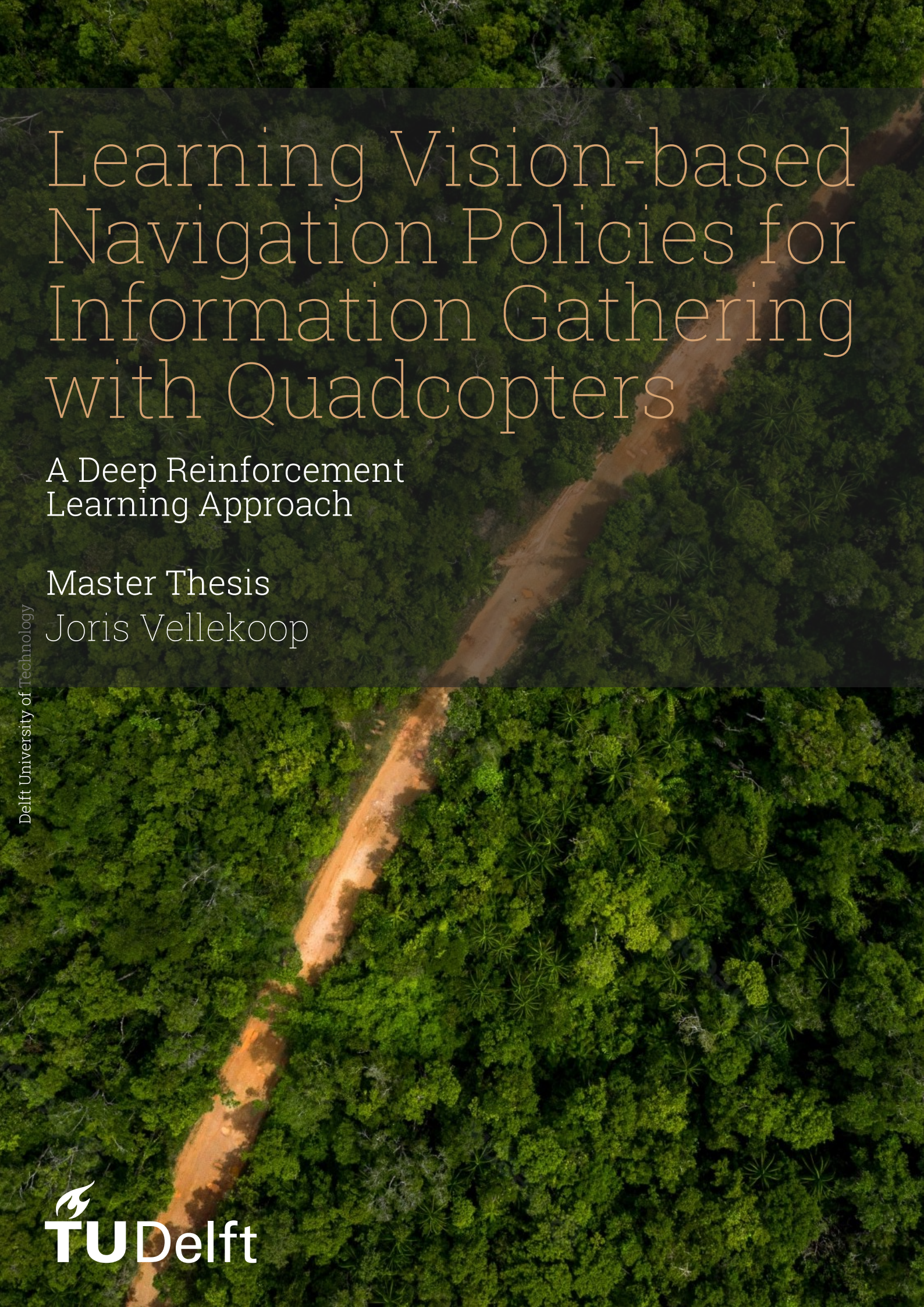


Learning Vision-based Navigation Policies for Information Gathering with Quadcopters

A Deep Reinforcement Learning Approach

Master Thesis

Joris Vellekoop

Delft University of Technology

TU Delft

Learning Vision-based Navigation Policies for Information Gathering with Quadcopters

A Deep Reinforcement Learning Approach

by

Joris Vellekoop

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Monday, 18 March 2024 at 10:00.

Student number:	4600851	
Thesis committee:	Dr. Javier Alonso-Mora	TU Delft, Primary supervisor
	Ir. Max Lodel	TU Delft, Daily supervisor
	Dr. Laura Ferranti	TU Delft, External member

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Abstract

Deep reinforcement learning presents a compelling approach for the exploration of cluttered 3D environments, offering a balance between fast computation and effective vision-based navigation. Yet, the use of 3D navigation for learning-based information gathering remains largely unexplored. Navigation in 3D space poses the challenge of having an increased state space but also provides possibilities due to the agent's increased mobility, so it is an interesting direction for research. Furthermore, current approaches to target mapping with 3D navigation do not consider cluttered environments, failing to address obstacle avoidance and occlusion handling.

This research introduces a novel deep reinforcement learning policy for vision-based information gathering with quadcopters, enabling efficient exploration of cluttered 3D environments. The core challenge is learning a time-efficient and collision-free exploration strategy, for which the policy design and the training procedure have been successfully developed. We formulate a target-searching task, where the goal is to reduce the agent's uncertainty about the target state. To achieve this goal, our method combines vision-based reasoning by deep reinforcement learning and probabilistic target mapping with an information-theoretic rewarding scheme to obtain a policy that makes informed exploration decisions.

Experiments comparing our method with a privileged greedy baseline show that in all tested environments, our policy achieves a significant outperformance. The results from our ablation study further validate our policy design, as every ablation performed results in worse exploration performance. Generally, our policy shows intelligent behaviour by effectively navigating through rooms and around obstacles. However, improvements can still be made, since failure cases where the agent gets stuck, can sporadically occur. Still, overall, the findings prove the feasibility of learning a 3D navigation policy for effective target mapping with quadcopters.

Contents

1

Introduction

Autonomous navigation is an essential part of mobile robot functionality, illustrated by the wide range of applications, such as environment monitoring [41, 64, 43], infrastructure or building inspection [36, 9] and search and rescue missions [40, 52, 61]. Quadcopters are well suited for these applications because of their high mobility. They can access hard-to-reach locations and quickly observe large areas by flying to high vantage points. In many applications, the environment is not known beforehand, in which case *exploration* is required. Generally, the goal of exploration is to gather information about the environment in the most time- or energy-efficient manner. Specifically in search and rescue scenarios, efficient exploration strategies are critical as the time is limited and the environment needs to be fully searched to guarantee no victim is overlooked.

Inherent to exploration, the environment and its obstacles are unknown beforehand. Following each action and resulting observation, the robot has increased knowledge about the environment, which improves its predictions about the optimal subsequent actions. Hence, the most effective exploration trajectory cannot be determined in a global offline fashion but should be constructed iteratively, adapting to information gathered during the task. This requires methods that allow more information processing and complex reasoning, which has to be done in real time [4]. This poses a challenge for quadcopters with on-board computers, as they are power and resource-limited, underscoring the importance of efficient exploration algorithms.

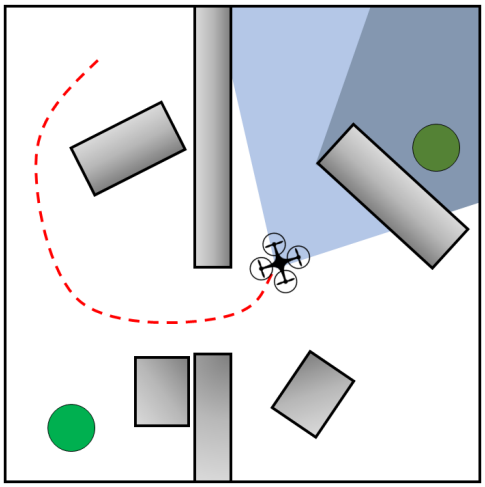


Figure 1.1: Problem schematic. The quadcopter traverses an environment containing obstacles (grey), in search of targets (green) by finding a collision-free path (red). The occlusions (dark blue) caused by the obstacles hinder target detection, so adequate view planning is necessary.

Another challenge arises from the limited sensory input. Quadcopters are typically equipped with camera sensors, which have a limited field of view. As a result, to fully observe their surroundings quadcopters need to turn and move which requires view planning [20]. Additionally, occlusions can occur in cluttered environments, which the quadcopter has to resolve by moving around obstacles.

In search and rescue scenarios, using quadcopters is advantageous. They are generally more agile and can reach remote places more easily than ground robots such as wheeled robots or quadrupeds. However, this agility brings about another challenge in quadcopter navigation, originating from the bigger degree of freedom in movement, compared to ground robots. This increases the dimensionality of the decision-making and motion-planning problem. As a result, finding the optimal actions for the quadcopter will require more complex reasoning.

1.1. Related work

Traditionally, the task of exploration has been approached as an informative path-planning (IPP) problem. In this context, the proposed strategies consist of selecting informative viewpoints and planning a path accordingly. More recently, learning-based methods have been used for exploration and work fundamentally differently. Informative viewpoints or motion primitives are chosen without explicit computations of information gain or heuristics. The following section discusses the related work on exploration methods within these categories and their relation to our method.

Informative path planning

Frontier exploration, as first introduced in [63], works by detecting boundaries between unseen and unoccupied space in a two-dimensional (2D) representation of the environment, which provides a selection of viewpoints. A cost function, e.g. travel distance, then dictates what viewpoint is chosen. This method is simple and guarantees full exploration of the environment as the viewpoints are computed globally. Viewpoints are selected in a 2D space, which might be insufficient in 3D cluttered environments. To combine frontier exploration with three-dimensional (3D) navigation, frontier detection needs to be extended to a 3D representation, as done in [65, 19]. Frontier exploration is a geometric approach to exploration that uses a maximum likelihood estimate of the map. The limitation of frontier exploration is the inability to use information metrics and to reason with probabilistic environment representations.

Information gain metrics can alternatively be leveraged to consider viewpoints anywhere in the environment instead of just the boundaries. Moreover, information gain can be computed about more than just geometric information. The information gain associated with possible viewpoints is explicitly quantified and the path resulting in the highest information gain is executed. Myopic methods, such as [13], select the next-best-view, by evaluating the information gain of possible viewpoints in the next time step. To make more informed exploration decisions, non-myopic methods estimate the information gain over a sequence of viewpoints. By using tree-search [1, 4, 33] or optimisation [21, 46, 37], near-optimal informative paths can be found. However, this comes at the cost of increasing computational complexity, as the planning horizon increases [28].

Occupancy maps are commonly used to compute information gain. These are geometric representations of the environment which divide the space into discrete cells that encode whether or not the space is occupied by an obstacle. The use of occupancy maps is sensible, as the geometric information gathered about the environment can be used directly for path planning as well. The method in [4] use the occupancy map representation to estimate unmapped volume that can be observed along randomly sampled paths to quantify information gain. While the work from [8], generates viewpoints such that the boundaries of a local sub-space are fully observed. The observed boundary surface between known free space and unknown space from given viewpoints quantifies information gain and is used to minimise the number of viewpoints. Both methods use a maximum likelihood estimate of the occupancy map to compute the information gain. However, information gain can also be quantified using a probabilistic approach by using information theory.

Information-theoretic methods quantify the reduction of uncertainty about the environment, e.g. the goal is to minimise the entropy of the occupancy map. The method from [12], formulates the information gain objective as the reduction of occupancy map entropy normalised by the time it takes to follow the trajectory. Trajectories are sampled from motion primitives and subsequently optimised by maximising the objective. Information-theoretic methods can extend information gain beyond occupancy mapping. For example, the work from [20] also formulates information gain as the reduction of map entropy but incorporates semantic information, which is extracted from RGB images using a convolutional neural network (CNN). By incorporating semantic information in the information gain objective, the agent will emphasize views where gathering semantic information is challenging.

Target maps are used for information gathering about a target variable and in contrast to occupancy maps, do not focus on the geometric properties of the environment. For environment monitoring, [47] considers quadcopter flight to minimise the uncertainty about target objects on the ground, i.e. to use a 2D representation for mapping the target variable. However, cluttered environments are not considered, so geometric information is not relevant and unused. The work from [37] also formulates

information gain as entropy reduction about a 2D target map, but does consider collision avoidance. However, the implementation is limited as the environment is not complex and the obstacle locations are assumed to be fully known. In our work, we formulate exploration as a target search problem, in which a 2D probabilistic target mapping objective is used, similar to [47, 37]. However, unlike these works, we consider indoor cluttered environments, where collision avoidance and occlusions are non-trivial problems.

Despite non-myopic IPP methods finding near-optimal paths over extended horizons, their high computational costs, resulting from repeated information gain estimations about future observations, make them less suited for time-sensitive and object-rich scenarios due to long planning times. Our method does not make navigational decisions based on explicitly estimated information gain from future viewpoints but chooses actions on a per-step basis evaluating only the current state, by leveraging deep reinforcement learning.

DRL for information gathering

Deep reinforcement learning (DRL) methods make up the majority of learning-based approaches for exploration. Reinforcement learning methods have the advantage that the state space is automatically explored during training, i.e. the policy is trained on a well varied dataset. Labelling data in RL is also done automatically by defining the reward function, which is a simple way of defining the task. Moreover, deep learning is leveraged to directly encode vision-based observations, such as RGB or depth images, to inform the policy.

Some learning-based methods have a similar approach to traditional IPP methods, i.e. the next best viewpoints are determined and navigated towards using a separate path-planning method. In the work of [10] a policy is trained to provide an informative 2D viewpoint, based on an observation of an incrementally built 2D occupancy map and the current agent position, trying to maximise a coverage reward. Similarly, in [11] the policy also learns 2D viewpoint recommendations, but is trained using a reward that incorporates semantic information. The agent receives rewards for correctly predicting semantic classes of objects, resulting in exploration behaviour that focuses on specific objects. With the approach from [31], a policy is trained to provide a 2D viewpoint recommendation with an MPC trajectory planner in-the-loop. Kinodynamic constraints are therefore considered. Similar to our work, an information-theoretic reward for target mapping is used to train the policy. The method from [23] generates informative motion primitives based on a target map. The agent is rewarded for entropy reduction, similar to our approach. However, the method does not consider obstacles in the environment. In our work, geometric information about the environment is not mapped, rather we rely on depth image encoding to provide sufficient geometric information to reason about occlusions and obstacle avoidance.

Other works skip path planning and directly map input observations to actions, i.e. motion primitives. Consequently, occupancy mapping is not strictly required but is often still done to help the agent reason about global navigation and obstacle avoidance. However, methods that use only vision-based observations have shown collision-free navigation [18, 51, 54]. In [15] a policy is trained to provide simple motion primitives based on 2D occupancy maps, with the use of a coverage reward. The policy is provided with a local and a global map that are egocentric, which our method also adopts. Additionally, obstacle avoidance is enforced by a collision penalty, which our method also employs. The method from [18] trains a policy for exploration using only RGB camera input. The algorithm is trained using ground truth data about 2D occupancy, agent location and observed parts of the environment and uses a coverage reward that increases as the amount of explored area increases. In both methods, the agent is provided with the observation if the previous action caused a collision. This helps the policy reason about obstacle avoidance and is used in our method too. Policies can also be trained with rewards that are not rooted in occupancy information. For example, [45] introduces a curiosity reward, which rewards the agent for observing unpredictable parts of the environment.

Coverage rewards are commonly used to train vision-based exploration policies. However, in our work we provide an information-theoretic reward for target mapping as seen in traditional IPP methods [37, 3]. Contrary to coverage rewards, this reward provides a continuous dense reward surface that is proven to attract the agent to unobserved parts of the environment [27]. Moreover, our method leverages deep reinforcement learning for vision-based reasoning about collisions and occlusions, similar to [15, 18].

This enables our policy to directly output motion primitive actions, eliminating the need for explicit online reasoning about information gain and collisions.

1.2. Thesis goal

Deep reinforcement learning presents a compelling approach for the exploration of cluttered 3D environments, offering a balance between fast computation and efficient vision-based navigation. Yet, the current body of literature reveals unsolved problems. Particularly, the application of DRL for exploration with 3D motion has not been researched. The challenge lies in the increased complexity of learning the behaviour for achieving collision-free navigation that can effectively handle occlusions. The increased dimensionality of agent states and possible actions results in a more challenging learning problem. Furthermore, works on probabilistic target mapping with quadcopters are scarce. Existing methods that consider 3D navigation for target mapping, do not consider cluttered indoor environments. These methods do not tackle the problem of obstacle avoidance and occlusions in target mapping scenarios. The challenge lies in training a policy that efficiently performs target search while also avoiding obstacles.

The main goal of this research is to enable target search in cluttered environments quadcopters. The mobility of quadcopters should be used by allowing 3D navigation which can also help to more easily uncover occluded targets. The focus is on the development of a vision-based reinforcement learning policy. This policy provides 3D navigation actions to the quadcopter, ensuring time-efficient exploration of the environment. It is specifically designed to learn to navigate past obstacles effectively to gather information about targets and fully explore the environment. To train and test the method, a realistic simulation framework that integrates a reinforcement learning pipeline is required. A variety of simulated training environments are needed to generate sufficient data for training. Furthermore, an adequate method for mapping potential target locations from camera observations is required, alongside a metric to measure information about these locations for policy training. Lastly, an ablation study comparing the components of the developed system will be performed. This study will consider the policy observation space and reward design, showing the relevance of the components to validate our policy design.

2

Preliminaries

This chapter provides preliminary information to give context and a basis for this thesis. We formally define the target search problem as an information-theoretic mapping problem and show how we update and quantify target information within a maintained agent belief map. Lastly, we define the partially observable Markov decision process (POMDP), which we use to model the problem for learning a target search policy.

2.1. Problem formulation

Consider a quadrotor agent that is tasked to find multiple targets in workspace $W \subset \mathbb{R}^3$ of which the locations are unknown. Additionally, the workspace contains obstacles $O \subset W$, about which the agent has no prior information and which it should avoid during flight.

The quadrotor is considered to be flying at slow speeds, allowing it to fly in a constant upright position, i.e. the quadcopter has constant zero pitch and roll angle. Consequently, the configuration space $C = \mathbb{R}^3 \times \mathbb{S}$. To navigate the environment, at each time step, the agent performs action a_t that changes its configuration $\mathbf{q}_t \in C$ according to a quadcopter transition model $\mathbf{q}_{t+1} = f(\mathbf{q}_t, a_t)$. Furthermore, $A(\mathbf{q}_t)$ denotes the volume occupied by the quadrotor given its configuration at time step t. The quadcopter shall not collide with obstacles, i.e. $A(\mathbf{q}_t) \cap O = \emptyset$.

The targets are assumed to be located on the floor and are modelled using a probabilistic occupancy grid map $\mathbf{M}$. Using this, the target search objective can be formulated as minimising the uncertainty about the target map, by choosing action a_t that results in an informative next target observation Z_{t+1}. This objective is equivalent to maximising the mutual information $I(\mathbf{M}; Z_{t+1}|z_{0:t})$ between the target map and the new observation Z_{t+1}, given the history of observations $z_{0:t}$. We formulate the agent's goal as finding a sequence of actions that maximise the cumulative mutual information in time budget T, given initial agent configuration $\mathbf{q}_0$ with a corresponding initial target observation z_0.

$$\begin{aligned} \max_{a_{0:T-1}} \quad & \sum_{t=0}^{T-1} I(\mathbf{M}; Z_{t+1}|z_{0:t}) \\ \text{s.t.} \quad & A(\mathbf{q}_t) \cap O = \emptyset \\ & \mathbf{q}_{t+1} = f(\mathbf{q}_t, a_t) \\ & a_t \in A, \quad \mathbf{q}_t \in C \\ & t = 0, 1, 2, \ldots, T \end{aligned} \tag{2.1}$$

We assume the quadcopter has perfect localisation and access to an RGB-D camera, which it uses to gather target observations Z_t.

2.2. Target belief mapping

The target map $\mathbf{M}$ models the target locations in the environment as a probabilistic 2D occupancy grid map. The grid map is defined as a n-tuple random variable $\mathbf{M} = (M_1, ..., M_n)$, where each binary random variable represents the target occupancy of a unique cell $M_i \in \{0, 1\}$, $\forall i \in \{1, ..., n\}$. The cells have the value 0 if no target is present and 1 if there is a target present.

During exploration, the agent maintains a belief state about the target locations in a probabilistic grid map, which is referred to as the target belief map. The belief for each cell i is represented as the log odds ratio $l(M_i|z_{0:t})$ of target probability, where $p_{i,t} = \mathbb{P}(M_i = 1|z_{0:t})$ gives the probability of target occupancy in cell i at time t [56].

$$l(M_i|z_t) = \log \frac{\mathbb{P}(M_i = 1|z_t)}{\mathbb{P}(M_i = 0|z_t)} \tag{2.2}$$

Initially, each cell has a uniform prior target probability, $\mathbb{P}(M_i = 1) = \mathbb{P}(M_i = 0) = 0.5$. At each time step t, observation Z_t about the true target map $\mathbf{M}$ is collected and used to update the belief target map. The observation Z_t is a vector composed of individual cell observations of the cells that are within view of the agent. The update step is then done using a Bayesian approach according to the following equation:

$$l(M_i|z_{0:t}) = l(M_i|z_{0:t-1}) + l(M_i|z_t) \tag{2.3}$$

Here, $l(M_i|z_t)$ is computed based on the inverse sensor model.

$$l(M_i|z_t) = \begin{cases} l_{occ}, & \text{if } z_t \text{ implies a target detection within cell } i, \\ l_{emp}, & \text{if } z_t \text{ implies no target detection within cell } i, \\ 0, & \text{if cell } i \text{ is not observed} \end{cases} \tag{2.4}$$

where $l_{occ} \in (0, \infty)$ and $l_{emp} \in (-\infty, 0)$ are the constant odds ratio values of the inverse sensor model.

2.3. Mutual information

As shown in Equation 2.1, we define the goal of target search to maximise the cumulative mutual information between the target map and the next observation, given the previous observations. The mutual information quantifies the extent to which future measurements will reduce the robot's uncertainty in the target map. We compute this across all grid cells in the target map, which is equivalent to the reduction of conditional Shannon's entropy about map $\mathbf{M}$. The following equation shows this relation [27].

$$I(\mathbf{M}; Z_t|z_{0:t-1}) = H(\mathbf{M}|z_{0:t-1}) - H(\mathbf{M}|Z_t, z_{0:t-1}) \tag{2.5}$$

Where $H(\mathbf{M})$ denotes the combined entropy of all cells in target map $\mathbf{M}$, which is computed according to the following formula:

$$H(\mathbf{M}) = \sum_{i=1}^{n} H(M_i) \tag{2.6}$$

The entropy is a function of target probabilities and since the target occupancy is a binary variable it is computed using the following formula [17]:

$$H(M_i) = -p_i \log_2 p_i - (1 - p_i) \log_2 (1 - p_i) \tag{2.7}$$

where, p_i is the target probability of cell i.

2.4. Partially observable Markov decision process

We model the problem as a finite-horizon Partially observable Markov decision process (POMDP). A POMDP is characterised by the incomplete information the agent receives about the environment, i.e. agent observation o_t does not fully specify the state $\mathbf{s}_t$. In the case of target search, the target and obstacle locations in the environment are unknown beforehand and not fully specified by the observations the agent receives. Furthermore, we limit the time the agent gets to search for targets, making it a finite-horizon problem.

A finite-horizon POMDP is formally defined as the tuple $(T, S, A, \Omega, \mathcal{T}, \mathcal{O}, R, \gamma, b_0)$. Here $T \in \mathbb{N}$ is the time horizon. S, A and Ω denote the admissible set of agent states, actions and observations, respectively. $\mathcal{T}$ and $\mathcal{O}$ are the stochastic state transition model and probabilistic observation model. The discount factor $\gamma \in [0, 1]$ defines the relative value of immediate and future rewards. Lastly, b_0 is the initial belief state prior to taking any actions or receiving any observations [30]. At each time step t the agent takes an action a_t, based on its current belief state b_t. This results in both the new state s_{t+1}, a new observation o_{t+1} and an immediate reward $r_t = R(s_t, a_t)$. The belief state is updated using a Bayes filter τ, given the action, prior belief state and the new observation. $b_{t+1} = \tau(b_t, a_t, o_{t+1})$

In the context of reinforcement learning (RL), POMDPs provide a theoretical foundation for dealing with environments where the agent does not have full visibility of the state space. In such scenarios, the agent must learn a policy —a mapping from histories or beliefs about the state to actions— that maximizes the expected cumulative reward over time. The belief state in a POMDP is a probability distribution over all possible states, representing the agent's knowledge about the current state based on past actions and observations. So the goal is to learn a policy π that maximises the discounted sum of rewards R over time horizon T [59], where policy π provides a probability distribution over possible actions, given the current observation o_t.

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right], \qquad a_t = \pi(\,\cdot \mid o_t)$$

3

Methodology

Our method's goal is to obtain a time-efficient and collision-free exploration policy π^* for target mapping with quadcopters. We formulate the target search problem as a POMDP, where the agent maintains a belief state about the target map, receiving partial observations of the environment, from RGB-D and GPS sensors. At each time step the agent receives a depth image, configuration, collision detection and target map observation, i.e. $o_t = \{D_t, \mathbf{q}_t, c_t, Z_t\}$. To solve the POMDP, we formulate the estimation of π^* as a learning problem, which we solve using a deep reinforcement learning (DRL) approach.

The DRL policy is trained to choose actions for 3D motion, for which we design a reward function based on information gain over time and collision detection. The action space results in higher agent mobility compared with other target mapping methods and can be leveraged to find more effective trajectories. Our method does not consider quadcopter dynamics and assumes start-coast-stop motion using a collision-aware low-lever trajectory planner such as in [32]. This allows us to define actions as configurations relative to the agent, and reject execution of those that result in collision.

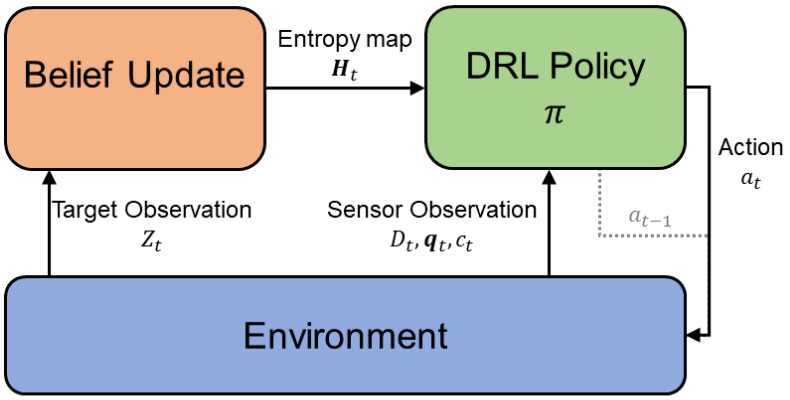


Figure 3.1: System structure

As shown in Figure 3.1, our method's pipeline consists of a target belief mapping method and a DRL policy. The belief map update and mutual information computations are done using target observations Z_t, which are obtained using our target observation model. We assume the agent is given access to a black-box classification model, such as [50], that can detect targets in RGB images. By leveraging the geometric information from the depth images the target cells and their occupancy are deduced, resulting in target observations Z_t.

We design our DRL policy to receive the the target belief entropy map $\mathbf{H}_t$ which spatially represents environment uncertainty, similar to the method in [31]. This informs the agent about what regions

have previously been explored and what regions are informative and promising to navigate towards. Furthermore, our DRL policy receives depth image observations, directly informing it about obstacles near the agent. This information is useful for finding trajectories that avoid obstacles and overcome target map occlusions. Lastly, the agent is provided with its previous actions and the corresponding collision detection boolean, allowing it to reason about its immediate history. The DRL architecture has a commonly seen structure, incorporating convolutional neural networks (CNNs) for image encoding, a long short-term memory network (LSTM) for temporal reasoning and value and policy heads for leveraging actor-critic algorithms. Training is done in randomly generated environments according to different scenario types, which is discussed further in chapter 4.

3.1. Observation modelling for target mapping

Using our observation model target observations Z_t are obtained from RGB-D data. We assume that target detection is achieved using an existing classifier method [50, 11], which can be used in combination with depth images to compute target(-free) cells. Furthermore, target observations are limited to a given depth in the depth image to ensure detection accuracy. The depth range depends on the agent height, which is further discussed in this section.

Depth image observation

The quadrotor agent is equipped with a depth sensor, that provides a depth image $D_t \in \mathbb{R}^{W \times H}$ at each time step. Here W and H denote the width and height of the depth image, respectively. For every pixel i in D_t, a target classification $c_i \in \{0, 1\}$ is moddeled an image classification model.

To gather target observations, we cannot solely use the classification in the image space, because to map the targets, their locations in 3D space also need to be determined Using the known agent position and the depth camera's intrinsic parameters, depth image observations can be mapped to 3d points according to the pinhole camera model [62].

We assume targets are located on the floor, so only the points that coincide with the ground plane, are used as target observations. Similarly to [15, 14], we select the points with a maximum distance to the ground and project them to the ground plane to determine their position within the 2D target map $\mathbf{M}$. The grid cells that contain one or more of the selected points are the visible cells, i.e. they are within the agent's view. Consequently, target observation Z_t consists of the target classification of these cells.

Ground sample distance

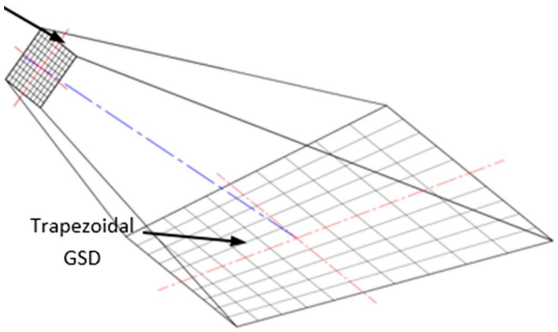


Figure 3.2: Camera view illustrating the effect of the camera pose on the ground sampling distance

Multiple pixels in the depth image can correspond to the same grid cell, in which case it is considered as one observation in the cell. In this case, the ground sample distance, i.e. the distance between points from neighbouring pixels in the depth image projected on the ground, is low. On the other hand, at sufficiently far distances or view angles, the ground sample distance can be bigger than the size of the grid cells that make up the belief target map. In this case, aliasing occurs so we disregard these pixels.

An equation that describes the distance at which this occurs can be derived using trigonometry and is shown below. For our target observations, only depth image values below the maximum distance are considered. The depth limit is a function of the agent's height p_z, the cell size in meters c, the horizontal and vertical resolution H and W and the corresponding field-of-view angles α_H and α_W.

$$d_{max} = \min\left(\sqrt{\frac{p_z c H}{2\tan(\frac{\alpha_H}{2})}}, \frac{cW}{2\tan(\frac{\alpha_W}{2})}\right) \tag{3.1}$$

3.2. Policy design

We present our solution reinforcement learning policy design individually and discuss the individual components: observation space, action space, reward function and network architecture.

3.2.1. Observation space

The observation space is designed with multiple considerations in mind. Primarily it should include features that are directly relevant to target search, so a representation of the target belief state is essential. Observations that allow the agent to learn collision avoidance and temporally consistent behaviour are also relevant. Furthermore, to facilitate learning, the observation space should be simple while retaining essential information. A complex or high-dimensional observation space can slow down learning or lead to suboptimal policies due to difficulty extracting useful patterns. Ideally, the observation space should enable the RL policy to generalize from learned experiences to new, unseen environments or variations in target characteristics.

We take inspiration from multiple other works on similar tasks for our observation space design. It consists of the depth sensor observation, a global and a local egocentric target entropy map, the action chosen in the previous step and its validity. Furthermore, the remaining time as a fraction of the total time is provided. These parts will be discussed in the following.

Depth observation

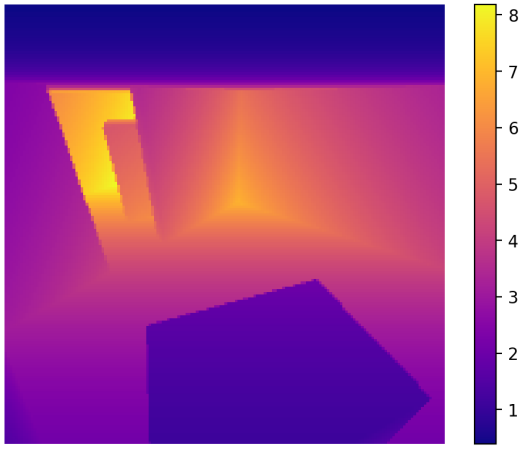


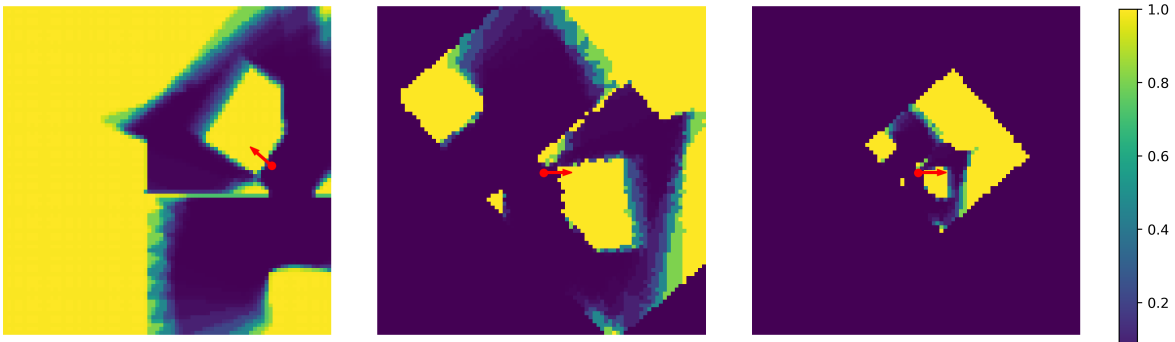
Figure 3.3: Example depth image observation

As previously mentioned, the agent receives depth image observations D_t from a front-facing camera of which resolution is 168x168 pixels and the field-of-view is 90 degrees. Depth images provide 3D spatial information about obstacles, which is useful for collision avoidance and reasoning about occlusions. Compared to 3D spatial representations, e.g. voxel maps, they are a more manageable representation of obstacles. Depth images also offer generalisability, as the spatial data applies to any environment.

Entropy maps

As explained in chapter 2, the agent maintains a belief state about the target map. For our policy, we represent the belief state as a map containing individual cell entropies $H(M_i)$ about the target map, i.e. the entropy map $\mathbf{H}_t = (H(M_1), ..., H(M_n))$. This representation quantifies the uncertainty in target locations as a single measure, directly informing the agent by highlighting areas of interest, i.e. regions it has not observed much. Furthermore, it is directly linked to the mutual information reward function. Our experiments consider a target map representing an area of 10m by 10m, with cells of size 0.1 m, resulting in a 100x100 entropy map resolution.

We provide the agent with two egocentric entropy maps on different scales. A local map that is more detailed and a global map that is more coarse, similar to [15, 24]. The egocentric representation aligns the entropy map observations with the agent's current viewpoint. By doing so, all observations are relative to the agent's current configuration, which is thought to aid learning [15]. Both map observations are downsampled to an 84x84 resolution.



(a) Base belief entropy map **(b)** Local egocentric transformation **(c)** Global egocentric transformation

Figure 3.4: Examples of belief entropy maps and how egocentric image transformation looks in relation to the base entropy map. The red arrow indicates the agent's position and orientation.

The local entropy observation encodes the 5m by 5m area surrounding the agent intended to help the agent in precise, short-term navigation by showing the immediate surroundings and detecting target occlusions in the environment. When an obstacle causes an occlusion it will result in the observable lack of entropy reduction in the region directly in front of the agent.

The global map observation is intended to give the agent a broader overview and help with long-term planning, as unexplored parts of the environment are not always visible in the local entropy map. The global entropy map is scaled down such that the whole map is visible for any agent configuration.

Action observation

The action vector observation is a 9-element vector $\mathbf{v}_t \in [0, 1]$ and encodes three things: 1. the action taken in the previous step, 2. if the previous action was valid and 3. the remaining time budget for the task.

The previous action observation provides the agent context about its immediate past, enabling it to learn about the outcomes of its actions which helps future decisions. For example, this was shown to help the recovery from stuck states in point-goal navigation problems [60]. The previous actions are represented using a one-hot encoding of $6 + 1$ elements. 6 actions from the discrete action space, with one value encoding the start of the episode, i.e. when the agent has not taken any action yet. This is referred to as the start token.

If the agent chooses an action that would cause collision it is considered invalid and the action is not executed. Similarly to [15], we provide the agent with the binary collision result of the previously chosen action. Specifically, in combination with the previous action observation, it gives immediate feedback about the correctness of the previous action and helps the agent understand what actions should be avoided.

Lastly, the agent is provided with the time remaining as a fraction of the time limit T. The paper from [44], shows that this is essential in time-limited scenarios. Optimal actions depend on the time remaining in the episode, which can intuitively be explained. If enough time is left, backtracking over already-seen parts of the environment to uncover unseen parts might result in the highest information gain. However, if there is little time left the most rewarding behaviour might be to exploit the parts of the environment near the agent that are less informative. Since our problem is time-limited, the observation of remaining time is useful.

3.2.2. Action space

The action space contains 6 discrete actions: move forward 0.2m, move up 0.2m, move down 0.2m, turn left 10°, and turn right 10° and do nothing, similar to [60] but with added vertical movements. The The action space does not represent all degrees of freedom of a quadcopter. For example, sideways movement or a combination of rotation and translation can also be considered. However, the simplicity of this action space reduces the complexity of the learning problem, while still allowing three-dimensional navigation.

3.2.3. Reward function

The reward function should guide the agent towards optimal target search behaviour by quantitatively evaluating its actions. We define it according to the following equation:

$$R(s_t, a_t) = \alpha\, I(\mathbf{M}; Z_t | z_{0:t-1}) - p_t - p_c \tag{3.2}$$

The reward function consists of three parts:

- Mutual information I: as discussed in chapter 2, the primary goal of the agent is to reduce the uncertainty about the target map which is equivalent to maximising the mutual information reward function.
- Time penalty p_t: to incentivise the agent to rapidly explore the environment a time penalty is used.
- Collision penalty p_c: to discourage the agent from choosing actions resulting in collision a collision penalty is applied.

Mutual information

The agent's goal is to maximise mutual information as captured by Equation 2.5. This is an intrinsic reward, i.e. the reward is computed based on the agent's belief state and not by interaction with the environment itself. Furthermore, we normalise the MI with α, to account for differences between randomly generated environments. If the environment contains obstacles on the floor they obscure the target map cells underneath, making them unobservable. Consequently, the uncertainty about the target belief map cannot be reduced at these locations. This creates a problem since we measure successful exploration as reaching a given belief map entropy threshold. In more obstacle-dense scenarios the threshold will be harder or impossible to reach because in some parts the entropy cannot be reduced. Therefore, we normalise the MI by the unobscured cells relative to the total area, such that the total obtainable reward is the same for every environment, regardless of obstacles and equal to the total area of the environment, i.e. $\alpha = A/n_o$, where A is the total environment area in m^2 and n_o, is the number of observable cells in the target map.

Coverage rewards are frequently used in the context of learning exploration policies [15]. When coverage is used, the agent receives a positive reward once for seeing a cell, after which it is mapped as seen and any further observations of the cell will not be rewarded. This reward does not encode uncertainty about the observations. Our goal is to probabilistically map targets, so the reward should be in accordance. Furthermore, an argument against coverage reward is that it is more sparse, as the agent only receives a reward for each seen cell once. In a situation where the agent reaches a dead end and needs to backtrack to get to an unvisited part of the environment, the agent receives no reward for doing so, thereby not enforcing it, even though it is the right behaviour.

Time penalty

To enforce the time-efficient exploration, we add a time penalty term $p_t = 0.05$ to the reward. This is an intrinsic penalty and is hypothesised to make the agent more inclined to finish the episode quickly, as the time penalty will cumulatively add negative rewards the longer the episode lasts. The desired policy obtains the fastest overall entropy reduction, by rejecting actions that return high rewards in the short term but are inefficient in the long term.

An alternative to this reward is modifying a terminal reward based on the time it takes to complete the task. We find this to have a similar effect on the trained policy, but for the sake of simplicity, we choose a constant time penalty.

Collision penalty

To enforce collision avoidance the agent receives a collision penalty $p_c = 0.1$ anytime it chooses an action that would result in the collision. This is an extrinsic reward, as the collisions depend on interactions with the environment and not the agent's belief. Actions that cause collisions are not simulated, to ensure the training can continue.

$$p_c = \begin{cases} 0.1, & \text{if taking action } a_t \text{ results in a collision,} \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

An alternative way to reduce collisions is penalising them harshly and ending the episode, in which case future rewards are limited. However, we have defined a constant negative time penalty, which could result in "reward hacking", where colliding and ending the episode might lead to a higher reward than trying to continue navigating in the environment, accumulating negative rewards. This could be solved by making the collision penalty orders of magnitude bigger than the time penalty, but that might result in excessively cautious behaviour.

3.2.4. Policy Architecture

The policy architecture can be subdivided into 3 components: encoding and concatenation of the observations, parameterisation by a recurrent neural unit, followed by policy and value function heads.

Encoders

Convolutional Neural Networks (CNNs) serve as the encoders for processing spatial data from depth images and entropy maps, compressing visual inputs into a latent space, which can be used to efficiently learn exploration behaviour. The visual encoders for the depth images and entropy maps are based on the same convolutional neural network, namely [39] and the standard encoder in the stable baselines 3 package [49]. It consists of 3 convolutional layers followed by a fully connected layer, which is summarised in the following table. Each layer has a ReLU [22] activation function, and no padding is used in the convolutional layers.

Table 3.1: Convolutional Neural Network Architecture

Layer	Output Size	Kernel Size	Stride
Input Image	$84 \times 84 \times 1$	-	-
Conv1	$20 \times 20 \times 32$	8×8	4
Conv2	$9 \times 9 \times 64$	4×4	2
Conv3	$7 \times 7 \times 64$	3×3	1
Fully Connected	256	-	-

An attentive reader might notice that the depth image resolution of 168×168 does not correspond to the input image size of the visual encoders. Similar to [60], we pass the depth images through an average pooling layer with kernel size and stride of 2 without padding. They find that this does not affect performance but helps reduce the computational load, as the network requires fewer parameters.

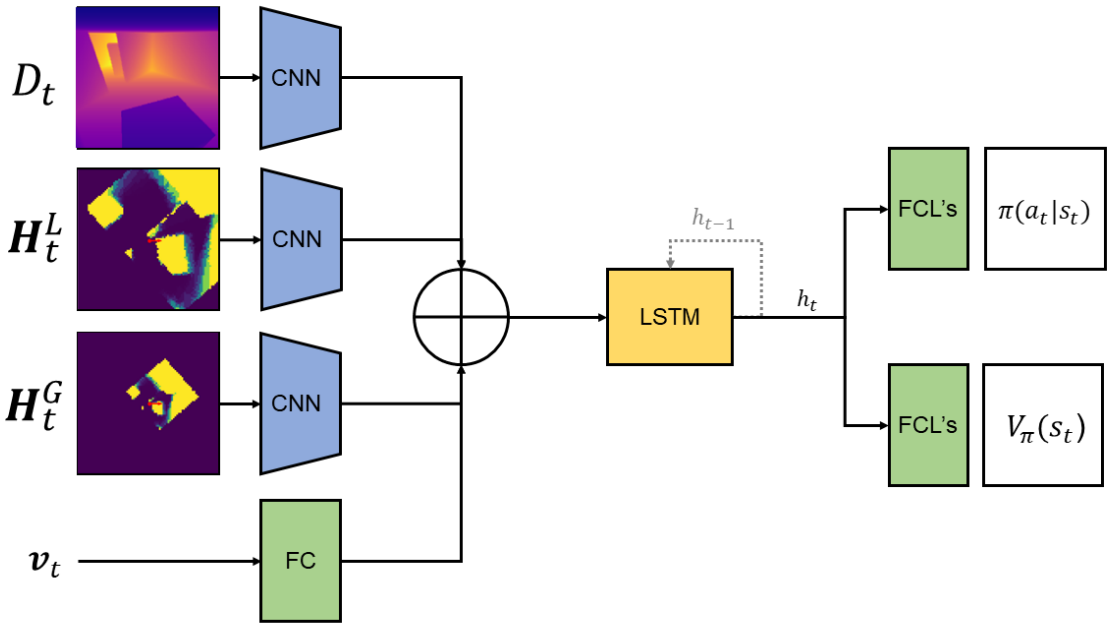


Figure 3.5: Policy architecture: Left is the input showing the depth entropy and action vector observations. Shown right are the police and value functions. The LSTM is provided with the hidden state from the previous step

Lastly, a 64-dimensional embedding is made from the action state vector observation by passing it through one fully connected layer. Considering all observations and their encoding, their resulting outputs are concatenated along the feature dimension, resulting in an 832-dimensional vector.

LSTM and heads

To aid the agent's sequential decision-making process and handle partial observability of the environment, a Long short-term memory (LSTM) network layer is incorporated. The latent 832-dimensional vector is passed to a 1-layer LSTM [26] with a 256-dimensional hidden layer. The agent can remember and use historical information to inform current actions. The target map serves as a kind of memory, showing what parts have not been observed yet. However, geometric information about the environment, which is relevant for obstacle avoidance and navigation planning, is not mapped. Hence it is useful for the agent to have a memory of previous depth image observations.

The architecture ends in separate actor and critic heads, that output the policy and value function. This allows us to use proximal policy optimisation (PPO) [53]. Both heads have fully connected layers of 256 and 64 neurons each, with ReLU activation functions. The actor head ends with a fully connected layer of size 6 with a softmax function, which corresponds to the 6 discrete actions. [5] activation function. The critic head ends in a single output neuron that estimates the state value function.

4

Training setup

This chapter presents our training procedure and how it relates to the simulation setup. Our experiments are done solely in the Flightmare simulator [55], where we train and test our policies and baselines. We discuss Flightmare's features and present the modifications made to accommodate our method. A method for generating random indoor environments is developed to incorporate domain randomisation [57, 51] during training. It allows us to easily generate environments with varying degrees of complexity and evaluate the influence on the policies' performance. Additionally, with random generation we can ensure that the test environments are different from the training environments, to evaluate our methods. For comparison, we define and show three different environment types for our experiments.

4.1. Traing procedure

To train the policy we simulate a reinforcement learning environment using the Flightmare simulator [55]. Proximal policy optimisation (PPO) is used to update the policy and value function [53]. Each episode, the agent is initialised in a random starting position in a randomly generated indoor environment, containing rooms and box obstacles. Examples can be seen in Figure 4.3 at the end of this chapter. The agent starts with a belief map state that is fully uninformed, i.e. with maximum entropy in each cell. During training, target detections are not simulated, i.e. the agent explores the environment and reduces uncertainty but only encounters target-free cells. This does not affect the training or the learned behaviour as the RL agent observes and is rewarded based on the belief entropy map, which only encodes uncertainty.

The agent is rewarded according to subsection 3.2.3 and is considered to have successfully explored the whole environment when the total residual entropy in the belief map is below threshold value $\beta = 0.05$. If the agent does not reach the entropy map threshold with the time limit of 512 steps, the episode is considered a failure. The episode is terminated when the time limit is reached or if the agent has reached the entropy threshold. Parts of the environment where obstacles cover the target cells are not observable, so the cell entropy cannot be reduced. To account for this, we compute entropy exclusively about observable cells, such that if the entropy is only high in obstacle-covered regions, the measured entropy is zero. The average belief map entropy $H_\mu(\mathbf{M})$ is given by the following formula. $H_\mu(\mathbf{M}) = \frac{1}{n_o} \sum_{i \in \mathcal{I}_o} H(M_i)$, where the set of observable cell indices is $\mathcal{I}_o = \{i_1, ..., i_{n_o}\}$ and n_o the total number of observable cells.

The Flightmare simulator allows us to run multiple quadcopter environments in parallel and vectorise the training environment. During training, we run 16 separate instances in parallel. In each policy update step, the agents collect 512 frames of experience with which 2 epochs with a minibatch size of 1024 are performed. We run the training a total of 10 million steps. For increased training speed, we initialise the learning rate at 2.5e-4, which is reduced to 1.0e-4 after 2.5 million steps. Furthermore, we choose the remaining hyperparameters following the most common PPO implementations, which are shown in the table below.

Hyperparameter (Symbol)	Value
Learning Rate (α)	$2.5 \cdot 10^{-4}/1.0 \cdot 10^{-4}$
Discount Factor (γ)	0.99
Clip Range (ϵ)	0.2
GAE Parameter (λ)	0.95
Number of Epochs (K)	2
Horizon (T)	512
Batch Size (N)	1024
Entropy Coefficient (c_{ent})	0.01
Value Function Coefficient (c_{vf})	0.5

Table 4.1: Hyperparameters of the PPO Algorithm

4.2. Simulation setup

The Flightmare simulator [55] is appropriate for this research as it has a unique combination of features not found in other simulators. Firstly, Flightmare can simulate many agents in parallel and achieve high simulation speeds, which is useful for rapid data collection and RL policy training. Secondly, it includes a standard OpenAI Gym [6] wrapper that can be used for many RL tasks and is easy to integrate with existing baseline RL algorithms. Thirdly, Flightmare uses the unity engine, with which custom environments can be built and realistically rendered using a varied sensor suite. The sensor simulation is customisable and has many modalities, such as RGB, Depth and semantic segmentation and can also account for sensor noise.

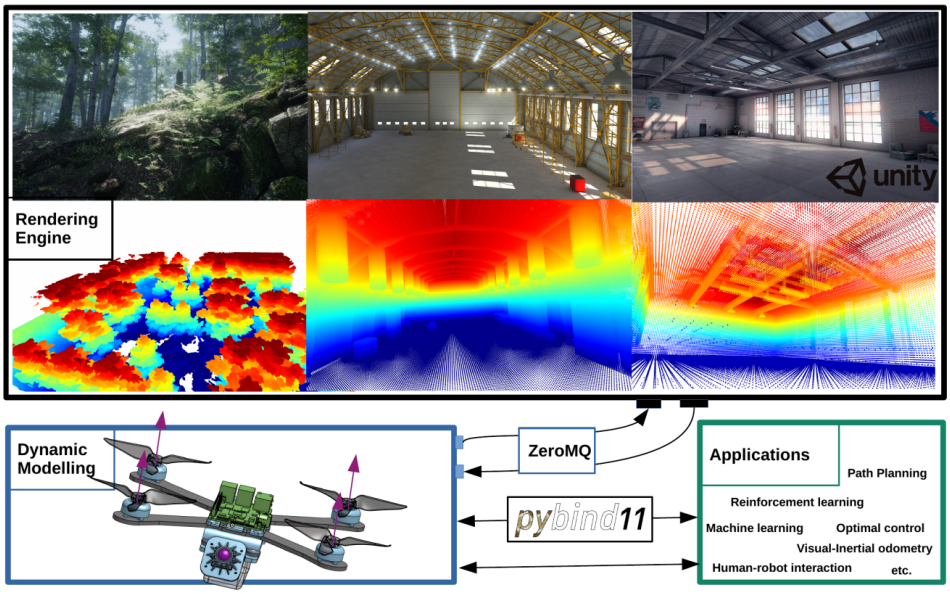


Figure 4.1: Flightmare simulator diagram. (figure taken directly from [55])

Flightmare consists of two decoupled modules: a photorealistic rendering engine built using the Unity Editor [58] and a quadrotor dynamics simulator built in C++. The decoupling of the quadrotor's dynamic model from the rendering engine allows faster dynamic simulation by using parallel programming. The asynchronous messaging library ZeroMQ [25], provides interfacing between the dynamics simulator and Unity. Furthermore, a Python wrapper for reinforcement learning tasks is provided, which interfaces with Flightmare using Pybind11 [48]. An overview of the Flightmare system is shown in Figure 4.1. For our experiments, Flightmare required additional functionality, for which the following modifications were made.

Set state function

Our method does not incorporate a trajectory planner or motion controller. Instead, our method provides decisions about a discrete set of actions, corresponding to viewpoints relative to the quadcopter. To forward simulate the actions, we remove the dynamic modelling within Flightmare and simply set the quadcopter's position and orientation to the chosen viewpoint at each time step. To accommodate this, the Python wrapper was modified by implementing a set state function. Furthermore, the physics step simulation is disabled.

Object management

Our method implements random environment generation, which is done on a per-episode basis. Practically, this means that during simulation, objects making up the environment need to be added and removed, according to the structure of the generated environments. Originally, the Flightmare simulator and its Python API did not support dynamically adding, removing and modifying existing objects in the environment. To address this, we added specific functionalities, through the following steps: First, we added new objects within the Unity simulator. These were elementary shapes, to keep the environment generation manageable. Additionally, we modified the unity simulator code to enable the addition and removal of these obstacles via the Python API. For this, a new ZeroMQ messaging interaction had to be implemented. From the C++ framework, an array specifying object types and their poses is sent to the unity simulator via ZeroMQ messages. Additionally, the unity simulator code was modified to process the incoming messages and manage the obstacle information by changing the environment accordingly. Lastly, the functionality was integrated into the Python API with Pybind11.

Collision checking

Our method disallows the agent to execute actions that would cause collisions. For this, we require the ability to check if an action causes a collision, before its execution. Originally, Flightmare already provided collision information, but only after a simulation step. Moreover, when executing a simulation step through the Python API both rendering and collision detection are processed concurrently. Thus, using this way to check for collisions is impractical and inefficient. Therefore, we added the functionality to check for collisions separately from the simulation step. For this another ZeroMQ messaging interaction was designed. The Unity simulator code was modified to process the incoming messages containing quadcopter poses and check these for collisions using a custom function. A ZeroMQ data stream back to C++ containing the results is designed. And lastly, Pybind11 was used to add the function to the Python API.

Quadcopter specifications

For our experiments, we use the following quadcopter specifications. The quadcopter has a size of approximately 0.2m $\times$ 0.2m $\times$ 0.06m, but we model the body as a sphere with a diameter of 0.2m for collision detection. This is done because Unity checks for geometry intersections, for which a spherical collision body is the most reliable and efficient. Furthermore, the camera resolution is set to 168 $\times$ 168 pixels and the field-of-view to 90 degrees. Additionally, the camera is positioned with a 30-degree downward pitch relative to the quadcopter's body, to increase the view of the ground plane and consequently target visibility.

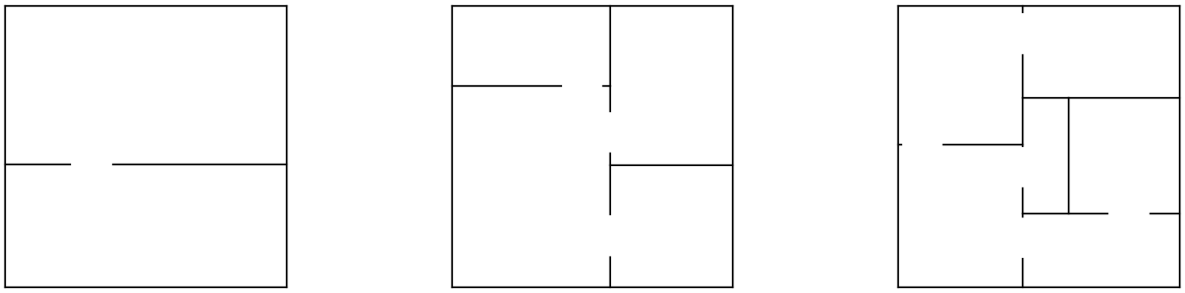
4.3. Environment generation

To make sure the policy is robust and generalises well, we perform training on randomly generated environments consisting of multiple rooms containing randomly sized and placed boxes, similar to [29]. For our experiments, we only consider depth image observations and exclude RGB image observations, so we only consider variations in geometry for environment generation. The environments are closed-off indoor environments with an area of 10m $\times$ 10m and a height of 3m. Within this space, walls can be placed to form separate rooms and boxes can be placed to further increase the complexity of the environment. An algorithm is developed that randomly generates room layouts, specifying the position of the walls and obstacles. The generation process can be subdivided into two subsequent parts: First,

the floor plan generator that forms coherent floor plans with a variable number of rooms among other settings; and Second, the random placement of box obstacles. For our experiments, we also consider an environment type that has doorways at varying heights, in which case additional walls are placed at the position of the doorways.

Floor plan generator

We develop a custom algorithm 1 for generating floor plans, inspired by the works from [38, 35]. Conceptually, the algorithm works by first dividing the floor area into randomly sized rooms, making use of the squarified treemap algorithm [7]. After this, doorways are added at random positions along the walls, such that all rooms are connected and the whole environment can be traversed. Additionally, the position, size and orientation of the walls are computed, such that the floor plan layout information is compatible with the Unity simulation.



(a) 2 room floor plan **(b)** 4 room floor plan **(c)** 6 room floor plan

Figure 4.2: Example floor plans with different number of rooms

The algorithm's input is the number of rooms and the maximal allowed size ratio between the smallest and the largest room, e.g. if the value is 2 the biggest room has at most 2 times as much area as the smallest room. This guarantees the algorithm does not create rooms too small or narrow to traverse. A list of random room sizes is generated by uniformly sampling values between 1 and the size ratio and normalising them to make the sum of areas of the rooms correspond with the total floor plan area. The list of room sizes is used as input for the squarified treemap algorithm [7], which computes the position and dimensions of every room, such that the rooms are close to square-shaped. By doing this, the rooms do not become long and narrow, which makes the generated floor plan more realistic.

After this, we need to convert the squarified treemap to the individual walls of the rooms. This is done because we need to know the position and size of all walls to place them in the Unity environment. Additionally, doorways that connect the rooms have to be generated, for which we need to know what rooms are adjacent to each other and what wall the adjacent rooms share. Adjacent rooms are saved as room pairs simultaneously with the wall conversion.

Doorways need to be generated such that all rooms are connected to the rest of the environment, i.e. every part of the environment can be reached from any point. The room pairs are used to determine what rooms are connected if a doorway is placed on the corresponding wall. A room can only be connected with a door if the wall is big enough for a door to be placed. An iterative algorithm is used to go through all the room pairs until all rooms are connected to the rest of the environment, or no pairs are remaining. In the latter case, making a fully connected floor plan with the treemap is not possible and the generation process is repeated.

Then, if the environment scenario with varying doorway heights is required, additional walls are placed at the doorway positions. This is done by randomly selecting a high or low doorway placement and adding a wall at the bottom or top of the environment, thereby blocking half of the doorway.

Random obstacle generation

After the floor plan is generated obstacles can be randomly added to the environment following our algorithm. Conceptually, the algorithm works by randomly sampling box dimensions and iteratively adding the boxes at positions where there is free space.

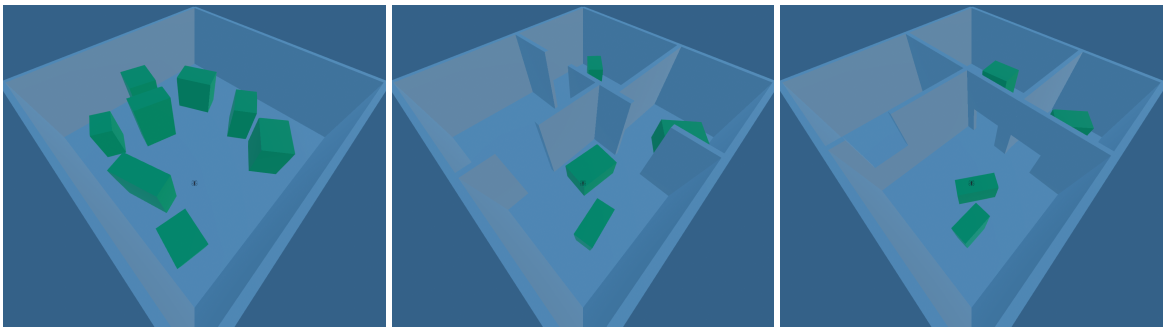
Algorithm 1 Generate Floor Plan

function generate($n\_rooms, size\_ratio$)
  for $i \leftarrow 0$ **to** $max\_iteration - 1$ **do**
    $room\_sizes \leftarrow$ list of size n_rooms with random values between 1 and $size\_ratio$
    Normalise $room\_sizes$ to fit within floor plan dimensions
    $rooms \leftarrow$ squarified_treemap_algorithm($room\_sizes$)
    $walls, room\_pairs \leftarrow$ get_walls($rooms$) ▷ Determine what rooms are adjacent
    $linked_room_pairs \leftarrow$link_rooms($walls, rooms\_pairs$) ▷ Link walls that can have doors
    if all rooms are linked **then**
      break
    end if
  end for
  $walls \leftarrow$ add_doorways($walls, linked\_room\_pairs$) ▷ Add doorways to walls from linked rooms
  return walls
end function

First, the size of the box is randomly and uniformly sampled within predefined size ranges, resulting in the box length, width and height. Since the boxes are placed on the floor, only their 2D placement has to be determined. To generate the box position, a 2D point is uniformly sampled within the dimension of the environment using Algorithm 3. This algorithm checks if the point is at least a given distance away from the walls, by comparing the position coordinates with the wall positions, given the margin distance. Then, the algorithm verifies that the new box does not intersect any of the previously added boxes. This is done by comparing the distance between the positions, given the boxes' radii, which are computed as the distance between the centre and the corners of the box. If the position is valid, a random angle is generated, and the box is added. Algorithm 2 shows the pseudo code for the random box pose generation.

Example environments in Unity

Figure 4.3 provides examples of the simulated unity environment using 3 different generation methods corresponding to the 3 different test scenarios as discussed in chapter 5. Each scenario contains box obstacles, but a difference can be seen in complexity due to the added walls. In the single-room scenario (Figure 4.3a) 8 boxes fill the room. In the multiple-room scenario (Figure 4.3b), the environment is divided into 4 rooms, containing a total of 4 boxes. The varied doorway scenario (Figure 4.3c), is similar to the prior scenario, but has doorways placed either low or high. We train our policy twice: one training is done in the multiple-room scenario and the other in the varied doorway scenario. The latter training is performed to evaluate our policy's 3d navigation capabilities.



(a) Single-room scenario **(b)** Multiple-room scenario **(c)** Varied doorway scenario

Figure 4.3: Examples of the Unity simulation for the different tested environments. All environments have an area of 10m $\times$ 10m with a height of 3m. For this visualisation, the ceiling is not added, but it is present during the actual experiments.

5 Experiments

This chapter presents the experiments and their results, demonstrating the effectiveness of our approach. Our method is evaluated in various scenarios and compared with an exploration baseline. First, we discuss the experimental setup and introduce the performance metrics used to evaluate our policy. We also outline the hypothesis testing conducted to determine the statistical significance of our findings.

Our experiments consist of two parts: A comparison with baseline methods and an ablation study. For the baseline comparison, we consider multiple environment types with varying degrees of complexity. This will show the strengths and weaknesses of the policies in relation to the characteristics of the environment. To validate our policy design, we ablate components from the observation space and reward function and measure the effect on the performance.

5.1. Experimental setup

In our experiments, we run our method and comparison methods in randomly generated environments for 200 episodes. These environments are different from the environments used for training, as different random seeds are used. However, the train and test environments are part of the same distribution of possible environments, since the same generation algorithm is used. To ensure a fair comparison between policies, the environments are generated using the same random seed between all experiments.

For the baseline comparison, we consider three different environment types as shown in chapter 4. The goal is to analyse the influence of the varying complexity on our method's performance. We discuss the expected challenges for each scenario.

- **Single room scenario:** Experiments from the single-room environment type will show whether the policies can effectively deal with occlusions from box obstacles and if the policy will be able to effectively avoid them.
- **Multi-room scenario:** Experiments from the cluttered four-room environment type will show if the policy has broader path-planning capabilities. With walled-off portions of the environment, the possible paths leading to the undiscovered parts of the environment are limited, thereby requiring more complex reasoning. Additionally, a greater portion of the environment will be occluded, resulting in more challenging exploration.
- **Varying doorway scenario:** The experiments from the environment with doorways at different heights will show better how capable the policies are in 3d navigation. In the other two environment types, a feasible exploration strategy is to move to a given height and fly horizontally for the rest of the task. However, with partially blocked doorways at different heights, the agent is required to move vertically to explore the entire environment.

In our experiments, each episode is run for a fixed number of 512 time steps, to measure the total reduction of belief map entropy given the total time budget. Success is defined the same as for training, i.e. achieving an average target belief map entropy below threshold $H_\mu(\mathbf{M}) < \beta$, with $\beta = 0.05$. We measure at which time step the agent reaches the threshold, to evaluate the agent's exploration efficiency.

Metrics

We measure performance using the following metrics:

- **Belief map entropy over time:** The reduction of belief map entropy is the main objective of the method and gives insight into how fast the agent can explore the environment.
- **Success rate:** The fraction of test episodes that reach the given entropy threshold.
- **Time steps to success:** The amount of time steps the agent takes to reach the entropy threshold. Episodes that do not result in success are not considered in this metric.
- **Invalid actions:** The number of invalid actions per episode indicates how well the policy reasons about obstacle avoidance.

Statistical significance tests

To test if the difference in performance between methods is statistically significant we make use of the Mann–Whitney U test and the binomial test. The Mann–Whitney U test is applicable for continuous variables, such as the residual entropy, steps to success and number of invalid actions per episode. The binomial test is applicable for binary data, such as the success rate. Both tests compute p-values that quantify the probability of obtaining test results at least as extreme as the measurements observed, given that the null hypothesis is true. A small p-value indicates that such an extreme test result would be unlikely under the null hypothesis. Commonly, a p-value lower than 0.05 is considered enough evidence to reject the null hypothesis in favour of the alternative. Appendix C provides more information on the statistical test methods.

The Mann-Whitney U test is required in our case as the obtained measurements from our experiments are not normally distributed. This test does not assume specific distribution types but only requires that the distributions are continuous and similarly shaped. [34, 16]

Specifically, the Mann–Whitney U test assesses the null hypothesis that the probability distribution of a randomly drawn observation from one group equals that of a randomly drawn observation from the other group against an alternative that those distributions are not equal [34]. Put simply, it measures the likelihood that any observed differences between two independent samples could have occurred by random chance. For our test, we formulate the null hypothesis as the equality of performance measurement distributions of our method and the compared methods. E.g. Under the null hypothesis, the distribution of steps to success, residual entropy and number of invalid actions are equal between methods. The alternate hypothesis is that the distribution underlying the performance of the compared method is stochastically worse than that of our method. E.g. Under the alternate hypotheses, the distributions of residual entropy, steps to success and number of invalid actions with the compared method are stochastically higher than that of our method.

We can describe the success rate as a Bernoulli experiment, in which case we can use the binomial test to determine the statistical significance. In this case, we formulate the null hypothesis as the equal probability for success between our method and the compared method. The alternate hypothesis is that the probability of a successful episode is higher with our method than with the compared method.

5.2. Baseline comparison

This section presents the baseline comparison experiments. First, the baseline methods and how they work are introduced. We present test results in three types of environments, to find out what environment characteristics are challenging for the policies.

Baseline methods

Similarly to the works [23, 2], we compare our method's performance to a myopic greedy policy, i.e. it chooses its actions in a one-step next-best-view manner. Each time step the next best action is computed using the current belief target map and the mutual information reward, resulting from the 6 possible actions. The greedy policy we define has privileged information about the obstacles in the environment, so it knows exactly where occlusions will occur and what actions will result in a collision. Using the privileged information information it will always choose the action that results in the highest one-step entropy reduction. We argue that this method will perform at least as well as any other myopic method that does not have privileged information, as estimation errors in the future mutual information can result in sub-optimal actions. Therefore, this baseline comparison will show if our policy can make coherent decisions over longer horizons.

Other planning-based methods, such as the works from [42, 12], are likely to outperform the greedy baseline, as they consider longer planning horizons. Such methods are therefore more relevant to compare. However, the implementation of these methods using our simulator setup was not feasible within the project's allotted time, so we only considered a myopic baseline.

Lastly, we evaluate exploration performance with a policy that randomly selects actions. This sets a lower bound for exploration policies that can reach success. Given enough time, randomly taking actions should allow the agent to fully explore the environment.

Qualitative results

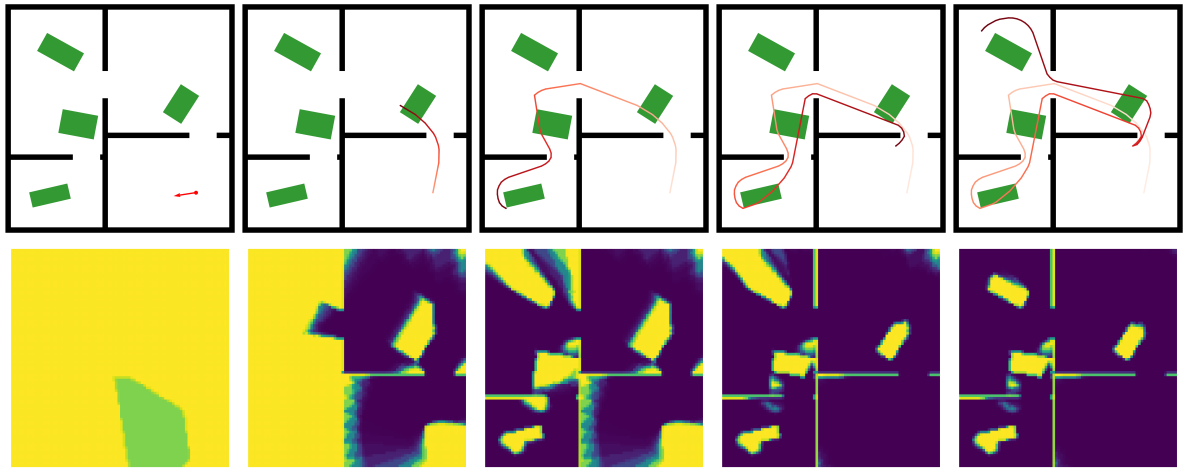
We present qualitative results through trajectory and belief map entropy figures over time, analysing the behaviours of the policies across different environments.

multi-room scenario

Figure 5.1 shows the trajectory throughout one episode in a cluttered 4 room scenario. Figure 5.1a shows the initial state, where only one target observation is gathered to update the entropy map. Figure 5.1b shows that since the start, the agent has turned right and navigated towards another room, leaving behind an unobserved section of the starting room. Figure 5.1c shows the agent progressing through all the rooms. In the last room it reaches, it navigates around the box obstacle to uncover an occluded area. Figure 5.1d shows that the agent fully backtracks to the starting room, where it observes the previously overlooked section. Furthermore, a small portion of cells occluded by boxes when moving one way, are observed on the return. Figure 5.1e shows that the agent navigates towards and around the box that occluded the last big unobserved section, finalising the episode. In the given trajectory the agent backtracks a long distance, which appears inefficient. However, given the agent's partial knowledge of the environment, this behaviour could be considered appropriate. If the starting room is connected to two or more rooms, opting for the first doorway in view can be an effective strategy. This scenario requires the agent to revisit the starting room to explore the entire environment, automatically observing it more in two instances. Lastly, while navigating close to boxes and doorways, small areas are overlooked, but the belief map entropy is reduced sufficiently.

Generally, in the cluttered single- and multi-room scenarios, our trained policy shows behaviour where the agent rises to a height close to the ceiling at the start of the episode. The policy has learned that at this height the observation range is bigger and fewer obstacles are in the way. From this point onward, the agent moves around the room and can fly over most box obstacles, allowing it to overcome occlusions of target cells caused by boxes.

Another noteworthy behaviour is that the agent often moves to a doorway, only to look around briefly and then return to where it came from. From a time-efficiency standpoint, it is logical to briefly look into



(a) $t = 0, H_\mu(\mathbf{M}) = 0.979$ (b) $t = 50, H_\mu(\mathbf{M}) = 0.504$ (c) $t = 150, H_\mu(\mathbf{M}) = 0.190$ (d) $t = 250, H_\mu(\mathbf{M}) = 0.061$ (e) $t = 350, H_\mu(\mathbf{M}) = 0.024$

Figure 5.1: Visualisation of the trajectory and entropy belief map over time in the 4-room scenario. The trajectory colour gradient transitions from lighter to darker shades representing earlier to later positions. In the belief map plots high to low entropy is represented with light to dark.

a room from its entrance and continue exploration. However, it occasionally misses large parts of the observable cells, when doing this. Depending on the situation, it can be inefficient behaviour, as the agent might need to return to a room multiple times for full coverage.

Our method exhibits two different failure cases. In case 1, the agent continually chooses an invalid action and does not show corrective behaviour, i.e. the agent is close to a wall or box and thinks it should be able to move forward. This results in the agent being stuck in this state and the episode being terminated due to the time-out. In case 2, the agent gets stuck moving back and forth within a limited space and never navigates to other parts of the environment. This occurs when doorways are poorly visible because the agent views them from a sharp angle, causing them to appear as a thin slice in the depth image. For both cases, the agent can exhibit corrective behaviour, i.e. after a certain extended time choosing an invalid action or circling the same area, it can redirect and continue navigating the rest of the environment.

Varied doorway scenario

Figure 5.2 shows the agent's trajectory in the varying doorway scenario. The agent effectively navigates between rooms, adjusting its height appropriately to match the doorway heights. This height adjustment mostly occurs at doorways, with the agent maintaining a similar height once within a room. Consequently, compared to the other scenario a lesser tendency to maintain a high position is seen. Still, the agent effectively resolves occlusions by navigating around the obstacles, which is more clearly shown in the top-down view in the figure.

Furthermore, the same types of stuck states occur more frequently, resulting from more complex environments. Instances of becoming stuck choosing invalid actions are more common because the scenario requires the agent to lower when the doorway is low. In case a box obstacle is near the doorway, the agent can get stuck trying to move down. This is not an issue in the other scenarios, where no vertical movement is required. Additionally, the smaller size of doorways results in worse visibility, thereby increasing the probability of the agent not finding the door and getting stuck moving around in the same confined area.

Privileged greedy

We also consider the privileged greedy policy's behaviour. Because the greedy policy does not consider future rewards beyond one step, the agent often goes to parts of the environment that are limited in view and mobility. The agent often goes into corners, where it stays until the local belief entropy is so

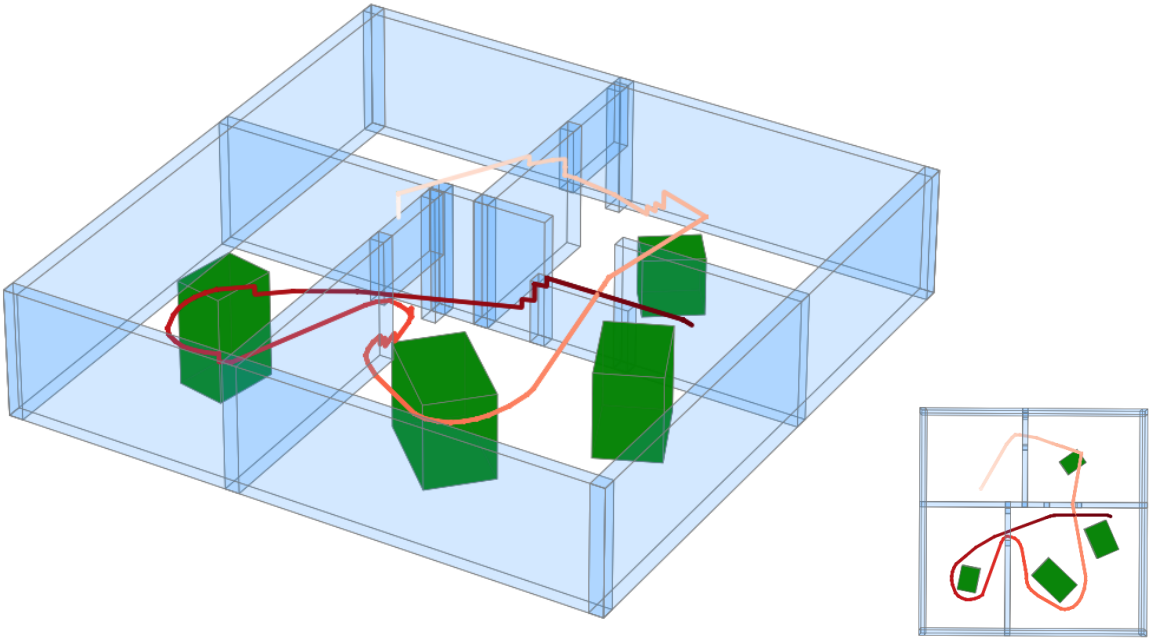


Figure 5.2: 3D visualisation of the trajectory and entropy belief map over time in the varied doorway height scenario. The colour gradient transitions from lighter to darker shades to represent points earlier and later in the trajectory. The right figure gives a top-down view of the trajectory and shows it going around the boxes.

low, that it turns around. This is inefficient behaviour, as much more entropy reduction can be achieved in the same time frame, by moving to regions of high entropy immediately. As a consequence of the increased environment complexity of the multi-room scenario, the probability of the policy guiding the agent to a corner or other areas with limited visibility only increases, which can be seen in the performance metrics.

Quantitative results

Figure 5.3 shows the mean belief map entropy and the percentage of successful episodes over time for three methods tested in three different environment types over 200 episodes. Figure 5.3a shows the baseline comparison of entropy reduction for the single-room scenario. Both greedy and RL methods reduce the mean belief map entropy below the specified threshold, leading to comparable success rates. However, as seen in Figure 5.3b, the speed of entropy reduction varies between the methods. By time step 250, our policy nearly achieves a 100% success rate, while the greedy method only reaches this at the time limit, indicating our policy's superior efficiency. The environment's relative simplicity allows the random policy to reduce the average entropy to 0.6, which is not good enough to reach any successful episode. Clearly, both the greedy and RL policies outperform random actions.

Figure 5.3c show that, on average, our policy achieves the target entropy level, whereas the greedy method only reduces the entropy to a residual value of approximately 0.15. The performance range of all policies is greater in the four-room scenario than in the single-room scenario, as indicated by the broader shaded interval. The introduction of more occlusions due to the added walls results in less consistent entropy reduction. Figure 5.3d shows that our policy maintains a success rate near 100%. In contrast, the success rate of the greedy method significantly drops compared to the single-room setting. The additional environmental structures, demonstrate the limitations of one-step-greedy methods in more complex scenarios. At the time limit, the greedy policy's success rate is around 30%, which our policy reaches approximately 300 steps earlier, showing the significant difference in exploration efficiency.

Figure 5.3e shows that in the varying doorway scenario our policy, on average, reaches the entropy threshold, whereas the greedy method only manages an entropy level above 0.3. Relative to the four-

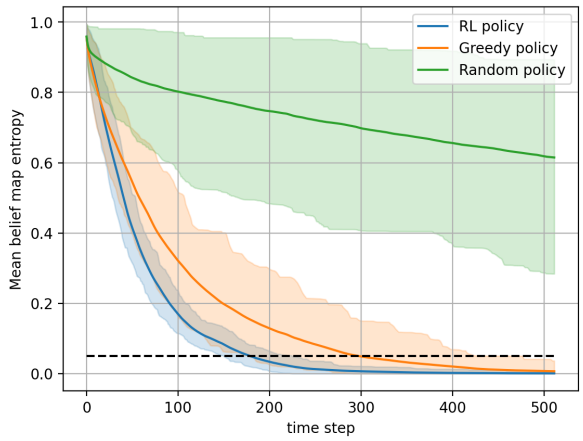
room scenario, our policy shows a bigger range in entropy reduction, indicating less consistency. The greedy method shows an even greater spread in performance with the 95th percentile value of residual entropy remaining high throughout the entire episode. This suggests that the policy may encounter scenarios where it gets stuck. Figure 5.3f shows a lower success rate for our policy compared to the multi-room scenario, suggesting that vertical movement is challenging with our approach. Nonetheless, the performance gap between our policy and the greedy method is greater compared with the multi-room scenario. The greedy method barely meets the entropy threshold and achieves a success rate of about 5%, while our policy achieves a success rate exceeding 90%.

		Success rate		Steps to success	Residual entropy
		t=256	t=512		
Single-room scenario	RL Policy (ours)	**0.98**	**1.00**	**180.4 ± 26.1**	**0.0008 ± 0.0024**
	Privileged Greedy	0.41	0.99	283.4 ± 84.6	0.0068 ± 0.0123
	Random	0.00	0.00	–	0.6148 ± 0.1846
Multiple-room scenario	RL Policy (ours)	**0.61**	**0.97**	**246.9 ± 48.8**	**0.0192 ± 0.0635**
	Privileged Greedy	0.04	0.29	369.8 ± 75.0	0.1638 ± 0.1429
	Random	0.00	0.00	–	0.7558 ± 0.1240
Varying doorway scenario	RL Policy (ours)	**0.11**	**0.92**	**309.1 ± 52.0**	**0.0346 ± 0.0958**
	Privileged Greedy	0.00	0.04	433.3 ± 44.0	0.3519 ± 0.2070
	Random	0.00	0.00	–	0.8103 ± 0.0904

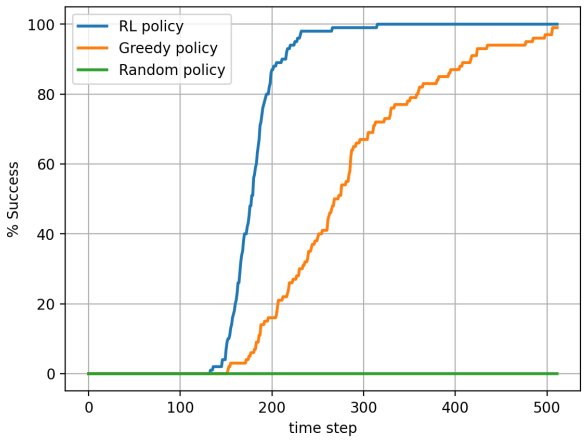
Table 5.1: Exploration performance metrics: policies are tested in 200 randomly generated environments for each test. For the steps to success and residual entropy metrics, the mean and standard deviation are provided.

Table 5.1 shows the methods and their performance averaged over 200 episodes according to the relevant metrics. First, our policy's success rate is significantly higher in the multiple-room and varying doorway experiments. Also, we see that at the time limit, on average, our policy has close to an order of magnitude less residual entropy remaining. This is a significant difference, especially considering as entropy decreases, decreasing entropy further is more difficult, as unexplored parts of the environment get more sparse. Also, in all scenarios, the variance in the residual entropy is less with our policy than with the privileged greedy policy, indicating that our policy exhibits more consistent behaviour. The random policy indicates the lower bound of the entropy decrease, which both methods significantly outperform.

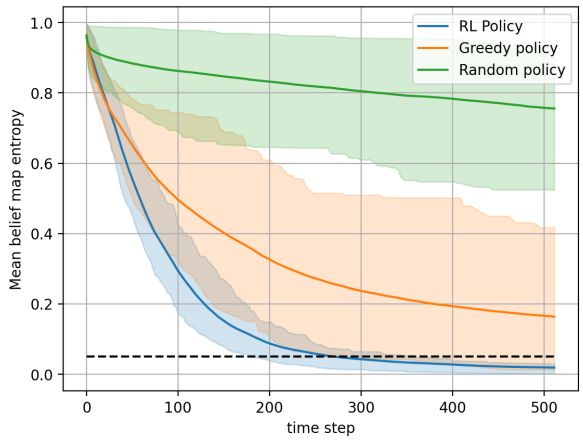
The time steps to success metric is computed using only the episodes that reached the entropy threshold. In all scenarios, our policy is on average at least 1.5 times faster in reaching the entropy threshold. It should be noted that this metric is skewed because the episodes are terminated for time-out. If the time-out was longer, episodes that are not considered in this metric, because of time-out, might have finished. Thereby increasing the average steps until success. In the case of our method, the average time to success is more accurate, as the success rate is higher. Overall, as the environment complexity increases the gap in exploration performance between the greedy and our policy increases, which is seen in the resulting steps to success and residual entropy metrics.



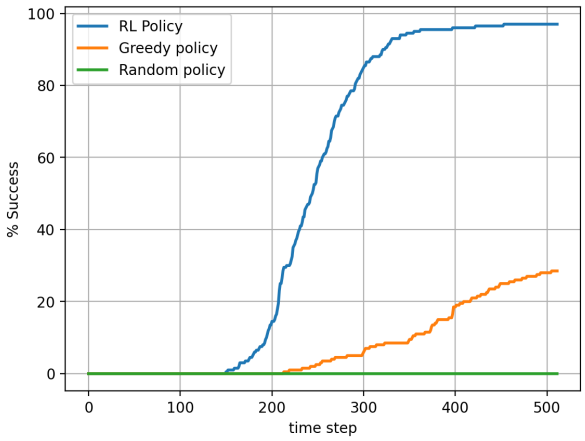
(a) Entropy reduction in the single-room scenario



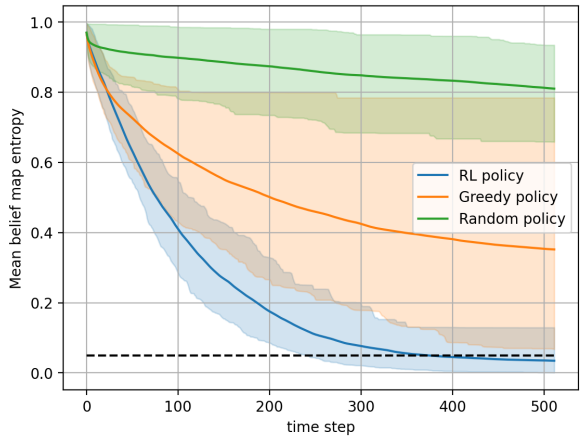
(b) success rate in the single-room scenario



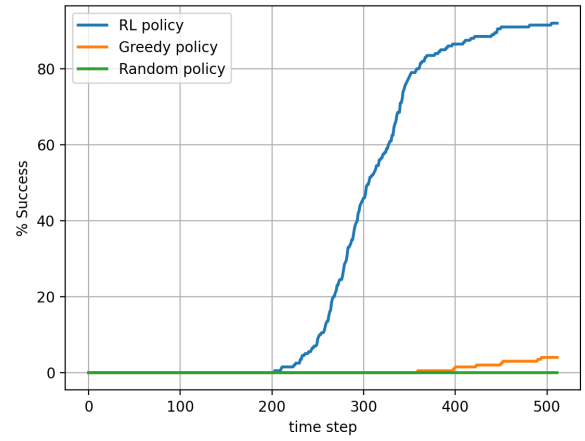
(c) Entropy reduction in the multiple-room scenario



(d) success rate in the multiple-room scenario



(e) Entropy reduction in the varying doorway scenario



(f) success rate in the varying doorway scenario

Figure 5.3: Exploration performance: policies are tested on 200 random environments. Each row shows the performance for a different environment type. The **left** column shows the mean belief map entropy as a function of time steps. Mean belief map entropy is computed considering only the cells that are not covered by obstacles. The area of each curve represents the 5 to 95 percentile interval. The **right** column shows the percentage of episodes that are successful as a function of time steps. An episode is considered a success when the mean belief map entropy $H_\mu(\mathbf{M}) < 0.05$

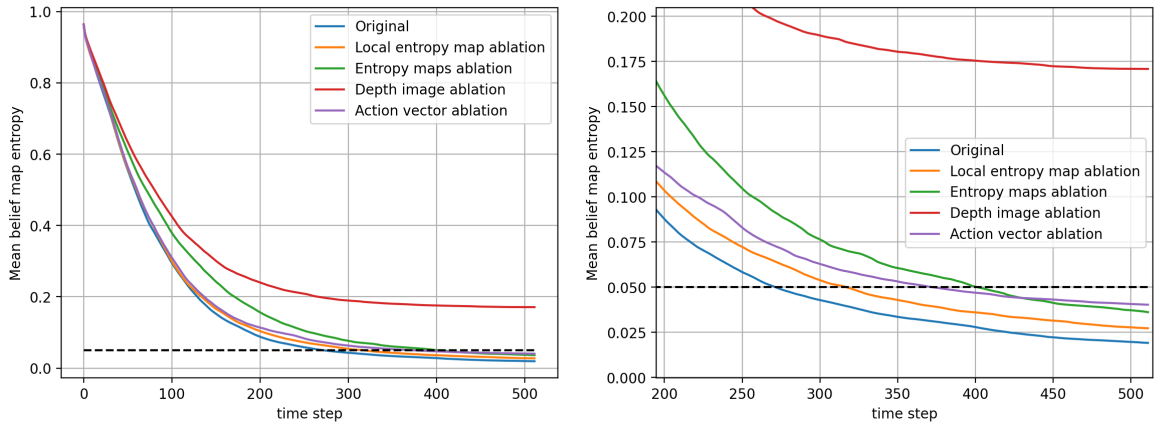
5.3. Ablation study

We conduct an ablation study to evaluate the impact of various elements within the observation space and reward design to motivate our policy design and training procedure. Also, it will clarify the functionality of each component. The ablation study exclusively considers the four-room exploration scenario, which corresponds to the environment types used during the training of the policies. To ensure a fair comparison, the policies are trained using the exact same training procedure as outlined in chapter 4.

We consider four observation ablations, that cover the whole observation space design. The first two focus on the belief entropy map observation. First, we ablate both local and global belief entropy maps to determine their contribution to the agent's exploration performance in comparison to the depth image observation. Second, we ablate only the local map to evaluate the added value of a detailed local map. Third, we ablate the depth images to understand their role in both collision avoidance and exploration. Fourthly, we ablate the action vector. Finally, we conduct two reward ablations focusing on collision and time penalties. The penalties aim to minimise collisions and time steps to success, respectively.

5.3.1. Observation space ablations

We present the findings from the observation space ablation study, where Figure 5.4 shows the average entropy of the belief map over time. The results indicate that all ablations negatively affect performance, but removing the depth image from the observation space has the biggest negative effect on performance, resulting in the highest residual entropy. In contrast, the average residual entropy for both entropy map ablations and the action vector ablations reaches the entropy threshold. It can be concluded that depth image observations play a crucial role in target search due to their representation of geometries. These are essential for collision-free navigation and identifying occlusions, without which the agent lacks direct information about the location of obstacles within the environment.



(a) Full time duration

(b) Zoomed in

Figure 5.4: Observation ablation study results: mean belief map entropy as a function of time steps over 200 test runs in random environment instances of 4 rooms containing 4 boxes.

Figure 5.5 depicts the success rates of episodes over time and shows the impact of different ablations more clearly. All ablations reduce the rate at which the agent successfully completes exploration. Notably, even though the success rate for the local entropy map ablation matches that of the original policy, it shows a reduced speed in reaching success. This suggests that the increased detail of the local entropy helps the policy to more precisely choose its actions.

Table 5.2 reaffirms that the success rates of the original policy and the local entropy map ablation are the same, with the original policy requiring fewer steps on average to complete exploration. Furthermore, all ablation policies achieve less entropy reduction compared to the original policy. Table 5.3 demonstrates statistical significance in the differences for both the time steps and residual entropy metrics across all ablations, underscoring the contribution of each observation space component to time-efficient and

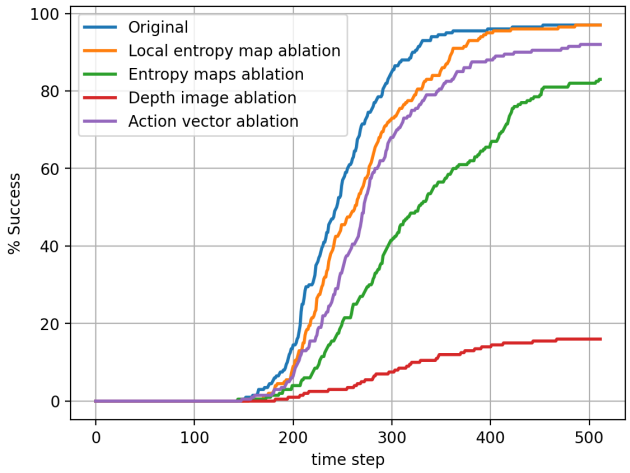


Figure 5.5: Observation ablation study results: the percentage of successful episodes as a function of time steps. An episode is considered a success when the mean belief map entropy $H_\mu(\mathbf{M}) < 0.05$

	Success rate		Steps to success	Residual entropy	Invalid actions
	t=256	t=512			
Original observation space	**0.61**	**0.97**	**246.9** $\pm$ **48.8**	**0.0192** $\pm$ **0.0635**	0.9 $\pm$ 9.4
Local entropy map ablation	0.38	**0.97**	267.8 $\pm$ 59.5	0.0273 $\pm$ 0.0619	**0.1** $\pm$ **0.4**
Entropy maps ablation	0.22	0.83	315.1 $\pm$ 78.3	0.0361 $\pm$ 0.0398	0.3 $\pm$ 0.7
Depth image ablation	0.04	0.16	309.1 $\pm$ 71.3	0.1709 $\pm$ 0.1385	2.6 $\pm$ 2.6
Action vector ablation	0.38	0.92	275.0 $\pm$ 60.9	0.0403 $\pm$ 0.1011	1.7 $\pm$ 9.7

Table 5.2: Exploration performance metrics: policies are tested in 200 randomly generated environments for each ablation. For the steps to success, residual entropy, and invalid action metrics, the mean and standard deviation are provided.

complete exploration. We note that the worse success rate performance is only statistically proven for the policies without any entropy maps and without depth image observations.

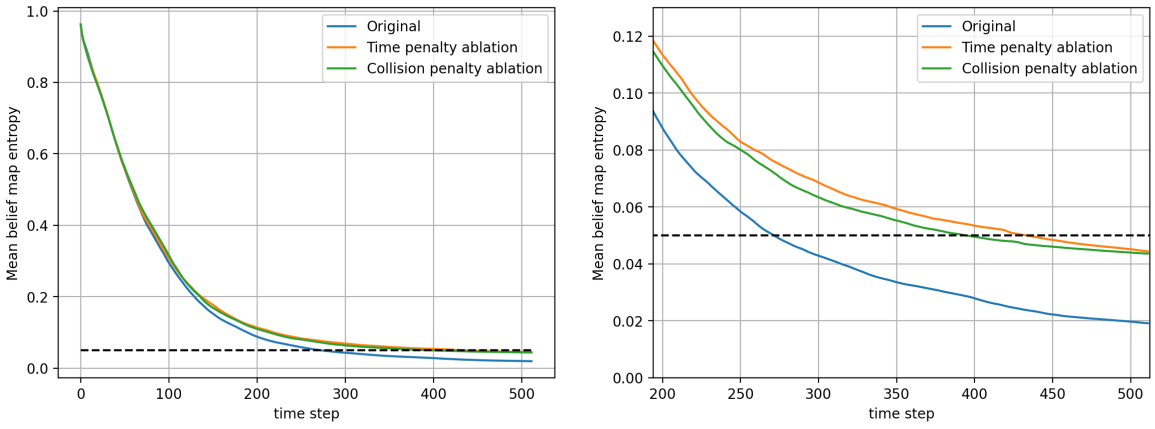
The invalid action results in Table 5.2 show that removing the local or both entropy maps results in fewer invalid actions per episode. We hypothesise that this is caused by the ambiguity of information from the belief entropy map. An area of high entropy in the target belief map can either mean that it is unexplored and that the agent should move there or that it is covered by an object and that the agent should move elsewhere. When removing the entropy maps, the agent can reason about collision avoidance using only the depth images, which more clearly represent if an action will lead to a collision. This theory is corroborated by the collision avoidance results from the depth image ablation, as here the collision rate is significantly higher still. Lastly, considering the results from Table 5.3, it can be concluded that the depth image and action vector observations are beneficial for collision avoidance performance.

	Binomial test	Mann–Whitney U test		
Original method comparison	Success rate p-value	Steps to success p-value	Residual entropy p-value	Invalid actions p-value
Local entropy map ablation	**0.557**	4.09e-4	3.46e-18	**0.951**
Entropy maps ablation	1.54e-3	4.86e-18	1.20e-22	**0.237**
Depth image ablation	2.98e-32	1.20e-6	1.17e-60	5.48e-39
Action vector ablation	**0.0866**	9.77e-7	4.04e-16	2.90e-13

Table 5.3: Statistical significance test results for the found metrics from the observation space ablation experiments. Bold numbers show the results that are not considered statistically significant.

5.3.2. Reward ablations

The reward design ablation study results are presented in Figure 5.6, illustrating the mean belief map entropy over time. While overall performance is comparable, it shows a small negative effect from both ablations.



(a) Full time duration

(b) Zoomed in

Figure 5.6: Reward ablation study results: mean belief map entropy as a function of time steps over 200 test runs in random environment instances of 4 rooms containing 4 boxes.

Figure 5.7 shows the percentage of successful episodes over time. The ablation of the time penalty reduces the agent's completion speed the most. Furthermore, we see a decrease in successful episodes, which is likely due to less effective exploration. The ablation of the collision penalty results in an even lower success rate and completion speed. This is likely caused by the agent choosing invalid actions more often, as this behaviour is not punished as much during training.

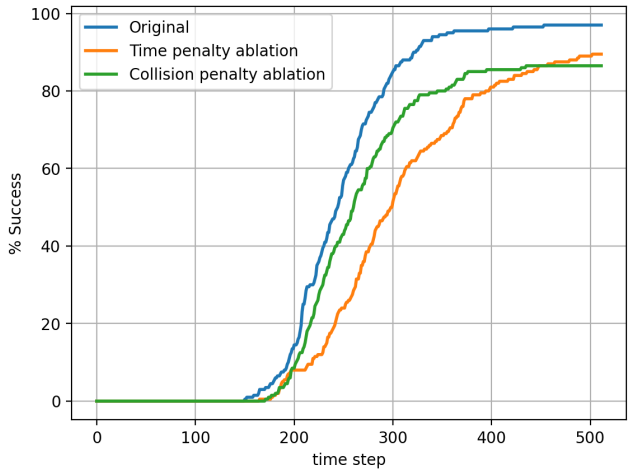


Figure 5.7: Reward ablation study results: the percentage of episodes that are successful as a function of time steps. An episode is considered a success when the mean belief map entropy $H_\mu(\mathbf{M}) < 0.05$

	Success rate		Steps to success	Residual entropy	Invalid actions
	t=256	t=512			
Original reward design	**0.61**	**0.97**	**246.9** $\pm$ **48.8**	**0.0192** $\pm$ **0.0635**	**0.9** $\pm$ **9.4**
Time penalty ablation	0.26	0.90	297.3 $\pm$ 71.6	0.0444 $\pm$ 0.0987	2.1 $\pm$ 13.3
Collision penalty ablation	0.47	0.87	257.6 $\pm$ 51.6	0.0436 $\pm$ 0.1037	4.1 $\pm$ 20.3

Table 5.4: Exploration performance metrics: policies are tested in 200 randomly generated environments for each ablation. For the steps to success, residual entropy, and invalid action metrics, the mean and standard deviation are provided.

Table 5.4 indicates that ablating the time penalty results in the policy being approximately 16% less efficient in reaching the entropy threshold. This, and the statistical test results presented in Table 5.5 gives strong evidence for the benefit of the time penalty on the time efficiency of the policy. This also results in improvements in both the success rate and the reduction of residual entropy.

Moreover, the ablation of the collision penalty increases both the average number and variance of invalid actions. The statistical tests validate that the collision penalty plays an important role in reducing invalid actions. As a consequence of the increased number of invalid actions the agent gets stuck more frequently and navigates the environment slower, which results in a worse residual entropy and success rate, as seen in Table 5.4

Interestingly, the time penalty ablation shows an increased number of invalid actions. In essence, the time penalty discourages the agent from taking useless inefficient actions. An invalid action is practically the most inefficient action, as it yields neither a state change nor information gain. Therefore invalid actions are also penalised by the time penalty, thereby reducing the number of invalid actions.

	Binomial test	Mann–Whitney U test		
Original method comparison	Success rate p-value	Steps to success p-value	Residual entropy p-value	Invalid actions p-value
Time penalty ablation	0.0301	1.84e-13	2.73e-33	0.0158
Collision penalty ablation	7.92e-3	0.0332	0.0138	2.41e-10

Table 5.5: Statistical significance test results for the found metrics from the reward ablation experiments.

6

Conclusion

The main goal of this research was to develop a vision-based reinforcement learning policy for quadcopters, that enables efficient 3D navigation for target search in cluttered environments. This involves navigating around obstacles to observe occluded areas, simultaneously taking obstacle avoidance into account. To analyse our method's performance a baseline comparison and an ablation study were performed, to validate our policy design. Our approach utilizes RGB-D and GPS sensor observations to maintain a probabilistic target belief state. Together with depth and pose information, the target belief map is used to inform a DRL policy to make navigational decisions. The policy is trained to choose from a discrete set of actions allowing 3D movement, optimising a mutual information reward and minimising time spent and actions that could lead to collisions.

Our experimental findings demonstrate the superior information-gathering performance of our policy over a privileged greedy baseline. A promising result is that as the environment complexity increases the performance gap between the greedy and our policy increases further. Our policy is better suited for complex cluttered environments, even though the greedy policy has privileged information about obstacles. Overall, our policy achieves a high success rate, so it is effective at fully exploring the environment. However, the policy is not flawless and occasionally selects globally inefficient actions. For example, this can happen when the agent has a small unobserved part nearby but opts to navigate the rest of the environment before having to return to the small unobserved region. Despite these shortcomings, our findings show the feasibility of learning 3D navigation policies for target search. Furthermore, we validate our policy design in our ablation study. All ablations result in less efficient target mapping shown by the time to success and residual entropy metrics.

The main contribution of this research is the following:

1. A deep reinforcement learning approach for probabilistic target mapping with quadcopters that provides 3D movement recommendations. Furthermore, the method for modelling target observations is provided.
2. The methodology for training the policy in simulation using a custom method for generating cluttered indoor environments. Additionally, the Flightmare simulator is modified to accommodate important features for training, such as dynamic object modification and collision checking.

Limitations and Future work

The ablation study reveals a limitation of our method. By removing one or both entropy maps from the observation space, the number of collision actions is greatly reduced. This is a result of the ambiguity of the belief entropy map observation. A high entropy region can mean either that it is unobserved or that an obstacle is present, in which case the agent cannot know if it should or should not navigate towards high entropy regions. Future work could add a geometric occupancy map observation, to overlay onto the entropy map, with which the right regions of interest in the target map can be deduced.

Furthermore, the action space design is limited and does not represent the full mobility of quadcopters. This simplifies the learning problem but limits the possible exploration behaviour which probably results in less efficient performance. The action space could be extended to include sideways movement and backwards movement, but this still limits the quadcopter's mobility. Real-life applications of autonomous quadcopter exploration must consider real quadcopter dynamics, so future work should focus on implementing a trajectory planner, to bring our method a step closer to real life.

Another limitation stems from the simulated environment. For our training and test simple environments were used. The complexity and realism of the environment should be improved if the policy is to used for real-life applications. Future work should thus focus on reducing the sim-to-real gap by simulating visually realistic environments, simulating sensor noise and applying domain randomisation.

References

[1] Reem Ashour et al. “Exploration for Object Mapping Guided by Environmental Semantics using UAVs”. In: *Remote Sensing* 12.5 (Mar. 2020), p. 891. issn: 20724292. doi: 10.3390/RS12050891.

[2] Shi Bai et al. “Inference-Enabled Information-Theoretic Exploration of Continuous Action Spaces”. In: (2018). doi: 10.1007/978-3-319-60916-4{_}24.

[3] Shi Bai et al. “Information-theoretic exploration with Bayesian optimization”. In: *IEEE International Conference on Intelligent Robots and Systems* 2016-November (Nov. 2016), pp. 1816–1822. issn: 21530866. doi: 10.1109/IROS.2016.7759289.

[4] Andreas Bircher et al. “Receding horizon next-best-view planner for 3D exploration”. In: *Proceedings - IEEE International Conference on Robotics and Automation* (June 2016), pp. 1462–1468. issn: 10504729. doi: 10.1109/ICRA.2016.7487281.

[5] John S. Bridle. “Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition”. In: *Neurocomputing* (1990), pp. 227–236. doi: 10.1007/978-3-642-76153-9{_}28. url: https://link.springer.com/chapter/10.1007/978-3-642-76153-9_28.

[6] Greg Brockman et al. “OpenAI Gym”. In: (June 2016). doi: 10.48550/arxiv.1606.01540. url: https://arxiv.org/abs/1606.01540v1.

[7] Mark Bruls, Kees Huizing, and Jarke J. van Wijk. “Squarified Treemaps”. In: (2000), pp. 33–42. doi: 10.1007/978-3-7091-6783-0{_}4. url: https://link.springer.com/chapter/10.1007/978-3-7091-6783-0_4.

[8] Chao Cao et al. “TARE: A Hierarchical Framework for Efficiently Exploring Complex 3D Environments”. In: (). url: www.cmu-exploration.com.

[9] Adrian Carrio et al. “UBRISTES: UAV-Based Building Rehabilitation with Visible and Thermal Infrared Remote Sensing”. In: *Advances in Intelligent Systems and Computing*. Vol. 417. Springer Verlag, 2016, pp. 245–256. isbn: 9783319271453. doi: 10.1007/978-3-319-27146-0.

[10] Devendra Singh Chaplot et al. “Learning to Explore using Active Neural SLAM”. In: (Apr. 2020). doi: 10.48550/arxiv.2004.05155.

[11] Devendra Singh Chaplot et al. “SEAL: Self-supervised Embodied Active Learning using Exploration and 3D Consistency”. In: *Advances in Neural Information Processing Systems* 16 (Dec. 2021), pp. 13086–13098. issn: 10495258. doi: 10.48550/arxiv.2112.01001.

[12] Benjamin Charrow et al. “Information-theoretic mapping using Cauchy-Schwarz Quadratic Mutual Information”. In: *Proceedings - IEEE International Conference on Robotics and Automation* 2015-June.June (June 2015), pp. 4791–4798. issn: 10504729. doi: 10.1109/ICRA.2015.7139865.

[13] Benjamin Charrow et al. “Information-Theoretic Planning with Trajectory Optimization for Dense 3D Mapping”. In: *Robotics: Science and Systems XI* 11 (2015). issn: 2330765X. doi: 10.15607/RSS.2015.XI.003.

[14] Fanfei Chen et al. “Self-learning exploration and mapping for mobile robots via deep reinforcement learning”. In: *AIAA Scitech 2019 Forum* (2019). doi: 10.2514/6.2019-0396.

[15] Tao Chen, Saurabh Gupta, and Abhinav Gupta. “Learning Exploration Policies for Navigation”. In: *7th International Conference on Learning Representations, ICLR 2019* (Mar. 2019). doi: 10.48550/arxiv.1903.01959.

[16] Cédric Colas, Olivier Sigaud, and Pierre Yves Oudeyer. “A Hitchhiker’s Guide to Statistical Comparisons of Reinforcement Learning Algorithms”. In: *RML@ICLR 2019 Workshop - Reproducibility in Machine Learning* (Apr. 2019). url: https://arxiv.org/abs/1904.06979v1.

[17] Thomas M. Cover and Joy A. Thomas. “Elements of Information Theory”. In: *Elements of Information Theory* (Apr. 2005), pp. 1–748. doi: 10.1002/047174882X.

[18] Alessandro Devo et al. “Autonomous Single-Image Drone Exploration with Deep Reinforcement Learning and Mixed Reality”. In: *IEEE Robotics and Automation Letters* (2022), p. 1. issn: 2377-3766. doi: 10.1109/LRA.2022.3154019.

[19] Christian Dornhege and Alexander Kleiner. “A frontier-void-based approach for autonomous exploration in 3d”. In: *9th IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2011* (2011), pp. 351–356. doi: 10.1109/SSRR.2011.6106778.

[20] Rui Pimentel de Figueiredo et al. “Real-Time Volumetric-Semantic Exploration and Mapping: An Uncertainty-Aware Approach”. In: (Sept. 2021). doi: 10.48550/arXiv.2109.01474.

[21] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. “Appearance-based active, monocular, dense reconstruction for micro aerial vehicles”. In: *Robotics: Science and Systems* (July 2014). doi: 10.15607/RSS.2014.X.029.

[22] Fukushima. “Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements”. In: *IEEE* 5.4 (1969), pp. 322–333. issn: 21682887. doi: 10.1109/TSSC.1969.300225.

[23] Harsh Goel et al. “Reinforcement Learning for Agile Active Target Sensing with a UAV”. In: (Dec. 2022). url: https://arxiv.org/abs/2212.08214v1.

[24] Saurabh Gupta et al. “Cognitive Mapping and Planning for Visual Navigation”. In: *International Journal of Computer Vision* 128.5 (Feb. 2017), pp. 1311–1330. issn: 15731405. doi: 10.1007/s11263-019-01236-7. url: https://arxiv.org/abs/1702.03920v3.

[25] Pieter Hintjens. “ZeroMQ: Messaging for Many Applications”. In: (2013), p. 493. url: https://books.google.com/books/about/ZeroMQ.html?hl=nl&id=KWtT5CJc6FYC.

[26] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. issn: 0899-7667. doi: 10.1162/NECO.1997.9.8.1735. url: https://dx.doi.org/10.1162/neco.1997.9.8.1735.

[27] Brian J. Julian, Sertac Karaman, and Daniela Rus. “On mutual information-based control of range sensing robots for mapping applications”. In: *IEEE International Conference on Intelligent Robots and Systems* (2013), pp. 5156–5163. issn: 21530858. doi: 10.1109/IROS.2013.6697102.

[28] Sertac Karaman and Emilio Frazzoli. “Sampling-based Algorithms for Optimal Motion Planning”. In: (May 2011). doi: 10.48550/arxiv.1105.1186.

[29] Srivatsan Krishnan et al. “Air Learning: a deep reinforcement learning gym for autonomous aerial robot visual navigation”. In: *Machine Learning* 110.9 (Sept. 2021), pp. 2501–2540. issn: 15730565. doi: 10.1007/S10994-021-06006-6/FIGURES/16. url: https://link.springer.com/article/10.1007/s10994-021-06006-6.

[30] Mikko Lauri, David Hsu, and Joni Pajarinen. “Partially Observable Markov Decision Processes in Robotics: A Survey”. In: *IEEE Transactions on Robotics* 39.1 (Sept. 2022), pp. 21–40. doi: 10.1109/TRO.2022.3200138. url: http://arxiv.org/abs/2209.10342%20http://dx.doi.org/10.1109/TRO.2022.3200138.

[31] Max Lodel et al. “Where to Look Next: Learning Viewpoint Recommendations for Informative Trajectory Planning”. In: *Proceedings - IEEE International Conference on Robotics and Automation* (2022), pp. 4466–4472. issn: 10504729. doi: 10.1109/ICRA46639.2022.9812190.

[32] Giuseppe Loianno et al. “Estimation, Control, and Planning for Aggressive Flight with a Small Quadrotor with a Single Camera and IMU”. In: *IEEE Robotics and Automation Letters* 2.2 (Apr. 2017), pp. 404–411. issn: 23773766. doi: 10.1109/LRA.2016.2633290.

[33] Liang Lu, A Martínez Novo, and Pascual Campoy. “FAST RRT* 3D-sliced planner for autonomous exploration using MAVs”. In: *Unmanned Systems* 10.2 (2022), pp. 175–186. doi: 10.1142/S2301385022500108.

[34] H. B. Mann and D. R. Whitney. “On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other”. In: *https://doi.org/10.1214/aoms/1177730491* 18.1 (Mar. 1947), pp. 50–60. issn: 0003-4851. doi: 10.1214/AOMS/1177730491.

[35] Fernando Marson and Soraia Raupp Musse. “Automatic real-time generation of floor plans based on squarified treemaps algorithm”. In: *International Journal of Computer Games Technology* (2010). issn: 16877047. doi: 10.1155/2010/624817. url: https://www.researchgate.net/publication/47696530_Automatic_Real-Time_Generation_of_Floor_Plans_Based_on_Squarified_Treemaps_Algorithm.

[36] Carol Martinez et al. “Towards autonomous detection and tracking of electric towers for aerial power line inspection”. In: *ICoUAS* (2014), pp. 284–295. doi: 10.1109/ICUAS.2014.6842267.

[37] Ajith Anil Meera et al. “Obstacle-aware Adaptive Informative Path Planning for UAV-based Target Search”. In: *Proceedings - IEEE International Conference on Robotics and Automation* (Feb. 2019), pp. 718–724. issn: 10504729. doi: 10.1109/ICRA.2019.8794345.

[38] Maysam Mirahmadi and Abdallah Shami. “A Novel Algorithm for Real-time Procedural Generation of Building Floor Plans”. In: (Nov. 2012). doi: 10.48550/arxiv.1211.5842. url: https://arxiv.org/abs/1211.5842v1.

[39] Volodymyr Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: (Dec. 2013). url: https://arxiv.org/abs/1312.5602v1.

[40] Farzad Niroui et al. “Deep Reinforcement Learning Robot for Search and Rescue Applications: Exploration in Unknown Cluttered Environments”. In: *IEEE Robotics and Automation Letters* 4.2 (Apr. 2019), pp. 610–617. issn: 23773766. doi: 10.1109/LRA.2019.2891991.

[41] Stephen Nuske et al. “Yield estimation in vineyards by visual grape detection”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems* (Dec. 2011), pp. 2352–2358. doi: 10.1109/IROS.2011.6095069.

[42] Helen Oleynikova et al. “Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning”. In: *IEEE International Conference* 2017-September (Dec. 2017), pp. 1366–1373. issn: 21530866. doi: 10.1109/IROS.2017.8202315.

[43] Miguel A. Olivares-Mendez et al. “Towards an Autonomous Vision-Based Unmanned Aerial System against Wildlife Poachers”. In: *Sensors* 15.12 (Dec. 2015), pp. 31362–31391. issn: 1424-8220. doi: 10.3390/S151229861.

[44] Fabio Pardo et al. “Time Limits in Reinforcement Learning”. In: *35th International Conference on Machine Learning, ICML 2018* 9 (Dec. 2017), pp. 6443–6452. url: https://arxiv.org/abs/1712.00378v4.

[45] Deepak Pathak et al. “Curiosity-driven Exploration by Self-supervised Prediction”. In: *34th International Conference on Machine Learning* 6 (May 2017), pp. 4261–4270. doi: 10.48550/arxiv.1705.05363.

[46] Marija Popovic et al. “Online Informative Path Planning for Active Classification Using UAVs”. In: *Proceedings - IEEE International Conference on Robotics and Automation* (Sept. 2016), pp. 5753–5758. issn: 10504729. doi: 10.1109/ICRA.2017.7989676.

[47] Marija Popović et al. “An informative path planning framework for UAV-based terrain monitoring”. In: *Autonomous Robots* 44.6 (July 2020), pp. 889–911. issn: 15737527. doi: 10.1007/S10514-020-09903-2/FIGURES/17. url: https://link.springer.com/article/10.1007/s10514-020-09903-2.

[48] *Pybind11 documentation*. url: https://pybind11.readthedocs.io/en/stable/basics.html.

[49] RaffinAntonin et al. “Stable-baselines3”. In: *The Journal of Machine Learning Research* 22 (Jan. 2021), pp. 1–8. doi: 10.5555/3546258.3546526. url: https://dl.acm.org/doi/10.5555/3546258.3546526.

[50] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2016-December (June 2015), pp. 779–788. issn: 10636919. doi: 10.1109/CVPR.2016.91. url: https://arxiv.org/abs/1506.02640v5.

[51] Fereshteh Sadeghi and Sergey Levine. “CAD2RL: Real Single-Image Flight without a Single Real Image”. In: *Robotics: Science and Systems* 13 (Nov. 2016). issn: 2330765X. doi: 10.15607/rss.2017.xiii.034.

[52] Carlos Sampedro et al. “A Fully-Autonomous Aerial Robot for Search and Rescue Applications in Indoor Environments using Learning-Based Techniques”. In: *Journal of Intelligent & Robotic Systems* 95 (2019), pp. 601–627. doi: 10.1007/s10846-018-0898-1.

[53] John Schulman et al. “Proximal Policy Optimization Algorithms”. In: (July 2017). url: https://arxiv.org/abs/1707.06347v2.

[54] Sang Yun Shin, Yong Won Kang, and Yong Guk Kim. “Automatic drone navigation in realistic 3d landscapes using deep reinforcement learning”. In: *6th International Conference on Control, Decision and Information Technologies* (Apr. 2019), pp. 1072–1077. doi: 10.1109/CODIT.2019.8820322.

[55] Yunlong Song et al. “Flightmare: A Flexible Quadrotor Simulator”. In: (Sept. 2020). doi: 10.48550/arxiv.2009.00563.

[56] Sebastian Thrun. “Probabilistic robotics”. In: *Communications of the ACM* 45.3 (2002), pp. 52–57. issn: 00010782. doi: 10.1145/504729.504754.

[57] Josh Tobin et al. “Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World”. In: *IEEE International Conference on Intelligent Robots and Systems* 2017-September (Mar. 2017), pp. 23–30. issn: 21530866. doi: 10.1109/IROS.2017.8202133. url: https://arxiv.org/abs/1703.06907v1.

[58] *Unity Real-Time Development Platform | 3D, 2D VR & AR Engine*. url: https://unity.com/.

[59] Martijn Van Otterlo and Marco Wiering. “Reinforcement learning and markov decision processes”. In: *Adaptation, Learning, and Optimization* 12 (2012), pp. 3–42. issn: 18674542. doi: 10.1007/978-3-642-27645-3{_}1/COVER. url: https://link.springer.com/chapter/10.1007/978-3-642-27645-3_1.

[60] Erik Wijmans et al. “DD-PPO: Learning Near-Perfect PointGoal Navigators from 2.5 Billion Frames”. In: (Nov. 2019). url: https://arxiv.org/abs/1911.00357v2.

[61] Chunxue Wu et al. “UAV autonomous target search based on deep reinforcement learning in complex disaster scene”. In: *IEEE Access* 7 (2019), pp. 117227–117245. issn: 21693536. doi: 10.1109/ACCESS.2019.2933002.

[62] Gang Xu and Zhengyou Zhang. “Epipolar Geometry in Stereo, Motion and Object Recognition”. In: Computational Imaging and Vision 6 (1996). doi: 10.1007/978-94-015-8668-9. url: http://link.springer.com/10.1007/978-94-015-8668-9.

[63] Brian Yamauchi. “Frontier-based approach for autonomous exploration”. In: *Proceedings of IEEE* (1997), pp. 146–151. doi: 10.1109/CIRA.1997.613851.

[64] Chenghai Yang. “A high-resolution airborne four-camera imaging system for agricultural remote sensing”. In: *Computers and Electronics in Agriculture* 88 (Oct. 2012), pp. 13–24. issn: 0168-1699. doi: 10.1016/J.COMPAG.2012.07.003.

[65] Boyu Zhou et al. “FUEL: Fast UAV Exploration using Incremental Frontier Structure and Hierarchical Planning”. In: ().

A

Additional Results

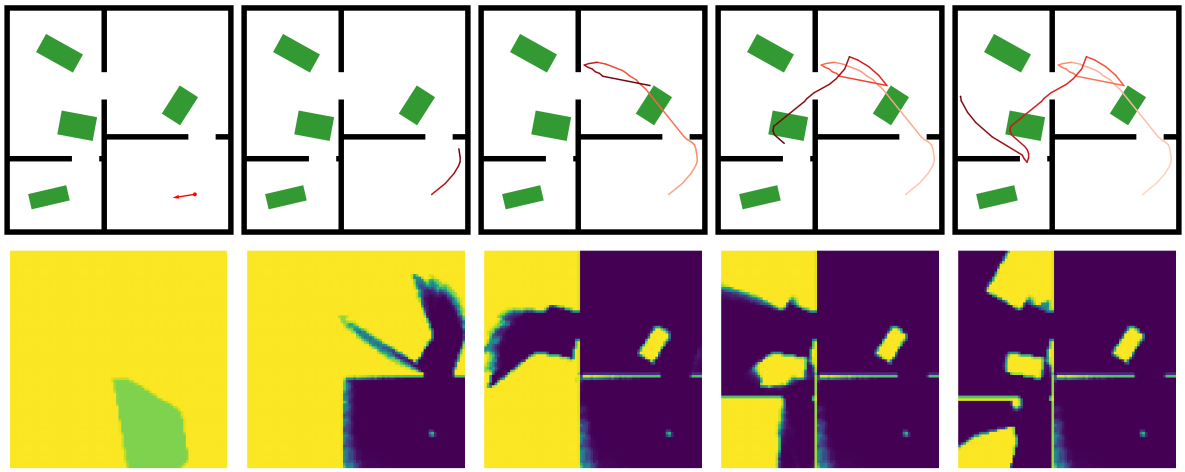
A.1. Statistical tests baseline comparison

		Binomial test	Mann–Whitney U test		
		Success rate p-value	Steps to success p-value	Residual entropy p-value	Invalid actions p-value
Single-room scenario	Greedy	0.394	2.14e-23	6.49e-06	–
	Random	7.89e-31	–	3.15e-37	6.80e-38
Multiple-room scenario	Greedy	8.99e-23	2.16e-20	3.89e-52-	–
	Random	2.30e-58	–	1.43e-66	2.85e-71
Varying doorway scenario	Greedy	4.83e-44	5.57e-06	1.26-57	–
	Random	3.01e-54	–	9.73e-67	2.31e-67

Table A.1: Statistical significance test results for the found metrics from the method comparison experiments.

Shown in Table A.1 are the p-values for all hypothesis tests, as formulated in section 5.1. We conclude that all but one measurement in exploration performance is statistically significant. Only the success rate in the single cluttered room environment is similar enough to not prove our policy's better performance. In all other cases, our policy is statistically better than the greedy policy.

A.2. Qualitative results greedy policy



(a) $t = 0, H_\mu(\mathbf{M}) = 0.979$ **(b)** $t = 100, H_\mu(\mathbf{M}) = 0.676$ **(c)** $t = 250, H_\mu(\mathbf{M}) = 0.328$ **(d)** $t = 400, H_\mu(\mathbf{M}) = 0.199$ **(e)** $t = 512, H_\mu(\mathbf{M}) = 0.102$

Figure A.1: Visualisation of the trajectory and entropy belief map over time in the 4-room scenario. The trajectory colour gradient transitions from lighter to darker shades representing earlier to later positions. In the belief map plots high to low entropy is represented with light to dark.

B

Environment generation details

B.1. Random box generation

Algorithm 2 Box Placement Algorithm

$n\_boxes \leftarrow$ number of boxes
$all\_poses \leftarrow$ empty list
for $i \leftarrow 1$ to n_boxes **do**
    for $j \leftarrow 1$ to 100 **do**
        $dims \leftarrow$ uniformly sampled random dimensions within box dimensions range
        $radius\_new \leftarrow \sqrt{dims[x]^2 + dims[y]^2}/2$
        $pos\_new \leftarrow$ get_random_location($radius\_new$)
        $pos\_valid \leftarrow$ **True**
        for all $pose$ in all_poses **do**
            $dist \leftarrow$ distance between pos_new and $pose[pos]$
            if $dist \leq pose[radius] + radius\_new$ **then**
                $pos\_valid \leftarrow$ **False**
                break
            end if
        end for
        if pos_valid = **True then**
            $angle \leftarrow$ uniformly sampled random angle
            Add pos, $angle$ and $radius$ to all_poses
            break
        end if
    end for
end for

B.2. Random location generation

Algorithm 3 Generate Random Location

procedure get_random_location($margin$)
  for $i \leftarrow 1$ to 100 **do**
    $pos \leftarrow$ uniformly sampled random 2d point within floor plan bounds
    for all $room$ in $rooms$ **do**
      if $room["x1"] + margin < pos[0] < room["x2"] - margin$ **and** $room["y1"] + margin < pos[1] < room["y2"] - margin$ **then** ▷ Check if point falls within a room and its margin
        return pos
      end if
    end for
  end for
end procedure

B.3. Random drone pose generation

Algorithm 4 Generate Random Drone Pose

procedure get_random_drone_pose($margin$)
  $objects \leftarrow$ list of objects in environment
  for $i \leftarrow 1$ to 100 **do**
    $pos\_valid \leftarrow True$
    $pos\_new[x, y] \leftarrow$ get_random_location($margin$)
    $pos\_new[z] \leftarrow$ uniformly sampled height within environment dimensions
    for all $object$ in $objects$ **do**
      $dist \leftarrow$ distance between pos_new and $object[pos]$
      if $dist \leq margin + object[radius]$ **then**
        $pos\_valid \leftarrow False$
        break
      end if
    end for
    if pos_valid = **True then**
      break
    end if
  end for
  $angle \leftarrow$ uniformly sampled random angle
  return pos_new and $angle$
end procedure

C

Statistical Testing

C.1. Mann-whitney U test

The Mann-Whitney U test is a non-parametric test used to compare differences between two independent groups, focusing on the ranks of observations rather than their actual values. The U statistic is calculated by ranking all observations from both groups together and then summing the ranks for each group. The U statistic for each group can be calculated as follows:

$$U_1 = R_1 - \frac{n_1(n_1+1)}{2}$$

$$U_2 = R_2 - \frac{n_2(n_2+1)}{2}$$

where R_1 and R_2 are the sum of ranks for Group 1 and Group 2, respectively, and n_1 and n_2 are the sample sizes.

For large sample sizes, the Z-score is used to approximate the normal distribution of U, calculated by:

$$Z = \frac{U - \mu_U}{\sigma_U}$$

where $\mu_U = \frac{n_1 n_2}{2}$ and $\sigma_U = \sqrt{\frac{n_1 n_2 (n_1+n_2+1)}{12}}$. The p-value is then determined from the Z-score using the standard normal distribution.

The Z-score in statistics measures how many standard deviations an element is from the mean of the distribution. It is used in hypothesis testing to compare against a critical value to decide on the null hypothesis. A high absolute Z-score indicates a significant result unlikely under the null hypothesis.

The p-value in the Mann-Whitney U test can be computed using pre-calculated tables for small sample sizes or approximated by the Z-score for larger samples. The p-value indicates the probability of observing a difference as extreme as the measured one (or more extreme) under the null hypothesis of no difference between the groups.

C.2. Binomial Test

The Binomial Test is a statistical procedure used to determine if the proportion of successes in a binary outcome experiment differs from a specified value. It is applicable when the data can be categorized into two mutually exclusive categories (success/failure) and the observations are independent.

Given a sample size n, a number of observed successes k, and the probability of success under the null hypothesis p_0, the test evaluates the probability of observing k or more extreme successes. The p-value is calculated based on the binomial distribution:

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

where $P(X = k)$ is the probability of observing exactly k successes out of n trials, and $\binom{n}{k}$ is the binomial coefficient.

The p-value for the binomial test can be obtained by summing the probabilities of outcomes as extreme as or more extreme than the observed result, under the assumption that the null hypothesis is true. For a two-tailed test, this involves calculating the probabilities of outcomes at both ends of the distribution:

$$p\text{-value} = \sum_{x=0}^{k} P(X = x) + \sum_{x=k}^{n} P(X = x)$$

where the first sum accounts for outcomes as extreme as or more extreme on the lower end, and the second sum accounts for the upper end, based on the observed value k.

The decision to reject or not reject the null hypothesis is made by comparing the p-value to a predetermined significance level α. A p-value less than α indicates that the observed proportion of successes is significantly different from the expected proportion under the null hypothesis, leading to the rejection of the null hypothesis.