Exploration of Reinforcement Learning in Radar Scheduling

An evaluation of the PPO algorithm on a radar scheduling task

Master's thesis in Engineering Mathematics and Computational Science

AXEL NATHANSON

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Exploration of Reinforcement Learning in Radar Scheduling

An evaluation of the PPO algorithm on a radar scheduling task

Axel Nathanson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2021

Exploration of Reinforcement Learning in Radar Scheduling
An evaluation of the PPO algorithm on a radar scheduling task
Axel Nathanson

Supervisor: Adam Andersson, Saab and Department of Mathematical Sciences
Examiner: Stig Larsson, Department of Mathematical Sciences

Master's Thesis 2021
Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone: +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2021

Exploration of Reinforcement Learning in Radar Scheduling
An evaluation of the PPO algorithm on a radar scheduling task
Axel Nathanson
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

The development of phased array antennas has enhanced the effectiveness of radars thanks to it's flexibility allowing the radar beam to be controlled and adapted almost instantaneously. This flexibility allows a radar to carry out multiple tasks simultaneously, such as surveillance of an area and tracking of targets. Traditionally the scheduling is performed according to hard-coded priority lists in combination with local optimisation, rather than global mathematical optimisation.

Reinforcement learning algorithms have in the last few years successfully solved several artificial control tasks and is slowly starting to show some successes in real-world scenarios. Encouraged by the success we study the application of the Proximal Policy Optimisation (PPO) algorithm on a radar scheduling task.

The algorithm is trained to track targets and search for new ones within a surveillance area. The proposed algorithm did not solve the scheduling task, but we identify and formalise the challenges that need to be addressed to be able to solve the radar scheduling task with the PPO algorithm.

Keywords: radar scheduling, reinforcement learning, PPO, machine learning.

Acknowledgements

First and foremost, I want to thank my supervisor Adam Andersson for the continuous discussions, support and feedback every step of the way. Furthermore, I want to thank Saab for providing me with the opportunity to work on this project. Finally, I want to thank everyone at Saab that I have had the pleasure of working with for their enthusiasm and support during the project.

Axel Nathanson, Gothenburg, June 2021

Table of Contents

1. Introduction

The development of phased array antennas has enhanced the effectiveness of radars thanks to its flexibility allowing the radar beam to be controlled and adapted almost instantaneously. This flexibility allows a radar to carry out multiple tasks simultaneously, such as surveillance of an area and tracking of targets.

Traditionally the scheduling is performed according to hard-coded priority lists in combination with local optimisation, rather than global mathematical optimisation. We, therefore, look to the recent advances in *Reinforcement Learning* (RL) to tackle this problem. RL is the machine learning discipline of learning by interacting with and exploring an environment to maximise a reward. This approach has been successful in solving complicated tasks where the desired outcome is easier to define than the means of achieving it. Most famously have RL algorithms been used to beat old Atari games [1], master the game of Go [2] and beat the best players in the world at the game of Dota [3].

The field of RL has developed gradually over the last decades into one of the most active research areas in machine learning. One of the breakthroughs that sped up the development is what sometimes is referred to as the "Deep Learning Revolution"[1], and then its translation to *Deep Reinforcement Learning* (DRL) [5]. Even though DRL research is performed in several closely related fields like robotics and control theory, limited research has been done on radar applications.

1.1 Objective

In this thesis, we study the possibility of using modern deep reinforcement learning algorithms to optimise the management of radar resources while performing surveillance and tracking of targets within a fixed area. We investigate the performance of the *Proximal Policy Optimisation* (PPO) algorithm on a simulated tracking scenario. As part of the thesis, simulations of a tracking scenario and a rule-based baseline agent are presented. The performance of the PPO algorithm on the simulated scenario is evaluated in comparison with the rule-based baseline.

1.2 Related Work

The PPO algorithm was proposed by Schulman et al. [6] and has been used in several different applications since, like solving a Rubik's cube with a robotic hand [7], in

[1] People usually attest the start of the "revolution" to Alex-Net in 2012 [4], even if deep architectures trained on GPU:s had been implemented earlier.

the designing of computer chips [8] and even beating professional players in complex games like Dota [3]. Popularity of the PPO algorithm comes from the fact that the introduced regularisation on the size of each update leads to more stable training updates.

In the radar field, solving scheduling tasks has historically been done with rule-based priority lists but the increased performance of machine learning methods in closely related fields suggests that there might be a potential for further studies. In previous studies, radar resource management problems have been solved with combinatorial optimisation methods combined with neural networks [9] and the radar scheduling task has been solved with reinforcement learning versions of the *Earliest starting time* algorithm [10]. Modern model-free methods have not been studied to the same extent but have been studied in related fields like control theory.

1.3 Thesis Outline

Chapter 2 outlines and explains the theoretical foundations of radar, tracking, deep learning and reinforcement learning. This is followed by the derivation of the PPO algorithm and real-world challenges of reinforcement learning.

In Chapter 3, the methods used are presented. The designed target simulation, tracking algorithm and rule-based baseline agent are presented. This is followed by our implementation of the PPO algorithm and how it is trained.

In Chapter 4, the performance of the two agents are evaluated and the decision making of the PPO agent is analysed. This is followed by conclusions and suggestion for future work in Chapter 5.

2. Theory

This chapter introduces important concepts used in this thesis. We cover key concepts in radar technology in Section 2.1, target tracking in Section 2.2, deep learning in Section 2.3 and reinforcement learning in Section 2.4.

2.1 Radar Basics

RAdio Detection And Ranging (RADAR/radar) is the technique of using radio waves to determine characteristics of targets, such as their position and velocity. By sending out radio waves and measuring the echoes, the distances to targets are calculated by the time the echoes take to return (pulse-delay ranging), and their velocities from the frequency shift of the radio wave due to Doppler shift [11]. Radars can generally be divided into those that send continuous waves and those that send pulses. The continuous radar both sends a signal and listens simultaneously, while the pulse radar sends several pulses and listens for the echoes in between the transmissions. We are focusing on airborne radars, which usually are pulse radars capable of measuring the Doppler shift of the transmitted signals, so called *pulse Doppler radar* [11].

When using airborne pulse radars the same antenna is usually used for transmitting and listening to the echoes. When observing a target at distance r the radar range equation describes the power reflected from a target. It is given by

$$P_{\mathrm{R}} = \frac{P_{\mathrm{T}} G A C F^4}{(4\pi)^2 r^4}, \tag{2.1}$$

where P_{T}, P_{R} are the powers leaving the transmitter and returning to the receiver, respectively. G is the gain of the transmitting antenna and A the effective aperture area of the receiving antenna. C is the radar cross section of the target which describes the target's ability to reflect the radar signal. Lastly, F is the pattern propagation constant, which describes how atmospheric conditions affect the propagation of the signal. These are all important characteristics when constructing a radar system; for a more thorough analysis of each parameter, see [11].

When making measurements of targets the radar system sends multiple pulses at a high frequency. This frequency is the Pulse Repetition Frequency (PRF), i.e., the number of pulses sent per second. Measuring the range of a target by pulse-delay can both be simple and extremely accurate. However, when sending multiple pulses there is no direct way of knowing which pulse the received echo originates

from, resulting in ambiguous measurements. We can solve this by *resolving* the measurements, which is a method of varying the PRF in consecutive measurements to remove the ambiguity; for a deeper explanation of the exact technique, see [11, p. 215]. If the distance is known from previous measurements, the detection can be matched with that of previous detections and an unresolved radar mode can be used.

The accuracy of a measurement can be increased by increasing the number of pulses sent in each individual measurement, however, this will increase the time each measurement takes to perform. Choosing the number of pulses is usually referred to as choosing the *waveform* of the measurement.

When measuring the speed of a target with a pulse Doppler radar, as the name suggests, the Doppler effect is used. By measuring the difference in frequency between the sent signal and the returning one a targets velocity relative to the radar is estimated by the size of the shift [11]. The Doppler frequency shift can be formulated as

$$f_{\mathrm{D}} = -2\frac{v_{\mathrm{radial}}}{\lambda}$$

where v_{radial} is the radial velocity of the target relative to the radar and λ is the wavelength of the radio wave. With a ground based radar system any relative motion is essentially caused by target movement. When our system moves, the ground also appears to have a velocity relative to the radar and gives rise to a range of Doppler shifts. Together with thermal noise entering the amplifier stage and indirect reflections, this give rise to false detections. In the signal processing a detection threshold is set that keeps the false detection probability essentially constant.

2.2 Target Tracking

The obtained radar observations are processed to obtain the radial distance, radial velocity and bearing of the target relative to the radar antenna. But since the observations are noisy we need to utilise filtering algorithms to estimate the states of the targets. This is referred to as target tracking and is naturally performed with Bayesian filtering algorithms. We refer to an estimation of a target as a *track*. When we have multiple targets moving in the area it also adds the challenge of estimating the number of unique targets and how to associate the observations to each track [12]. Here we present a simple tracking algorithm in four steps listed in Algorithm 1 inspired by the Global Nearest Neighbour algorithm [13]. This tracking algorithm was chosen for its simplicity since this studies focus not is on the tracking, but modern target tracking would instead focus on the two paradigms, *Multi Hypothesis Tracking* [14] and *Random Finite Set* algorithms [15].

The basic idea of the algorithm is to use a Extended Kalman filter, presented in Sec-

Algorithm 1: Tracking Algorithm

1: Associate measurement to existing tracks, **Hungarian method**.
2: Update existing tracks, **Extended Kalman Filter**.
3: Terminate tracks that fulfil termination criteria.
4: Initiate new tracks for unassociated measurements.

tion 2.2.1, to estimate the position of each target being tracked and match new observations to the existing tracks using the Hungarian matching algorithm, presented in Section 2.2.2. The termination criteria are presented later in Section 3.1.3.3.

2.2.1 Bayesian Filtering

When discussing filtering the term *optimal filter* is sometimes used interchangeably with Bayesian filtering. This comes from the fact that estimating the state with regard to the mean square error involves computing a posterior with Bayes formula, explaining the name Bayesian filtering [16]. If we assume that the model noise is additive, we can write the model of the state x_k and observation z_k at time k as

$$\begin{aligned} x_k &= f(x_{k-1}) + q_{k-1}, \\ z_k &= h(x_k) + r_k, \end{aligned} \tag{2.2}$$

where $q_{k-1} \sim \mathcal{N}(0, Q_{k-1})$ is the process noise, $r_k \sim \mathcal{N}(0, R_k)$ the measurement noise, f the dynamic function and h the measurement function and $x_0 \sim \mathcal{N}(m_0, P_0)$ the initial value. The purpose of Bayesian filtering is to calculate the posterior distribution of the state x_k at time k given the history of measurements $z_{1:k}$ up until and including the time k

$$p(x_k \mid z_{1:k}).$$

The fundamentals of Bayesian Filtering can be expressed as a recursion. In Bayesian filtering the predicted distribution $p(x_k \mid z_{1:k-1})$ and the posterior distribution $p(x_k \mid z_{1:k})$ are computed by the recursion

- Initialization. The recursion starts from the prior distribution $p(x_0)$.
- Prediction step. The predicted distribution of the state x_k at time k, given a dynamical model, can be computed by the Chapman–Kolmogorov formula

$$p(x_k \mid z_{1:k-1}) = \int p(x_k \mid x_{k-1})\ p(x_{k-1} \mid z_{1:k-1})\ \mathrm{d}x_{k-1}.$$

- Update step. Given the measurement y_k at time k the posterior distribution of the state x_k can be computed by Bayes rule

$$p(x_k \mid z_{1:k}) = \frac{1}{Z_k} p(z_k \mid x_k)\ p(x_k \mid z_{1:k-1}),$$

with a normalizing term, Z_k, given by

$$Z_k = \int p(z_k \mid x_k)\ p(x_k \mid z_{1:k-1})\ \mathrm{d}x_k.$$

This holds for several dynamic models, among which are the linear models

$$x_{k+1} = A_{k-1}x_k + q_{k-1},$$
$$z_k = H_k x_k + r_k,$$

where the initial value and the noise terms are as in (2.2) and H_k and A_{k-1} are the transition and measurement models. A closed form solution to the Bayesian filtering recursion for this system can formulated in the form of the Kalman filter [17]

$$p(x_k \mid z_{1:k-1}) = \mathcal{N}(x_k \mid m_k^-, P_k^-),$$
$$p(x_k \mid z_{1:k}) = \mathcal{N}(x_k \mid m_k, P_k),$$
$$p(z_k \mid z_{1:k-1}) = \mathcal{N}(z_k \mid H_k m_k^-, S_k),$$

where

$$\mathcal{N}(x \mid m, P) = \frac{1}{(2\pi|P|)^{n/2}} \exp\left(\frac{(x-m)^T P^{-1}(x-m)}{2}\right).$$

The parameters can be computed with the following prediction and update steps:

- Prediction:

$$m_k^- = A_{k-1}m_{k-1},$$
$$P_k^- = A_{k-1}P_{k-1}A_{k-1}^T + Q_{k-1}.$$

- Update:

$$v_k = z_k - H_k m_k^-,$$
$$S_k = H_k P_k^- H_k^T + R_k,$$
$$K_k = P_k^- H_k^T S_k^{-1},$$
$$m_k = m_k^- + K_k v_k,$$
$$P_k = P_k^- - K_k S_k K_k^T.$$

In practical applications, however, it is common that the models are nonlinear and then the Kalman filter is not applicable. Instead, one can use the Extended Kalman filter (EKF), an extension of the Kalman filter to non-linear filtering problems, where the non-Gaussian, filtering distribution is approximated by a Gaussian distribution by using linearisation. The true distribution is non-Gaussian since the dynamics of the model is non-linear. The idea of the EKF is to assume Gaussian approximations to the filtering densities by utilising a first order Taylor approximation of the non-linearities around the latest prediction.

If the model noise is assumed to be additive the EKF model can be written as (2.2) with the difference being that h and f do not need to be linear functions but must be differentiable. The filtering density can then be approximated as

$$p(x_k \mid z_{1:k}) \approx \mathcal{N}(x_k \mid m_k, P_k).$$

Similarly to the Kalman filter, the prediction and update equations are stated as

- Prediction:

$$\begin{aligned} m_k^- &= f(x_{k-1}), \\ P_k^- &= \mathcal{J}_f(m_{k-1}) P_{k-1} \mathcal{J}_f(m_{k-1})^T + Q_{k-1}. \end{aligned}$$

- Update:

$$\begin{aligned} v_k &= z_k - h(m_k^-), \\ S_k &= \mathcal{J}_h(m_k^-) P_k^- \mathcal{J}_h(m_k^-)^T + R_k, \\ K_k &= P_k^- \mathcal{J}_h(m_k^-)^T S_k^{-1}, \\ m_k &= m_k^- + K_k v_k, \\ P_k &= P_k^- - K_k S_k K_k^T. \end{aligned} \tag{2.3}$$

where $\mathcal{J}_h(m_k^-)$ and $\mathcal{J}_f(m_k^-)$ are the Jacobians of f and h with regard to the state x evaluated at the latest prediction m_k^- [12].

2.2.2 Hungarian Algorithm

The Hungarian method is an algorithm developed to solve assignment problems. The foundations of the algorithm comes from two Hungarian mathematicians, König and Egerváry, but Kuhn completed the algorithm and named it the Hungarian algorithm [18].

When associating observations to a track, a cost-function representing the distance from a observation to a track is needed. As a measure of the distance between each observation and the distributions estimated by the EKF-filters, the *Mahalanobis distance* is used. The distance can be interpreted as the number of standard deviations separating the observation and the mean of each distribution. It is given by

$$D(x, \mu, P)_{\text{Mahalanobis}} = \sqrt{(x-\mu)^T P^{-1} (x-\mu)},$$

where x is a observation, μ, P the mean and covariance matrix of the distribution we are measuring the distance to. In the tracking application these means and covariances correspond to the m_k and P_k in (2.3), estimated by the EKF-filters.

Given N tracks and M observations to be associated we create a cost-matrix $C \in \mathbb{R}^{M \times N}$. The element $C_{i,j}$ represents the cost of assigning measurement i to track number j, i.e. the Mahalanobis distance between observation i to track j. The algorithm is described at a high level in Algorithm 2. The algorithm utilises the idea that if a number is added to or subtracted from all of the entries of any row or column of the cost matrix, then an optimal assignment for the resulting matrix is also an optimal assignment for the original matrix. By adding and subtracting from

each row and column in a strategic way the assignment problem can be simplified.

Algorithm 2: Hungarian Algorithm

Pre-processing: Any cost exceeding the cost threshold is not eligible for association and removed from the matrix.

1: For each row, find the smallest element and subtract it from each element in that row, resulting in the smallest entry in each row equals 0.
2: For each column, find the smallest element and subtract it from each element in that column, resulting in the smallest entry in each row equals 0.
3: Draw the minimal number of horizontal and vertical lines on the matrix, covering every 0 entry.
4: If M lines are required, an optimal assignment exists among the zeros and the algorithm stops. If not, go to step 5.
5: Find the smallest entry not covered by any line. Subtract this entry from each row that is not covered by a line and add it to each column that covered and go back to Step 3.

Any measurements not assigned to any track after matching is done, is instead used to initiate new tracks.

2.2.3 Dynamic Model

To be able to model the movement of targets we need dynamics models. The targets are modelled with *constant velocity* and *constant turn rate* models.

The constant velocity model is a simple model to simulate and predict [19]. The state at time k is given by

$$\vec{x_k} = [x,\ \dot{x},\ y,\ \dot{y}]^T, \quad x, y \in \mathbb{R}.$$

Its transition dynamics reads

$$\vec{x}_{k+t} = \begin{bmatrix} 1 & t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & t \\ 0 & 0 & 0 & 1 \end{bmatrix} \vec{x_k} + \begin{bmatrix} t^2/2 & 0 \\ t & 0 \\ 0 & t^2/2 \\ 0 & t \end{bmatrix} \vec{w}_k, \tag{2.4}$$

where t is a time increment and $\vec{w}_k$ is the state noise acting on the acceleration. This model can be used both as a way of simulating the movement of targets in the area but also as a dynamic model for EKF, where multiplication by the transition matrix acts as $f(\cdot)$. We use the discrete versions because in our simulation different actions take different amounts of time. To simulate that the targets do not actually fly with a constant speed because of turbulence and other factors a small disturbance in their acceleration in each update is added.

The targets will, with a small probability, initiate a manoeuvre during the simulation and to model them a constant turn rate model is used. With a constant turn rate

of ϕ [radians/second] the movement can be modelled as

$$\vec{x}_{k+t} = \begin{bmatrix} 1 & \frac{\sin(\phi \cdot t)}{\phi} & 0 & \frac{-(1-\cos(\phi \cdot t))}{\phi} \\ 0 & \cos(\phi \cdot t) & 0 & -\sin(\phi \cdot t) \\ 0 & \frac{1-\cos(\phi \cdot t)}{\phi} & 1 & \frac{\sin(\phi \cdot t)}{\phi} \\ 0 & \sin(\phi \cdot t) & 0 & \cos(\phi \cdot t) \end{bmatrix} \vec{x}_k. \tag{2.5}$$

The state update is performed with one of the two models, never both.

2.3 Deep Learning Basics

The fields of machine learning (ML) and deep learning (DL) has gathered a lot of attention in recent years thanks in part to what sometimes is referred to as the "deep learning revolution". The increased availability of large datasets and computing power have made training complex models possible [20]. These deep learning models consist of several layers, which are flexible function approximators that can be utilised for an array of different tasks [21].

A *multilayer perceptron*-architecture (MLP) is a fully connected feed-forward network that consists of N layers of computational units. An MLP maps an input x_0 to an output x_N. The networks are parameterized by weight vector $\theta = (\theta_1^1, \theta_1^2, \theta_2^1, \theta_2^2, ..., \theta_N^1, \theta_N^2)$ where θ_i^1 are matrices and θ_i^2 are bias vectors. An MLP is defined as

$$x_i = \sigma_i(\theta_i^1 x_{i-1} + \theta_i^2), \quad i = 1, ..., N,$$

where σ is called an *activation functions*. Where the last layer either have no activation function or a special activation function for the output. The activation function is a non-linear function introduced to increase the complexity of the models. Examples of popular activation functions are

- *Rectified Linear Unit* (ReLU): $\sigma_{\text{ReLU}}(x) = \max\{0, x\}$,
- *Leaky ReLU*: $\sigma_{\text{Leaky ReLU}}(x) = \max\{0, x\} + \xi \min\{0, x\}$, where ξ is a hyperparameter,
- *Softmax*: $\sigma_{\text{Softmax}}(x) = \frac{(\exp(x_1), ..., \exp(x_N))}{\sum_{j=1}^{N} \exp(x_j)}$.

The ReLU-based activation functions are performed elementwise. Softmax is usually used as activation of the last layer of a neural network, called the output layer, rather than between hidden layers. This is useful when we want to interpret the output as a probability distribution since Softmax result in $\sigma_{\text{Softmax}}(x_i) \in [0, 1]$ and $\sum_{i=1}^{N} \sigma_{\text{Softmax}}(x_i) = 1$. When constructing activation functions both performance and computational efficiency is of importance.

The training objective is to update the parameters θ in order to maximise the objective function $J(\theta)$. Given a training set $\mathcal{D}$ and an MLP with learnable weights θ the objective is given by

$$J(\theta) = \mathbb{E}_{x\sim\mathcal{D}}\left[\ell(x_N)\right],$$

where ℓ is a scalar valued loss function. In Section 2.4.6 we specify the loss functions of relevance in this thesis. The optimisation is regularly performed with the *gradient descent* algorithm, defined as

$$\theta_t = \theta_{t-1} - \alpha\nabla_\theta J(\theta_{t-1};\ \mathcal{D}),$$

where α is the *learning rate*. However, computing the actual objective function for the entire dataset is usually too computationally expensive and instead it is approximated by computing it for a smaller subset of $\mathcal{D}$, which is called a *batch*. In optimisation literature algorithms calculating the update from the entire dataset are called deterministic, while methods that use subsets are called stochastic [20]. The most famous and still popular algorithm is *Stochastic Gradient Descent* (SGD).

There have been several extensions of the SGD-algorithm, one of the most popular ones is the Adam algorithm (whose name is derived from adaptive moment estimation) [22]. Adam introduces an adaptive low-order estimation of the individual parameter's moments, which has shown a lot of success in classic supervised learning [20, p. 276]. Adam calculates an individual learning rate for each parameter and its update rule is defined as

$$\begin{aligned}
\theta_t &= \theta_{t-1} - \alpha\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\zeta}, & \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t}, & m_t &= \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta J(\theta_t), \\
\hat{v}_t &= \frac{v_t}{1-\beta_2^t}, & v_t &= \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta J(\theta_t))^2,
\end{aligned}$$

where $\hat{v}_t$ and $\hat{m}_t$ are estimations of the first and second order moments of the gradient, ζ is a small constant to avoid division by zero and β_1 and β_2 are hyperparameters. With β_i^t we denote β_i to the power of t.

2.4 Reinforcement Learning

Reinforcement Learning (RL) can in simple terms be described as the act of learning by interacting with an environment. It is one of the three main paradigms of machine learning together with supervised and unsupervised learning. By creating agents that can learn by interacting with their environment, desirable behaviour can be trained without needing to annotate data. Instead, a reward function is created and finding the optimal behaviour is defined as an optimisation problem.

In this section, we will first formulate the RL problem in Section 2.4.1. This is followed by a derivation of the PPO algorithm in Section 2.4.6 with the help of Policy Gradients methods, Section 2.4.2, and TRPO, Section 2.4.4.

2.4.1 Problem Statement

The task in RL is usually expressed as a infinite horizon discounted *Markov Decision Process* (MDP) [16]. The MDP model is represented by the tuple $(\mathcal{S},\mathcal{U},\mathcal{P},\gamma,R)$, where $\mathcal{S}$ is the set of states, $\mathcal{U}$ the set of actions, $\mathcal{P}$ the state-action transition probability, γ a discount factor and $R: \mathcal{S}\times\mathcal{U}\to\mathbb{R}$ the reward function. To decide what action to perform given a state, the agent follows the *policy* $\pi: \mathcal{S}\to\mathcal{U}$. Policies map every state to a distribution over the space of actions $u\sim\pi(s)$, where the case of $u=\pi(s)$ is called a deterministic policy since the probability distribution has converged to one single action.

The agent's objective is to choose a sequence of actions that maximises the expected return. Given a trajectory $(s_0,u_0,s_1,u_1,...)$ the return at time t is defined as

$$G_t := \sum_{i=0}^{T}\gamma^i\ R(s_{t+i},u_{t+i},s_{t+i+1}). \tag{2.6}$$

Here T is the length of one episode which differs between applications. In games one episode is one game, while for real-world applications an episode is usually infinite. In the latter case, the constraint on the discount factor $\gamma<1$ is required as otherwise, the expected return would explode.

The term value function is sometimes used for both *state-value function* and *action-value function*. Both of them are measurements of the expected return but with different conditioning. They are given by

$$V^\pi(s) = \mathbb{E}_\pi\left[G_t \mid s_t=s\right] = \mathbb{E}_\pi\left[\sum_{i=0}^{T}\gamma^i\ R(s_{t+i},u_{t+i},s_{t+i+1})\middle| s_t=s\right], \tag{2.7}$$

$$Q^\pi(s,u) = \mathbb{E}_\pi\left[G_t \mid s_t=s,\ u_t=u\right] = \mathbb{E}_\pi\left[\sum_{i=0}^{T}\gamma^i\ R(s_{t+i},u_{t+i},s_{t+i+1})\middle| s_t=s, u_t=u\right]. \tag{2.8}$$

The state-value function $V^\pi(s)$ is the expected future discounted return of state s if policy π is followed. This function is used to estimate how desirable it is for the agent to inhabit a particular state. The state-action function $Q^\pi(s,u)$ is interpreted as the expected return if action u is taken in state s and subsequent actions are taken by the policy π.

The expectation $\mathbb{E}_\pi$ is with regard to the actions following the policy π and states follow the transition probability $\mathcal{P}$, but the transition probability is omitted to simplify notation.

The combination of (2.7) and (2.8) is called the *advantage function* given by

$$A^\pi(s,u) = Q^\pi(s,u) - V^\pi(s).$$

The advantage function can be interpreted as a measure of how "good" it is to take a fix action u in comparison to randomly taking an action according to the policy

[23]. In the case $A^{\pi}(u, s) > 0$ it is more preferable to choose action u compared to sample from the policy.

An important distinction is if an algorithm is *model-based* or *model-free*. This makes the distinction of wether the agent has access to or learns the dynamics of the environment it interacts with. A famous example of a model-based algorithm was the Go-playing agent *AlphaGo* [2]. It is model-based since it knows, given a board state and action, what the resulting state is. However, in most real-world applications the agent doesn't have access to the real model since they are too complicated. This is why a lot of recent research has focused on the model-free reinforcement learning.

Model-free RL algorithms are divided into *Policy Gradient Methods* and *Q-learning*.[1] In Q-learning the task is to learn an approximation of the state-value representation (2.8). The policy is then constructed by choosing the state-action pair resulting in the greatest estimated value. Policy gradient methods instead directly parametrize the policy and optimise it.

2.4.2 Policy Gradient Methods

The aim of policy gradient algorithms is to maximise the expected reward while optimising the policy directly. The policy $\pi(\cdot \mid s; \theta)$ is assumed differentiable in the parameters θ [25]. For the ease of notation we write π_θ. As discussed in Section 2.3, given a scalar objective $J(\theta)$ with respect to the parameters the objective is to maximise the policy's performance by gradient ascent according to

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla_\theta J(\theta_t)}, \tag{2.9}$$

where $\widehat{\nabla_\theta J(\theta)}$ is a stochastic estimate whose expectation approximates the gradient of the objective [25]. For an episode starting in state s_0, the objective can be written as

$$J(\theta) = V^{\pi_\theta}(s_0) = \mathbb{E}_{\pi_\theta}\left[G_t \mid s_t = s_0\right].$$

The objective is expressed as maximising the expected return, given the initial state s_0. It is complicated to compute the gradient of J since both the policy and the stationary distribution are dependent on the parameter θ. The *policy gradient theorem* provides a analytical expression for computing the gradient as

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[Q^{\pi_\theta}(u, s) \nabla_\theta \log \pi_\theta(u \mid s)\right]. \tag{2.10}$$

The theorem provides a expression for the gradient of J with respect to θ, which is what we need for (2.9) that does not involve the derivative of the state distribution. The gradient computed by (2.10) is proportional to the exact solution, but for complete proof we refer to [25, p. 325].

This expression is the theoretical foundation for policy gradient algorithms and can be plugged straight into (2.9). It can be shown that subtracting a baseline $B(s)$ not

[1] An algorithms however do not exclusively need to belong to one or the other but utilise both methods for example DDPG [24].

dependent on the action u from the action-value function will not change value of the policy gradient while reducing the variance of the estimation [25]. A popular baseline which yields almost the lowest possible variance [23] is $B(s) = V^{\pi_\theta}(s)$. The policy gradient is then described by

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[A^{\pi_\theta}(u, s) \nabla_\theta \log \pi_\theta(u \mid s) \right]. \tag{2.11}$$

For an overview of other baselines and policy gradient methods see Schulman et al [23].

2.4.3 Generalized Advantage Estimation

When computing the approximation of the policy gradient (2.11) the advantage function must be approximated since it is not possible to compute exactly. Schulman et al. [23] proposed the *Generalized Advantage Estimation* (GAE) as a stable method that reduces the variance in the advantage estimations at time t, $\hat{A}_t^{\pi_\theta} \approx A^{\pi_\theta}(u_t, s_t)$. To estimate the advantage, the state-value function, V_θ, is learned as a parametrized function. This is achieved by adding a second term to the total objective J for the state-value function as

$$J^{\mathrm{VF}}(\theta) = \mathbb{E}_\pi \left[(V_\theta(s_t) - V_t^{\mathrm{target}})^2 \right] \tag{2.12}$$

The target can be chosen as $V_t^{\mathrm{target}} = G_t$, with G_t defined as (2.6) [23], i.e. as the return approximated with Monte Carlo sampling.

Using the state-value function the advantage is approximated as

$$\begin{aligned} \delta_t^\theta &= R(s_t, u_t, s_{t+1}) + \gamma V_\theta(s_{t+1}) - V_\theta(s_t), \\ \hat{A}_t^{\pi_\theta} &= \sum_{k=0}^{T'} (\gamma\lambda)^k \delta_{t+k}^\theta, \end{aligned} \tag{2.13}$$

where $\lambda \in [0, 1]$ a parameter with a wide interpretation, see [23, p. 5], and T' a hyper parameter determining how far into the future to we compute the return. δ_t^θ can be considered as a simple estimation of the advantage at time t, i.e. $\hat{A}(s_t, u_t)$ [23]. GAE, $\hat{A}_t^{\pi_\theta}$, is then the exponentially weighted average of the advantage function, over T' time-steps.

2.4.4 Trust Region Policy Optimisation

Policy gradient methods are examples of on-policy learning. This refers to the fact that the policy updates are sampled from the policy being updated. When performing the opposite, off-policy learning, samples generated from older policies can be reused but too big parameter updates in one training step have also empirically been shown to lead to poor performance for policy gradient methods [6]. On-policy learning leads to sample inefficient methods since new samples need to be gathered for each update step. The *Trust Region Policy Optimisation*-algorithm (TRPO)

[26] increases the sampling efficiency using importance sampling and enforcing a constraint on the size of each policy update to avoid too big updates.

Let $p(x)$ and $q(x)$ be two different probability distributions with the same support. With importance sampling the expected value of function $f(x)$ can be calculated by $\mathbb{E}_{x\sim p}\left[f(x)\right] = \mathbb{E}_{x\sim q}\left[\frac{p(x)}{q(x)}f(x)\right]$.

When performing off-policy learning, the policy μ used for collecting samples is different from the policy being optimised π_θ. Meaning when we compute the advantage over the visited states and actions, this is done with regard to the policy we have gathered samples from, $A^\mu(u,s)$. The objective for off-policy learning is expressed as

$$J^{\text{TRPO}}(\theta) = \mathbb{E}_{\pi_\theta}\left[A^\mu(u,s)\right].$$

The mismatch between the distribution used to gather the samples and the policy being optimised for π_θ is solved by using importance sampling as

$$J^{\text{TRPO}}(\theta) = \mathbb{E}_{\pi_\theta}\left[A^\mu(u,s)\right] = \mathbb{E}_{\mu}\left[\frac{\pi_\theta(u \mid s)}{\mu(u \mid s)}A^\mu(u,s)\right].$$

In order for TRPO to work in practice μ need to be close to π_θ, so it is chosen as $\mu = \pi_{\theta_{\text{old}}}$, where θ_{old} refers to the policy parameters from the previous training iteration. Except for making optimisation possible, utilising importance sampling also increase the sampling efficiency of the policy gradient algorithm as samples can be reused for mutiple updates. As mentioned TRPO also addresses the size of each update step by adding the constraint

$$\mathbb{E}_{\pi_{\theta_{\text{old}}}}\left[D_{KL}(\pi_\theta \mid\mid \pi_{\theta_{\text{old}}})\right] \leq \delta,$$

where D_{KL} is the Kullback-Leibler divergence and δ is the trust region parameter. The KL divergence is a measure of how much two probability distributions differ from each other, so δ is the maximally allowed distance between π_θ and π_{old} in one training step.

This motivates the TRPO optimisation problems formulation as

$$\underset{\theta}{\text{argmax}}\ J^{\text{TRPO}}(\theta) \qquad \text{s.t.} \qquad \mathbb{E}_{\pi_{\theta_{\text{old}}}}\left[D_{KL}(\pi_\theta \mid\mid \pi_{\theta_{\text{old}}})\right] < \delta.$$

This problem can approximately be solved using the conjugate gradient algorithm after making a linear approximation of the objective and a quadratic approximation of the constraint [6].

2.4.5 Entropy Penalty

The entropy of a probability distribution can be interpreted as a measure of a distribution's level of uncertainty [27]. As an example, the discrete uniform distribution results in maximum entropy since every outcome has the same probability, i.e. high level of uncertainty. The opposite is a discrete distribution where the probability of

every outcome but one is zero and therefore has the lowest possible entropy since the outcome is predictable.

The entropy of a discrete random variable $\mathcal{X}$ with distribution p over K states is defined as

$$\mathcal{H}(\mathcal{X}) = -\sum_{i=1}^{K} p(\mathcal{X} = i) \log p(\mathcal{X} = i),$$

where the choice of logarithm base varies in different applications, here the natural logarithm is used.

It has been observed in previous studies that adding an entropy bonus to the objective function of policy gradient methods encourage the agent to explore [6]. The policy explores as long as it does not converge to a deterministic one. The possibility of choosing an action not considered the optimal one under the current policy can help the agent escape local minima.

When the policy π consist of several categorical distributions, as is the case when several actions are chosen, the entropy can be calculated as the sum of all distributions entropies as long as the actions are independent of each other.

2.4.6 Proximal Policy Optimization

The TRPO is theoretically correctly stated but relatively complicated to compute. The *Proximal Policy Optimisation*-algorithm (PPO) [6] was formulated with the expressed goal to retain the data efficiency and reliable performance of TRPO without the approximations needed to compute the objective [6].

The PPO clipping objective at time t for one sample is described by

$$\begin{aligned} r_t(\theta) &= \frac{\pi_\theta(u_t \mid s_t)}{\pi_{\theta_{old}}(u_t \mid s_t)}, \\ J_t^{\text{CLIP}}(\theta) &= \min\left(r_t(\theta)A^{\pi_{\theta'}}(u_t, s_t),\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A^{\pi_{\theta'}}(u_t, s_t)\right), \end{aligned}$$

where ϵ is a hyperparameter (usually $\epsilon \in \{0.1, 0.2\}$) and $r_t(\theta)$ the probability ratio representing how close the current policy is to the old one where $r_t(\theta_{\text{old}}) = 1$. The first part in the min-function, $r_t(\theta)A^{\pi_{\theta'}}(u_t, s_t)$ is the objective of TRPO without the constraint, see (2.14), which is called the unclipped objective. As discussed in Section 2.4.4, using this expression without a constraint would lead to excessively large policy updates. The clipping term: $\text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)A^{\pi_{\theta'}}(u, s)$ was therefore introduced to ensure that the probability ratio stay within the interval $[1-\epsilon, 1+\epsilon]$. By taking the minimum of these two terms the new objective is a lower bound of the unclipped objective.

Using the generalized advantage estimation presented in Section 2.4.3 to estimate the advantage function a value-function V_θ also needs to be learned. The loss term presented in (2.12) is therefore added to the PPO objective. The last part added to the objective is the entropy bonus $H[\pi_\theta]$ mentioned in Section 2.4.5 to encourage

the exploration. Putting these three terms together the objective function for the PPO-algorithm is described by

$$J^{\text{PPO}}(\theta) = \mathbb{E}_{\pi_{\theta_{\text{old}}}}\left[J_t^{\text{CLIP}}(\theta) - C_{\text{VF}}\ J_t^{\text{VF}}(\theta) + C_{\text{E}}\ H[\pi_\theta]\right],$$

where $C_{\text{E}},\ C_{\text{VF}}$ are hyperparameters. Using modern automatic differentiation tools[2] the objective function only needs to be computed and then easily differentiated and plugged into the gradient ascent update steps. To increase the variance in the gathered data, it is common to gather data in several parallel processes. An agent using P parallel processes to collect data for T time steps in parallel and then performing K update steps can be summarised as Algorithm 3.

Algorithm 3: PPO

for *training iteration = 1, 2, ...* **do**
  for *parallel process = 1, 2, ..., P* **do**
    Gather trajectories following policy $\pi_{\theta_{\text{old}}}$ in the environment for T time steps.
    Compute advantages estimation $\hat{A}_t^{\pi_{\theta_{\text{old}}}}$ according to (2.13).
  Perform K optimisation steps of J^{PPO} with regard to θ with a SGD-algorithm.
  $\theta_{\text{old}} \leftarrow \theta$

2.5 Real-World Challenges of Reinforcement Learning

In the last few years, reinforcement learning has shown increasing performance in artificial environments and is continuing to improve in real-world settings. However, many of the research advances have been hard to leverage into real-world applications. Dulac-Arnol et al. [28] suggest this is due to "a large gap between the casting of current experimental RL setups and the generally poorly defined realities of real-world systems". They identified nine challenges that are holding back RL in this transition defined as

1. Being able to learn on live systems from limited samples.

2. Dealing with unknown and potentially large delays in the system actuators, sensors, or rewards.

3. Learning and acting in high-dimensional state and action spaces.

4. Respecting system constraints that should never or rarely be violated.

5. Interacting with partially observable systems, which can alternatively be viewed as systems that are non-stationary or stochastic.

[2] Like Tensorflow or Pytorch.

6. Learning from multi-objective or poorly specified reward functions.

7. Being able to provide actions quickly, especially for systems requiring low latencies.

8. Training offline from the fixed logs of an external behaviour policy.

9. Providing system operators with explainable policies.

When working with simulating radar systems items 2, 5 and 6 are relevant challenges but a real-world implementation would include almost all of the items.

These challenges can be expressed in this study as

- When performing a measurements, there is a delay in the update of the state, since it needs to be resolved. Since the state update is delayed, so is the reward associated with it. (Challenge 2.)

- Radar systems are partially observable, since each measurement is a noisy observation of only a part of the system. (Challenge 5.)

- Target tracking is a balance between finding new targets, track known targets and update tracks within a fixed time limit. (Challenge 6.)

Efforts to address these challenges one at a time have been presented, but extended work combining the results have not been performed [28].

3. Method

This thesis aims to investigate how to use reinforcement learning to control radar in a surveillance setting. To evaluate the performance of the algorithms, a simulation environment is used that simulates targets and the radar. In Section 3.1 the target movement, target detection and the tracking algorithm are presented. In Section 3.2 the experiment is presented as a *Markov Decision Process*, described by a state representation, reward function and action space. In Section 3.3 the reference agent is presented. Lastly, in Section 3.4 the reinforcement learning agent is presented.

3.1 Simulation Environment

The simulation performed can be broken down into four parts. The spawning of new targets, the movement of the targets, detection of the targets and the tracking algorithm processing the observations. We next introduce the different steps of the simulator, one by one.

3.1.1 Simulation of Targets

The simulation environment models the movement of aeroplanes within a circle sector representing the surveillance area of an airborne radar system. See Figure 3.1 for an example of the surveillance area with the red line representing at what angle the radar beam is directed and each marker corresponds to a target. The modelled radar is electronically steered, meaning that the radar beam can be redirected almost instantaneously without having to physically move the radar. A simplification of our simulator is that the radar system is not moving, while in real applications an airborne radar is naturally moving.

The simulation of the moving targets are performed within a 2-dimensional surveillance area with coordinates $x,\ y$ oriented as in Figure 3.1. The circle sector is defined as

$$\begin{aligned} r &= \sqrt{x^2 + y^2}, & r &\in [20,\ 450] \text{ km}, \\ \varphi &= \arctan(y/x), & \varphi &\in [-\frac{\pi}{3},\ \frac{\pi}{3}] \text{ rad}. \end{aligned}$$

The angle φ is the azimuth angle, and the centre of the radar, $\varphi = 0$, is called the boresight.

3.1.1.1 Spawning

New targets are spawned uniformly on the boundary of the surveillance region and are terminated when they leave the surveillance area. When spawning new targets

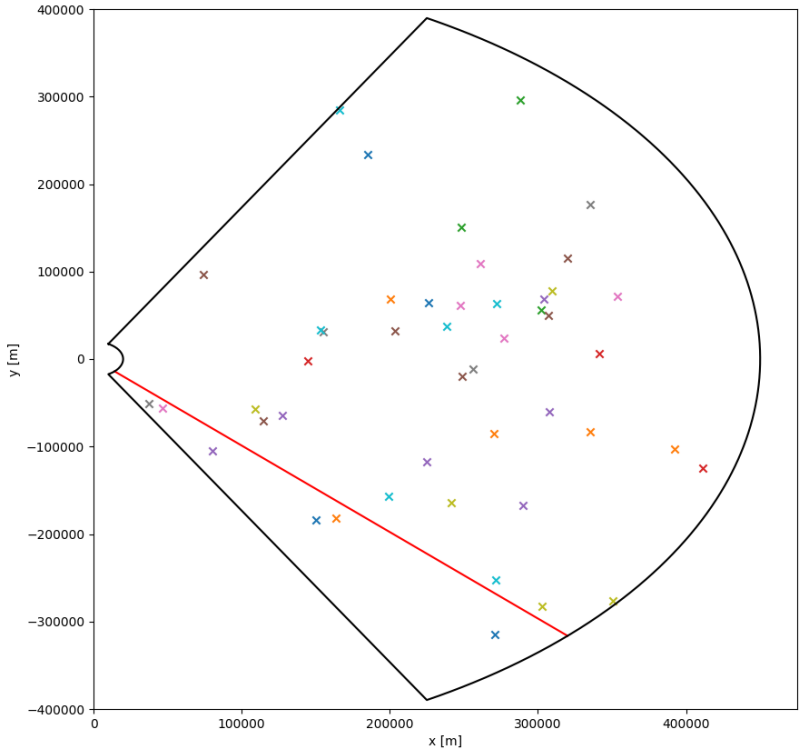

Figure 3.1: An illustration of the surveillance area where the red line represents the direction of the current radar measurement. Each cross corresponds to a target.

a number of parameters are sampled as described in Algorithm 4.

Algorithm 4: Initiating a new target

The following values are sampled:

1: The initial position (x, y): uniformly on the boundary of the surveillance area.
2: The initial velocities $(\dot{x}, \dot{y})$: uniformly among the directions pointing inwards the surveillance area with (truncated) normally distributed speed with mean 200 m/s and standard deviation 25 m/s.
3: The target's radar cross section: uniformly in dB from interval $[5, 20]$.
4: Time to first manoeuvre according to Algorithm B1.
5: Time to next spawning event: exponentially distributed with rate $\lambda_{\text{New target}}$. This rate is updated as a compound Poisson distribution with intensity 10^{-6}. In each update the intensity is updated by adding a sample from the uniform distribution $U(\{-1, 0, 1\})$ to the current intensity.

To model the fact that air traffic intensity changes in time, the rate $\lambda_{\text{New target}}$ changes according to a compound Poisson process as above. This Markov process is chosen

for its simplicity and to introduce varying traffic density into the training. The model is not realistic since sharp transitions in traffic intensity are not expected.

3.1.1.2 Motion

After each action of the radar, the targets are updated using the dynamic models presented in Section 2.2.3 according to Algorithm 5. The available actions and duration are presented in Section 3.1.2.

Algorithm 5: Target update

Input: Time increment T'.
if *Target is performing a manoeuvre* **then**
  Update the target's state using the constant turn rate model (2.5) with time increment T'.
else
  Update the target's state using the constant velocity model (2.4) with time increment T'.
if *Target just finished a manoeuvre* **then**
  Sample new manoeuvre according to Algorithm B1.
if *Target is outside the surveillance area* **then**
  Remove target from simulation.

3.1.1.3 Initialisation

When each simulation is initiated the number of targets and $\lambda_{\text{New target}}$ is set to a value in the range $\{10, ..., 70\}$. All targets are spawned on the boundary of the surveillance area. Therefore the simulation is run for $T \sim U(20, 30)$ minutes of simulation time for the simulation to enter a stationary state.

3.1.2 Simulation of the Radar Detection

Here the models used to perform detections are presented.

3.1.2.1 The Lobe and Lobe Stepping

In the simulator, the lobe width is two degrees in boresight (zero degrees azimuth angle). When steering the lobe electronically the lobe becomes broader as azimuth increases. More precisely the lobe width is taken to be $2/\cos(\varphi)$, where φ is the azimuth angle at the centre of the lobe. When stepping the lobe the step length is also increased. The step lengths are taken to be $0.0125/\cos(\varphi)$ and $0.5/\cos(\varphi)$ for search sweeps (resolved modes) and track sweep (unresolved mode), respectively. Here φ is taken at the current lobe position.

The step length has been chosen so that it takes roughly 8 and 2 steps to reach the border of the current search sweep and track sweep, respectively. In a resolved mode, more measurements are needed for the same target and this is the reason for the smaller step length.

In our simulator, the sweeps are always performed clockwise. Thus a search sweep starts by default at 60 degrees and step by step reaches -60 degrees.

3.1.2.2 Waveforms

In the simulator, radar measurements are made with one out of three waveforms. Waveform here refers to the number of pulses used for a detection. A pulse in the simulator takes 0.07 ms and the three waveforms used are 128, 256 and 512 pulses (waveform 1,2,3), respectively. Using more pulses gives better detection probability and is favourable in the Doppler processing. Before the first pulse has hit the horizon and returned to the antenna, the received data is transient and useless. By a waveform with N pulses, we mean that it has N pulses after having removed the transient pulses. Thus the total number of pulses exceeds N. To model this, in addition to the $0.07 \cdot N$ ms a measurement takes, a 3 ms delay is added to the time each measurement takes. This corresponds to $3/0.07 \approx 43$ additional pulses being removed.

In a resolved measurement we let the simulator use 9 consecutive lobe steps to resolve ambiguities (in range and velocity). In an unresolved measurement, 3 lobe steps are used to illuminate a specific target. There are thus three different actions, starting a resolved/unresolved sweep and continue sweep (resolved or unresolved) that give rise to different times. We summarise the times in Table 3.1.

Number of lobe steps	Waveform 1 [ms]	Waveform 2 [ms]	Waveform 3 [ms]
1	23.92	41.84	77.68
3 (starting new search sweep)	71.76	125.52	233.04
9 (starting new search sweep)	215.28	376.56	699.12

Table 3.1: Duration of each combination of actions expressed in [ms].

In reality, the airborne radar has two surveillance areas, one on each side. Since we only model a one-sided radar but don't want to give it twice as much time to perform its tasks, the times in Table 3.1 are doubled.

Initiation of a new search sweep is needed in two situations. The first is when the sweep has reached -60 degrees and starts over at 60 degrees. The other case is when more than one second has passed since the sweep was updated, in which case the old measurements are considered obsolete and a new search sweep has to be initiated at the same spot where the last was terminated.

3.1.2.3 Detection Probability

The probability of detection is dependent on the Signal to Noise Ratio (SNR) of each performed measurement. By rewriting the radar equation (2.1) in dB the SNR

of a target at distance r is described by

$$SNR_{\text{dB}}(r, v_r, \beta, C, i) = K_i - 40 \cdot \log r - F \cdot r + C + 20 \cdot \log(\cos \beta^{1.5}) + 10 \cdot \log B(v_r), \tag{3.1}$$

$$B(v_r) = \min(1, \max(0, \frac{v_r - 15}{40})),$$

where $i = 1, 2, 3$ is the waveform, K_i is the SNR_{dB} without losses, F is the exponential loss rate due to atmospheric conditions, C the radar cross section, β the angle of the target relative to the radar centre and v_r the radial velocity of the target relative to the radar. Low velocities are hard or impossible to distinguish from ground clutter which is why B is added to model a linear decrease in SNR_{dB} when the velocity v_r decrease from 55 to 40 m/s with no detectability in the range 0-40 m/s.

Converting the SNR_{dB} from dB to a ratio the observation probability of a target at the distance r from the radar is modeled as

$$P(\varphi, \beta, SNR_{\text{ratio}}, i) = \max(0, \min(1, 2 - 2|\varphi - \beta| \cos \varphi)) \cdot \exp\left(\frac{-S_i}{1 + SNR_{\text{ratio}}}\right), \tag{3.2}$$

where

$$SNR_{\text{ratio}}(r, v_r, \beta, C, i) = 10^{\frac{SNR_{\text{dB}}(r, v_r, \beta, C, i)}{10}},$$

and i and β are as defined above, φ is the azimuth angle and S_i a SNR threshold. The detection problem is a binary classification problem that classifies a measurement as target if SNR exceeds the SNR threshold and as no target otherwise. The above form of the detection probability is derived from Gaussian assumptions on signal and noise. For more about detection, see [11].

3.1.2.4 False Detections

The maximal unambiguous range is the maximal range from which a pulse can be transmitted and received by the antenna before a new pulse is sent, i.e., within 0.07 ms in our case. Taking into account that the pulse needs to travel twice this distance, the maximal unambiguous range is $r = c \cdot 0.07 \cdot 10^{-3}/2 = 10500$ m , where $c = 3 \cdot 10^8$ m/s is the speed of light. Another way to see this is that a detection with detected range r can come from any range $r + n \cdot 10500$ m, $n \geq 0$, explaining the term ambiguous. For unresolved measurements, we look for a certain target at a position and velocity that are predicted by the tracker. Under the assumption that the radar sends 10% of the time and listens only 90% of the time the range that the system actually can receive a signal before sending a new is $10500 \cdot 0.9 = 9450$ m. In the simulator, we assume that we look for a target in a range and azimuth cell of 9450 m times 2 degrees, symmetrically centred around the target. At each lobe step of a nonresolved mode, with 2% probability, a false detection in the cell is sampled uniformly in azimuth and range. The velocity is sampled precisely as for spawning targets.

For resolved radar modes we consider a setting with one false alarm per 10 seconds on average. This is simulated as follows: When doing a measurement that takes Δt seconds, with probability $\Delta t/10$ sample a false detection. The position of the false detection is sampled uniformly in the entire lobe. Velocity is sampled just as for nonresolving modes.

3.1.2.5 Time for Detection Release

A detection is not released to the tracker until the lobe has passed it fully. In other words, when the factor $\max(0, \min(1, 2 - 2|\varphi - \beta| \cos \varphi))$ in (3.2) is zero, then the detection is released to the tracker. In case the target was detected several times in the sweep the last detection is used.

3.1.2.6 Measurements

Here we describe the simulation of the actual measurement once a target is detected. For a detected target with coordinates (x, y) and velocity $(\dot{x}, \dot{y})$ the measurement is given by

$$h(\vec{x}) = (r,\ \sin\phi,\ \dot{x},\ \dot{y}), \quad r = \sqrt{x^2 + y^2}, \quad \sin\phi = \frac{y}{x}. \tag{3.3}$$

In true radar systems only the radial velocity, i.e., the velocity in the direction of the antenna is measured. Here, for simplicity and since our primitive tracker is considerably less sophisticated than those in real systems, we help it by giving it the full velocity vector. Besides the range, azimuth and velocity we indirectly estimate the radar cross section from solving for C in (3.1).

3.1.3 Target Tracking

In this section the models used are presented in Section 3.1.3.1, the association of targets in Section 3.1.3.2 the termination criterium in Section 3.1.3.3.

3.1.3.1 Models

As the measurement and movement models, f and h in (2.2), a constant velocity model is used as defined in (2.4) and the measurement model defined in (3.3) is used. The parameters used for the tracker is presented in Appendix B.2.

3.1.3.2 Association

As mentioned in Section 2.1, the unresolved modes cannot estimate a targets position without previous knowledge of the targets position. This is realised by only allowing detections made by unresolved modes be associated to know tracks. If an unresolved measurement cannot be associated, according to the Hungarian algorithm in Algorithm 2, a new track is not initiated. If a detection obtain with a resolved mode cannot be associated, we instead initiate a new track.

3.1.3.3 Termination Thresholds

The number of times each track is observed and the number of times each track is attempted to be observed is saved. Each track is then assigned one of three levels of how probable it is a true target. The levels are *Candidate*, *Tentative* and *Confirmed*. A track is terminated when the number of failed observations exceed a set threshold for each certainty level. If a target is not observed for 60 seconds, regardless of the number of attempts, the track is also terminated. The certainty levels and termination threshold for the three certainty levels are presented in Table 3.2.

Certainty level	Obs. to obtain level	Missed obs. to terminate
Confirmed	3	5
Tentative	2	3
Candidate	1	2

Table 3.2: The table present the number of observations needed to obtain the different certainty levels as well as the number of missed observations to terminate a track with this level.

3.2 MDP Representation

We formulate the problem of learning to control the radar to monitor the surveillance area using an MDP. The MDP is represented by the tuple $(\mathcal{S},\mathcal{U},\mathcal{P},\gamma,R)$ as defined in Section 2.4.1. To be able to solve the task, we need to design a state representation able to capture the targets movement pattern while fulfilling the Markov property. With the Markov property we are referring to that the transition $s_t \rightarrow s_{t+1}$ only depend on s_t.

The movement of each target fulfils the Markov property since the dynamic models used, see Section 2.2.3, are only dependent on the current state of the targets. The times to initiate new targets, to change dynamic models (manoeuvre or not) and change the spawning rates are sampled from exponential distributions because the exponential distribution retains the Markov property.

At each point in time, the entire state including the exact state of every target is not known, but instead, an estimation of the state is available, calculated by the tracker, from noisy measurements being collected as a result of the controller. Such systems are called partially observable and can be modelled as a *Partially Observable Markov Decision Process* (POMDP). With a Bayesian filtering algorithm, the posterior state distribution of the POMDP can be computed, called the *belief state*. Solving a POMDP is equivalent to solving a continuous-state MDP where the states are belief states [16, p. 150]. Thus using the tracking algorithm's posterior as the state the experiment can be expressed as an MDP.

The experiment can be expressed as an MDP and in this section this MDP will be

described more closely. The state $\mathcal{S}$ is the estimated position of each target by the tracking algorithm presented in Section 3.2.1. The action space $\mathcal{U}$ is presented in Section 3.2.2 and the reward in Section 3.2.3. The transition $\mathcal{P}$ is the transition of the tracking algorithm as expressed in Section 2.2 and the discount factor γ is a hyperparameter set during the experiment.

3.2.1 Tracker State

The state of the tracker consists of 27 features for each tracked target. The used features are

- Certainty level: {Confirmed, Tentative, Candidate} represented as {0, 1, 2} encoded using one-hot encoding, meaning confirmed is represented as $(1, 0, 0)^T$, tentative as $(0, 1, 0)^T$ and candidate as $(0, 0, 1)^T$.
- Time since last successful observation in seconds.
- Number of failed observations since last successful one.
- Estimated radar cross-section.
- Estimated position and velocities by the EKF.
- Covariance matrix estimated by the EKF.

In deep reinforcement learning applications, it has been observed that normalising the input to neural networks have increased the training speed. How each feature is normalised is presented in Appendix A.2.

3.2.2 Actions

In each iteration of the simulation the agents, both the RL based one and the rule-based baseline, can choose between two actions. Ideally, the RL agent would choose the waveform, azimuth and measurement mode but we have constructed a simpler problem in this thesis, and as an extension, more control can be given to the reinforcement learning agent. The two possible actions are:

3.2.2.1 Continue Search

When each experiment is initiated, the agent set a *current search angle*. The angle is set to the top end of the surveillance area, i.e. 60°, see Figure 3.1 for illustration of the area. When the action to continue searching is chosen the agent performs one measurement with a resolved mode at the current search angle and then decreases the angle by one step. When the angle reaches the end of the area, -60°, the search angle is reset to the top again.

If more than one second has passed since the last search action, a new search sweep is

initiated at the current search angle, as described in Section 3.1.2.2. The waveform for each search sweep is chosen depending on how many tracks the tracker is holding. If the tracker is holding less than 15 tracks waveform 1 is used, between 15 and 49 tracks waveform 2 and otherwise waveform 3.

3.2.2.2 Update Track

When the agent chooses to update a track it also chooses which of the tracks in the tracker to update. It then performs several measurements with an unresolved mode, covering the estimated position of the chosen target plus minus the standard deviation estimated by the EKF. Given the estimated positions, $\hat{x}$, $\hat{y}$, and standard deviations σ_x, σ_y of the chosen track, we calculate the four angles: $\arctan(\hat{y} \pm \sigma_y, \hat{x} \pm \sigma_x)$. The highest and lowest valued angles are then chosen, and the track sweep is performed between them as described in Section 3.1.2.2.

The waveform used in each measurement is dependent on the number of failed update attempts of the track. Starting with waveform 1, the waveform is increased with 1 for each failed attempt up to waveform 3.

3.2.2.3 Action Duration

The two actions give the agent the flexibility of switching in-between tracking targets and searching for new ones. The two actions however take different amounts of time to perform. Continuing a search can be either one measurement if there is an active search, or 9 if a new resolved sweep needs to be initiated. Updating a track take at least 3 measurements but can take more depending on the estimated standard deviation.

3.2.3 Reward Functions

When designing reward functions it is important to not introduce bias into the behaviour of the agent. The reward is our way of communicating to the agent what we want it to achieve, not how [25]. For that reason we designed a reward function that incentives the agent to update tracks and search for new targets. This is achieved with two types of bonuses and one penalty.

The first bonus is for finding new candidates, $B_{\text{new}} = 50$. The second one is for reobserving a track, B^i_{observed}, where i is the certainty level of the track, defined in Table 3.2. To incentivise the agent to not reobserve the tracks too fast, the bonus is only achieved if the observation is performed within a time window defined by the limits L^i_{lower} and L^i_{upper}.

The agent is penalised if a track is not updated in the time window defined by L^i_{lower} and L^i_{upper}. The penalty is proportional to the time exceeding L^i_{upper}, multiplied by a factor P^i. This goes against the notation of not showing the agent how to do things, but will be discussed further in Section 4.4.

When calculating the reward for taking action u_t, the penalty and bonus for each track in state s_t is calculated and summed. Lastly, the calculated reward is scaled by a constant $C_{\text{reward}} = 50$, this has been shown in previous RL applications to speed up training. Each parameter value is presented in Table 3.3.

Certainty level	$(L^i_{\text{lower}},\ L^i_{\text{upper}})$ [s]	P^i	B^i_{observed}
Candidate	(0, 5)	-0.5	5
Tentative	(0, 5)	-0.5	5
Confirmed	(5, 10)	-1	10

Table 3.3: Reward function parameters, where i correspond to the certainty level.

3.3 Baseline Agent

As a reference to compare the reinforcement learning agent against, a baseline-agent is designed as a much simpler version of how real radar systems historically have been controlled.

The main idea of the baseline agent is to reobserve known tracks within a set time limit, while regularly performing sweeps of the surveillance area. The sweeps are crucial to not miss new targets in the area. This is achieved by defining two time limits, a revisit time- and a sweep time-limit. The revisit-limit defines how long time to wait between observing each track. The sweep-limit is a time limit on how often a sweep of the entire area needs to be performed. The revisit-limit is set to 7 seconds while sweep-limit is set to 60 seconds. At each iteration, the agent checks if it is breaking any of the limits, if not it continues searching the area. The behaviour of the baseline agent in each iteration is summarised in Algorithm 6.

Algorithm 6: Baseline agent

if *Time to finish sweep > Sweep limit* **then**
  Perform action: **Continue search**.
else if *Time since last observed a track > Revisit limit.* **then**
  Choose track with the highest time since last observed.
  Perform action: **Reobserve track**.
else
  Perform action: **Continue search**.

3.4 Reinforcement Learning Agent

The reinforcement learning agent can be divided into three parts, the encoders, the state-value function and the policy, connected as described in Figure 3.2. All three are constructed by neural networks, each with its own learnable weights. Each

network is denoted by θ to represent the weights, but they do not share weights between the different networks.

The encoder's task is to learn a lower dimensional representation, ψ_t, of the current state s_t and is presented in Section 3.4.1. The lower dimensional representation is the input to the policy and value-networks. The value network approximates the state-value function used in the PPO algorithm as discussed in Section 2.4.6. The policy network produces the policy $\pi_\theta(\cdot \mid s_t)$, presented in Section 3.4.2.

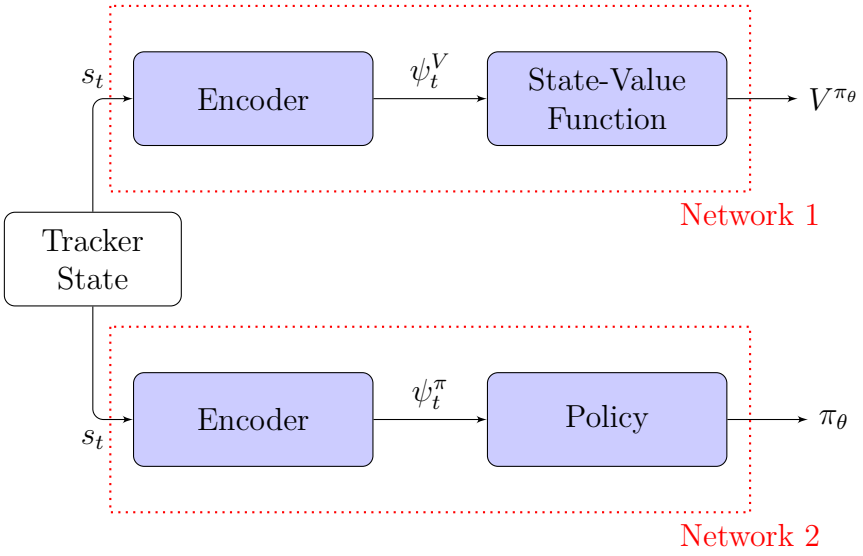

Figure 3.2: Schematic of the agent design. The architecture of the two encoders are identical. They process the state of the tracker to provide a fixed-size input to the policy and and state-value networks.

3.4.1 Encoder

In traditional machine learning tasks, the input is usually fixed dimensional vectors but the number of tracks varies over time. To handle this issue the encoding step is added, which can output a fixed-size input to the PPO algorithm.

A simple way of encoding a varying input size is to pad the input. This is widely used within the field of computer vision to pad images to a fixed size [20], which convolutional neural networks are adept at interpreting. Fully connected networks on the other hand are not as flexible when padding the input, since each weight is connected to a feature in the input. To compensate for this we use an encoding step that randomly orders the inputs to prevent the network from overfitting specific inputs. The idea of the encoder is to assign each track in state s_t a unique integer in the span $[1, N_{\max}]$. Each track is added to a matrix in the row corresponding to its identifier.

In the system, there are delays between the execution of an action and the resulting update of the state and reward. To compensate, information about previously

performed actions I_t is added to the encoding. I_t contain information about the 20 previously chosen actions. For each iteration in {t-20,...,t} it contains: the *current search angle*, chosen action u represented as One-hot encodings, meaning *Continoue search* is encoded as (1, 0), *Update track* as (0, 1) and the chosen track as a vector of zeros the length of length $N_{\max}$, where the index corresponding to the chosen track is 1.

The matrix and the information vector I_t is then reduced in dimension by a neural network. The entire algorithm is presented in Algorithm 7.

Algorithm 7: Encoder

Input: State $s_t \in \mathbb{R}^{N_t \times N_f}$, $N_t \leq N_{\max}$ and information about previous actions $I_t \in \mathbb{R}^{N_I}$.

1: Create matrix $M \in \mathbb{R}^{N_{\max} \times N_f}$ filled with zeros.
2: Add each track in s_t to the row corresponding to its identifier.
3: Unroll M into a vector of size $N_{\max} \cdot N_f$.
4: τ_t =Concatenate(M_{Unrolled}, I_t)
5: $\psi_t = h_\theta(\tau_t)$, where h_θ is a fully connected neural network.

Output: ψ_t

This encoder introduces a limitation on the number of tracks a trained agent can hold since the agent is trained for a fixed $N_{\max}$.

3.4.2 Policy

Given the state calculated by the encoding step ψ_t, the agent samples two actions from the policy $(a_t,\ \phi_t) \sim \pi_\theta(\cdot \mid s_t)$. The action a_t concerns which of the two actions {Continue search, Update track}, presented in Section 3.2.2 is chosen. If the second option, reobserve, is chosen, the agent also chooses the track ϕ_t to update, where ϕ_t correspond to the identifiers of each track.

Each policy is represented as a categorical distribution since both action spaces are discrete. The probability of each action is calculated by a neural network. The number of available actions varies over time since the number of tracks is not fixed but the output of the network needs to be. This is solved by masking the computed probabilities of all actions not available, setting them to 0 which have shown promise in increasing the training speed of previous applications [29].

3.4.3 Network Design

The three building blocks presented in Figure 3.2 are all three represented by MLP networks. The chosen size of each network and chosen activation functions are presented in Table 3.4. The Softmax functions are used as output activation functions of the policy networks to be able to interpret the results as probabilities. The value network needs to be more flexible, why no activation function is used.

Layers	Encoder	Value	Action policy	Track policy
Hidden	$(10 \cdot N_{\max}, 5 \cdot N_{\max})$	(100, 100)	(100, 100)	(100, 100)
Output	$(2 \cdot N_{\max})$	(1)	(2)	$(N_{\max})$
Activation, hidden	LeakyReLU	LeakyReLU	LeakyReLU	LeakyReLU
Activation, output	None	None	Softmax	Softmax

Table 3.4: Neural network architectures used for the different parts of the PPO algorithm. Each neural network is a MLP, where the numbers in the parentheses represent the size of each layer. $N_{\max}$ is the maximum number of tracks the algorithm can track described in Algorithm 7.

The hyperparameters have been chosen through trial and error and with inspiration from the original publications of the methods used.

3.4.4 Training Scheme

The four parts of the agent are trained with the PPO algorithm as described in Algorithm 3. The two networks, as defined in Figure 3.2, are optimised with two separate Adam optimisers.

To increase the variance in the training data, it is gathered in 10 parallel processes where each process run a unique simulation initiated as described in Section 3.1.1.3. Each simulation is initiated with a different number of targets sampled uniformly in the range $\{10, \ldots, 70\}$ to train the agent on different amounts of targets. After the initialisation, the baseline agent runs for the time equivalent to two "Sweep time-limits" to create the initial state. This is done to evaluate the agent's performance in scenarios in which the state already holds tracks. Each training simulation is reset every 5 training epochs, sampling new initial conditions.

4. Results

The purpose of this thesis is to evaluate the PPO algorithm presented in Section 2.4.6 on the presented radar simulation. In this chapter, the specifics of each experiment is presented in Section 4.1 and the performance of the algorithm is presented in Section 4.2. This is followed by a discussion of the results in Section 4.3 and other methods explored during the project in Section 4.4. Unfortunately, the reinforcement learning agent was not able to learn to solve the problem and for this reason the presented results are chosen not to be very extensive. Comparisons with the baseline are presented for six different scenarios. Besides this some diagnostics of the training and the trained agent are presented.

4.1 Experimental Setup

After training the agent, it and the baseline are evaluated on six different scenarios. We use three levels of target intensities, see Table 4.1, and for each intensity one stationary and one non-stationary scenario is used. In the stationary scenarios, the spawning rate corresponds to the initial number of targets. For the non-stationary case, the spawning rate is set to 10 higher than the initial number of targets.

Experiment	Initial number of targets
Low intensity	10
Medium intensity	40
High intensity	70

Table 4.1: The stable amount of targets within the surveillance area during evaluation.

As evaluation metric, the average reward per second is used. It is computed for the trained agent and the baseline, for each scenario, based on five simulations over 30 minutes simulation time.

For the hyperparameters used for the PPO agents training see Appendix A.1.

4.2 Results

The results for each experiment are presented in Table 4.2. The first obvious observation is that the PPO algorithm performs worse than the baseline for every number of tracks. Only for the lower intensities of targets do the PPO agent achieve

a positive score. We recall that this means that, on average, the reward gained from finding new and reobserving tracks is larger than the penalties of failing to update existing tracks timely. For the medium and high-intensity experiments, the PPO agent achieves a negative score but 0 is within one standard deviation in the stationary experiment with medium intensity. The PPO agent is not able to optimise the problem, to understand why we look closer into the action probabilities of the agent.

Simulation	Intensity	PPO	Baseline
Stationary	Low	0.09 ± 0.01	0.24 ± 0.04
Non-stationary	Low	0.09 ± 0.01	0.52 ± 0.05
Stationary	Medium	-0.02 ± 0.08	0.83 ± 0.13
Non-stationary	Medium	-0.12 ± 0.06	0.81 ± 0.13
Stationary	High	-0.44 ± 0.11	0.42 ± 0.47
Non-stationary	High	-0.44 ± 0.10	0.43 ± 0.34

Table 4.2: Result of the experiments with each agent evaluated on the same 5 simulations. The score is reward per seconds with one standard deviation.

Statistics for the taken actions during the first 10 minutes of one evaluation run is presented in Figure 4.1. It is only when we have few tracks in the simulation that the agent has a non-negligible search frequency. The same behaviour was displayed during every run, so one of them was chosen to illustrate the behaviour.

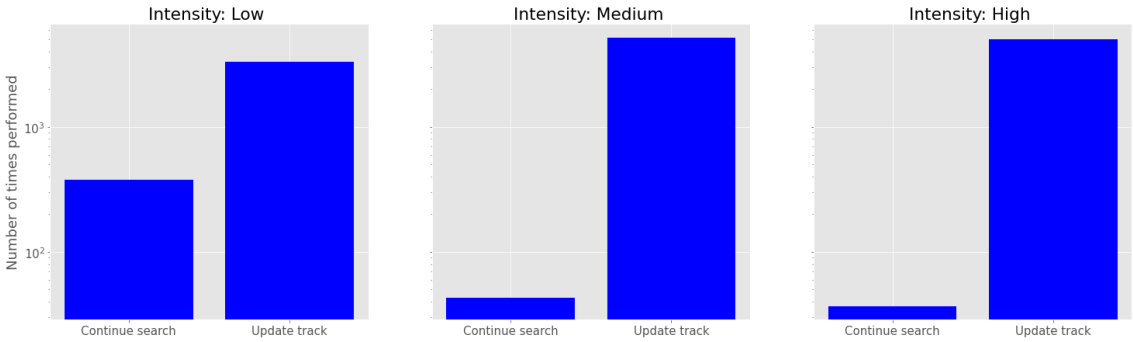

Figure 4.1: The taken actions during the initial 10 minutes of one evaluation run for each intensity level.

We see that the agent heavily favours to update tracks, it is therefore interesting to investigate how it chooses which track to update. The observation probability of each track given the time since it was last observed during one of the evaluation runs is presented in Figure 4.2. The figure only contains instances when the tracker holds 35 tracks to remove the dependence of the number of tracks but the probabilities were similarly distributed for every fixed number of tracks. The dotted line represents the probability of a uniform distribution, i.e. probability of 1/35. It can not be detected any preference or correlation between time since last observation and

tracking probability. The spread of probabilities is also very low, with the majority of targets within the span of 2-5% probability of detection. In other words, the model does not show any strong preferences between different tracks.

It would be interesting to investigate how the certainty level effect the update probability but that was not possible in this study. This is because as we can see in Figure 4.1, the agent do not search for new candidates, so all the samples represented in Figure 4.2 are of the Confirmed level.

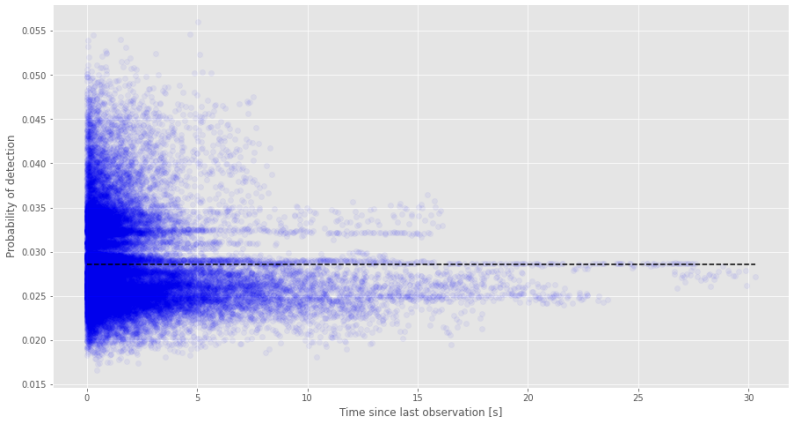

Figure 4.2: The probability of updating each track given the time since the track was last observed. Each sample is taken from a setting when the tracker is holding 35 tracks to remove dependency on the number of tracks. The dotted line represent a uniformly distributed probability of 1/35.

As can be seen, the agent does not learn to prioritise between tracks depending on time since last observation, which is the parameter that influences negative rewards. We next see how the time between observations of tracks are distributed. In Figure 4.3 the times between each successful observation of a track during the first 10 minutes of one of the evaluation runs are presented. As shown in the figure the majority of observations are made early, within the first few seconds. The shape of each distribution is close to a linear function, but note that the axis is logarithmic. This suggests that the distribution of time between observations follow an exponential distribution. This supports the claim that the agent chooses which track to reobserve at random since an exponential distribution is the distribution of time between events sampled at a constant rate. The presented distributions are not completely exponential, probably because the number of tracks changes over time as targets leave the surveillance area so the sampling rate is not constant. The negative score for higher target intensity can also be observed here since the time between observation increase with an increased amount of targets if the sampling is random and not informed.

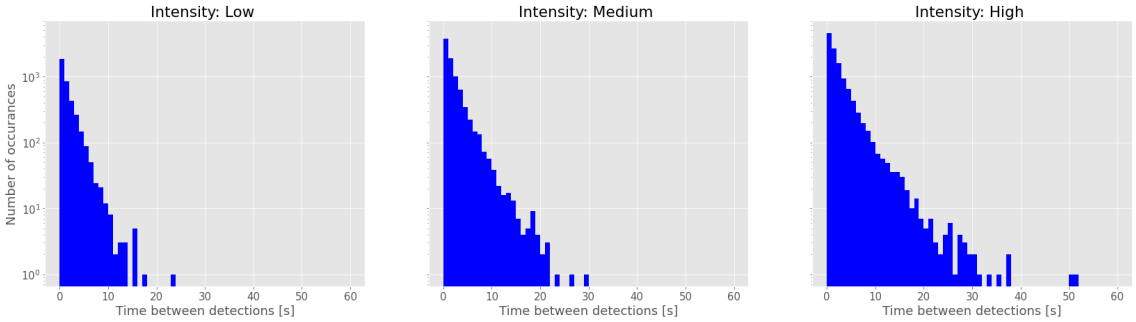

Figure 4.3: The number of times each "Time between detections" occur during the first 10 minute of a evaluation run for each level of intensity.

As mentioned in Section 2.4.5, the entropy of a reinforcement learning agent can be interpreted as how predictable its behaviour is, where a uniform distribution results in the maximum entropy since it is the least informative distribution. When working with reinforcement learning, looking at the entropy during training is a way of evaluating if the policy is learning. In Figure 4.4 the entropy of the best performing training run is presented. Each line represents the average entropy during one of the parallel processes gathering the data while the black lines in each figure represent a rolling mean of the last 5 iterations. As can be seen the entropy decreases during training for the choice of action, but still show high variance for each parallel process. The entropy for which track to update however seems to initially decrease but then increase again. It is worth noting that even if the entropy of the tracking decision decreases, it is with such a low percentage that the decision is still close to uniformly distributed, supporting the previous observations.

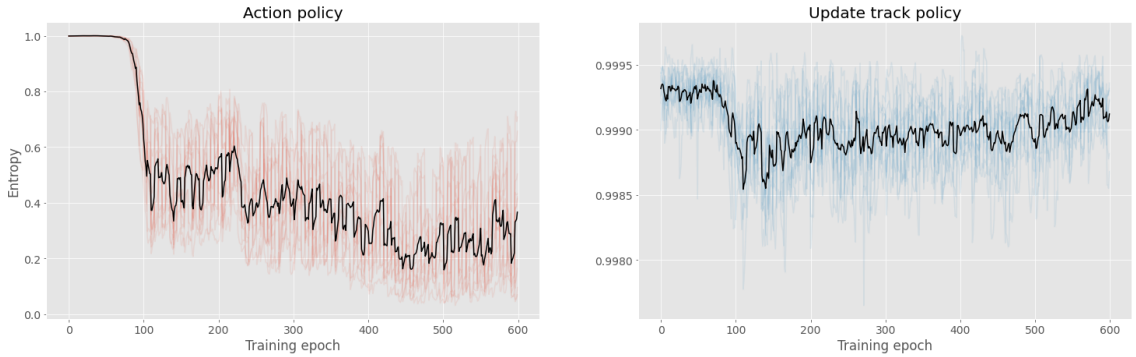

Figure 4.4: Entropy of the two policies used to decide which action to perform and what track to reobserve. Each coloured line corresponds to one of the 10 parallel processes gathering data, while the black line is the rolling mean of all processes.

In Figure 4.5 the PPO agent's reward per action is presented as the mean of each training iteration. We see how the initial decrease in entropy corresponds with an increased average reward, showing the agent has learned something. However, this increase does not on average result in a positive score.

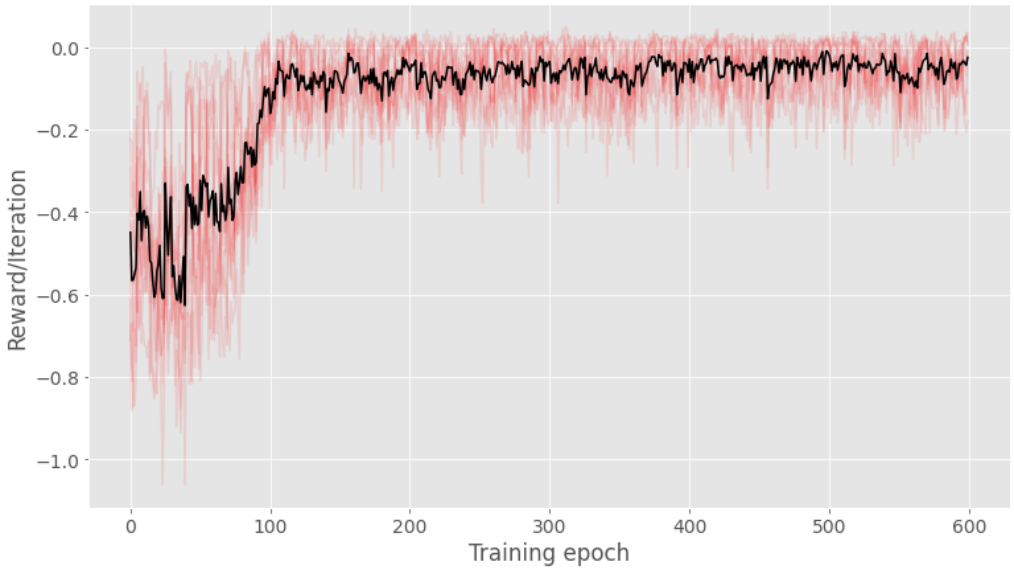

Figure 4.5: Mean reward per iteration during each training episode. Each coloured line corresponds to one of the 10 parallel processes gathering data, while the black line is the rolling mean of all processes. During training only the chosen action is logged but not which waveform is used since that is not a decision made by the agent, why we don't save the duration of each action. This is why the reward per action is tracked during training and not reward per second.

4.3 Discussion

The results presented in Section 4.2 suggest that the agent is not learning the desired behaviour of the reward function. The baseline agent outscoring the PPO agent show that a higher rewarding policy is possible for the presented reward function. One possible explanation for this is that the agent does not learn to interpret the tracker state successfully. This is supported by the fact that the agent doesn't learn any preference for which track to update. This is an indicator because the time since the last observation is a feature of the state, and is directly proportional to the negative reward received. Instead, the agent's strategy is to randomly update each track it holds. By randomly updating tracks the agent keeps the score closer to zero without being able to get positive. When there are few targets to update the agent find a balance between updating and searching, but still receives a lower score than the baseline. As the number of targets increases the agent give up on searching and instead only updates tracks at random, reducing the negative reward but not achieving a positive reward.

4.4 Other Methods

Other methods for the encoder architecture were also evaluated, but excluded from the presentation because the performance was similar. Two methods that were

evaluated were the DeepSets architecture [30] and a random order encoding. The DeepSets architecture was constructed to handle a varying number of inputs of the same size and has been used in applications like autonomous driving [31] and robotic sorting tasks [32]. The random order encoding was designed similarly to the one used, but the order of the tracks was randomly permuted in each iteration. The network architectures showed similar performance on the radar-control experiment but failed at simple regression tasks when evaluated to decide hyperparameters of the architectures which is why they were left out. The fact that the architectures showed similar performance on the more advanced radar task, but failed at simpler tasks is also an indication that further studies are needed.

The proposed reward function in this report was not the only one evaluated. The one presented in this report was a simple version, constructed to guide the agent to a very specific behaviour rather than a general one. A proposed general reward function gave rewards for the number of tracks in the tracker while penalising too long times between observations. The simpler version, only giving rewards for finding new candidates and updating tracks within time windows was proposed to guide the agent but go against the rule of using the reward to ”communicating to the agent what we want it to achieve, not how”. This did, as observed, not help.

5. Conclusions

In this thesis, we have evaluated the PPO algorithm on a tracking task and compared it to a rule-based model. Unfortunately, the proposed reinforcement learning agent was not able to learn to solve the problem, but the PPO algorithm has previously performed well on control tasks and could be a valid algorithm. Therefore, further work is needed in order to conclusively decide if it is suitable. When working with reinforcement learning the parameters during training are extremely important, and small changes can be crucial for success [33]. This is why we cannot say anything conclusive about the application of the PPO algorithm but only that our implementation and hyperparameters are not suitable for this experiment.

This chapter will discuss the challenges faced during this study in Section 5.1 and present suggestions for future work in Section 5.2

5.1 Challenges

We have identified three challenges for our experiment, which we present here. It is worth remembering that recreating results from RL studies and tuning RL algorithms to new tasks are a challenge in itself [33] but will not be discussed here.

5.1.1 Varying System Delay

As mentioned in Section 3.1.2.5, there is a delay between performing an action and updating the state with the observation. This has been observed to be a challenging task to solve, as discussed in Section 2.5. What set our simulation apart from previously studied scenarios is that the delay is varying for the different actions. This can be one of the explanations why the agent does not learn to update the right tracks. It does not associate the received reward with the performed action because of the varying delay. This was however not investigated further because of time constraints, but is a logical follow up to this study.

5.1.2 Varying State Size

The varying size of the input was the second challenge that also can be an attributing factor to why the agent did not learn the desired behaviour. As mentioned in Section 4.4, other methods were evaluated but yielded similar results, which is why only the simplest one is presented in this study. However, the DeepSets architecture has worked successfully in autonomous driving scenarios and could potentially be suitable for this application.

Similar to the delay in the system, if the agent does not properly learn to associate the features of each track with the probability of each decision this can explain why the agent did not learn to update the tracks proportional to their negative reward. We cannot, however, say too much about the policy, since it is not interpretable. By interpretable we mean that we do not know why it assigns the probability to each action. Interpretability of machine learning systems is an active research field which could be interesting to research further if this kind of systems where to be applied in real-world scenarios.

5.1.3 Simulation

A big limitation of this study comes from the implementation of the simulation and tracking algorithm. Because of the wide scope of this study, where both simulations, tracking algorithms and a reinforcement learning algorithm were implemented, each part could not be optimised for the task as extensively as could be desired. For example, the parameters of the tracking algorithm were tuned for a tracking scenario of 30 targets.

More thorough tuning of each component would allow for a more in-depth analysis of each component's influence on the final results. For a functioning scheduling algorithm, it would be interesting to include metrics analysing how the behaviour of the scheduler affects the tracking algorithm. We left such metrics out of this study since we simply could prove that it behaved in a simple and undesirable manner and such metrics would not add to the comparison since the agent fundamentally did not work.

5.2 Future Work

We have not been able to make our method work on the proposed problem. As a future study, it would be interesting to adapt methods designed to solve the fixed-delay problem presented in [28] to the varying delay problem. Combining these methods with the DeepSets architecture, which has been applied in previous similar applications, would be a logical next step.

The discounted reward G_t as defined in (2.6) is defined for uniform time steps. An interesting extension would be to study how the varying length of each iteration can be compensated for theoretically. This could be a method for redistributing the received reward to the corresponding action.

A recommendation for future work is to start with each problem individually, with smaller and simpler simulations adapted for the individual problems being investigated. Starting smaller and scaling up the complexity of the tasks make analysing the agent's behaviour easier. The presented challenges are suitable to be investigated in simpler and more well-tested environments as a first step before implementing one's own simulation environment.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, *Playing Atari with deep reinforcement learning*, 2013. arXiv: 1312.5602 [cs.LG].

[2] D. Silver, A. Huang, C. J. Maddison, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016-01. DOI: 10.1038/nature16961. [Online]. Available: https://doi.org/10.1038/nature16961.

[3] C. Berner, G. Brockman, B. Chan, *et al.*, *Dota 2 with large scale deep reinforcement learning*, 2019. arXiv: 1912.06680 [cs.LG].

[4] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 2012-01. DOI: 10.1145/3065386.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, *Playing Atari with deep reinforcement learning*, 2013. arXiv: 1312.5602 [cs.LG].

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

[7] OpenAI, I. Akkaya, M. Andrychowicz, *et al.*, *Solving rubik's cube with a robot hand*, 2019. arXiv: 1910.07113 [cs.LG].

[8] A. Mirhoseini, A. Goldie, M. Yazgan, *et al.*, *Chip placement with deep reinforcement learning*, 2020. arXiv: 2004.10746 [cs.LG].

[9] M. Shaghaghi and R. S. Adve, "Machine learning based cognitive radar resource management," in *2018 IEEE Radar Conference (RadarConf18)*, 2018, pp. 1433–1438. DOI: 10.1109/RADAR.2018.8378775.

[10] Z. Qu, Z. Ding, and P. Moo, "A machine learning radar scheduling method based on the est algorithm," in *2019 IEEE 18th International Conference on Cognitive Informatics Cognitive Computing (ICCI*CC)*, 2019, pp. 22–27. DOI: 10.1109/ICCICC46617.2019.9146101.

[11] G. Stimson, H. Griffiths, C. Baker, and D. Adamy, *Stimson's Introduction to Airborne Radar (3rd Edition)*. Institution of Engineering and Technology, 2014, ISBN: 978-1-61353-022-1. [Online]. Available: https://app.knovel.com/hotlink/toc/id:kpSIARE004/stimson-s-introduction/stimson-s-introduction.

[12] S. Särkkä, *Bayesian Filtering and Smoothing*, ser. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2013. DOI: 10.1017/CBO9781139344203.

[13] S. S. Blackman and R. Popoli, *Design and Analysis of Modern Tracking Systems*. Ser. Artech radar library. Artech House, 1999, ISBN: 1580530060. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=cat07470a&AN=clc.6be50f17.e688.423e.8d43.e2abec6b4e2f&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds.

[14] S. Blackman, "Multiple hypothesis tracking for multiple target tracking," *IEEE Aerospace and Electronic Systems Magazine*, vol. 19, no. 1, pp. 5–18, 2004. DOI: 10.1109/MAES.2004.1263228.

[15] R. P. S. Mahler, *Statistical multisource-multitarget information fusion*. Ser. Artech House information warfare library. Artech House, 2007, ISBN: 1596930926. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=cat07470a&AN=clc.a4bb4488.f2a7.4e8a.a0c9.7806c3f4945e&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds.

[16] V. Krishnamurthy, *Partially Observed Markov Decision Processes*. Cambridge University Press, 2016. DOI: 10.1017/cbo9781316471104. [Online]. Available: https://doi.org/10.1017/cbo9781316471104.

[17] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[18] H. W. Kuhn, "The hungarian method for the assignment problem," in *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 29–47, ISBN: 978-3-540-68279-0. DOI: 10.1007/978-3-540-68279-0_2. [Online]. Available: https://doi.org/10.1007/978-3-540-68279-0_2.

[19] X. Rong Li and V. P. Jilkov, "Survey of maneuvering target tracking. Part I. Dynamic models," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 39, no. 4, pp. 1333–1364, 2003. DOI: 10.1109/TAES.2003.1261132.

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

[21] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303–314, 1989-12. DOI: 10.1007/bf02551274. [Online]. Available: https://doi.org/10.1007/bf02551274.

[22] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[23] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, *High-dimensional continuous control using generalized advantage estimation*, 2018. arXiv: 1506.02438 [cs.LG].

[24] T. Lillicrap, J. Hunt, A. Pritzel, *et al.*, "Continuous control with deep reinforcement learning," *CoRR*, 2015-09.

[25] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018, ISBN: 0262039249.

[26] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, *Trust region policy optimization*, 2017. arXiv: 1502.05477 [cs.LG].

[27] K. P. Murphy, *Probabilistic Machine Learning: An introduction*. MIT Press, 2021. [Online]. Available: probml.ai.

[28] G. Dulac-Arnold, D. J. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," in *ICML Workshop*, 2019.

[29] C.-Y. Tang, C.-H. Liu, W.-K. Chen, and S. D. You, "Implementing action mask in proximal policy optimization (PPO) algorithm," *ICT Express*, vol. 6, no. 3, pp. 200–203, 2020-09. DOI: 10.1016/j.icte.2020.05.003. [Online]. Available: https://doi.org/10.1016/j.icte.2020.05.003.

[30] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov, and A. Smola, *Deep sets*, 2018. arXiv: 1703.06114 [cs.LG].

[31] M. Huegle, G. Kalweit, B. Mirchevska, M. Werling, and J. Boedecker, "Dynamic input for deep reinforcement learning in autonomous driving," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 7566–7573. DOI: 10.1109/IROS40897.2019.8968560.

[32] V. Dasagi, R. Lee, S. Mou, J. Bruce, N. Sünderhauf, and J. Leitner, *Sim-to-real transfer of robot learning with variable length inputs*, 2019. arXiv: 1809.07480 [cs.LG].

[33] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, *Deep reinforcement learning that matters*, 2019. arXiv: 1709.06560 [cs.LG].

A. PPO

In this chapter the hyperparameters used during training is presented in Appendix A.1 and how each track state was normalised are presented in Appendix A.2.

A.1 Hyperparameters

The hyperparameters used when training the PPO algorithm are presented in Table A.1. The learning rate was set through trial and error, while the other Adam parameters are the default values.

The discount factor, GAE constant and clipping coefficient are the default values recommended by the original authors.

The remaining hyperparameters where set through testing and computational limitations. No extensive hyperparameter search was possible to conduct within the frame of the study.

Name	Symbol	Value
Adam, Learning rate	α	10^{-5}
Adam, Moment coefficients	(β_1, β_2)	(0.9, 0.999)
Adam, Non zero division coefficients	ζ	10^{-8}
Discount factor	γ	0.99
GAE factor	λ	0.95
Entropy loss coefficient	C_{E}	0.01
Value loss coefficient	C_{VF}	0.1
Clipping coefficient	ϵ	0.2
Updates per training epoch	K	10
Episode length	T	500
Parallel actors	P	10

Table A.1: Hyperaprameters used for the training of the PPO algorithm.

A.2 Normalisation

Each of the features presented in Section 3.2.1 are normalised individually, before the total state is fed to the encoder architecture as described in Section 3.4.1.

- **Certainty level**

The one-hot encoding is not changed but used as defined in Section 3.2.1.

- **Time since last successful observation**
  The maximum before termination is 60. The feature is divided by 60, putting all values in the interval $[0, 1]$.

- **Number of failed observations since last successful one**
  The maximum before termination is 4. The feature is divided by 4, putting all values in the interval $[0, 1]$.

- **Estimated radar cross-section**
  The Baseline-agent ran for 15 scenarios and 500 iteration per scenario, with a target intensity of 50. The mean and standard deviation of the targets estimated RCS:s was calculated from these runs, and used to normalise the RCS C as $\frac{C - 21.1}{10^2}$.

- **Estimated position by the EKF**
  The maximum and minimum value of x and y are calculated, and each feature is shifted by the minimum and then scaled into the interval $[0, 1]$. I.e. $\frac{x - x_{\min}}{x_{\max} - x_{\min}}$, and similarly for y.

- **Estimated velocities by the EKF**.
  The initial velocities for each target is sampeled from the distribution $\mathcal{N}(200, 25)$ m/s. Each velocity, $\dot{x}$, $\dot{y}$ is then normalised as $\frac{\dot{x} - 200}{25^2}$, normalising the values around 0.

- **Covariance matrix estimated by the EKF**
  Since the covariances between the features are of different magnitudes, the positions are in the magnitude of 10^5 while the velocities are in 10^2, there is no easy way to normalise a covariance matrix. What we do instead is to use the eigenvectors and eigenvalues of the covariance matrix to represent it. The covariance matrix is of size 4×4, meaning we can represent it with 4 eigenvalues and 4 eigenvectors. A vector of the eigenvalues is constructed and normalised by its 2-norm. This results in 20 new features, 4 eigenvalues and 4 eigenvectors of length 4, and are used to represent the covariance matrix.

B. Simulation

B.1 Sample New Turn

When sampling a new turn in the simulation we follow Algorithm B1

Algorithm B1: Sample new turn

The following values are sampled:

1: Next time to perform a manoeuvre from a exponential distribution with rate 10^{-4}.
2: New turn duration from a normal distribution with mean 60 seconds and standard deviation 10 seconds.
3: New turning rate from a normal distribution with mean of 1.5 degrees/seconds and standard deviation of 0.5 degrees/seconds.

B.2 Tracking

The process and measurement noise of the EKF-model as defined in (2.2), are defined as

$$Q = \begin{bmatrix} t^3/4 & t^2/2 & 0 & 0 \\ t^2/2 & t & 0 & 0 \\ 0 & 0 & t^3/4 & t^2/2 \\ 0 & 0 & t^2/2 & t \end{bmatrix} \cdot (\sigma_x)^2, \qquad R = \begin{bmatrix} \sigma_r & 0 & 0 & 0 \\ 0 & \sigma_{\sin\phi} & 0 & 0 \\ 0 & 0 & \sigma_{\dot{x}} & 0 \\ 0 & 0 & 0 & \sigma_{\dot{x}}, \end{bmatrix} \tag{B.1}$$

where t is the time increment of the update and the parameters used in the experiments are set to the values presented in Table B.1.

Parameters	Value
σ_r	50 m
$\sigma_{\dot{x}}$	2 m/s
$\sigma_{\sin\phi}$	0.15
σ_x	50 m

Table B.1: Caption

The Jacobian of the measurement model with regard to the state used for the Ex-

tended Kalman Filter (2.3) is defined as

$$\mathcal{J}_h((x, y, \dot{x}, \dot{y})) = \begin{bmatrix} \frac{x}{r} & 0 & \frac{y}{r} & 0 \\ \frac{-x \cdot y}{r^3} & \frac{x}{r} & \frac{y^2}{r^3} & \frac{y}{r} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{B.2}$$

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY