Aerial View Image-Goal Localization with Reinforcement Learning

John Backsund, Anton Samuelsson

Master's thesis
2022:E30

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Aerial View Image-Goal Localization with Reinforcement Learning

John Backsund and Anton Samuelsson

Supervisor
Aleksis Pirinen (RISE)

Supervisor
Kalle Åström (LTH)

Examiner
Alexandros Sopasakis (LTH)

June 14, 2022

Abstract

With an increased amount and availability of unmanned aerial vehicles (UAVs) and other remote sensing devices (e.g. satellites), we have recently seen an explosion in computer vision methodologies, tailored towards processing and understanding aerial view data. One application for such technologies is in the area of search-and-rescue (SAR), where the task is to localize and assist one or several people who are missing, for example after a natural disaster. In many cases the rough location may be known and a UAV can be deployed to explore a given, confined area to precisely localize the missing people. In such a time- and resource-constrained setting, controlling the UAV in an informed and intelligent manner – as opposed to exhaustively scanning the whole area along a pre-defined trajectory – could significantly improve the likelihood of succeeding with the mission. In this master's thesis we approach this type of problem by abstracting it as an *aerial view image-goal localization* task within a framework that emulates a SAR-like setup without requiring access to actual UAVs. In this framework, an agent operates on top of a given satellite image and is tasked with localizing a specific goal, specified as a rectangular region within the satellite image, from a given location in the image. The agent is never allowed to observe the underlying satellite image in its entirety, not even at low resolution, and thus it has to operate solely based on sequentially observed partial glimpses when navigating towards the goal location. To tackle our suggested aerial view image-goal localization task, we propose *AiRLoc*, a fully trainable reinforcement learning (RL)-based model. AiRLoc can be trained with no annotations of any kind and is hence able to learn the localization task in an entirely self-supervised manner. Extensive experimental results suggest that AiRLoc outperforms heuristic search methods as well as non-RL-based machine learning methods. The results also indicate that providing AiRLoc with mid-level vision capabilities (specifically, a pre-trained semantic segmentation network) can lead to even better performance. We also conduct a proof-of-concept study which suggests that AiRLoc – with or without semantic segmentation as input – outperforms humans on average.

Acknowledgements

Firstly, we would like to thank our supervisor Aleksis Pirinen (RISE) who has proved a constant source of inspiration and encouragement. Your extensive knowledge of the field and your willingness to share it with us has been instrumental for our understanding and for the completion of this thesis. Additionally, we would like to thank our supervisor from Lund University, Kalle Åström who has provided insights and direction during integral parts of the project. Additionally, we got the opportunity to present our work to the Intelligent Systems group at RISE, thank you for the valuable feedback. Finally, we want to thank everyone who volunteered to take part in the study of the human skill level at the proposed task. Your contributions enabled an interesting comparison between human and AI capabilities.

Contents

1 Introduction **1**

1.1 Limitations . . . 2

1.2 Ethical Considerations . . . 3

1.3 Related Work . . . 3

2 Background **6**

2.1 Machine Learning . . . 6

2.2 Neural Networks . . . 7

2.2.1 Fully Connected Layers . . . 7

2.2.2 Convolutional Layers . . . 8

2.2.3 Sequential Neural Networks . . . 8

2.2.4 Optimization of Neural Networks . . . 9

2.3 Reinforcement Learning . . . 10

2.4 Policy Gradient . . . 12

3 Problem Formulation and Task Setup **16**

4 Description of the AiRLoc Agent **20**

4.1 Rewards, Action Space and State Space . . . 21

4.2 AiRLoc Model Structure . . . 22

4.2.1 Patch Embedder . . . 22

4.2.2 Positional Encoding . . . 26

4.2.3 Temporal Unit . . . 28

4.2.4 Decision Unit . . . 28

4.3 Early Design Choices . . . 29

5 Baselines **30**

5.1 Human Baseline . . . 32

6 Datasets **34**

7 Experimental Results **35**

7.1 Evaluation Metrics . . . 37

7.2 General Performance . . . 37

7.3 Agent Behavior . . . 41

8 Discussion of Results **45**

8.1 Comparisons with Baselines . . . 45

8.2 AiRLoc Dataset Generalization . . . 46

8.3 Benefits from Mid-level Vision Capabilities . . . 46

8.4 Ablations . . . 47

8.5 Longer Trajectories . . . 48

8.6 Agent Behavior 48
8.7 Human Baseline Comparison 49

9 Future Work **50**
9.1 Improvements 50
9.2 Expansions 50

10 Conclusions **51**

A Initial Results and Interpolation Errors **56**

B Additional Results **59**

C Trajectory Visualizations **63**

1 Introduction

Recent technological developments of unmanned aerial vehicles (UAVs) and satellites has seen an enormous increase in the amount of aerial view landscape and urban data that is available to the public [41, 6, 18, 33, 15]. Many businesses use UAVs as a core element in their operation, and it is generally an efficient way of gathering accurate aerial data. The main benefit of using UAVs instead of satellites is that UAVs are able to capture higher-quality images of a more local area, while satellites are better at capturing a more general understanding of large domains. While the lower-quality satellite data may be enough for some applications, it comes with the disadvantage that the images may not be captured with a sufficiently high frequency, which means that the image data might not be up-to-date on the occasion that one might need it.

UAVs can also be used in search-and-rescue (SAR) operations, where the task is to localize and assist one or several people who are missing, for example after a natural disaster. In many cases the rough location may be known and a UAV can be deployed to explore a given, confined area to precisely localize the missing people. The navigation of UAVs is often based on an pre-specified search approach, where the UAV moves in a fixed pattern which guarantees that the entirety of a given area is inspected in detail given sufficient time. In addition to always finding the target location, this approach has the benefit of being autonomous. The obvious downside is that it is a very time-consuming process to exhaustively scan every location in the area. Another common option is to let a human operator pilot the UAV, as it can yield a more intelligent behavior and thus quicker localization. The downside, however, is that this approach requires the operator to be trained in the navigation system, and it may be costly or inconvenient to hire the operator. The ideal solution would essentially be a merge between the two types of approaches, i.e., a system that can be used for autonomous localization in a more efficient way than a pre-specified search approach.

Motivated by the above, the main objective of this master's thesis is to investigate whether it is possible to train a machine learning (ML)-based system that is able to perform UAV-based localization more efficiently than heuristic alternatives. We will study this within a setup that is reminiscent of a SAR operation within a confined area.[1] More specifically, we abstract the problem into a framework that emulates a SAR-like setup without requiring access to actual UAVs – see Figure 5. In this framework, an agent operates on top of a given satellite image and is tasked with localizing a specific goal, specified as a rectangular region within the satellite image, from a given location in the image.

In many time-constrained settings, e.g. as in the SAR operations described above, global information (from satellite imagery or previous drone flights) might not be immediately available. To mimic this situation, in this thesis we limit the perception of the models to only partial glimpses of the underlying image, with no global information available at all.

[1]Our proposed task formulation may also be relevant for many types of environmental monitoring applications, such as in forestry management, or for UAV-based infrastructure surveillance and maintenance.

In particular, the models are not assumed to have access to GPS coordinates of the goal location (note that there are real-world scenarios where reliable access to GPS coordinates cannot be guaranteed, e.g. because global satellite navigation systems are susceptible to radio frequency interruptions and fake signals). We denote this task, which to the best of our knowledge has not been studied in the existing literature, *aerial view image-goal localization*. As the task is naturally abstracted as a sequential decision-making problem, the main focus will be on exploring and implementing reinforcement learning (RL)-based models, but we will also compare with other ML-models. In addition to this, we will conduct an early proof-of-concept human performance evaluation to get a rough assessment of how humans fare on this novel task.

While the specific formulation we consider in this thesis makes sense as an initial investigation – not the least because it allows for reproducible and controllable experimentation – it is not to be seen as a full solution. However, we believe our contributions can be part of a procedure that may eventually be used in, for example, environmental monitoring and SAR operation systems. To make our methodology more useful in practice, an obvious direction would be to extend it to accept as input a general scene description (e.g. one or several ground-level images), instead of it having to be an aerial view image. This could be done, for example, by having a separate module that is trained to translate more generic scene descriptions into aerial view images (these types of predictions are considered within the framework of geo-localization, see e.g. [30, 11]), followed by navigating to the target using our proposed aerial view image-goal localization system.

To conclude this subsection, we here list the main contributions of this master's thesis:

- To the best of our knowledge, we are the first to investigate the aerial view image-goal localization task.
- We develop and evaluate several ML-based methods, and we compare these with a heuristic search approach. In particular, a wide range of various RL-based methods are implemented and evaluated, most of which are trained without annotated data of any kind.
- We find that ML-based methods outperform heuristic approaches, and that RL-based methods outperform non-RL-based ML methods.
- We also perform a proof-of-concept human performance evaluation, which indicates that our RL-based methods outperform humans on average.

1.1 Limitations

An ideal evaluation of our proposed methods would be to deploy the systems on real UAVs and see how well they perform in real world settings. However, this comes with three main issues: i) lack of reproducibility, ii) performance limitations, and iii) implementation overhead. In short, such experiments would not be reproducible and thus verifiable, the size of our models would likely exceed the computational resources on the edge[2], and implementing this requires time that falls outside the scope of a thesis.

To simulate the UAV data, satellite imagery is used, the UAV locations are defined by crops of the full satellite image, which simulate a top-view of what the UAV currently sees. In a realistic application, wind, weather and obstacles could potentially affect the navigation possibilities of the UAV; in this project these effects have been neglected, and the UAV is always able to move in any direction and with a deterministic outcome associated with each movement. The search area has also been restricted and any move outside this area by the drone yields an entirely black visual input. This restriction, while perhaps appearing artificial at first glance, can be realistic in such SAR missions where the rescue targets are known to be within a certain area – in such cases, any movement outside could simply be considered erroneous. Furthermore, the issue of independently determining if the correct location has been found is not considered here, and if the simulated UAV finds the goal the algorithm stops automatically without agent intervention. Finally, the image representation of the target is not altered in any way, whereas in a real use case the target is not likely to look exactly the same at the time of searching as it does in the latest observation of that location (note that this would further motivate a principled methodology for automatically determining whether the target has been found, which again is not considered here). This is a very interesting future research area which is out of scope for this project.

1.2 Ethical Considerations

Naturally, as in most projects there are possible applications that may be unethical. In this case these mostly revolve around UAV navigation for military and surveillance purposes, and it is not impossible for somebody to utilize our research for such a purpose. However, thus far the research is missing many of the required components. We therefore decided to proceed with this research with caution in how we chose the intended application and in the direction that we took the investigations.

1.3 Related Work

Several prior works have investigated and proposed methods for autonomous control a UAVs [32, 16, 8, 3, 28, 45, 23]. Many of these works (e.g. [32, 28, 45]) revolve around methodologies for efficient scanning of large areas (e.g. agricultural landscapes) such that certain types of global-level downstream inferences – for example, determining the general health status of a field of crops – can be accurately performed based on a limited number of high-resolution observations. Hence, the goal in these works is to obtain a high downstream accuracy without having to exhaustively observe each part of a scene in detail, which reduces the UAV flight time and energy consumption. Aside from differing in task formulation (ours requiring precise localization of a particular observation, while the aforementioned works

[2]A first step to alleviate this could be to have the UAV communicate with a remote computing device; future work could include improving the efficiency of various algorithms to allow for on-device computations.

often revolve around global-level inference), these prior works assume access to a global lower-resolution observation of the whole area of interest, while we do not. Moreover, in contrast to our end-to-end trainable methodology, these existing approaches are often hand-crafted, rely on ad-hoc decision criteria, and/or optimize local rather than global objectives.

There are also works that are closer to us in terms of task setup, such as [3, 16, 8]. For example, in [3] a hierarchical planning approach for a goal reaching task is proposed, where a rough plan is first proposed using A*. The rough plan is subsequently used as an initial guess by a finer-grained planner which parametrizes the initial trajectory as continuous B-splines and performs trajectory optimization. The system assumes access to ground truth detections of moving objects and ground classifications. Overall, while promising, methodologies such as those of [3, 16, 8] are often more convoluted than our RL-based approach, which also has the advantage of not relying on annotations in training nor testing.

Our work is also related but orthogonal to the increasingly studied problem of geo-localization [40, 35, 47, 44, 24, 38, 30, 5, 4, 48, 11]. Such works revolve around predicting relationships between two or more images of different modalities, e.g. predicting the satellite or drone view corresponding to a ground-level image. Most such methods perform the task by an exhaustive comparison within a large image set, and are thus very different to our setup which is rather about minimizing the amount of images (local patches) inspected when performing localization. However, our proposed methodology could further benefit from, or be generalized by, incorporating geo-localization methods. For example, if the goal location is specified from a ground-level perspective (which may be more realistic in practice), geo-localization methods can be used to match the top-view images observed by our proposed method during goal localization.

From a pure task formulation perspective, and setting aside the application area, our setup may be most closely related to embodied image-goal navigation [1, 49, 17]. In this framework, an agent (model) is tasked to navigate in a first-person perspective within a 3d environment towards a goal location which is specified as an image within the environment. In [17], the agent is tasked with the navigation to an image of the goal location, which is related to what we are trying to do. The difference being the contents of the images, where ours portray 2d aerial images while they use panoramic images of indoor scenes. In [46], an agent is tasked with semantically mapping the entire environment in which it is moving. Using this information, the agent should estimate the remaining effective distance to the goal. This task as well is conducted using indoor scenes as data. To the best of our knowledge, prior to us, no image-goal localization tasks have been conducted using aerial data as input, and RL as the navigation system.

In addition to us, relatively few prior works have considered inference based solely on partial glimpses of an underlying image [25, 26]. In contrast, most earlier RL-based methods that have been proposed for computer vision tasks – e.g. for object detection [7, 12, 22] and aerial view processing [34, 2] – assume access to at least a low-resolution version of the entire scene or image being processed. Even the seminal work [19] uses lower-resolution full image input in addition to high-resolution glimpses during its sequential processing, even though in principle it may be possible to re-design it to operate based on high-resolution glimpses alone.

Finally, [9] is related to our work in terms of task formulation, even if the application area differs. They propose a self-supervised task for training an embedding function that can be used in downstream tasks, such as image classification. This self-supervised task is similar to our setup, as it revolves around learning a spatial mapping function that relates a given image patch (corresponds to the start location in our framework) to a particular one of its eight neighboring patches (corresponds to the goal location in our framework). Hence, our task can be seen as a strict generalization of that in [9], as in our setup the goal may be further away from the start location. Also, different from us, [9] employs patch jittering to prevent the model from learning to recognize patterns in the image (since during downstream inference they do not want to rely on pairwise inputs to the model). As we have an entirely different end objective, we have decided not to perform jittering so that the model may learn to pick up on relevant spatial relationships when performing goal localization.

2 Background

In this section we provide background of machine learning, with a focus on neural networks and reinforcement learning, which is needed for understanding the methods we develop and evaluate in this master's thesis.

2.1 Machine Learning

The proposed solutions to the task presented in Section 1 are all based on machine learning (ML). ML is the concept of using data to optimize algorithms such that they can perform a specific task. On a high level, ML splits into supervised and unsupervised, where in supervised learning there is a correct answer for the machine to find. While in the unsupervised setting, the task is usually to find patterns in the data without the use of annotations. For example, common supervised tasks include the classification and localization of objects in images. Common unsupervised tasks include clustering of data into categories and anomaly detection.

In the supervised setting, the task is to create an algorithm that learns the relation between a predefined set of inputs and labels. That is, given the input, the goal is that the algorithm should predict the correct labels. To train such an algorithm, a *loss function* which measures how well the algorithm predicts the correct labels is introduced – the better the prediction, the smaller the loss. Calculating the gradient of the loss function and using this information with an *optimizer*, the parameters of the algorithm are changed in many steps until the loss is minimal. This is further discussed in Section 2.2.4. One severe drawback of this setup is that obtaining quality labels for most types of data is very expensive and time consuming. Since the performance of the model is highly dependent on the available training data and corresponding labels, this is a major issue.

To verify that the model is learning something *generalizable*, there typically is a validation and test set, in addition to the training set (data) which is used to optimize the model parameters. Testing the model on previously unseen data gives a more accurate representation of the performance of the model. The purpose of the validation set is to use it to monitor the performance and perhaps alter the hyperparameters of the algorithm. As the process of tuning the parameters uses the validation data, a test set allows testing the model on entirely unseen data, generating a more valid result.

Constructing labels for the proposed goal patch localization task is not trivial, as creating those labels will inevitably induce a bias to the behavior of the agent. It would also be a time-consuming endeavour to manually allocate a desired action for each input to the model. Another, more natural way of creating labels for our task would be to assign a label to the input corresponding to the action that moves the agent closer to the goal patch. This could later be trained using imitation learning where the task is to replicate a behavior, defined by an "oracle" which has the "correct" strategy. While this could create an optimal behavior in theory, it would more likely induce plenty of bias to the system and render the problem unsolvable. Even though, it is the shortest path, the agent might not be given enough information to find the optimal behavior. It is more reasonable to allow the agent to move more freely, allowing it to explore and learn from the environment and experience before learning to move the target.

2.2 Neural Networks

ML is a large topic which covers many models. Having narrowed it down to supervised learning, there are still many models remaining, such as logistic regression and support vector machines. The arguably most popular and flexible method currently in use is neural networks (NNs) which are constructed from a series of layers with linear matrix multiplications followed by non-linear elementwise functions. These will be discussed further in Section 2.2.1. A related concept is that of *deep* neural networks (DNNs). This term generally refers to any neural network that is sufficiently "deep", i.e, consists of sufficiently many layers. The two terms are almost interchangeable, and all applications of neural networks in this project are considered deep neural networks. The benefit of depth is the complexity of functions that the network may learn.

2.2.1 Fully Connected Layers

There are many types of layers that one can use to create NNs. These types of layers are suited for different types of data and are as such used for different tasks. The most basic block of a neural network is the fully connected (FC) layer which directly connects each input node to each output node with a matrix multiplication. Nodes are defined as the elements in a vector, illustrating that output $\boldsymbol{y}$ from a fully connected layer,

$$\boldsymbol{y} = \psi(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}), \tag{1}$$

is the result of a matrix multiplication of the weights $\boldsymbol{W}$ and the input $\boldsymbol{x}$ summed with the bias $\boldsymbol{b}$ and then all passed to the activation function ψ. If the input $\boldsymbol{x}$ contains n nodes and the output $\boldsymbol{y}$ contains m nodes, the weight matrix will be of the shape $(m \times n)$. The weight matrix thus describes the strength of the connections between individual input and output nodes. Together with the bias, $\boldsymbol{b}$, $\boldsymbol{W}$ are the trainable parameters of a neural network. The activation function allows several of these layers to be placed after each other. Consider the two layers, layer 1 and layer 2, if there is no activation function the output would be

$$\boldsymbol{y} = \boldsymbol{W}_1(\boldsymbol{W}_2\boldsymbol{x} + \boldsymbol{b}_2) + \boldsymbol{b}_1 = \boldsymbol{W}_1\boldsymbol{W}_2\boldsymbol{x} + \boldsymbol{W}_1\boldsymbol{b}_2 + \boldsymbol{b}_1$$

which is just one set of matrix multiplications that can only learn linear functions. Including a non-linear activation function prevents this and allows for stacking of the layers to learn more complex (non-linear) functions between the input and output.

2.2.2 Convolutional Layers

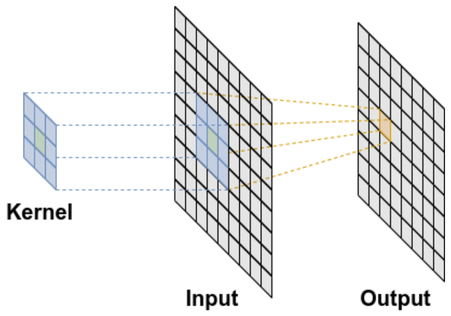


Figure 1: *A basic illustration of the workings of a convolutional layer. A kernel, i.e a square matrix, is moved and applied over the entirety of the input to produce an output.*

Due to the high dimensional nature of images, they are generally difficult to interpret with only FC layers. Even small images would require very large fully connected layers to be handled, which prolongs and complicates the training process. Additionally, treating each individual pixel in an image as a separate input means that the spatial information on where in the image they are located is lost. To remedy this issue, we use the well established architecture of Convolutional Neural Networks (CNN) to process the visual state input to the agent. CNNs use a kernel with fixed dimensions to map a local part of the image to the output. An illustration of a convolutional layer can be seen in Figure 1. In contrast to the FC layers where both input and output are one dimensional, the inputs and outputs of CNNs are high dimensional, commonly of rank 3 (height, width, color). This means that each node in the output layer only utilize a small region in the input image. This method is better at extracting local features in the image and at the same time reduces the number of parameters needed. Another benefit with this structure is that it allows for the possibility of using multiple kernels in the same layer, which are trained to extract different local features. Commonly, the kernels learn to enhance certain edges or colors. Stacking convolutional layers after each other allows the network to build up an understanding of increasingly complex features extracted by the previous layers, for example corners as a combination of horizontal and vertical lines.

2.2.3 Sequential Neural Networks

Fully connected layers map one set of inputs to a set of outputs with no regard for previous sets of inputs. For some problems that consist of sequences of inputs, such as sentences, this is not ideal. *Recurrent Neural Networks* (RNN) address this problem by retaining a hidden state of combined information about the previous inputs. This hidden state in conjunction with the current input produce the output, the hidden state is then updated and reserved for the next input. This allows the network to have a memory and usually improves performance for sequential problems. A variant of the classic RNN that is widely used in many sequential machine learning tasks is the *Long Short Term Memory* (LSTM) network. It sacrifices some simplicity of the basic recurrent structure for significantly increased performance and robustness. The LSTM network uses two types of hidden states, the cell state, c_t and the hidden state, h_t and their updates are governed by a set of different gates. These update rules limit the networks susceptibility to exploding and vanishing gradients by clipping how much of the previous states' information passes to the next state [13].

A different architecture which is even better at learning long term dependencies of sequences are the transformers. They were originally introduced in natural language processing (NLP) as an alternative to LSTM for learning long range dependencies between words. What makes the transformer good at this task is that it does not process the sequence as a sequence with an order, rather it processes it as a set and uses positional encoding to attend to the order. This allows the transformer to relate each element in the sequence to all other elements in the sequence, with no bias towards elements that are closer together. Although the transformer was originally developed for NLP it has been used successfully with images, and with a fitting positional encoding of the different states in the trajectory, it might be successful in our problem setting. See [36] for a detailed explanation of the transformer architecture.

2.2.4 Optimization of Neural Networks

In essence, NNs are complex functions with many coefficients, and to solve the problem we need to tune these parameters, this is done during the training phase. Training involves formulating an objective loss function, for which a low value indicates an effective network. Then the network weights are optimized to minimize the value of the objective function and to increase the desired performance of the network.

There are several optimization schemes, but most rely on the principle of taking a small step in the descent direction determined by the estimated gradient of the objective function with respect to the model weights, these are known as gradient descent methods. Due to the sheer scale of the datasets typically used in ML, this direct approach is generally unfeasible. Typically, the training dataset is instead divided into smaller *batches* and the objective function is calculated and optimized sequentially for these smaller units of data. This method is referred to as the *Stochastic Gradient Descent*(SGD) algorithm and it is widely used in machine learning. It allows for the use of massive dataset even with relatively limited memory capacities but, it also introduces noise in the gradient estimation. This noise may result in the algorithm being unable to estimate the gradient well enough to reach the global minima. However, it could also be beneficial since the introduction of a stochastic step may allow algorithm to escape local, shallow minima.

The volatile optimization landscape of loss functions for deep neural networks contains many local minima and many low slope surfaces which leads to problems in the training of the networks. To overcome such problems and reduce the risk of being stuck in a local optimum many variations of the SGD algorithm have been proposed. One such popular algorithm, Adam, is used in this project. The main benefit of Adam is that it is usually quick to converge and decent at avoiding local minima. It keeps running averages of the gradient and the squared gradient, and uses these averages over time to calculate the update step,

$$
\begin{aligned}
m_w^{(t)} &= \beta_1 m_w^{(t-1)} + (1-\beta_1)\nabla L \\
v_w^{(t)} &= \beta_2 v_w^{(t-1)} + (1-\beta_2)(\nabla L)^2 \\
\hat{m}_w &= \frac{m_w^{(t)}}{1-\beta_1^t} \\
\hat{v}_w &= \frac{v_w^{(t)}}{1-\beta_2^t} \\
w^{(t)} &= w^{(t-1)} - \eta \frac{\hat{m}_w}{\hat{v}_w + \epsilon}
\end{aligned}
$$

where m_w and v_w are the running means of the gradient and the gradient squared and β_1 and β_2 are hyperparameters describing how fast the running means decay towards the current value. In the last step the weights, w^t are updated using the previously computed bias corrected running means, where η is the learning rate controlling the overall step size of the algorithm. ϵ is a small fixed value used to reduce numerical risks of division. The advantage with this implementation is that even in situations where the gradient is very small, i.e, in local minima, the update need not be small. This allows the algorithm to avoid becoming trapped in local minima. In addition, it typically accelerates the rate of convergence[14].

2.3 Reinforcement Learning

Reinforcement learning (RL) is a machine learning framework for teaching an *agent* a sequential behavior within a given *environment*. RL is typically described as a Markov Decision Process (MDP), this is a normal Markov process with the ability to insert actions. A Markov process is used to describe a system in which each state is only dependent on the directly previous and none of the others. Additionally, a transition distribution governs the probability of moving from one state to the next. Our RL setup make use of this MDP setting.

Reinforcement learning utilizes three important concepts to define the agent's interactions with its environment:

- The **state** s_t is the input from the environment to the agent, it encompasses everything that the agent can percept about its surroundings. In this project, the state consists of the current patch, the goal patch and the location of the current patch.
- The **action** a_t is the desired action of the agent determined by the inputs from the environment, i.e, the state, s_t.
- The **reward** r_t is the response from the environment to the action taken. It is a real number and is a measure of how well the agent is performing. The reward supplied

by the environment is controlled by the designer of the reinforcement learning system and is a crucial part of the successful development of such a system.

Note that these three objects all depend on the time, t. Given an initial state, s_0, the agent will advance to the next by supplying the first action, a_0, and in return receiving the first reward, r_1. The environment will then provide the next state, s_1. This cycle continues until a stop criterion has been met.

There are two main components deciding the evolution of the trajectory; firstly, the agent's policy, π, which determines the action of the agent given the input. It can either be stochastic in that the policy generates a distribution from which the action is sampled or it can be deterministic. Secondly, the transition distribution, p, determines how the environment reacts to the action taken by the agent. The process of generating a trajectory in the MDP setting can bee seen as,

$$s_0 \xrightarrow{\pi(*|s_0)} a_0 \xrightarrow{p(*,*|s_0,a_0)} r_1, s_1 \xrightarrow{\pi(*|s_1)} a_1 \xrightarrow{p(*,*|s_1,a_1)} \dots \xrightarrow{p(*,*|s_{T-1},a_{T-1})} r_T, s_T \tag{2}$$

where each arrow indicates a sample from the corresponding distribution. Hence, one step in the trajectory consists of taking two different samples. Note that the initial state s_0 is drawn from a third initial distribution, p_0 which is an environment dependent distribution. Possible interpretations of the transition probability in the setting of this project could be the influence of wind or energy requirements of a specific action. However, for all experiments in this thesis the effect of the transition distribution has been omitted and the UAV will always move in accordance with its intended action. Hence, $p = 1$ for all states and actions. This turns Equation (2) to

$$s_0 \xrightarrow{\pi(*|s_0)} a_0, r_1, s_1 \xrightarrow{\pi(*|s_1)} a_1, r_2, s_2 \xrightarrow{\pi(*|s_2)} \dots \xrightarrow{\pi(*|s_{T-1})} a_{T-1}, r_T, s_T \tag{3}$$

where there is one more state than action and reward. The last step in any trajectory does not have a corresponding action. Now the trajectory extracted from Equation 3 is defined as

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, s_2, \dots, a_{T-1}, r_T, s_T,) \tag{4}$$

and contains the information that describe the trajectory. The term "episode" is used interchangeably to describe a trajectory. The details of the environment for this thesis is outlined in Section 3, but in short, the MDP is described by: a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a set of rewards $\mathcal{R}$, an initial state distribution p_0, and a discount factor γ. The discount factor determines how much of the future reward should be propagated back to the previous states.

A key problem in reinforcement learning is the inability to immediately determine whether an action by the agent was beneficial or not. In many problem setups the reward signal from the environment is very sparse and often concentrated to key events during the trajectory. In navigation tasks the only reward that might be provided by the environment could be whether the agent successfully located the target or not. This reward would only present itself at the final time step of the trajectory and all rewards until then would be zero. This leads to the problem of how to determine the effectiveness of all actions but the last. The discount factor is one attempt to solve this problem. In this case the reward from a specific action is a discounted sum of all future rewards received in the trajectory. This allows the algorithm to understand that early actions in the trajectory influences and benefits the future states and rewards of the trajectory.

In addition to the basic environment structure outlined in the previous paragraph, there are further concepts which relate the optimization of the agent to the optimal behavior. First, the most central part in any RL system is the agent policy $\pi_\theta(s_t)$ which makes the decision of which action to take in respect to the current state, using the parameters θ. In our case, the policy is a NN and thus θ are the weights of the connections. The optimization objective in RL is to maximize the *value function* $v_\pi(s)$ for all states. The function represents the expected return for the current state, meaning the current reward and all the discounted future rewards. Hence, the value function is defined as

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=1}^{T-t} \gamma^{k-1} r_{t+k} \middle| S_t = s \right] \tag{5}$$

where $\gamma \in (0, 1]$ is the discount factor, which makes sure that the value function is finite even if the trajectory length approaches infinity. The optimal value function is denoted $v_*(s)$ and is the expected return under an optimal policy. There are two main ways of tuning a policy to the environment; on policy and off policy. In off-policy, the environment is explored by a behavior not dictated by the current policy. In contrast, on-policy always moves according to the current policy. This project will use a method from the on-policy family, which generally are more sample inefficient and slower to train but can learn more complex systems, due to being able to explore longer trajectories. More specifically, we will use the REINFORCE algorithm, which is one of the fundamental methods in the set of policy gradient methods.

2.4 Policy Gradient

In reinforcement learning, one of the central concepts is the trade-off between exploration and exploitation, the agent needs to both learn from its past trajectories, and it needs to explore new ones. In REINFORCE, this is achieved by implementing a stochastic policy, meaning that instead of $a_t = \pi_\theta(s_t)$ where the action is returned from a function, the action is drawn from a distribution $a_t \sim \pi_\theta(a_t|s_t)$. This gives the agent the ability to simultaneously explore and exploit, as the action preferred by the agent is the most likely, but there is a probability that other actions are chosen as well. Note that this is different from a stochastic environment where a chosen action does not generate a deterministic state, in our problem set up the environment is deterministic. This is a simplification of the end goal, as the UAV would rarely be moving deterministically, due to wind and other control problems.

The following derivation of the REINFORCE method is based on [21], note that this is in respect to the objective function, which is maximized. The negation of the objective function is referred to the loss function throughout the report. Maximizing the objective function is equal to minimizing the loss function. The value function shown in Equation (5) is similar to the objective function, however the loss needs to be expressed as dependent upon the parameters, $\boldsymbol{\theta}$. Hence, the objective function is

$$J(\boldsymbol{\theta}) = \mathbb{E}_{p_\theta}\left\{\sum_{k=1}^{T}\gamma^{k-1}r_k\right\} \tag{6}$$

where the parameter dependent on θ is the underlying distribution p_θ generating the trajectories. The parameters update according to the gradient update rule

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + \alpha\nabla_{\boldsymbol{\theta}}J_{|\boldsymbol{\theta}=\boldsymbol{\theta}_n}$$

where n is the update index, and α is the learning rate factor. The index n is relevant as it allows the parameters to be updated with an arbitrary frequency, not in relation to a number of episodes. The purpose of α is to control the magnitude with which the parameters are updated and is standard practice in optimization problems. The above equations are the basic outline as to how the parameters of a policy gradient RL-agent should be updated. Within this system there are several ways of approximating the gradient to perform the update. As previously mentioned the approach that we will use is the REINFORCE algorithm[39].

REINFORCE solves the problem of approximating the gradient by using that

$$\nabla_{\boldsymbol{\theta}}\log p_{\boldsymbol{\theta}}(\tau) = \nabla_{\boldsymbol{\theta}}p_{\boldsymbol{\theta}}(\tau)\frac{1}{p_{\boldsymbol{\theta}}(\tau)} \iff \nabla_{\boldsymbol{\theta}}p_{\boldsymbol{\theta}}(\tau) = p_{\boldsymbol{\theta}}(\tau)\nabla_{\boldsymbol{\theta}}\log p_{\boldsymbol{\theta}}(\tau), \tag{7}$$

which is a property used in the derivation. Now, for the derivation of the gradient, the previous step based variables transforms into trajectory based variables. One collection of states from start to end of a game is referred to as a trajectory τ and collect all the state, action, reward tuples. Now this collection of states will occur with a certain probability dependent on the current policy, hence $\tau \sim p_{\boldsymbol{\theta}_n}(\tau)$ and the cumulative reward to go is written as, $r(\tau) = \sum_{k=0}^{T}\gamma^k r_k$ which simplifies the notation. The objective function in these new terms now becomes simply $J(\boldsymbol{\theta}) = \mathbb{E}_{p_\theta}\{r(\tau)\}$. From basic statistics we know that the expectation is simply

$$J(\boldsymbol{\theta}) = \int_\tau p_{\boldsymbol{\theta}}(\tau)r(\tau)d\tau \tag{8}$$

which is exactly the same as in Equation (6). To update the weights in the policy π and optimize the solution we need to find the gradient of Equation (8) by

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \int p_{\boldsymbol{\theta}}(\tau) r(\tau) d\tau \\ &= \int \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\tau) r(\tau) d\tau \\ &= \int p_{\boldsymbol{\theta}}(\tau) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) r(\tau) d\tau \\ &= \mathbb{E}\{\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) r(\tau)\}\end{aligned} \tag{9}$$

where we have utilized the log-derivative trick from Equation (7) to get to an expression which is simple to calculate. The expectation can be replaced by the sample mean of several trajectories using the law of large numbers, further, including more trajectories in this expression will lower the variance of the gradient estimate. Calculating the probability of the trajectory as

$$\begin{aligned} p_{\boldsymbol{\theta}}(\tau) &= p(s_0)\Pi_{k=0}^{T} p(s_{k+1}|s_k, a_k)\pi_{\boldsymbol{\theta}}(a_k|s_k) \\ \implies \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) &= \sum_{k=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_k|s_k)\end{aligned}$$

allows us to replace the probability term in Equation (9). This means the gradient is written as an expectation independent of the environment dynamics, this is not an issue in our case, but for the generalizability of the derivation it is included here. The final gradient expression is

$$\nabla_{\theta} J(\boldsymbol{\theta}) = \mathbb{E}_{p_\theta}\left\{\left(\sum_{k=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_k|s_k)\right)\left(\sum_{l=0}^{T} \gamma^l r_l\right)\right\} \tag{10}$$

and the expectation allows the gradient to be estimated using the sample mean of trajectories. Given M different trajectories, the gradient is estimated as

$$\nabla_{\theta} J(\boldsymbol{\theta}) \approx \frac{1}{M}\sum_{m=0}^{M}\left(\left(\sum_{k=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_k^m|s_k^m)\right)\left(\sum_{l=0}^{T} \gamma^l r_l^m\right)\right) \tag{11}$$

which is the core of the REINFORCE algorithm. In practical implementation, the gradient is computed in an auto-differentiating framework, such as PyTorch, eliminating the need to explicitly differentiate $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_k^m|s_k^m)$. In these frameworks calculating the mean estimate of J is enough as the backend takes care of the gradient calculations. Also due to how gradient optimization is implemented in these frameworks, the loss function is the actual function being minimized instead of the objective function. The loss function,

$$\mathcal{L}(\boldsymbol{\theta}) = -J(\boldsymbol{\theta})$$

is simply the negative objective function.

A common issue in the REINFORCE algorithm is that it is notoriously sample inefficient, and may require massive amounts of training to reach an optimal policy. While reaching the optimal policy is guaranteed in theory given the correct gradient estimates, the high variance of these estimates introduce issues that require a very large batch size or very long training times. One common solution to this problem is to introduce a baseline that estimates the value function in a given state. The estimate can then be subtracted from the expected reward, giving a positive result if the trajectory was better than expected and a negative result otherwise. Denoting the baseline as $b(s_t)$ the new gradient estimate becomes,

$$\nabla_\theta J(\boldsymbol{\theta}) \approx \frac{1}{M} \sum_{m=0}^{M} \left(\left(\sum_{k=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_k^m | s_k^m) \right) \left(\sum_{l=0}^{T} \gamma^l r_l^m - b(s_l^m) \right) \right) \tag{12}$$

which is essentially Equation (11) with a subtracted value from the value function. Estimates of the gradient using this equation will have a lower variance than estimates using Equation (11) due to the fact that variance scales quadratically with the difference from the expectation. An accurate baseline will closely follow this expectation which will make the absolute values of the second factor smaller. This will in turn lower the variance of estimates which leads to a faster convergence.

The use of baselines however, requires some extra implementation and increased computational power. Another option is to normalize the rewards which works as a simplified estimate of the value function, and thus a baseline. This normalization can be done in several ways. Either globally, where the mean is subtracted from all rewards or more precise normalization groups can be used. For example rewards from states that are equally far away from the goal patch or that are of equal difficulty can be normalized separately.

3 Problem Formulation and Task Setup

Imagine a UAV is given the task of examining a predefined area in search of a location defined by a given aerial image of said location[3]. Such a task can be relevant in several contexts, e.g. in search-and-rescue (SAR) operations, critical infrastructure inspection, or for various types of environmental monitoring. In this thesis, we consider a simulated proxy for this setup by using a top view image of a landscape as an analogy of the predefined area and only letting the agent (UAV) observe partial crops, or patches, of this underlying image. The initial state of this image is defined by a start crop and a goal crop, and the agent then chooses which action to take which results in a newly observed crop. At this stage the agent is tasked with predicting which action to take based on the first two visited crops and the goal crop. This process is repeated until either the goal crop has been reached or until a maximum number of steps T have been taken. If the agent reaches the goal, it is defined as a success, otherwise it is defined as a failure. An overview can be seen in Figure 5.

The choice of the size and scale parameters are selected to mimic the task described above as closely as possible. The full images are scaled such that the image size of $H_{\text{im}} \times W_{\text{im}}$ in pixels represents a desired real world scale of $H_{\text{world}} \times W_{\text{world}}$ meters. The patches are of dimensions $H_{\text{patch}} \times W_{\text{patch}}$ pixels; the size is selected such that a full image fits 5 patches with 4 pixels padding in between (cf. Figure 2). This is a relatively small number of patches that makes the game solvable, yet requires the agent to behave intelligently – despite a 5×5-sized setup potentially sounding simple, it turns out to be a challenge even for humans to solve tasks of this size (see Section 7). The various mentioned parameters should be chosen such that the agent is able to see both local and global structure, and we found that choosing the parameters such that each patch covers about 100×100 meters is appropriate. This is of the scale of a few city blocks, but not entire neighborhoods, hence there should be some learnable structures here.

There are two sets of action spaces the agent could operate in, either within a *continuous* or a *discrete* setup. In the discrete action space, the agent is only allowed to choose from locations in a grid, either only the eight neighboring patches or from any of the locations in the grid. When the agent can only choose from the adjacent patches, it is forced to move towards the goal in several steps. In contrast, when the whole grid is available as an action space, it is possible to reach the target with a single action. Finally, a continuous setup has been tested where the agent outputs a move in pixel values (dx, dy) and is therefore not constrained to move within a pre-defined grid.

The purpose of distinguishing between and considering these two setups (discrete and con-

[3] If GPS coordinates of the goal location are provided the task becomes much simpler, if not trivial. However, there are many circumstances where reliable access to GPS coordinates can not be guaranteed, as global satellite navigation systems are susceptible to radio frequency interruptions (malicious or accidental) and fake signals. It can also be that the goal image was captured without an active GPS system (for example, one may have access to images captured by a drone pilot who at the time of flying did not know that the captured images were going to be critical, e.g. for a SAR operation in the same area).

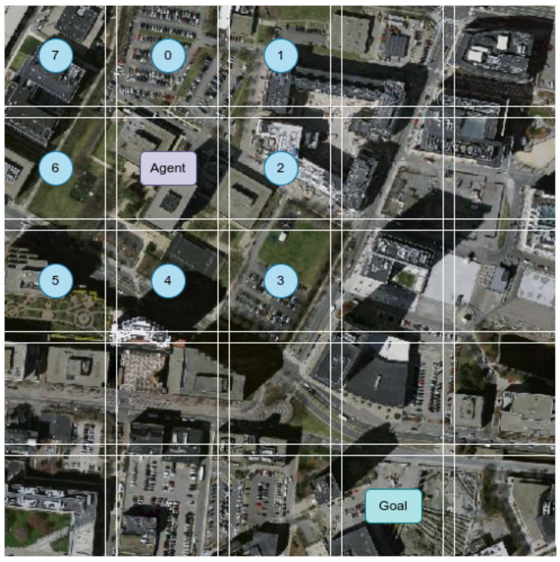
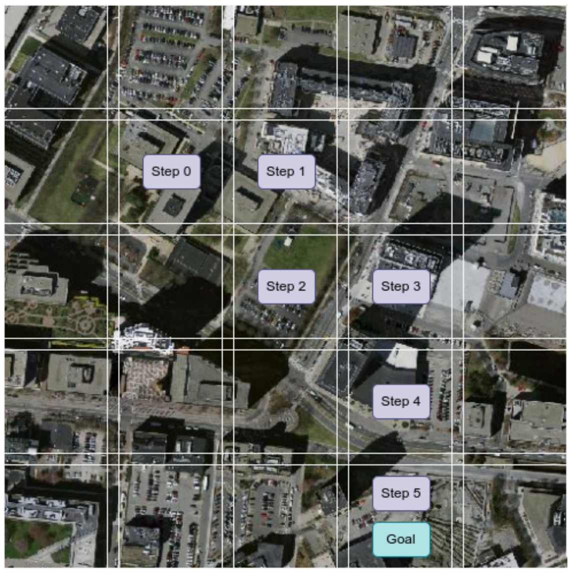


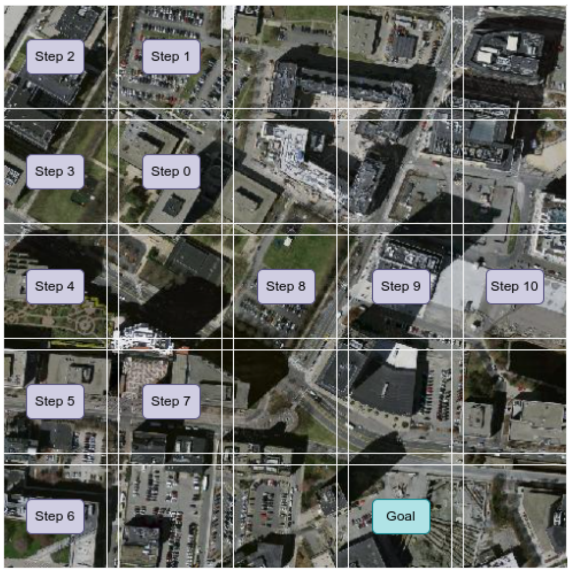
Figure 2: *The eight possible actions for the discrete 8-dimensional movement setup (number 0-7 around the agent's current location). A possible goal location is also indicated.*

tinuous) is that the continuous agent, which is more directly connected to the proposed real world use case, may be much more difficult to optimize. This is where the discrete agent helps by reducing the search space for the optimization algorithm, as it performs predictions in a much sparser action space. Moreover, the step size in the discrete setup implicitly helps the agent to cover more area in the image than the continuous setup, by eliminating the risk of partial overlap. An example of the different action space is shown in Figure 2 where the figure shows the 8-dimensional action space (agent-centric). The continuous agent is allowed to take any step (dx,dy). Sample trajectories of the discrete agent are shown in Figure 3a and Figure 3b; these show how the agent could move in the 8-dimensional discrete setup for a success case and a failure case, respectively. The 80-dimensional setup is similar, except that the agent is able to take arbitrarily long steps in each action.

An important design choice to make is if the agent should be allowed to operate outside the full image or not. This is a trade-off, where on one hand the agent should not be able to go outside the image as the search area is predefined to the area presented in the full image. On the other hand, restricting the agent to staying inside the image requires restricting the action space of the agent, and it would not be strictly self controlled. By this reasoning, we decided to investigate the problem mainly with the agent having the ability to take any action in any state. In the case that the agent moves outside the image it will receive an all black image, showing the agent that it has exceeded its bounds. Some investigated models will restrict the agent behavior to avoid the area outside the image (cf. Section 7.2) and in these cases this will be clearly stated. The ideal setup would have a larger underlying image with only a smaller part of the central image defined as the search area, that way the agent



(a) *An example game where the 8 dimensional agent is successful.*



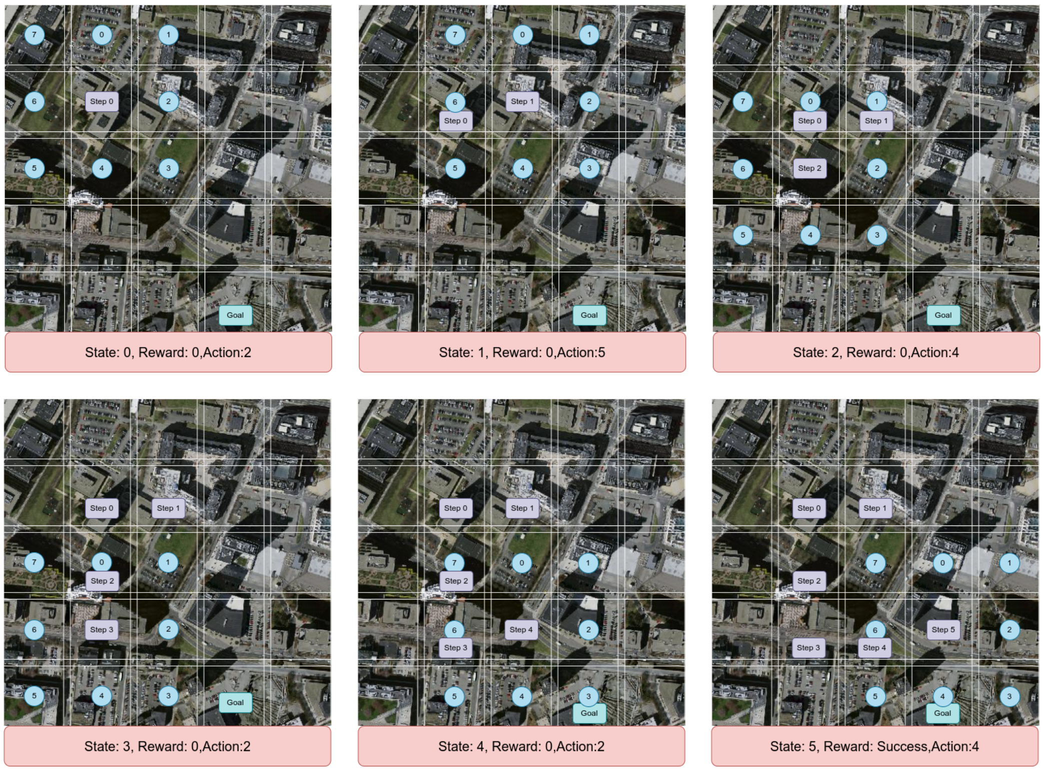
(b) *An example game where the 8 dimensional agent is unsuccessful.*

Figure 3: *Examples of a successful and an unsuccessful trajectory for an agent with an 8-dimensional action space.*

could move outside the search area while still receiving landscape images. However, this setup results in computational and programming issues that make it much more difficult to implement, and hence it was omitted from the scope of the thesis.

In ML, a larger dataset helps prevent overfitting, forcing the model to learn rather than just memorize. One of the ways we introduce this is during the training phase, where the start and goal patches are sampled from a discrete uniform random distribution in a 5×5 grid setting. First, the start patch is sampled freely in the image grid, after which the goal patch is sampled, also in the grid, such that there is no overlap between the start and goal patches. This ensures that we get random games each time, which effectively increases the dataset size, as one image may be used for several different setups ($25 \cdot 24 = 600$ setups in fact). To further enlarge the training dataset, we include data augmentations of horizontal and vertical flipping, which increases the effective dataset size even further.

The sampling could also be done in a continuous fashion; however, sampling the patches in this discrete grid setting has two main benefits. First, when subjecting the agent in the discrete action space, we ensure that it always perfectly overlaps the goal patch when reaching the goal. This provides a consistent reward signal to the agent during training, where there is always one and only one solution to each setup. Otherwise, it will raise the issue of choosing the intersection-over-union (IoU) threshold in such a way that there would either be multiple solutions to each game or that there would be no solutions to some



State: 0, Reward: 0,Action:2
State: 1, Reward: 0,Action:5
State: 2, Reward: 0,Action:4
State: 3, Reward: 0,Action:2
State: 4, Reward: 0,Action:2
State: 5, Reward: Success,Action:4
Step 0
Step 1
Step 2
Step 3
Step 4
Step 5
Goal

Figure 4: *A more detailed example game, where the state, reward and action is presented for each time step. This presents a successful trajectory .*

games. Second, sampling the games in this discrete manner allows us to categorize them in groups of difficulty, which is useful when examining the behavior of the various models and baselines. Games with 1 discrete step away are of difficulty 1, games with 2 discrete steps away are of difficulty 2, and so on. This provides in total four well-defined difficulties (numbered 1-4).

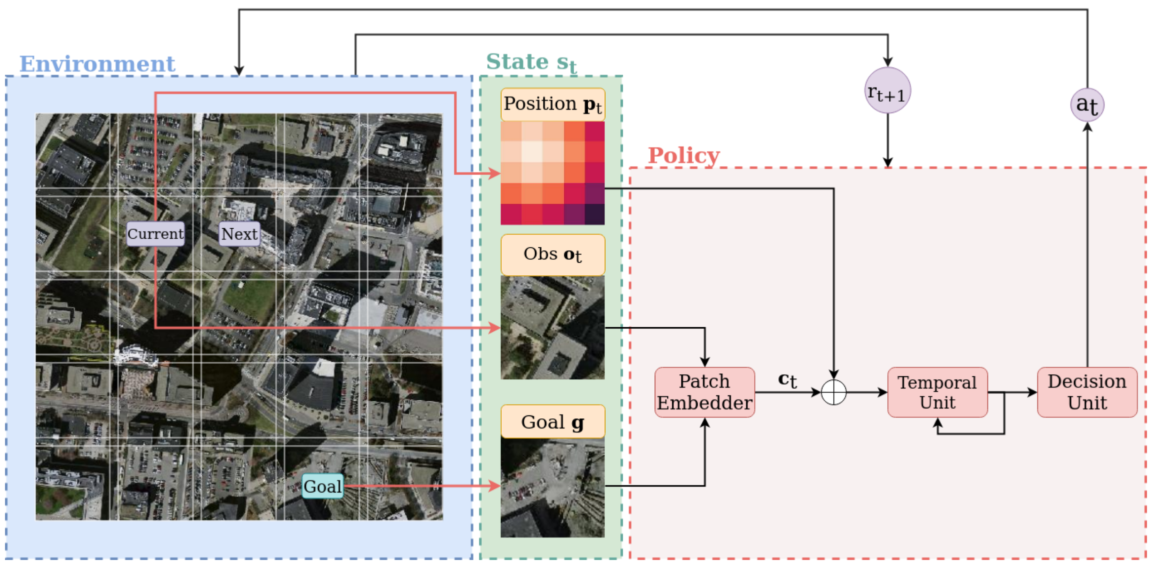


Figure 5: *Overview schematic of* AiRLoc*, our RL-based agent for aerial view image-goal localization. The input state s_t consists of the agent's current position $\boldsymbol{p}_t$, its currently observed patch $\boldsymbol{o}_t$, the goal patch $\boldsymbol{g}$ and the hidden state $\boldsymbol{h}_t$ of the LSTM-based policy. First, $\boldsymbol{o}_t$ and $\boldsymbol{g}$ are fed through an embedder to generate a common representation $\boldsymbol{c}_t$. Then the positional encoding $\boldsymbol{p}_t$ is added onto that embedding. The result $\boldsymbol{c}_t + \boldsymbol{p}_t$ is subsequently processed by an LSTM in the* Temporal Unit*. This output is then feed to the* Decision Unit *which yields an action probability distribution, $\pi(\cdot|s_t)$. This is either a 2d Gaussian (continuous setup) or a softmax vector (discrete setup). The current movement action a_t is then sampled from $\pi(\cdot|s_t)$, which generates the next state s_{t+1} and reward r_{t+1} (reward only provided during training). The process is repeated, either until the agent reaches the goal patch, or after a maximum number of steps T have been taken. Note that AiRLoc is* never *given an observation of the full underlying environment (even at a low resolution), i.e., it has to operate solely based on its partial observations $\boldsymbol{o}_0, \ldots \boldsymbol{o}_t$ and the goal $\boldsymbol{g}$.*

4 Description of the AiRLoc Agent

In this section we describe *AiRLoc*, the reinforcement learning (RL)-based model that we propose for tackling the aerial view image-goal localization task. The other methods and baselines that we compare with are described in Section 5. We first outline the details of the environment, such as formally defining rewards, actions, and state. Thereafter, the architecture of the policy, which is implemented as a neural network (NN), is described. This includes an overview of the purpose of each component in the NN and which approaches that were attempted for each component.

4.1 Rewards, Action Space and State Space

In Figure 5 we show the inner workings of how the agent interacts with the environment. Note in particular that the environment is of a recurrent nature, where output information feeds back into the policy and generates the next state. We next describe the details of the state space, thereward action space and the reward design. The **state space** consists of the parts of the environment that are observed by the agent. In our setup, the state s_t at timestep t contains the currently observed RGB patch $\boldsymbol{o}_t \in \mathbb{R}^{48\times48\times3}$, the goal RGB patch $\boldsymbol{g} \in \mathbb{R}^{48\times48\times3}$, the current positional encoding $\boldsymbol{p}_t \in \mathbb{R}^{256}$ of the agent (see Section 4.2.2), as well as the hidden state $\boldsymbol{h}_t \in \mathbb{R}^{256}$ of the LSTM which is a part of the policy network (see Section 4.2.3).

The **action space** consists of the possible actions that the agent may take in any given state. In this thesis, we have considered two different forms of action spaces, a discrete and a continuous one (cf. Section 3). The discrete agent has 8 possible actions a_t at time step t, reaching its entire 8-neighborhood, hence $a_t \in \{1, \ldots, 8\}$. The continuous agent is allowed to take an arbitrary step in both x- and y-directions, meaning that $a_t \in \mathbb{R}^{2\times1}$.

The **reward structure** is subdivided into two main types, as follows.

- *Terminal* rewards are awarded at the end of a trajectory, either when the goal patch is found or when the maximum number of steps T is reached.
  - *Success* - A positive reward awarded to the agent when it successfully reaches the goal patch. Otherwise zero.
  - *Difficulty* - A positive reward awarded when the agent successfully reaches the goal patch. However, it scales with the difficulty of the just solved game, i.e., the actual reward awarded to the agent is the distance from start to goal multiplied by a constant.
  - *Failure* - A negative reward awarded when the agent fails to reach the goal patch within the maximum number of steps. If the agent succeeds in finding the goal patch, the *failure* reward is zero.
- *Incremental* rewards are awarded at each time step of the agent, including the step that leads to the terminal state.
  - *Step* - A negative reward awarded at each time step constantly.
  - *Closer* - A positive reward awarded when the agent performs an action that results in it moving closer to the goal patch. If the action has moved the agent further away from the goal, the *closer* reward is set to zero.
  - *IoU-scale* - A positive reward awarded when the agent has reached a sufficiently high IoU overlap with the goal patch. Note that the overlap is smaller than the needed threshold for achieving a success. The actual reward given to the

agent is the IoU multiplied with the *IoU-scale* reward when the IoU is above the threshold. Only enabled for the continuous agent.

– *Outside* - A negative reward awarded when the agent moves outside the image. If the agent remains within the image the *outside* reward is set to zero.

The effects of different reward settings in this structure was investigated during the project. However, it was found during early development that the incremental rewards only complicated the reward signal for the agent and did not yield any performance improvement in the measured metrics. Therefore, these metrics are not used during training of the models that are evaluated in Section 7. These simplifications of the available rewards leads to the following expression for the reward r_t:

$$r_t = \begin{cases} r_{\text{success}} + d \cdot r_{\text{difficulty}}, & \text{if current state is terminal and successful} \\ r_{\text{failure}}, & \text{if current state is terminal and failed} \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

where $r_{\text{success}} > 0$, $d > 0$ is the distance from the start to goal patch, $r_{\text{difficulty}} \in \{0, 1\}$, and $r_{\text{failure}} \leq 0$. In the discrete case this distance is the minimum number of required steps from the start to the goal patch. In Section 7 we will specify the parameters r_{success}, d, $r_{\text{difficulty}}$ and r_{failure} of the reward structure (13) that we have used during model training.

4.2 AiRLoc Model Structure

Given that the problem setup described in Section 3 is quite general, there exists many different ways to devise an RL method for tackling the task. Our main AiRLoc model structure, however, consists of the following four parts: i) the raw visual inputs from the current patch $\boldsymbol{o}_t$ and the goal patch $\boldsymbol{g}$ are passed through the *patch embedder* which yields a low dimensional embedding $\boldsymbol{c}_t$ of what the agent currently sees and looks for; ii) the current *positional encoding* $\boldsymbol{p}_t$ is then added to $\boldsymbol{c}_t$; iii) $\boldsymbol{c}_t + \boldsymbol{p}_t$ is then passed to a *temporal unit* which combines the information of what the agent currently observes with previous inputs; and iv) the output from the temporal unit is finally passed to a *decision unit* which maps the information to a probability distribution π of the agent's next action. In Section 7 we will also evaluate and compare different variants of AiRLoc that lack one or several of these components. We next go over the four main parts of AiRLoc in more detail.

4.2.1 Patch Embedder

The RGB visual inputs $\boldsymbol{o}_t$ and $\boldsymbol{g}$ are first handled by the patch embedder. We performed early experimentation with different types of patch embedders, which have in common that they generally consist of a series of convolutional layers followed by rectified linear unit (ReLU) activations and max pooling operations. Non-RL-based pretraining of backbone vision components is very common when tackling RL problems, since it typically yields significantly better end performance [29, 20, 37, 42, 43]. Inspired by these earlier works, different non-RL-based pretraining tasks (depending on which patch embedder is used) have been tried for pre-optimizing the patch embedder network weights, before further refining these weights within the full model structure (cf. Figure 5) during RL training.

Some of the datasets we have used contain segmentation masks for buildings in the aerial view images (see Section 6 for more details). The effect of including this semantic information as an additional input to the RL agent has also been investigated (see Section 7), and in these experiments the patch embedder has been altered to accept inputs with four channels – three for the RGB image, and one for the segmentation mask. Note that in these cases, only the very first layer of the patch embedders have been altered.

DoerchNet. Inspired by the spatial context self-supervised learning approach by Doerch et al. [9], a similar network structure was implemented as a patch embedder. This network will from here on be referred to as *DoerchNet*. The purpose of the original DoerchNet approach is to learn context from images and to gain a conceptual understanding of the images it processes. More precisely, the task presented in [9] is about predicting the location of a goal patch relative to a start patch, where the goal patch is always adjacent to the start patch. Hence, the start patch is first selected randomly in an image and then the goal patch is set to any of the eight adjacent patches of equal size. To limit the network's ability to overfit to patterns in the raw image values, which is shown to severely degrade performance on downstream tasks such as image classification, the patches are separated by a few pixels. This prevents the learning of boundary regions and promotes more global patterns. As can be seen in Figure 6, the setup is very similar to the problem setup of this project. In fact, when the initial distance between start and goal patch is a single step in the discrete 8-dimensional version of our setup, it is identical to this pretraining task of DoerchNet. Naturally, when this initial distance increases, the problem diverges from the pretraining setup. However, we concluded that the more limited task in [9] would likely serve as a valuable pretraining task for the patch embedder – our results also clearly show this in Section 7.

The actual DoerchNet have implemented in this project is a slightly different variant compared to the one proposed in [9]. The authors of the original paper developed an architecture that after pretraining can be used as an individual patch embedder for various downstream tasks, such as image classification. This means their model includes a late fusion architecture that only shares information in the last layers between the two input branches. In stark contrast, the main goal of the patch embedder in the setup of this project is to create an efficient embedding of the *relation between two patches*, not of each patch individually.

Our DoerchNet implementation consists of two parallel branches with four convolutional layers, each with ReLU activation functions and max pooling operations. The start and goal patch are first fed separately into one branch each. To enable early information sharing between the two patches, after two convolutional layers, the outputs of the two branches are concatenated and sent through the rest of their respective branches (i.e. an additional two

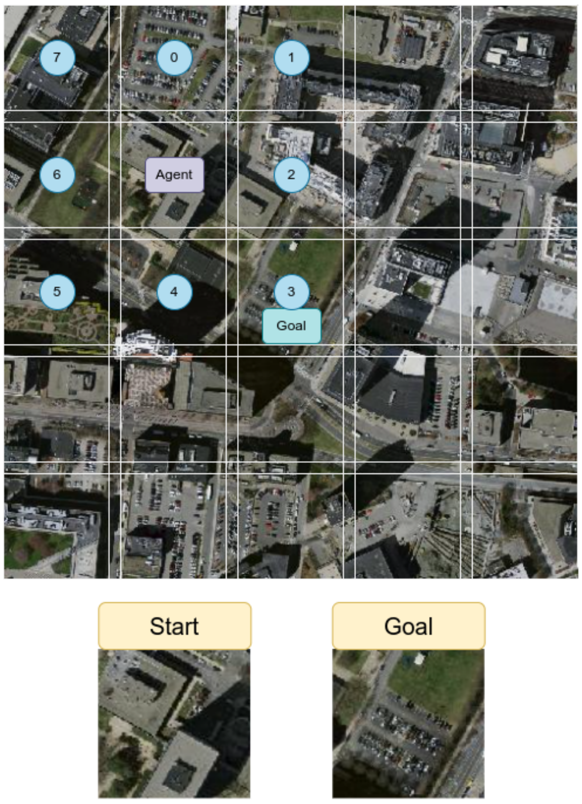


Figure 6: *The setup of the DoerchNet pretraining task. Note the similarities with Figure 2, where the only difference is that the goal patch is guaranteed to be one of the neighboring patches.*

convolutional layers for each). The two resulting 128-dimensional embeddings are then concatenated. These are subsequently passed through two FC layers (with ReLUs in between), where the last FC results in an the eight dimensional output that is fed through a softmax function. These eight outputs correspond to the eight possible relative locations of the goal patch. A visualization of the implemented architecture can be seen in Figure 7. The network is pretrained using the categorical cross-entropy loss, calculated using the network prediction and a one hot encoding of the true relative goal patch direction.

For pretraining our DoerchNet patch embedder, we resort to the previously discussed Adam optimizer, with batches of 256 pairs of image patches and a learning rate of 10^{-3}. When utilizing the network within the AiRLoc agent, the final fully connected layer were discarded and the 256-dimensional concatenated embedding from the two branches was used as input to the next block of the full AiRLoc model.

Segmentation Network. The satellite datasets used sometimes include labeled segmentation masks for buildings in the images. Therefore, we have tried a patch embedder based on

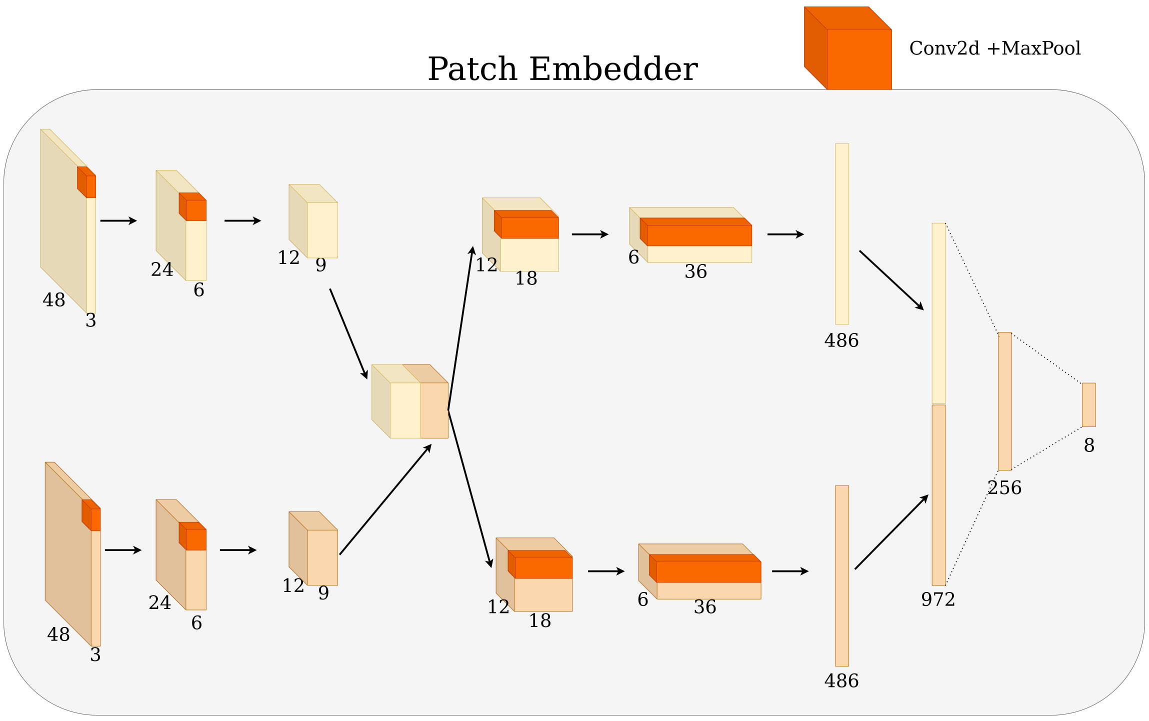


Figure 7: *The model structure for the patch embedder, inspired by the overall structure proposed in [9]. The input images are located to the left and the final embedding to the right (the two different beige colors represent the current and goal patches, respectively). The orange block symbolizes a 3-by-3 convolution followed by a 2-by-2 max pooling which reduces the height and width of the image by a factor of 2. The dotted lines symbolize fully connected layers, and the rightmost rectangle is the classification output. When used as a patch embedder in AiRLoc, the last linear layer is discarded and the embedding size is 256.*

pretraining a fully convolutional network (FCN) to predict this segmentation mask and use an intermediate stage of the network as an embedding for AiRLoc. The main inspiration for this embedder is the U-Net model for biomedical segmentation applications [27]. A publicly available implementation of the U-Net[4] was used as a starting point for this patch embedder. However, the inputs to the network in this project has significantly smaller dimensions than the inputs in the original task. Therefore, the network structure was altered. The final network consist of four downsampling convolutional blocks, which reduce the spatial dimensions of the input image. Leading to a three dimensional embedding space with 64 channels. Then, four upsampling convolutional blocks recreate the input image dimensions. The loss used during pretraining of this patch embedder was binary or regular (depending on the number of classes) cross-entropy loss calculated with the predicted segmentation masks and the labeled masks from the dataset.

[4]https://github.com/milesial/Pytorch-UNet

Worth noting is that this network type was pretrained on a single patch at a time and the learned representation contains no information about the surrounding image or any relation to any other part of the image. In particular there is no relationship prediction between the current patch and the goal patch. The idea in this setup is that learning the segmentation forces the embedder to attend to structure in the image, and learning relevant relationships between the patches is left for later stages in this variant of AiRLoc. In contrast, the DoerchNet embedder learns the spatial relation between two patches directly in the embedder. During training of the the full AiRLoc model, the upscaling blocks of the segmentation embedder are discarded and the latent space representation of the input patch is flattened and sent to the next step of the model. The start and goal patch are therefore handled completely separately by the segmentation embedder.

Training of this network also utilizes the Adam optimizer with batches of 128 patches and a learning rate of 10^{-4}. We have also used the segmentation network during evaluation as an alternative to the ground truth semantic segmentation masks for a specific variant of AiRLoc which obtains semantic segmentation as input (see Section 7). In these cases the pretrained segmentation network predicts the building masks of the patch inputs and appends them to the RGB input. This special variant of AiRLoc then uses this predicted mask instead of the ground truth information.

Autoencoder. Similar to the latent space approach in the segmentation embedder, an autoencoder architecture was developed to capture the vital visual information in the patches. The network structure is very similar to the segmentation embedder implementation with four downsampling convolutional layers followed four upsampling convolutional layers. As previously, the network was pretrained using the same training data as the AiRLoc models, and the patches were sampled in the same way as the start patches during RL training. Using mean squared regression loss, the network was trained to recreate the input patches as accurately as possible. During RL-training the upsampling layers are discarded and the bottleneck after the four downsampling layers is used as the latent space representation of the visual input.

4.2.2 Positional Encoding

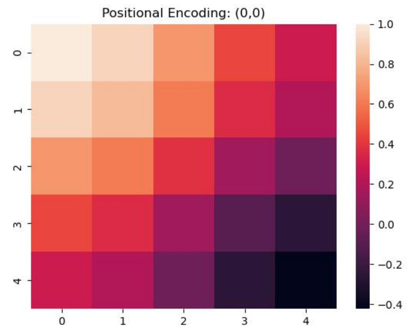
Besides the visual inputs of the current and goal patches, another type of input we have experimented with during the project is different encodings of the position of the agent within the environment. Solving the task proved difficult without any context about the agents location (we also show this in Section 7). Therefore, the positional encoding was introduced, but note that none of the methods receive global positional information (e.g. GPS coordinates), i.e., the positional information is always relative to the search area that constitutes the agents local environment. Such information may be available in many practical use-cases, e.g. during search-and-rescue within a confined area, where a UAV could easily keep track of its location relative to the borders of this area. For all models, this encoding was introduced after the patch embedder step and before the temporal unit, as indicated in Figure 5. Two main types have been experimented with: a basic positional encoding and a version of the Transformer positional encoding [36].

The basic encoding simply keeps track of a 5×5 array (the size is set to match the number of horizontal and vertical grid cells in the task setup), initialized with all zeros, and where the element corresponding to the agent's current position is set to one. This vector is flattened and concatenated to the patch embedding before entering the temporal unit. The strength of this approach is its simplicity. However, early experimentation showed that this approach did not improve results relative to not having any positional information. A possible reason is that the appended values dilute the information of the visual embedding vector.

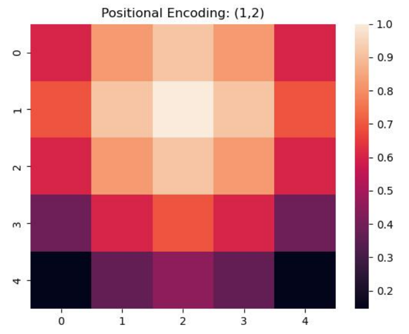
For this reason, another approach – inspired by the positional encoding of the Transformer model – was developed. In this approach a pre-calculated embedding vector

$$PE_{pos,i} = \begin{cases} \sin\left(pos/100^{2i/(d/2)}\right), & \text{i is even} \\ \cos\left(pos/100^{2(i-1)/(d/2)}\right), & \text{i is odd} \end{cases} \tag{14}$$

is used, where the position pos corresponds to the position within the environment in one dimension (in other words, (14) is computed for the x- and y-directions independently), in our case ranging from 0 to 4. Here d is the size of the embedding from the patch embedder. To create a full positional embedding $\boldsymbol{p}_t$ to add to the patch embedding $\boldsymbol{c}_t$, a vector of size d is created. The first half of the vector is filled with the contents of PE_{x-pos} and the second half with the contents of PE_{y-pos}. This vector $\boldsymbol{p}_t$ is then added to the output $\boldsymbol{c}_t$ from the patch embedder. This method generates encodings with high cosine similarity between close positions and decreasing further away, as can be seen in Figures 8a and 8b. This property is then used by the agent to discern where in the search area it is located.



(a) *Cosine similarity of the embedding for position (0,0) with the embeddings of all other positions.*



(b) *Cosine similarity of the embedding for position (1,2) with the embeddings of all other positions.*

Figure 8: *Two examples of cosine similarites between different postional encodings.*

4.2.3 Temporal Unit

The patch embedder creates a low dimensional representation of what the agent sees in the current time step, but with no information about what the agent has seen previously. The hypothesis is that to successfully navigate during longer episodes there is a need for drawing conclusions based on previous as well as current inputs. The temporal unit addresses this problem. Two main options for the temporal unit have been explored, an LSTM module [13] and a Transformer block [36].

LSTM. The maximum sequence of inputs to the temporal module is limited to the number of steps in an episode. For the types of scenarios investigated in this project the number of steps very rarely exceeded 10. This relatively short sequence of inputs – as well as its lower requirement on compute – motivates the usage of the LSTM architecture instead of the more advanced Transformer (described in the next paragraph). Hence the LSTM version of the temporal unit is considered the default setup. It takes as input the 256-dimensional flattened embedding from the patch embedder, along with its positional encoding, and outputs an equally sized vector which is passed to the decision unit (see Section 4.2.4). The hidden and cell states are 256-dimensional vectors that are zero initialized at the start of each episode.

Transformer. Inspired by the impressive performance of the Transformer architecture for image classification proposed by Dosovitskiy et al. [10], an alternative to the LSTM-based temporal module is the Transformer [36]. Instead of the sequential methods in the LSTM where each patch embedding successively is passed to the LSTM module, the Transformer module receives all previously seen patches as input simultaneously. The main motive for implementing this architecture is the Transformer's superior ability to make connections between relevant patches using the learnable attention mechanism. This ability should allow the module to draw conclusions based on early visual inputs in conjunction with late inputs more efficiently than an LSTM. This was implemented using a Transformer encoder based on the version proposed in [36]. The input to this encoder is a variable length sequence with the embeddings of all previously seen patches, including the patch at the current time step. As described previously, the embeddings already contain a positional encoding. All outputs from the Transformer are discarded except the final head corresponding to the input of the current patch embedding. This output is passed to the final linear layers.

4.2.4 Decision Unit

As indicated in Figure 5, the output from the temporal unit is passed to the decision unit, which maps the condensed low dimensional information of the agent to a decision on where to move. The structure of the decision unit is a single linear layer followed by either a softmax activation function or no activation function at all. The purpose of this module is to map the low dimensional embedding to a policy distribution π. This allows for testing the different action spaces without modifying the basic structure of AiRLoc.

4.3 Early Design Choices

Due to the task setup being novel, at least to the best of our knowledge, early parts of the project explored very different approaches to both the task and general model structure. These investigations gave some initial insights which were utilized during the rest of the project. We here mention some key aspects that were settled during early model development.

Action space. Initially, we proposed the idea of using a continuous action space, as we imagine this being most similar to the real world application. However, it proved difficult to tackle the task with the continuous agent. There may be several reasons why the continuous agent did not prove as effective as the discrete one. The main reason may simply be that we spent significantly more time on developing and evaluating discrete agents, and perhaps with more time and effort we could have made the continuous agent work equally well. Another reason could be that the continuous action space was implemented as a 2d gaussian, which is unimodal. A unimodal distribution may be poorly fit for our task setup, where the goal location may often have multiple high-probability modes as to where it is relative to the current location. In other words, given the input patches there might be several locations with a high estimated probability of moving there. This cannot be reflected using a single 2d-Gaussian action distribution. Therefore, the effort was refocused to the discrete 8-dimensional action space setup, and unless explicitly stated otherwise, this is the setup referred to.

Patch embedder. Comparing the effects of different patch embedder architectures, it became evident that the Doerch-style embedder far surpassed all other approaches. Therefore, the Doerch-style embedder was selected as the default patch embedder for all ML-based models. Pretraining and development of the segmentation network was however continued, as it is used as a predictor of the segmentation masks of a particular variant of AiRLoc.

Temporal unit. While the Transformer is a novel and exciting model architecture that has proven to work well for a wide range of sequential problems, we were unable to see any of these performance gains translate to our task. Likely due to limited time and hyperparameter optimization, we were unable to tune the Transformer to match or exceed the LSTM in task accuracy. Additionally, the Transformer architecture proved very computationally heavy and requires longer training time compared with the LSTM-based AiRLoc model. Therefore, the decision was made to abandon the Transformer completely and solely use the LSTM-version of the temporal unit.

5 Baselines

To assess the efficiency and accuracy of our proposed reinforcement learning-based AiRLoc model, it will be compared to several learnable and non-learnable baselines in Section 7. These baselines are evaluated in the same setting as AiRLoc.[5] We now proceed with explaining these baseline methods.

Privileged Random. This is a non-learnable baseline. At each time step, the next move is sampled at uniform random from the eight adjacent patches, with the following two exceptions (privileges, as they are to its advantage). First, any move that would take the agent outside the image is prohibited. Second, whenever possible, the agent is prevented from moving onto a patch it has already visited before. If the agent has to choose between revisiting locations and going outside the image, it always chooses to revisit patches. To be clear, the actions are selected at uniform random with respect to the current set of available actions, based on the mentioned movement restrictions. Note that with these movement limitations, *Privileged Random* is in fact close to an upper bound (in expectation) in terms of results that are obtainable without visual inputs.

Local. This is a learnable baseline. Recall that the DoerchNet embedder architecture is trained to predict the move that brings the agent from its current patch to an *adjacent* goal patch (cf. Section 4.2.1). However, by simply repeating this process in the landing patch it becomes possible to assess how well this baseline fares even for setups where the goal is further away. Therefore, during evaluation of this baseline, the process is performed repeatedly until the model either finds the goal patch or the maximum number of steps T is reached.

Privileged Local. Same as *Local*, but with the same privileged movement restrictions as *Privileged Random*.

Global. This is a learnable baseline that is quite similar to *Local*. More precisely, it is a self-supervised model that given the start and goal patches predicts the location of the goal patch relative to the start patch. In this case, the goal may be arbitrarily far away from the start patch, as shown in Figure 9. For this baseline, the search process is always terminated after the first step. The range of model means that it is always, in theory, able to reach the goal patch in a single step. The purpose of this baseline is to assess whether it is possible to solve the aerial view image-goal localization simply by abstracting it as a simple self-supervised classification problem.

Human. We developed a framework for assessing human performance on the aerial view image-goal localization task. Se details in Section 5.1.

[5]Except for the human baseline which is only evaluated on a subset of the validation set of *Massachusetts Buildings* (see more about datasets in Section 6).

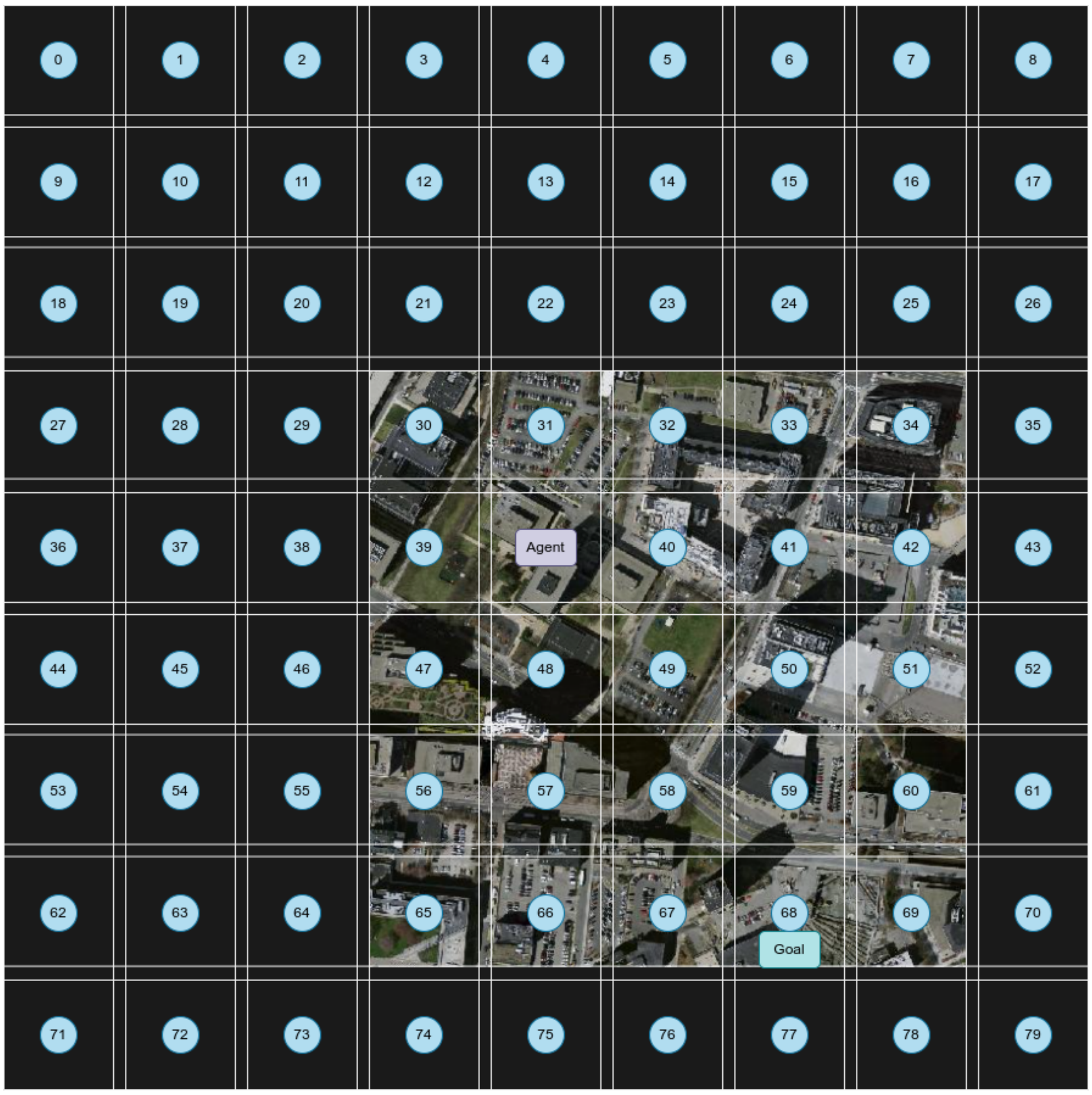


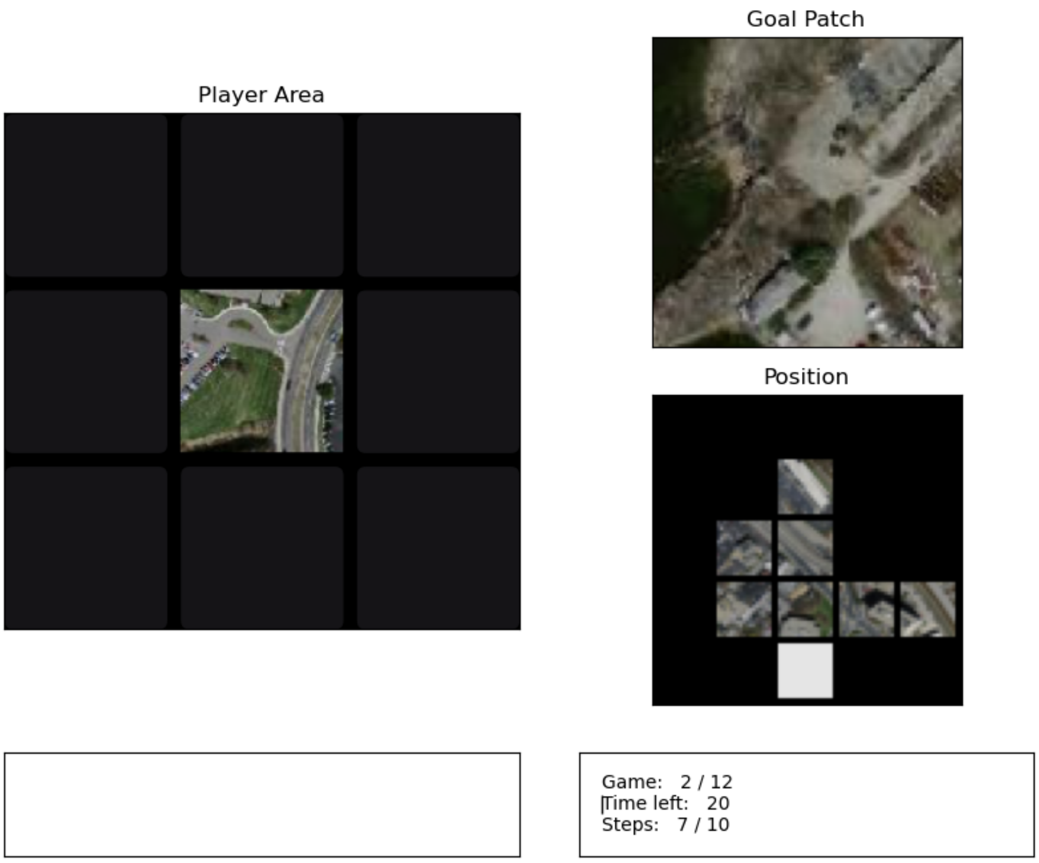
Figure 9: *The agent-centric action space of the* Global *baselines. The model is able to move four steps relative to its current position in any direction, resulting in a 80-dimensional action space. Hence, the baseline is theoretically always able to reach the goal patch in a single step. In this example the correct move is action 68 which results in moving three steps down and two to the right.*

5.1 Human Baseline

To compare the performance of AiRLoc with a human operator in a similar setting, a game version of the task was developed. This game was designed to resemble how AiRLoc perceives the environment, for fair comparisons. Therefore, in addition to receiving the current and goal patches, the human operator is also aware of the borders of the environment, and knows the current position as well as the history of all previously visited positions within the confined environment – see Figure 10. Based on this input, the human operator is tasked with moving to any of the eight adjacent patches (the movement is selected by clicking with a mouse cursor on one of the eight dark squares surrounding the current location). In fact, the human operator can even see all the previously visited patches, while this information is not provided to AiRLoc. We decided to provide humans with this additional information as they have not been trained for the task at hand.

The age span of the 19 people who participated is between 14 and 42 years, with an average of 26.4 years and a median of 25 years. Of the 19 people, 13 were men and 6 were women (i.e., 68% and 32%, respectively). For each human operator, 12 unique images from the *Massachusetts Buildings* validation set were used (see Section 6), as well as a few sample images for the player to get acquainted with the controls of the game – the participants are able to practice as long as they desire, and no statistics are tracked during this warm up phase. The exact games provided span a subset of the games that AiRLoc and the other baselines are evaluated on, to ensure the most fair comparison. However, each human is not tested on the entire validation set, which results in a higher variance for the human performance evaluation. The difficulty settings were split equally over these twelve games, with three games per difficulty (recall that difficulty refers to the distance between the start and goal patches, ranging from 1 to 4 steps away). Statistics similar to those recorded for AiRLoc and the other baselines were collected by the game (see Section 7.1).

Even though the human setup is very similar to that of AiRLoc, there are key differences and concepts that do not translate well to a human controlled setup. First, the positional encoding used for AiRLoc is difficult to translate to human visual processing, and instead a plain map of the position was implemented (this is arguably in favor for the human baseline, as the participants receive explicit information from past locations, different to AiRLoc). Second, the human participants have not extensively trained on the task like AiRLoc, and their visual systems are likely not as tailored towards handling the low resolution of the patches. On the other hand, humans have a lifetime worth of generic visual pretraining which AiRLoc lacks. Third, the human participants have a limited time to complete each game (60 seconds). Such a time limit was used for the convenience of the participants – we wanted to avoid that the participants feel like they had to spend several minutes per individual action in order to squeeze out the maximum possible performance. The 60 second time limit was assessed to be more than sufficient for completing each game, and the participants that we asked fully agreed with this. These discrepancies, in conjunction with the limited number of human controlled trajectories, somewhat limit the reliability of this baseline. Nonetheless, it is still a useful indication of the human performance for this novel task.



Game: 2 / 12
Time left: 20
Steps: 7 / 10

Figure 10: *An example of the performance evaluation setup for the human baseline. Each participant is given a set of 12 different such games, and there is no overlap in the games played by different participants. The system tracks the number of steps taken, success rate and time spent per game.*

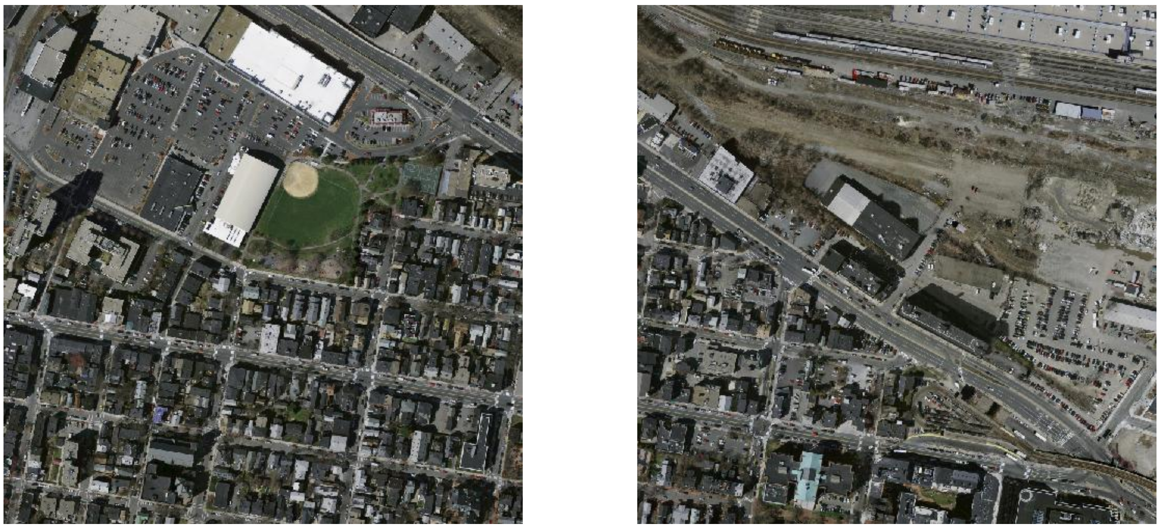


Figure 11: *Two examples of images from the Massachusetts dataset.*

6 Datasets

When selecting datasets, a mix of different environments was sought after with depictions of areas with both urban and rural features. The *Massachusetts Buildings* dataset [18] contains images of Boston as well as the surrounding suburban and forested areas. It shows houses, roads and other clearly identifiable man-made structures as seen in the example in Figure 11, but also woods and less developed regions. This dataset also contains a segmentation mask indicating where buildings are located – this was used for certain models that receive semantic segmentation as input (see Section 7).

We use *Massachusetts Buildings* as the main dataset for model development and evaluation, and thus we split it randomly into training, validation and test sets (these consist of 70%, 15% and 15% of the data, respectively). The images are initially cropped to fit the desired real world scale, after which they are split according to the training, validation and test ratios. This generates 832 training, 178 validation, and 178 test images. The fixed evaluations consist of $4 \cdot 178 = 712$ validation and test images, respectively, i.e. one setup per difficulty and image.

The *Dubai* dataset [33] also contains depictions of man-made structures, as evident in Figure 12. However, the surrounding areas are not temperate, suburban neighbourhoods and woods, but dry deserts. This out-of-domain dataset is used to assess the generalizability of the developed methods.

Figure 12: *Two examples of images from the Dubai dataset.*

7 Experimental Results

Our proposed AiRLoc agent, its variants, and the learning-based baselines are implemented in and trained using PyTorch on the training partition of *Massachusetts Buildings*. During all training and evaluation we have used GPU-equipped computers (Titan V100). Training an AiRLoc agent, which contains about one million parameters, takes about 80 hours. The machine learning based baselines, segmentation network and the patch embedder all require circa eight hours of training. Due to the relatively small size of all models utilized in this project(AiRLoc and AiRLoc sem. seg, has 827 698 and 948 008 trainable parameters respectively), parallel training of multiple networks on a single GPU is possible and was used extensively. Thus a total evaluation of an agent using the test or validation split of any of the datasets takes about one minute. For AiRLoc and its variants, we train the policy networks using REINFORCE (cf. Section 2.4) with a batch size of 64, a maximum trajectory length T of 10, a learning rate of 10^{-4}, and a discount factor γ of 0.9. Employing data augmentation for the training set and using random initial setups (i.e random start and goal locations) for the training split generates more than 60,000 unique batches. We also employ within-batch reward normalization based on distance left to the goal. Hence, all rewards from states with distance one from the goal in the current batch are normalized to mean zero and standard deviation one, and this is repeated for rewards associated with distances two, three and four as well.

Each model was trained until convergence on the validation set in the sense that the success rate remains approximately constant for several thousand batches, or until the model has trained for a maximum of 50,000 batches. To verify the performance of AiRLoc is not overly dependent on the random initialization of its model parameters, five different random seeds were used during model training. Four out of five models reached a very similar result[6] on the validation set (see Appendix B), and the median performing AiRLoc agent is evaluated in the following results and comparisons. Unless specified otherwise, all models are evaluated in deterministic mode meaning that the most probable action is selected in each step. There is also the possibility of running the models in stochastic mode meaning that the actions are sampled from the policy distribution, but in general this was not used.

When evaluating and comparing models and baselines, it is critical that they have the exact same environmental conditions, such that the task is not more difficult for one model than for another. Therefore, we create a list containing one initial setup (start and goal coordinates) for each difficulty (initial distance between start and goal) for each image in the evaluation sets (validation and test). All models and baselines are evaluated following the list corresponding to each dataset. This allows for consistent evaluation of the models, giving a reproducible indication of which model performs best.

The main evaluation (see Section 7.2) consist of four different setups, all except one on *Massachusetts Buildings*:

- an evaluation on the test set with trajectory length $T = 10$;
- an evaluation on the validation set with $T = 10$;
- an evaluation on a different dataset (*Dubai*) with $T = 10$.
- an evaluation on the test set with longer trajectories ($T = 100$)

Two main reward structures were investigated, cf. Equation (13). The only difference between the two reward structures is that one of them has one more component and is thus referred to as the *complex* reward. Similarly, the simpler reward is referred to as the *simple* reward. The simple reward

$$r_t^{\text{simple}} = \begin{cases} 3 & \text{if current state is terminal and successful} \\ -1 & \text{if current state is terminal and failing} \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

treats each trajectory equally, meaning that the agent is equally rewarded independently of the difficulty of the setup. In contrast, the complex reward

$$r_t^{\text{complex}} = \begin{cases} 3 + d & \text{if current state is terminal and successful} \\ -1 & \text{if current state is terminal and failing} \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

takes the difficulty $d \in \{1, 2, 3, 4\}$ into account where the more difficult setups get a larger reward. Note that the penalty for not converging stays the same for both types of rewards.

[6]One of the five seeds resulted in a somewhat lower success rate, but it is still higher than the best comparable non-RL-based method.

Unless otherwise specified, all AiRLoc models that are evaluated have been trained with the simple reward shown in Equation (15)**.**

7.1 Evaluation Metrics

To evaluate the performance of the various models, we introduce five metrics. These metrics are designed to capture both the overall performance and details regarding model behaviors.

- *Success* measures the percentage of setups where a model reaches the goal patch (higher is better). See examples of a successful trajectory in Figure 3a and an unsuccessful one in Figure 3b for further clarification.
- *Steps* is the average number of actions taken by a model (lower is better). For all failure trajectories this metric will reach the maximum value (mostly $T = 10$), while for successful trajectories it is simply the number of actions required to reach the goal patch. Hence, the steps metric lies in the range $[1, T]$.
- *Step ratio* measures the average ratio between the minimum required number of steps and the taken number of steps (higher is better). Hence it is a value in the range $(0, 1]$, where 1 corresponds to always taking the minimum number of steps to reach the goal. This metric is ill-defined for the trajectories that do not reach the goal patch; hence it is *only computed for successful trajectories.* This metric is also ill-defined for the *Global* baselines, as it can reach the goal patch in a single step from any location, which is not a fair comparison – hence we omit reporting this metric for these baselines.
- *Residual distance* is a metric tailored to the unsuccessful trajectories (lower is better). This metric measures the average distance (in steps) between the final location of the agent in relation to the goal location.
- *Time* is the mean time per action in milliseconds.

7.2 General Performance

In Table 1 we compare our principal AiRLoc agent, presented in Figure 5, to the random, local and global baselines. The results presented in this table are from the test set of *Massachusetts Buildings.* Note especially that in regard to success rate, AiRLoc surpasses the baselines.

Table 1: *Results on the test set of* Massachusetts Buildings*, where models are allowed to take at most 10 actions per trajectory. Note that AiRLoc surpasses all baselines in terms of success ratio. While the step ratio is slightly better for the local restart models, the success rate of these models is too low to be comparable to AiRLoc. Further, the random baseline is the only approach with better residual distance results, which is likely due to the fact that the random agent is unable to go outside the image borders. As expected the non-learnable random baseline is the fastest while AiRLoc with the additional segmentation network is the slowest.*

Agent type	Success (%)	Step ratio	Steps	Residual dist.	Time (ms)
AiRLoc	59.2	0.61	7.0	2.5	14
AiRLoc (sem. seg.)	61.0	0.56	7.1	2.5	20
Privileged Random	41.0	0.39	8.0	1.6	6.1
Local	12.6	0.79	9.0	6.4	11
Privileged Local	51.5	0.57	7.5	2.5	13
Privileged Local (sem. seg.)	59.7	0.58	7.3	2.5	18
Global	7.0	-	1.0	2.9	17

Table 2 shows a more comprehensive study of AiRLoc. This table includes both ablations and expansions of AiRLoc. To avoid over-exploiting (and hence potentially overfitting to) the test set, this table presents results obtained through evaluations on the validation set of *Massachusetts Buildings*. The default model in this table is the same AiRLoc as the one presented in Table 1 and the contents in the parenthesis describe what components have been added or removed. The focus of the abltion study has been to asses the effect of the various submodules on the success rate. Hence, the time statistic was deemed non-relevant and therefore omitted from Table 2. We next describe these various AiRLoc variants.

- *Complex* is the complex reward structure Equation (16) that provides a bonus for more difficult setups – all other models are trained using the simple reward structure Equation (15)[7].
- *Priv.* has the same heuristic privileges as the random agent (i.e. never moves outside and always avoids visiting past locations), but the actions are taken according to the predicted policy subject to these movement constraints.
- *Sem. seg. GT* has an additional ground truth semantic segmentation channel as input to the model. This segmentation mask indicates the location of buildings within the patches. Note that this model uses ground truth segmentation masks during evaluation.
- *Sem. seg.* is semantic segmentation with prediction, identical to the other sem. seg. except that the segmentation mask is predicted by a pretrained U-Net from the

[7]Additional results from this setup are presented in Appendix B, but in general the simpler reward proved superior.

visual patch inputs. This model does not use any ground truth information during evaluation.

- *Frozen emb.* uses a frozen patch embedder, meaning that the weights of the patch embedder are not further refined during RL training.
- *From scratch* is a setup where the entire AiRLoc agent, including the patch embedder, is trained solely using RL (no pretraining of patch embedder).
- *No LSTM* is an AiRLoc agent where the temporal unit is removed and the patch embedder is directly connected to the decision unit.
- *No RGB* is an AiRLoc agent where the input patches have been replaced by all zeros, meaning that the only information passed to the agent is the positional encoding of the current location.
- *No pos. enc.* is an AiRLoc agent without any positional encoding.

Table 2: *Results on the validation set of* Massachusetts Buildings*, where models are allowed to take at most 10 actions per trajectory. First, note that AiRLoc outperforms humans on average. Second, most ablations result in a distinct performance decrease, noting especially the severe performance drop when omitting pretraining* (from scratch) *and positional encoding – these modifications even result in AiRLoc variants that perform worse than the random baseline. One also sees that visual (RGB) input and temporal processing are critical to performance. Moreover, the simpler reward yields better results than the complex one, and adding the movement constraint privileges of the random baseline does not yield any significant improvements for AiRLoc. This indicates that AiRLoc has learnt relevant 'movement restrictions' during RL training. The least impact on performance seems to be the freezing of the embedder as this shows no clear effect on the results.*

Agent type	Success (%)	Step ratio	Steps	Residual dist.
AiRLoc	58.6	0.60	7.1	2.5
AiRLoc (complex)	53.5	0.60	7.5	2.3
AiRLoc (priv.)	60.7	0.59	7.0	2.6
AiRLoc (sem. seg. GT)	64.6	0.61	6.8	2.3
AiRLoc (sem. seg.)	63.5	0.59	7.0	2.3
AiRLoc (frozen emb.)	60.7	0.60	6.9	2.5
AiRLoc (from scratch)	38.9	0.59	8.2	2.3
AiRLoc (no LSTM)	42.0	0.58	8.0	2.3
AiRLoc (no RGB)	40.7	0.61	8.2	2.6
AiRLoc (no pos. enc.)	33.0	0.49	8.7	2.5
Privileged Random	39.0	0.51	8.2	2.6
Local	14.4	0.83	8.9	6.4
Privileged Local	51.2	0.59	7.5	2.6
Privileged Local (sem. seg.)	59.8	0.57	7.7	2.6
Global	8.0	-	1.0	2.9
Human	55.7	0.54	7.6	2.3

As previously mentioned, the main training and validation datasets depict urban regions in Boston, Massachusetts. To investigate the generalizability of the models, we also evaluate them on the secondary *Dubai* dataset, see Table 3. The AiRLoc model in this table is the same as in Table 1 and Table 2. It is important to emphasize that the agent is *not trained* on any data in the Dubai dataset – it is only evaluated on this data. Still, the model performance remains in the same region as before, indicating that AiRLoc is robust to environment changes. In particular, as before, AiRLoc outperforms the other learning-based methods and the heuristic random baseline.

Table 3: *The AiRLoc model and baselines evaluated on the previously unseen* Dubai *data (i.e., all learnable models were only trained on* Massachusetts Buildings*). AiRLoc generalizes well to the unseen data from Dubai, noting only a slight decrease in performance in comparison to* Massachusetts Buildings *– for example, the success rate for AiRLoc drops less than 2 percentage points. The baselines also remain fairly constant in terms of their performance relative to* Massachusetts Buildings*.*

Agent type	Success (%)	Step ratio	Steps	Residual dist.
AiRLoc	56.9	0.60	7.2	2.8
Privileged Random	41.0	0.51	8.0	2.5
Local	15.0	0.87	8.9	6.4
Privileged Local	55.9	0.63	6.9	2.6
Global	5.6	-	1.0	3.0

Another important aspect to analyze is how the models perform when increasing their maximum number of allowed steps, T. We present such results in Table 4, where the models are allowed to run for up to 100 steps in comparison to the previous 10. As this is an out-of-domain setup for AiRLoc – which is always trained assuming at most 10 steps per trajectory – there is also the case for using a privileged AiRLoc variant which has the movement priviliges of the random baseline (i.e. it avoids moving outside and on previously visited locations). As can be seen, adding such privileges in this longer trajectory setup improves the performance of AiRLoc, which it did not for the 10-step evaluation in Table 2.

Table 4: *Results when allowing a virtually unlimited number of steps (100) on the test set of* Massachusetts Buildings*. This is reported as* (stochastic | deterministic) *in each cell, where the stochastic models sample from the policy distribution while the deterministic select the most likely action. These results suggest that the best self-contained model is AiRLoc. When adding the movement privileges of the random agent to other models it seems that the local baseline with segmentation ground truth slightly outperforms our RL model under the same conditions – however, it does not guarantee convergence in the deterministic case. An important note to make here is that like the out-of-domain data of Dubai, this evaluation also represents a severe out-of-domain setup for AiRLoc, which is always trained assuming at most 10 steps per trajectory.*

Agent type	Success (%)	Step ratio	Steps
AiRLoc	91.8 | 78.2	0.42 | 0.49	21.0 | 28.3
AiRLoc (priv.)	100 | 100	0.37 | 0.38	11.7 | 11.3
AiRLoc (sem. seg.)	92.9 | 82.5	0.42 | 0.46	19.3 | 24.4
AiRLoc (sem. seg. priv.)	100 | 99.9	0.40 | 0.41	10.8 | 10.5
Privileged Random	99.9 | -	0.30 | -	14.5 | -
Local	20.8 | 12.4	0.53 | 0.81	80.9 | 87.9
Privileged Local	99.0 | 89.6	0.34 | 0.41	13.8 | 19.8
Privileged Local (sem. seg.)	100 | 97.1	0.47 | 0.52	9.3 | 10.5

7.3 Agent Behavior

In this section we provide some insights for how AiRLoc behaves in various situations. As before, unless otherwise specified, we investigate the AiRLoc agent described in Figure 5 which does not have semantic segmentations as input, and is trained with the simpler reward structure Equation (15). Example trajectories of this AiRLoc agent are shown in Figure 13 (on the validation set of *Massachusetts Buildings*), and more can be found in Appendix C.

During evaluation (and training), the agent encounters setups with different difficulties, in that the start to goal distance varies between episodes. The probability of success in solving the task for different difficulties may naturally vary, e.g. since it is likely more difficult to reach the goal when it is far away. To assess this quantitatively, we show in Figure 14 the success rates of AiRLoc and baselines versus episode difficulty. This data is gathered from the evaluation of AiRLoc and baselines used for Table 1 (i.e. the validation set of *Massachusetts Buildings*). For the respective models, dashed lines correspond to the global averages over all difficulties.

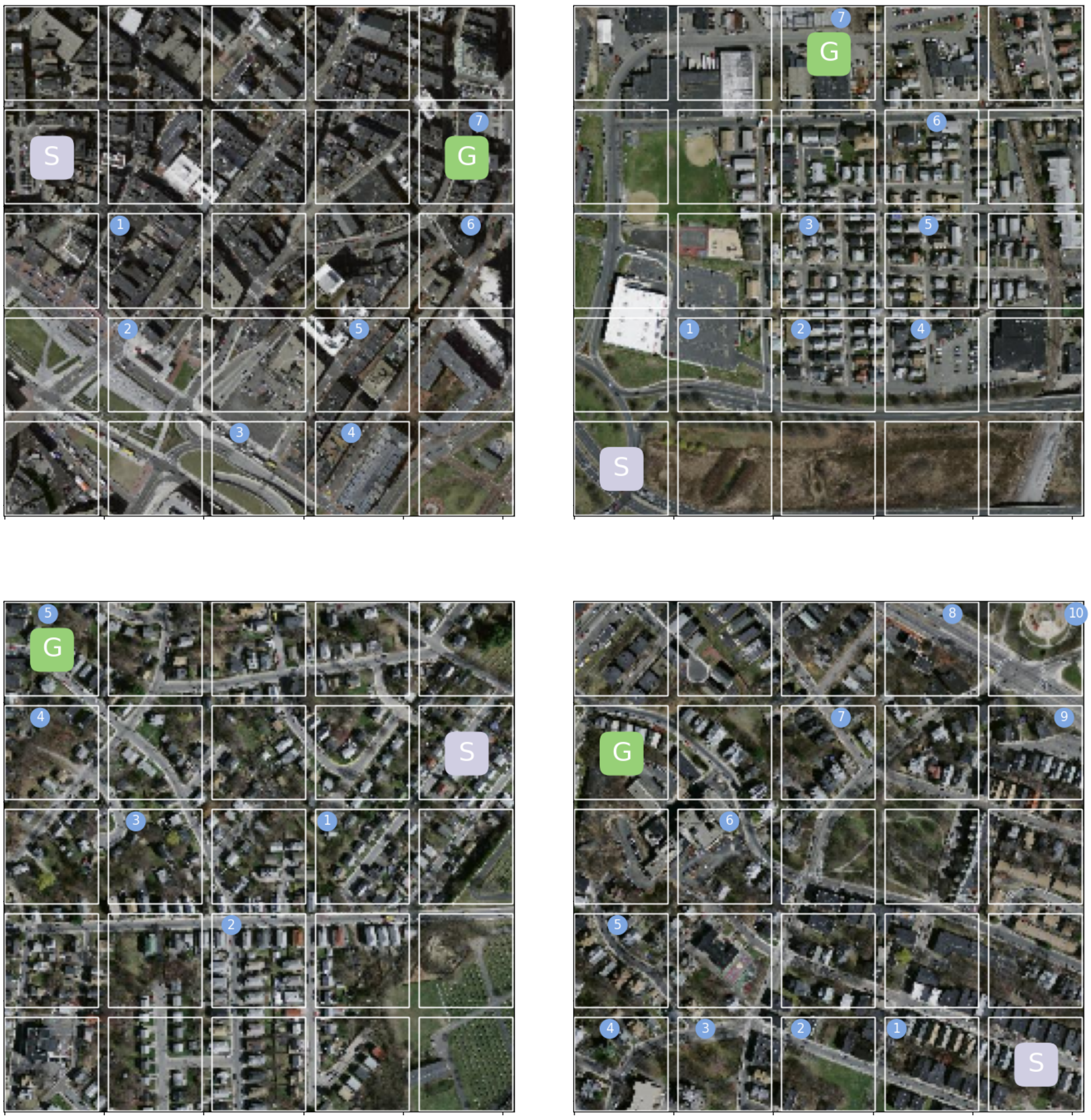


Figure 13: *Four example trajectories of the AiRLoc agent, where "S" indicates the start patch, "G" indicates the goal patch, and the small numbered circles correspond to the patches visited before reaching the goal. Note that the underlying search area is* never *observed in its entirety (not even at low resolution), i.e., AiRLoc operates solely based on the sequentially observed partial glimpses and the goal patch. AiRLoc is successful in all but the lower right case. Note in particular the exploration behavior which often resorts to diagonal movements.*

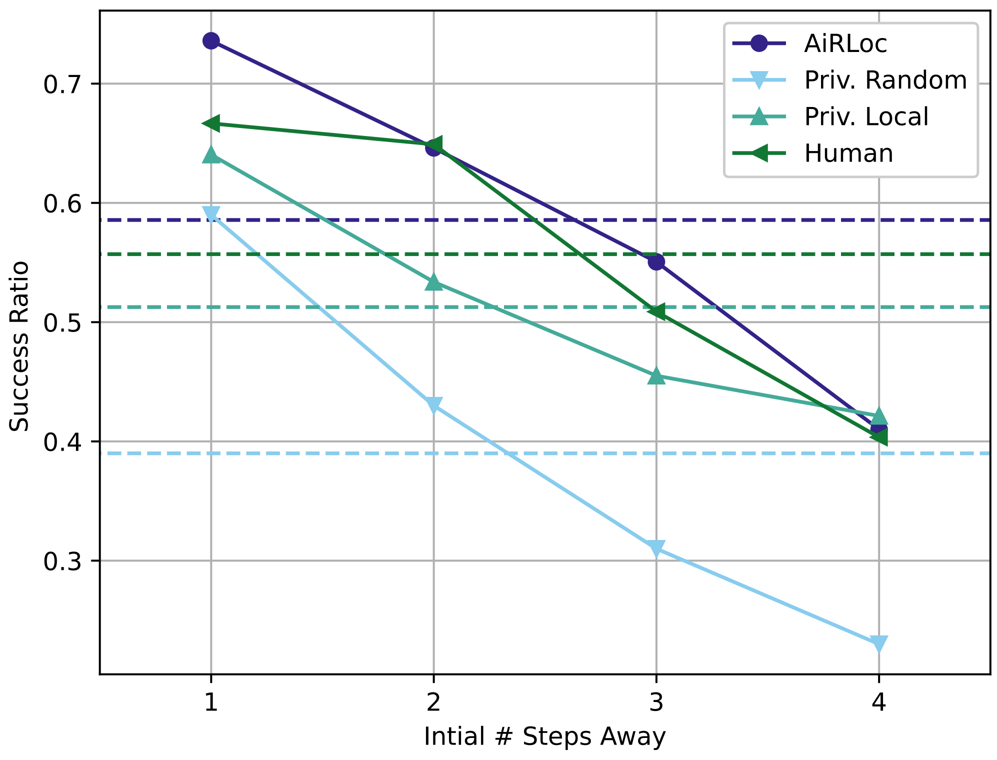


Figure 14: *The success rate of AiRLoc and various baseline methods during evaluation on the validation set of* Massachusetts Buildings *versus episode difficulties. Dashed lines (in matching colors) correspond to global averages over all difficulties. Unsurprisingly, all approaches are generally more successful at episodes where the agent starts closer to the goal. Our AiRLoc agent is consistently better than the non-human baselines, and also surpasses or matches the human baseline independently of episode difficulty.*

Additionally, to examine the behavior of the agent in specific situations, the policy distribution in a particular initial start state is inspected. More specifically, a new setup is devised where the trained AiRLoc agent has a fixed starting location in the lower left corner of the search area. Then, for each image in the validation set of *Massachusetts Buildings*, the initial policy distribution for each goal location along either the left or the bottom border of the search area is examined. The probabilities of taking each action during the first step of the trajectory are presented in Figure 15, where values are separated by the distance from the starting position (in the lower left corner) to the goal. Note that the y-axis in the respective subfigures are differently scaled. In both scenarios, when the goal is adjacent to the starting location, the agent is most likely to take the action that results in immediate success. Similarly, in both cases the same type actions ('Up Right' or 'Right') are most likely to be selected when the goal is further away, which indicates a generic exploratory behavior when AiRLoc is less confident of the goal location.

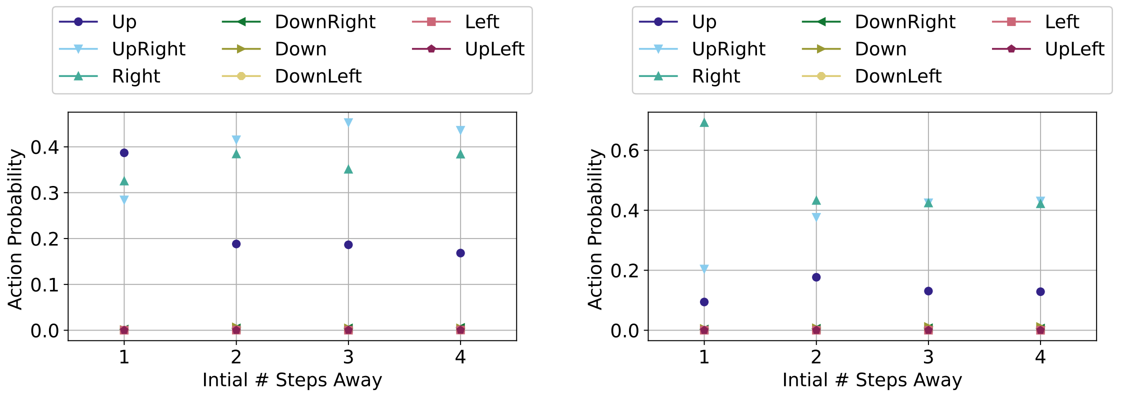


Figure 15: *To the left we show the probability of the 8 different actions of AiRLoc versus initial goal distance, in the cases where AiRLoc starts in the bottom-left corner of the environment and the goal is straight above the agent (1 to 4 steps away). In each case, a shortest trajectory is obtained by taking the 'Up' action until reaching the goal. When the goal is one step above the start location, we see that the most probable first action is indeed 'Up', resulting in immediate task completion. As the initial distance to the goal increases, other first actions ('Up Right' and 'Right') become more probable than 'Up', which indicates that when AiRLoc is not confident of the goal location it may resort to a more generic exploratory behavior. Note that, due to the positional encoding, AiRLoc assigns virtually zero probability to actions that takes it outside the search area. Similar results can be seen on the right, where the the goal location is instead to the right of the agent.*

8 Discussion of Results

To reiterate, the principal objective of this master's thesis is to assess whether reinforcement learning (RL) provides a suitable framework for tackling our newly proposed aerial view image-goal localization task. In the following subsections we investigate – based on the experimental evaluations in Section 7 – this objective, and discuss the general properties of the proposed AiRLoc agent. This includes comparisons to the baselines, as well as investigations of the action space and of the overall behavior of AiRLoc. We will also draw conclusions regarding the generalization and limits of AiRLoc with respect to out-of-domain data and longer trajectories.

8.1 Comparisons with Baselines

Table 1 shows that AiRLoc is able to solve the proposed goal localization task more efficiently than all baselines, be them learnable or heuristic. This result confirms the validity of our RL-based approach to the aerial view-goal localization task, which is the main objective of this master's thesis. Analyzing Table 1 further, it is evident that the one-shot global approach to the proposed task performs abysmally and that sequential solutions are needed. The global approach to the problem – which is trained to always move to the goal in a single step – yields a very low success rate. Hence, the ability of the global model to move to any position in the search area does not seem to compensate for the fact that the very large action space renders most available moves useless (most actions of this approach result in moving the agent outside the search area, as shown in Figure 9). This indicates that our proposed task is simply too difficult to be solved as a one-shot classification problem, which is what the global method abstracts it into.

The local ML baseline performs better than the global one. This is most likely related to the fact that the local pretraining task, in which the start and goal locations are always adjacent, is inherently more learnable as a one-step prediction task, compared to the case when the start and goal can be arbitrarily far apart. Still, AiRLoc is far superior to the local baseline(59.2 % and 12.6 % success rate respectively), which indicates that any ML-based solution to the aerial view image-goal localization task requires consideration of the temporal dimension. One way to force any model to incorporate information about the temporal aspect of the problem is to simply add the privileged heuristic used by the random baseline. By doing this the algorithm is "aware" of the previously visited locations which is a form of temporal information. Using this heuristic the performance of the local baseline *drastically* improves, surpassing that of the random agent, while still remaining inferior to AiRLoc. AiRLoc only benefits slightly from this heuristic, confirming that its temporal unit is capable of learning a similar behavior.

Although typically beneficial, using the priviliged movement heuristic does not always lead to the optimal action being taken. By removing the option of moving to a previously visited locataion, the agent is denied the possibility of taking shortcuts to new regions of the search area by crossing a previous path. Under unfavorable circumstances this could lead to areas of the environment being closed of. This is likely to become especially problematic if one wants to generalize our setup to multi-goal localization, wherein not a single but multiple image-goals are provided. In such a setup, it is obvious that an optimal search trajectory may involve visiting the same locations multiple times. This points to the general undesirability of strictly enforcing such a behavior. Additionally, using the local baseline with privilege is reasonably efficient in the current 5x5 grid environment, as shown in Table 2. However, it is unlikely to generalize to larger grids since this would probably require a more sophisticated search strategy, similar to the one we believe AiRLoc employs. This leads to the conclusion that utilizing RL methods to learn effective strategies in conjunction with a temporal unit is the preferred way to tackle the aerial view image-goal localization problem. This is also confirmed by Tables 1, 2 and 3.

The execution time of AiRLoc is comparable to the simpler ML-based baselines, as also seen in Table 1. This is important for the potential application in a UAV setting as this would constitute a time critical execution environment. Also noticeable is the considerable increase in time per action when utilizing the privileged heuristic. Although no effort was spent to optimize this operation for speed and its implementation can probably be improved, the effect should be noted.

8.2 AiRLoc Dataset Generalization

Comparing the results presented in Table 1 and Table 2 it is evident that AiRLoc performs well on different splits of previously unseen data from the same dataset. Additionally, model generalization is confirmed by the results presented in Table 3. AiRLoc, even as it is only trained on *Massachusetts Buildings*, performs similarly on the *Dubai* dataset. This shows that AiRLoc is robust to domain shifts in the visual environment, which bodes well for the applicability of the model in new settings.

8.3 Benefits from Mid-level Vision Capabilities

Also evident from Tables 1, 2 and 4 is that the addition of ground truth building segmentation mask further improves the performance of AiRLoc. This could be an indicator that the patch embedder does not quite capture all relevant information from plain RGB input. Furthermore, using a separate U-Net to predict this semantic segmentation also improves performance relative to the RGB-only AiRLoc variant, but slightly less so than the ground truth variant. The fact that this performance drop (predicted versus ground truth segmentation) is only marginal indicates that AiRLoc is robust with respect to the segmentation input – recall that this AiRLoc variant is *trained* using ground truth segmentation, and that we switch to the U-Net predicted segmentations during evaluation. However, the increased performance from mid-level vision capabilities – which has also been observed in earlier work, e.g. [29] – comes with the caveats that an additional network needs to be pre-trained and the increased complexity of the model. Additionally, as can be seen in Table 1, the inclusion of the semantic segmentation module significantly increases time per action.

Also note that the pretraining of a semantic segmentation network requires an annotated dataset which renders the AiRLoc training cycle no longer entirely self-supervised. Hence, the addition of this submodule should be considered optional.

8.4 Ablations

The ablation studies in Table 2 show that the various components and design choices for AiRLoc are critical to performance – removing any component causes a large performance decrease. Note especially the result of omitting the pretraining phase of the patch embedder and instead training the full AiRLoc agent from scratch with RL. This approach results in a large drop in performance; it even becomes worse than not receiving any visual input at all (*no RGB*). This confirms that the pretraining of vision systems in RL is critical, as has also been observed in earlier works [29, 20, 37, 42, 43]. A possible reason for this is that the RL loss function is not sufficiently dense, or informative, to train the patch embedder from scratch. The policy gradient loss function only gives a notion if an action was beneficial or not, compared to all other actions taken during the batch. In contrast, the cross entropy loss function, which is used during vision pretraining, compares the prediction of the network with the absolute correct answer. This results in a much more stable gradient to the network during training. This ambiguity in the policy gradient loss provides one explanation for the general need for vision pretraining in RL.

The importance of pretraining the patch embedder is further confirmed by the fact that freezing a pretrained embedder during RL training shows no significant impact on the results. This indicates that the visual representation from the pretraining is sufficient to build a reliable downstream RL policy. In general, fine-tuning the embedder weights may have both positive and negative effects. On one hand, fine-tuning the embedder could improve performance due to it becoming tailored for particular RL task at hand. On the other hand, there is a risk of corrupting the embedder during the initial stages of RL training, where the rest of the system is untrained. During the initial RL training phase, when the temporal and decision unit have randomly initialized weights, the resulting trajectories collected would be of very low quality, i.e. contain mostly unsuccessful trajectories. While these unsuccessful trajectories are most likely due to the untrained parts of the network, *all* weights – including those of the embedder – will be updated according to the gradient of the loss, and this could potentially corrupt the pretrained embedder. However, Table 2 shows that AiRLoc is not affected positively nor negatively, as the frozen and unfrozen embedder perform similarly.

The component with the largest impact upon the success metric is the positional encoding. Table 2 shows that without the positional encoding, the performance of AiRLoc does not surpass the random baseline. The most obvious reason for this performance drop is simply that AiRLoc has a much harder time learning to avoid moving into previously visited locations, as well as outside the search area. The LSTM-based temporal unit is also proven by the results in Table 2 to be a critical part of the AiRLoc model. Thus, unsurprisingly, the ability to integrate evidence over time, and being aware of the current position within the search area, are vital for the performance of AiRLoc.

8.5 Longer Trajectories

Training in a setup that allows for at most 10 steps, and subsequently evaluating in a setup that allows for at most 100 steps, the original AiRLoc variant performs worse than the random baseline, as shown in Table 4. While AiRLoc outperforms its supervised learning counterpart *Local (restart)*, it is not directly comparable to the privileged random search baseline. Note that AiRLoc is not guaranteed to find the goal in a 5×5 grid containing 25 locations when taking 100 steps, which is obviously undesirable. However, for the successful trajectories the step ratio is significantly better than for the random baseline. This suggests that AiRLoc's strategy is focused on exploring and finding goal patches efficiently, rather than guaranteeing full exploration. This is not surprising, since AiRLoc is inherently not trained to fully explore the search areas, as during training it is allowed to take at most 10 steps. In particular, this means that AiRLoc is only trained to temporally process trajectories of at most 10 steps, and beyond this its memory may be unreliable, resulting in re-visiting past locations with a higher probability.

Much of the reason for the success of the random agent stems from its privilege that ensures that it never moves outside and avoids locations it has already visited. Granting such a privilege to AiRLoc also results in a performance gain which is reasonable due to the above described problem with repeating movement patterns. The same heuristic applied to the *Local sem. seg.* model seems to generate the best performance overall for this task. While this model fails to guarantee convergence in the deterministic case, the mean number of steps and the stochastic results show great promise. However, there are two main issues with this model. Firstly, not being able to guarantee deterministic convergence even in this privileged setting proves that the model is incapable of forming longer term strategies on how to cover the entire image. Secondly, even if the stochastic performance of the model is very promising, in a real world drone application one would not want to rely on stochastic evaluation.

8.6 Agent Behavior

Our investigations indicate that there may be two stages of AiRLoc's behaviour. The first phase could be mainly an exploration process where the agent moves in a semi-fixed pattern, potentially mostly using positional encoding, to gather more information about the contents of the image. This learned initial strategy seems to involve selecting a few actions and using these to move in straight lines across the image. The exact actions chosen differ in each training session, but often involves two diagonal actions and one orthogonal. When the agent has collected enough information and deems it is close to the goal patch, it deviates from the initial strategy and starts searching in the local area for the target.

In Figure 15, a small glimpse into the decision making of AiRLoc can be seen, and these results align with the general behavior description in the previous paragraph. The data for this figure is collected from trajectories where the agent starts in the bottom left corner and the goal location is either along the bottom or the left border of the image, at increasing distance. The probabilities for taking any action shows that the agent is generally better at determining the correct action when the start and goal patches are adjacent. As this data is from the very first step of trajectories, AiRLoc has not been able to gather information about the search area. The predictions by the agent, to move in the various directions, is therefore based only on the analysis of the initial input. These results suggests that AiRLoc is able to deviate from a exploratory behavior when the goal patch is near, even at the first step of the trajectory.

Analyzing Figure 15 further, it is clear that when the distance from start to goal increases to more than one step away, AiRLoc is unable to determine the direction of the goal patch. This is also reasonable, since as soon as patches are not adjacent it is very hard to determine their relative location, even for humans. Thus, AiRLoc seems to instead fall back to a learned exploration behavior which usually dictates that it moves diagonally or orthogonally in the search area. Also note that the probabilities in Figure 15 are not symmetric with respect to moving up and moving right. As discussed earlier, this is likely due a coincidence during training where AiRLoc adapted an exploration strategy which favored the moves 'Up Right' and 'Right', and is therefore generally more likely to move in these directions when it is uncertain of where to go. In other training runs with different random seeds, other actions may be favored. The main takeaway from Figure 15 is that when the goal patch is adjacent, AiRLoc's perception abilities can more often determine the correct action, but when the goal patch is further away it seems to rely more on a default exploration behavior.

8.7 Human Baseline Comparison

The result of the proof-of-concept investigation into the human performance at the aerial view image-goal localization task presented in Table 2 shows that the proposed task is in general difficult. The fact that only slightly more than half of all human controlled trajectories are successful supports this assessment. In this context, it is interesting to note that AiRLoc achieves a higher success rate compared to the human operators. However, it should be noted that this investigation is limited and the results are not necessarily statistically significant. As seen in Figure 14, the performances of the human operators and AiRLoc are very similar in how they vary with increasing difficulty. As expected, the task becomes harder to solve when the distance between start and goal increases which is reflected in the performance of both humans and AiRLoc. The fact that AiRLoc is able to closely follow and mostly surpass the human performance is a further indication that the learnt behavior is intelligent.

9 Future Work

The findings of this thesis present an optimistic outlook for the application of reinforcement learning to the aerial view image-goal localization and similar task. However, we are confident that even better results are achievable in this setting, and in Section 9.1 we propose some possible improvements to our model. Moreover, for this system to be utilized in a real world setting, additional expansions of the proposed framework is needed to better reflect the non-ideal environments experienced by an actual UAV. Section 9.2 below expands upon this topic.

9.1 Improvements

The architecture and pretraining of the patch embedder has been crucial for the performance of all RL-based agents investigated during this project and we conclude that even more effort could be put into this area. New pretraining tasks such as predicting solely the distance between two input patches could potentially yield even more relevant patch embeddings for the RL agent. Furthermore, since the addition of semantic segmentation improves agent performance, a pretraining task including both prediction of the segmentation mask as well as the Doerch task might additionally improve the created embeddings.

9.2 Expansions

The proof-of-concept study performed in this master's thesis confirms the feasibility of using RL to solve the proposed aerial view image-goal localization task. However, to be useful in real world applications, AiRLoc would have to be trained and tested on a larger scale, with longer trajectories spanning larger areas. Future studies should examine the methods ability to solve this challenge in much larger setups. Additionally, in a real world scenario the model would also need the ability to determine when it has reached the goal patch.

The ability of AiRLoc to generalize well to new datasets and circumstances is promising. This indicates robustness towards potential real world applications, as the system can be expected to operated in previously unseen and variable environments. However, misalignment's (e.g. seasonal changes) between the provided goal image and the actual goal may become problematic, as it has the potential to adversely affect AiRLoc's perception. Studies into these kinds of outcomes and on the general effects of domain shifted data could lead to a more robust and useful system.

There are also real world scenarios where top view image data of the goal location might not be available. In these cases ground level images captured using cellphones could still provide important information of the target. Investigations into how this type of input data could be utilized within the system could potentially drastically improve agent performance and robustness.

10 Conclusions

In this master's thesis we have presented the novel *aerial view image-goal localization* task. This framework allows for controllable and reproducible development and evaluation of methodologies that can eventually be useful for several relevant real-world scenarios, e.g. within search-and-rescue operations. A fully trainable reinforcement learning based approach, *AiRLoc*, has also been proposed for tackling this task, in addition to several non-RL-based learnable and heuristic methods. AiRLoc can be trained without any annotations and can thus learn the localization task in an entirely self-supervised manner. Extensive experimental evaluations have been conducted and these clearly show the benefits of the proposed AiRLoc model, which surpasses the various presented baselines, both heuristic and ML-based ones. In particular, these results demonstrate the viability of using RL for training an agent that, based solely on partial observations, can effectively navigate in aerial view images.

Furthermore, AiRLoc has been shown to generalize well to a separate domain-shifted dataset, noting only a slight performance decrease despite evaluating on unseen data. The generalization of AiRLoc to significantly longer trajectories was also verified, even if some other methods performed equally well in this setting. Moreover, providing AiRLoc with the capacity to predict building segmentation masks improved performance further, although at the expense of increasing computational cost and requiring training annotations. A proof-of-concept study of human performance at our aerial view image-goal localization task has also been conducted. The results of this study indicate a higher performance for AiRLoc compared to humans, but this may not be a statistically significant difference. To summarize, while AiRLoc is not yet ready for real-world application, we have undeniably demonstrated that there is promise in using RL to find goal locations in aerial imagery based solely on partial glimpses.

References

[1] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

[2] Kumar Ayush, Burak Uzkent, Kumar Tanmay, Marshall Burke, David Lobell, and Stefano Ermon. Efficient poverty mapping using deep reinforcement learning. *arXiv preprint arXiv:2006.04224*, 2020.

[3] Luca Bartolomei, Lucas Teixeira, and Margarita Chli. Perception-aware path planning for uavs using semantic segmentation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5808–5815. IEEE, 2020.

[4] Gabriele Berton, Carlo Masone, and Barbara Caputo. Rethinking visual geo-localization for large-scale applications. *arXiv preprint arXiv:2204.02287*, 2022.

[5] Gabriele Berton, Riccardo Mereu, Gabriele Trivigno, Carlo Masone, Gabriela Csurka, Torsten Sattler, and Barbara Caputo. Deep visual geo-localization benchmark. *arXiv preprint arXiv:2204.03444*, 2022.

[6] Adrian Boguszewski, Dominik Batorski, Natalia Ziemba-Jankowska, Tomasz Dziedzic, and Anna Zambrzycka. Landcover.ai: Dataset for automatic mapping of buildings, woodlands, water and roads from aerial imagery, 2020.

[7] Juan C Caicedo and Svetlana Lazebnik. Active object localization with deep reinforcement learning. In *Proceedings of the IEEE international conference on computer vision*, pages 2488–2496, 2015.

[8] Tung Dang, Christos Papachristos, and Kostas Alexis. Autonomous exploration and simultaneous object search using aerial robots. In *2018 IEEE Aerospace Conference*, pages 1–7. IEEE, 2018.

[9] Carl Doersch, Abhinav Gupta, and Alexei A. Efros. Unsupervised visual representation learning by context prediction, 2015.

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020.

[11] Lena M Downes, Dong-Ki Kim, Ted J Steiner, and Jonathan P How. City-wide street-to-satellite image geolocalization of a mobile ground agent. *arXiv preprint arXiv:2203.05612*, 2022.

[12] Mingfei Gao, Ruichi Yu, Ang Li, Vlad I Morariu, and Larry S Davis. Dynamic zoom-in network for fast object detection in large images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6926–6935, 2018.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[15] Danil Kuzin, Olga Isupova, Brooke D Simmons, and Steven Reece. Disaster mapping from satellites: damage detection with crowdsourced point labels. *arXiv preprint arXiv:2111.03693*, 2021.

[16] Ajith Anil Meera, Marija Popović, Alexander Millane, and Roland Siegwart. Obstacle-aware adaptive informative path planning for uav-based target search. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 718–724. IEEE, 2019.

[17] Lina Mezghani, Sainbayar Sukhbaatar, Thibaut Lavril, Oleksandr Maksymets, Dhruv Batra, Piotr Bojanowski, and Karteek Alahari. Memory-Augmented Reinforcement Learning for Image-Goal Navigation. working paper or preprint, March 2022.

[18] Volodymyr Mnih. *Machine Learning for Aerial Image Labeling*. PhD thesis, University of Toronto, 2013.

[19] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. *Advances in neural information processing systems*, 27, 2014.

[20] Simone Parisi, Aravind Rajeswaran, Senthil Purushwalkam, and Abhinav Gupta. The unsurprising effectiveness of pre-trained vision models for control. *arXiv preprint arXiv:2203.03580*, 2022.

[21] J. Peters. Policy gradient methods. *Scholarpedia*, 5(11):3698, 2010. revision #137199.

[22] Aleksis Pirinen and Cristian Sminchisescu. Deep reinforcement learning of region proposal networks for object detection. In *proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6945–6954, 2018.

[23] Marija Popović, Teresa Vidal-Calleja, Gregory Hitz, Jen Jen Chung, Inkyu Sa, Roland Siegwart, and Juan Nieto. An informative path planning framework for uav-based terrain monitoring. *Autonomous Robots*, 44(6):889–911, 2020.

[24] Shraman Pramanick, Ewa M Nowara, Joshua Gleason, Carlos D Castillo, and Rama Chellappa. Where in the world is this image? transformer-based geo-localization in the wild. *arXiv preprint arXiv:2204.13861*, 2022.

[25] Samrudhdhi B Rangrej and James J Clark. A probabilistic hard attention model for sequentially observed scenes. *arXiv preprint arXiv:2111.07534*, 2021.

[26] Samrudhdhi B Rangrej, Chetan L Srinidhi, and James J Clark. Consistency driven sequential transformers attention model for partially observable scenes. *arXiv preprint arXiv:2204.00656*, 2022.

[27] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[28] Seyed Abbas Sadat, Jens Wawerla, and Richard Vaughan. Fractal trajectories for online non-uniform aerial coverage. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 2971–2976. IEEE, 2015.

[29] Alexander Sax, Bradley Emi, Amir R Zamir, Leonidas Guibas, Silvio Savarese, and Jitendra Malik. Mid-level visual representations improve generalization and sample efficiency for learning visuomotor policies. *arXiv preprint arXiv:1812.11971*, 2018.

[30] Yujiao Shi and Hongdong Li. Beyond cross-view image retrieval: Highly accurate vehicle localization using satellite image. *arXiv preprint arXiv:2204.04752*, 2022.

[31] Irwin Sobel. An isotropic 3x3 image gradient operator. *Presentation at Stanford A.I. Project 1968*, 02 2014.

[32] Felix Stache, Jonas Westheider, Federico Magistri, Cyrill Stachniss, and Marija Popović. Adaptive path planning for uavs for multi-resolution semantic segmentation. *arXiv preprint arXiv:2203.01642*, 2022.

[33] Humans In the Loop. Semantic segmentation of aerial imagery.

[34] Burak Uzkent and Stefano Ermon. Learning when and where to zoom with deep reinforcement learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12345–12354, 2020.

[35] Andrea Vallone, Frederik Warburg, Hans Hansen, Søren Hauberg, and Javier Civera. Danish airs and grounds: A dataset for aerial-to-street-level place recognition and localization. *CoRR*, abs/2202.01821, 2022.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[37] Che Wang, Xufang Luo, Keith Ross, and Dongsheng Li. Vrl3: A data-driven framework for visual deep reinforcement learning. *arXiv preprint arXiv:2202.10324*, 2022.

[38] Tingyu Wang, Zhedong Zheng, Yaoqi Sun, Tat-Seng Chua, Yi Yang, and Chenggang Yan. Multiple-environment self-adaptive network for aerial-view geo-localization. *arXiv preprint arXiv:2204.08381*, 2022.

[39] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[40] Daniel Wilson, Xiaohan Zhang, Waqas Sultani, and Safwan Wshah. Visual and object geo-localization: A comprehensive survey. *arXiv preprint arXiv:2112.15202*, 2021.

[41] Gui-Song Xia, Jian Ding, Ming Qian, Nan Xue, Jiaming Han, Xiang Bai, Michael Ying Yang, Shengyang Li, Serge Belongie, Jiebo Luo, Mihai Datcu, Marcello Pelillo, Liangpei Zhang, Qiang Zhou, Chao-Hui Yu, Kaixuan Hu, Yingjia Bu, Wenming Tan, Zhe Yang, Wei Li, Shang Liu, Jiaxuan Zhao, Tianzhi Ma, Zi-Han Gao, Lingqi Wang, Yi Zuo, Licheng Jiao, Chang Meng, Hao Wang, Jiahao Wang, Yiming Hui, Zhuojun Dong, Jie Zhang, Qianyue Bao, Zixiao Zhang, and Fang Liu. Luai challenge 2021 on learning to understand aerial images. In *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 762–768, 2021.

[42] Tete Xiao, Ilija Radosavovic, Trevor Darrell, and Jitendra Malik. Masked visual pre-training for motor control. *arXiv preprint arXiv:2203.06173*, 2022.

[43] Karmesh Yadav, Ram Ramrakhya, Arjun Majumdar, Vincent-Pierre Berges, Sachit Kuhar, Dhruv Batra, Alexei Baevski, and Oleksandr Maksymets. Offline visual representation learning for embodied navigation. *arXiv preprint arXiv:2204.13226*, 2022.

[44] Zelong Zeng, Zheng Wang, Fan Yang, and Shin'ichi Satoh. Geo-localization via ground-to-satellite cross-view image retrieval. *IEEE Transactions on Multimedia*, pages 1–1, 2022.

[45] Leyang Zhao, Li Yan, Xiao Hu, Jinbiao Yuan, and Zhenbao Liu. Efficient and high path quality autonomous exploration and trajectory planning of uav in an unknown environment. *ISPRS International Journal of Geo-Information*, 10(10):631, 2021.

[46] Minzhao Zhu, Binglei Zhao, and Tao Kong. Navigating to objects in unseen environments by distance prediction. *CoRR*, abs/2202.03735, 2022.

[47] Runzhe Zhu. Sues-200: A multi-height multi-scene cross-view image benchmark across drone and satellite, 2022.

[48] Sijie Zhu, Mubarak Shah, and Chen Chen. Transgeo: Transformer is all you need for cross-view image geo-localization. *arXiv preprint arXiv:2204.00097*, 2022.

[49] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017.

A Initial Results and Interpolation Errors

Initially, the image processing pipeline consisted of the following stages,

$$\text{Large Image} \xrightarrow{Cropping} \text{Cropped Image (500x500)} \xrightarrow{Resize} \text{Final Image (256x256)}$$

each image cropped to the correct real world scale and then resized to directly fit the NN architecture. This is standard practice in ML and tend to work well for most tasks. A general problem with resizing is that it requires some form of interpolation, downsizing removes some information, which need to be preserved best as possible and upsizing images adds information which need to be filled somehow. We used bilinear interpolation for the downsampling step, which is the default interpolation method in the PyTorch framework. Using this pre-processing pipeline we trained and evaluated several models, Table 5 show the results from the best models as well as the baselines on this data.

Table 5: *Results for the discrete setup on the test set of* Massachusetts Buildings *with the initial image processing pipeline. Note the incredible results achieved with the initial setup and the comparably much lower success rate for the corrected image pipeline.*

Agent type	Success (%)	Step ratio	Steps	Distance
AiRLoc (edge corruption)	87.0	0.72	5.0	2.3
AiRLoc	58.6	0.60	7.1	2.5
Privileged Random	41.0	0.39	8.0	1.6
Local (edge corruption)	21.1	0.85	8.4	4.3

The results presented in Table 5 seems very promising, but comparing these results with the human baseline, that reached slightly above 50 % success rate, gave a first indication that something could be wrong. Attempting to understand the problem, a through investigation into the agents actions was performed. The start location of the agent was fixed to the bottom left corner and the goal location was varied along either the bottom or left border of the image. The produced probabilities of moving in the different directions were then collected for the first step of the agent. A reasonable expectation is that the agent would be quite certain of where to move when the goal location is adjacentt to the bottom left corner. But as the distance is increasing it should be increasingly more difficult to determine the direction in which to move. However, as can be seen in Figure 16 this was not the case. Instead, the agent is almost equally certain of where to move no matter how far away the goal is and it is somehow capable of determining in which direction the goal location is. This should not be possible as the visual resemblance of patches in different corners is likely very low. This indicates that the model is able to correctly identify which patches are located on the border of the image. As this also includes the goal patch it would entail significant information leakage about the environment to the agent which could increase the success rate drastically.

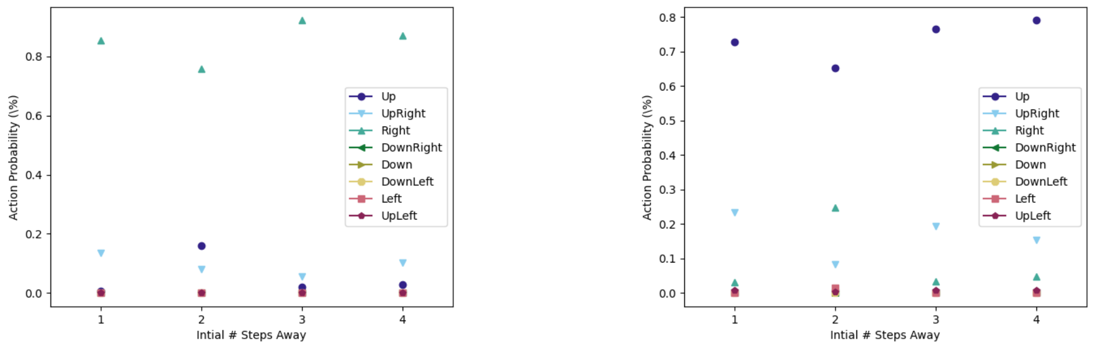


Figure 16: *The mean probabilities of taking any of the eight available actions during the first step of the trajectory. For every image in the validation set trajectories were created with the start location fixed to the lower left corner. The goal location is placed along the bottom(seen in the left image) or left (seen in the right image) border with increasing distance. As can be seen, the agent seems aware of the exact location of the goal patch despite it being very far from the agents location. Increasing distance between agent location and goal does not seem to affect the models certainty of where to move. Note, that the agent has the highest probability of moving in the correct direction.*

Careful analysis of the images, before and after resizing, reveals no edge artifacts present in the patches. Variance and absolute values of the pixel values on the edges were compared with pixels in the center of the image, with no obvious differences. Sobel's kernels[31] were used to calculate the gradients on the edge of the image, with no clear patterns differentiating the edge pixels from the center pixels. To definitively determine whether there were any identifiable edge artifacts present in the image, a new evaluation for a trained model was completed. This time the images were resized to a slightly larger pixel size than the desired final size, two pixels were added to each edge. Then the edges of the image were removed by cropping the center of the image, removing the two edge pixels which were the suspected source of corruption. This means that some information in the image are lost due to the cropping but not enough that this should have a significant impact on the performance of the model.

With this new setup the performance of the model plummeted, achieving only 41 % success which resembles the result of a random agent. After retraining the agent with this new pre-processing the performance never reached near the previously reported results. This confirms the presence of some artifact in the initial images which the network was able to identify and thereby received privileged information it was not supposed to be given by the environment.

While the above experiments confirmed the presence of image border artifacts in the previous setup, they do not specify the nature of these artifacts or why they arise. Evaluating the models with the original image pipeline using different types of interpolation (instead of bilinear) also showed similar, but varying, performance changes which lead to the suspicion that interpolation was the cause of the problems. Research into the issue confirmed that interpolation can easily cause artifacts in images but we found no evidence of *border* artifacts and their relation to artificial performance increase in ML models.

Bilinear interpolation uses a neighborhood of the four closest pixels to calculate the approximate value of the pixels in the resized image. Hence, it should only affect at most one of the edge pixels, other pixels are too far from the edge to possibly be affected by the padding. To make sure that this problem does not continue to interfere with the training of the models a new image resizing pipeline was introduced. Replicating the experimental setup of cropping the edge pixels by adding an intermediate step where the image is resized slightly larger than the final image size and then cropped to the correct dimensions. The new pipeline,

$$\text{Large Image} \xrightarrow{Cropping} \text{Cropped Image(500x500)} \xrightarrow{Resize} \text{Border Image(260x260)} \xrightarrow{Center\,Crop} \text{Final Image(256x256)},$$

seem to have removed the edge artifacts and allowed us to refocus on the task of creating a well functioning RL model.

B Additional Results

In this appendix we present a seed sensitivity analysis of our AiRLoc agent (Table 6), success rate versus episode difficulty for models equipped with a separately trained semantic segmentation network (Figure 17), as well as additional results when training AiRLoc with the *complex* reward structure (Table 7), cf. (16).

Table 6: *Seed sensitivity analysis of the main AiRLoc agent on the validation set of* Massachusetts Buildings*, where models are allowed to take at most 10 actions per trajectory. The result on the first line in the table corresponds to the AiRLoc agent we have evaluated in the main part of this thesis, and as can be seen its results corresponds to the median in terms of performance. We also include the non-AiRLoc results from Table 2 for easier comparison. All seeds except one yield a very similar performance, but even the worst-performing AiRLoc agent is better than the best non-RL-based ML baseline (*Privileged Local*).*

Agent type	Success (%)	Step ratio	Steps	Residual dist.
AiRLoc	58.6	0.60	7.1	2.5
AiRLoc (other seed #1)	59.0	0.60	7.2	2.5
AiRLoc (other seed #2)	59.3	0.62	7.0	2.4
AiRLoc (other seed #3)	58.6	0.62	7.1	2.4
AiRLoc (other seed #4)	53.8	0.58	7.4	2.5
Privileged Random	39.0	0.51	8.2	2.6
Local	14.4	0.83	8.9	6.4
Privileged Local	51.2	0.59	7.5	2.6
Global	8.0	-	1.0	2.9
Human	55.7	0.54	7.6	2.3

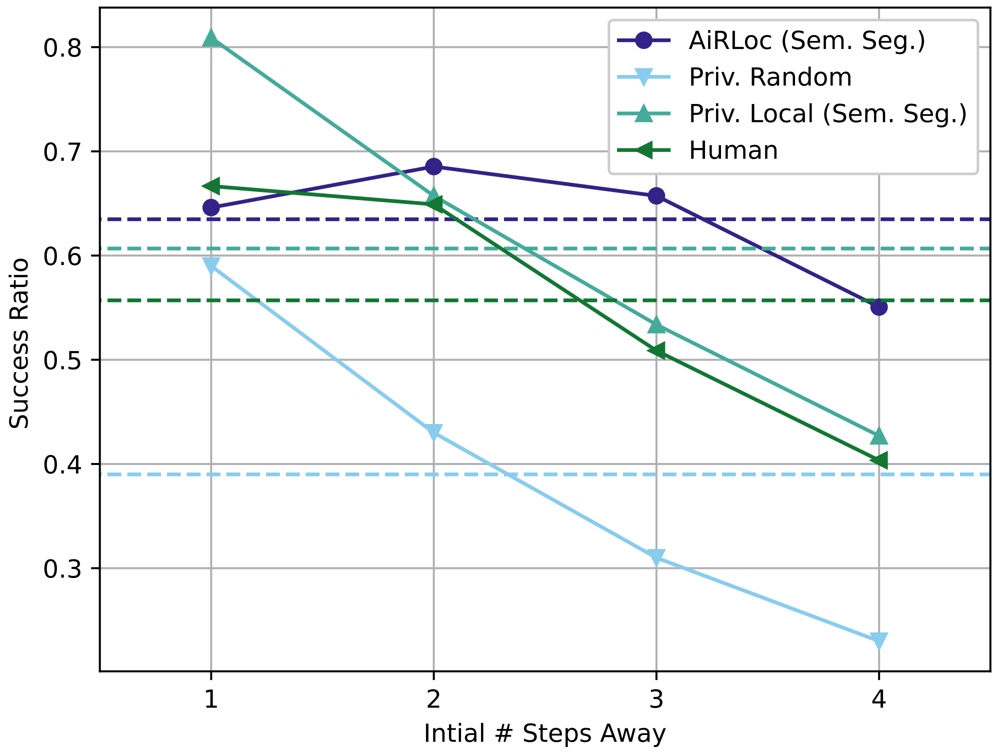


Figure 17: *The success rate of the segmentation network-equipped AiRLoc and various baseline methods during evaluation on the validation set of* Massachusetts Buildings *versus episode difficulties (see the corresponding results without semantic segmentation in Figure 14). Dashed lines (in matching colors) correspond to global averages over all difficulties. When AiRLoc is equipped with mid-level vision capabilities, its performance gap over the heuristic random and human baselines increases on average. The non-RL-based ML approach* Privileged Local *also becomes superior to humans when it is equipped with a semantic segmentation network (AiRLoc is however even better on average). Interestingly, AiRLoc with semantic segmentation is the model which is the least sensitive to episode difficulty, as its performance curve is the flattest one. While AiRLoc with semantic segmentation is not the best at the simplest episodes – note that* Local *is trained exclusively in the setup corresponding to the simplest difficulty – it is best by a large margin on the more difficult episodes.*

Table 7: *Results for the discrete setup on the validation set of* Massachusetts Buildings *trained with the* complex *reward. The below presented data represents a full ablation study of the proposed model trained with this additional reward. However, due to its lower performance compared with the AiRLoc model trained with the simpler reward (which is re-reported below for easier comparison), it is omitted from Section 7. We also include the non-AiRLoc results from Table 2 below for easier comparison.*

Agent type	Success (%)	Step ratio	Steps	Residual dist.
AiRLoc	53.5	0.60	7.5	2.3
AiRLoc (sem. seg. GT)	61.0	0.61	7.1	2.3
AiRLoc (sem. seg. pred.)	59.8	0.62	7.2	2.4
AiRLoc (frozen emb.)	50.7	0.63	7.5	2.5
AiRLoc (from scratch)	45.1	0.69	6.9	2.5
AiRLoc (no LSTM)	44.2	0.59	7.8	2.4
AiRLoc (no RGB)	40.7	0.61	8.2	2.7
AiRLoc (no pos. enc.)	25.8	0.58	8.7	3.0
AiRLoc (simpler reward)	58.6	0.60	7.1	2.5
Privileged Random	39.0	0.51	8.2	2.6
Local	14.4	0.83	8.9	6.4
Privileged Local	51.2	0.59	7.5	2.6
Global	8.0	-	1.0	2.9
Human	55.7	0.54	7.6	2.3

In addition to the ablation studies of AiRLoc trained with the complex reward, two hand-picked trajectories from the evaluation of the model on the validation set of *Massachusetts Buildings* are shown in Figure 18.

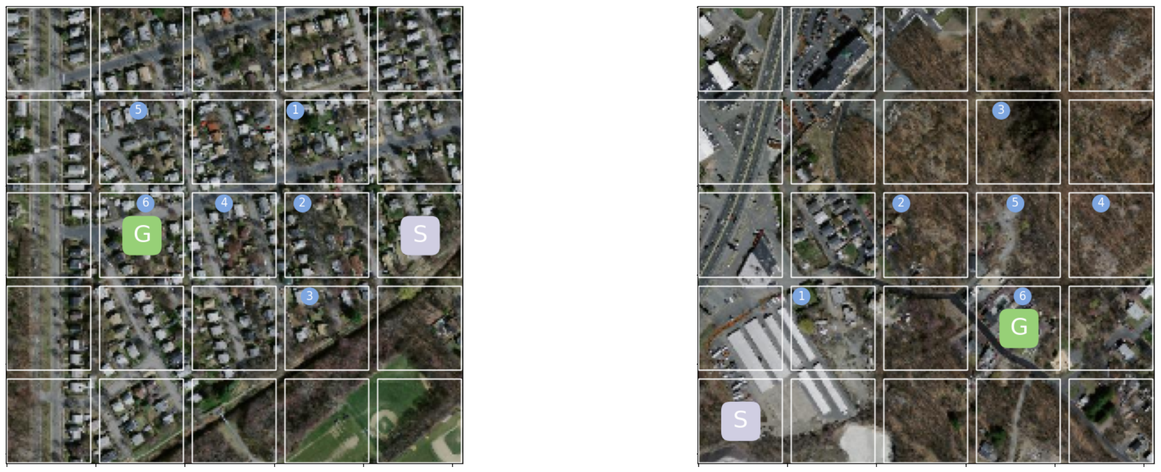


Figure 18: *Two example trajectories of AiRLoc trained with the* complex *reward. Note in particular the pattern of navigating diagonal and straight in an alternating fashion.*

C Trajectory Visualizations

In this appendix (Figure 19 - 23), trajectory visualizations of AiRLoc are displayed. Recall that AiRLoc never obtains the underlying search area in its entirety – all action selection is performed based on the sequentially observed partial glimpses and the goal patch. The model is trained on the training set of *Massachusetts Buildings*, and the trajectories shown are generated by evaluating the model on the validation partition of the same dataset. During evaluation, AiRLoc is run in deterministic mode.

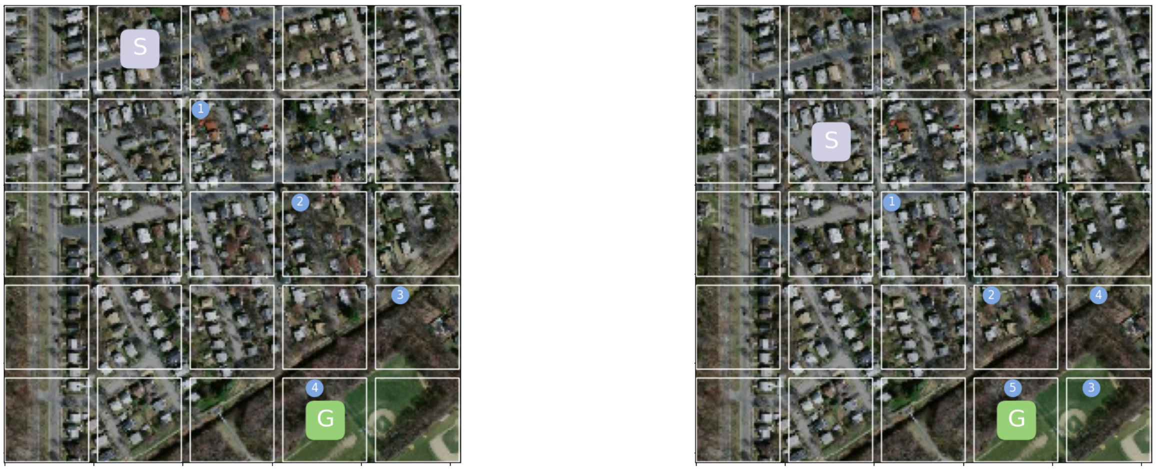


Figure 19: *Two examples of successful AiRLoc trajectories. Note that the two setups are identical except that the start position is shifted one position between the two. The initial behavior is the same in both cases, where a diagonal exploration towards the bottom-right is pursued.*

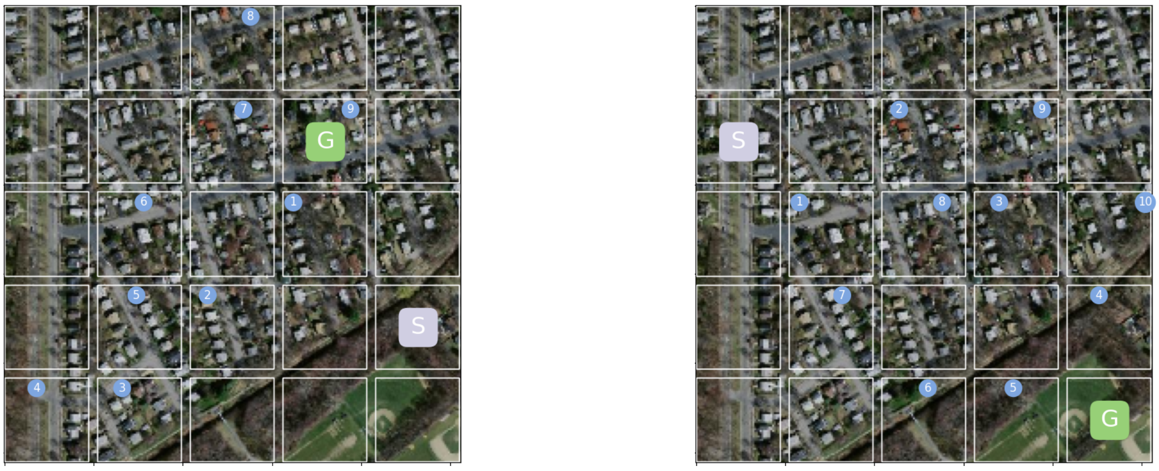


Figure 20: *Two examples of AiRLoc trajectories. Note the diagonal exploration strategy utilized successfully to the left and unsuccessfully to the right.*

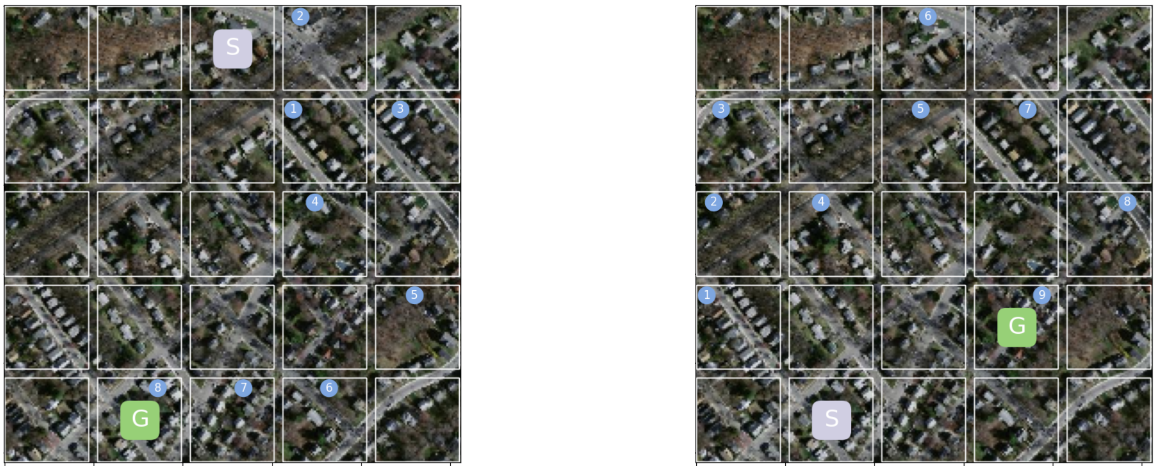


Figure 21: *Two examples of successful AiRLoc trajectories.*

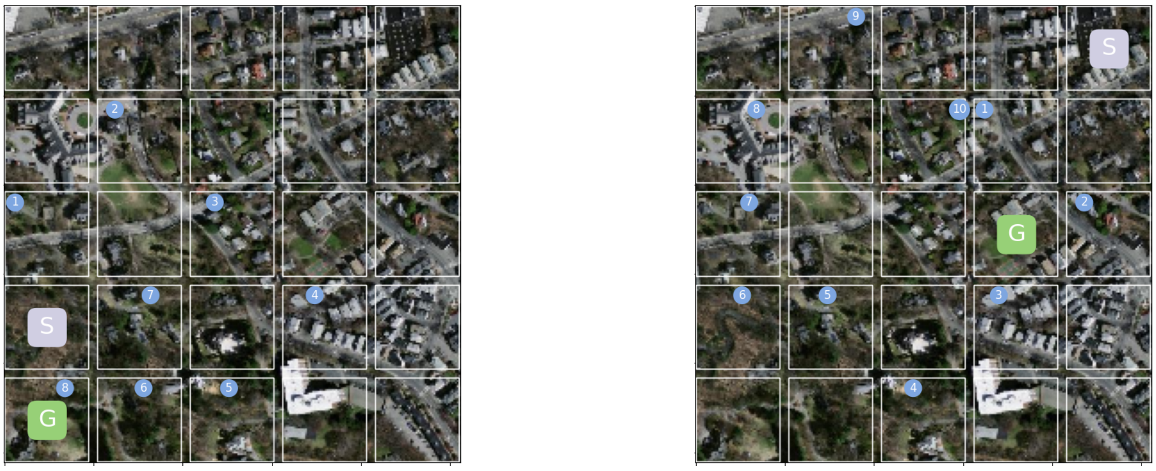


Figure 22: *Two examples of AiRLoc trajectories. In the left trajectory, it seems like the agent uses the exploration strategy with only diagonal moves until reaching the patch numbered "5", after which it focuses on straight moves until finding the goal. The right trajectory is an unsuccessful example.*

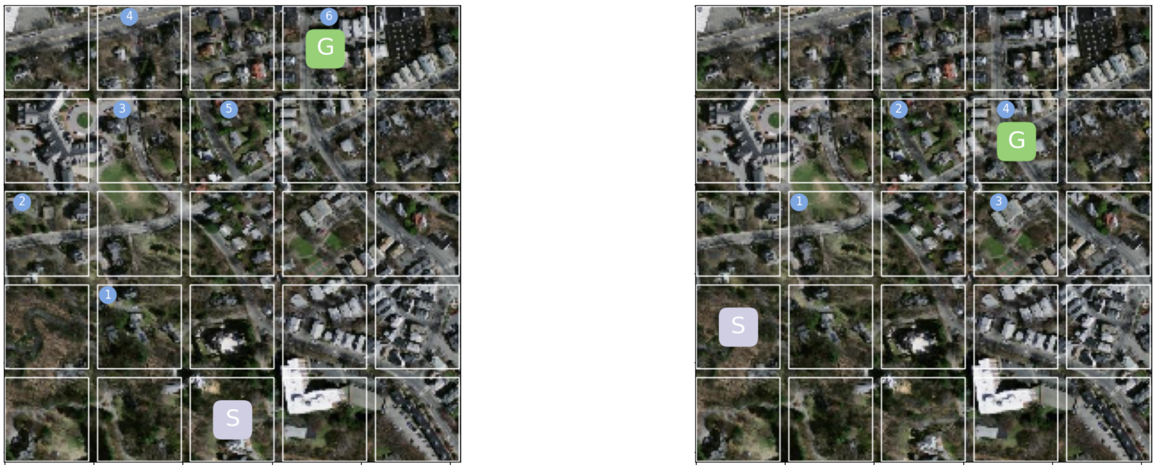


Figure 23: *Two examples of successful AiRLoc trajectories.*

Master's Theses in Mathematical Sciences 2022:E30
ISSN 1404-6342

LUTFMA-3479-2022

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/