

Autonomous UAV Path Planning using RSS-signals in Search and Rescue Operations

Axel Anhammer and Hugo Lundeberg

Master of Science Thesis in Electrical Engineering
**Autonomous UAV Path Planning using RSS-signals in Search and Rescue
Operations**

Axel Anhammer and Hugo Lundeberg
LiTH-ISY-EX-22/5497-SE

Supervisor: **Carl Hynén Ulfsjö**
ISY, Linköpings universitet
Emma Jonsson
Combitech AB
Michael Petterstedt
Combitech AB

Examiner: **Fredrik Gustafsson**
ISY, Linköpings universitet

*Division of Automatic Control
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden*

Copyright © 2022 Axel Anhammer and Hugo Lundeberg

Abstract

Unmanned aerial vehicles (UAVs) have emerged as a promising technology in search and rescue operations (SAR). UAVs have the ability to provide more timely localization, thus decreasing the crucial duration of SAR operations. Previous work have demonstrated proof-of-concept in regard to localizing missing people by utilizing received signal strength (RSS) and UAVs. The localization system is based on the assumption that the missing person wears an enabled smartphone whose Wi-Fi signal can be intercepted.

This thesis proposes a two-staged path planner for UAVs, utilizing RSS-signals and an initial belief regarding the missing person's location. The objective of the first stage is to locate an RSS-signal. By dividing the search area into grids, a hierarchical solution based on several Markov decision processes (MDPs) can be formulated which take different areas probabilities into consideration. The objective of the second stage is to isolate the RSS-signal and provide a location estimate. The environment is deemed to be partially observable, and the problem is formulated as a partially observable Markov decision process (POMDP). Two different filters, a point mass filter (PMF) and a particle filter (PF), are evaluated in regard to their ability to correctly estimate the state of the environment. The state of the environment then acts as input to a deep Q-network (DQN) which selects appropriate actions for the UAV. Thus, the DQN becomes a path planner for the UAV and the trajectory it generates is compared to trajectories generated by, among others, a greedy-policy.

Results for Stage 1 demonstrate that the path generated by the MDPs prioritizes areas with higher probability, and intuitively seems very reasonable. The results also illustrate potential drawbacks with a hierarchical solution, which potentially can be addressed by considering more factors into the problem. Simulation results for Stage 2 show that both a PMF and a PF can successfully be used to estimate the state of the environment and provide an accurate localization estimate. The PMF generated slightly more accurate estimations compared to the PF. The DQN is successful in isolating the missing person's probable location, by relatively few actions. However, it only performs marginally better than the greedy policy, indicating that it may be a complicated solution to a simpler problem.

Acknowledgments

We would like to take this opportunity to express gratitude towards all the people who have donated their time and energy to helping us through the execution of this thesis. First and foremost, we are grateful to our supervisor from Linköping University, Carl Hynén Ulfsjö. Thank you for countless hours of questions, resulting in many insightful discussions and much valuable input.

This thesis would not have been possible without all of the supportive people at Combitech AB. We would like to thank our supervisors, Michael Petterstedt and Emma Jonsson, for always being helpful and warmly welcoming us into the community. A special thanks to Jesper Tordenlid for establishing the initial connection between us and the company and iterating the thesis idea. Additionally, we would like to thank Daniel Femerström and all of the thesis students at Combitech for creating a fun and enjoyable workplace.

We would like to extend our thanks to our examiner at Linköping University, Fredrik Gustafsson, who helped with the initial thesis idea and provided valuable input.

Lastly, we would like to thank our family and friends for their support and patience throughout this thesis.

Linköping, May 2022
Axel Anhammer & Hugo Lundeberg

Contents

Notation	v
1 Introduction	1
1.1 Aim	2
1.2 Problem formulation	2
1.3 Assumptions and delimitations	3
1.4 Swedish Police SAR methods	4
1.5 Related work	5
2 Theory	7
2.1 Sensor models	7
2.1.1 Received signal strength	7
2.2 Filter theory	8
2.2.1 Models	8
2.2.2 Bayesian filtering	9
2.2.3 Point mass filter	9
2.2.4 Particle filter	11
2.3 Positioning techniques	12
2.3.1 Trilateration	12
2.3.2 Gauss-Newton	13
2.4 Planning algorithm frameworks	14
2.4.1 Markov decision process	14
2.4.2 Partially observable Markov decision process	16
2.5 Reinforcement learning	18
2.5.1 Value iteration	19
2.5.2 Q-Learning	19
2.5.3 Deep Q network	20
3 Method	24
3.1 RSS-signal	25
3.2 SAR environment	26
3.3 Stage 1 - signal localization	28
3.3.1 Formulation of MDP	28

3.3.2	Hierarchical solution	28
3.4	Stage 2 - signal isolation	29
3.4.1	Formulation of POMDP	29
3.4.2	Simulation environment	35
3.4.3	DQN	36
3.5	Evaluation methods	38
3.5.1	Path planning evaluation	38
3.5.2	Localization evaluation	39
4	Results and evaluation	41
4.1	Hierarchical MDPs	41
4.2	DQN training	45
4.3	State estimator comparison	46
4.4	Path planning	49
4.5	Localization	51
4.6	DQN-PMF simulation	53
5	Conclusions and future work	54
5.1	Conclusions	54
5.2	Future work	55
A	Particle filter resampling	59
Bibliography		61
Bibliography		61

Notation

ACRONYMS

Acronym	Meaning
MM	Mattson Method
ROW	Rest of World
POA	Probability of Area
POD	Probability of Detection
POS	Probability of Success
IPM	Initial Probabilistic Map
SAR	Search and Rescue
UAV	Unmanned Aerial Vehicle
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
RSS	Received Signal Strength
GPS	Global Positioning System
PF	Particle Filter
PMF	Point Mass Filter
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
DQN	Deep Q-network
NN	Neural Network
CNN	Convolution Neural Network

1

Introduction

Every year, the Swedish police conduct over 300 rescue operations to search for people who have disappeared under such circumstances that their health and life could be in serious danger [1]. These search and rescue (SAR) operations often involve large search areas. Depending on factors such as the missing person's health, the terrain, and the weather, the time it takes to complete a successful SAR operation is crucial. Unmanned Aerial Vehicles (UAVs), commonly known as drones, have emerged as a promising technology that can be utilized in SAR operations [2]. The Swedish Police have successfully used UAVs in SAR operations since they offer the ability to quickly and efficiently search through large open areas, inaccessible terrain and watercourses. Additionally, they can be deployed when there is a low cloud base or poor visibility, factors that prevent the use of helicopters [3]. Thus, the use of UAVs have the potential to provide more timely localization and thereby minimizing the crucial duration of SAR operations.

Multiple alternative localization techniques, to global positioning system (GPS) have been studied in literature since not all communicating devices are equipped with GPS [4]. This is due to factors such as expensive cost, poor performance in certain weather conditions and vulnerability to jamming [5]. Additionally, a base station to collect an object's location using GPS may be unavailable in disaster situations. Among the alternatives, radio received signal strength (RSS) is attractive due to its cheap functionality and simplicity. However, RSS-based localization is considered very inaccurate [6]. The usage of UAVs offers the ability to measure RSS from different angles with a higher probability of line-of-sight, thus achieving improved localization accuracy [7].

Previous work has shown that it is possible to build a localization system based on RSS, using widely available commercial-off-the-shelf (COTS) products [8]. The utilization of such a system could benefit SAR operations by reducing the search area and providing position estimates of the missing person. The lo-

calization system is based on the following two assumptions:

- The missing person wears an enabled smartphone.
- The Wi-Fi signal transmitted by the smartphone can be intercepted by mobile agents at known positions.

In addition, the localization system was shown to be successful when using remote controlled UAVs, providing a localization error of roughly 15 meters [8]. By combining a localization system based on RSS with a path planner for UAVs, several potential advantages become available. These include faster position estimates and reduced operation time. Due to the autonomy of such a system, it could be deployed quickly and would require few resources to operate.

1.1 Aim

This thesis aims to investigate if a novel planning algorithm can enable more efficient use of UAVs in SAR missions, allowing faster and more accurate localization of missing people. The planning algorithm is intended to work as an extension to the current SAR methods used by the Swedish police.

The algorithm's ability to make intelligent decisions largely depends on it having a correct understanding of the environment it acts in. The Swedish police utilizes a self developed method to generate a probabilistic map of the missing person's location. RSS-measurements from UAV on-board sensors will be used to recursively update this probabilistic map, which will act as input to the planning algorithm, allowing an efficient decision-making process that maximizes the probability of finding a missing person and minimizes the duration of the mission.

The main task of this master thesis is to implement the following two complementary tasks:

- Planning algorithm for UAVs based on a probabilistic map of the missing person's location.
- Recursive updating of the probabilistic map based on RSS-observations.

1.2 Problem formulation

Based on the previously stated goals, this thesis will mainly center around two categories. These two categories are *Search problem* and *Exploration and map-building*. RSS-measurements will be used to gather information and recursively update a probabilistic map, which will be used to conduct a search for a missing person of an unknown location. This leads to the following questions:

- How can the initial probabilistic map be utilized to efficiently locate an RSS-signal?

- How can RSS-measurements effectively be used to establish an UAV trajectory that isolates the missing person's probable location?

The questions identified above are independent but complementary, meaning that in order to reach the best possible result, both need to be answered in consideration of each other. Since the location of the missing person is unknown, the environment can be defined as *partially observable*. Assumptions made in Section 1.3 are used to determine the remaining properties of the environment.

1.3 Assumptions and delimitations

Finite state-space

The search area will be divided into a grid with a finite amount of cells. The missing person is assumed to be located within one of these cells. Thus, there are a limited amount of configurations where the agents and the target can be located in. Therefore, the problem will have a *finite* state-space.

Stochastic

The system is assumed to be *stochastic*, which means that the system will model the uncertainties of the problem in sensor information and action effects. Interference with RSS-measurements will be modelled as noise.

Missing person

The following three assumptions are made regarding the missing person:

- The missing person wears an enabled smartphone.
- The Wi-Fi signal transmitted by the smartphone can be intercepted by mobile agents at known positions.

The environment is assumed to be *static*, i.e., changes will only happen as a consequence of the agent making an action. This implies that the missing person's location is the same during an entire mission.

Terrain

Terrain such as trees, mountains, rivers or other types of obstacles is not taken into consideration in this thesis. The simulation environment will consist of a two-dimensional map of longitudinal and latitudinal coordinates. This delimitation will not affect the direct evaluation of this thesis, but it could affect the real world usability. Terrain can both disturb signals detected by the drone's sensors and interfere with the drone's path.

1.4 Swedish Police SAR methods

An essential factor when searching for missing people is the search area. When the search area has been defined, it is divided into different sectors. These usually have an area of 1 to 1.5 square kilometers, and their borders can often be found in the terrain (i.e., roads, paths, power lines and watercourses) [9]. When a search area has been created and divided into sectors, the question remains, which of these sectors should be prioritized? To solve this, the police use a self developed method called the Mattson-method which will be explained in more detail in the coming section.

The Mattson-method

The Swedish police use a method called *The modified Mattson Consensus*, henceforth referred to as the Mattson-method (MM), to create a structured assessment of which sectors should be prioritized. First, an assessment group is informed about the details regarding the disappearance and the missing person. After that, each member of the assessment group individually rates the sectors and the area outside the search area (ROW, rest of the world), based on the perception of where it is most likely that the missing person is located. Every sector is given a rating in the range of 1-9, where a higher rating indicates that there is an increased probability that the missing person is located within the sector. These ratings are based on the individual's previous experience in SAR operations and sometimes local terrain knowledge [9].

The results from all the individuals are then compiled, which generates the probability distribution regarding the missing person's location within the search area (POA, probability of area). Experience from actual events and educational situations has shown that the missing person is found in one of the three to four most prioritized sectors, in a majority of cases. The Mattson-method introduces multiple concepts which are essential for the search process:

- **Probability of Area (POA):** The probability that a missing person is located in a sector. If a perfect search area has been created, then it is guaranteed that the missing person is located inside the search area. The total POA-values for all sectors in the search area will thus be 100%, based on the assumption that the POA for the ROW equals zero.
- **Probability of Detection (POD):** This probability represents a value for the accuracy with which a search resource has conducted a search of a sector. If a sector has been searched thoroughly, the POD is 100%. POD is thus a measurement of how carefully a sector has been searched.
- **Probability of Success (POS):** This probability is used to calculate the value of the performed search job. The value is obtained through the POA- and POD-value according to the equation $POS = POA \cdot POD$.
- **POScum:** The cumulative value of the completed searches is defined as POScum. POScum can therefore be seen as a measurement of the search that has been completed.

1.5 Related work

There are multiple works in the literature that investigate localization based on RSS-measurements. These can be divided by the use of terrestrial or mobile anchors. Localization using terrestrial anchors is studied in [6, 10]. In [6], the authors provide an analysis of the main factors that affect the variability of the received signal power and the accuracy of the RSS-measurements and suggest possible techniques to alleviate such problems. In [10], the authors investigate and evaluate how the accuracy of the RSS-measurements are affected by changing the height and distance between the anchors (signal receiver) and the terrestrial objects (signal emitter).

Localization with mobile anchors naturally involves path planning and is studied in [11, 12]. The authors of [11] studied three different pre-determined trajectories for mobile anchors to traverse an area and demonstrated that any deterministic trajectory offers significant benefits compared to random movement. In [12], the authors proposed a novel trajectory that localized all nodes with high precision using short time.

Localization with both RSS-measurements and mobile anchors is investigated in [8] where the authors present a feasibility study on smartphone localization of missing persons in SAR operations using widely available COTS products. A proof-of-concept is presented, which consists of several mobile agents carrying smartphones that measure the RSS of Wi-Fi messages transmitted by the smartphone of the missing person. In addition, several successful tests were conducted with a Quadcopter to show the feasibility of using UAVs in SAR operations.

Path planning for UAVs is studied thoroughly in literature, but differs depending on the task at hand. UAV path planning with the purpose of localizing terrestrial objects through the use of RSS-measurements is addressed in [13, 14]. The authors of [13] propose a novel framework, based on RSS-measurements and reinforcement learning (RL), that enables autonomous planning for UAVs. The agent follows an initial scan trajectory of the region to know the number of nodes, estimate their location and train the agent online during the operation. Then, the agent forms its trajectory by using an RL trained model to choose the next waypoints. It results in an improved localization accuracy of multiple objects in a shorter time and path length compared to state-of-the-art pre-defined trajectory methods. Similarly, [14] formulates an UAV trajectory based on RSS-measurements by reformulating the problem into two complementary sub-problems for a disaster scene with multiple regions or cells with varying levels of importance. The first sub-problem identifies a minimal number of strategic positions, which act as input to the second sub-problem that constructs an efficient UAV trajectory that traverses all waypoints. Simulation results demonstrate the accuracy and effectiveness of the proposed approach in localizing an unknown number of mobile devices in disaster scenes with regions of varying importance levels.

Different from above, [15] propose multiple path planning algorithms based on the traveling salesman problem (TSP) for a UAV that traverses through a sequence of waypoints. The results show that the algorithms are able to securely

localize all the positions in a generic deployment area, even in the presence of drone control errors. The algorithms produce short path lengths within a reasonable processing time.

The use of deep reinforcement learning (DRL) for path planning is studied in [16] where the path planning problem for multiple UAVs (agents) is translated into a decentralized partially observable Markov decision process (Dec-POMDP), which is solved through a DRL approach. A combination of global and local map representations are exploited and fed into a convolutional neural network (CNN). The network architecture enables the agents to cooperate effectively by dividing tasks among themselves and make movement decisions based on goals.

The author of [17] extensively investigate different algorithms for exact and approximate solution to POMDPs. Highlighting reinforcement learning techniques to have great potential for approximate solving of large POMDPs. Many years later, the authors of [18] presents a novel DRL model called Deep Q-Network (DQN) which is capable of learning human level control policies on different Atari 2600 games using high-dimensional neural networks. The model use raw pixels of the Atari games as input and gave state-of-the-art results in six out of seven games tested on. To handle the fact that the games are partially observable, meaning that a single frame is not enough to determine the state of the game, four consecutive frames were stacked in the state representation, which allowed the authors to model the problem as a standard MPD.

The authors of [19] introduce a drone surveillance problem modeled as a POMDP. The drone's mission is to survey two specific regions, in a grid of equally sized square regions, while avoiding flying over a ground agent. The drone has a limited field of view and moves deterministically, while the agent moves probabilistically in the grid. The problem shares certain similarities to our problem formulation.

In [20], the authors use particle filters to map trajectory history into belief states for agents, whose position can not be directly observed, in a POMDP environment. Consequently, off-policy reinforcement learning is used to map actions from belief states. The results show that multi-layer resampling methods improved belief state accuracy with respect to ground truth for scenarios in which sensor fusion may be impracticable.

2

Theory

This chapter introduces the theoretical foundation needed to solve the problem described in Chapter 1. Section 2.1 provides the reader with the basic knowledge about RSS-signals, while Section 2.2 and 2.3 are used to explain filters and positioning techniques, which will later be used to process the RSS-signal. Lastly, Sectors 2.4 and 2.5 cover several planning algorithm frameworks as well as reinforcement learning theory.

2.1 Sensor models

Localization in sensor networks can be described as the application of *sensor fusion* to distance, distance differences and angle measurements. Sensor fusion can be defined as the combining of sensory data or data derived from sensory data from disparate sources such that the resulting information is in some sense better than what would be possible when these sources were used individually [21]. The basic sensor model for sensor networks applications is:

$$y = h(x, p) + e \quad (2.1)$$

Here, $x = (x_1, x_2)^T$ denotes the (two-dimensional) position of a target and $p = (p_1^T, p_2^T, \dots, p_n^T)$ contains the locations $p_k = (p_{k,1}, p_{k,2})^T$ of each sensor. The distance (r_i) between the target (x) and each sensor (p_i) can be calculated through:

$$r_i = \|x - p_i\| \quad (2.2)$$

2.1.1 Received signal strength

One possibility is that the sensor estimates the received signal power, or received signal strength (RSS). Essentially, this means integrating the received signal power

within a certain frequency band during an integration interval to estimate the received signal energy during the time interval. RSS provides coarse range information if the emitted power is known. In the unknown case, two or more sensors can compare their RSS observations to eliminate the unknown emitted power [21]. If the set of available measurements is summarized as $y = h(x, p) + e$, the Okumura-Hata model can be used to estimate x for RSS measurements:

$$h(x, p_i) = P_0 + 10\beta \log_{10} r_i \quad (2.3)$$

The parameter P_0 denotes the received power at reference distance d_0 and β is the path loss exponent. The path loss exponent depends on the environment. In free space, the path loss increases by 20 dB when the distance between the transmitter and the receiver increases ten times. This implies that the path loss exponent has a value of 2 in free space. RSS is thus a measurement of the signal strength at the location of the receiver. It is usually taken from the received signal strength indicator (RSSI) of the device. There is no exact definition of how to achieve this measurement. Different chip manufacturers use different techniques and parameters to calculate the RSSI result. Therefore, the results can vary by chipsets of the same manufacturer [22].

2.2 Filter theory

In signal processing, filtering is a process that removes unwanted components or features, for example noise, from a signal. Filtering can be used to estimate a hidden state from a potentially incomplete and noisy set of observations. A *dynamic system* can be defined by a state space model with a hidden state vector, from which partial information is obtained by observations. The *nonlinear filtering* problem is to make inference on the hidden state, which can be done by computing or approximating the posterior distribution for the state vector given all available observations [21].

2.2.1 Models

In application, nonlinear filtering is based on discrete time nonlinear state space models which relates a hidden state x_k to the observations y_k :

$$x_{k+1} = f(x_k, v_k), \quad v_k \sim p_{vk}, \quad x_0 \sim p_{x0} \quad (2.4a)$$

$$y_k = h(x_k) + e_k, \quad e_k \sim p_{ek} \quad (2.4b)$$

The stochastic noise process, v_k , is specified by its known probability density function (PDF) p_{vk} . The additive measurement noise, e_k , is expressed similarly with known PDF p_{ek} while p_{x0} denotes the PDF of the initial state x_0 [21]. In statistical literature, a general Markov model and observation model are often used, in terms of conditional PDFs:

$$x_{k+1} \sim p(x_{k+1} | x_k) \quad (2.5a)$$

$$y_k \sim p(y_k | x_k) \quad (2.5b)$$

2.2.2 Bayesian filtering

The Bayesian solution, to the nonlinear filtering problem, is to compute the posterior distribution $p(x_k | y_{1:k})$ of the hidden state vector, given past observations [21]. The posterior distribution is given by the general Bayesian update recursion:

$$p(x_k | y_{1:k}) = \frac{p(y_k | x_k)p(x_k | y_{1:k-1})}{p(y_k | y_{1:k-1})} \quad (2.6a)$$

$$p(y_k | y_{1:k-1}) = \int_{\mathbb{R}^{n_x}} p(y_k | x_k)p(x_k | y_{1:k-1}) dx_k \quad (2.6b)$$

$$p(x_{k+1} | y_{1:k}) = \int_{\mathbb{R}^{n_x}} p(x_{k+1} | x_k)p(x_k | y_{1:k}) dx_k \quad (2.6c)$$

A measurement update is described in (2.6), which follows from Bayes' law. A normalization constant and a time update is described in (2.6b) respectively (2.6c), both follows from the law of total probability. For non-Gaussian or nonlinear models, there is in general no finite dimensional representation of the posterior distributions. These can instead be numerically approximated by point mass filters or particle filters [21].

2.2.3 Point mass filter

The point mass filter (PMF) grids the state space and computes the posterior over this grid recursively. It is able to represent any posterior distribution and applies to any nonlinear and non-Gaussian model [21]. Suppose that we have a deterministic grid $\{x^i\}_{i=1}^N$ of the state space \mathbb{R}^{n_x} over N points. At time k , based on observations $y_{1:k-1}$, we have computed the relative probabilities (assuming distinct grid points):

$$w_{k|k-1}^i \propto P(x_k = x^i | y_{1:k-1}) \quad (2.7)$$

Satisfying $\sum_{i=1}^N w_{k|k-1}^i = 1$ (relative normalization with respect to the grid points). The prediction density, the expected value and the covariance can then be approximated by:

$$\hat{p}(x_k | y_{1:k-1}) = \sum_{i=1}^N w_{k|k-1}^{(i)} \delta(x_i - x_k^{(i)}) \quad (2.8a)$$

$$\hat{x}_{k|k-1} = E(x_k) = \sum_{i=1}^N w_{k|k-1}^{(i)} x_k^{(i)} \quad (2.8b)$$

$$P_{k|k-1} = \text{Cov}(x_k) = \sum_{i=1}^N w_{k|k-1}^{(i)} (x_k^{(i)} - \hat{x}_{k|k-1})(x_k^{(i)} - \hat{x}_{k|k-1})^T \quad (2.8c)$$

The Dirac impulse function is denoted by $\delta(x)$. The Bayesian recursion (2.6a) is then used, which gives:

$$\hat{p}(x_k | y_{1:k}) = \sum_{i=1}^N \underbrace{\frac{1}{c_k} p(y_k | x_k^{(i)}) w_{k|k-1}^{(i)}}_{w_{k|k}^{(i)}} \delta(x_i - x_k^{(i)}) \quad (2.9a)$$

$$c_k = \sum_{i=1}^N p(x_k | y_{1:k}) w_{k|k-1}^{(i)} \quad (2.9b)$$

$$\hat{p}(x_{k+1} | y_{1:k}) = \sum_{i=1}^N w_{k|k}^{(i)} p(x_{k+1} | x_k^{(i)}) \quad (2.9c)$$

The normalizing constant c_k assures that $\sum_{i=1}^N w_{k|k}^{(i)} = 1$. It can be noted that the recursion starts with a discrete approximation (2.8a) and ends with a continuous distribution (2.9c) [21]. The standard approach, to close the recursion, is to sample (2.9c) at the grid points x^i , which computationally can be seen as multi-dimensional convolution:

$$w_{k+1|k}^{(i)} = \hat{p}(x_{k+1}^{(i)} | y_{1:k}) = \sum_{j=1}^N w_{k|k}^{(j)} p(x_{k+1}^{(i)} | x_k^{(j)}), \quad i = 1, 2, \dots, n \quad (2.10)$$

The PMF advantage lies in its simple implementation and tuning, basically only the size and resolution of the grid needs to be considered. The curse of dimensionality however limits the application of PMF to small models (with dimensions, n_x , less than two or three) for two reasons. The first reason is that a grid is an inefficiently sparse representation in higher dimensions, and the second reason is that the multidimensional convolution becomes a real bottleneck with quadratic complexity in n [21].

2.2.4 Particle filter

The particle filter (PF) has many similarities with the point mass filter (PMF). Both filters approximate the posterior distribution with a discrete density of the form (2.8a). Additionally, both filters are based on a direct application of (2.6) leading to the numerical resolution in (2.9). There are, however, some major differences:

- The PF uses a dynamic stochastic grid x_i^k that changes over time, instead of the deterministic grid x_i that is used in the PMF.
- The PF aims at estimating the whole trajectory $x_{1:k}$ rather than the current state x_k . This affects (2.6a) in the following way:

$$\begin{aligned}
 p(x_{1:k+1}^i | y_{1:k}) &= \underbrace{p(x_{k+1}^i | x_{1:k}^i, y_{1:k})}_{p(x_{k+1}^i | x_k)} \underbrace{p(x_{1:k}^i | y_{1:k})}_{w_{k|k}^i} \\
 &= w_{k|k}^i p(x_{1:k}^i | y_{1:k})
 \end{aligned} \tag{2.11}$$

- The PF obtains its new grid by resampling from (2.9c), instead of reusing the old grid as is done in the PMF.
- The PF includes a crucial *resampling* step. Without the resampling step, the PF would break down to a set of independent simulations yielding trajectories $x_{1:k}^i$ with relative probabilities w_k^i . The lack of feedback mechanism from the observations to control the simulations means that they would soon diverge. As a result, all weight would tend to zero except for one that tends to one [21].

There exists multiple different resampling techniques which includes multinomial, stratified, systematic-and residual resampling. These methods are described in more detail in Appendix A. The PF algorithm is described below in Algorithm 1 [21].

Algorithm 1 Particle Filter

Choose a proposal distribution $q(x_{k+1} | x_{1:k}, y_{k+1})$, resampling strategy and the number of particles N .

Initialization: Generate $x_1^i \sim p_{x_0}$, $i = 1, \dots, N$ and let $w_{1|0}^i = 1/N$.

for $k = 1, 2, \dots$ **do**

1. *Measurement update:* For $i = 1, 2, \dots, N$

$$w_{k|k}^i = \frac{1}{c_k} w_{k|k-1}^i p(y_k | x_k^i)$$

$$c_k = \sum_{i=1}^N w_{k|k-1}^i p(y_k | x_k^i)$$

2. *Estimation:* Approximate filtering density and mean

$$\text{Filtering density: } \hat{p}(x_{1:k} | y_{1:k}) = \sum_{i=1}^N w_{k|k}^i \delta(x_{1:k} - x_{1:k}^i)$$

$$\text{Mean: } \hat{x}_{1:k} \approx \sum_{i=1}^N w_{k|k}^i x_{1:k}^i$$

3. *Resampling:* Optionally at each time, take N samples with replacement from the set $\{x_{1:k}^i\}_{i=1}^N$ where the probability to take sample i is $w_{k|k}^i$ and let $w_{k|k}^i = 1/N$.
4. *Time update:* Generate predictions according to the proposal distribution

$$x_{k+1}^i \sim q(x_{k+1} | x_k^i, y_{k+1})$$

and compensate for the importance weight

$$w_{k+1|k}^i = w_{k|k}^i \frac{p(x_{k+1}^i | x_k^i)}{q(x_{k+1} | x_k^i, y_{k+1})}$$

2.3 Positioning techniques

2.3.1 Trilateration

Trilateration is a method to determine an unknown location using two or more other known locations and the measured distances between the known and unknown locations. The method can be used with different geometries depending on the needs of the application. It is usually required to have at least three distance measurements, but in theory, an increased number of measurements should lead to better estimation of the location [21].

The measured distance between the target, i.e., the transmitter with an unknown location, and the receiver creates a circle where the measured distance is the radius and origin is the location of the receiver. After collecting a number of distance measurements, the intersection between the circles can be computed as a point of interest. In a two-dimensional space, the trilateration can be illustrated as in Figure 2.1, and computed using (2.12).

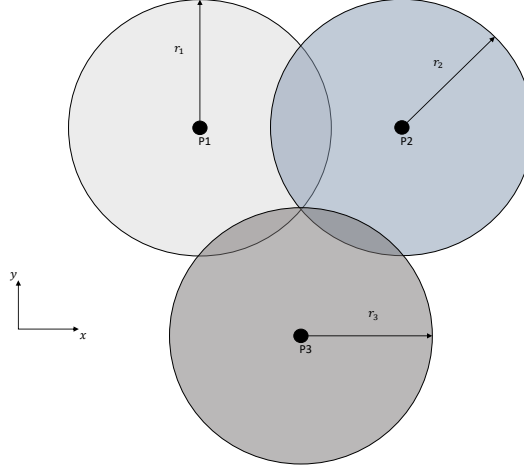


Figure 2.1: Trilateration with three distance measurements in two-dimensions.

$$\begin{aligned} r_1^2 &= x^2 + y^2 \\ r_2^2 &= (x - d)^2 + y^2 \\ r_3^2 &= (x - i)^2 + (y - j)^2 \end{aligned} \quad (2.12)$$

Here, r_1 , r_2 and r_3 are the radii of the circles, while x and y denotes the potential position of the missing person. The variables i , d and j are offsets in position.

2.3.2 Gauss-Newton

The Gauss-Newton method is a modification of Newton's method, used to find the minimum of a function. The method is numerically optimizing and finds the target starting with an initial guess $\hat{x}^{(0)}$ which is then moved towards the target along a search direction $f^{(i)}$ with step size $\alpha^{(i)}$ according to:

$$\hat{x}^{(i+1)} = \hat{x}^{(i)} + \alpha^{(i)} f^{(i)} \quad (2.13)$$

The Gauss-Newton algorithm includes 6 steps.

1. Determine initial guess $\hat{x}^{(0)}$, function $f(x)$ and compute its gradient $J(x) = -\frac{\partial h^T(x)}{\partial x}$, also set $i := 0$.
2. Set $\alpha^{(i)}$ to some value.
3. Compute new estimate using (2.13)

$$\hat{x}^{(i+1)} = \hat{x}^{(i)} + \alpha^{(i)}(J(x)J^T(x))^{-1}J(x)(y - h(x)) \quad (2.14)$$
4. If cost $V(\hat{x}^{(i+1)}) > V(\hat{x}^{(i)})$, set $\alpha^{(i)} := \alpha^{(i)}/2$ and repeat step 3.
5. Stop the process if the change in cost, estimate or size of the gradient is small enough or if the maximum number of iterations has been reached.
6. If not terminated, set $i := i + 1$ and repeat from step 2.

2.4 Planning algorithm frameworks

2.4.1 Markov decision process

A Markov decision process (MDP) is a mathematical framework for sequential decision-making problems in situations where the outcomes are partly random and partly under the control of the decision maker, referred to as the agent. At each time step t , the agent receives some representation of the environment's state $s_t \in S$ and the agent may choose any action $a_t \in A(S_t)$ that is available in s_t . One time step later, in part as a consequence of its action, the agent finds itself in a new state s_{t+1} and receives a corresponding numerical reward $r_{t+1} = R(s, a)$ [23]. The process is illustrated in Figure 2.2.

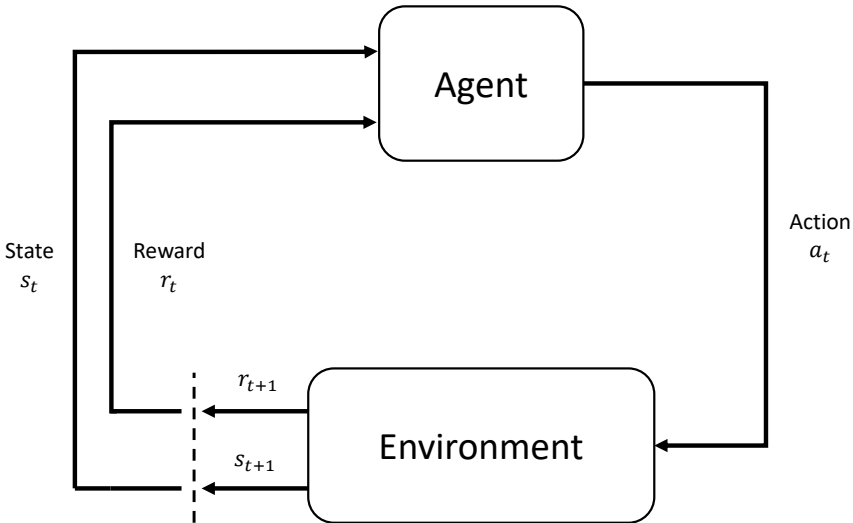


Figure 2.2: Interaction between agent and environment.

Framework

A Markov decision process can be described as a 4-tuple (S, A, T, R) where:

- S is a set of states called *state space*.
- A is a set of actions called *action space*.
- $T : S \times A \rightarrow \Pi(S)$ is the *state-transition function* giving, for each state and action, a probability distribution over states. Denoted as $T(s' | s, a)$ for the probability of ending up in state, s' given state s and action a .
- $R : S \times A \rightarrow \mathbb{R}$ is the reward function. Denoted as, $R(s, a)$ it returns the immediate reward for taking action a in state s .

The state and action spaces may be finite or infinite. In this model, the next state and expected reward only depend on the previous state and the action taken. This is known as the *Markov property* - the state and reward at time $t + 1$ is only dependent on the state and action at time t [24].

Optimization objective

The goal in a Markov decision process is for the agent to act in a way that maximizes the cumulative sum of rewards over the long run. In the *finite-horizon* model, the agent should act in order to maximize the expected sum of rewards that it gets on the next k steps. Thus, the agent should act to maximize:

$$\mathbf{E} \left[\sum_{t=0}^{k-1} r_t \right] \quad (2.15)$$

Here, r_t denotes the reward received on step t . However, it is rare that an appropriate k will be known exactly. In these cases the *infinite-horizon discounted* model is used which sums the rewards over an infinite time, but discounts the rewards geometrically using the *discount factor* $0 \leq \gamma \leq 1$. The agent should then act so as to optimize:

$$\mathbf{E} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \quad (2.16)$$

The discount factor determines the present value of future rewards. In (2.16), rewards received earlier in its lifetime have a higher value. The smaller the discount factor, the less effect future rewards have on the current decision-making [24].

Policy

A policy π is the description of the behavior of the agent. A policy can be *stationary* and *non-stationary*. A stationary policy, $\pi(s) : S \rightarrow A$ is a state action mapping that specifies, for each state, an action to be taken. The choice of action

depends only on the state and is independent of the time step. A non-stationary policy $\pi(s_t)$ is used to choose the action on the t :th-to-last step as a function of the current state, s_t [24].

In the finite-horizon model, the optimal policy is not typically stationary. The chosen actions towards the end of the horizon are generally going to be different from those in the beginning. In the infinite-horizon discounted model, the agent always has a constant expected amount of time remaining, so there is no reason to change action strategies: there is a stationary optimal policy [24].

2.4.2 Partially observable Markov decision process

In a Markov decision process the underlying states can be directly observed, and therefore the optimal policy π can be calculated and used by executing $\pi(s)$ for the current state s . In a partially observable Markov decision process (POMDP) the agent can not directly observe the underlying states (e.g., position of the agent) with complete reliability. Instead, the agent makes an observation based on the action and resulting state [24].

Framework

A partially observable Markov decision process can be described as a 6-tuple (S, A, T, R, Ω, O) where:

- S, A, T, R describe a Markov decision process.
- Ω is a finite-or continuous set of observations the agent can experience.
- $O : S \times A \rightarrow \Pi(\Omega)$ is the *observation function* which gives, for each action and resulting state, a probability distribution over possible observations. Denoted as $O(o | s', a)$ for the probability of making observation o given that the agent took action a and reached state s' .

At time step t , the environment is in some state $s_t \in S$. The agent takes action $a_t \in A(s_t)$. One time step later, the environment have transitioned to $s_{t+1} \in S$ with probability $T(s_{t+1} | s_t, a)$. The agent receives observation $o \in \Omega$ with probability, $O(o | s_{t+1}, a)$ and at the same time the agent receives a reward r_{t+1} equal to $R(s_t, a)$. The process repeats itself for each action taken by the agent. The agent's goal remains to maximize the expected discounted future reward [24].

Belief MDP

Since the agent can not directly observe the environment's states, it must make decisions under the uncertainty of the true state of the environment. A solution to this problem is to keep internal *belief states*. The agent makes observations, generates actions and keeps an internal belief state b which represents the probability distributions over states of the world (e.g., the probability that the agent is located in a certain cell in a grid world). These distributions provide a basis for acting under uncertainty and compromise a sufficient statistic for the past

history and initial belief state of the agent. Given the agent's current belief state, no additional data about its past observations or observations would supply any additional information about the state of the world. The process over belief states is, therefore, *Markov* and no additional data about the past would help increase the agent's expected reward [24].

An illustration of the *Belief MDP* is presented below in Figure 2.3. The component labelled SE is the *state estimator*, responsible for updating the belief state based on the current observation, last action and previous belief state. The component labelled π is, as previously, the policy that is responsible for generating actions as a function of the agent's belief state.

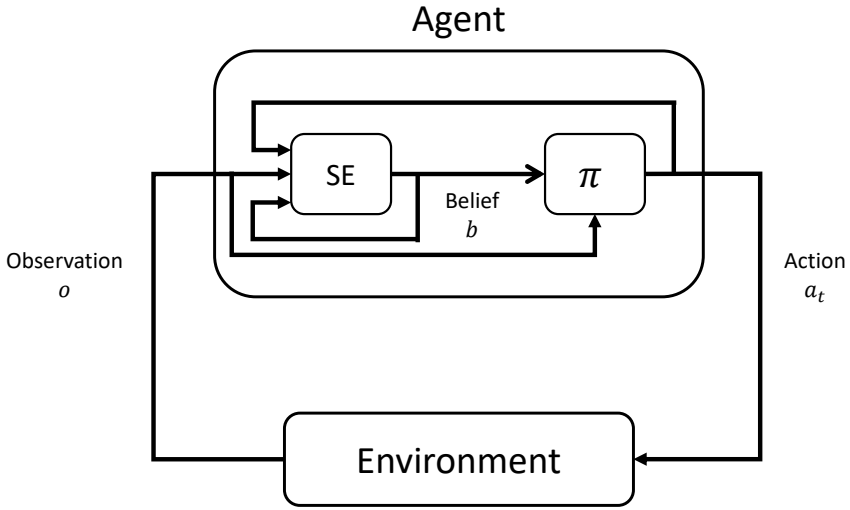


Figure 2.3: Decomposition of a POMDP agent into a state estimator (SE) and a policy (π).

Computing belief states

A belief state b is defined as the probability distribution over S . Let $b(s)$ denote the probability assigned to state s by belief state b . Probabilistic laws require that $0 \leq b(s) \leq 1$ for all $s \in S$ and that $\sum_{s \in S} b(s) = 1$. The new degree of belief in some state s' , $b'(s')$ can be obtained from basic probabilistic theory:

$$\begin{aligned}
 b'(s') &= P(s' \mid o, a, b) \\
 &= \frac{P(o \mid s', a, b)P(s' \mid a, b)}{P(o \mid a, b)} \\
 &= \frac{P(o \mid s', a) \sum_{s \in S} P(s' \mid a, b, s)P(s \mid a, b)}{P(o \mid a, b)} \\
 &= \frac{O(o \mid s', a) \sum_{s \in S} T(s' \mid s, a)b(s)}{P(o \mid a, b)}
 \end{aligned} \tag{2.17}$$

The denominator $P(o | a, b)$, can be treated as a normalizing factor, independent of s' , that causes b' to sum to 1. The output of the state-estimation function $SE(b, a, o)$ is therefore the new belief state b' [24].

Framework and optimal policy π

A belief MDP can be described as a 4-tuple (B, A, τ, R) where:

- B is a set of belief states over the POMDP states
- A is the same set of actions as for the original POMDP
- τ is the *state-transition* function, defined as

$$\tau(b, a, b') = P(b' | a, b) = \sum_{o \in \Omega} P(b' | a, b, o)P(o | a, b) \quad (2.18)$$

where

$$P(b' | a, b, o) = \begin{cases} 1 & \text{if } SE(b, a, o) = b' \\ 0 & \text{otherwise} \end{cases} \quad (2.19)$$

- $R : B \times A \rightarrow \mathbb{R}$ is the reward function on belief states, defined as

$$R(b, a) = \sum_{s \in S} b(s)R(s, a) \quad (2.20)$$

The policy component of a belief MDP is responsible, as presented in Figure 2.3 for mapping the current belief state into an action. Even if the original POMDP has a *finite* number of states, the resulting belief MDP will have a *continuous* state space since there is an infinite amount of belief states (in B) as a result of the infinite amount of probability distributions over the states (S). The belief MDP is defined in such a way that the optimal policy (π), coupled with the correct state estimator, will give rise to optimal behavior for the original POMDP [24].

2.5 Reinforcement learning

Reinforcement learning (RL) is one out of three main subareas of machines learning, and is unique in the way it learns. The idea is to learn not by being provided the correct solution, but by interacting with the environment [23]. The method have proven to be very effective when faced with interactive problems. Instead of providing an immense number of scenarios and what decisions to make, one can define a framework for what a bad or good decision is and let the agent learn using trial and error. The three main elements in RL, if we disregard the obvious agent and environment, are:

- **Reward:** An instant numeric score the agent receives from the environment after successfully reaching a new state. The purpose of the reward is to encourage the agent to make decision which lead to some kind of desired result.

- **Value function:** An estimation of how good an action is in the long run. One could describe the value given by the value function as the total amount of rewards an agent can expect over a given future. The purpose of the value function is to guide the agent to not only select actions which lead to a high instant reward, but to encourage actions which generates a high rewards over a sequence of actions.
- **Policy:** The policy describes what actions to take in a specific state. The purpose of the policy is to make sure that the agent, placed in a specific state, takes the action that will generate the highest sum of rewards.

The agent will initially explore the environment to learn its characteristics, and later exploit the gathered knowledge to evolve into making more and more intelligent decisions. This balance between exploration and exploitation (exploration rate) is one of the main challenges in RL and revolves around the trade-off between taking actions that the agent know will generate high rewards and discover new actions that could potentially lead to higher rewards in the future [23].

2.5.1 Value iteration

Value iteration is used to compute the optimal MDP policy and its state values. The value iteration algorithm uses the Bellman optimality equation to update the state values iteratively:

$$v_{k+1}(s) = \max_a p(s', r|s, a)[r + \gamma v_k(s')] \quad (2.21)$$

Where $v_{k+1}(s)$ is the updated value in state s , $p(s', r|s, a)$ is the transition probability dependent on state s and action a , r is the reward received in the new state s' , γ is the exploration rate and $v_k(s')$ is the value of the new state s' . The optimal MDP policy is computed by iterating over all states and feasible actions until a value convergence is reached. As (2.21) describes, a state value for each action is computed where the highest state value determines the new state value and the policy [23].

2.5.2 Q-Learning

Q-learning is a type of off-policy temporal difference learning method which tries to learn optimal policies without explicit knowledge of the environment. The idea is to update the Q-value, $Q(s_t, a_t)$ (also called state action value) for state s_t and action a_t by adding the temporal difference after each step taken by the agent:

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \underbrace{\alpha [R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]}_{\text{temporal difference}} \quad (2.22)$$

The state action value is updated using the temporal difference error containing α , which is the learning rate and determines to what extent the new observation will impact the new state action value. R_{t+1} is the instant reward generated by the action taken. γ is the discount factor, determining the importance of future state action values. A low discount factor results in a solution that tends to care less about future rewards than previous ones [23].

2.5.3 Deep Q network

When approaching a problem with large state spaces, Q-learning becomes inefficient [23]. The issue with big and sometimes infinite state spaces can however be solved using neural networks (NN). Instead of storing Q-values for each state and action, the state space is fed into an NN which in return generates an approximation of the Q-values for each feasible action. The NN is essentially an approximation of the Q-function. The DQN is an example of this and includes a convolutional neural network (CNN), an experience replay buffer and a target network [18], all explained in the following sections.

Neural network

A neural network is a group of connected nodes, loosely modeled as the neurons in a biological brain. The goal of a neural network is to approximate a function $y = f^*(x)$, able to predict the output y given an input x . To acquire this function f^* , the feed-forward and back-propagation techniques are used to learn parameters θ (weights and biases) for $y = f^*(x; \theta)$ that in the best way possible approximate the function [25]. Each training instance contains the following steps:

- Forward pass input x through the NN to make prediction y .
- Measure the error compared to the true value.
- Backward pass through all layers in reverse to measure the error's contribution from each connection.
- Modify the weights to reduce error.

Compared to many other methods, a feed-forward NN do not use a feedback loop when predicting the output. Instead, the parameters are changed sequentially during training until an equilibrium is reached, meaning that the network predictions converge to some kind of true or desired value [25].

A feed-forward NN with one input, two hidden and one output layer is illustrated in Figure 2.4. Each node (or neuron) in the network is a mathematical function which generates a real value given the weighted sum of the outputs from the previous neuron layer. The equations in Figure 2.4 are formulated for layer k and layer $j = k - 1$ with n and m nodes. (2.23a) and (2.23b) describe the two functions that build one node operation:

$$z_m = \sum_{i=1}^n w_{i,j} x_i + b_j \quad (2.23a)$$

$$y_k = f_a(z_m) \quad (2.23b)$$

The linear combination in (2.23a) depends on the output $x_{j,i}$ and weights $w_{j,i}$ from the previous layer j . The linear combination, z_m acts as input together with a bias b_j into the activation function f_a in (2.23b). The activation function can be both non-linear and linear, but non-linear activation functions are more often used since they make the network more flexible [25].

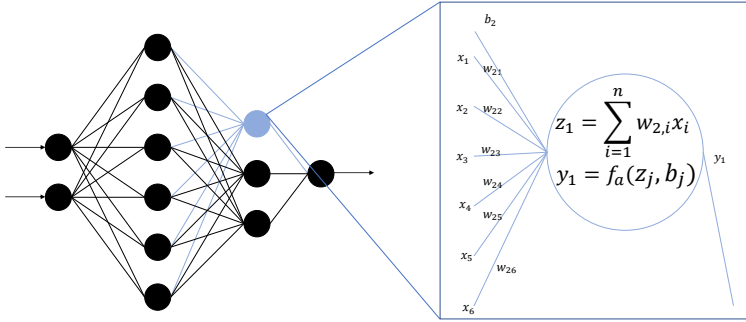


Figure 2.4: Illustration of a neural network including node computations.

Convolutional neural network

CNNs are a subclass of neural networks created to process grid-like data. The difference between the NN explained above and the CNN are the convolutions, a linear operation performed on the input to enable a more efficient computation. The convolutional layers use a filter (called kernel) to slide over the input data, as illustrated in Figure 2.5. This operation provides something called feature maps, which are linear combinations of the information visible to the kernel in each step. The trick is in using a kernel which is smaller than the input. For example, processing an image containing thousands or millions of pixels using a traditional NN would require each pixel to be processed as an individual input variable. However, using a kernel which slides over a group of pixels, small features can be detected using only tens or hundreds of pixels [25].

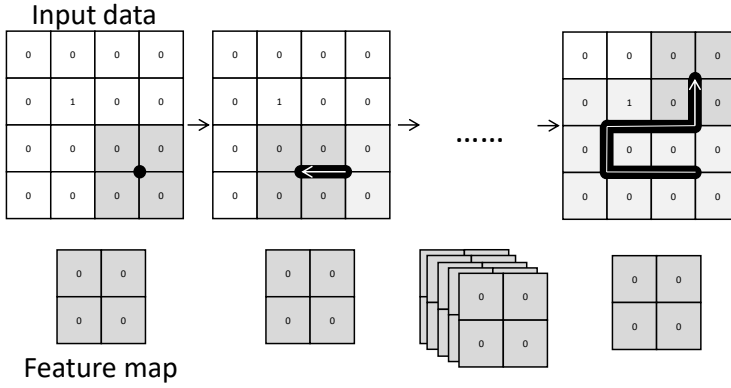


Figure 2.5: Illustration of the process of computing feature maps including kernel and stride in convolutional layers.

An illustration of a CNN can be found in Figure 2.6. The example CNN contains three convolutional layers, one flattening layer (converts the data into 1D) followed by a NN with two hidden layers.

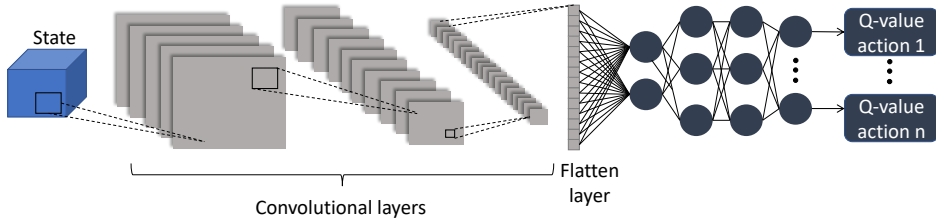


Figure 2.6: Illustration of deep Q network with convolutional layers.

Experience replay and target network

A problem in using neural networks in reinforcement learning is the instability the non-linearity causes. Experience replay is one of the main breakthroughs which improved the stability of the DQN greatly. The method involves storing the agent experience $e_t = (s_t, a_t, r_t, s_{t+1})$ at each time step t in a data set $D_t = \{e_1, \dots, e_t\}$. During learning, batches are sampled uniformly at random from the stored data set $(s, a, r, s') \sim U(D)$ and used to update at iteration i using the loss function (2.24). The benefit in this is that the network will be optimized using not only the most recent experience, but also experience from other parts of the training run. This leads to the network constantly being faced with different scenarios, not only the most recent, making it more stable over time. The loss function:

$$L_i(\theta_i) = E_{(s,a,r,s') \sim U(D)}[(r + \gamma \max_a Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2] \quad (2.24)$$

reveals the second method to partly solve the problem with instability. By introducing a target network, identical to the policy network except using the parameters θ_i^- which are updated by the main policy network's parameters θ_i every C steps. This sequential update of the parameters lead to the target Q-value provided by the target network is fixed between updates. Meaning that there will be no moving target to optimize towards [18].

3

Method

This thesis aims to investigate if a novel planning algorithm can enable more efficient use of UAVs in SAR missions. The planning algorithm should be able to locate a missing person based on RSS-measurements and an initial belief of the person's whereabouts. Since SAR operations often involve large search areas, the planning algorithm has been divided into two stages:

- **Stage 1:** Locate an RSS-signal.
- **Stage 2:** Isolate the signal and provide a location estimate.

The objective of the first stage is to locate an RSS-signal. The initial belief, of the missing person's whereabouts is used to create a trajectory that prioritizes areas with a high probability (POA). As soon as the RSS-signal is detected, the second stage is initiated. The second stage uses a state estimator, based on filtered RSS-measurements, to update the belief regarding the missing person's location. A DQN is simultaneously used to select appropriate actions which creates a trajectory, where the goal is to locate the missing person with as high precision as possible. The movement is goal oriented since the network has been trained on millions of simulations and therefore can understand the consequences of different actions. As illustrated in Figure 3.1, Stage 1 is an open loop system where the initial belief is the only input. Stage 2 is a closed looped where each step generates input to the upcoming decision. The agent will initially follow the path given by the Stage 1 algorithm, and switch to Stage 2 as soon as an RSS-signal is detected.

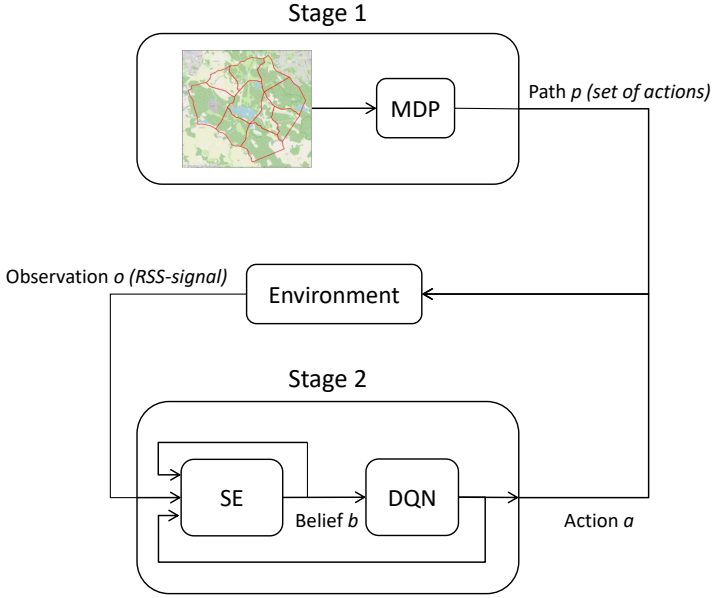


Figure 3.1: System overview including Stage 1 and 2 as well as the environment.

3.1 RSS-signal

The simulated RSS-signal is modeled according to the theory in Section 2.1. A normal distributed error is added to the signal to model interference due to factors such as terrain and weather. The result of the added error are illustrated in Figure 3.2 and the parameters are presented in Table 3.1. Let y denote the measured RSS-signal, which is a function of the missing person's position x and the positions of the UAVs sensor p according to:

$$y = h(x, p) + e, \quad e \sim N(0, \sigma^2)$$

$$h(x, p) = P_0 + 10\beta \log_{10}(\|x - p\|)$$

Table 3.1: Table of the parameters used to model the simulated RSS-signal.

Path loss exponent β	Reference distance d_0	$P_0(d_0)$	Standard deviation σ
2	1 m	40 dBm	8 dBm

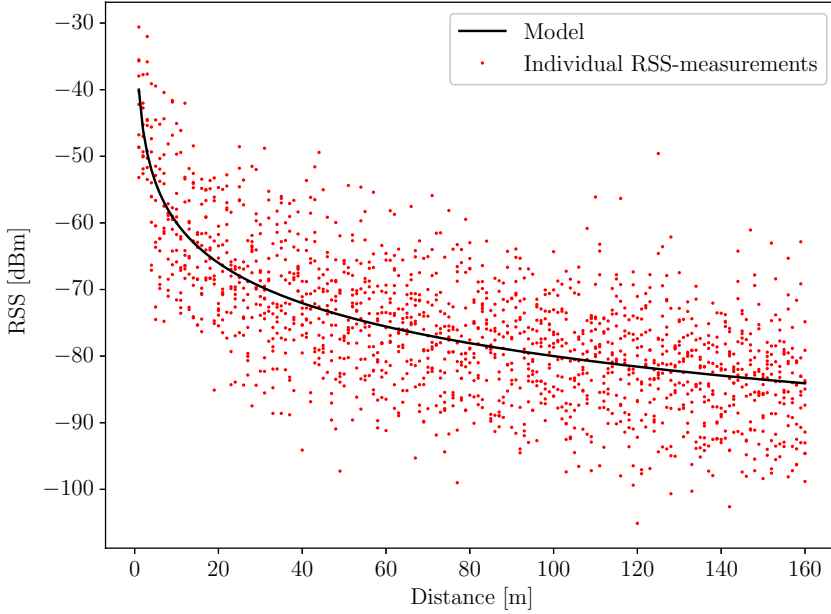


Figure 3.2: RSS over distance distribution. Red dots represent single RSS-measurements, while the black curve represents the log-distance path loss model.

3.2 SAR environment

The SAR environment in this thesis consists of an area outside of Linköping, located in "Tinnerö Eklandskap". The area is divided into ten different sectors, which all are assigned probabilities p , based on how likely it is that the missing person is located in the sector. The areas of the different sectors range from 1.04-2.65 km² and the total area is 15.47 km². The environment and its sectors are presented in Figure 3.3 and the different probabilities are illustrated in Figure 3.4.

$$\sum_{i=1}^{10} p_i = 1$$

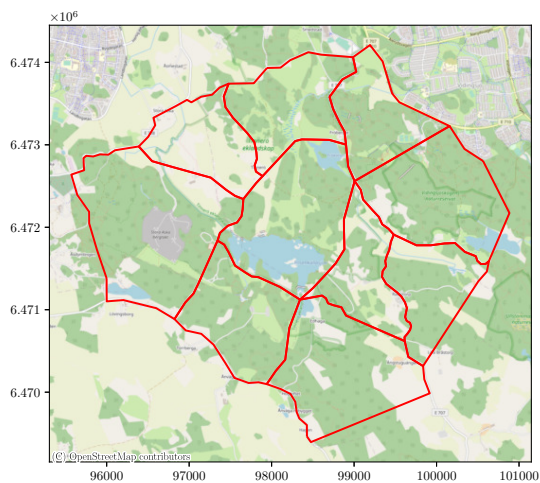


Figure 3.3: SAR environment divided into 10 different sectors. The red lines represent the border of the sectors.

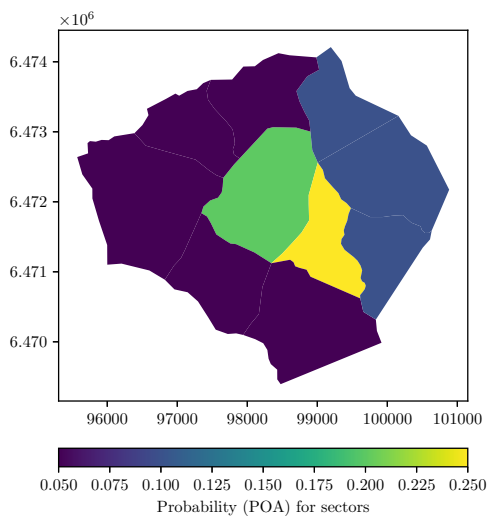


Figure 3.4: Initial probabilities for different sectors. It can be noted that the sectors east of the centre of the search area generally hold higher probabilities.

3.3 Stage 1 - signal localization

The objective of the first stage is to locate an RSS-signal using the initial belief regarding the missing person's location. By formulating the problem as a MDP, a trajectory that prioritizes areas with a high initial belief, is developed.

3.3.1 Formulation of MDP

Environment: The SAR environment is discretized into an n by n grid with square cells, where the cells in the grid are $\mathcal{C} = \{c_{i,j} \mid 1 \leq i, j \leq n\}$, i.e., $c_{i,j}$ refers to the cell in the i -th row and the j -th column. Each cell is assigned an individual probability $p_{i,j}$, based on the probabilities of the sectors it constitutes parts of. Cells that are located outside of the search area are assigned zero probability. Additionally, each cell has the binary feature $\mathcal{V} = \{Visited, Unvisited\}$ indicating if the cell has been visited by the UAV (agent) previously. This binary feature correlates with the cell's probability. Visited cells are assigned zero probability while cells, with initial probability zero, are marked as visited.

UAV: Let \mathcal{U} denote the UAVs position in the grid. The UAV can move in the grid from its current cell to any adjacent cell in the grid, or hover over the current cell. The motion of the UAV is modeled by a set of actions $\mathcal{A} = \{H, N, NE, E, SE, S, SW, W, NW\}$. The first action corresponds to hovering (staying) in the current cell, while the next actions correspond to the cardinal and intercardinal compass directions. The effects of all the actions are deterministic and correspond to moving in the desired direction.

State space: The state space \mathcal{S} is the Cartesian product of the UAVs position (\mathcal{U}) and the status of the cells (\mathcal{V}). Thus, the state space can be denoted as $\mathcal{U} \times \mathcal{V}$. The number of states can be calculated through $(n \cdot n) \cdot 2^{n \cdot n}$.

Rewards: When the agent takes an action and moves into a cell, it receives a reward based on the cell's probability. Additionally, the rewards for the cells in the inner grids are based on both their probabilities and their distance to the next cell in the route of the outer grid. This is explained in more detail in Section 3.3.2.

3.3.2 Hierarchical solution

Due to the nature of the state space, the number of states quickly becomes very large when the number of cells (n by n) increases. For example, a 5 by 5 grid results in $25 \cdot 2^{25} \approx 839$ million states. Therefore, a 3 by 3 grid is selected which results in a manageable 4608 possible states.

Each cell is then divided into 3 by 3 sub-grid, and the process is repeated until a cell side length of less than 200 meters is achieved. Thereafter, the problems are solved hierarchically through value iteration. This process is illustrated in Figure 3.5 where the blue squares represent the cells in the outer grid, while the red squares represent the cells in the inner grid. To allow smooth transitions and

thus avoiding visiting the same cells multiple times, rewards in cells of the inner grids are based on both their probabilities and their distance to the next cell in the route of the outer grid. As an example, consider $\mathcal{C} = \{c_{i,j} \mid 1 \leq i, j \leq 3\}$ the cells of the outer grid in Figure 3.5. The rewards of the cells in the inner grid (red squares) would then depend on their probability and their distance to cell $c_{3,2}$.

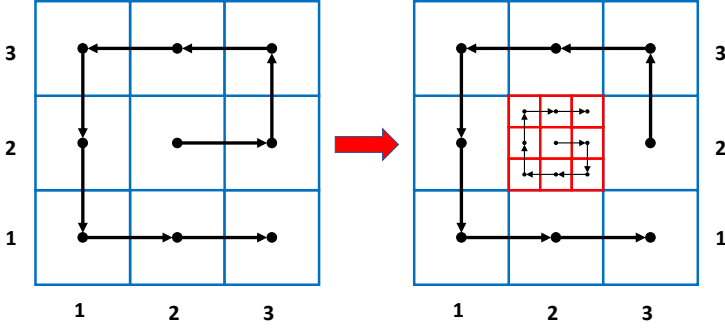


Figure 3.5: Example of hierarchical solution for multiple MDPs.

3.4 Stage 2 - signal isolation

The objective in the second stage is to isolate the received RSS-signal. This is done by formulating the problems as a belief-MDP and developing a state estimator, able to estimate a belief state given the RSS-measurements as well as a deep Q-network which acts as an action policy.

3.4.1 Formulation of POMDP

Environment: The environment, in Stage 2, shares multiple similarities to the environment in Stage 1. It is discretized into an m by m grid with square cells of size δ . The cells in the grid are $\mathcal{C} = \{c_{i,j} \mid 1 \leq i, j \leq m\}$, i.e., $c_{i,j}$ refers to the cell in the i -th row and the j -th column. However, the binary feature \mathcal{V} is not included.

UAV: The motion of the UAV (agent) is modeled by the same set of actions as in Stage 1, i.e., $\mathcal{A} = \{\mathbf{H}, \mathbf{N}, \mathbf{NE}, \mathbf{E}, \mathbf{SE}, \mathbf{S}, \mathbf{SW}, \mathbf{W}, \mathbf{NW}\}$. The effects of all the actions are deterministic and correspond to moving in the desired direction. The position of the UAV, in the grid, is denoted as \mathcal{U} and can be directly observed. Non-feasible actions, i.e., actions that would move the UAV outside the grid, results in the UAVs remaining in the current cell.

Missing person: The notation \mathcal{MP} is used to represent the missing person's location in the grid. The person's location in the grid remains stationary and can not be directly observed.

State space: The state space \mathcal{S} is the Cartesian product of the UAV's position (\mathcal{U}) and the missing person's location (\mathcal{MP}). Thus, the state space can be denoted as $\mathcal{U} \times \mathcal{MP}$. The number of states can be calculated through $(n \cdot n)^2$.

Observations: The UAV observes a set of continuous RSS-signals which is a function of the UAV and the missing person's location according to (3.1). The probability of observing a signal ($\mathcal{O}(o|s)$) is 1.

Belief MDP

Since the missing person's location can not be directly observed, a set of belief states (\mathcal{B}) over the POMDP states is formulated. Even though the originating POMDP has a finite number of states, $(n \cdot n)^2$, there is an infinite amount of belief states in \mathcal{B} since there are an infinite number of probability distributions over the states of \mathcal{S} .

Belief states: The belief state is denoted as \mathcal{B} and is a result of the POMDP states, i.e., $\mathcal{B} = \{\mathcal{U}, \mathcal{MP}\}$. It represents the belief regarding the position of the UAV and the location of the missing person. As an example, consider a 2 by 2 grid where the environment is in state $b \in \mathcal{B}$:

$$b = \left(\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} \end{bmatrix} \right)$$

The belief state b indicates that the UAV is located in the top left cell and that there is an equal probability that the missing person is located in the three remaining cells.

Rewards: The reward function contains one positive and two negative rewards. All actions result in a negative reward of -1 while non-feasible actions result in a negative reward of -2. A continuous positive reward of 1 is given when the sum of the belief regarding the missing person's location $\mathcal{B}(\mathcal{MP})$, in a sector of 2 by 2 cells, is larger than 0.95. Thus, the reward is a function of the action and current belief, i.e., $\mathcal{R}(b, a)$.

State estimator

The state estimator (SE) was introduced in Section 2.4.2 and is used to compute a new belief state based on the current observation, last action and previous belief state $\text{SE}(b, a, o)$. In this thesis, the main difficulty lies in updating the belief state in regard to the missing person's location, $\mathcal{B}(\mathcal{MP})$ since the agent's position can be directly observed. Two different, but similar filters have been developed, a point mass filter (PMF) and a particle filter (PF). The main difference between these are that the grid in the PMF is deterministic, while the PF has a dynamic grid. Consider the notation, $b(mp_{i,j})$ as the belief that the missing person is located in cell $c_{i,j}$.

PMF: The PMF has a deterministic grid $\{x^k\}_{k=1}^N$ of a two-dimensional state space with $N = (m \cdot \delta)^2$ points. Each distinct grid point has an associated weight $(\{w^k\}_{k=1}^N)$ which indicates the probability that the grid point represents the true location of the missing person, satisfying $\sum_{k=1}^N w^k = 1$.

- Each grid point is assigned a cell index r , based on which cell $c \in \mathcal{C}$ that the grid points lies within. In an m by m grid, the index r for the cell located in the i -th row and the j -th column can be calculated through $r = i \cdot m + j$.
- The belief states in regard to the missing person's location, $\mathcal{B}(\mathcal{MP})$, can now be calculated. The value, of each belief state, is computed as the sum of all weights for the grid points it contains. For example:

$$b(mp_{i,j}) = \sum_{k=1}^N w^{k,r}, \quad \text{where } w^{k,r} = \begin{cases} w^k, & \text{if } r = i \cdot m + j \\ 0, & \text{else} \end{cases}$$

- The weights are updated through the use of a probability density function (PDF), illustrated in Figure 3.6. Suppose that, at time t based on observations $y_{1:t-1}$, we have computed the relative weights $w_{t|t-1}^k$. The agent receives a new RSS-measurement y_t . The probability for observation y at time t can then be calculated by the following equations:

$$p(y_t | x_t^k, p_t) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{1}{2} \left(\frac{y_t - h(x_t^k, p_t)}{\sigma} \right)^2}$$

$$h(x_t^k, p_t) = P_0 + 10\beta \log_{10} \|x_t^k - p_t\|$$

Where p_t is the agent's position at time t , σ is the standard deviation for RSS-signals and x_t^k is the location of the grid point at time t . The notation x_t may seem unnecessary, since the location of the grid points are constant and thus independent of time. However, it becomes important for the PF, where the location of the particles are dependent on the time. The new weights can then be calculated by:

$$w_{t|t}^k = \sum_{k=1}^n \frac{1}{c_k} p(y_t | x_t^{(k)}) w_{t|t-1}^k$$

$$c_k = \sum_{k=1}^n p(x_t | y_{1:t}) w_{t|t-1}^{(k)}$$

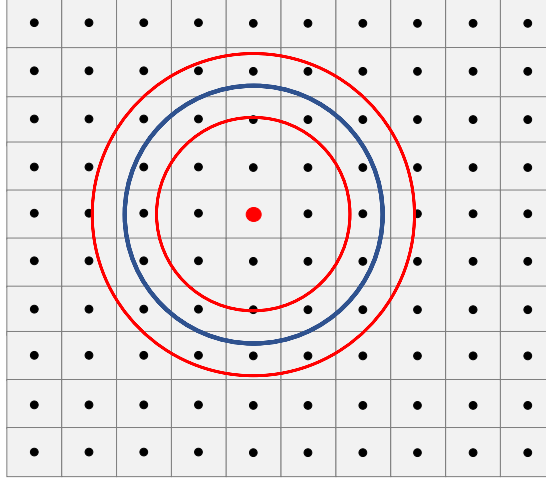


Figure 3.6: Illustration of PDF assuming normal distribution, $N(\mu, \sigma^2)$. The red dot represents the position of the agent and the black dots represent grid points. The blue line represents μ based on measurement y while the red circles represents $\mu \pm \sigma$.

PF: The PF works similarly to the PMF. The main difference is that the grid points are replaced by particles. Each particle has an associated weight that indicates the probability that it represents the missing person's true location. Since the grid is dynamic, the particle's cell index continuously needs to be updated. The value of the belief state $b(mp_{i,j})$ is computed as the sum of the weights for the particles located within the cell $c_{i,j}$. The PF is implemented according to Algorithm 1 and the parameters are presented below in Table 3.3.

Table 3.2: Table of the parameters used to model the particle filter

Parameter	
Number of particles	$N = (m \cdot \delta)^2$
Particles density (particles/m ²)	1
Resample threshold	$N_{th} = N_{eff}/2$

Multiple different resampling methods were evaluated. The agent and missing person were initially placed at random locations in a 30 by 30 grid ($m = 30$) with cell size $\delta = 10$ m and uniform initial particle weights. The agent then sampled random actions, totaling 900 steps within the grid, and the estimation error, for each step, was saved. This process was repeated 300 times for each resample method, and the results are presented below in Figure 3.7. The results demonstrate that all the different methods yield acceptable results. In the end, stratified resampling was selected since it resulted in the lowest final estimation error and

proved to have a lower computational complexity than multinomial-and residual resampling.

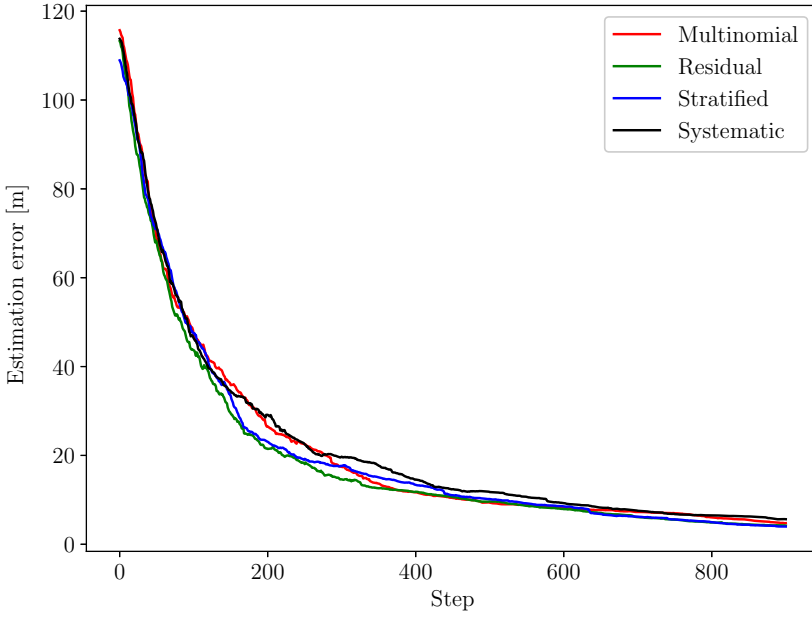


Figure 3.7: Results for different PF resampling methods.

Grid example

To create an understanding of how a belief MDP works, the following grid example has been developed. It consists of a 2 by 2 grid and is presented below in Figure 3.8. The agent's position can be directly observed and is indicated by the circle, while the goal state remains unknown and is indicated by the star. Assume that a state $s \in \mathcal{S}$ is defined by the position of the agent $p \in \mathcal{P}$ and the position of the goal state $g \in \mathcal{G}$. This results in $\mathcal{S} = \mathcal{P} \times \mathcal{G}$ where \mathcal{S} is finite and consists of 16 possible states. The agent maintains a sensor model and can therefore observe when it is located in the goal state ($o \in \mathcal{O}$, $o = 0$ if $p \neq g$ and $o = 1$ if $p = g$). There are four possible actions, *North*, *East*, *South* and *West*. These actions always succeed and only affect the agent's position, the goal state remains stationary. If no movement is possible in a particular direction, the agent remains in the same location. Assume that the agent initially is located in cell 1 and have made observation $o = 0$, meaning non-goal state. Assume that the goal state is equally likely to be located in one of the remaining cells. The initial belief state then becomes:

$$b = \left(\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} \end{bmatrix} \right)$$

A belief state is denoted by two variables, the probability of the position of the agent and the probability of the position of the goal state. For example, $b(1, 2) = (1, 0)$ implies that the probability that the agent is located in cell 1 and that the goal is located in cell 2 is 1 respectively 0.

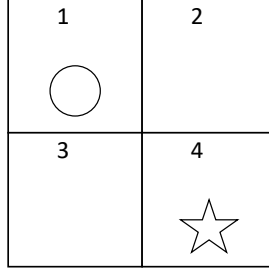


Figure 3.8: Grid POMDP environment.

The agent then takes action *East* and observes its own position and a non-goal state ($o = (2, 0)$). The new belief states can be calculated by the following equations:

$$\begin{aligned}
 b'(2, 1) &= \frac{O((2, 0) \mid (2, 1), East) \sum_{s \in S} T((2, 1) \mid s, East) b(s)}{P(o \mid a, b)} = \dots = \frac{(1, 0)}{P(o \mid a, b)} \\
 b'(2, 2) &= \frac{O((2, 0) \mid (2, 2), East) \sum_{s \in S} T((2, 2) \mid s, East) b(s)}{P(o \mid a, b)} = \dots = \frac{(1, 0)}{P(o \mid a, b)} \\
 b'(2, 3) &= \frac{O((2, 0) \mid (2, 3), East) \sum_{s \in S} T((2, 3) \mid s, East) b(s)}{P(o \mid a, b)} = \dots = \frac{(1, \frac{1}{3})}{P(o \mid a, b)} \\
 b'(2, 4) &= \frac{O((2, 0) \mid (2, 4), East) \sum_{s \in S} T((2, 4) \mid s, East) b(s)}{P(o \mid a, b)} = \dots = \frac{(1, \frac{1}{3})}{P(o \mid a, b)}
 \end{aligned}$$

The new belief state thus becomes:

$$b = \left(\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0.5 & 0.5 \end{bmatrix} \right)$$

3.4.2 Simulation environment

A simulation environment was created using OpenAI Gym, an open source Python library for developing and comparing reinforcement learning algorithms and environments by providing a standard API [26]. The Gym API implements an agent-environment loop, illustrated in Figure 3.9, which is suitable for MDP and POMDP problems.

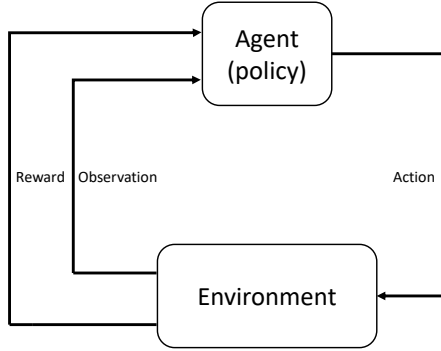


Figure 3.9: Agent-environment loop in OpenAI Gym.

The agent-environment loop introduces a concept called *timestep*, continuously used in this thesis. A timestep consists of the agent performing an action and observing how the environment's state changes. In the Gym, this is done by executing the *step* function which takes an action and then returns an observation, a reward, and a done signal indicating if the environment has entered a terminal state. A done signal may be issued after a fixed number of timesteps or if the agent has succeeded in completing some tasks within the environment.

In this thesis, the observation consists of the current belief state described in Section 3.4.1, which also describes possible actions space and rewards. Two different criterions are used to issue the done signal. The first is based on a fixed number of timesteps while the second criterion is based on Kullback-Liebler (KL) divergence (also called relative entropy), which is a statistical measure of how two probability distributions differ. Consider P the probability distribution, in regard to the missing person's location within the grid, at timestep $t - 1$ and Q the same probability distribution at timestep t . The relative entropy $D_{KL}(Q||P)$ is calculated for each timestep. If the average relative entropy $D_{KL,average}$, for a fixed number of timesteps t_{fix} , falls below a certain limit $D_{KL,limit}$, a done signal is issued ($D_{KL,average} < D_{KL,limit} \rightarrow done$). The second criterion can only be issued when a 2 by 2 section of the grid consists of 0.95 of the missing person's probability distribution within the grid, to ensure that the person's location has been isolated within a certain area. It is also worth mentioning that the agent moves strictly between the centroids of the cells, while the missing person is located in a random, arbitrary position within a cell. The simulation environment was defined according to the parameters presented in Table 3.3.

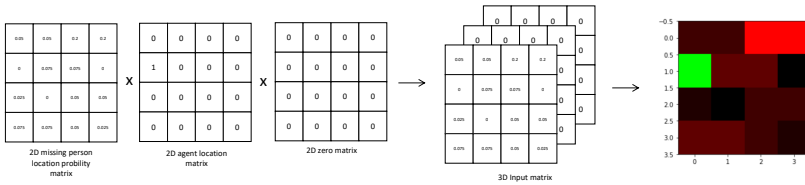
Table 3.3: Parameters used to define the simulation environment.

Parameters	
Grid size [rows, columns]	30 x 30
Cell size [m]	10 x 10
Maximum allowed timesteps	rows * columns
t_{fix}	10
$D_{KL, \text{limit}}$	5e-3

3.4.3 DQN

As mentioned previously, a DQN can be constructed in a few different ways. To simplify the implementation of the DQN and speed up the training, Stable Baselines 3 were used. It is a set of reinforcement learning algorithms implemented in PyTorch, based on the API from OpenAI gym. Stable Baselines 3 includes features such as vectorized environments, thereby allowing faster training of RL agents. The DQN used in this thesis comes from the famous article *Play Atari with Deep Reinforcement Learning* by DeepMind Technologies [27] which used an RGB-image representing the environment as input. The letters, (R)ed, (G)reen and (B)lue are each inputs to one of the three input channels in the first convolutional layer where the agent position is green, and the missing person probability is red. These two are the only variables used to represent the state space, hence the third layer (Blue) contains only zeros.

Figure 3.10 shows the RGB representation of the environment, defined according to the belief state space, with the known location of the agent represented as 1 in the agent's cell and the unknown missing person location representation as a probability distribution. The 3D input matrix, including all three matrices, is transformed to an RGB image, i.e., rescaling the original matrix values from 0-1 to 0-255 before it is feed into the CNN.

**Figure 3.10:** Input frame to DQN

The network is constructed by three convolutional layers followed by a flattening layer and fully connected output layer, all shown in Figure 3.11. The flattening

layer is used to convert the two-dimensional feature maps created by the convolutional layers into one dimensional arrays, able to be feed into each neuron in the output layer. Each output neuron represents a state action value (Q-value), hence nine output neurons are used, one for each action. Additionally, the DQN uses experience replay and a target network, which are both described in Section 2.5.3. All DQN parameters are presented in Table 3.4.

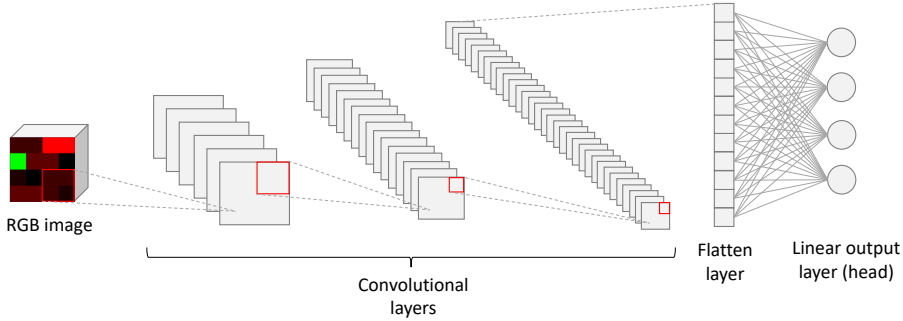


Figure 3.11: Architecture of CNN from Stable Baselines 3.

Training the network

As explained in Section 2.5.3, the CNN is trained to find the set of parameters which best map the input to the output. This is done using Algorithm 2.

Algorithm 2 Training loop

```

for episode in training_session do
  state = environment.reset()
  done = False
  while done == False do
    action = select_action(state)
    observation, reward, done = environment.step(action)
    next_state = state_estimator(observation)
    memory.push(state, action, next_state, reward)
    batch = optimize_model(memory)
    state = next_state
  if episode % TARGET_UPDATE == 0 then
    target_network = policy_network

```

The action was selected using the ϵ - Greedy policy, which selects the action based on the exploration rate, denoted as ϵ , previously mentioned in Section 2.5. Initially ϵ was set to 1, meaning that 100% of the actions were randomly sampled and not selected using the network. The variable was then linearly decreased over the training session to a final value of 0.05.

The initial location of the agent is randomly sampled from one of the cell centroids and the missing person location is sampled depending on the initial prob-

ability distribution, meaning that the person is more likely to be placed in a cell with high probability. Both positions are resampled each time the environment is reset.

Table 3.4: Hyperparameters used in DQN model.

Hyperparameter	
ϵ_{\min}	0.05
ϵ_{\max}	1
Optimizer	Adam
Activation function	Tanh
Learning rate	1e-4
Loss function	Huber loss
Batch size	32
Buffer size	1e6
Target update	1e4
γ	0.99

3.5 Evaluation methods

As mentioned in Section 3.4, Stage 2 consists of two main parts. A state estimator, used to estimate the current belief state of the environment and a deep Q-network, used to select appropriate actions based on the estimator’s belief state. The state estimator provides a localization estimate, while the DQN acts as the path planner, hence the algorithm developed in Stage 2 was evaluated on those two attributes, its localization and path planning capabilities. Since two different state estimators were developed, an additional evaluation of the PMF and PF was carried out.

3.5.1 Path planning evaluation

The DQNs path planning capabilities were evaluated using three different path planners based on three different policies. These consist of a greedy, a lawn-mower and a random policy. All path planners use the PMF as the state estimator and the KL-divergence is deactivated, meaning that the episode is only terminated when the maximum allowed steps are reached.

The greedy policy has access to the same information as the DQN, i.e., the agent’s location and the missing person’s probability distribution generated from the state estimator. The greedy policy always select the action which moves the agent towards the global maximum, i.e., the cell with the highest probability. The method relies on the current state representation and exploits it to the fullest. An illustration of the greedy policy is presented in Figure 3.12b.

The second evaluation algorithm used to test the performance of the DQN is a lawn-mower policy, illustrated in Figure 3.12a. The last evaluation policy is a random policy, i.e., each agent action is sampled randomly.

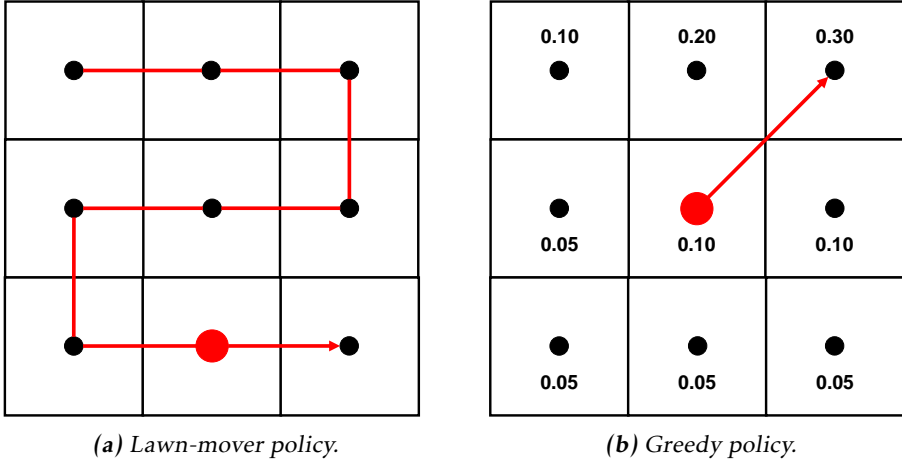


Figure 3.12: Different policies used to evaluate the DQN.

3.5.2 Localization evaluation

To evaluate the best performing state estimator, developed in this thesis, Gauss-Newton and trilateration were used as comparisons. The different localization systems were compared using identical agent paths, and the initial location of the missing person and agent was the same for all the simulations. The path was determined by the greedy policy which, as explained in Section 3.5.1, always guides the agent towards the global probability maximum.

At each timestep t , the RSS-measurement y_t was used to directly estimate the distance $d_{e,t}$ between the agent and the missing person. The estimated distance was then saved, together with the agent's position p_t , in a list. A distance limit of 100 meters was used to sort out unreliable measurements, i.e., estimated distances larger than 100 meters were neglected. Additionally, both Gauss-Newton and trilateration used a measurement limit of 5 measurements, i.e., at least five saved measurements $(d_{e,t}, p_t)$ were required to compute localization estimates for both methods. The Gauss-Newton method needs a good initial estimate to function well. In this thesis, the initial estimation of the algorithm was the position of where the strongest RSS had been received. If no RSS-signal had been received, the initial estimation was randomly sampled from the grid. The function, minimized by the Gauss-Newton algorithm, was:

$$S(\hat{p}) = \sum_{i=1}^m r_i(\hat{p})^2 \quad (3.2)$$

The equation shows the sum of squared residuals, where \hat{p} is the position estimate of the missing person and m is the number of observations. The residuals were computed as follows:

$$r_i(\hat{p}) = d_e - |\hat{p} - p| \quad (3.3)$$

Here, p denotes the position of the agent. At each timestep, the Gauss-Newton position estimate \hat{p}_t , was saved to a list, as long as the estimated position was within the grid. The final position estimate was computed as the mean of all prior estimates.

The trilateration method iterates over the gathered measurement list, where each position estimate was computed using 3 sequential measurements. The agent position p , from each measurement, represented the origin of a circle with each related d_e as the radius. If an intersected area between the three circles exists, the position estimate was set to the centroid of the area. If no intersection existed, no estimation was given. The final position estimate was computed as the mean of all prior position estimations.

4

Results and evaluation

This chapter consists of the thesis results and an evaluation of these. In Section 4.1, the results for the system of hierarchical MDPs, used in Stage 1, is presented and evaluated. Section 4.2 presents and evaluates the training of the two DQN-models used in this thesis, while Section 4.3 compares the performance of the these models against each other. Section 4.4 and Section 4.5 are used to present and evaluate the path planing capabilities of the DQN and the localization capabilities of the selected state estimator. Finally, Section 4.6 present an illustration of the selected state estimator and DQN for a sequence of belief states.

4.1 Hierarchical MDPs

The purpose of Stage 1 is to locate an RSS-signal. This is done by solving a system of hierarchical MDPs which results in a trajectory that should prioritize areas with a high probability. Thus, tests were conducted to evaluate how the initial probabilistic map (IPM) affects the trajectory for the hierarchical solution. These tests were based on two different sets of precalculated trajectories, referred to as *Set 1* and *Set 2*. For both sets, the agent was initially located in the centroids of the different cells in the outermost grid. Each set therefore contained nine different trajectories, since there are nine different cells in the outermost grid. The difference, between the two sets, was that the trajectories in Set 1 considered the IPM while the trajectories in Set 2 did not take the IPM into consideration. This was done by assigning individual probabilities, based on the IPM, to the cells in Set 1 while all cells in Set 2 were assigned equal probabilities. Figure 4.1 and Figure 4.2 illustrates the trajectory, for Set 1 respectively Set 2, when the agent was initially placed in the centroid of the cell located to the upper left in the outermost grid.

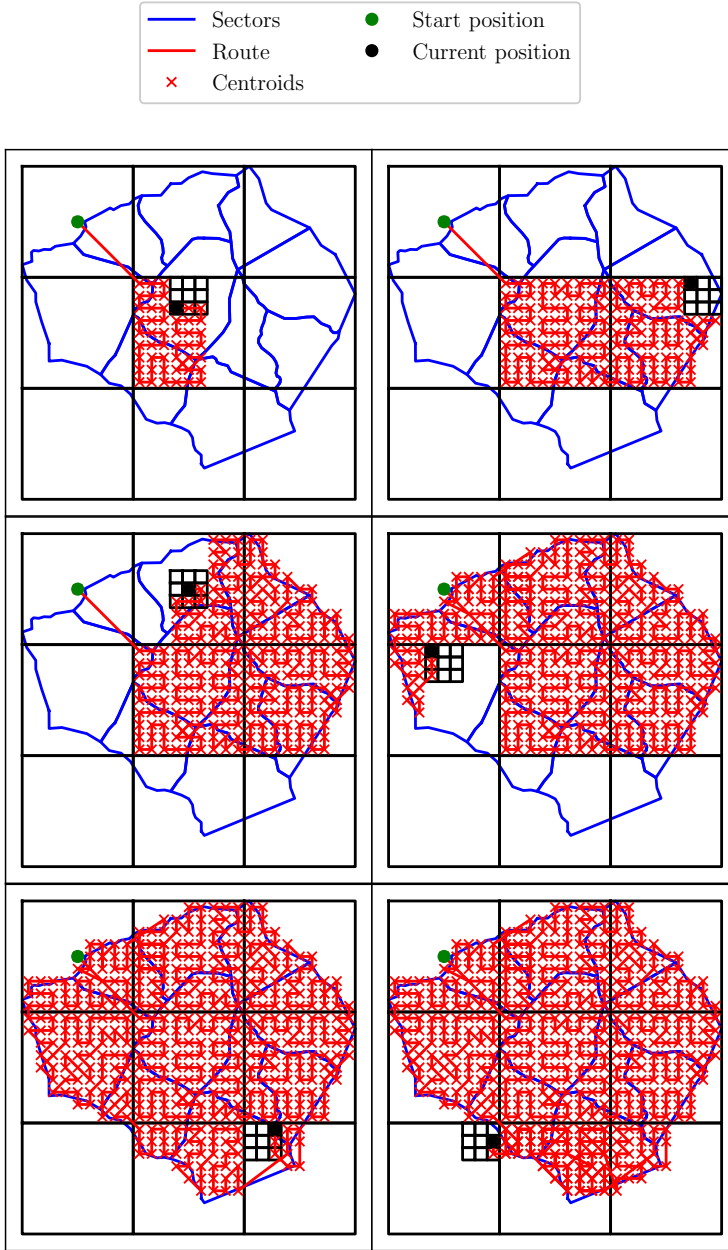


Figure 4.1: Agent trajectory for Set 1 when initially placed in the centroid of the cell located in the upper left of the outermost grid. The sectors in the search area are represented by the blue lines. The red lines represent the path taken by the agent, while the green and black dot represent the start-respectively current position of the agent. The red crosses represent the centroids of the innermost grids.

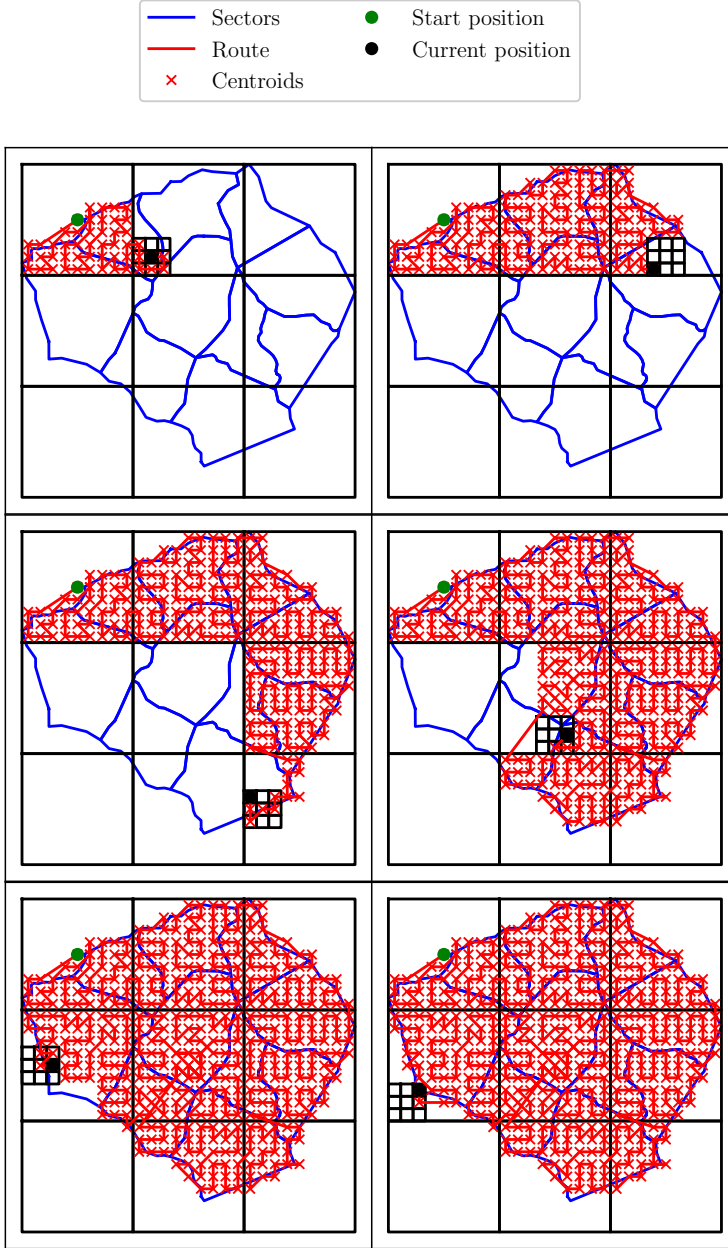


Figure 4.2: Agent trajectory for Set 2 when initially placed in the centroid of the cell located in the upper left of the outermost grid. The sectors in the search area are represented by the blue lines. The red lines represent the path taken by the agent, while the green and black dot represent the start-respectively current position of the agent. The red crosses represent the centroids of the innermost grids.

Based on the initial probabilities of the sectors, presented in Figure 3.4, the trajectory in Figure 4.1 prioritizes areas with higher probability compared to the trajectory in Figure 4.2. Generally, the method of hierarchically dividing the search area into 3 by 3 grids, thus avoiding huge state spaces, generates good results within a reasonable processing time. The results also demonstrate the drawbacks with the hierarchical solution. As Figure 4.1 shows, the large cell in the bottom left corners is given last priority. This is logical considering the fact that the cell only encapsulates a small fraction of the search area, with low initial probability. However, this leads to the cell being searched last in order, which creates a detour, since the cell relatively quickly could have been ticked off earlier when the agent naturally passed it. A possible solution to this problem would be to take the area of the sectors, that the cells in the outermost grid encapsulates, into consideration in the MDP. It can be noted that no such problem is present in Figure 4.2 which, in this case, is a result of not considering the probabilistic map.

Simulations were performed to further evaluate the trajectories from Set 1 and Set 2. The initial sector probabilities were used to place a missing person in a random position within a sector. A higher sector probability meant an increased probability of the missing person being located within the sector. The agent then travelled the routes for the precalculated two sets of different trajectories. An RSS-signal was deemed to be located when the distance between the agent and the missing person was less or equal to 200 meters. 10 000 simulations were performed for each trajectory from the two respective sets and the average travelled distance until detection (ATDD), for the agent was calculated. The results from the simulations are compiled in Table 4.1.

Table 4.1: Simulation results for the trajectories in Set 1 and Set 2. ATDD denotes the average travelled distance until detection. 10 000 simulations were performed for each starting cell.

Start cell	Set 1: ATDD [km]	Set 2: ATDD [km]
Upper left	32.27	44.42
Upper center	31.77	39.64
Upper right	32.81	38.21
Middle left	32.17	52.42
Middle center	31.60	31.73
Middle right	32.30	32.91
Lower left	32.52	53.21
Lower center	32.30	38.06
Lower right	33.29	35.69

Table 4.1 shows that the starting cell has a relatively small impact on the ATDD, for the trajectories in Set 1. The same is not true for the trajectories in Set 2, where the starting cell has a large impact on ATDD. Even though the trajectories in Set 1 always outperform the trajectories in Set 2, the differences are relatively small for certain starting cells, i.e., middle center and middle right. These cells con-

tain large parts of the sectors within the search area, that have high probabilities. Thus, it can be concluded that the initial probabilistic map only offers minor benefits if the agent starts its route within an area with high probability. All of the simulations were performed with precalculated trajectories for one type of search area, where a few neighboring sectors holds a large proportion of the probability. It is possible that a different type of search area would generate different results. An alternative would be to generate new probabilities for the sectors for each simulation, but due to the processing time (around five minutes) it thus creates a tradeoff with the total number of simulations that can be performed.

4.2 DQN training

Figure 4.3 shows the training results for two separate DQN models, one trained using the PMF and one using the PF. The models were trained on 4 million timesteps, and the performance is evaluated by the mean episode length and the mean episode reward.

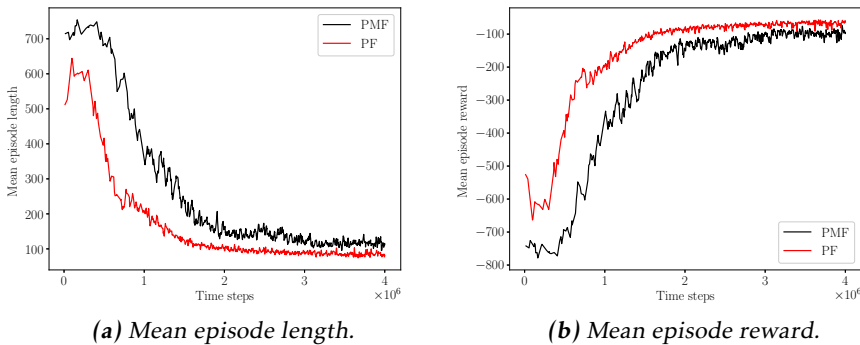


Figure 4.3: Episode length and reward for 4M training episodes. The model trained with PMF is represented by the black line, and the red line represents the PF model.

Both models converged at around 2 million time steps, indicating that they have learned to act efficiently in the environment. The models behaved similarly, however, the PF-DQN performed slightly better, converging towards a higher reward and lower episode length, with fewer timesteps. The relation between the episode length and the reward is reasonable, since the model receives a negative reward of -1 at each step and a positive reward of 1 if the probability is isolated. Figure 4.3 indicates that the models have learned to isolate the probability using approximately 100 steps.

It can be observed that, at the start of the training, the mean episode length and mean episode reward is lower respectively higher for the PF-DQN, compared to the PMF-DQN. This behavior has been visible in several other models trained

during the thesis. At the start of the training, actions are almost exclusively sampled randomly, as described in Section 3.4.3. It can therefore be assumed that, by almost exclusively random actions, the PF-DQN is more successful at isolating the probability and achieving the second stop criterion (KL-divergence), described in Section 3.4.2. Further testing verified the same result and the reason is probably due to a higher concentration of weights within a smaller area, which can be attributed to the resample step included in the PF-DQN. Figure 4.3 demonstrates that the difference in episode-length and reward becomes smaller towards the end of the training, when actions are less random and more often sampled by the models.

As described in Section 2.1, the PMF and PF have similar structures, which explained the similarity in performance. However, the PF includes the resampling step which makes the filter more computationally heavy, resulting in a longer processing time compared to the PMF.

4.3 State estimator comparison

The results of the filter evaluation are shown in Figure 4.4 and 4.5, where the average estimation error for 500 simulations using both the PMF-DQN and PF-DQN is displayed. Each simulation is initiated by randomly sampling the agent's and missing person's location. The initial belief regarding the missing person's location within the grid is uniform, thus all cells have an equal initial probability. The KL-divergence used as stop criterion in the training is deactivated, meaning that each simulation is run until the maximum allowed steps are reached.

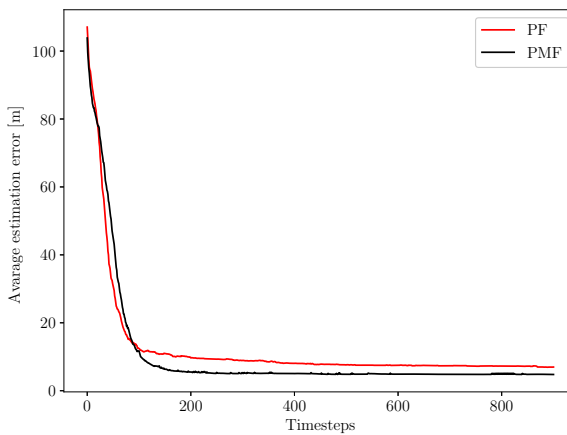


Figure 4.4: Average estimation error for PF-DQN (red) and PMF-DQN (black).

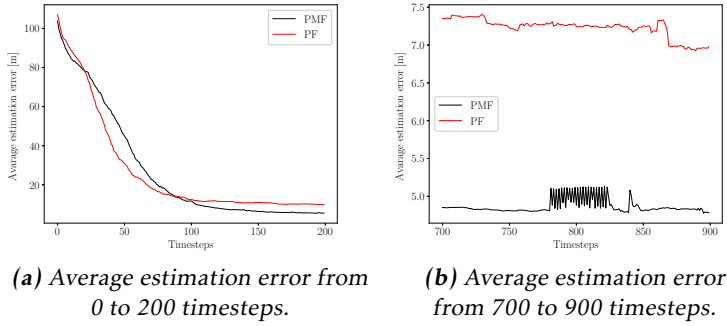


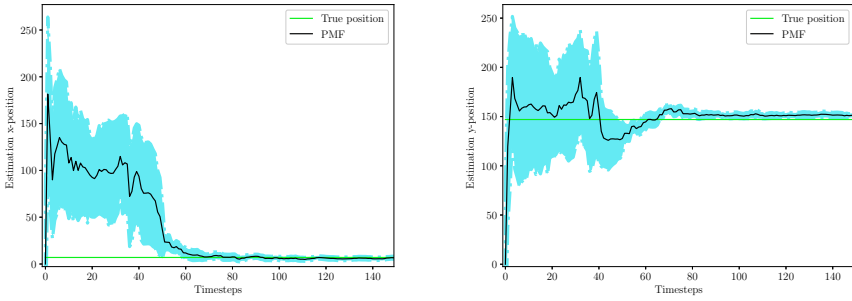
Figure 4.5: Average estimation error in intervals for PF-DQN (red) and PMF-DQN (black).

Figure 4.5a demonstrates that the PF initially performs better than the PMF, estimating the missing person's location with higher accuracy during the approximately first 95 timesteps. Thereafter, the estimate for the PMF keeps on improving while the estimate for the PF converges. The final average estimation error for both models can be seen in Figure 4.5b, where the PMF archives 4.8 meters compared to the PF 6.9 meters in the final timestep.

Table 4.2: Filter evaluation results.

Average estimation error at	PMF	PF
50 steps [m]	45.0	30.1
100 steps [m]	11.7	12.5
200 steps [m]	5.5	9.7
400 steps [m]	5.0	8.1
800 steps [m]	4.9	7.2
900 steps [m]	4.8	6.9

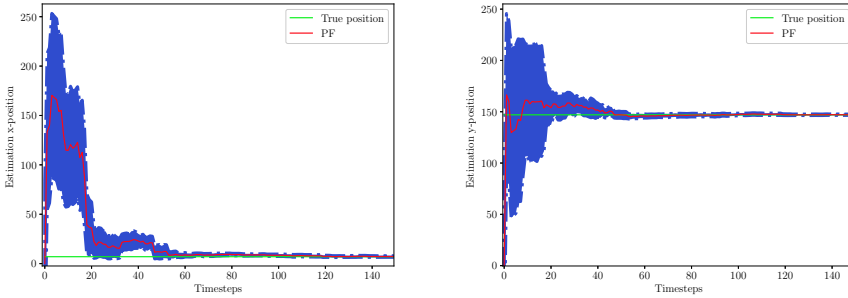
One standard deviation of each models estimation along the x-and y-position, during a single simulation, is displayed in Figure 4.6 and 4.7. The simulations were run separately but with the same initial configuration and policy. The simulations were 900 timesteps long, but only 150 are displayed, since both models converge at around 100 timesteps.



(a) PMF estimates along x-position with 1 standard deviation.

(b) PMF estimates along y-position with 1 standard deviation.

Figure 4.6: The estimates (black) from 0 to 150 timesteps presented with 1 standard deviation (blue) during a simulation with the PMF-DQN. True location is represented as a green line.



(a) PF estimates along x-position with 1 standard deviation.

(b) PF estimates along y-position with 1 standard deviation.

Figure 4.7: The estimates (red) from 0 to 150 timesteps presented with 1 standard deviation (purple) during a simulation with the PF-DQN. True location is represented as a green line.

As Figure 4.7 and 4.6 show, both models converge toward the ground truth in under 100 timesteps. The PF shows the best result and continuously outperforms the PMF, in regard to estimating the missing person's location, at the majority of the presented timesteps. The difference in performance could be attributed to the dynamic grid in the PF, which becomes "finer" when the probability becomes concentrated within an area. This is illustrated in Table 4.3, where the standard deviation for the PF is lower than the standard deviation for the PMF, for all presented timesteps.

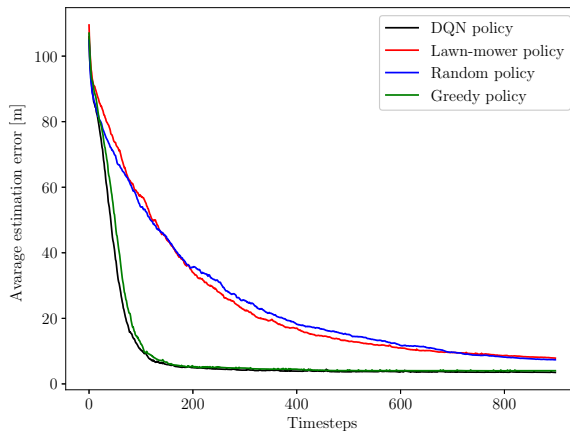
Table 4.3: Filter evaluation results.

1 standard deviation at	PMF	PF
25 steps	50.8	12.5
50 steps	24.2	4.9
100 steps	2.7	1.2
200 steps	1.2	0.3
400 steps	0.9	0.2
800 steps	0.6	0.01

Along with the result and discussion above, the PMF has been selected as the state estimator. The decision is partly based on the similarity in performance, demonstrating that both filters show good abilities in the requested application, thereby making the decision come down to the complexity of the filters. The PF has the additional resampling step which seems to result in a lower standard deviation, but the PMFs estimation has proven to be better over time, which makes the more computationally demanding PF unnecessary. The PMF is therefore selected to be used as the state estimator.

4.4 Path planning

The result of the path planning evaluation in Figure 4.8 shows average estimation errors of the DQN, greedy, random and lawn-mower policies over 500 simulations, all using the PMF. The agent is initially positioned in the same location each simulation while the missing person's position is randomly sampled.

**Figure 4.8:** Average estimation error for all 4 path planners.

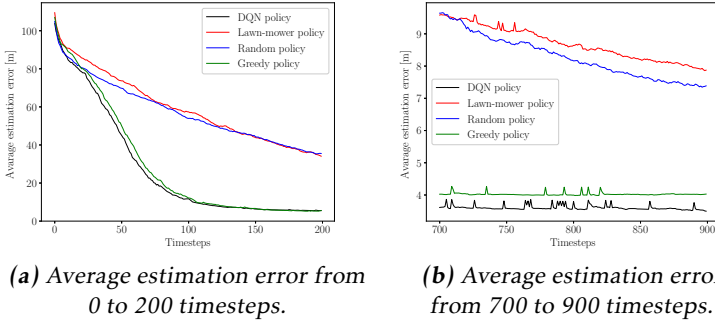


Figure 4.9: Average estimation error for all 4 path planner in different intervals.

The results in Figure 4.8 show that the policies form two different groups. The greedy and DQN policies perform well with similar behavior throughout the simulations. The lawn-mower and random policies perform alike but worse than the greedy and DQN policy. The lawn-mower and random policies average estimation error decreases continuously compared to the DQN and greedy policy, whose estimation error seems to converge at an average of 200 timesteps. It can be seen in Figure 4.9b that the average estimation error of the lawn-mower and random policies still decreases. The similar results of the random and lawn-mower policies are reasonable since the state estimator can receive measurements from any location within the grid. This implies that the state estimation should be able to improve its belief state as long as the agent moves within the environment, which is achieved in both policies. However, as explained in Figure 3.2, the spread in RSS-measurements increases when the distance between the receiver and emitter increases. This could explain the less favorable results, since neither the lawn-mower or random policy selected their trajectories based on the belief state, hence not always moving efficiently in regard to the missing person's location.

The similarity between the DQN- and greedy policy is not as obvious. The actions of the greedy policy are always based on the current state, while the DQN uses previous experience, from millions of timesteps, to select actions. The DQN model has through training learned that the missing person is most likely located in the area with the highest probability, but not always. This can be compared to the much more aggressive greedy policy which always explores the grid cell with the highest probability regardless of the remaining probability distribution. There is a difference between these two approaches, but how much it affects the result is hard to tell. However, the DQN is trained on isolating the probability with KL-divergence as a stop criterion, resulting in a different movement pattern compared to the greedy policy which could explain the marginally better performance.

Although the performance of the DQN is marginally better than the greedy

policy, the greedy policy is significantly more simple than the DQN. The greedy policy consist of less than 100 lines of code compared to the thousands included in the DQN. That, as well as the fact that the DQN needs to be trained on millions of timesteps to perform similarly to the simple greedy policy, strongly indicates that the DQN is a complex solution to a simple problem. However, the performance of the DQN is strongly connected to the reward function, which have been iterated during the thesis. It should be noted that there could be a more suitable reward function which was not explored in this thesis.

Table 4.4: Mean estimation error for all 4 path planners at 5 different timesteps.

Mean estimation error at	DQN	Greedy	Lawn-mower	Random
100 steps	10.3	12.1	57.4	53.9
300 steps	4.2	4.8	22.4	25.4
500 steps	3.7	4.1	12.96	15.0
700 steps	3.6	4.0	9.6	9.6
900 steps	3.5	4.0	7.9	7.4

4.5 Localization

The localization evaluation is a performance test of the PMF compared to Gauss-Newton and trilateration. Figure 4.10 shows the estimation error for a simulation where all estimators traveled along the same path. The agent's and missing person's initial location was the same for all estimators.

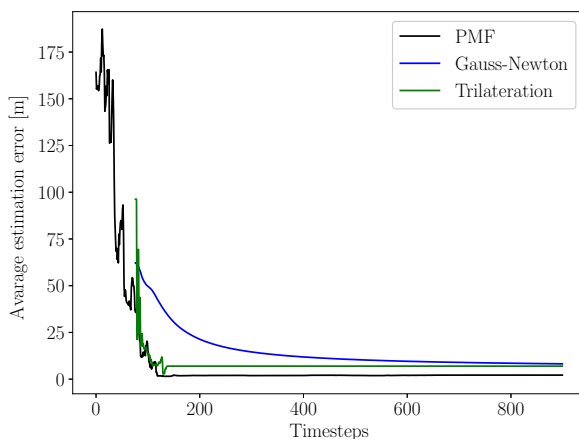


Figure 4.10: Estimation error for all three estimators.

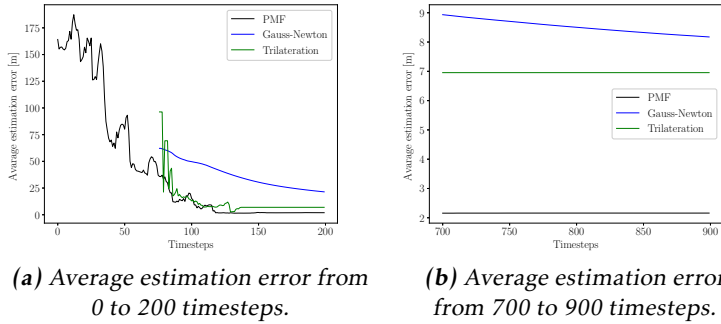


Figure 4.11: Average estimation error for all 3 estimators in different intervals.

The results, presented in Figure 4.10, show that the PMF performs better than Gauss-Newton and trilateration during the majority of the simulation. The Gauss-Newton and trilateration algorithm establish their first estimation after approximately 75 timesteps. The main reasons behind the missing estimations are the distance limit and measurement limit, which restricts the algorithms from making estimations until at least five estimated distances of less than 100 meters have been collected. These limits act much like a filter, preventing the algorithms from making very incorrect estimations due to noisy measurements.

As can be seen in Figure 4.11a, all localization systems manage to decrease their individual estimation errors, especially the PMF and trilateration. There could be several reasons for the less favorable behavior of the Gauss-Newton algorithm, but one feature that influences the performance largely is the initial guess. The initial guess, as explained in 3.5.2, is the estimation related to the highest RSS-measurement. This is a logical guess since a large RSS-signal indicates that the agent is close to the missing person, which should result in a good estimation. However, trusting a single measurement could be dangerous given the unstable nature of the RSS-signal. An alternative method, not explored in this thesis, which could improve the performance is using the trilateration estimations to initiate Gauss-Newton.

Table 4.5: Filter evaluation results.

Average estimation error at	PMF	Gauss-newton	Trilateration
50 steps [m]	80.0	NaN	NaN
100 steps [m]	15.3	49.7	13.0
200 steps [m]	1.9	21.3	7.0
400 steps [m]	1.9	11.9	7.0
800 steps [m]	2.2	8.5	7.0
900 steps [m]	2.2	8.1	7.0

4.6 DQN-PMF simulation

Figure 4.12 presents a sequence of belief states gathered during a single simulation using the DQN-PMF. The first belief state is shown in Figure 4.12a where the probability, visible in the bar below the figure, ranges between very low values. This demonstrates that no clear estimate of the missing person's location has yet been established. The following two figures (b and c) illustrate how the probability, of the cells located in the area of the missing person, increases and how the agent moves towards the person. The last three figures (d, e and f) demonstrates how the agent moves around the missing person, trying to isolate the probability to the fullest extent.

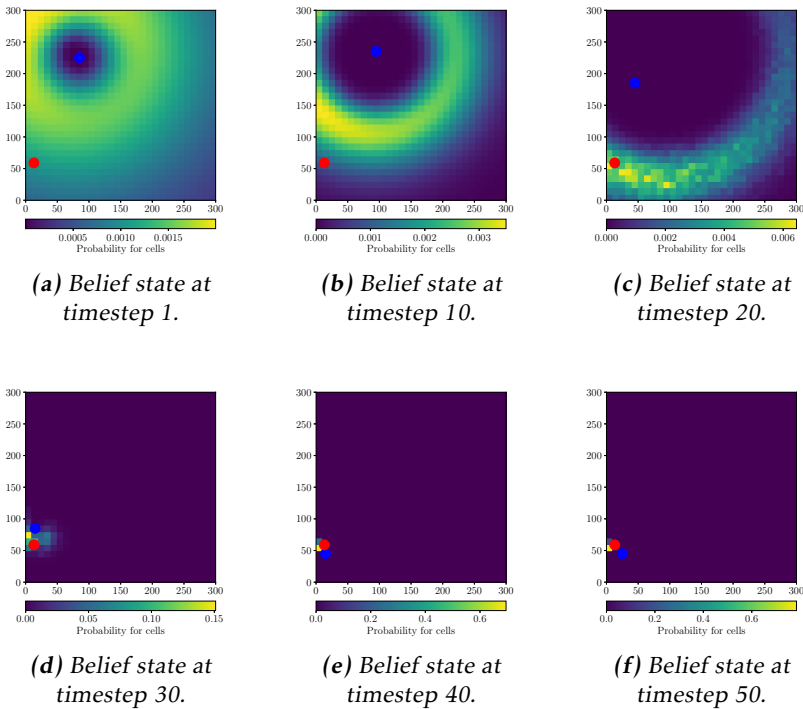


Figure 4.12: The blue dot represents the agent's position, while the red dot represent the position of the missing person.

The resulting movement made by the agent corresponds to the intuitive solution to the problem. The state estimation is able to update the belief state close to the true location of the missing person, making it feasible for the agent to move in a target-oriented manner towards the area with the highest probability. When reaching the missing person, the agent moves over and around the identified location of interest until the stop criterion is reached, which is a reasonable strategy.

5

Conclusions and future work

5.1 Conclusions

This thesis has, through Stage 1, evaluated how a probabilistic map can be utilized to create a path planner capable of efficiently locating an RSS-signal. Formulating the problem as a Markov decision process generated generally satisfying results and also showed promise due to its relatively low processing time. The method allowed multiple factors to be taken into consideration, such as distance, probability and potentially area. However, further tests are required, with different type of SAR environments and alternative path planning methods, to determine the effectiveness and adaptability of the method.

This thesis has also investigated how RSS-measurements effectively can be used to establish a UAV path planner that isolates a missing person's probable location. Simulation results demonstrate that formulating the problem as a belief MDP and letting the action-policy depend on the current belief, could be an efficient approach. Two different filters were evaluated and although the PMF was regarded as the preferred choice, the PF could also be considered a good alternative. Both filters showed good abilities in estimating the belief state of the belief MDP, which gave the DQN a good perception of the environment to act upon. The characteristics of the PMF and PF proved to be suitable for the thesis problem, which lead to a simple implementation and usage.

The fact that a simple greedy-policy generated similar results to the DQN indicates that the DQN may be a far too complicated solution to a less complicated problem. However, based on experience from this thesis, the behavior of the DQN is dependent on the reward function and although multiple different reward functions were evaluated, it is possible that it exists one that leads to better results.

It is important to point out that all results from this thesis are based solely on simulations. In a real-life outdoor scenario, multiple factors exist that could affect the results, such as sparse RSS-measurements and terrain. Another important factor is that this thesis is based on the assumption that the missing person wears an enabled smartphone. In a real-life scenario, this is far from guaranteed, which affects the usability in real SAR operations.

5.2 Future work

There are several extensions that could be made to this thesis. The following sections describe some of our suggestions.

Stage 1 extensions

Extending Stage 1 to consider factors such as area and terrain would possibly create a more efficient path planner. By considering area, the probability of unnecessary detours could be reduced and since terrain has a great impact on the possible location, of a missing person, it would thus make the path planner more suitable for use in real-life SAR operations.

Field tests

A natural extension to this thesis is real-life field tests to demonstrate that the localization systems works as expected. Even though uncertainty in RSS-measurements has been taken into consideration in this thesis, the real-life implementation is far from normally distributed, and reality often comes with many unwanted features.

Non-stationary missing person

Another interesting extension is to consider the scenario where the missing person is non-stationary. The two state estimators both consider dynamics in their standard implementation, which should make it a natural extension.

Extending the sensor model

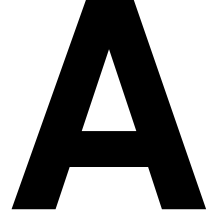
An interesting extension that could improve the real-world implementation of the system is to extend the sensor model to include other sources of information than only the RSS-measurement. Heat camera images or GPS-signals are two examples which could be used as input to the existing state estimator through relatively small adjustments. Enabling different sensor input and extending the state estimator to use several sources of information at once would most likely improve the belief state of the environment.

Uncertainty in measurement

Uncertainty in sensor measurement can become a problem when the frequency of the source is low and velocity of the agent is high. This thesis assumes that a measurement is received each timestep which is of course not always the case.

The system would have to take the uncertainty in mind to not act too aggressively on the current belief, which could be the case if the existing system would be faced with less frequent measurements. By implementing an uncertainty into the system or by having knowledge about the frequency of the signal source, this problem could be solved.

Appendix



Particle filter resampling

There exists multiple different resampling techniques. The following options are the most common in PF literature. All of the algorithms are unbiased, but have different computational complexities [21].

1. Multinomial resampling

Generate N ordered uniform random numbers.

$$u_k = u_{k+1} + \tilde{u}_k^{\frac{1}{k}}, \quad u_N = \tilde{u}_N^{\frac{1}{N}}, \quad \text{with } \tilde{u}_k \sim U(0, 1)$$

Use them to select x_k^* according to the multinomial distribution described below

$$x_k^* = x(F^{-1}(u_k)) = x_i \text{ with } i \text{ s.t. } u_k \in \left[\sum_{s=1}^{i-1} w_s, \sum_{s=1}^i w_s \right]$$

where F^{-1} denotes the generalized inverse of the cumulative probability distribution of the normalized particle weights.

2. Stratified resampling

Generate N ordered random numbers.

$$u_k = \frac{(k-1) + \tilde{u}_k}{N}, \quad \text{with } \tilde{u}_k \sim U(0, 1)$$

Use them to select x_k^* according to the multinomial distribution.

3. Systematic resampling

Generate N ordered numbers.

$$u_k = \frac{(k-1) + u_k}{N}, \quad \text{with } u_k \sim U(0, 1)$$

Use them to select x_k^* according to the multinomial distribution.

4. Residual resampling

Allocate $n'_i = \lceil Nw_i \rceil$ copies of the particle x_i to the new distribution. Resample $m = N - \sum n'_i$ particles from x_i by making n''_i copies of the particle, x_i where the probability for selecting x_i is proportional to $w'_i = Nw_i - n'_i$ using one of the previously mentioned resampling schemes.

Bibliography

Bibliography

- [1] Swedish police, "Efterforskning av Försvunna Personer - polisens arbete," 2020. Last accessed 4 February 2022, <https://polisen.se/om-polisen/polisens-arbete/forsvunna-personer>.
- [2] D. Gilman, *Unmanned Aerial Vehicles in Humanitarian Response*. United Nations Office for the Coordination of Humanitarian Affairs, 2014.
- [3] Swedish police, "Äldre kvinna hittad med hjälp av drönare i Glanshammar," 2020. Last accessed 15 May 2022, <https://polisen.se/aktuellt/nyheter/2020/september/effektiv-raddningsaktion-vid-forsvinnande--aldre-kvinna-hittad-med-hjalp-av-dronare>.
- [4] G. Han, J. Jiang, C. Zhang, T. Q. Duong, M. Guizani, and G. K. Karagianidis, "A Survey on Mobile Anchor Node Assisted Localization in Wireless Sensor Networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2220–2243, 2016.
- [5] W. Alshrafi, U. Engel, and T. Bertuch, "Compact Controlled Reception Pattern Antenna for Interference Mitigation Tasks of Global Navigation Satellite System Receivers," *IET Microwaves, Antennas & Propagation*, vol. 9, no. 6, pp. 593–601, 2015.
- [6] A. Zanella, "Best Practice in RSS Measurements and Ranging," *IEEE Communications Surveys Tutorials*, vol. 18, pp. 2662 – 2686, 2016.
- [7] A. Al-Hourani, S. Kandeepan, and A. Jamalipour, "Modeling Air-to-Ground Path Loss for Low Altitude Platforms in Urban Environments," in *2014 IEEE global communications conference*, pp. 2898–2904, IEEE, 2014.
- [8] J. Sundqvist, J. Ekskog, B. Dil, F. Gustafsson, J. Tordenlid, and M. Petterstedt, "Feasibility Study on Smartphone Localization using Mobile Anchors in Search and Rescue Operations," *Proceedings of 19th International Conference on Sensor Fusion*, pp. 1448 – 1453, 2016.

- [9] S. police, *Polismyndighetens Handbok för Efterforskning av Försvunna Personer*. Polisen, 2016.
- [10] T. Stoyanova, F. Kerasiotis, C. P. Antonopoulos, and G. D. Papadopoulos, "rss,"
- [11] D. Koutsonikolas, S. M. Das, and Y. C. Hu, "Path Planning of Mobile Landmarks for Localization in Wireless Sensor Networks," *Comput. Commun.*, vol. 30, no. 13, pp. 2577 – 2592, 2007.
- [12] D. Koutsonikolas, S. M. Das, and Y. C. Hu, "Superior Path Planning Mechanism for Mobile Beacon-Assisted Localization in Wireless Sensor Networks," *IEEE Sensors Journal*, vol. 14, pp. 3052 – 3064, 2014.
- [13] D. Ebrahimi, S. Sharafeddine, P.-H. Ho, and C. Assi, "Autonomous UAV Trajectory for Localizing Ground Objects: A Reinforcement Learning Approach," *IEEE Transactions on Mobile Computing*, vol. 20, pp. 1312 – 1324, 2021.
- [14] F. Demiane, S. Sharafeddine, and O. Farhat, "An Optimized UAV Trajectory Planning for Localization in Disaster Scenarios," *Computer Networks*, vol. 179, 2020.
- [15] P. Perazzo, F. Sorbelli, M. Conti, and C. Dini, G and Pinotti, "Drone Path Planning for Secure Positioning and Secure Position Verification," *IEEE Transactions on Mobile Computing*, vol. 16, pp. 2478 – 2493, 2017.
- [16] H. Bayerlein, M. Theile, M. Caccamo, and D. Gesbert, "Multi-uav Path Planning for Wireless Data Harvesting with Deep Reinforcement Learning," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 1171 – 1187, 2021.
- [17] A. R. Cassandra, *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. Brown University, 1998.
- [18] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level Control Through Deep Reinforcement Learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [19] M. Svorenová, M. Chmelik, K. Leahy, H. F. Eniser, K. Chatterjee, I. Cerná, and C. Belta, "Temporal Logic Motion Planning using POMDPs with Parity Objectives: case study paper," in *HSCC*, pp. 233 – 238, ACM, 2015.
- [20] S. Obadan and Z. Wang, "A MULTI-AGENT APPROACH TO POMDPS USING OFF-POLICY REINFORCEMENT LEARNING AND GENETIC ALGORITHMS," *International Journal of Computing*, vol. 19, pp. 377 – 386, 2020.
- [21] F. Gustafsson, *Statistical Sensor Fusion*. Studentlitteratur AB, 2018.

- [22] G. Lui, T. Gallagher, B. Li, A. Dempster, and C. Rizos, "Differences in RSSI Readings made by Different Wi-Fi Chipsets: A Limitation of WLAN Localization," *2011 International Conference on Localization and GNSS (ICL-GNSS)*, p. 53–57, 2011.
- [23] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 2018.
- [24] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and Acting in Partially Observable Stochastic Domains," *Artificial Intelligence*, vol. 101, pp. 99 – 134, 1998.
- [25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [26] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai Gym," 2016. cite arxiv:1606.01540.
- [27] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv preprint arXiv:1312.5602*, 2013.