember information. In more complex environments with e.g. maze-like structure, navigating around multiple corners could prove difficult since this would require moving in directions significantly different from the compass direction to the target. An improvement could be to create a sequence of the most recent observations and actions and present it to the agent, similar to how frames are stacked in other DRL approaches [15]. This would provide missing information resulting in a richer observation, however the number of stacked frames required could be too high to be practical. Another way to tackle these problems would be to add recurrent layers, giving the agent an artificial memory [5].

In order to function properly this module needs reliable information about the position and orientation of the UAV to calculate the distance and angle to the target. This is also an issue for the obstacle and object detection modules, which need to be able to transform the position of the identified objects to the global coordinate system. As stated in the limitations, this problem has not been looked into here, but it must be addressed before these modules can be deployed outside a simulated environment.

5.3 Obstacle Detection

The obstacle detection module performs well, as shown by the results in Section 4.3 where it can be seen that the overall structure of the scene is captured. These results indicate that the module should give sufficiently reliable information to serve as input to the global planning module.

It should be noted that in this work a noise-free depth map is used as input to the obstacle detection module. In reality sensors are never perfect and accuracy will vary due to lighting conditions and other factors, as discussed in [32]. This raises some questions about how well the ability to identify obstacles and to determine their relative positions will translate to a real-world setting. Depth information could be obtained in different ways for a physical UAV. One way would be to have a stereo camera mounted on the UAV. Another approach would be to estimate the depth from mono camera images, something that is done in [35].

5.4 Object Detection

The results in Section 4.4 show that the object detection module is able to identify all objects in a scene, given an appropriate flight path. The module exhibits a precision that is significantly lower than the recall. As previously stated, in Section 1.2, the complete system must include a module for approaching the identified objects. This system could also include methods for rejecting false positives, which is simpler to do at a closer distance. Therefore it can be argued that it is more important that the object detection module achieves a high recall rather than precision as it is easier to reject false positives compared to finding the missed targets (false negatives) at a later stage.

Even if the verification results are promising it should be noted that the flight of the UAV was chosen to give a clear view of all objects in the scene. It is not very likely that the global planner will achieve an equally optimized path and a drop in performance is therefore to be expected in the fully autonomous situation.

As long as the objects in question are identical, or belong to a small set of objects, SIFT or other algorithms based on feature extraction work well. If the group of target objects would be more diverse, e.g. if it would include cars of different models and colors, the solutions might be more cumbersome to use, because of the large number of reference images needed. In that case it could be interesting to investigate other approaches, e.g. NN based solutions. They have the benefit of being able to identify objects belonging to certain categories, e.g. cars, instead of particular instances from such groups. One drawback of NNs is that they require large amounts of annotated data for training, which can be expensive to obtain. For common classes, including cars or other vehicles, it should be possible to find and use pre-trained networks after some fine-tuning.

It can also be noted that the object detection module stands for a significant part of the total computational time of the full system. If a speed up of the system is desired this module should be the first one to be improved. As shown in [21], other faster feature extraction algorithms exist that could achieve a speed up, but likely at the cost of the detection performance.

5.5 Global Planning

The exploration agent with a naive navigator trained on toy maps manages to detect a high degree of all cells when there are no obstacles present, see Figure 4.6, but struggles to do the same in most other scenarios. The exploration over time comparison between the DRL agent and the random policies shows that the trained agent is more efficient during the start of an episode, demonstrating a steep exploration rate. The agent is however not capable of fully exploring most of the maps, due to its behavior to keep predicting waypoints in the same spot in certain states, e.g. when being trapped in a corner. Over time the random policy on the other hand will manage to escape most enclosures, which leads to a higher score. In summary, the DRL agent is more efficient in exploring convex areas whereas the random policy

detects more cells in complex settings with obstacles present, given enough time.

For the random policy, increasing the σ, which defines the covariance matrix, leads to a higher exploration rate as expected. This is natural since the random policy by design is unable to collide with obstacles; if the agent is approaching an *occupied* cell, a new waypoint is sampled. A higher σ will therefore produce waypoints further away, which is beneficial when there is no risk for collision penalties.

A possible explanation why the DRL agent is unable to fully explore non-empty scenes is that the input (the observation) is a crop of the full map, only showing the close surroundings. This will make it hard to explore scenes with branches and multiple areas, but should not affect the ability to explore maps where there is a single clear way around the map (e.g. the right-most map in Figure 4.6). Adjustments to the observation size, architecture of the NN (potentially adding recurrent layers) and the reward function can be made to try to improve the performance of the agent.

The exploration agent trained on the *Viktoria* scene did not manage to learn any useful policy during training as indicated by the results. It is especially disappointing that the trained policy was beaten by a random walk. On the other hand the performance of the random walk is surprisingly good and with a suitable covariance it is able to explore a large portion of the scene. Further, the *Viktoria* scene is especially well suited to be explored with a random walk since it consists of a single room without any areas that are hard to reach.

Why the training of the agent fails is not completely clear. It could be due to the same reason mentioned above for the navigator of the toy maps, but other factors could also be important. One possible issue is that the local navigator agent can be unreliable; it sometimes causes collisions with obstacles and often terminates incorrectly. This could introduce extra noise to the training, making it even harder for the global agent to adapt the policy. It is however clear that additional work on the global planning module is needed to have a well performing system. Right now the better choice of policy is to use random walk instead of a trained agent.

5.6 Full System

As expected from the results for the global navigator the full system performs much better when a random walk is used instead of the trained policy. Reaching a recall of 50 % on average is not enough to consistently identify the objects. The performance of the object detection module is dependent on the flight path, generated by the local and global navigation modules, providing a clear view of the targets. This explains the performance discrepancy between the object detection verification and the full system runs. The performance level indicates that more work is needed to produce a system that performs well enough to be a competitive solution for the problem described in the background. Since the object detection module is proven to work given suitable flight paths, the global planner is likely the weak link of the full system. Despite the shortcomings of the system, the results still indicate that a UAV based approach is a potential solution.

6

Conclusion

To meet the aim of the thesis, a modular UAV control system was produced. The system meets the different requirements from the issue specification, but to varying degrees. The system is able to navigate in the environment, but not flawlessly. Some collisions occur and often the trajectory is terminated prematurely. The global planner, which is responsible for the exploration, is the weakest part of the system. The attempted DRL approach was unsuccessful and a suboptimal random walk policy had to be used instead.

The other tasks were solved more successfully. The system for detecting obstacles is able to find present obstacles to a high degree and the internal map captures the surroundings with a high fidelity. The requirements of the object detection module were met and it was shown that with a suitable flight path the module is able to find all targets. Due to the global planner, the same performance could not be achieved with the full system.

Despite all parts of the system not reaching a sufficient performance level to clearly state that this approach is suitable for keeping track of vehicles on board a RoRo ship, it is possible that with some improvements the solution could be useful. Mainly, there is a lot of room for improvement regarding the global planner. The time-consuming training of the NN-based solution was a limiting factor that prohibited the search for better sets of hyperparameters and network designs that potentially could have improved the performance. If a more thorough search would not prove useful it could be necessary to try entirely different approaches the the problem.

The same problem applies to the local navigator. A difference is that the navigation module already has an acceptable performance. With more training or some modifications to the network and hyperparameters it should be possible to fully meet the requirements. Specifically, adding recurrent layers to the NN should help avoiding obstacles that are close to the UAV but still outside its field of view, e.g. underneath or above it.

The issues excluded in the limitations also need to be addressed before giving a clear answer about the suitability of a UAV based solution for monitoring cargo decks. These are areas that can be investigated in future work.

Bibliography

[1] C. Englund, R. Rylander, and B. Duran, "In-door positioning on RoRo vessels," tech. rep., RISE Viktoria, 2017.

[2] T. Dang, C. Papachristos, and K. Alexis, "Autonomous exploration and simultaneous object search using aerial robots," in *2018 IEEE Aerospace Conference*, pp. 1–7, IEEE, 2018.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[4] T. Dang, S. Khattak, F. Mascarich, and K. Alexis, "Explore locally, plan globally: A path planning framework for autonomous robotic exploration in subterranean environments," in *2019 19th International Conference on Advanced Robotics (ICAR)*, pp. 9–16, IEEE, 2019.

[5] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva, and D. Batra, "DD-PPO: Learning near-perfect pointgoal navigators from 2.5 billion frames," *arXiv preprint arXiv–1911*, 2019.

[6] D. Gordon, A. Kadian, D. Parikh, J. Hoffman, and D. Batra, "Splitnet: Sim2sim and task2task transfer for embodied visual navigation," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1022–1031, 2019.

[7] B. G. Maciel-Pearson, L. Marchegiani, S. Akcay, A. Atapour-Abarghouei, J. Garforth, and T. P. Breckon, "Online deep reinforcement learning for autonomous UAV navigation and exploration of outdoor environments," *arXiv preprint arXiv:1912.05684*, 2019.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[9] O. Walker, F. Vanegas, F. Gonzalez, and S. Koenig, "A deep reinforcement learning framework for UAV navigation in indoor environments," in *2019 IEEE Aerospace Conference*, pp. 1–14, IEEE, 2019.

[10] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, 2015.

[11] R. Madaan, D. M. Saxena, R. Bonatti, S. Mukherjee, and S. Scherer, "Deep flight: Autonomous quadrotor navigation with deep reinforcement learning," tech. rep., The Robotics Institute Carnegie Mellon University, Pittsburgh, PA, 2017.

[12] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *arXiv preprint arXiv:1708.05866*, 2017.

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge, Massachusetts, USA: MIT press, 2 ed., 2018.

[14] C. J. C. H. Watkins, *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, UK, 1989.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[16] T. Degris, P. M. Pilarski, and R. S. Sutton, "Model-free reinforcement learning with continuous action in practice," in *2012 American Control Conference (ACC)*, pp. 2177–2182, IEEE, 2012.

[17] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, 2016.

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[19] J. Schulman, O. Klimov, F. Wolski, P. Dhariwal, and A. Radford, "Proximal policy optimization." `https://openai.com/blog/openai-baselines-ppo/`, 2017. [Online; accessed April 26, 2020].

[20] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, p. 91–110, Nov 2004.

[21] E. Karami, S. Prasad, and M. Shehata, "Image matching using sift, surf, brief and orb: performance comparison for distorted images," *arXiv preprint arXiv:1710.02726*, 2017.

[22] A. P. Witkin, "Scale-space filtering," in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'83, (San Francisco, CA, USA), p. 1019–1022, Morgan Kaufmann Publishers Inc., 1983.

[23] T. Lindeberg, "Scale-space theory: A basic tool for analysing structures at different scales," *Journal of Applied Statistics*, vol. 21, pp. 224–270, 09 1994.

[24] K. Mikolajczyk, *Detection of Local Features Invariant to Affine Transformations*. PhD thesis, Institut National Polytechnique de Grenoble, France, 2002.

[25] P. Sturm, "Pinhole camera model," in *Computer Vision: A Reference Guide* (K. Ikeuchi, ed.), pp. 610–613, Boston, MA: Springer US, 2014.

[26] DrBob and Pbroks13, "Pinhole-camera.svg." `https://commons.wikimedia.org/wiki/File:Pinhole-camera.svg`, 2008. [Online; accessed April 24, 2020].

[27] J. A. Michon, "A critical view of driver behavior models: What do we know, what should we do?," in *Human Behavior and Traffic Safety* (L. Evans and R. C. Schwing, eds.), pp. 485–524, Boston, MA: Springer US, 1985.

[28] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017.

[29] J. Achiam, "Proximal policy optimization." `https://spinningup.openai.com/en/latest/algorithms/ppo.html`, 2018. [Online; accessed April 26, 2020].

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[31] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.

[32] D. Pohl, S. Dorodnicov, and M. Achtelik, "Depth map improvements for stereo-based depth cameras on drones," in *2019 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pp. 341–348, IEEE, 2019.

[33] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, p. 603–619, May 2002.

[34] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, p. 381–395, June 1981.

[35] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in *CVPR*, 2017.

A

Hyperparameters

In this appendix chapters the hyperparameters used in the *CartPole* and *Pong* experiments are presented.

Table A.1: PPO hyperparameters used for training the NN agents on the *CartPole* and *Pong* environments.

Parameter	**CartPole**	**Pong**
Discount (γ)	0.99	0.99
GAE parameter (λ)	0.95	0.95
Clipping parameter (ϵ)	0.2	0.2
Learning rate	1e-4	5e-5
Epochs	200	1500
Steps per epoch	4096	2048
Minibatch size	64	128
Backprop. repetitions	8	8
KL divergence threshold	0.03	0.03
Value coefficient (α)	0.5	0.5
Entropy coefficient (β)	0.01	0.01

B

Experiment platform

This appendix chapter describes the hardware and software used during this project.

B.1 Hardware

All experiments were performed on a Windows 10 desktop computer with a Intel Core i7 4.2 GHz processor, 16 GB of RAM and a NVIDIA GeForce GTX 1080 Ti graphics card with 11 GB of dedicated memory.

B.2 Software

All code used in this project can be found on GitHub: `https://github.com/95ep/AutoDrone`. To run the code the following Python version and Python packages were used in this thesis:

- Python 3.7.6
- AirSim 1.2.5
- Pytorch 1.4.0+cu92
- Numpy 1.18.1
- OpenCV-Contrib-Python 3.4.2.16
- OpenAI Gym 0.15.4