Adaptive beam alignment and tracking in mmWave systems using Reinforcement Learning

Master thesis

Group 974

MSc. in Signal Processing and Acoustics

with Specialisation in Signal Processing and Computing

Aalborg University

01-06-2022

This project was typeset with the online LaTeX editor Overleaf.

AALBORG UNIVERSITY
STUDENT REPORT

Department of Electronic Systems
2nd year of study
MSc. in Signal Processing and Acoustics
Fredrik Bajers Vej 7B
9220 Aalborg Øst
https://www.es.aau.dk/

Title:

Adaptive beam alignment and tracking in mmWave systems using Reinforcement Learning

Project:

Master thesis

Project-period:

September 2021 - June 2022

Project-group:

Group 974

Participants:

Dennis Kjærsgaard Sand
Peter Kjær Fisker

Counsellors:

Carles Navarro Manchon

No. of pages: 165
Appendix: 85 - 165
Date of completion 01-06-2022

Abstract:

This project investigates if Reinforcement Learning is a suitable solution for the problem of beam alignment and tracking in mmWave communication systems with a single base station and a single piece of user equipment. First the mmWave communication systems is modelled, using QuaDRiGa for channel modelling, and a realistic mobility model is implemented for user equipment movement. This is followed by an analysis of the RL framework, first from a mathematical view and then in the context of fitting it to the mmWave communication problem. Two implementations are then proposed using tabular methods, a centralized version with a single agent and a distributed version with two agents cooperating. The parameters of these implementations are then tuned in LOS conditions with broad sweeps over their parameter spaces. After being tuned, the implementations are evaluated in different scenarios to determine the usefulness of RL as a solution. The tested scenarios include LOS/N-LOS conditions and environments with different amounts of noise. As a point of reference a simple heuristic algorithm is implemented as well and tested in the same scenarios. The evaluation showed that a simple multi agent implementation performs best, with promising results in both LOS, NLOS and noisy conditions, that outperforms the heuristic reference algorithm.

Dennis Kjærsgaard Sand

Peter Kjær Fisker

Preface

This is a 9-10th semesters Master thesis in the subject of machine learning. It is made by group 974 of Signal Processing and Computing at Aalborg University. The project was written and developed in the period from September 2021 to June 2022.

The code developed, as a part of the project, can be found in a GitHub repository at the following address: `https://github.com/PeterKjaerFisker/RL-for-Adaptive-Beamforming`.

Chapter 1 and Chapter 2 are written based on a worksheets which was written in full cooperation with another study group at Aalborg university, who also study signal processing and computing. This group is: group 975, designated es-21-spa-9-975, consisting of Nicolai Almskou Rasmussen and Victor Mølbach Nissen. Note however that while the worksheet was written in cooperation with another group, the entire report was written solely by the authors given in the title page.

Reading Guide

The report follows the rules established by the ISO/IEC 80000 standard. Consequently, SI units are used. In this report point (.) is used consistently as decimal separator. Citations are made according to the IEEE citation method. All figures without citations are created by the authors for the project. A list of acronyms used in the report and their meanings is found after the table of contents. Figures, tables, equations, and algorithms are inserted and enumerated according to chapter and order of insertion.

Contents

Acronyms

AoA
Angle of Arrival.

AoD
Angle of Departure.

BS base station.

KED
knife edge diffraction.

LOS
Line Of Sight.

NLOS
Non Line Of Sight.

RL Reinforcement Learning.

SNR
Signal-to-Noise Ratio.

UE user equipment.

ULA
Uniform Linear Array.

UTD
uniform theory of diffraction.

1 Introduction

Technology is a big part of daily life in the 21st century, which is evident from the fact that, 3 of the 5 largest companies, by market cap, are technology firms [1]. However, technology does not only consist of the hardware it's built of, a large part of technological development pertains to data, and the transfer of data. The global demand for data transfer is ever-present and increasing. It has been estimated that 1,5 zettabytes was transferred throughout 2017, which is predicted to increase to 4,8 zettabytes in 2022 [2], [3].

The total demand is not only increasing but the distribution of demand is also shifting. In 2017, wired networks accounted for 48% of IP traffic, while wireless accounted for 52%. This balance is projected to change to 29% for wired networks and 71% for wireless in 2022. Additionally, mobile (cellular) networks is projected to account for 22 % of the global internet traffic in 2022, an increase from 12% in 2017 [2].

This shows a shift towards wireless transfer, however it also shows that the wireless traffic itself is increasingly coming from the mobile network. The rising demand for wireless data transfer will require more infrastructure, but also different infrastructure. Technologies capable of higher bandwidth is needed, and in the case of the mobile network this technology is 5G. 5G is the 5th generation of cellular network and is, as of this reports writing, in the process of being deployed around the world, both in terms of 5G compatible mobile devices, but also cellular towers. While 5G is being actively deployed, it still has problems which, if solved would improve performance. In order to develop fitting and efficient solutions these problems have to be identified and analyzed, both of which is done in the next section.

1.1 mmWave: Technology and challenges

The goal of this section is to explain the reasons behind 5G utilizing the mmWave range of frequencies and the consequences which stem from this.

The mmWave frequencies range from 30 GHz to 300 GHz, which is higher than the current frequencies used for 3G and 4G amongst others [4, p. 5]. The newer 5G communication standard is expected to be able to use frequencies a lot closer to mmWave than the previous generations. From a report from the Danish Energy Agency it has been noted that the frequency band at 26 GHz will be reserved for 5G [4, p. 6].

A higher frequency band is desirable in communication networks as there is room for broader bandwidths or more bands when using the same bandwidths as used at lower frequencies. A broader bandwidth can lead to a higher throughput and makes it possible for more users to use the same frequency range and cause minimal interference. These desirable features acquired by using higher frequencies do not come without a cost. One part of it is having hardware which

can handle operating at the higher frequencies and another part is how the higher frequencies behave in the environment compared to lower frequencies. The first problem of hardware is out of the scope of this report.

1.1.1 Power Loss in free-space

By analysing the Friis transmission equation [5], (seen in Eq. (1.1)), the free-space power loss' dependency on the distance given a certain frequency can be found. The frequency plays into the equation by its wavelength (λ), where a higher frequency equals a shorter wavelength. When the wavelength becomes shorter the values of the right side of the equation becomes smaller by a squared factor. From this it can be concluded that higher frequencies experience a higher degree of power loss over distance travelled compared to a lower frequency when propagating in free-space.

$$\frac{P_r}{P_t} = D_t D_r \frac{\lambda^2}{(4\pi d)^2} \tag{1.1}$$

where:

P_r :	Received Power	[W]
P_t :	Transmitted Power	[W]
D_r :	The receiving antenna's directivity with respect to an isotropic antenna	[.]
D_t :	The transmitting antenna's directivity with respect to an isotropic antenna	[.]
d :	Distance between antennas	[m]
λ :	Wavelength of the radio frequency	[m]

1.1.2 Atmospheric Absorption Rate

The Friis transmission equation only accounts for the free-space loss and therefore can only be used to give an estimate on how fast the power will be lost when transmitting in free-space based on a set frequency. For this reason the other impacts the environment will have on the higher frequencies will also be studied.

From a study about mmWave and its behaviour - *Millimeter Wave Mobile Communications for 5G Cellular: It Will Work!* [6], there are several factors worth noticing. The study included a figure on how much the power will attenuate due to the atmospheric absorption. A recreation of the figure is shown in Fig. 1.1. Overall, the figure shows that the atmospheric absorption increase as the frequency increase, but it also shows areas where the absorption increases drastically compared to nearby frequencies. These frequencies should be avoided for transmitting. The figure also shows some valleys where the absorption is relatively low and it is at these frequencies it's preferred to transmit.

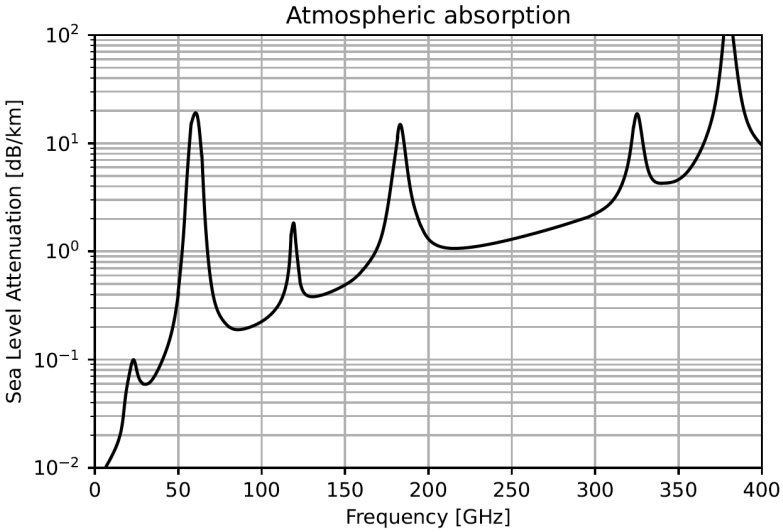


Figure 1.1: Atmospheric absorption across mm-wave frequencies in dB/km, based on [6, p. 338].

1.1.3 Scattering effects

The same study [6] also include a study of some scattering effects, namely penetration and reflection. This study investigated how well signals transmitted at 28 GHz penetrates different materials typically used for building construction. For example for tinted glass and brick pillars it was found that they respectively causes a penetration loss of 40.1 dB and 28.3 dB. The study also included reflection loss for these materials, where it was found that for tinted glass and concrete at an incident angle of 10° the coefficient was respectively 0.896 and 0.815 These aren't the only significant scattering effects, another being diffraction. According to multiple diffraction models given in [7], knife edge diffraction (KED) and uniform theory of diffraction (UTD), the power of the diffracted signal, all else being equal, diminishes as the frequency increases.

The high penetration losses and minuscule diffraction effects makes mmWave signals vulnerable to objects in their paths, and blockage occurs often.

1.1.4 Alleviating the Consequences

In the previous section, the characteristics of communicating in the mmWave range of frequencies were presented, and the reasons behind using it for 5G was presented. The disadvantages however needs to be alleviated and are summarized below:

1. Higher power loss in free-space decreases the communication range.
2. Increased atmospheric absorption further increases the power loss.
3. Low penetration power increases the difficulty in transmitting past blockages.

The penetrative power cannot be changed without changing the building materials, however this is infeasible because of costs. While the penetration power can't be changed, the high reflection coefficient can be advantageous, as it allows the signals to travel along non-line-of-sight paths without being too attenuated.

The power loss due to transmission through free-space and the atmosphere cannot be eliminated, but it can be counteracted. Of course the transmission power can simply be increased to

overcome the attenuation, however this becomes a problem for mobile units, as they have limited power budgets. Another way to counteract the power-loss is by utilizing the transmission power differently. Instead of broadcasting the signal omnidirectionally, signal power can be concentrated by way of beamforming.

To be able to do beamforming, a linear or planar antenna array is needed. The size of the antenna is dependent on which frequency it operates at, as for higher frequencies the antenna becomes smaller. Likewise, in usual constellations of antenna arrays, the antennas are spaced proportionally to the wavelength of the signal they receive. This has the effect that complex and effective beamforming can be achieved using physically small arrays with many elements as 5G uses relatively high frequencies. With more antenna elements in an array, it is possible to achieve higher directivity for the array, which is an advantage in mmWave systems.

1.2 Wireless comm. problem

Using beamforming to overcome the transmission loss however introduces new problems. As the signal is no longer omnidirectional, the beams from the terminals have to be aligned, or beamforming will result in worse performance. This gives rise to two problems if beamforming is used in communication systems, initial access, and tracking. Initial access is the problem of establishing a connection, with limited or no prior knowledge being available. Here a balance of performance and time is important, as a direction with high performance can always be found by an exhaustive search. However, this might be too time consuming, and as such other methods will most likely be needed. Tracking is the problem of keeping a connection, ie. after connection has been established, as such at least one prior direction where a connection is viable is known.

1.3 Current solutions

The problems mentioned above are not entirely novel, and as such several solutions have already been developed. As mentioned earlier, exhaustive search is a way to find beams with high performance. This is implemented by having both transmitter and receiver test all possible beam pairs in sequence. While this gives good performance, is does not scale very well as the amount of possible beams increases. This example is illustrated in a 2D case in Fig. 1.2.

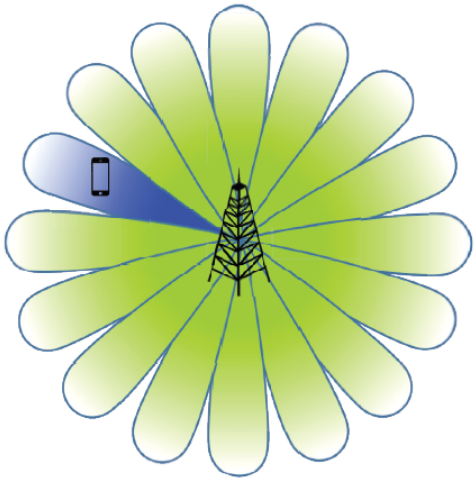


Figure 1.2: Beams in an exhaustive search case [8]

Another method is a hierarchical search, where beams of different width are available. By checking broader beams first, the general direction of highest signal strength can be found, and further search in that direction can be conducted with narrower beams. A downside of this method is that the broader beams has shorter range than the narrow ones, increasing the risk that the receiver is out of range, leading to wrong decisions. Fig. 1.3 illustrates this, where on the left broad beams are used to find the sector with highest signal strength, and the right shows that sector being more thoroughly searched. Both these methods implies that there is a finite set of beams to choose from. Such a set is known as a codebook, with the individual beams called codewords.

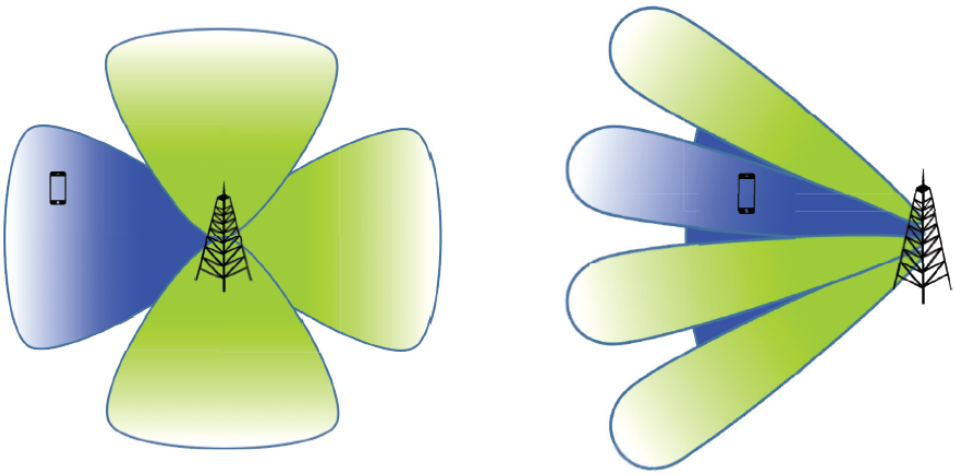


Figure 1.3: Beams in an hierarchical search case [8]

While these solutions work, they are rather simple and obvious, and aren't the forefront of beam alignment algorithms. A more advanced algorithm is the optimized two state search (OTSS) developed by Li et al.[9]. It operates on the principle of assuming a fixed time and power budget and allocating these resources differently from exhaustive search. While an exhaustive search allocate equal time and energy to each codeword in the codebook the OTSS conducts, as the name suggests, the search in two steps. The fist step is much like exhaustive search where all codewords are tested, however this is done with a lower power budget. The second stage then remeasures the most promising codewords to find the best and reduce error rate due to noise corruption of initial measurements. This method was shown to outperform both exhaustive and hierarchical search assuming equal energy budget. That is, it achieved the same performance as the other algorithms, but faster with a strict energy budget, and better performance if the budget was more lenient.

Another search method, not based on heuristics but on machine learning, is one developed by Rezaie [10]. This method utilizes context information such as position, feeds this to a deep neural network which proposes a couple of codewords, which are then measured and the optimal is chosen. This method utilizes machine learning to find and exploit structure in the problem which may not be apparent. This method was found superior to another data driven algorithm, a generalization of the inverse fingerprint algorithm.

1.4 Research question

While the last section presented a number of solutions, the problem is by no means perfectly solved, and therefore another method is explored in this report. As has been pointed out in [10] the propagation environment is not guaranteed to be stationary, which means it could be beneficial to use a technique capable of compensating for, or otherwise handle this dynamic environment. Preferably this technique should be able to adapt to its environment as it changes, without the need for manual tuning of an algorithm. As machine learning has proven well suited for this problem in the past, further research in this direction could prove beneficial. A machine learning method which is known for its capacity to train and and keep learning in an online manner, adapting to its environment, is Reinforcement Learning (RL). Its capability to learn indefinitely through continuous interaction with the environment as it changes, makes it something that in theory would be well suited for the problem of beam alignment and tracking. Thus the research question of this Master Thesis is as follows:

> Can Reinforcement Learning be implemented as a suitable solution for beam alignment and tracking in a mmWave communication system, consisting of both stationary and mobile units?

1.5 Problem scope

The research question has multiple solutions, and to ensure time, and budget constraints are kept the project will be delimited in this section.

Data collection can happen through a number of different channels. Data can be obtained by conduction measurements, simulating the measurements or by obtaining a third party's measurement/simulation data. Conducting real world measurements is outside the scope of this project, as the measurements themselves isn't the focus of this project, and real measurements are prone to being affected by unknown variables. Data from a third party is also prone to having not controlled for variables. Thus this project will gather data by simulation.

System model 2

As mentioned in Chapter 1, the data to be used in this project will be acquired by simulations. The problem is centered around the successful transfer of signals from a transmitter attached to a base station (BS) to a receiver attached to a user equipment (UE), when both are using beamforming. Here the BS is synonymous with the cell tower, and the UE synonymous with a mobile phone. As the alignment of beams heavily impacts the signal received, the variable of main interest is the received signal at the receiver terminal.

This chapter presents first the models chosen to describe the received signal, the beamformers and the wireless communication channel. Then follows a brief description of the simulation tool used for the actual calculations. Finally the different models used to determine the UE's movement through the environment are presented in detail.

2.1 Received Signal Power

To calculate the received signal power at the receiver, a model from [11] used. The model assumes a 2D multipath propagation scenario combined with the use of beamformers at both the BS and the UE. (2.1) calculates how much power is received at the receiving antenna based on transmitted power, chosen beam pair and a channel matrix.

$$\mathbf{R}[p,q] = \left\| \sqrt{P_t}\boldsymbol{w}_q^H \boldsymbol{H} \boldsymbol{f}_p s + \boldsymbol{w}_q^H \boldsymbol{n} \right\|^2 \tag{2.1}$$

where

p :	Beam index for transmitter antenna	$p \in \{0, \ldots, N_{b,t} - 1\}$
q :	Beam index for receiver antenna	$q \in \{0, \ldots, N_{b,r} - 1\}$
$N_{b,r}$:	Number of different beams for the receiver antenna	$N_{b,r} \in \mathbb{N}$
$N_{b,t}$:	Number of different beams for the transmitter antenna	$N_{b,t} \in \mathbb{N}$
P_t :	Transmission power	$P_t \in \mathbb{R}^+$
$\boldsymbol{w}_q$:	qth combiner from beamforming codebook	$\boldsymbol{w}_q \in \mathbb{C}^{N_r \times 1}$
$\boldsymbol{f}_p$:	pth precoder from beamforming codebook	$\boldsymbol{f}_p \in \mathbb{C}^{N_t \times 1}$
$\boldsymbol{H}$:	$N_r \times N_t$ channel matrix	$\mathbb{C}^{N_r \times N_t}$
N_r :	Number of receiver antennas	$N_r \in \mathbb{N}$
N_t :	Number of transmitter antennas	$N_t \in \mathbb{N}$
s :	Transmitted symbol with normalised power	$s \in \mathbb{C}$
$\boldsymbol{n}$:	Complex Gaussian noise vector	$\boldsymbol{n} \sim \mathcal{CN}(0, \sigma^2 \mathcal{I})$

The symbol can be set to 1 for simplicity, the combiners and precoders are to be defined from a pair of beamforming codebooks and a narrowband model is used for the channel matrix.

2.2 Beamformers

Beamforming is a form of signal processing applied to antenna arrays, where certain directivity patterns can be achieved. The patterns achievable depends on the number of antennas and their intrinsic directivity pattern. Beamforming is implemented by combining signals from or to the antenna array such that they constructively interfere with each other in certain directions. When beamforming only a single directivity pattern is obtainable at any one time. Many patterns are in theory obtainable with arbitrary beamforming, however doing this is costly, and as such a set of discrete beamformers are usually used.

As mentioned in Chapter 1, such a collection is called a codebook, and each individual beamformer is a codeword. Codebooks usually cover all angles to a greater or lesser extent, however how many codewords are used to do this differ, which means each codeword covers a different range of angles.

2.2.1 Analog and Digital Beamforming

2.2.2 Beamforming at mmWave frequencies

Using beamforming at mmWave frequencies presents some new challenges compared to lower frequencies. At lower frequencies it is possible to perform the signal processing before the signals are converted to analog, passed through the RF-chain and presented to the physical antennas. Thus the signal processing task is mostly a digital one and not much hardware is shared between the antenna elements. Fig. 2.1 illustrates this type of setup.

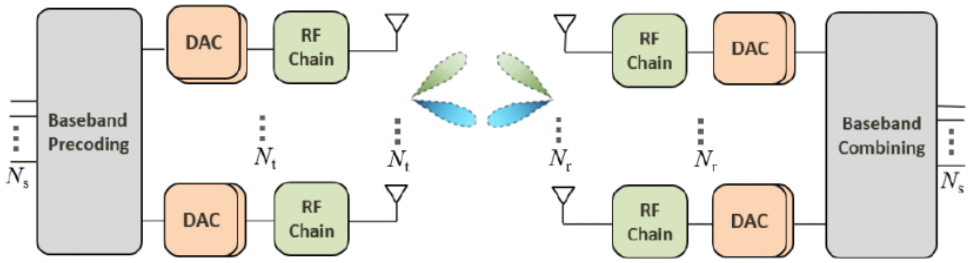


Figure 2.1: Conventional MIMO architecture at frequencies below 6 GHz [12].

In mmWave systems, the antenna elements are numerous, very small and placed very close to each other, making it infeasible to pack circuits for ADCs and the RF-chain for each element. Additionally, power consumption also becomes a problem for these circuits at very high sample rates [12].

Therefore, conventional designs are not feasible for mmWave systems and special designs have been developed that compliment the challenges and limitations present. One of the simplest methods is using analog beamforming, where a network of phase-shifters are connected to the antenna elements at one end and a single RF-chain at the other, as illustrated in Fig. 2.2.

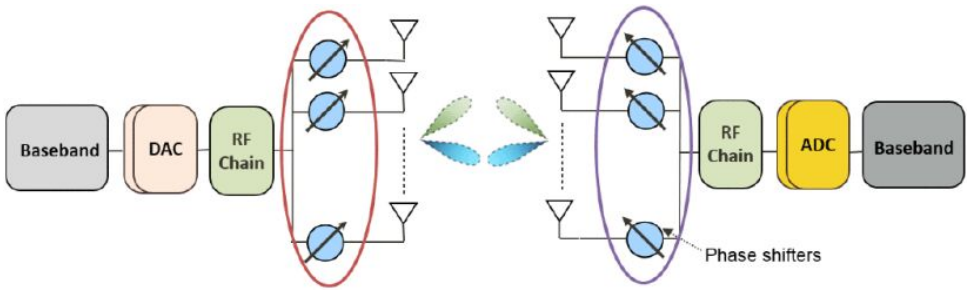


Figure 2.2: MmWave MIMO system using analog only beamforming [12].

The weights of the phase-shifters are then controlled individually to create the beam-patterns. By using analog beamforming, only a single signal can be carried at a time, meaning that multi-stream processing normally possible with MIMO systems are not available [12].

Another option is to use a hybrid architecture that combines analog and digital solutions. Here a number of RF-chains are used, although the number is less than the total number of antenna elements, and the RF-chains are then connected to an RF-precoding/combining stage as illustrated in Fig. 2.3.

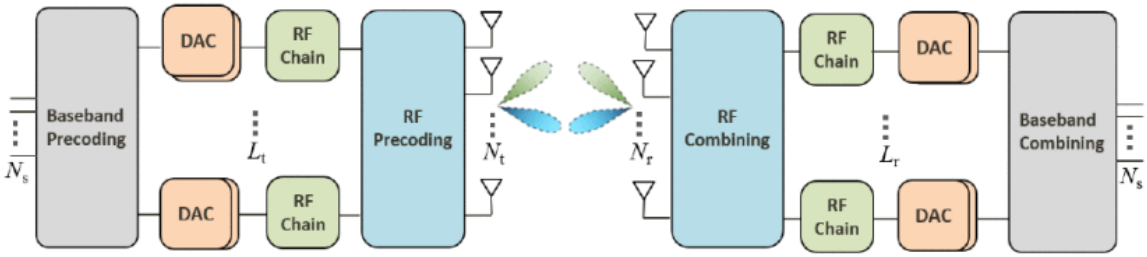


Figure 2.3: MIMO architecture at mmWave based on hybrid analog-digital precoding and combining [12].

The RF-precoding/combining stage can then be implemented using analog methods such as phase-shifters or switches. The hybrid method allows for some of the usual MIMO communication benefits, at the cost of extra hardware.

To keep the problem simple, only a single-user scenario is considered and as such there is no need for the MIMO features of the hybrid methods and thus this project is limited to the use of analog beamforming.

2.2.3 Hierarchical codebook

It is sometimes advantageous to have access to both broad and narrow beams, which is possible with a hierarchical codebook. Such codebooks contain multiple layers, referred to as codepages, with different numbers of beams of different width. For this project it has been chosen to use a hierarchical codebook for beamforming. The one implemented is described in the paper "Hierarchical Codebook Design for Beamforming Training in Millimeter-Wave Communication" [13]. This codebook consists of codewords created based on two criteria: 1) that the union of all codewords in a layer covers the entire angle domain, and 2) that the coverage of a codeword in one layer is covered by a union of the codewords in the next layer. It should be noted lower layers have broader beams, while higher layers have narrower beams. Furthermore the codewords are

designed to have a flat response over their coverage. This flatness of the gain is illustrated in Fig. 2.4.

The codebook is generated in such a way, that each codeword in a layer is covered by an integer positive number of codewords in the layer above. For this project it's chosen that each codeword gets covered by two narrower codewords on the layer above, which is illustrated on Fig. 2.5

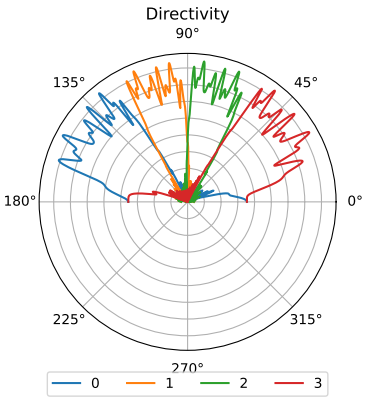


Figure 2.4: Beam pattern for four codewords in layer two, for beamformer with 128 antennas

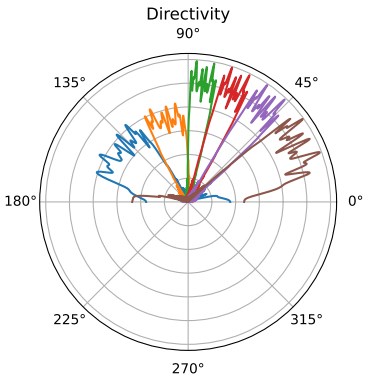


Figure 2.5: Beam pattern for six codewords, the two first, from the right, from layer two, the next four from layer three

The beam pattern plots in Fig. 2.4 and Fig. 2.5 are generated based on calculating the gain across the entire azimuth range, based on Eq. (2.2).

$$g(\theta, W) = |W^H \cdot a(\theta)| \tag{2.2}$$

g: Antenna gain in a given direction θ
θ: Azimuth angle
W: Codeword, a series of weights, one for each antenna
a: Array steering vector
a^H: Raised H denotes Hermitian transpose

While this codebook is hierarchical, a non-hierarchical codebook can be generated from it by simply picking out a codepage, and treating it as the entire codebook. The method for codebook generation from the paper is used for both receiver and transmitter. With this definition of $\boldsymbol{w}_q$ and $\boldsymbol{f}_p$, what is left to explain is the channel matrix, which is done in the next section.

2.3 Narrowband model

The last variable in the signal model is H, which is the channel matrix. For simplicity it has been chosen to use a narrowband channel model. It makes simulation easier and the principles used with wide-band channels are similar as well, meaning that results should not be entirely dependent on this choice.

The model is given by Eq. (2.3), based upon [11]. This model assumes that the antenna array at both the transmitter and the receiver is a Uniform Linear Array (ULA), with N_t and N_r array

elements respectively.

$$H = \sum_{l=0}^{L} \sqrt{N_r N_t}\, \beta_l\, \boldsymbol{a}_r\left(\theta_r^l\right) \boldsymbol{a}_t^H\left(\theta_t^l\right) \tag{2.3}$$

where

$L:$	Number of multi paths	$L \in \mathbb{N}$
$a_t:$	Steering vector for the transmitter antenna array	$a_t \in \mathbb{C}^{N_t \times 1}$
$a_r:$	Steering vector for the receiver antenna array	$a_r \in \mathbb{C}^{N_r \times 1}$
$\theta_t:$	Angle of Departure (AoD) $\theta \in [0; 2\pi)$	
$\theta_r:$	Angle of Arrival (AoA)	$\theta \in [0; 2\pi)$

Here β_l is the channel parameter for the lth multi path component and $\boldsymbol{a}_r$, $\boldsymbol{a}_t$ are the array steering vectors for the receiver and transmitter, respectively. θ_r^l and θ_t^l denotes the Angle of Arrival (AoA) and Angle of Departure (AoD) w.r to the ULA axes.

By assuming the antenna elements are spaced evenly by $\lambda/2$ and are all isotropic, the array steering vectors are defined as,

$$\boldsymbol{a}_r\left(\theta_r^l\right) = \frac{1}{\sqrt{N_r}}\left[1, e^{-j\pi\cos\left(\theta_r^l\right)}, \ldots, e^{-j\pi(N_r-1)\cos\left(\theta_r^l\right)}\right]^T \tag{2.4}$$

$$\boldsymbol{a}_t\left(\theta_t^l\right) = \frac{1}{\sqrt{N_t}}\left[1, e^{-j\pi\cos\left(\theta_t^l\right)}, \ldots, e^{-j\pi(N_t-1)\cos\left(\theta_t^l\right)}\right]^T \tag{2.5}$$

Which results in H being a $N_r \times N_t$ matrix. It should be noted that these array steering vectors only take a single angle as an argument, and therefor cannot be steering vectors for all directions in three dimensions. These are 2D steering vectors where the angle is in the plane, and the height is disregarded.

2.4 Antenna configuration

As mentioned in Section 2.3 the antenna configuration is limited to a ULA if the models assumptions should not be violated. While the antenna being a ULA specifies the placement of the antennas it doesn't dictate the number. Typical values for number of antennas can be seen in Table 2.1

Base station	32	64	128
Mobile terminal	8	16	32

Table 2.1: Typical number of antennas for BSs and UEs [14]

2.5 Radio Channel Simulation

There are a couple of requirements for the channel simulator, the first is the calculation of the channel parameters β_l. This requirement is fulfilled by all channel simulators, the second requirement is calculation of the AoA and AoD for each path. It is guaranteed to obtain meaningful AoA and AoDs' from geometrical models, one of which will be chosen. While not a technical requirement a balance between performance and accuracy is essential, as the project

is time limited. The model chosen, which fulfil all the requirements, is QuaDRiGa. QuaDRiGa runs through MATLAB or Octave and is used for "generating realistic radio channel impulse responses for system-level simulations of mobile radio networks" [15].

QuaDRiGa supports a variety of standardised channel models and simulation settings, for different radio network scenarios. In this project the scenarios "3GPP_38.901_UMi_LOS" and "3GPP_38.901_UMi_NLOS" are used for Line Of Sight (LOS) and Non Line Of Sight (NLOS) cases, respectively [16]. These scenarios supports carrier frequencies in the range 500 MHz to 100 GHz and for this project a carrier wave with center frequency of 28 GHz is used. The maximum transmit radius supported in the scenarios is 200 m and thus such an area surrounding the transmitter is chosen as the location used in the simulations. For all simulations, QuaDRiGa is set to simulate a multi path propagation situation, with scatterers placed randomly in the simulation area. The scatterers emulate objects in the area that reflect incoming signals, thus creating NLOS components.

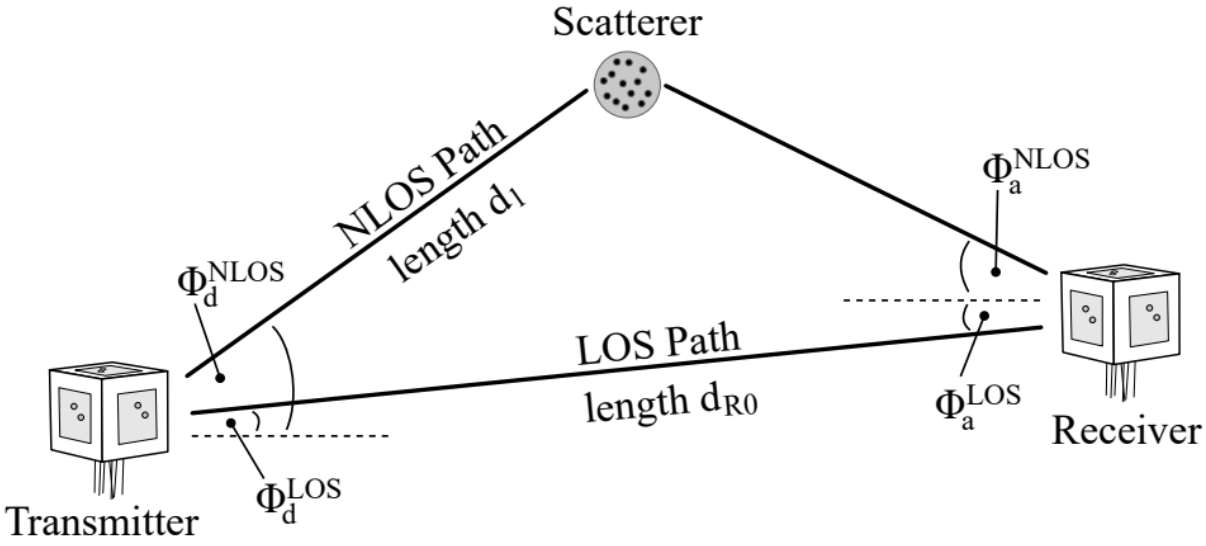


Figure 2.6: Simplified overview of the modelling approach used in QuaDRiGa [17].

The simulations return the complex path parameters, the AoA and the AoD for each multi path component at each position given by a track for each antenna terminal. Additionally, the movement direction is returned as an orientation parameter. In the simulations, both the transmitter and the receiver is set to consist of an isotropic antenna, such that the parameters only rely on the channel characteristics.

2.6 User Mobility Model

As mentioned earlier, Quadriga needs a number of inputs, one of which being a track of positions for each antenna terminal in the environment. Because it's intended that the UE is mobile, simulation of realistic movement patterns is required.

The mobility model used is based on an implementation of the model described in the paper "Smooth is better than sharp: a random mobility model for simulation of wireless networks" [18].

This model is described both for continuous and discrete time, the implementation for this project is for discrete time. The model is defined for a two dimensional space, where a unit walks around in the plane. The movement is based on random events which change the speed, v, and direction, ϕ, the unit is moving. Eq. (2.6) calculates the next position based on these factors, and the sampling interval Δt.

$$\begin{aligned} x_{n+1} &= x_n + cos(\phi) \cdot v \cdot \Delta t \\ y_{n+1} &= y_n + sin(\phi) \cdot v \cdot \Delta t \end{aligned} \tag{2.6}$$

How v is calculated is described in the next section, and ϕ in Section 2.6.2.

2.6.1 Speed

The speed at any given point is calculated based on seven inputs:

- vchange $[s]$, mean time between changing target speed
- vmax $[m/s]$, maximum speed
- vmin $[m/s]$, minimum speed
- vpref $[m/s]$, list of preferred speeds
- pvpref $[-]$, list of probabilities associated with preferred speeds
- acc_max $[m/s^2]$, maximum acceleration
- dec_max $[m/s^2]$, maximum deceleration

A specific parameter set belongs to a class of mobile units, like pedestrians or cars driving in an urban environment. The speed changes when a speed change event occurs, which happens with the probability given in Eq. (2.7), each time step.

$$p(\text{speed change event}) = \frac{\Delta t}{vchange} \tag{2.7}$$

When such an event occurs, a new target speed will be found, which is based on the probabilities given in pvpref, in the manner presented in Eq. (2.8).

$$p(v_{target}) = \begin{cases} p(v_{target} = \text{vpref}_1) & = \text{pvpref}_1 \\ p(v_{target} = \text{vpref}_2) & = \text{pvpref}_2 \\ & \vdots \\ p(v_{target} = \text{vpref}_N) & = \text{pvpref}_N \\ p(v_{target} = \mathcal{U}[\text{vmin,vmax}]) & = 1 - \sum_{i=1}^{n} \text{pvpref}_i \end{cases} \tag{2.8}$$

After a target speed is chosen an acceleration or deceleration is drawn. The probability distribution of the acceleration is given in Eq. (2.9). The deceleration is closely related, with the difference being the uniform distribution is for the interval [dec_max, 0].

$$p(acceleration) = \mathcal{U}[0,\text{acc\_max}] \tag{2.9}$$

The velocity is changed each time step to accelerate/decelerate to the target velocity, according to Eq. (2.10).

$$velocity_{n+1} = velocity_n + acceleration_n \cdot \Delta t \tag{2.10}$$

The velocity keep increasing/decreasing until the target speed is reached, in which case the acceleration/deceleration is changed to zero.

2.6.2 Direction

The direction at any given point is calculated based on three inputs:

- dirchange [s], mean time between changing target direction
- curvetime_max [Δt], maximum curve time
- curvetime_min [Δt], minimum curve time

In a similar manner to a speed change, a direction change is a random event with a given chance to happen at each time step, which is given by Eq. (2.11).

$$p(\text{direction change event}) = \frac{\Delta t}{dirchange} \tag{2.11}$$

If such an event occurs a target direction, dir_{target}, is drawn with the distribution $\mathcal{U}[-\pi,\pi]$. Then a curve time is drawn, which is the time from the turn is initiated to the unit is pointing in the target direction, in time steps. This time is drawn as given in Eq. (2.12).

$$p(\text{curvetime}) \sim \lfloor \frac{(\mathcal{U}[0,1] \cdot (\text{curve\_max} - \text{curve\_min}) + \text{curve\_min})}{\Delta t} \rfloor \tag{2.12}$$

2.6.3 Speed and direction correlation

In the previous sections the behavior of the speed and direction were described as independent events. According to the model this isn't generally the case [18]. They are interdependent, and two scenarios which demonstrate these dependencies are integrated as part of the model. The first scenario is the "stop-turn-and-go", where the direction changes are correlated with speed changes. If the mobile unit decelerates to a stop, and after some time accelerates again, a direction change is likely to also occur, such as stopping and driving through an intersection. This scenario is described by the parameter "stop_dirchange [−]". This is the chance to enter the "stop-turn-and-go" scenario, whenever the velocity of the unit reaches zero. If this scenario is entered there's a 67% chance to not incur a direction change, and 33% to incur. If a direction change is incurred there's a 50% chance to turn 90° left, and 50% to turn right. These probabilities are expressed in Eq. (2.13).

$$\begin{aligned} p(\text{stop-turn-and-go}|v=0) &= stop\_dirchange \\ p(dir_{target} = dir|\text{stop-turn-and-go}) &= 0.67 \\ p(dir_{target} = dir + 90°|dir_{target} \neq dir) &= 0.5 \\ p(dir_{target} = dir - 90°|dir_{target} \neq dir) &= 0.5 \end{aligned} \tag{2.13}$$

The second scenario describes a speed dependency on direction changes. When changing direction, there is a chance to decelerate to a calculated maximum speed, to emulate safe navigation of corners. The maximum speed at which corners can be navigated is calculated based on the friction coefficient, static_friction[−], as done in Eq. (2.14)

$$v_{curve} = \sqrt{\mu_s \cdot g \cdot r_c} \tag{2.14}$$

Where g is the gravitational acceleration, and r_c is the curve radius, which is calculated by:

$$r_c = \frac{v_{target} \cdot curvetime \cdot \Delta t}{|\Delta\phi|} \tag{2.15}$$

When the curve speed has been calculated, a check is performed to gauge if deceleration is necessary. If so, the unit slows down until the curve speed, v_{curve}, is reached and the actual turn is performed.

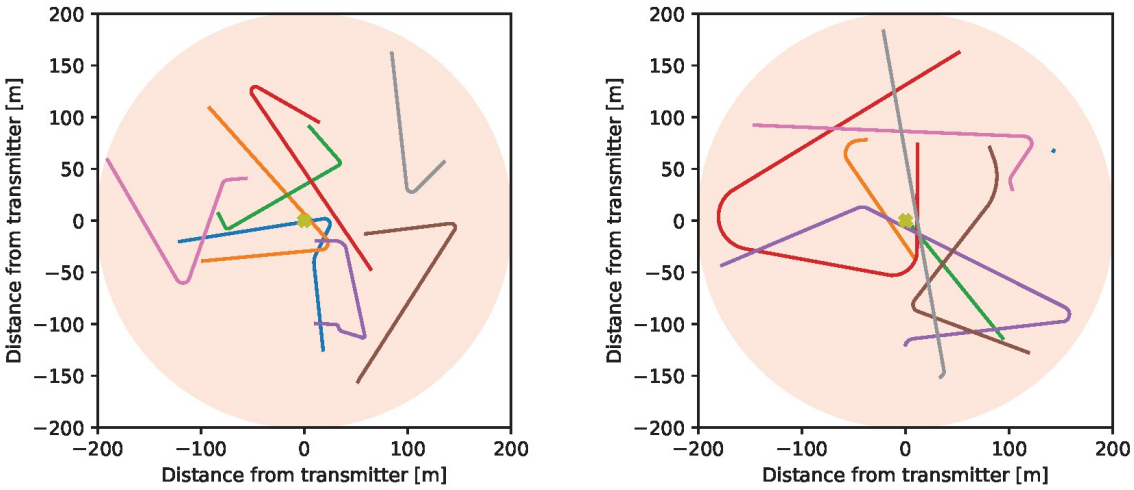
2.6.4 Mobility case settings

A multitude of different parameters have been mentioned throughout Section 2.6 which differ depending on which mobile unit the model is emulating. In this project, two different cases have been identified as interesting and representative for data gathering. The first represents a pedestrian in an urban environment and the second represents a car in an urban environment. Parameters describing the behavior of both cases can be found in Table 2.2.

	Pedestrian	Urban car
vmax $[m/s]$	2	14
vmin $[m/s]$	0	0
vpref $[m/s]$	[0, 0.7, 1.4]	[0, 8.3, 14]
pvpref [−]	[0.1, 0.1, 0.4]	[0.3, 0.1, 0.4]
vchange $[s]$	100	50
dirchange $[s]$	100	50
stop_dirchange [−]	0.33	0.33
acc_max $[m/s^2]$	1.44	2.5
dec_max $[m/s^2]$	0.6	4
curvetime_max $[s]$	10	10
curvetime_min $[s]$	5	1
static_friction [−]	1	0.7

Table 2.2: Movement parameters of two different moving units

The values in Table 2.2 are derived from [18], [19] and [20, §42]. Examples of mobility traces for both cases are illustrated on Fig. 2.7.



(a) Urban pedestrian scenario.

(b) Urban car scenario.

Figure 2.7: plots of seven random mobility traces made using by the mobility model parameters from Table 2.2. The BS is placed at the origo, and the colored circle represents the range of the BS.

The pedestrian traces are in general shorter than the car traces, and the turn radii are larger for the car due to the higher travel speeds.

2.7 User Orientation Model

In addition to a mobility model for the UE, a model for the orientation of the UE is developed. From the QuaDRiGa simulation the orientation of the UE at a given position is given simply as the direction of movement from the last position to the current one. While this is sufficiently realistic for the case of a car moving with the terminal fixed to the chassis, in the case of a pedestrian the orientation of the terminal is expected to vary slightly even when travelling along a straight line. As an example, consider how a phone in the hand of a person might change orientation, even if the person is walking straight ahead.

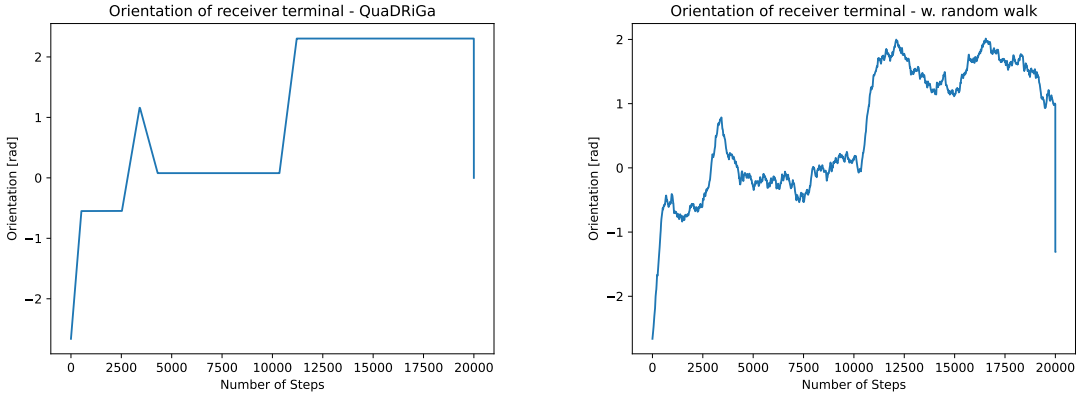
To better represent the random behaviour of the orientation in the pedestrian case, a filtered random walk is added to the orientation supplied by QuaDRiGa. The random walk is modelled after the walk proposed in [21, p. 51840].

The added random walk is a time series calculated by:

$$\bar{\alpha}_t = \bar{\alpha}_{t-1} + \mathcal{N}(0, \sigma^2) \tag{2.16}$$

where the start value $\bar{a}_0 = 0$. The filter used to smooth the random walk is a simple k'th-order moving average filter:

$$\alpha_t = \frac{1}{K} \sum_{k=0}^{K-1} \bar{\alpha}_{t-k} \tag{2.17}$$



(a) Without added filtered random walk **(b)** With added filtered random walk

Figure 2.8: Orientation of receiver terminal in pedestrian case, without and with added filtered random walk

The smoothness of the filtered walk, can be adjusted with the value of K, where a larger K gives a slower variation in orientation. The output from the filter is then added to the orientation parameter from QuaDRiGa and wrapped to be within [-π, π] to keep the orientation in absolute angles. An example of the orientation before and after the added random walk can be seen on figure Fig. 2.8a and Fig. 2.8b respectively.

2.8 Coordinate Transformation

When using beamforming, the direction of the beam is relative to the fixed coordinate system of the antenna array and not the global coordinate system in which the terminal moves around. When determining the steering vectors used in the channel matrix, the local angles are needed, but only the global angles are supplied from the QuaDRiga simulation. Global AoA can be converted to local AoA given the orientation of the UE. This is done by subtracting the global orientation from the global AoA. However, the angles used for steering vector are relative to the positive y-axis and not the x-axis, thus an offset $\pi/2$ is added to get the final local angle. This is also described by equation Eq. (2.18).

$$AoA_{local} = \frac{\pi}{2} + AoA_{global} - Orientation_{global} \tag{2.18}$$

The conversion is illustrated in Fig. 2.9, where the black arrow is the orientation of the antenna, and the blue arrow is the angle of arrival of the signal.

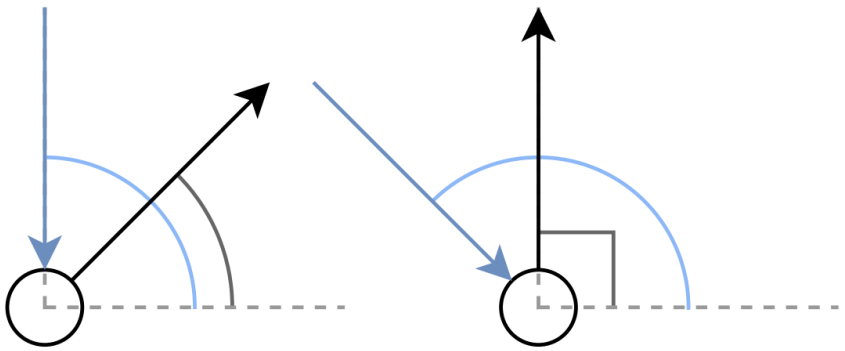


Figure 2.9: Orientation changes from global to local

Throughout this chapter, the system model was defined, and explained, which is one of two important precursors to developing solutions to the problem, the other being RL, which is described in the next chapter, Chapter 3.

Reinforcement Learning Methods 3

In order to design a RL scheme for the specific task at hand, first a study of the general principles and methods of RL is conducted.

In this chapter the concepts and central elements of RL are briefly introduced. Then, a small selection of popular methods for implementing RL is presented.

3.1 The RL-framework

The purpose of RL is to learn some desired behaviour from interaction with a system. This interaction can be seen as taking different actions when presented with different situations. To better develop and analyze methods and algorithms a solid mathematical framework is an invaluable tool.

This kind of interaction is precisely modelled by Markov decision processes (MDPs), as these involve going from one state to another based on an action and receiving some reward, based on the state and choice of action.

Typically an MDP is represented by a 4-tuple $(\mathcal{S}, \mathcal{A}, P_a, R_a)$, where:

- $\mathcal{S}$: is a set of possible states called the state space.
- $\mathcal{A}$: is a set of possible actions called the action space, also seen as $\mathcal{A}(s)$, the set of possible actions when in state s.
- $P_a(s'|s, a)$: is the probability of transitioning from state s to state s' when taking action a.
- $R_a(s, s', a)$: is the reward function denoting the reward received when transitioning from state s to state s' having taken action a.

3.1.1 The Agent-Environment interaction

Using MDPs the RL problem is modelled as two systems interacting with each other. The first is known as the agent, which represent the entity that learns and take actions. The other is presented as the environment, which represent all that which is not part of the agent itself. Thus, the environment consists of all that which the agent can readily observe and all it cannot.

The environment presents the agent with a situation, called a state, and the agent then decides on an action, which the environment responds to with a new state and a reward, as illustrated in Fig. 3.1.

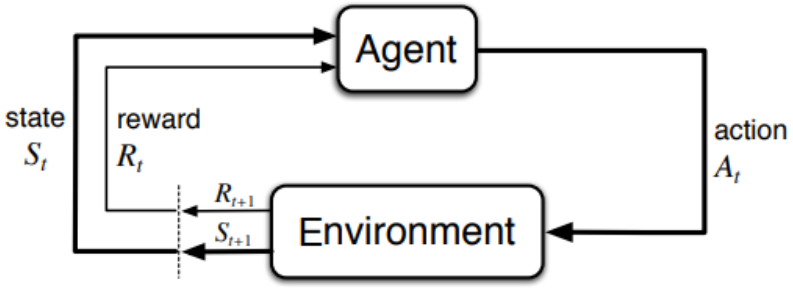


Figure 3.1: The Agent-Environment interaction model in an MDP.

This process then repeats indefinitely or until some terminal state is reached. In the simple case, the agent and environment is said to interact at discrete moments in time, $t = 0, 1, 2, 3, ...$ At each time step, t, the agent receives a state from the environment, $S_t \in \mathcal{S}$ and takes an action, $A_t \in \mathcal{A}(s)$. This action presumably has some impact on the state of the environment, and at the next time step the environment responds with this new state, S_{t+1}, and a reward, $R_{t+1} \in \mathbb{R}$.

In the special case where the sets of states, actions and rewards are all finite sets, the MDP is called finite, and it has well defined discrete probabilistic characteristics. These probabilities can be collected into a single function p:

$$p(s', r|s, a) = P(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a), \tag{3.1}$$

for all $s', s \in \mathcal{S}$, $r \in \mathbb{R}$ and $a \in \mathcal{A}(s)$. Here the '|' is borrowed from the usual notation for conditional probability to indicate the Markov nature of the MDP. The Markov property means that the transition probabilities described by p are only dependent on the previous state and action and entirely independent of any states and actions prior to these.

The probabilistic characteristics are known as the dynamics of the MDP, and based on these it is possible to find optimal solutions by solving sets of equations known as the Bellman equations [22, p. 74].

The derivation and solving of these equations is out of scope in this report. Additionally, in most of the cases of practical interest no such solution exists as the strict assumptions of MDPs are no longer valid and approximations are used instead. However, even though most of the concepts and methods from RL are restricted to MDPs, they have proved to apply more generally as well.

This really shows how powerful a tool RL can be. Instead of needing valuable expertise to create approximate models of a given problem and environment and then using further resources to find a policy that works well with this model, RL can be used. By placing agents into the environment and letting them interact with each other through actions and rewards, it is possible to learn useful policies without any models at all. In some cases, the policy found through interaction with the environment may even be very close to the optimal policy usually only obtainable through solving the Bellman equations.

3.1.2 States and Actions

The MDP framework allows for very flexible notions of states and actions based on the problem at hand. The state is some representation of how the environment is at the current time and can

take almost any form. It can be something simple and objective like measurements from sensors or a history of such data, but it can also be more abstract and subjective like not knowing if a sensor is working or not. But importantly, the state can only consist of information that is directly observable by the agent at the current time, meaning that the state may not be a complete and exact representation of the environment at that time. Likewise, actions can be more or less abstract, like which way to turn at an intersection or changing ones behavior from careful to risky.

In general there is no best way to represent states and actions, it all depends on the task at hand. However, care should be taken when designing the state and action spaces, as they have a large impact on how well the RL algorithms work.

3.1.3 The goal of the agent; Reward and Return

Another important part of the design process is the reward function, as it is central to the goal the agent tries to achieve. In simple terms the goal of the agent is always to maximize the amount of reward it gets, not just immediately but also further into the future, i.e. the cumulative reward over time. Thus, the reward function directly controls what kind of behaviour the agent learns to follow. Similar to the state, the reward is a random variable presented to the agent from the environment as a reaction to the previous action. This means that the reward also has to be directly observable from the environment by the agent.

Rewards can be used in many ways to guide the agent towards learning the designers goals, usually by giving positive or negative rewards to either encourage or discourage certain behaviour. However, care should be taken when designing the reward function, as it is important that the reward does not tell the agent how to learn, eg. by achieving subgoals, but only what to learn.

As an example of a poorly designed reward function, imagine a cleaning robot tasked with removing dirt from an office, that is rewarded based on the amount of dirt it removes. The agent in this case might learn to knock over plants in the office so it has more to clean and thus receive more reward, instead of simply cleaning the initial dirt as intended.

To further formalize the goal of the agent, it helps to introduce the concept of return. As stated earlier, the agent strives to maximize not immediate reward, but the cumulative reward long into the future. The return is precisely this cumulative reward, denoted as G_t. In cases where time steps continue indefinitely, the return might go to infinity if the individual rewards are all positive.

To counter this, often the notion of discounted return is utilized instead. The discounted return is defined as:

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{3.2}$$

where T is the last time step, which might be infinity, and $\gamma \in [0, 1]$ is the discounting factor. To avoid infinite returns, the discounting factor must be less than 1 for MDPs with episodes of infinite steps. Besides solving the problem of infinite returns, the discounting factor gives the possibility to control how farsighted the agent is regarding future rewards. With a large discounting factor, it takes longer for the weight γ^{k-t-1} to become negligible and thus the future rewards contribute to the return for a longer time.

3.1.4 Value- and Policy Functions

When determining how good a state is to be in, or how good it is to take a given action in a given state, the agent uses value functions. As the agent wants to maximize the cumulative rewards it is tempting to define the value as the return from a given state. However, as the reward received by taking a given action in a given state is a random variable, governed by the dynamics of the MDP, the same action may not give the same reward when taken in the same state consecutive times. Thus the value function is defined as the expected return instead.

The rewards the agent can expect to receive depends on how the agent behaves, thus a value function is always defined relative to a specific behaviour, called a policy. A policy is a function, $\pi(a|s)$, that maps a state to probabilities of taking each possible action in that state. The goal of RL methods are to specify how to update an agent's policy, based on its experiences.

The state-value function of a state s, using the policy π, is the expected return when starting in s and then following π from that point on. This is denoted by $v_\pi(s)$ and for MDPs the function can be defined as:

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right], \text{ for all } s \in \mathcal{S} \tag{3.3}$$

where $\mathbb{E}_\pi [\cdot]$ is the expectation of a random variable, given that the agent follows the policy π. Furthermore, the value for a terminating state, if it exists, is defined to be zero. In a similar way an action-value function, for the value of starting in a state s, taking action a and then following the policy π, is defined as:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right] \tag{3.4}$$

3.2 Value approximation

Given a finite MDP with known dynamics, it is possible to find an optimal policy and value function using dynamic programming and the Bellman equations. However, as this is rarely the case, many practical reinforcement methods instead try to approximate the optimal value function and update the policy based on the approximations instead.

In cases where the state-space is sufficiently small, it may be feasible to keep a table containing the value estimates for all state-values, or even action-values. The entries in the table are then updated based on experience. In cases where the state-space is very large, or even continuous, it becomes unfeasible to keep a table of all the possible states. In such cases it can be better to use function approximation methods to directly approximate the value or policy functions.

3.2.1 Tabular methods

When trying to solve finite MDPs with tabular methods, there are three methods which are considered fundamental: dynamic programming, Monte Carlo methods and temporal-difference learning. Dynamic programming is a very well developed mathematical field, however it requires full knowledge of the dynamics of the MDP. Monte Carlo methods do not have this limitation, however they are not very suited for online, step-by-step computations. Temporal-difference learning has neither of these limitations, at the cost of being harder to analyze [22, p. 23].

Due to their widespread use, only the temporal-difference methods are described in this report. Likewise, only a select few methods are described in detail.

In the tabular case, the agent's policy is updated based on the value estimates at each time step. A popular choice is to estimate the action-value function, as it does not require any knowledge of the dynamics. An often used type of policy is the greedy, or the closely related ϵ-greedy, policy. Under the greedy policy, at each time step the agent takes the action with the highest estimated action-value associated with it. The greedy policy is given by:

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, q_\pi(s,a) \tag{3.5}$$

The greedy policy is easily implemented when a table of action-values is available, as the maximisation can be done directly on the entries in the table. For the ϵ-greedy policy, the agent takes the greedy action with probability $(1-\epsilon)$ and a random action otherwise. The ϵ-greedy policy forces the agent to explore its environment to get better estimates of all action-values.

By always taking actions based on the current estimates in the action-value table, each time the estimates in the table improves so does the policy. Thus, the task is how to update the entries in the table based on the experience, such that the value estimates get better. Mutual for the upcoming methods is a general update-rule:

$$NewValue = OldValue + stepsize[Target - OldValue] \tag{3.6}$$

with the difference between the methods being what they use as target. The target is supposed to be something which the value estimate should go towards. Based on (3.4), an intuitive choice of target would be the expected return or an estimate of it.

SARSA

Indeed, this is also the case when using a method called SARSA, which utilizes the fact that (3.4) can be written recursively as

$$\begin{aligned} q_\pi(s,a) &\doteq \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi\left(S_{t+1}, A_{t+1}\right) \mid S_t = s, A_t = a\right] \end{aligned} \tag{3.7}$$

The SARSA update-rule uses an estimate of the expression in (3.7) as its target. Instead of the actual action-value for the next state and action $q_\pi\left(S_{t+1}, A_{t+1}\right)$, it uses the current estimate and instead of the expected value, a single sample is used. Thus the SARSA update-rule, under the policy π, can be written as:

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\left[R_{t+1} + \gamma Q\left(S_{t+1}, A_{t+1}\right) - Q\left(S_t, A_t\right)\right] \tag{3.8}$$

where $Q(s,a)$ is the estimated action-value for the state-action pair (s,a) and A_{t+1} is the next action taken when following the policy π. This update is performed after each transition from a non-terminal state S_t to the S_{t+1}. SARSA is illustrated in pseudocode in Algorithm 1.

Algorithm 1: SARSA update rule [22][p. 130]

Input: $\alpha \in (0,1]$, $\epsilon > 0$
Initialize $Q(s,a)$, for all $\mathbf{s} \in \mathcal{S}^+$,$\mathbf{a} \in \mathcal{A}(\mathbf{s})$, arbitrarily except that $Q(terminal, \cdot) = 0$
for *each episode* **do**
  Initialize S Choose A from S using policy derived from Q
  for *each step in episode* **do**
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,a)]$
    $S \leftarrow S'; A \leftarrow A'$
  end
  Until S is terminal
end

The SARSA method is called an on-policy method, as the policy used in the target is the same as the one used by the agent to choose actions. Thus by using this method, the action-value estimate converges not to the optimal action-value, but rather a sub-optimal action-value function that takes exploration into account. The learned policy is therefore more risk averse, as it knows that sometimes a random action is performed.

Q-learning

A method that instead directly estimates the optimal action-value function is Q-learning. The update-rule for Q-learning is similar to SARSA, however the policy used in the target is always the greedy policy, regardless of which policy is followed by the agent, thus making it an off-policy method. The Q-learning update-rule is defined by

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha \left[R_{t+1} + \gamma \max_a Q\left(S_{t+1}, a\right) - Q\left(S_t, A_t\right)\right]. \tag{3.9}$$

Q-learning is illustrated in pseudocode in Algorithm 2.

Algorithm 2: Q-learning update rule [22][p. 131]

Input: $\alpha \in (0,1]$, $\epsilon > 0$
Initialize $Q(s,a)$, for all $\mathbf{s} \in \mathcal{S}^+$,$\mathbf{a} \in \mathcal{A}(\mathbf{s})$, arbitrarily except that $Q(terminal, \cdot) = 0$
for *each episode* **do**
  Initialize S **for** *each step in episode* **do**
    Choose A from S using policy derived from Q Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha \left[R + \gamma \max_a Q(S',a) - Q(S,a)\right]$
    $S \leftarrow S'; A \leftarrow A'$
  end
  Until S is terminal
end

While the action-value function learned by Q-learning corresponds to a policy different to that followed by the agent, the policy is still important. As the policy controls which state-action pairs that gets visited, and thus which action-values that get updated, it is important to choose a policy that explores the state-action space.

However, even though it converges to the optimal action-values, an agent using Q-learning might have worse online performance than one following SARSA. This is because the policy learned by SARSA is tuned to give high return assuming that the agent explores, while the Q-learning policy is tuned to give high return when the agent behaves greedily.

3.2.2 Approximate methods

When tabular methods prove infeasible, the domain of function approximation may be of use instead. This happens in cases where the state-space becomes so large that it cannot fit easily into a table and the probability that the agent encounters entirely new states after each transition is high. Here it is desired that the knowledge the agent learns from visiting one state also generalizes to other states that might not have been visited yet.

This is a goal known from supervised learning, a field which has been very thoroughly studied and has many well-defined methods on its own. Fortunately many of the methods from supervised learning can be modified to fit the problem of function approximation in RL.

There exists both parameterized and non-parameterized methods for function approximation. Due to time and resource limitations, non-parameterized methods are put outside the scope of this report. In parameterized methods the approximated value function is represented by a vector of weights, $\boldsymbol{w} \in \mathbb{R}^d$. Thus the state-value, is retrieved not by look-up in a table, but by evaluating the function $\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s)$. As with the tabular methods the purpose of the function approximation methods is to make the estimations of the values more accurate based on experience. This is done by modifying the weights, usually using some sort of gradient method.

Stochastic-Gradient and Semi-Gradient Methods

In short, gradient methods are used to minimize an objective function relative to one of its parameters, by moving the parameter in the direction of the negative gradient of the function at the current parameter values. This depends on the objective function being differentiable with relation to the parameter. When the update of the parameter happens based on a single sample, maybe drawn stochasticly, and not the entire data set it is called stochastic gradient descent. Due to the online nature of RL, stochastic gradient methods are especially suitable.

In the case of value function approximation, a useful objective function to minimize is the sample value error between the actual value $v_\pi(S_t)$ at state S_t and the approximation $\hat{v}(s, \boldsymbol{w_t})$ using the current parameters. The parameter update-rule at each time step t is the defined as:

$$\begin{aligned} \boldsymbol{w}_{t+1} &\doteq \boldsymbol{w}_t - \frac{1}{2}\alpha\nabla \left[v_\pi\left(S_t\right) - \hat{v}\left(S_t, \boldsymbol{w}_t\right)\right]^2 \\ &= \boldsymbol{w}_t + \alpha\left[v_\pi\left(S_t\right) - \hat{v}\left(S_t, \boldsymbol{w}_t\right)\right]\nabla\hat{v}\left(S_t, \boldsymbol{w}_t\right) \end{aligned} \tag{3.10}$$

where α is a positive step-size parameter and ∇ represents the gradient with respect to $\boldsymbol{w}$. In the more general case where $v_\pi(s)$ is not available, another target can be used instead, e.g the update targets known from the tabular methods. However, in the case of update targets used in temporal-difference, the target depends on the approximated value function and the equation in (3.10) is no longer valid. The use of (3.10) is then no longer a case of gradient descent, but semi-gradient descent and convergence is not as robust in general, but in some cases it has been proved to still converge reliably. One such case is when $\hat{v}(s, \boldsymbol{w_t})$ is a linear function of $\boldsymbol{w}$ [22, Chapter 9].

Linear Methods

When using linear methods, the value function is approximated by the inner product between the weight vector $\boldsymbol{w}$ and a vector $\boldsymbol{x}(s) \doteq (x_1(s), x_2(s), \ldots, x_d(s))^\top \in \mathbb{R}^d$:

$$\hat{v}(s, \boldsymbol{w}) \doteq \boldsymbol{w}^\top \boldsymbol{x}(s) \doteq \sum_{i=1}^{d} w_i x_i(s) \tag{3.11}$$

The vector $\boldsymbol{x}(s)$ is called a feature vector representing the state s, where the value of each component $x_i(s)$ is known as a feature of s. Using a linear function, the update in the stochastic gradient and semi-gradient methods becomes especially simple:

$$\boldsymbol{w}_{t+1} \doteq \boldsymbol{w}_t + \alpha \left[U_t - \hat{v}\left(S_t, \boldsymbol{w}_t\right) \right] \boldsymbol{x}\left(S_t\right) \tag{3.12}$$

where U_t is some target function.

Some of the advantages of using linear methods are their favourable convergence characteristics and their efficiency. [22, p. 210]. However, in order for the methods to give useful results it is important that the features are chosen well. The features are used by the designer to add important prior knowledge about the RL task at hand, such as ways it might be relevant to generalize across the state space.

There is no right or wrong way to choose features, as it depends on the RL problem at hand, but there exists some different general methods for constructing features that can be used as frameworks [22, pp. 210–222].

Non-linear functions

Another widely used method for function approximation in RL is the use of artificial neural networks, ANN. ANNs are used to approximate non-linear functions and a lot of work has been done in the field of machine learning on the use of ANNs, especially the case of networks with many hidden layers known as deep learning.

There exists many different types of ANNs, however in this overview only the classic feedforward will be discussed as it illustrates well the general principles of ANNs. The feedforward ANN consist of a number of units, grouped together in a number of layers, such that the outputs from the units in one layer only feeds into the next layer. An example of a simple ANN is shown in figure 3.2.

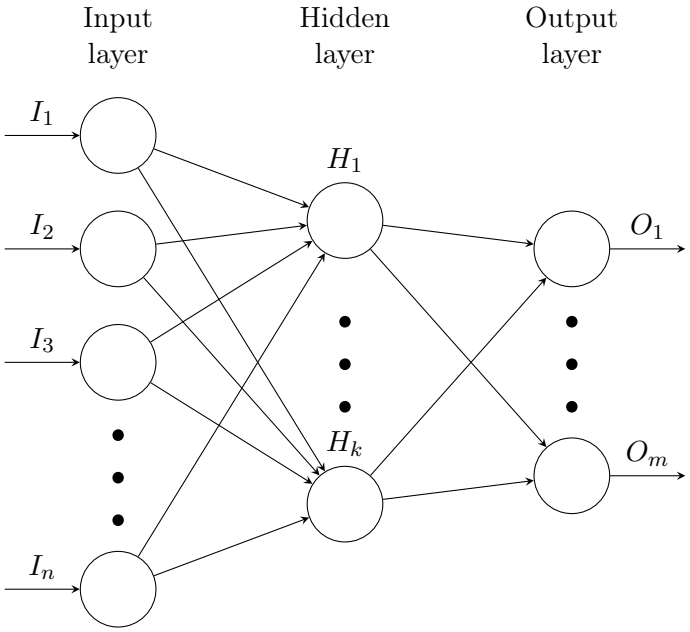


Figure 3.2: A simple feedforward ANN with n input units, m output units and a single hidden layer with k units.

Each unit in an ANN computes a weighted sum of all its inputs and passes this sum through a non-linear function called an activation function and then outputs the result to the units in the next layer. A variety of different activation functions can be used for different tasks, but some popular choices are logistic Sigmoid functions, ReLU and softmax [23].

Like the linear methods, the neural network is parameterized by its weights as well, and during training the weights are updated based on a stochastic gradient method.

3.2.3 Tabular or Approximate Methods

Whether tabular methods or function approximation is the appropriate choice for an agent, largely depends on how large the state-action space is. Another contributing factor is how stationary the environment is, i.e. how fast the true value of a state changes over time. In cases with small state-action spaces and sufficiently stationary environments it may be possible to use tabular methods, which demands fewer computation resources than function approximation methods. In other words, if your state-action space is small enough to fit in a table in memory, and it is possible to visit all relevant state-action pairs multiple times in a relatively short time, then tabular methods offer guarantees of convergence that function approximation methods cannot.

However, in more complex cases where the state-action space grows large it may be infeasible to keep a record of each state-action pair and its value. Here it may be better to use function approximation methods, where only a number of weights, less than the number of state-action pairs, are needed. The reduction in memory usage comes at the cost of computation, which might not be feasible on small devices. Thus both the size of the state-action space and the intended device used has an impact on the choice between tabular and approximate methods.

RL in Adaptive Beamforming 4

The problem was analyzed and clarified in Chapter 1, the modelling of the problem presented in Chapter 2, and the principle of RL explained in Chapter 3. In this chapter, the different aspects of applying the RL framework to the problem at hand are explored.

When using RL to solve a problem, the general framework described in Chapter 3 has to be configured to the problem. Reviewing the framework it becomes clear that it can be divided into three parts: The agent, the environment and the signals between them. In order to apply RL each part of the RL framework has to be appropriately defined for the wireless communication problem. The three parts can't be designed separately, as design choices made for one part also influences the others, making the design an iterative process. In the following sections some of the design considerations for each part are described.

4.1 The modelling of the agent

The agent is the part that interacts with the environment through its actions, trying at all times to optimize the total amount of reward it receives. The choices made when designing the agent can impact both how the environment is defined and the interaction between the agent and the environment. Some of the major choices are listed and discussed below.

4.1.1 Single vs. Multi Agent RL

First there is the high level abstraction of deciding where to put the agent relative to the environment. One way is to have each antenna array in the network controlled by a separate agent. This way, each UE and the BS has its own agent, and the actions control only the beamforming happening at one device. From the perspective of each separate agent, the other agents then become parts of the environment, their actions beyond the control of the agent and information about them is only accessible through the combined state of the environment. The agents may thus possibly have information about each other through their state, but keep separate value estimates, update rules and hyper-parameters such as discounting factor, step size and ϵ value when using an ϵ-greedy policy.

Another way is to have a single agent control all the devices in the network. Here the actions control the beamforming performed at all the antenna arrays at once. Then at each time step, the agent chooses a codeword for each device. In this case the agent has more information available, as all devices have been removed from the maybe only partially observable environment and become a part of the agent.

The case of a single agent interacting with the environment most closely resembles the framework presented in Chapter 3, with the multi agent case being somewhat more complicated although

many similarities exists. While an extensive exploration of Multi Agent RL is outside the scope of this report, a brief introduction to some of the main concepts seems appropriate. In multi agent RL, two or more agents are placed in the same environment, tasked with either competing with each other to gain the most of some limited resource or cooperating with each other to maximise the total reward of all agents.

When modelling multi agent RL an extension of the MDP framework can be used. For the case of multiple agents that move through multiple states, the term multiplayer stochastic games is used. Like MDPs, stochastic games can be described by a tuple consisting of state space, action space, transition probabilities and reward functions, however now it is necessary to keep a different action space and reward function for each agent, and the transition probability is conditional not only on a single state and action, but on a state and the combination of actions from all agents. The goal is still to find a policy, also known as a strategy, for each agent, such that some equilibrium is reached where each agent no longer want to change their strategy to gain better rewards. This is known as the Nash equilibrium [24, Chapters 3 and 4].

As with single agent RL, a variety of different methods have been developed to try and solve multiplayer stochastic games. However, researching and implementing many of such methods is out of the scope of this report. A single method has been implemented to act a reference later, to investigate how well it performs compared to the more usual methods implemented in this report. This is presented in Chapter 6.

4.1.2 Actions

The other part of the design of the agent is determining the action space. The action space can be defined rather arbitrarily and at different levels of abstraction. For the problem of beam alignment the intuitive choice of action space is the collection of codewords from the codebook. The agent's task is then, at each time step, to choose which codeword to use for beamforming until the next time step. In the multi agent case, each agent may choose only a single codeword and in the single agent case, the agent has to choose codewords for all antenna arrays at once. However, even the simple action of choosing codewords, presents some different design possibilities, because even if the total action space is just the codewords of the codebook, the subset of the action space made available to the agent at each time step need not be constant. Below, some methods of choosing the beamforming direction are outlined.

- **one-of-all**
  In this method the agent can choose freely between all codewords or combinations of codewords possible. This results in a large action space, that is kept constant for all states, and thus the optimal choice of codewords is always possible.
- **Adjacent**
  In this method the action space of the agent is limited by the currently used codeword. Instead of being able to choose any codeword in the codebook, only those codewords adjacent to the current codeword are available. By changing the action from choosing a specific codeword, to picking a direction the action space becomes; Stay, navigate left, right, down, up left and up right. The reason there are two and only two actions pertaining to up is due to how the codebook is constructed. As seen on Figure 4.1 a beam in a given layer is covered by exactly two narrower beams in the layer above. In the edge case of residing in beam 5 and navigating to the right the next beam will become beam 2, because

of wrap around, and so that navigating left/right only navigates within a single layer.

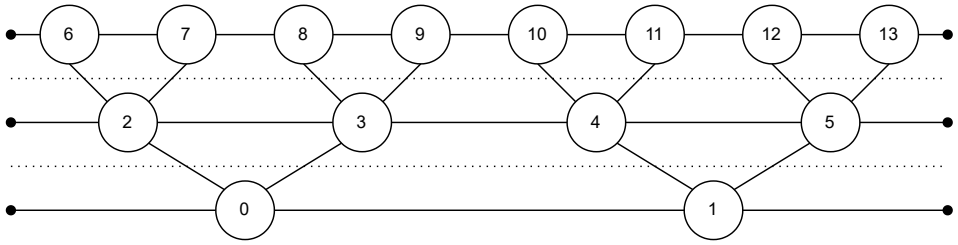


Figure 4.1: Illustration of a hierarchical codebook, with each beam numbered

The one-of-all action encoding should lead to a more versatile agent, as any beam is available at all times, however ensuring the performance of such an agent might be more difficult, especially if the codebook has many codewords. Adjacent encoding will always only have six actions, and as such a larger codebook will not necessarily make the decision process more complex. Even if it does, then not to the same degree as the one-of-all encoding. Of course the obvious limitation of this encoding is the fact that only six different beams are available at any one time.

4.2 The environment and the state

The environment is everything outside the agent, which in this case is primarily elements from the propagation environment. The propagation environment and how it is simulated is described comprehensively in Chapter 2. However, depending on where the agent/environment boundary is placed, even some information about the UE and the BS can be said to be part of the environment. The boundary may be placed very close to the abstract concept of the agent, such that the physical and mechanical conditions of the UE is outside of the agent. The agent can then be thought of as the "mind" making decisions, with the "body" being part of the environment.

The state then, as mentioned in Chapter 3, is a representation of the environment, presented to the agent by the environment at each time step. Using the state the agent chooses an action based on its policy. A way to represent the state is as a combination of state parameters, each describing a different aspect of the environment. The task when designing the state space is then determining and combining state parameters.

4.2.1 State parameters

Using multiple sets of different state parameters it is possible to control the detail of the knowledge of the environment. As the information about the environment becomes more detailed, the agent becomes better at discerning different situations from each other and can act accordingly. However, not all information about the environment might be relevant to the problem at hand or even available to the agent, as in practice the information usually comes from some finite amount of sensors in the environment. Additionally, by having a very detailed state the state space grows larger which might pose problems when using tabular methods. Thus, the main problem when designing the state is to determine which state parameters are available and have a significant impact on which action to choose, so as to keep the state space from becoming unnecessarily large.

Determining the impact a state parameter has on the decision process of the agent is often more difficult than determining which ones are available, as this relies more on specific field knowledge and intuition. Thus, designing the state is more of a creative process than a direct method and it involves trial and error to find out what works best. In the case of adaptive beamforming as presented in this report, with simulation data from QuaDRiGa which uses the geometry of the propagation environment, it is assumed that information about the placement and orientation of the UE and BS relative to each other will have an impact on the decision process of the agent. In addition to this, knowledge about the current codewords in use and histories of the mentioned information might add some relevant information as well, as looking a bit into the past might help the agent to make better decisions in the present.

All of the information mentioned above can easily be encoded separately and used in a state. In the following list each piece is described briefly together with a short argument for its relevance.

- **Codeword selection**
  This is simply the codeword most recently chosen, which might add valuable information depending on how the actions are encoded.
- **History of actions**
  A history of actions which can contain any number of previous actions, as previous actions could have implication about how to act in the present.
- **Orientation, and history hereof**
  The orientation of the UE, and a history hereof could be useful. If a drastic enough direction change occur, assuming some consistency in the direction of strong paths, it would indicate a change of codeword is needed.
- **Distance, and history hereof**
  The distance between the BS and the UE. A change in distance could indicate a change in behavior as the angle toward a BS changes at different rates at different distances.
- **Location, and history hereof**
  Location would directly indicate the best direction to beamform in, assuming a LOS scenario.

A state could be compromised of any combination of these, from having only the most recent beam selection to the state being compromised of all of them. The length of the histories is also variables which can be arbitrarily changed, which needs to be evaluated later. To find a good state representation, tests with different combinations will be conducted and their performance compared. More on these tests will be presented in Chapter 5.

4.3 The Reward

The reward is the metric dictating what the agent is trying to optimize and should be considered carefully. The goal of the agent should be to choose a beamforming direction such that in the long run a good connection is achieved. In the short term, the agent might take sub-optimal actions if it has experienced that these actions increases the possibility of better possible actions in the future. To discern between a good and a bad action, the logical solution is to let the reward be high when an action results a good connection and low otherwise. This of course leads to the question, what makes a connection "good". There are different metrics to describe a wireless connection, and some of them are given below.

- **Signal power**
- **SNR**
- **Error-rate**

A simple measure of the quality of the codewords in use, is to look at the signal power of the signal at the receiver and/or transmitter. If the signal power is high, one can usually assume that the connection is better, although this might not always be the case. One might imagine a situation where a system consists of multiple BSs, and the UE receives interfering signals from the stations it is not currently connected to. However, in mmWave communication where range is short and beamforming is used, this wouldn't be very likely as the UE would need to cross the beams of multiple BSs at once. Regardless, for this report this sort of noise is out of scope and only thermal noise and the noise factor of the equipment is considered.

A way to differentiate between noisy signals and clearer signals is to instead use the Signal-to-Noise Ratio (SNR). The SNR is slightly harder to estimate than the signal strength as the noise also has to be estimated. Additionally, the SNR doesn't say anything about total signal power it could be possible the agent optimizes with respect to low noise rather than high signal.

Lastly, a somewhat more complex metric is the error-rate, a measure of how many packets were received that contained errors. The error-rate is very different than the two other measures, as this also depends on the communication protocol and if it has error correction. Depending on how fast the RL algorithm runs this reward might not be immediately available. Additionally, this method also needs the simulations to include the transfer of data packets between the BS and the UE.
The error-rate is discarded as a measure of how high a reward the agent receives, as it is too complicated to implement given the scope of this report. The SNR is discarded as well based on the assumption that the noise added to each multipath component is from thermal noise and thus the noise level is mostly independent of codeword choice.

Given this assumption one would predict that directions showing good SNR would also be the directions that has high signal power. Thus, the reward is based on received signal power, with high power giving a high reward. To further keep the problem simple, the reward signal is chosen to be the received signal power directly, however other possibilities could be some linear or non-linear functions of the signal power, which will be investigated further if needed.

4.4 Design of RL algorithms

4.4.1 Encoding of state parameters

As the state parameters consists mainly of information gained by observing the environment with sensors, their values are inherently discrete, resulting in a finite state space. To represent the information efficiently in the state, some thought has to go into how to encode it. Information about the current codeword and a history of actions can be represented simply by numbering each codeword and then using this number when referring to a codeword. In the following the codeword selection parameter and its history consists of integers in the range $[0 - Nb - 1]$ with Nb being the amount of codewords in the codebook.

Information about the orientation of the UE is more complex, as the reference orientation used can be defined in many ways. The orientation may be relative to the initial orientation, the last

known orientation, to some fixed coordinate system etc. If relative to a fixed coordinate system the orientation would provide information about the absolute direction of the UE. If the agent has obtained good state-value estimated, the absolute orientation likely contains information about which direction is best to beamform in. In contrast a relative orientation would not provide information about the beamforming direction, but the change in beamforming direction which would be good. The absolute orientation have been chosen give the agent information about it's direction. The resolution denotes how many discrete values the orientation can take. The directions each discrete orientation corresponds to constitute equally large parts of the entire 360° spectrum, being numbered counter-clockwise. The first discrete orientation extend from 0° up to $\frac{360°}{R_o}$.

The information about distance can easily be encoded with some usual distance metric such as meters, with the resolution being a tunable parameter. The location can be encoded using GPS coordinates or by using some other coordinate system. An example of a coordinate system could be a grid centered around the BS and the location of the UE can then be represented in either rectangular or polar coordinates. In the following the location is represented by polar coordinates, as then the distance parameter can be transformed to a location by adding the corresponding angle to the UE. By controlling the resolution of both distance and angle, the size of the state space can be made smaller or bigger.

4.4.2 Value approximation

A very central part of the design of the RL agent is choosing how the value of states or state-action pairs is approximated. This often involves choosing between tabular methods and function approximation together with a choice of update rule.

Tabular vs. Approximate Methods

As mentioned in Chapter 3 two factors are very important when deciding between tabular and approximate methods, namely the size of the state-action space and the device intended to be used for calculations. To better determine which method to use, the size of the state-action space is investigated for different combinations of state-parameters and action encodings. The size of the state-action is dependant on the size of the action-space which is different for the single and multi agent case. The generated table will assume the action space of a single centralized agent, to obtain the most extremes state-action space sizes. Any conclusions about what's feasible will also apply to the multi agent case.

In order to derive useful numbers it is assumed in the following calculations that the receiver has eight antennas, and three layers, resulting in a total of 14 codewords. It is further assumed that the transmitter has 32 antennas and four layers resulting 30 codewords. The number of antennas are the minimum typical amounts given by Table 2.1. Additionally it is assumed that for each state-action pair a double precision float can be used for the value estimate. It might be possible to use less and keep the same performance, but as this is the default size used in our Python simulations, this value is chosen as a baseline. Not all combinations of the state parameters will be investigated due to time limitations, instead some combinations with low, medium and high complexity are chosen. The states chosen are shown in Table 4.1.

Table 4.1: Table of the state configurations used for size investigation.

	Codeword history (Receiver)	Codeword history (Transmitter)	Orientation history	Orientation resolution	Distance resolution	Angle resolution
Low 1	0	0	0	0	8	32
Low 2	0	0	3	16	0	0
Low 2	2	2	0	0	0	0
Medium 1	0	0	2	8	4	8
Medium 2	2	2	0	0	4	8
Medium 3	2	2	2	8	0	0
High 1	2	2	2	8	4	8
High 2	3	3	2	8	4	8
High 3	3	3	2	8	8	32

The codeword history of size N is defined as the currently used codeword and the $N-1$ previously used codewords. Calculating the total number of state-action pairs when using the one-of-all action encoding is simple, as any codeword can be chosen in all states, meaning that simple multiplication is sufficient. However, the adjacent action encoding does not allow for all actions in all states, as descending to a lower layer is not possible at the lowest layer and similarly for ascending at the topmost layer. For short histories of earlier actions, exact calculations are still easy to make, but as the history increases, the combination space becomes hard to calculate and it is easier to perform monte carlo simulations with an agent having an ϵ value of 1. By letting the agent run for sufficiently many steps it is possible to directly observe how many different state-action pairs are visited.

Table 4.2: Total state-action pairs and storage needed for value estimates for each state configuration.

	One-of-all encoding		Adjacent encoding	
State configuration	Total state-action pairs	Size in bytes	Total state-action pairs	Size in bytes
Low 1	$108 \cdot 10^3$	864 kB	$9.22 \cdot 10^3$	73.8 kB
Low 2	$1.72 \cdot 10^6$	13.8 MB	$147 \cdot 10^3$	1.18 MB
Low 3	$74.1 \cdot 10^6$	592 MB	$189 \cdot 10^3$	1.51 MB
Medium 1	$860 \cdot 10^3$	6.88 MB	$73.7 \cdot 10^3$	590 kB
Medium 2	$2.37 \cdot 10^9$	19.0 GB	$6.05 \cdot 10^6$	48.4 MB
Medium 3	$4.74 \cdot 10^9$	38.0 GB	$12.1 \cdot 10^6$	96.8 MB
High 1	$152 \cdot 10^9$	1.22 TB	$387 \cdot 10^6$	3.10 GB
High 2	$63.7 \cdot 10^{12}$	510 TB	$8.11 \cdot 10^9$	64.9 GB
High 3	$510 \cdot 10^{12}$	4.08 PB	$64.9 \cdot 10^9$	519 GB

From Table 4.2 it follows that when using the one-of-all encoding the state-action space quickly becomes too big for it to be implemented with tabular methods, as the needed storage size exceeds tens and hundreds of terabytes. In this case it probably would be better to use function approximation methods instead.

For the adjacent encoding, most of the configurations chosen here would probably be viable for a single agent scenario, where storage is less of a concern. However, it is not easy to determine how many of these state-action pairs are actually frequently visited and if their true value will change

faster than they are visited. Likewise, it is also hard to determine how well each configuration of state parameters will perform, based only on the size of the q-table. The easiest way to determine this, is to run a simulation for some time and observe how the agent acts. For the multi agent scenario the conclusion is largely the same, for the agent at the BS the conclusions for single agent applies. An agent at the UE likely have more strict memory requirements, however the smaller action space contributes to less stringent requirements. Again value approximation would be more appropriate for the one-of-all encoding, and adjacent encoding is viable for investigation with tabular methods.

4.4.3 Choosing an update rule

Both tabular and approximate methods require a choice of how to use experience to update value estimates. There exists a large variety of different methods for this, as it is still currently an area of research. To keep the design process simple, it has been chosen to only focus the analysis on whether to use on-policy or off-policy methods. These categories encompass many of the current popular methods and present some relatable high-level differences, which is convenient for comparisons.

As mentioned in Chapter 3, the on-policy methods learn a sub-optimal policy that accounts for the exploration that the agent performs due to a policy like ϵ-greedy. Thus, in situations where exploratory actions during learning might have expensive consequences, like a self-driving car choosing to run a red light, it is preferable to use on-policy methods. However, if the consequences of exploratory moves are not that important it might be better to use an off-policy method, as it learns the optimal policy directly.

As the actions in the framework presented so far consists of choosing a pair of codewords, the consequences for taking an exploratory step are not very expensive. At worst the pair chosen results in a loss of connection until the next time step, but it might also just give a slightly worse connection. As the time steps between actions are in the order of milliseconds, a connection loss for a few time steps is assumed to be acceptable.

Additionally, the consequences of choosing a bad beam pair do not differ substantially between states. Therefore it is difficult to argue that a conservative policy from an on-policy method results in much better online performance, than an off-policy method when assuming the same ϵ-soft exploration.

Thus. it is not possible to discard one method based only on how the problem is formulated. When using tabular methods, the choice of update rule is often not a complicated parameter to change in the implementation of the agent, and different methods can conveniently be interchanged and their performance compared directly. The same cannot necessarily be said for approximate methods, especially when using deep neural networks (DNN), as the update rules are somewhat more intertwined with the design of the neural network. However, even for DNNs both on-policy and off-policy methods exists, with Deep Q-learning being one of the most popular off-policy methods.

4.4.4 Hyper-parameters

Another part of implementing RL is finding the right values for the hyper-parameters of the algorithms. When using TD-methods the core hyper-parameters are the step-size parameter,α,

dictating how much the current value estimate is updated towards the target at each step, and the discounting factor, γ, which determines how strongly the agents takes future rewards into account when defining the target.

Another important hyper-parameter is the exploration factor, ϵ, if an ϵ-soft algorithm is used to ensure exploration by the agent. In stationary environments it is generally advised that the step-size and the exploration factor are decreased over time to ensure convergence to an optimal policy. A simple way of implementing this is to set both parameters to $\frac{1}{t}$ or some variation of this.

However, in non-stationary environments it is often best to at least keep ϵ from decaying indefinitely to ensure that the agent always takes some exploratory moves once in a while.

This gives rise to the introduction of methods that decreases and increases the value of ϵ dynamically over time, based on different metrics as seen in [25] or [26].

These methods however introduce hyper-parameters of their own, adding complexity to the algorithms.

Calculating optimal values for hyper-parameters is not a feasible task, as they are very problem specific, and thus usually a few different values are evaluated empirically and then compared. Based on initial intuition about the problem, it may be possible to determine viable ranges before testing.

To gain insight into how different choices of hyper-parameters impact the performance of the agent, parameter sweeps are performed for the step-size parameter, the discounting factor and exploration factor. Additionally some experiments are performed with decaying and varying exploration factors. These experiments and their results are described further in Chapter 5 and the accompanying test journals in appendices.

4.5 Summary

Based on the analysis and discussion in this chapter it has been decided that tabular methods may be viable solution, given an hierarchical codebook and an adjacent-only encoding on the actions. Additionally, a reward signal depending directly on the received signal power is deemed sufficient under the constraint of keeping the calculations simple. However, other decisions about the agent design, like state space, update rule, values of hyper-parameters and whether to use a centralized or distributed approach are not easily made. Regarding the state space, some possible state parameters have been uncovered by analysing the environment, however how to combine them to obtain the best performance can best be determined through tests. Likewise for the hyper-parameters, by using intuition about the problem it is possible to formulate likely ranges for some of the parameters, but their exact influence have to be found through experiments.

Running such tests, and making the final design decisions, is the subject of the next chapter, where the mentioned parameters are going to be tuned.

Tuning of Agent Parameters 5

As described in Chapter 4, the design of an RL algorithm involves many different choices, from defining the environment and which parts of it are observable and relevant for the agent, to deciding update rules and hyper-parameters. Some of the choices can be made based solely on the problem at hand, as some possibilities may be infeasible, but for others it is only possible to define some broad constraints or hypothesises.

In this chapter a sequence of experiments and their results will be presented, with the goal of tuning the state space and the hyper-parameters. First the state space is explored through sweeps over different combinations of state parameters, with fixed choices for hyper-parameters, with the goal of finding the best combination. Then, using this combination, the hyper-parameters are swept in turn. This procedure is performed for both a centralized and a distributed implementation, in order to compare the two methods. In an urban scenario, with a dense network of transmitters, the probability of the UE being close enough to at least one of the transmitters to have LOS is very high. Thus the state and hyper-parameters are tuned to such a scenario. How well the agent then performs in an NLOS scenario will we investigated in the next chapter among other things.

5.1 Test metric(s)

The performance metrics used in this report are all based on the misalignment between the received signal power and the highest achievable signal power at each time step. This metric is used instead of the received signal power alone, as this value is expected to vary significantly as the agent moves around the environment, regardless of the choices made. This is due to phenomena like free space loss and absorption, and thus a low received signal power might not necessarily correspond to bad performance of the agent.

Thus the misalignment is used instead, which is calculated as the difference between the received signal power and maximum achievable signal power in dB scale. The maximum signal power is defined as the maximum found when looking at all possible combinations of codewords in the entire codebook and not just those available to the agent at the current time step.

$$M = S_m - S_{max}$$

where

M :	$[dB]$	Misalignment	$RM \in \mathcal{R}$
S_m :	$[dB]$	Received signal power	$S_m \in \mathcal{R}$
S_{max} :	$[dB]$	Maximum achievable signal power	$S_{max} \in \mathcal{R}$

Both the received and maximum power used for calculating the misalignment are without added noise, to better allow the metric to measure the quality of the choices made. This is based on the notion that choices should be made based on the quality of the channel, which is assumed to be independent of the thermal noise in the equipment. It should be noted that this metric is only obtainable because the test is simulated, as normally the maximum achievable signal power is not directly available.

The metrics used to compare the different agent configurations are:

- Average misalignment
- Median misalignment
- 1^{st} quartile misalignment
- 3^{rd} quartile misalignment

5.2 Test setup

For all tests in this chapter, the update rule SARSA is used, based on an arbitrary choice between it and Q-learning, under the hypothesis that the results from tuning the different parameters should generalize across the two methods due to their similarity. The codebooks used are a 3 layer and a 4 layer hierarchical codebook for the UE and BS respectively, and the actions are encoded with the adjacent-actions-only encoding. The environment is based on the "3GPP_38.901_UMi_LOS" scenario with tracks based on pedestrian movement.

Table 5.1: Technical parameters used when tuning agent parameters.

Parameter	Value	Description
B	400 MHz	Total bandwidth [27]
f_c	28 GHz	mmWave carrier frequency
P_{TX}	40 dBm	Transmission power
r_c	200 m	Cell radius
N_r	8	Number of receive antennas
N_t	16	Number of transmit antennas
M	10000	Number of episodes pr. simulation
N	7000	Number of steps pr. episode

Further elaborations about how the simulations were designed and performed can be found in the accompanying test journals in Appendices A to D. However, a note should be added to the methodology used when tuning hyper-parameters in this chapter. When tuning the hyper-parameters, first a training period is undergone, followed by validation period. The training and validation periods both follow the general setup described in Appendices A to D, except during the validation period, the tracks are chosen from a validation data set. In the training period the hyper-parameters take on a fixed set of default values regardless of the test. Then in the validation period the hyper-parameters switch to the values to be tested. This extra step was intended to ensure the parameters are tested on similar agents, and to investigate if there is overfitting, which is tested by having the environment in the validation period be independent from the one in the training period.

All the results from hyper-parameter tuning in this chapter are from tests following this methodology. However, after the tests were performed, further reflection on the method found

that it does not make a lot of sense to do it this way, if the intention is to investigate overfitting. Further comments about the effect of this mistake and a discussion about what should have been done instead can be found in Chapter 7.

5.3 Single centralized agent

In this section a single centralized agent will be tuned, where the tuning parameters are the state space, and a collection of hyper-parameters. The state space will be tuned first as it's expected to have the largest impact on performance. It's also expected that a given set of hyper-parameters will have the same impact independent of the state parameters. This means the impact of their specific values can be ignored during the state space tuning, assuming sensible values.

5.3.1 State space

The state has been limited to a number of, thought to be useful, pieces of information as described in Section 4.2.1. This encoding was further solidified in Section 4.4.2, which means the state encoding consists of six values. They are given below with corresponding notation.

Nb_r : Codeword history for UE
Nb_t : Codeword history for BS
N_o : Orientation history
R_o : Orientation resolution
R_d : Distance resolution
R_a : Angle resolution

Testing all possible combinations of these state parameters is infeasible, so certain assumptions and simplifications will be made to narrow the search space. First the state parameters are grouped into three groups, whose values will be tuned in turn, as these groups are deemed to add independent information to the state space. The parameters are grouped as follows; codeword information: $[Nb_r, Nb_t]$, orientation information: $[N_o, R_o]$ and location information: $[R_d, R_a]$. The last two groups is further denoted commonly as context information.

First the codeword histories will be tuned where, to simplify further, the values for the histories of the UE and BS will be equal. Then the parameters for the context information will be tuned in turn, however these tests will also include the codeword history found to be best from the first test, as this information is deemed important enough to keep even when tuning the context information.

In order to run the tests, some values for the hyper-parameters needs to be chosen. The initial values have been chosen based on some basic heuristics, as they will be tuned properly later. The exploration factor is chosen to be 5% or 1 out of 20 steps is an exploration step, based on preliminary tests, where 0.05 lead to promising results. The step size is chosen to be 0.05 which, according to [22, p. 222], can be thought of as that the action-value estimate converges in approximately 20 steps, assuming the target is unchanged. For the discounting factor no good heuristic could be found, and a value of 0.7 was chosen at random. The hyper-parameters and the values used for tuning the state space are shown in Table 5.2.

Table 5.2: Initial hyper-parameter values for state space tuning.

Parameter	Value
Exploration rate, ϵ	0.05
Step size, α	0.05
Discounting factor, γ	0.7

All the tests performed can be found in Appendix A, with only a subset of the results presented and analyzed here.

Tuning of codeword information

For tuning the amount of codeword history to include in the state, three different choices were selected and their performance compared. From Section 4.4.2 it was found that having a history of 3 codewords in the state would put the state space at the limit of what would be viable to store in a table. Thus having any history beyond this would not be a viable solution. Similarly, the trivial state with no history at all would not be a viable solution as well. Thus the tests are with a history of one, two and three codewords respectively, and the results from the test is shown in Table 5.3 and Fig. 5.1.

Table 5.3: Table with misalignment measurements from tests with a single centralized agent with different length codeword history. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
History of 1 codeword	30.2 dB	30.1 dB	22.3 dB	37.8 dB
History of 2 codewords	24.9 dB	24.9 dB	16.0 dB	33.4 dB
History of 3 codewords	28.4 dB	28.3 dB	20.6 dB	36.1 dB

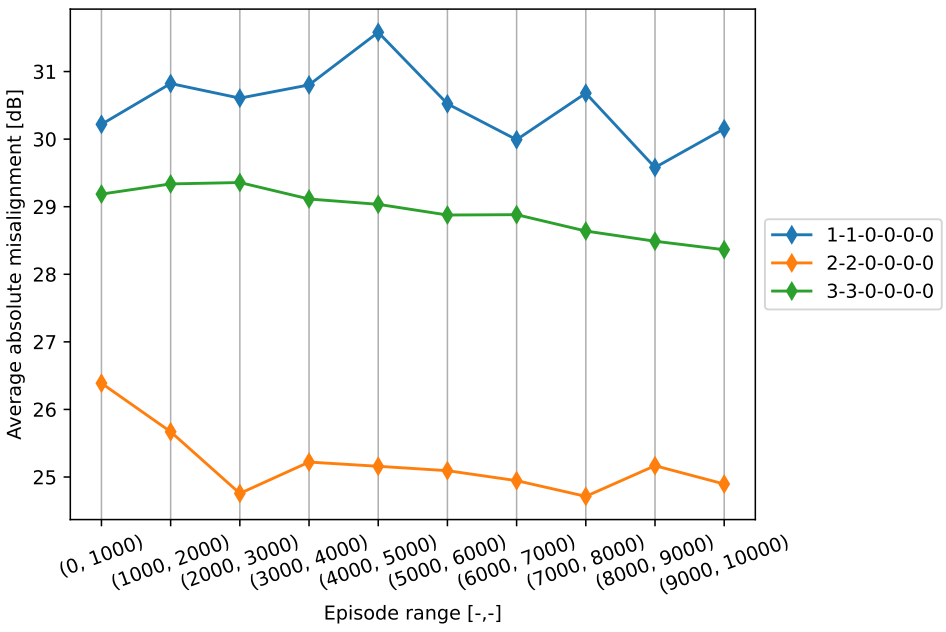


Figure 5.1: Average absolute misalignment from tests with a single centralized agent having only codeword histories for receiver and transmitter.Legend values indicate: Nb_r-Nb_t-N_o-R_o-R_d-R_a

It's clear the best performance is from the state with a history of 2 codewords. The state with only the current codewords most likely contains not enough information to be useful, as its performance seems to be the same for all episodes. The state with a history of 3 codewords seems to become better with more episodes, however with the rate visible from Fig. 5.1 it will take a long time before it reaches the performance of having only a history of 2 codewords. Thus a history of 2 codewords is chosen as the value for this state parameter.

Tuning of orientation information

Next the amount of orientation information was tuned. The orientation information is controlled by two parameters; the amount of orientation history and the resolution of the orientation. The sweeps for these parameters covered all combinations of four different values for each parameter. The sweeps were intended to be very broad to investigate the impact over a large range of values. The orientation history was given the range $[1, 2, 3, 4]$ and the resolution was given the range $[4, 8, 16, 32]$. The resolution range was made to correspond loosely with the codebook, where each layer is also a multiple of 2. A collection of the best results from the test is shown in Table 5.4 and Fig. 5.2, with the result from the state without orientation information added as a reference.

Table 5.4: Table with misalignment measurements of a single centralized agent with different orientation information. Results are from the last 1000 episodes in each test.

Test config.	Absolute Misalignment			
Nb_r-Nb_t-N_o-R_o-R_d-R_a	Average	Median	1st quartile	3rd quartile
Reference, no context information	24.9 dB	24.9 dB	16.0 dB	33.4 dB
2 - 2 - 1 - 8 - 0 - 0	24.5 dB	24.7 dB	14.9 dB	33.7 dB
2 - 2 - 2 - 8 - 0 - 0	25.1 dB	25.2 dB	15.2 dB	34.5 dB
2 - 2 - 3 - 16 - 0 - 0	25.1 dB	25.5 dB	15.3 dB	34.5 dB
2 - 2 - 4 - 16 - 0 - 0	24.7 dB	25.0 dB	15.1 dB	33.9 dB

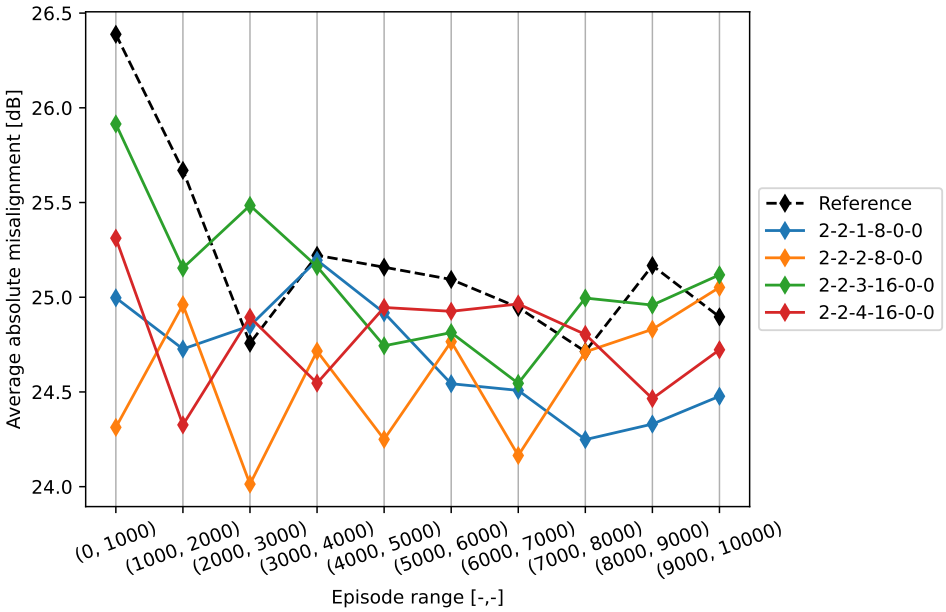


Figure 5.2: Average absolute misalignment from tests with a single centralized agent having different orientation information. Legend values indicate: Nb_r-Nb_t-N_o-R_o-R_d-R_a

In general it seems like adding orientation information to the state improves the performance, however not by very much. Even so, the best option seems to be having a resolution of 8 with a history of either 1 or 2. The former has worse performance to begin with, but becomes better towards the end. The latter has a much higher variability in performance. Because of this inconsistency and the better performance in the last steps, a history of 1 and resolution of 8 is assessed to be the best.

Tuning of location information

Finally the amount of location information was tuned. The location information is controlled by two parameters as well; the resolution of the distance and the resolution of the angle. As with the orientation information, the sweeps covered a broad range of combinations of different values for each parameter. The distance resolution was given the range $[2, 4, 8, 16, 32]$ and the angle resolution the range $[4, 8, 16, 32]$ again based on how the codebook is implemented. A collection of the best results from the test is shown in Table 5.5 and Fig. 5.3 , with the result from the state without location information added as a reference.

Table 5.5: Table with misalignment measurements of a single centralized agent with different location information. Results are from the last 1000 episodes in each test.

Test config.	Absolute Misalignment			
Nb_r-Nb_t-N_o-R_o-R_d-R_a	Average	Median	1st quartile	3rd quartile
Reference, no context information	24.9 dB	24.9 dB	16.0 dB	33.4 dB
2 - 2 - 0 - 0 - 2 - 32	14.2 dB	12.4 dB	4.4 dB	21.3 dB
2 - 2 - 0 - 0 - 4 - 32	14.7 dB	13.1 dB	5.1 dB	21.9 dB
2 - 2 - 0 - 0 - 8 - 32	13.9 dB	11.7 dB	4.2 dB	21.1 dB
2 - 2 - 0 - 0 - 16 - 32	13.1 dB	10.4 dB	3.6 dB	20.5 dB
2 - 2 - 0 - 0 - 32 - 32	11.9 dB	8.5 dB	3.0 dB	18.5 dB

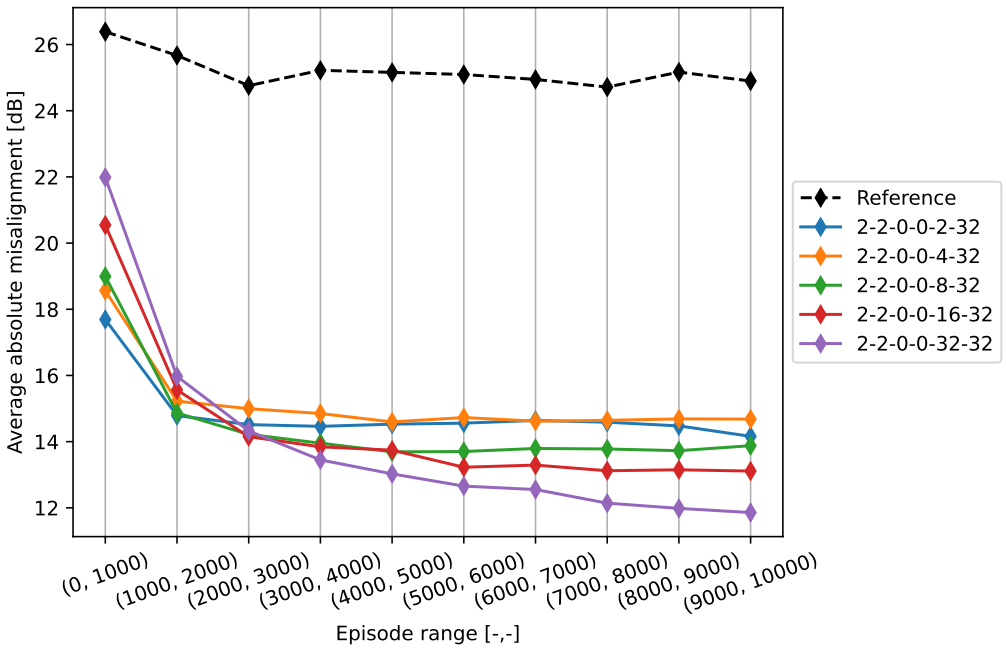


Figure 5.3: Average absolute misalignment from tests with a single centralized agent having different location information. Legend values indicate: Nb_r-Nb_t-N_o-R_o-R_d-R_a

The performance improves drastically by adding the relative location, with an improvement of around 10 to 14 dB. The improvement in performance is not unexpected as the location would directly indicate in which direction the BS should beamform given the line-of-sight scenario. The state with the highest distance and angle resolution is the best here, although the difference from changing the distance resolution is very little.

Combining the state parameters

Having tuned the state parameters separately, a final comparison is made with combinations of the best results from earlier. From the best results from tuning of location information, both the state with the lowest and the highest distance resolution has been chosen for this comparison, as their difference in performance is small, but their state space size very different. Thus hopefully the smaller state space will yield good results when combined with the orientation information.

During the tests, the simulation with the combination of the best results from all previous tests repeatedly crashed because the machine ran out of memory and thus it is not in the results. It is assumed that the table containing the action-values became too big to keep in memory. Thus only a single combined state is presented here. The results of combining the state parameters, and the results from the earlier tests are shown in Table 5.6 and Fig. 5.4, with the result from the state without context information added as a reference.

Table 5.6: Table with misalignment measurements of a single centralized agent with different combinations of state parameters. Results are from the last 1000 episodes in each test.

Test config.	Absolute Misalignment			
Nb_r-Nb_t-N_o-R_o-R_d-R_a	Average	Median	1st quartile	3rd quartile
Reference, no context information	24.9 dB	24.9 dB	16.0 dB	33.4 dB
2 - 2 - 0 - 0 - 2 - 32	14.2 dB	12.4 dB	4.4 dB	21.3 dB
2 - 2 - 0 - 0 - 32 - 32	11.9 dB	8.5 dB	3.0 dB	18.5 dB
2 - 2 - 1 - 8 - 0 - 0	24.5 dB	24.7 dB	14.9 dB	33.7 dB
2 - 2 - 1 - 8 - 2 - 32	11.9 dB	8.6 dB	2.9 dB	18.7 dB

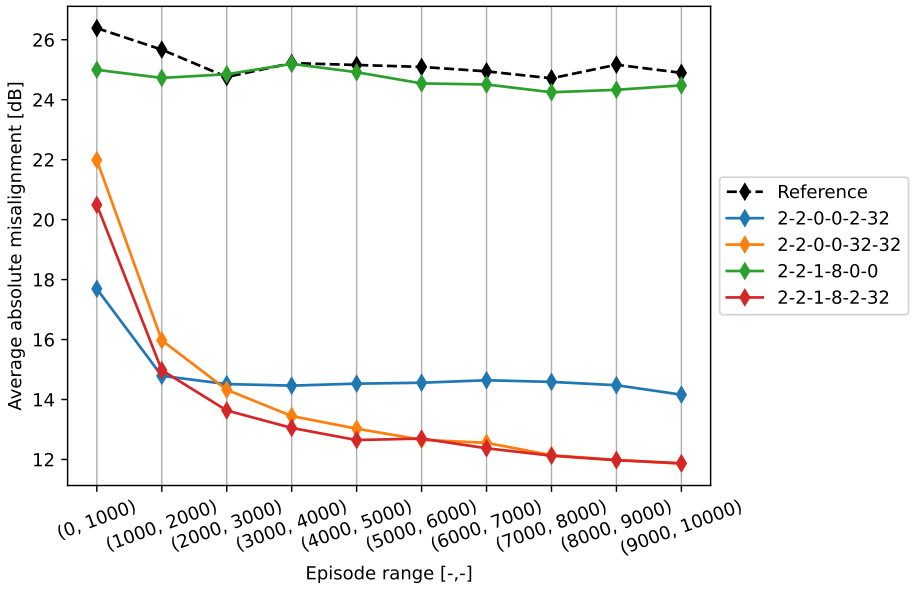


Figure 5.4: Average absolute misalignment from tests with a single centralized agent having different combined states. Legend values indicate: Nb_r-Nb_t-N_o-R_o-R_d-R_a

While the orientation only provided marginal performance improvement by itself, as seen by the green curve, it results in a larger improvement when combined with the relative location. This is expected as the optimal beamforming direction cannot be derived from the orientation itself, but can be with orientation and location. When comparing the combined state, with $\mathcal{S}$ = 2-2-0-0-32-32 which has no orientation, they have almost identical performance. This does not mean they are equal however, when considering the size of the state space it can be seen the size of context information part of the former is half that of the latter as, $32 \cdot 32 = 1024$ and $1 \cdot 8 \cdot 2 \cdot 32 = 512$. The codeword history affects the state space size equally in both and thus $\mathcal{S}$ = 2-2-1-8-2-32 is smaller.

The best performing state of the ones tested is $\mathcal{S}$ = 2-2-1-8-2-32 as its average misalignment is among the best while being of reasonable size.

5.3.2 Hyper-parameters

The hyper-parameters, outlined in Section 4.4.4, are the exploration rate ϵ, step size α and discounting factor γ. As with the state parameters, testing all combinations is infeasible and thus certain assumptions and simplifications will be made to narrow the search space. The hyper parameters will only be tested for a single state, $\mathcal{S}$ = 2-2-1-8-2-32, which was found to be the best.

First the discounting factor is tuned, followed by the step size and finally the exploration rate. Initially all hyper-parameters start at the values found in Table 5.2. When a value has been tuned, this value will carry on to the tests after it, instead of the default value. For the discounting factor and the step size, only constant values will be tested, but for the exploration rate a few different methods will be tested. In addition to constant values, some methods where the exploration rate changes over time are tried as well. One method is to let ϵ decay over time, which is a simple and popular method often used in practice for RL. Another method is to adapt ϵ continuously, using the algorithm described in [25].

All the tests performed can be found in Appendix C, where a subset of the results will be presented and analyzed here. In the following presentation, the reference results shown in black are from the test with $\mathcal{S}$ = 2-2-1-8-2-32 and the initial hyper-parameters.

Tuning the discounting factor

For tuning the discounting factor the range of values was chosen to be broad, as there was no good prior knowledge to narrow the choice. The discounting factor was given the range $[0.25, 0.50, 0.70, 0.95]$, with 0.7 being the reference from the state tuning tests. The results from the test are shown in Table 5.7 and Fig. 5.5.

Table 5.7: Table with misalignment measurements of a single centralized agent with different discounting factors. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
$\gamma = 0.50$	11.8 dB	7.2 dB	1.9 dB	19.3 dB
$\gamma = 0.70$	10.4 dB	6.5 dB	1.4 dB	16.5 dB
$\gamma = 0.25$	15.6 dB	12.5 dB	2.8 dB	26.4 dB
$\gamma = 0.95$	10.8 dB	7.8 dB	2.5 dB	17.0 dB

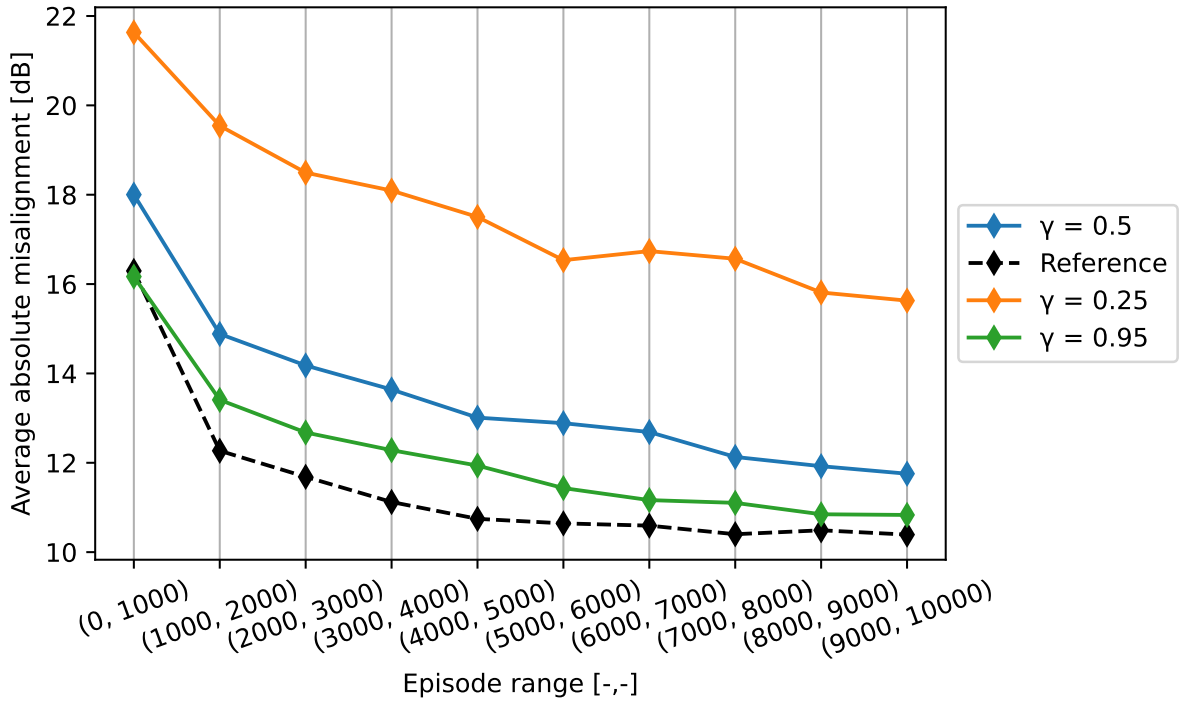


Figure 5.5: Average absolute misalignment from tests with a single centralized agent having different discounting factors.Legend values indicate: Nb_r-Nb_t-N_o-R_o-R_d-R_a & ϵ α γ

The test shows that the value of the discounting factor has a large impact on the performance. The discounting factor should be large rather than small, however not too large. The initial value of 0.7 gave the best performance, and most likely even better performance can be seen by fine-tuning further around the value. However, such fine-tuning is out of the scope for the current tuning process and thus the following tests are performed with $\gamma = 0.70$

Tuning the step size

Next the step size parameter was tuned. Based on the convergence interpretation from [22, p. 222], the step size was varied such that convergence would be expected in 3 steps to 200 steps, with the following exact values used: $[\frac{1}{3}, \frac{1}{6.66}, \frac{1}{20}, \frac{1}{100}, \frac{1}{200},]$. A collection of the results from the test is shown in Table 5.8 and Fig. 5.6.

Table 5.8: Table with misalignment measurements of a single centralized agent with different step sizes. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
$\alpha = 0.01$	9.9 dB	5.8 dB	1.7 dB	15.5 dB
$\alpha = 0.005$	10.9 dB	7.3 dB	2.7 dB	17.2 dB
Reference	10.4 dB	6.5 dB	1.4 dB	16.5 dB
$\alpha = 0.15$	10.6 dB	6.3 dB	0.9 dB	17.4 dB
$\alpha = 0.33$	11.5 dB	7.8 dB	1.9 dB	18.4 dB

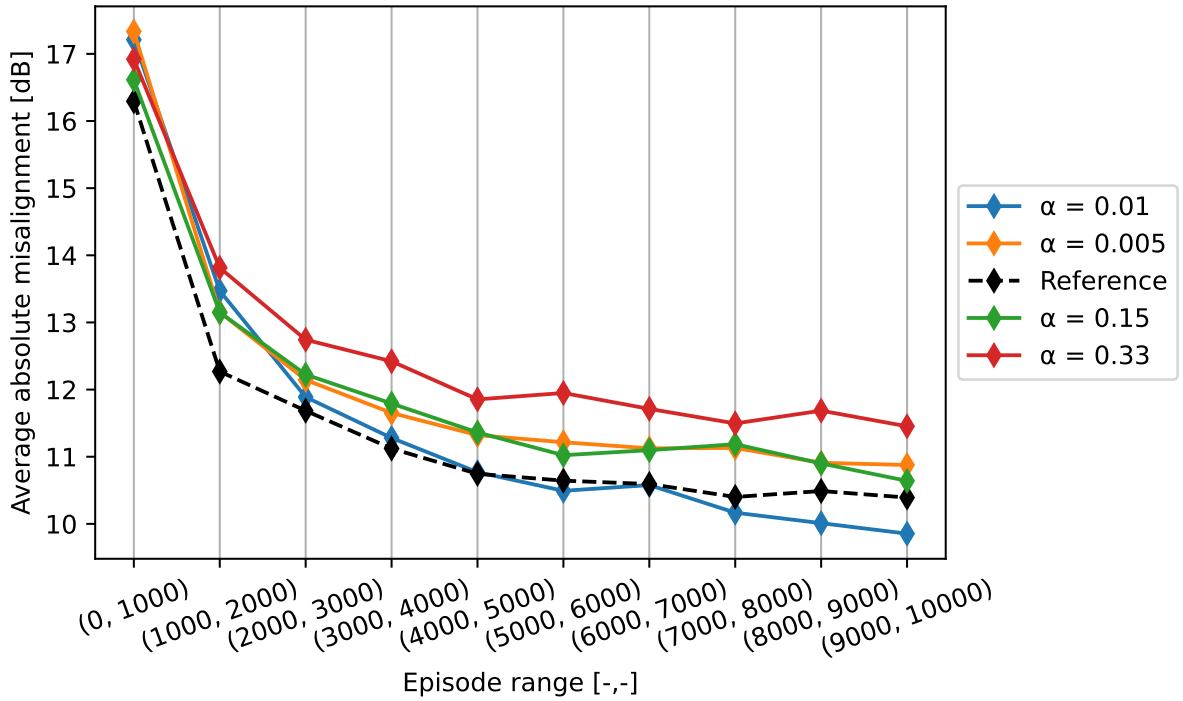


Figure 5.6: Average absolute misalignment from tests with a single centralized agent having different step sizes.Legend values indicate: Nb_r-Nb_t-N_o-R_o-R_d-R_a & ϵ α γ

This test indicates that the step size does not have a great impact on performance. However, smaller values does seem to generally perform better, with the exception of the smallest of 0.005. This might not be surprising, as with smaller values one would expect convergence to be slower and thus the smallest value might not have converged yet in the 10000 episodes used in the test. Regardless, the performance of the different values does not differ significantly, which is fairly surprising as given a convergence interpretation of the parameter and the broad range of values represented. Possibly this is because the environment is not stationary, making the action-values moving targets which makes convergence impossible.

From this test, a step size of 0.01 is chosen for the next tests.

Tuning the exploration rate

Finally the exploration rate was tuned. As mentioned earlier, the exploration rate has been implemented with three different methods, to investigate if an exploration rate that changes over time is an advantage compared to keeping it constant.

Table 5.9: Values used for tuning each exploration rate method.

Method under test	Value 1	Value 2	Value 3	Value 4
Constant	0.005	0.01	0.05	0.1
Decaying	300	600	900	1200
Adaptive	$0.1 \cdot 10^{-3}$	$3.0 \cdot 10^{-3}$	$15.0 \cdot 10^{-3}$	$20.0 \cdot 10^{-3}$

For the constant values, the heuristic for choosing values was a simple one; how often should the agent make an exploratory move. From early experiments it was determined that a small

value, $\epsilon < 0.1$, would probably be best. The values for the tests with constant exploration rate can be found in Table 5.9.

For the decaying exploration rate, the ϵ-value is initiated to 1 at the beginning of each episode and then decays exponentially. To control the rate of the decay, a weight factor, w_d, is added in the exponent as shown in (5.1).

$$\epsilon(t) = e^{\left(-\frac{t}{w_d}\right)} \tag{5.1}$$

The values of the weight factor can be found in Table 5.9.

For the adaptive exploration rate, a separate ϵ-value is kept for each state. The value is initiated with $\epsilon(s)_0 = 1$ and then updated after each visit to the state using:

$$\epsilon(s)_{t+1} = \delta \cdot \epsilon(s)_t + (\delta - 1) \cdot \epsilon_{adj} \tag{5.2}$$

where $0 < \delta < 1$ is a constant and ϵ_{adj} is a value calculated based on the TD-error as given below:

$$\epsilon_{adj} = \frac{1 - exp\left(\frac{-|\alpha \cdot \text{TD-error}|}{\sigma}\right)}{1 + exp\left(\frac{-|\alpha \cdot \text{TD-error}|}{\sigma}\right)} \tag{5.3}$$

where α is the step size and σ is a positive constant called the inverse sensitivity, adjusting how sensitive ϵ_{adj} is to changes in the TD-error. Some consideration went into choosing the range of the σ parameter, however after a few tests, it was found that by using the initial reasoning the performance was not very good. The initial considerations can be found in the included work sheets in Appendix H and in short the values from this reasoning proved to be too small. A range of tests with larger values were conducted afterwards, and it is the results from these tests that are shown in Appendix C. The values for σ can be found in Table 5.9.

A collection of the best result for each method used is shown in Table 5.10 and Fig. 5.7.

Table 5.10: Table with misalignment measurements of a single centralized agent with different epsilon methods. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Adaptive	8.6 dB	5.1 dB	2.2 dB	12.3 dB
Constant	9.3 dB	5.9 dB	2.3 dB	13.7 dB
Reference	10.4 dB	6.5 dB	1.4 dB	16.5 dB
Decaying	10.9 dB	7.0 dB	3.2 dB	16.3 dB

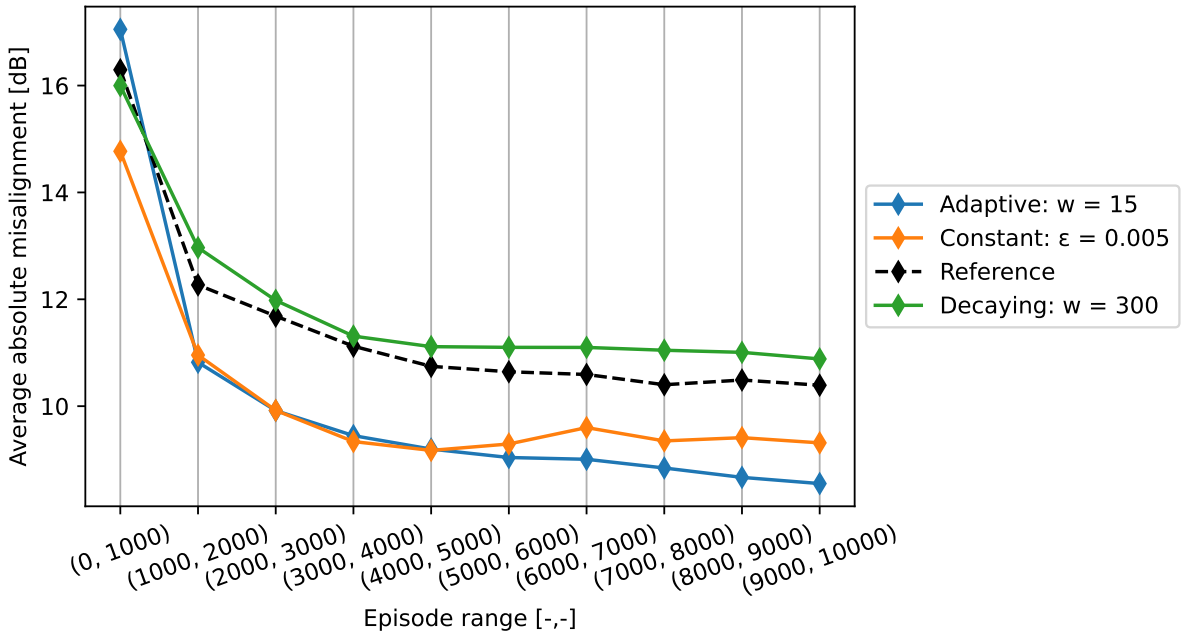


Figure 5.7: Average absolute misalignment from the best tests with a single centralized agent having different epsilon methods.Legend values indicate: Nb_r-Nb_t-N_o-R_o-R_d-R_a & ϵ α γ $\sigma \cdot 10^3$

It is evident that the decaying ϵ is unfit for the problem, as the performance gets worse than the reference. This is expected as the environment is dynamic, and when ϵ reaches a low enough value no more exploration steps will be taken. This will likely lead to outdated value estimates. The adaptive ϵ method is a bit slower to learn, most likely because of the initial value of 1. The constant and adaptive ϵ methods then perform similarly until the 5000-6000 episode range, where the constant method suddenly stops improving, while the adaptive method keeps improving. While it's hard to conclude if this is because of the inherent differences in the methods or simply a result of the random nature of the tests, it can be concluded that an adaptive method is either better than or equal to the constant method.

As the adaptive method should be more robust against a dynamic environment, this method is preferred over other two methods. A collection of the tuned hyper-parameter values is shown in Table 5.11.

Table 5.11: Tuned hyper-parameter values for single agent implementation.

Parameter	Value
Inverse sensitivity, σ	$15.0 \cdot 10^{-3}$
Step size, α	0.01
Discounting factor, γ	0.7

5.4 Distributed agents

In addition to the centralized single agent implementation, a distributed multi agent implementation is developed and tuned as well. For the multi agent implementation a very simple setup is made, where two agents almost identical to the one in the centralized implementation are placed in the same environment, such that each agent also becomes part of the others

environment. The only difference between them is their action spaces, with each agent choosing codewords from only a single codebook, belonging to the transmitter and the receiver respectively. The same state- and hyper-parameters will be used as with the single agent case, with the same encodings. As with single agent case, the state space will be tuned first followed by the hyper-parameters. As the proposed multi agent implementation is very similar to the single agent implementation, a complete walk-through of the tuning process is divided and put into Appendices B and D, and only brief comments about the results are presented here.

5.4.1 State space

The state notation is consistent across both single and multi agent, the difference being that in multi agent each agent posses their own state. The multi agent case is different than the single agent in the sense that communication is implicitly assumed in the single agent case. A set of states for the cases with and without communication was therefore be identified and tested.

Tuning of codeword information

At first a set of states consisting of only the codeword history for each agents own actions were tested, followed by a test where the codeword history of the other terminal was introduced to each agent, as two cases for differing amount of information about the two agents. These tests showed that when the task of choosing codewords is separated into two agents, it is best for each agent to only have a codeword history of its own actions. In fact performance gets much worse when adding the other agents actions to the state, as seen in Table 5.12 and Fig. 5.8.

Table 5.12: Table with misalignment measurements from tests with two agents with and without communication of codeword information between agents. Results are from the last 1000 episodes in each test.

Test config.	Absolute Misalignment			
	Average	Median	1st quartile	3rd quartile
Without communication	21.3 dB	20.8 dB	10.7 dB	30.4 dB
With communication	26.2 dB	26.3 dB	18.1 dB	34.4 dB

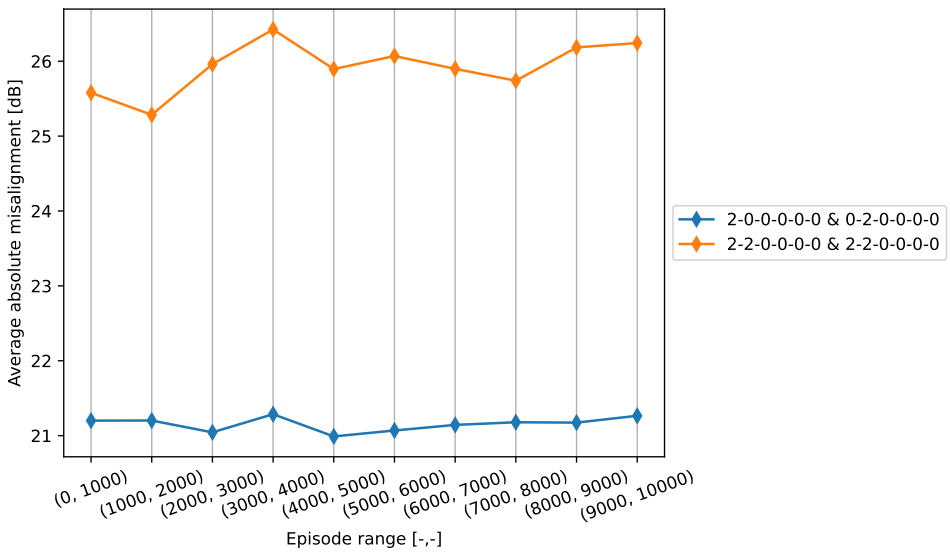


Figure 5.8: Average absolute misalignment from tests with two agents with and without communication of codeword information between agents. Legend values indicate: Nb_r-Nb_t-N_r-R_o-R_d-R_a

This might be a sign that the information added by including the actions of both agents in each state, does not outweigh the increase in state space which makes it harder for the agents to keep accurate action-value estimates.

Tuning of context information w. partial and full communication

While adding communication about codewords did not improve performance, the same might not be the case with context information. Thus, it was investigated what happens when communication is gradually added regarding context information. First only partial communication is added, by giving both agents access to the relative position of the UE and keeping the orientation information about the UE only available to the UE.

From this case it was determined that the relative location by itself provides a considerable improvement in performance. Furthermore, adding the orientation information improved performance even more, even to a higher degree than in the single agent case. This can be caused by the fact that this information is largely useless for the BS, as the best direction to beamform in a LOS scenario is always the direction pointing at UE, regardless of the orientation of the UE. However, for the UE the best direction to beamform depends very much on its current orientation. Thus it makes sense that when the two actions are decentralized the addition of orientation information only where it is useful has a large effect.

Following these tests, the case with full communication was investigated, where both agents has access to all information, as in the centralized case. Here the results showed that adding the orientation information to the state at the BS, produces slightly worse results than the state without, when all other state parameters are equal. This supports the logic presented earlier, that the BS has little to gain from knowing the orientation of the UE.

Thus in summary, for the multi agent case the best performance is found when there is partial communication between the agents, with location information being available to both agents. The misalignment plot for this state is illustrated in Fig. 5.9.

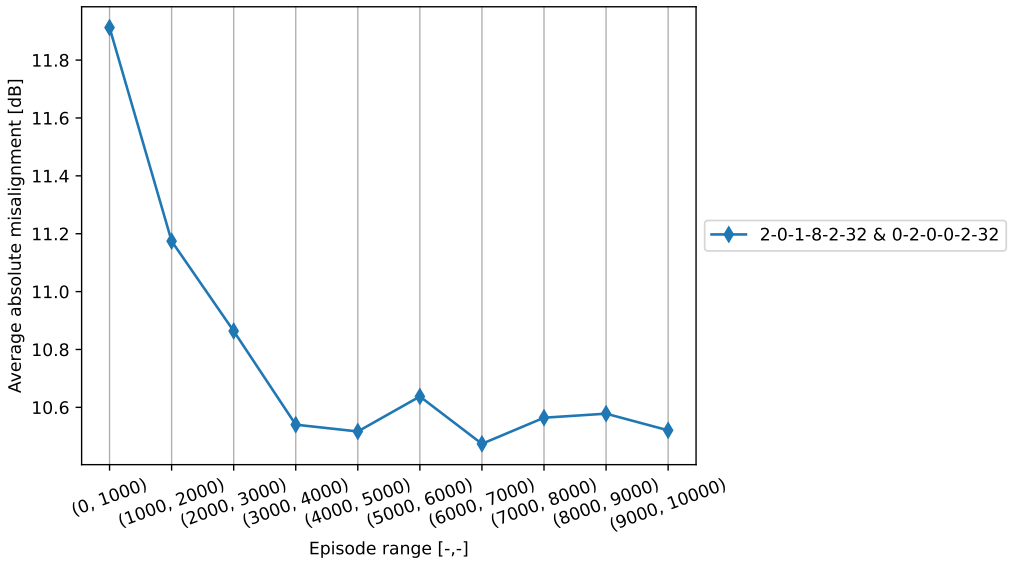


Figure 5.9: Average absolute misalignment from the best performing combination of state parameters in the multi agent case.Legend values indicate: Nb_r-Nb_t-N_r-R_o-R_d-R_a for UE & BS

5.4.2 Hyper-parameters

As before, a subset of hyper parameters will be tested independently in the same order, first discounting factor, then step size and finally exploration factor. The test methodology remains the same, and the state pair used is $\mathcal{S}_r =' 2-0-1-8-2-32'$ & $\mathcal{S}_t =' 0-2-0-0-2-32'$, which was found to be best in the previous section.

To keep the amount of tests manageable, only tests where the hyper-parameters are the same for both agents are performed. This might not be the best method, as the two agents might benefit from having different hyper-parameter values, however it is assumed that in a broad sense the best values will be similar. During training, the hyper-parameters are the same as the initial values found in Table 5.2. All the tests performed can be found in Appendix D, and because of the similarity with the single agent case only some brief comments will be presented here, together with a summary of which values were found to be the best.

Assuming that the multi agent case is very similar to the single agent case, the ranges chosen for the hyper-parameters are similar as well, only discarding some of the extreme values that clearly gave bad performance. For the discounting factor, the best value ended up being smaller than for the single agent. However, by fine-tuning the value for the single agent, a smaller value might prove better there as well. Additionally the results show that varying the discounting factor also has a large impact on performance in the multi agent case. Smaller values being better in the multi agent case might be because in the multi agent case, the action of each agent is uncoordinated with the other agent. Thus it might be better for an agent to focus more on its own immediate reward instead of looking to far ahead, as its action-value approximations only account for the expected return when it follows its own policy.

Table 5.13: Tuned hyper-parameter values for multi agent implementation.

Parameter	Value
Inverse sensitivity, σ	$10.0 \cdot 10^{-3}$
Step size, α	0.01
Discounting factor, γ	0.6

For the step size, the overall trend in the results were very similar to the single agent case, with not much difference in performance for the different values. The same value ended up being best in both cases, but this might just be random as the margin to the second best is very small. Finally, for the exploration rate the trend in the results were again almost the same as for the single agent case. The decaying ϵ method is clearly worst and a constant value is very similar to an adaptive one. Based on the same logic as before, the adaptive method is preferred over the constant value as it is probably more robust to dynamic environments.

5.5 Summary

Through this chapter two different RL implementations have been tuned, with the intention of getting a set of configurations and additionally an intuition about the influence of difference parameters in an RL algorithm together with a deeper knowledge of the problem at large.

Based on the results from all the conducted tests in this chapter, it is possible to make some initial conclusions. For the state parameters it has been proven that in a LOS scenario, context information about the position of the UE relative to the BS is very informative for the RL agent.

Something similar is also true of the orientation, however this information is likely only valuable for determining the best codeword selection at the UE.

For the hyper-parameters results show that especially the discounting factor has a significant impact on the performance, with the best value being somewhere above 0.5 and below 0.8. This indicates, that in the specific scenario used in this chapter, an agent benefits from being farsighted only to some degree. The step size proved to be not as sensitive as the discounting factor, with the best performance being attained with a step size around 0.05, and very large values resulting in worse performance. Finally, it was found that an adaptive exploration rate performs just as well or better than a constant rate.

Table 5.14: Tuned hyper-parameter values for both the single and the multi agent implementations.

Parameter	Single agent	Multi agent
Inverse sensitivity, σ	$15.0 \cdot 10^{-3}$	$10.0 \cdot 10^{-3}$
Step size, α	0.01	0.01
Discounting factor, γ	0.7	0.6

The final settings for both implementations are collected in Table 5.14, and a comparison of their performance is illustrated in Table 5.15 and Figures 5.10 and 5.11.

The two implementations have similar average misalignment in the end, however the multi agent implementation has almost no learning curve compared to the single agent implementation. This can likely be contributed to the much larger state space of the single agent implementation.

In addition to the average misalignment plots, two ECDF plots are included here, to give a more nuanced picture of the performance of the two implementations.

Table 5.15: Table of different misalignment measurements from the tuned agents for both implementations. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Multi agent implementation	7.8 dB	4.3 dB	0.4 dB	11.8 dB
Single agent implementation	8.6 dB	5.1 dB	2.2 dB	12.3 dB

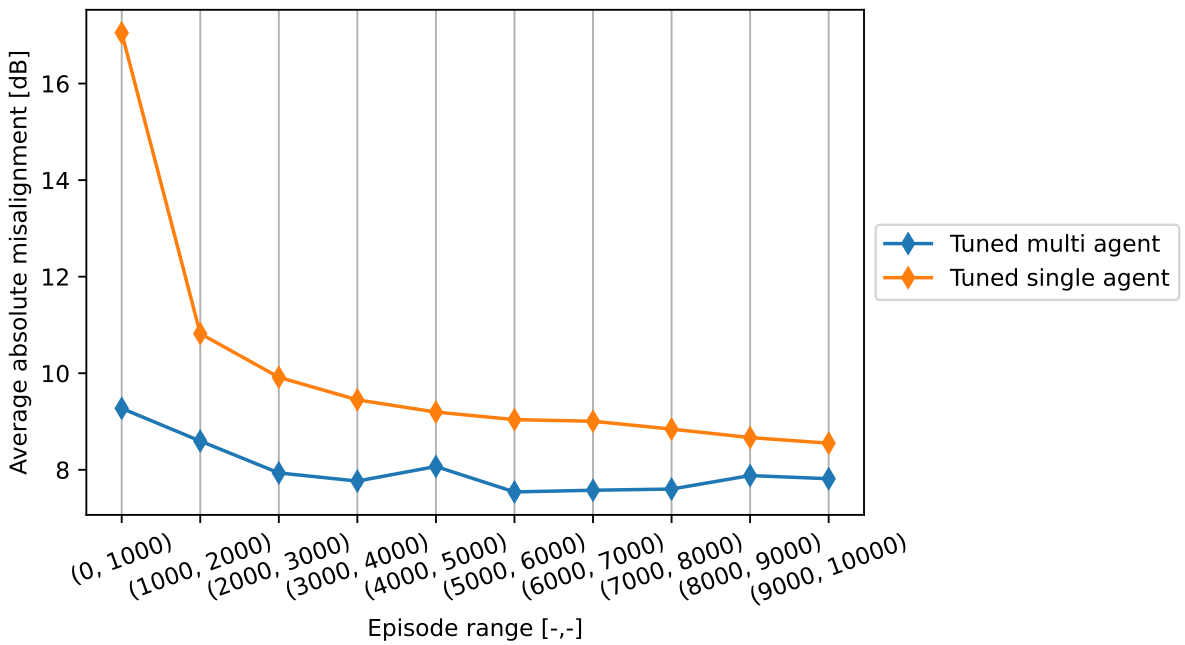
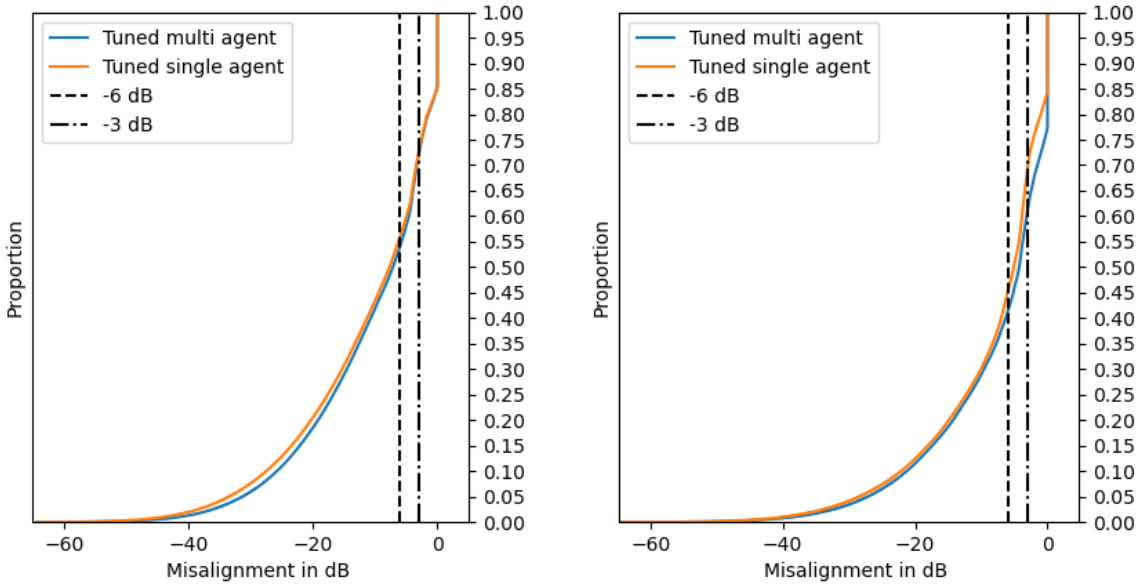


Figure 5.10: Average absolute misalignment from the tuned agents for both implementations.

As mentioned during the presentation of the test setup, the methodology used when tuning the hyper-parameters is somewhat flawed, however the results are kept and are used in the next chapter, as time did not allow for re-running all the simulations. In the next chapter, it will be investigated how well the tuned parameters found in this chapter performs when the update rule or the environment is changed significantly.



(a) After training, before validation

(b) After validation

Figure 5.11: ECDF plots for the misalignment of the tuned agents. The plots show the ECDF after the initial 10000 episodes of training, (a) and again after further 10000 episodes during validation. Results are from the last 1000 episodes in each test

Evaluation of tuned agent 6

In Chapter 5 a set of tuned settings were found for both a single centralized agent and a multi agent RL implementation. However, as the tuning was based on a single choice of update rule and fixed environment settings, there is an increased risk that the values are only valid in this case. To investigate how well the tuned settings generalize, this chapter is devoted to tests where the agent has other update rules and tests in different environments. The results from these tests are of course compared with the previous results, but in addition to this an entirely new algorithm is introduced to act as a reference for non-ML performance. Additionally, the actions taken by the agent are investigated in an attempt to describe qualitatively what behaviour the agent learns in different scenarios.

6.1 Test setup

For all tests in this chapter the codebooks used are a 3 layer and a 4 layer hierarchical codebook for the UE and BS respectively, and the actions are encoded with the adjacent-actions-only encoding.

Table 6.1: Technical parameters used when tuning agent parameters.

Parameter	Value	Description
B	400 MHz	Total bandwidth [27]
f_c	28 GHz	mmWave carrier frequency
P_{TX}	40 dBm	Transmission power
r_c	200 m	Cell radius
N_r	8	Number of receive antennas
N_t	16	Number of transmit antennas
M	10000	Number of episodes pr. simulation
N	7000	Number of steps pr. episode

Further elaborations about how the simulations were designed and performed can be found in the accompanying test journals in Appendices E to G.

6.2 Non-ML reference algorithm

To get an idea of how well the developed RL algorithms perform, it is interesting to compare them with a heuristic algorithm, where the behavior is determined by a human hand. A simple algorithm without learning, is one where actions instead are based on the last received signal power and a threshold value. At each time step the received signal power is compared with a fixed threshold value, and if the received power is lower than the threshold, a new pair of

codewords are chosen at random, otherwise the current codeword pair is used again. To make the comparison more even, the adjacent-actions-only encoding for actions is used for the heuristic algorithm as well. The algorithm is illustrated in Algorithm 3.

Algorithm 3: Heuristic beam alignment algorithm

Input: $Threshold > 0$, action space $\mathcal{A}$
for *each episode* **do**
  Choose A from $\mathcal{A}$ at random;
  for *each step in episode* **do**
    Take action A, observe R
    if $R < Threshold$ **then**
      Choose A' from $\mathcal{A}$ at random;
    else
      $A' \leftarrow A$;
    end
    $A \leftarrow A'$
  end
  Until episode is done
end

A quick tuning of the threshold value found that a value of 0.0002 gives the best average misalignment in a pedestrian LOS scenario.

6.3 Update rule

The update rules that will be tested are Sarsa, Q-learning, which were discussed in Section 3.2.1, and a simple update rule that has the immediate reward as target. The simple update rule approximates the action-value as the average immediate reward received when taking that action in that state, and is implemented by modifying Sarsa with a discounting faction of $\gamma = 0$. The three updates will be compared with the simple heuristic algorithm to gauge whether this RL solution is worthwhile. The update rules will be judged based on their performance in terms of misalignment measurements as in Chapter 5. They will further be analyzed based on the actions the agent take, with a certain update rule, and how often each specific beam was visited.

6.3.1 Single centralized agent

Table 6.2: Table of different misalignment measurements of single centralized agents with different update rules. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Heuristic	9.1 dB	6.5 dB	3.7 dB	10.8 dB
Q-learning	10.1 dB	7.1 dB	3.3 dB	14.7 dB
Sarsa	11.3 dB	7.7 dB	2.6 dB	17.6 dB
Simple	12.6 dB	6.9 dB	1.8 dB	21.3 dB

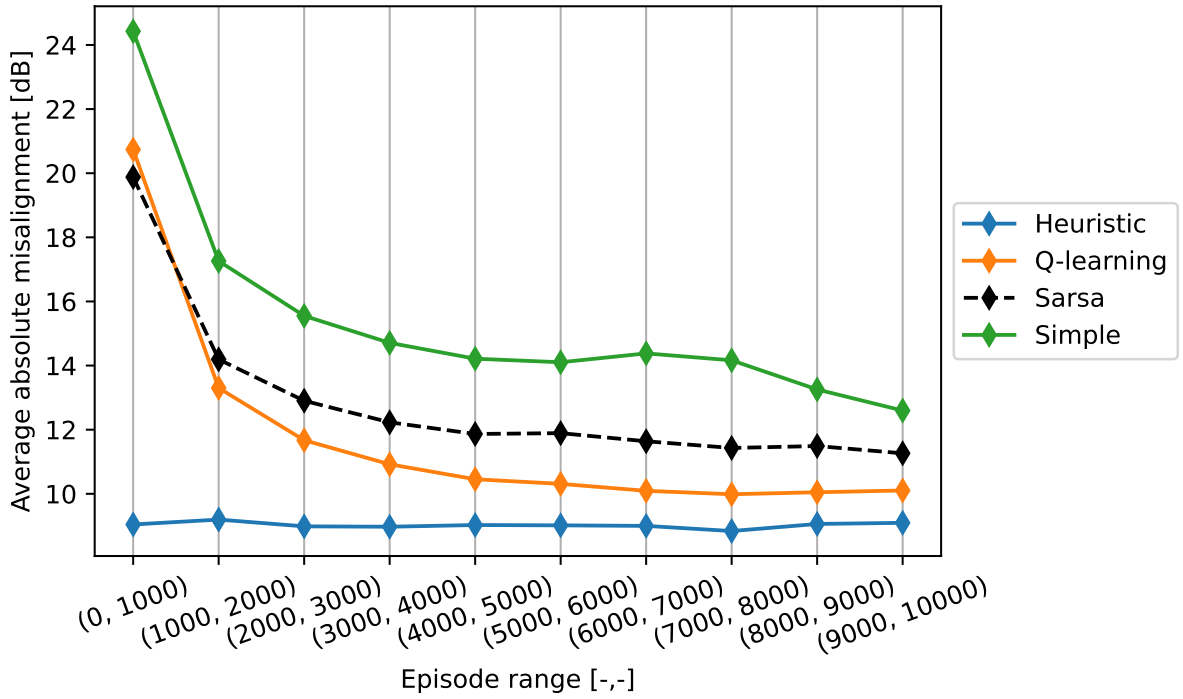


Figure 6.1: Average absolute misalignment of single centralized agents with different update rules.

In Figure 6.1 graphs for three different update rules are plotted. Sarsa, which was used during state and hyper-parameter tuning, is plotted in black. The heuristic algorithm performs well despite it's simplicity, however some of this performance can likely be assigned to the adjacent-action-only encoding. If the received signal power at a given beam-pair goes below the threshold, it is likely that a beam-pair consisting of codewords adjacent to the first pair has high power as well. The Sarsa algorithm gets outperformed by the heuristic by around 2 dB which is interesting, as the scenario is LOS and should be fairly simple. The size of the value space, and the dynamic environment might be to blame for this. Q-learning outperforms Sarsa slightly, and their performance being close isn't unexpected as when ϵ is small their policies should approach each other. While the actual ϵ-values are adapting all the time, they should become smaller as approximations get better. The worst of them all is the simple update rule, indicating that prioritizing only the immediate reward is not a good strategy.

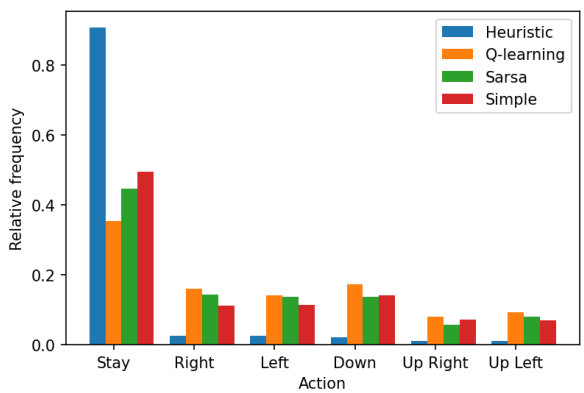


Figure 6.2: Histogram of actions for the UE for single centralized agents with different update rules.

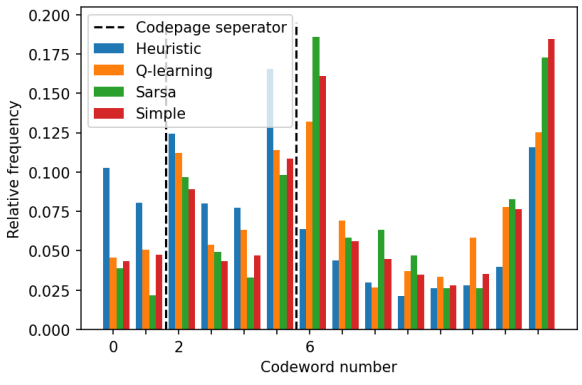


Figure 6.3: Histogram of codewords chosen for the UE for single centralized agents with different update rules

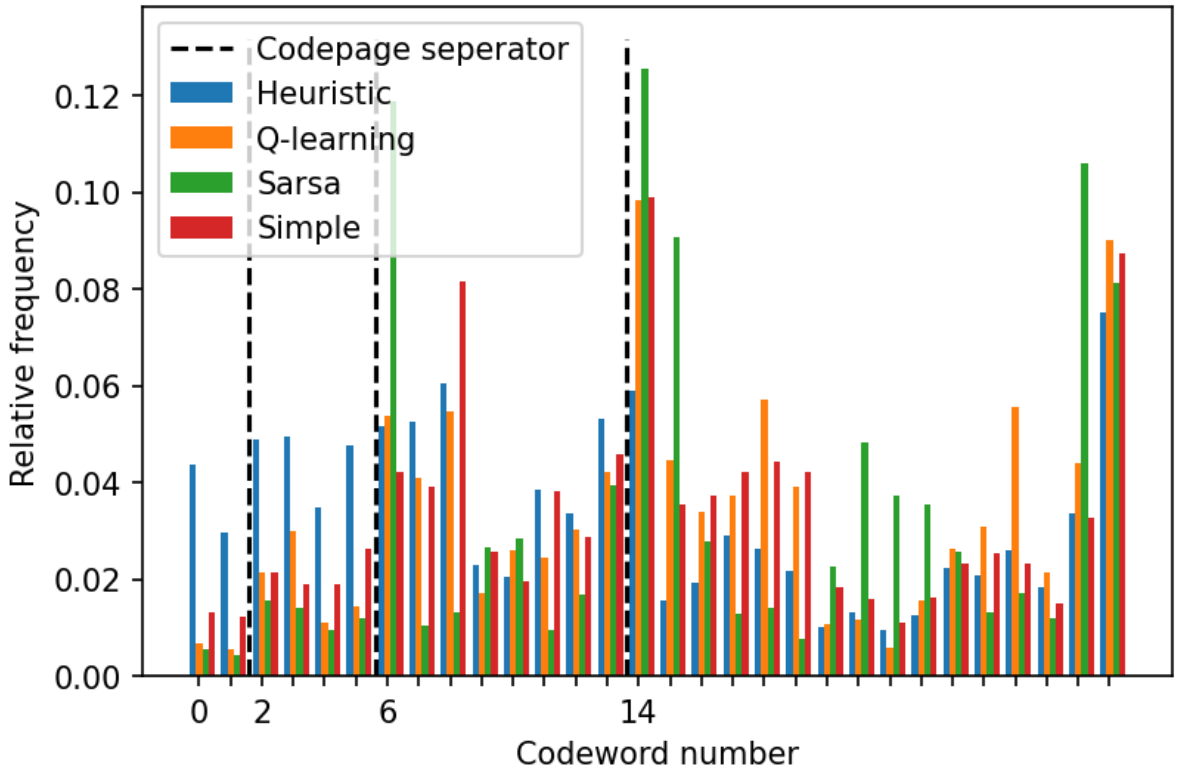


Figure 6.4: Histogram of codewords chosen for the BS for single centralized agents with different update rules

When inspecting the actions, given in Fig. 6.2, as expected the heuristic algorithms chooses the stay most often. Meanwhile the behavior for the three update rules are similar. From Fig. 6.3 and Fig. 6.4 it can be seen that the heuristic algorithm has the greatest propensity to stay in the broader beams, and lower layers of the codebook. A likely explanation is that, when the UE is close to the BS and a broad beam has sufficient power, the heuristic algorithm will stick with it, even more so because of its large coverage. For the UE and to a lesser extent for the BS all update rules tend to gravitate towards beams which lie at the edge of layers in the codebook, see Fig. 4.1. This is not unexpected as these beams, while having broader coverage still have similar gain compared to the other beams in the same layer, making them generally better to choose. This is the case even more so for Sarsa, which is sensible as it's an update which values states that are close to low value state less, unlike Q-learning. The beam choices of Q-learning and simple being more similar than Q-learning and Sarsa is peculiar as the two latter are more similar.

6.3.2 Distributed agents

Table 6.3: Table of different misalignment measurements of multiple distributed agents with different update rules. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Heuristic	9.1 dB	6.5 dB	3.7 dB	10.8 dB
Q-learning	10.6 dB	7.6 dB	3.2 dB	16.0 dB
Sarsa	10.6 dB	7.2 dB	2.5 dB	16.7 dB
Simple	6.4 dB	3.7 dB	0.1 dB	9.2 dB

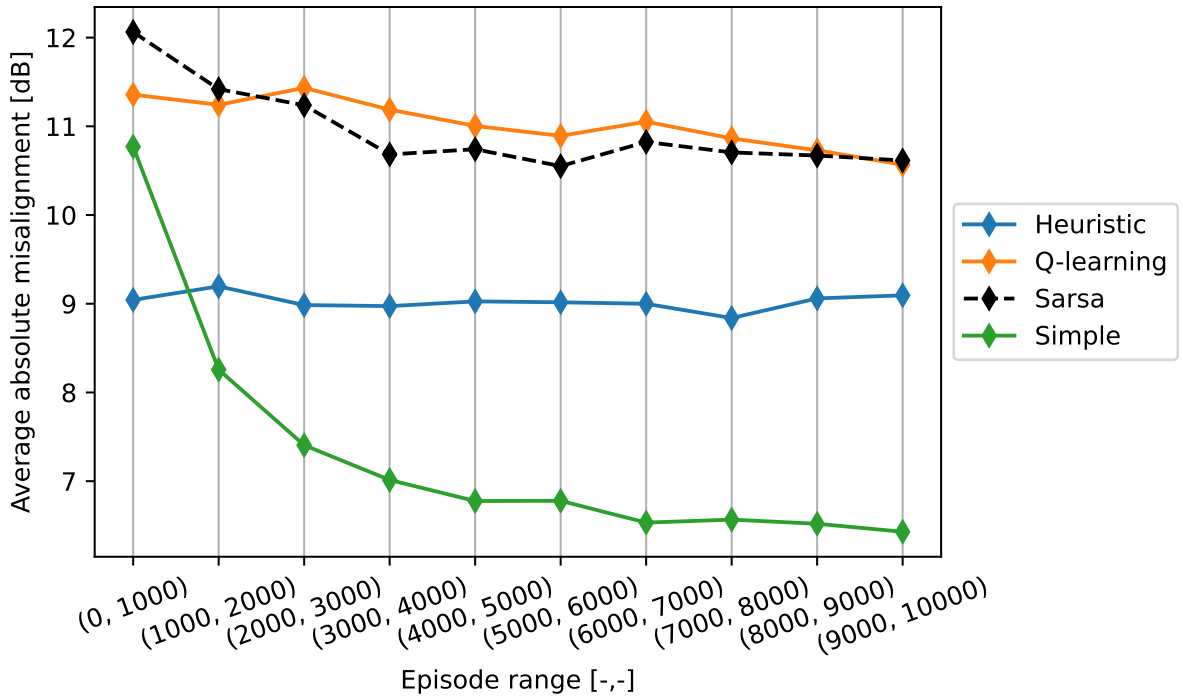


Figure 6.5: Average absolute misalignment of multiple distributed agents with different update rules.

In Fig. 6.5 graphs for three different update rules are plotted. Sarsa, which was used during state and hyper-parameter tuning, is plotted in black. Here Sarsa has very comparable performance to Q-learning and both get outperformed by the heuristic. Based on the single agent results this is expected. Unlike in the single agent case, Q-learning and Sarsa does not require a large amount of episodes to converge to stable performance, which is expected as the state space is smaller.

The simple update rule converges to a considerably better performance than both the heuristic and the other two update methods quickly. This is surprising considering both that it's a simpler method, but also that the results for the single agent case showed something entirely different. However, this could be because the larger state space could have impaired the algorithm in the single agent case.

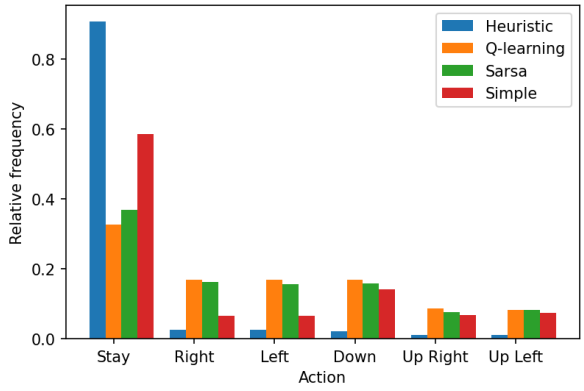


Figure 6.6: Histogram of actions for the UE for multiple distributed agents with multiple update rules.

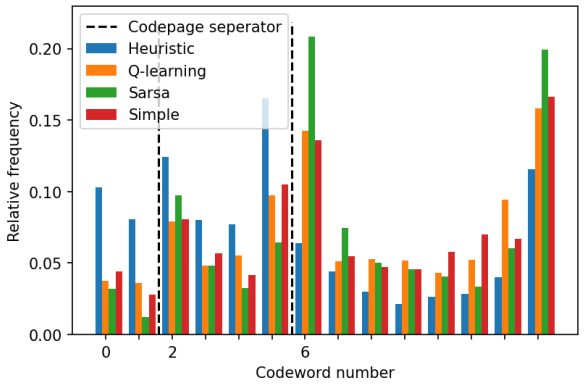


Figure 6.7: Histogram of codewords chosen for the UE for multiple distributed agents with different update rules.

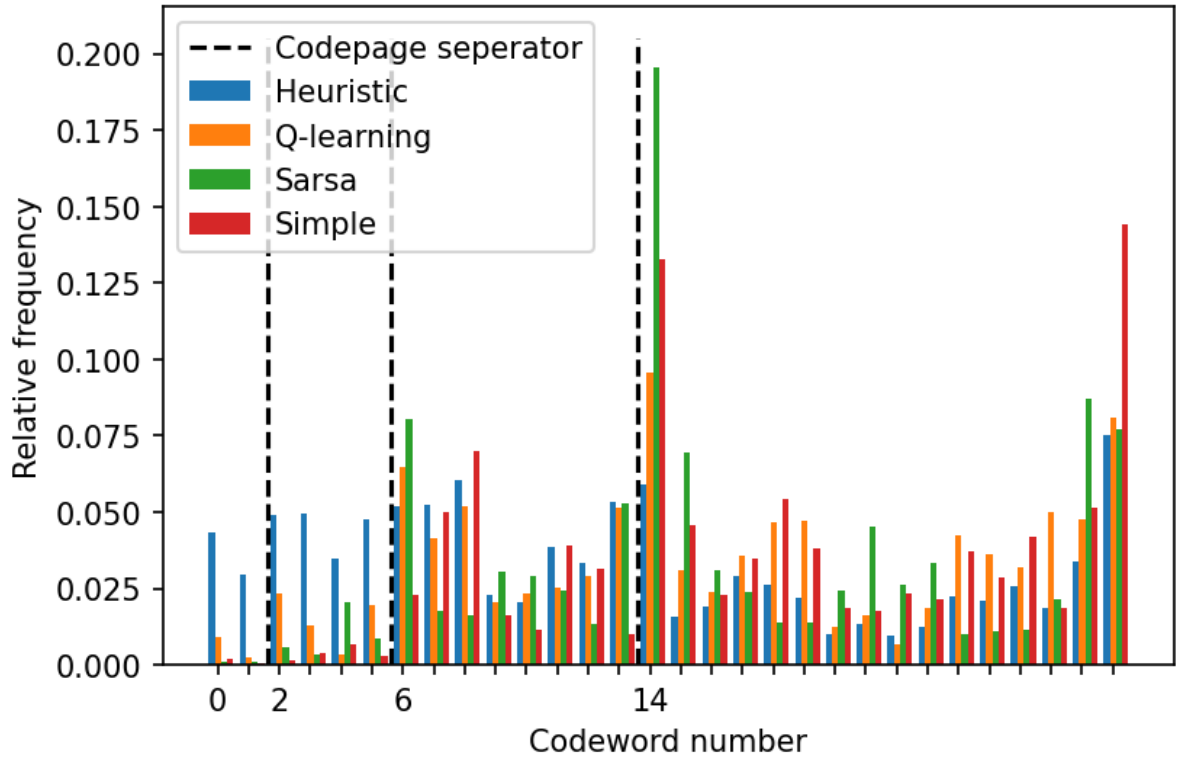


Figure 6.8: Histogram of codewords chosen for the BS for multiple distributed agents with different update rules.

Looking at the actions of the four methods, the heuristic algorithm still favors the stay option more than the update rules. The simple update rule also favors the stay option significantly more than the other update rules, unlike in the single agent case. While the simple algorithm moves up and down layers with a similar frequency as the other update rules it chooses left and right less often, likely meaning the simple algorithm is more likely to stick with a good beam. The beam choices, in Fig. 6.7 and Fig. 6.8 are largely similar to those for the single agent case. The simple algorithm behaves less like Q-learning than in the single agent case, while still being somewhat similar. Again broader beams are largely preferred.

In addition to the update rules above, another algorithm has been implemented for the multi agent case as mentioned in Section 4.1.1. The algorithm is called "Win or Learn Fast - Policy Hill Climbing", WoLF-PHC for short. This is a somewhat more sophisticated multi agent algorithm, which updates action-values using Q-learning but uses policy hill climbing to update the policy based on experience.

Table 6.4: Table of different misalignment measurements of multiple distributed agents with different update rules and the WoLf-PHC algorithm. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Q-learning	10.6 dB	7.6 dB	3.2 dB	16.0 dB
Sarsa	10.6 dB	7.2 dB	2.5 dB	16.7 dB
WOLF	11.0 dB	7.9 dB	2.9 dB	17.1 dB

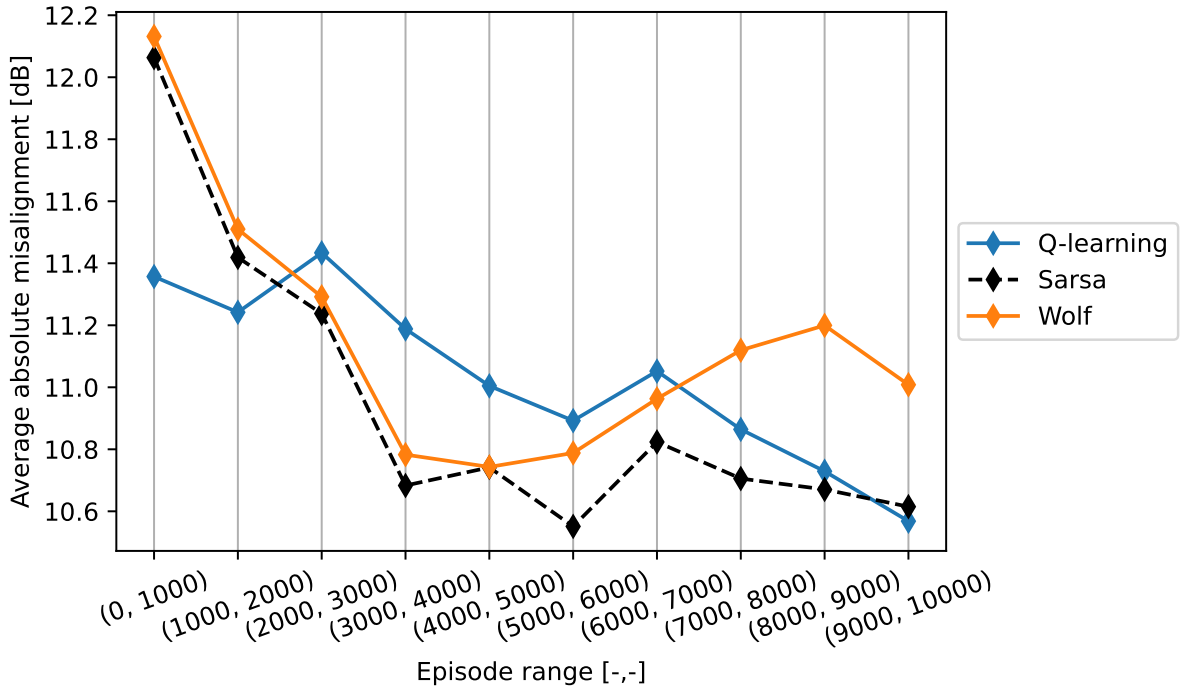


Figure 6.9: Average absolute misalignment of multiple distributed agents with different update rules and the WoLf-PHC algorithm.

In Fig. 6.9 graphs for two different update rules are plotted. Sarsa, which was used during state and hyper parameters optimization, is plotted in black. The performance of the WoLF-PHC algorithm has been plotted for comparison. The performance WoLF-PHC is similar to the two update methods. This isn't unexpected as WoLF-PHC uses the Q-learning update rule, which was previously shown to perform similarly to Sarsa. The fact that it does not improve the performance despite being an algorithm developed specifically for multi agent environments is interesting. However, as it does not give worse performance it might still be viable to investigate other dedicated multi agent RL-algorithms, even though this is outside the scope of this report.

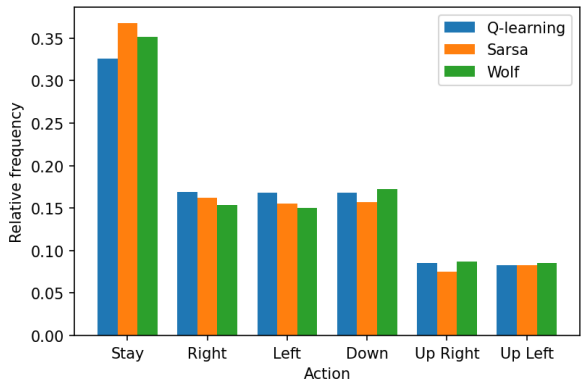


Figure 6.10: Histogram of actions for the UE for multiple distributed agents with different update rules and the WoLf-PHC algorithm.

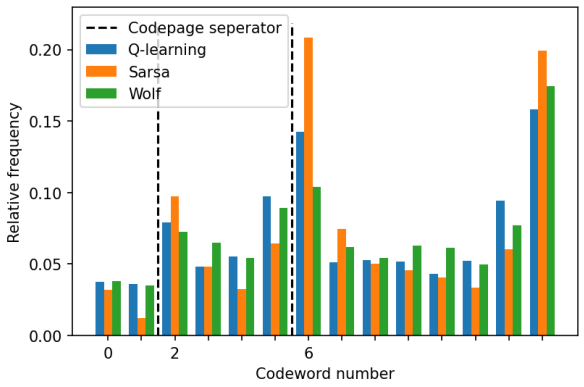


Figure 6.11: Histogram of codewords chosen for the UE for multiple distributed agents with different update rules and the WoLf-PHC algorithm.

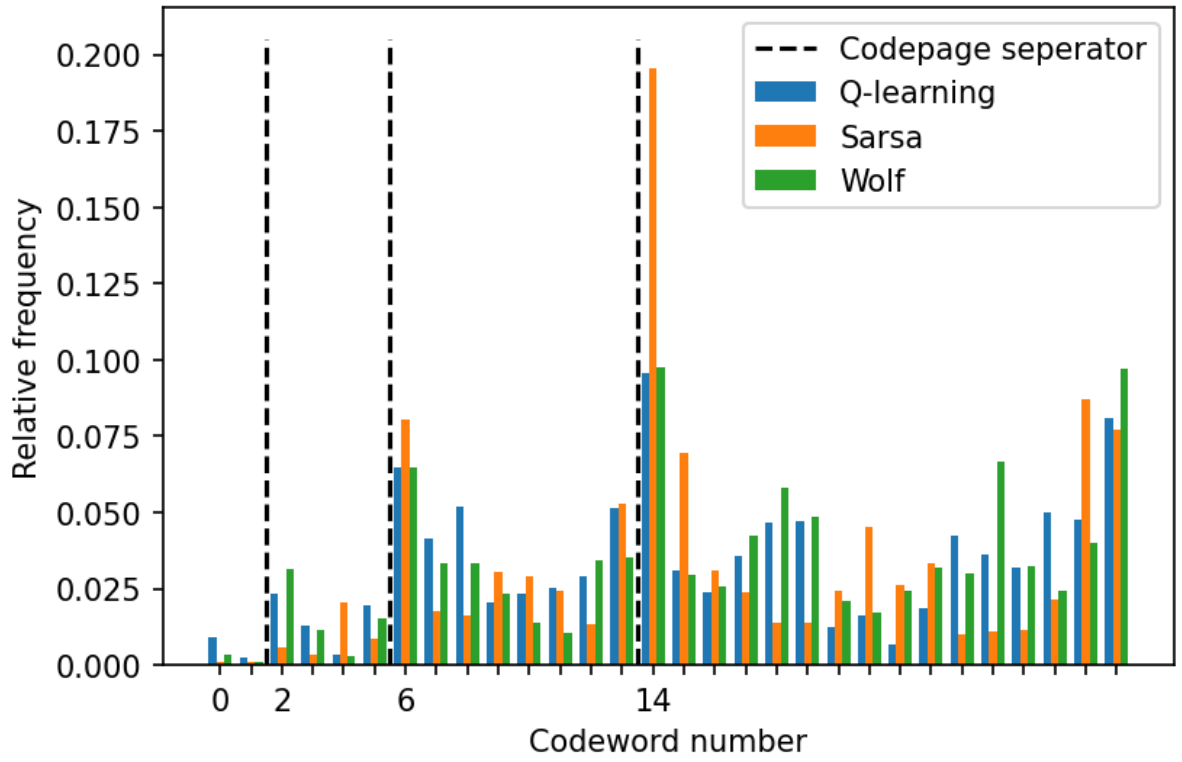


Figure 6.12: Histogram of codewords chosen for the BS for multiple distributed agents with different update rules and the WoLf-PHC algorithm.

The distribution of the actions are almost indistinguishable between the methods. The beams chosen are very similar between Q-learning and WoLF-PHC which isn't surprising given their similarity in both their update rule and performance.

6.4 Environment

The agent settings has been tuned using a line-of-sight scenario, with pedestrian movement tracks. It is thus of interest to investigate how the agent performs when the environment changes to a non-line-of-sight scenario and if the performance seen in pedestrian movement

tracks is replicated when car movement tracks are used instead. In Chapter 2 it is mentioned that one of the differences between the two movement models are that the car tracks move over longer distances and has larger turn radii. Additionally, for the car the orientation of the UE is fixed to the direction of movement, where in pedestrian movement the orientation is more erratic.

6.4.1 Single centralized agent

Pedestrian case

Table 6.5: Table of different misalignment measurements of single centralized agents in different environments in the pedestrian case. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
LOS Heuristic	9.1 dB	6.5 dB	3.7 dB	10.8 dB
LOS Sarsa	11.3 dB	7.7 dB	2.6 dB	17.6 dB
NLOS Heuristic	24.6 dB	24.7 dB	16.3 dB	32.3 dB
NLOS Sarsa	24.1 dB	24.1 dB	15.8 dB	31.8 dB

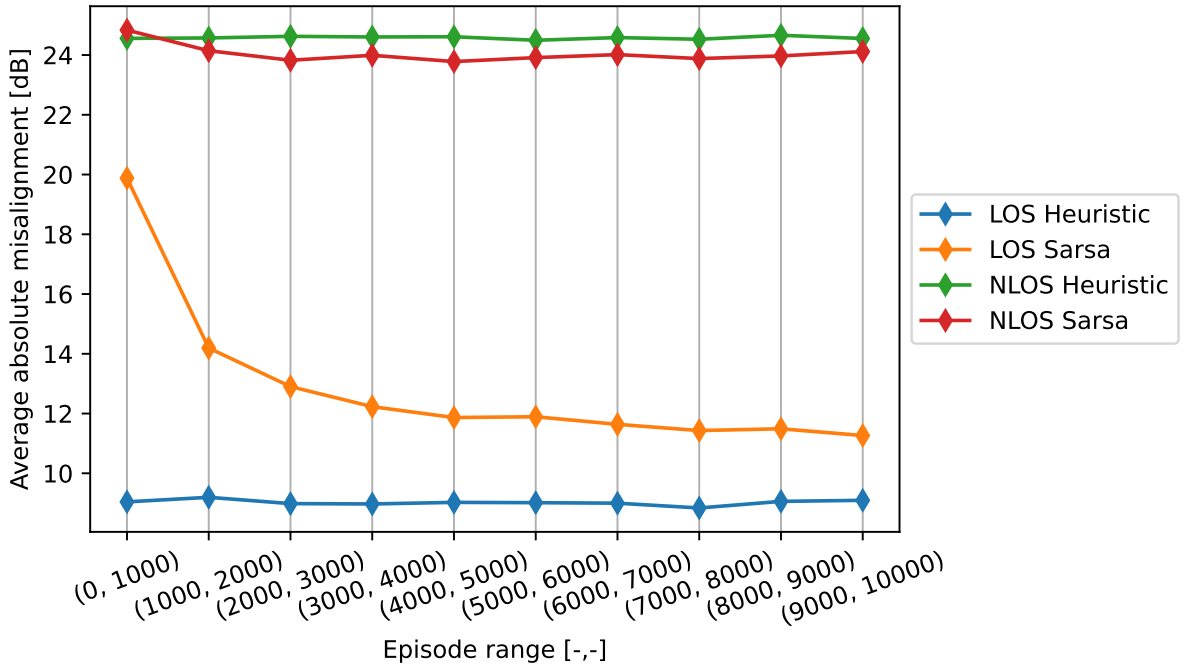


Figure 6.13: Average absolute misalignment of single centralized agents in different environments in the pedestrian case.

In Fig. 6.13 the performance of the tuned centralized Sarsa agent is shown, together with the heuristic for comparison. The performance of the two methods in the LOS scenario is significantly better than in the NLOS, as expected. The signal strength should be less erratic and the best signal direction shouldn't change drastically over short periods in LOS compared to NLOS. The Sarsa performing worse in the NLOS scenario is also partially explained by the fact the state lends itself better to the LOS scenario rather than NLOS. Pieces of information that would be useful in the NLOS scenario like information about the geometry of the environment is considerably more costly to obtain and process.

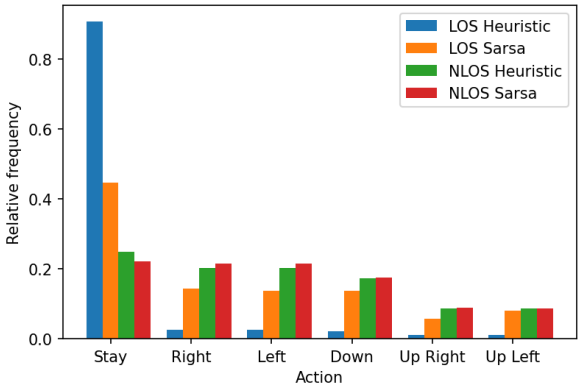


Figure 6.14: Histogram of actions for the UE for single centralized agents in different environments in the pedestrian case.

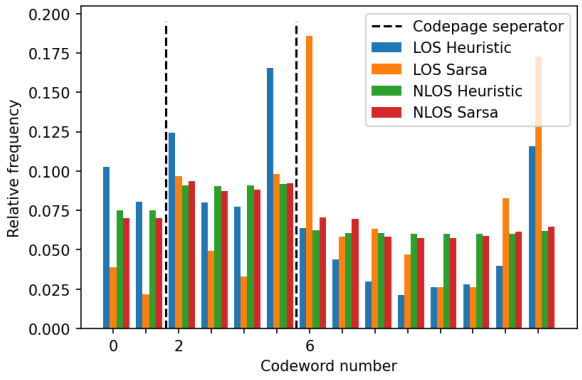


Figure 6.15: Histogram of codewords chosen for the UE for single centralized agents in different environments in the pedestrian case.

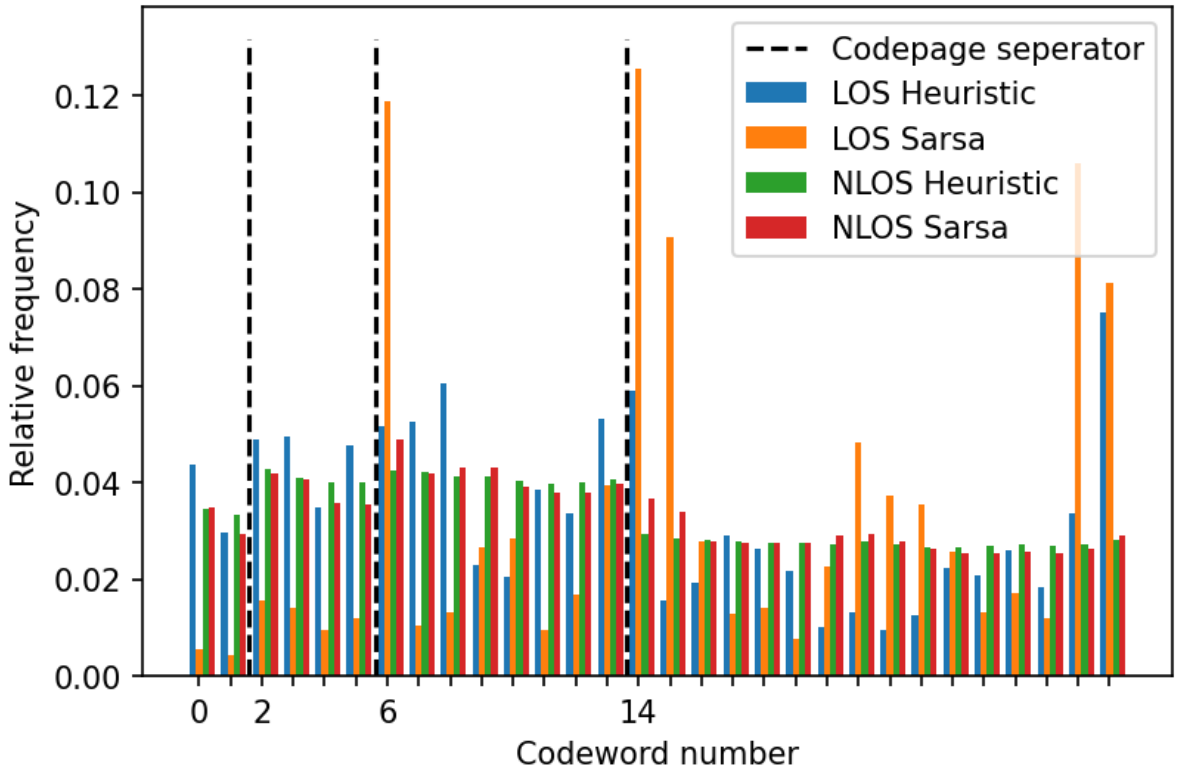


Figure 6.16: Histogram of codewords chosen for the BS for single centralized agents in different environments in the pedestrian case.

Looking at the actions taken in the NLOS conditions compared to the LOS conditions, they are more evenly distributed among the possible actions for both Sarsa and the heuristic algorithm. It's sensible the action stay would be less useful as the NLOS scenario is more dynamic, which likely would result in good codeword choices to shift around more.

Looking at Figures 6.15 and 6.16, the codewords used in NLOS are drastically different than in LOS, as here neither algorithm is inclined to use codewords close to the edge of the codepages. Instead, all codewords on any given codepage seems chosen equally often. One would still expect the codewords at the edges of codepages to be more valuable by virtue of their larger coverage,

this might still be the case as these choices aren't optimal as seen on Fig. 6.13.

Car case

Table 6.6: Table of different misalignment measurements of single centralized agents in different environments in the car case. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
LOS Heuristic	8.0 dB	6.1 dB	3.0 dB	10.7 dB
LOS Sarsa	7.9 dB	5.2 dB	1.5 dB	11.5 dB
NLOS Heuristic	23.6 dB	23.4 dB	14.8 dB	31.9 dB
NLOS Sarsa	23.1 dB	22.8 dB	14.0 dB	31.4 dB

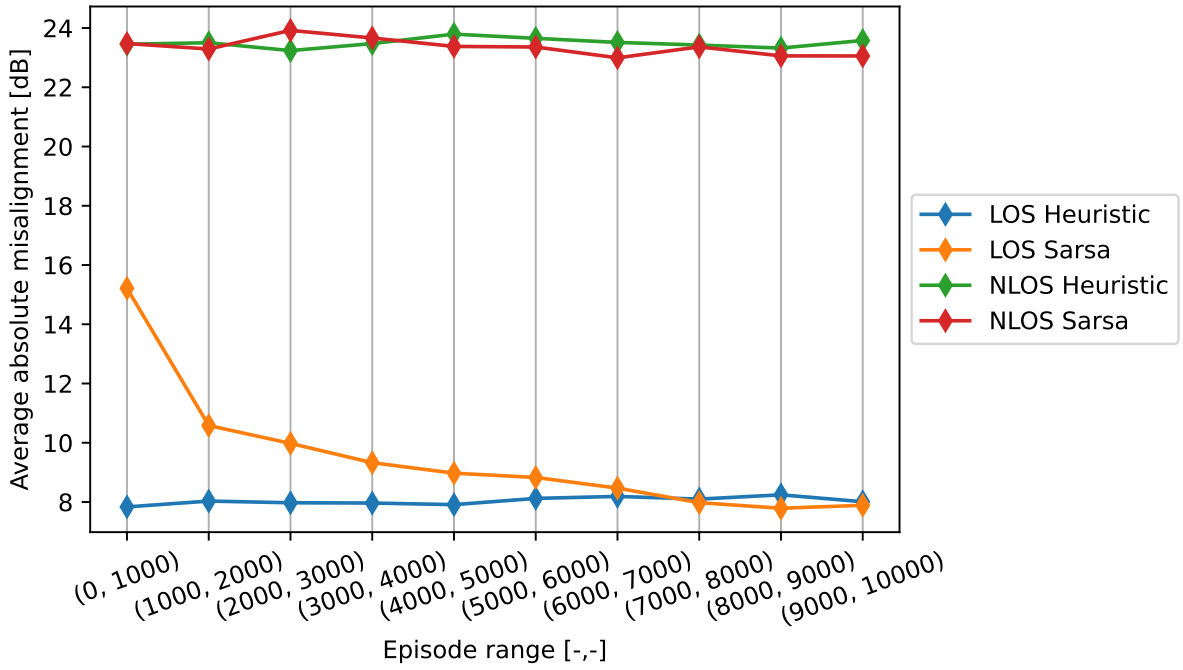


Figure 6.17: Average absolute misalignment of single centralized agents in different environments in the car case.

In Fig. 6.17 the performance of the tuned centralized Sarsa agent is shown, together with the heuristic for comparison. The performance in the car case is similar to that of the pedestrian case, however with some improvement. This isn't unexpected as both cases are somewhat similar. The small performance improvement likely comes from the fact that the UE has a less random orientation, because it's simply the movement direction of the vehicle. This has also resulted in the Sarsa agent reaching similar performance as the heuristic, which means the agent was capable of exploiting the simpler environment more than the heuristic algorithm.

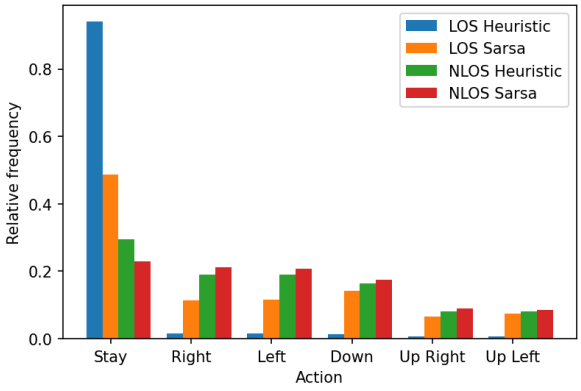


Figure 6.18: Histogram of actions for the UE for single centralized agents in different environments in the car case.

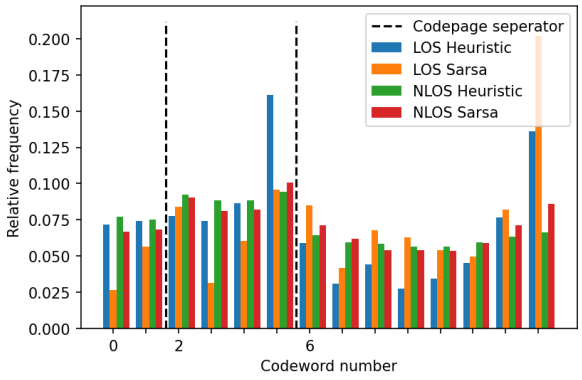


Figure 6.19: Histogram of codewords chosen for the UE for single centralized agents in different environments in the car case.

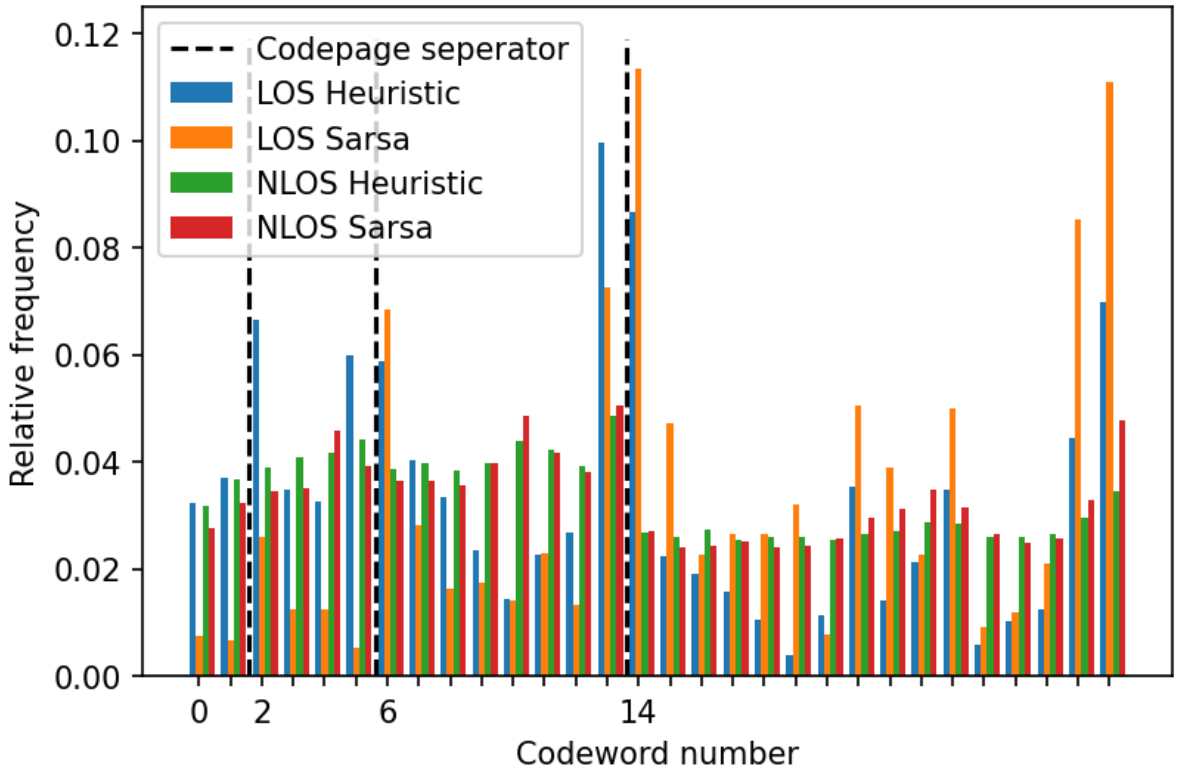


Figure 6.20: Histogram of codewords chosen for the BS for single centralized agent in different environments in the car case.

The actions, seen in Fig. 6.18, largely resemble those for the pedestrian case, which isn't unexpected due to the similarity of the cases. The codeword choices for the NLOS scenario are close to being equally distributed, with a slight preference toward the edges of codepages. For the BS some codewords close to the middle of codepages have been used more often, why this is the case is unknown. Again the performance isn't impressive, so any information interpreted from the graphs can only be used as example of what not to do.

6.4.2 Distributed agents

Pedestrian case

Table 6.7: Table of different misalignment measurements of multiple distributed agents in different environments for the pedestrian case. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
LOS Heuristic	9.1 dB	6.5 dB	3.7 dB	10.8 dB
LOS Sarsa	10.6 dB	7.2 dB	2.5 dB	16.7 dB
NLOS Heuristic	24.6 dB	23.4 dB	16.3 dB	32.3 dB
NLOS Sarsa	17.7 dB	10.5 dB	9.4 dB	24.8 dB

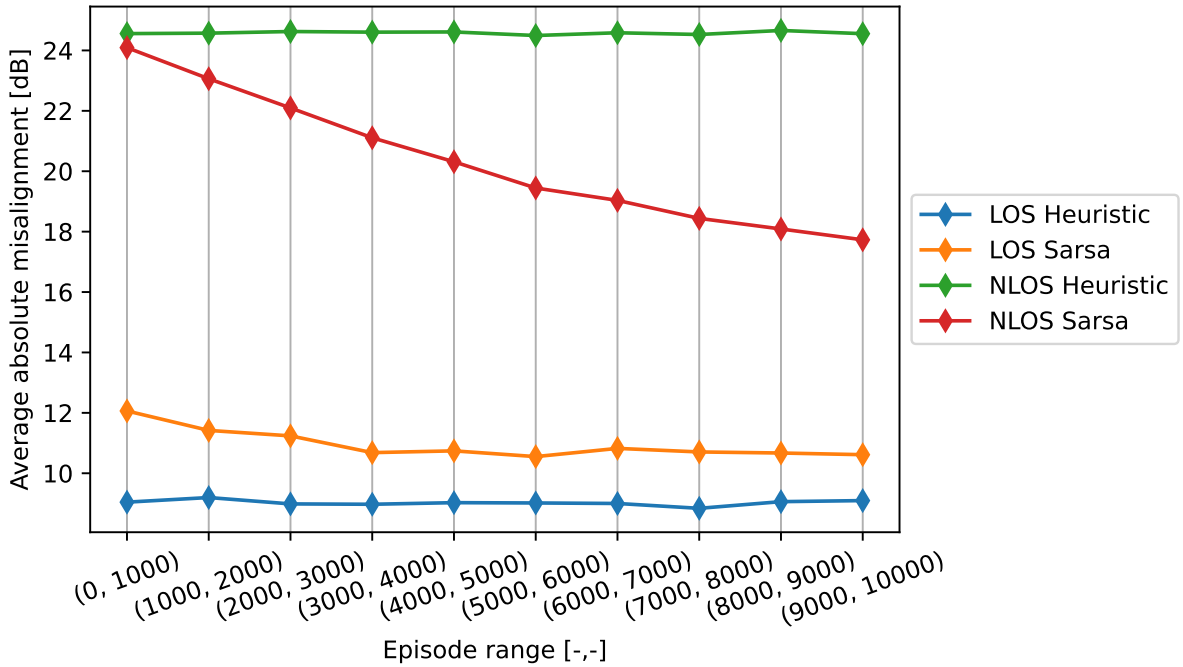


Figure 6.21: Average absolute misalignment of multiple distributed agents in different environments for the pedestrian case.

In Fig. 6.21 the performance of the optimized Sarsa multi agents is shown, together with the heuristic for comparison. Unlike in the single agent case, here the agents are learning continuously during the entire simulation in the NLOS scenario. While learning naturally would be slower in the single agent case because of the larger state space, it's performance seemed like it has converged. The fact that multi agent not only beats it in NLOS but does so convincingly, is interesting as the single agent would have an easier time coordinating a pair of beams along a strong reflected path. Regardless, the multi agent case outperforms the heuristic algorithm by a wide margin in NLOS.

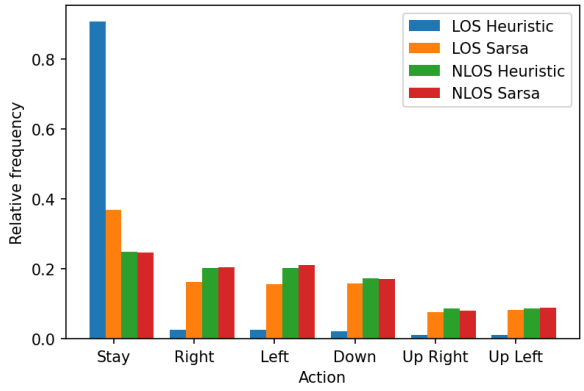


Figure 6.22: Histogram of actions for the UE for multiple distributed agents in different environments for the pedestrian case.

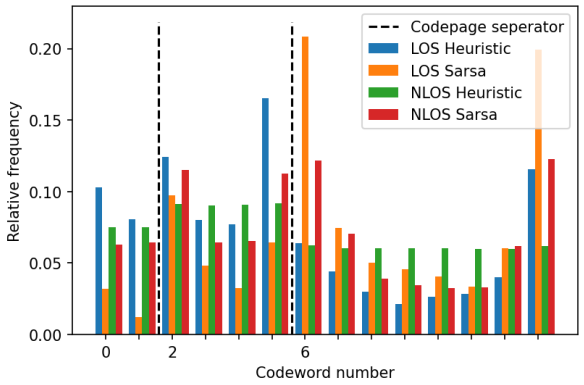


Figure 6.23: Histogram of codewords chosen for the UE for multiple distributed agents in different environments for the pedestrian case.

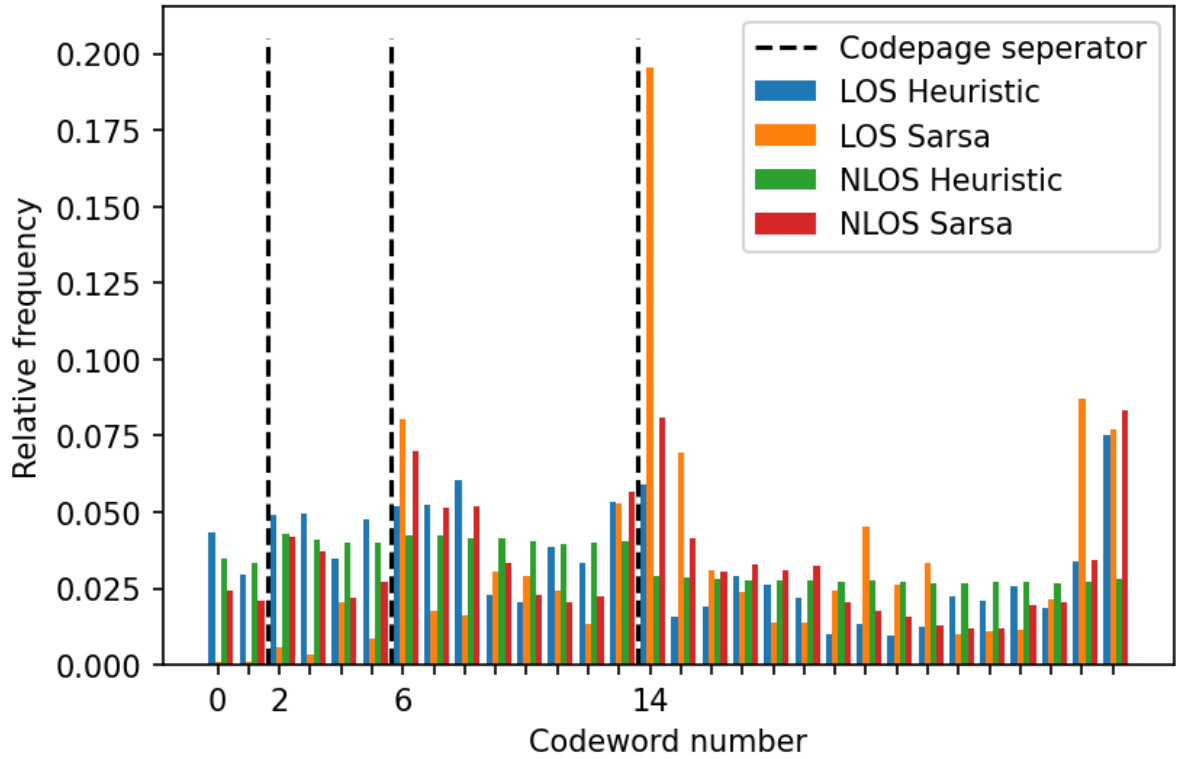


Figure 6.24: Histogram of codewords chosen for the BS for multiple distributed agents in different environments for the pedestrian case.

Like in the single agent case, the choices for NLOS, as reflected in Fig. 6.22, are more evenly distributed. This means an equal action distribution is not exclusively a trait of bad performance, at least not in the NLOS scenario. When inspecting the codeword distributions for Sarsa in NLOS it's close to that of LOS, where the edges of codepages are preferred. This indicates that such codeword distributions are correlated with having good performance. This behavior is still slightly less pronounced for NLOS than LOS, however the performance is also worse in NLOS. The slight preference for codewords in the lower half of codepages is interesting, and might point toward trends in the data set, however the actual cause is unknown.

Car case

Table 6.8: Table of different misalignment measurements of multiple distributed agents in different environments for the car case. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
LOS Heuristic	8.0 dB	6.1 dB	3.0 dB	10.7 dB
LOS Sarsa	7.9 dB	4.8 dB	1.6 dB	11.5 dB
NLOS Heuristic	23.6 dB	23.4 dB	14.8 dB	31.9 dB
NLOS Sarsa	12.7 dB	10.5 dB	4.0 dB	19.5 dB

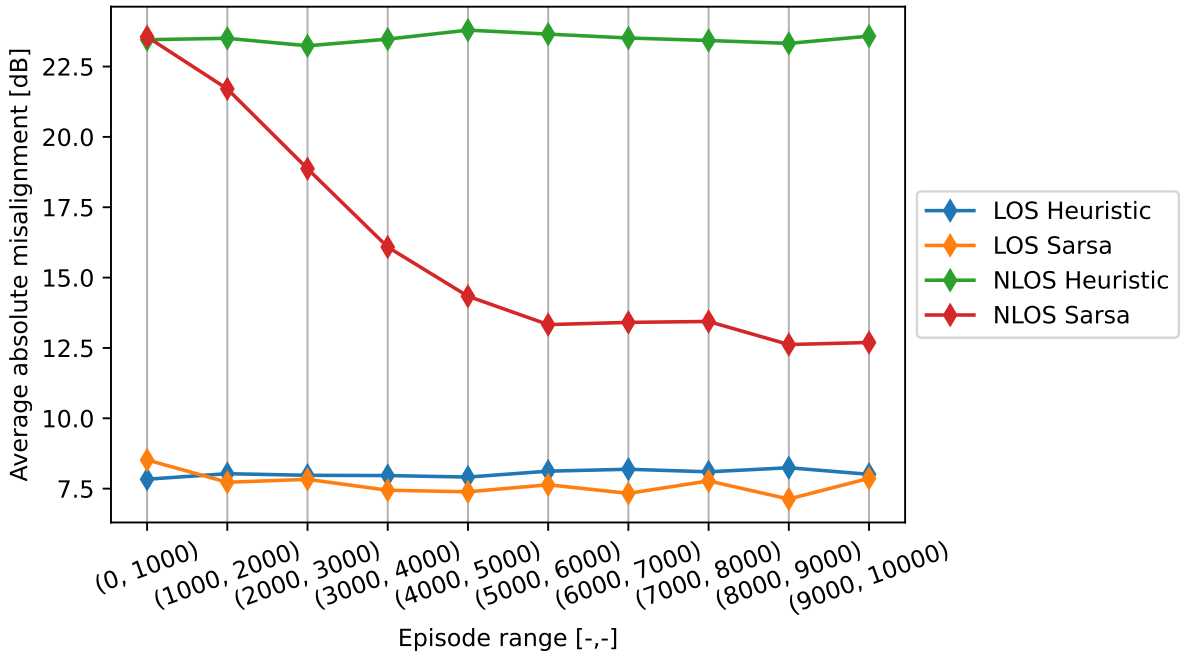


Figure 6.25: Average absolute misalignment of multiple distributed agents in different environments for the car case.

In Fig. 6.25 the performance of the optimized Sarsa multi agents is shown, together with the heuristic for comparison. This results is not unexpected based on the pedestrian multi agent and car single agent results. This performance closely mirrors that of the pedestrian case, although slightly better, with the exception of the NLOS Sarsa which is significantly better. The less challenging environment of the car case is likely the cause.

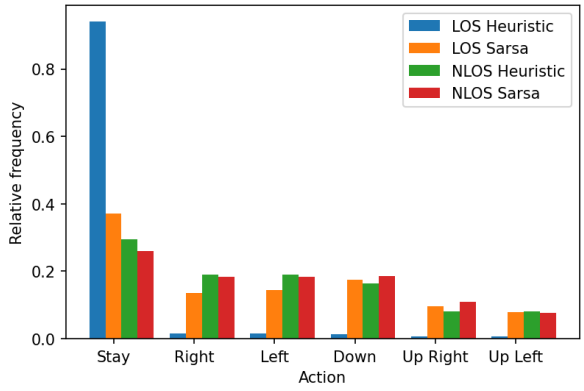


Figure 6.26: Histogram of actions for the UE for multiple distributed agents in different environments for the car case.

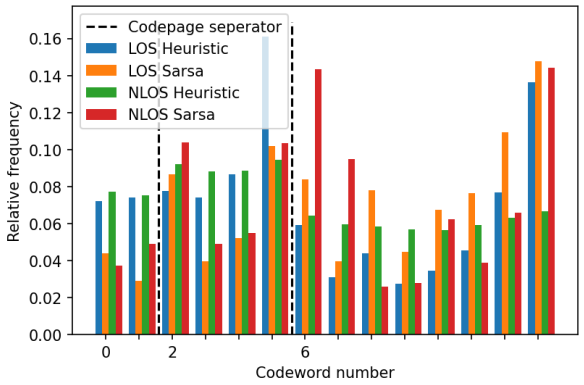


Figure 6.27: Histogram of codewords chosen for the UE for multiple distributed agents in different environments for the car case.

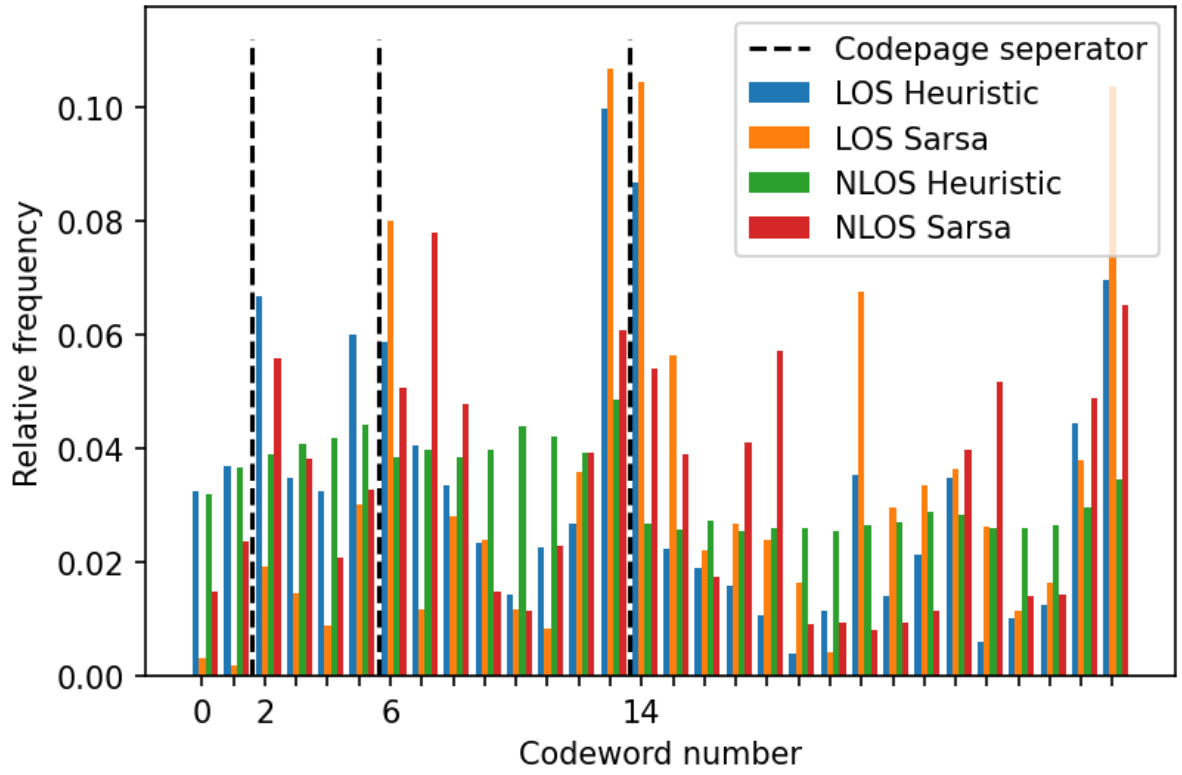


Figure 6.28: Histogram of codewords chosen for the BS for multiple distributed agents in different environments for the car case.

The actions, in Fig. 6.26, resemble those of the pedestrian case. The behavior for Sarsa in car NLOS has the same preference for the edges of codepages, especially for the UE in Fig. 6.27. The behavior for the BS is slightly different as it also prefers a couple of codewords somewhere between the edges and middle of the codepages. While previously unseen behavior it's evidently very efficient in the NLOS scenario in comparison to both the heuristic and single agents behavior.

6.5 Noise

Finally it is of interest to know how robust the implemented algorithms are towards noise in the received signal power. As mentioned in Chapter 4, only the thermal noise and the noise factor from the circuits at the transceivers are considered. The noise is modelled as additive white circular Gaussian noise:

$$n \sim \mathcal{CN}(0, P_n)\,, \tag{6.1}$$

where
P_n is the noise power. As the noise is considered to be thermal noise, the noise power can calculated as:

$$P_n = k_\mathrm{B} \cdot T \cdot B\,, \tag{6.2}$$

where

$k_\mathrm{B}:$	$[J/K]$	Boltzmann constant
$T:$	$[K]$	Temperature
$B:$	$[Hz]$	Bandwidth

Under normal room temperature the temperature is 290 K, and the bandwidth used for mmWave systems is here assumed to be 400 MHz, which results in a noise power of approximately −88 dBm.

In addition to the thermal noise, which can't be changed, it is also interesting to investigate the performance if using less ideal equipment that add further noise due to its noise figure. The noise figure is based on the assumptions made in the 3GPP channel model specifications, of 9 dB [16]. However, as the previous tests were all performed with a transmission power of 40 dBm, which is 10 dBm higher than in usual mmWave scenarios, an additional 10 dBm is added to the noise in both cases.

Single centralized agent

Table 6.9: Table of different misalignment measurements of multiple distributed agents with different amounts of noise. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Heuristic: No noise	9.1 dB	6.5 dB	3.7 dB	10.8 dB
Heuristic: Thermal noise	9.1 dB	6.6 dB	3.7 dB	10.7 dB
Heuristic: Noise factor	8.9 dB	6.5 dB	3.6 dB	10.7 dB
Sarsa: No noise	11.3 dB	7.7 dB	2.6 dB	17.6 dB
Sarsa: Thermal noise	10.2 dB	7.4 dB	3.5 dB	14.6 dB
Sarsa: Noise factor	9.9 dB	6.9 dB	3.2 dB	14.2 dB

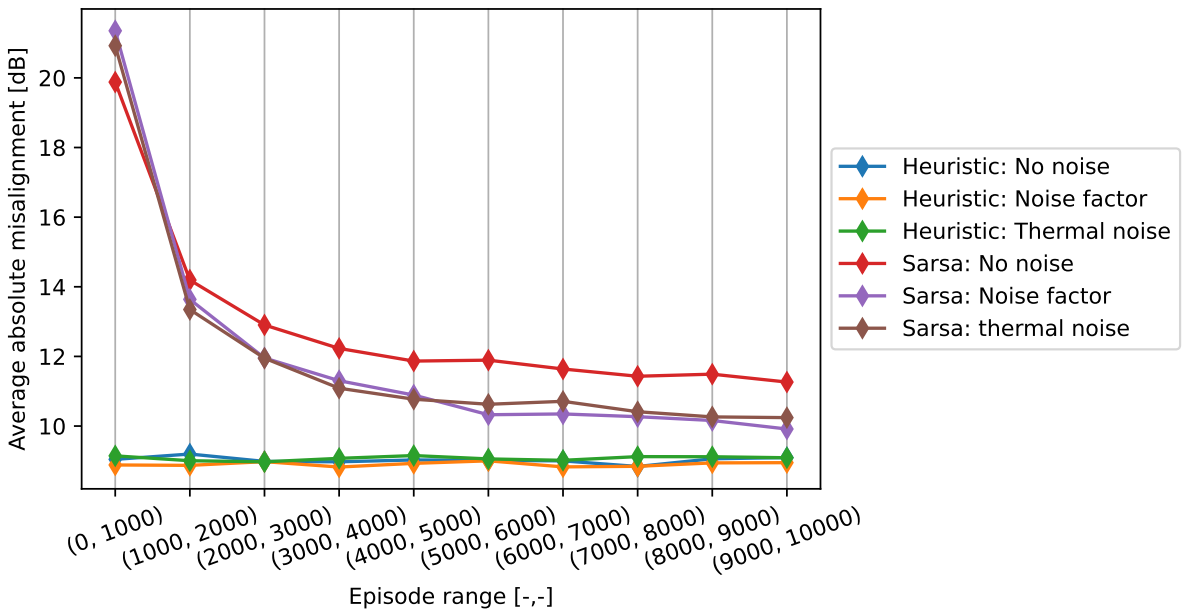


Figure 6.29: Performance of single centralized agents with different amounts of noise.

In Fig. 6.29 curves for the performance of a single centralized agent in a noisy environment is plotted. From this it's clear that the heuristic algorithm suffers no degradation of performance in noisy environments, at least for the LOS scenario. The agents performance actually improves when noise is added which unexpected. Noise is expected to make the value estimates worse, which would normally result in worse performance. This difference could be because of the variations within each test, however the fact that the two noise tests produce practically identical results would imply otherwise. Thus no real explanation for the difference has been found. The fact that the performance is equal between the two noise levels is also unexpected, as one noise level is approximately 40 dBm higher than the other. A possible explanation is that noise of this level still only meaningfully affects beam pairs with already poor signal power. If the performance of the agent in general results in high signal power, then the noise would only affect the small amount of actions in which the signal power is low, and as such would only result in a slight degradation of performance.

Distributed agents

Table 6.10: Table of different misalignment measurements of multiple distributed agents with different amounts of noise. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Heuristic: No noise	9.1 dB	6.5 dB	3.7 dB	10.8 dB
Heuristic: Thermal noise	9.1 dB	6.6 dB	3.7 dB	10.7 dB
Heuristic: Noise factor	8.9 dB	6.5 dB	3.6 dB	10.7 dB
Sarsa: No noise	10.6 dB	7.2 dB	2.5 dB	16.7 dB
Sarsa: Thermal noise	10.7 dB	7.4 dB	3.2 dB	16.3 dB
Sarsa: Noise factor	10.7 dB	7.7 dB	3.1 dB	16.2 dB

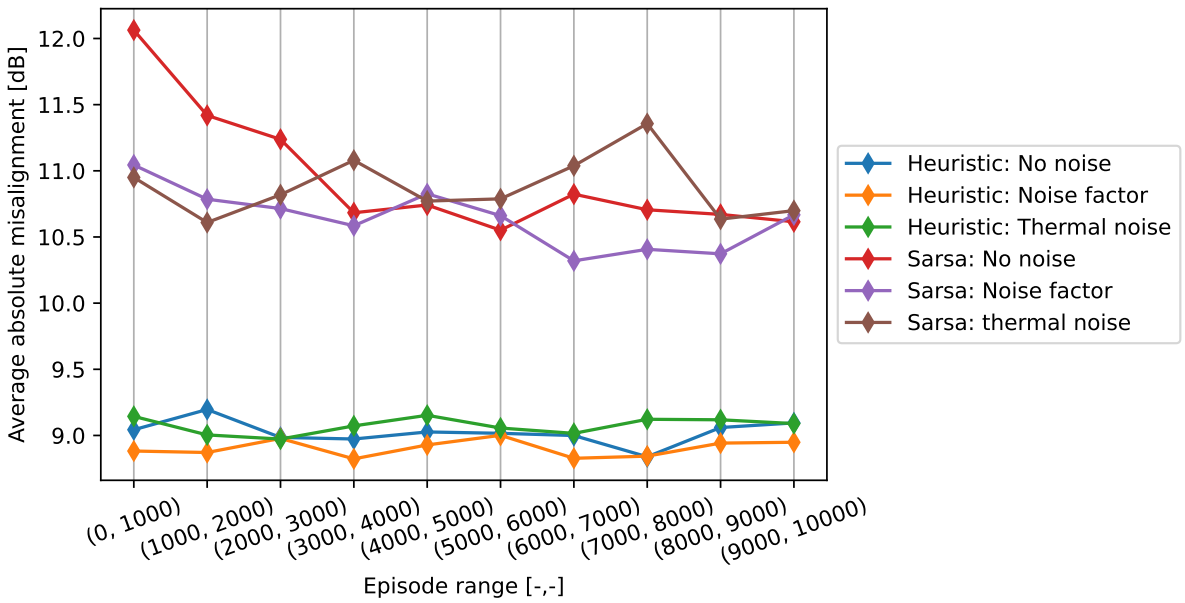


Figure 6.30: Performance of multiple distributed agents with different amounts of noise.

In Fig. 6.30 in can be seen the noise affects neither the heuristic algorithm nor the Sarsa agent very much. While unexpected this is still more reasonable than for the single agent case where noise improved performance. The step size could be so low to the point where the levels of noise added does not matter, which would mean the agent is resilient to noise.

6.6 Simple update rule

Because of the performance depicted in Fig. 6.5, where the simple update rule outperformed all other methods considerably, the simple update rule will be further evaluated here. The update rule will only be tested for the multi agent case as only it showed promise. It will be tested in different environments, as multi and single agent was in Section 6.4, to discern whether this performance improvement also applies to the NLOS scenario, and car case. The test methodology is identical to Section 6.4.

Table 6.11: Table of different misalignment measurements of multiple distributed agents with the simple update rule in different environments. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
LOS Car	4.2 dB	2.3 dB	0.0 dB	5.6 dB
NLOS Car	10.9 dB	8.6 dB	4.0 dB	15.7 dB
LOS Pedestrian	6.4 dB	3.7 dB	0.1 dB	9.2 dB
NLOS Pedestrian	13.4 dB	11.9 dB	6.1 dB	19.2 dB

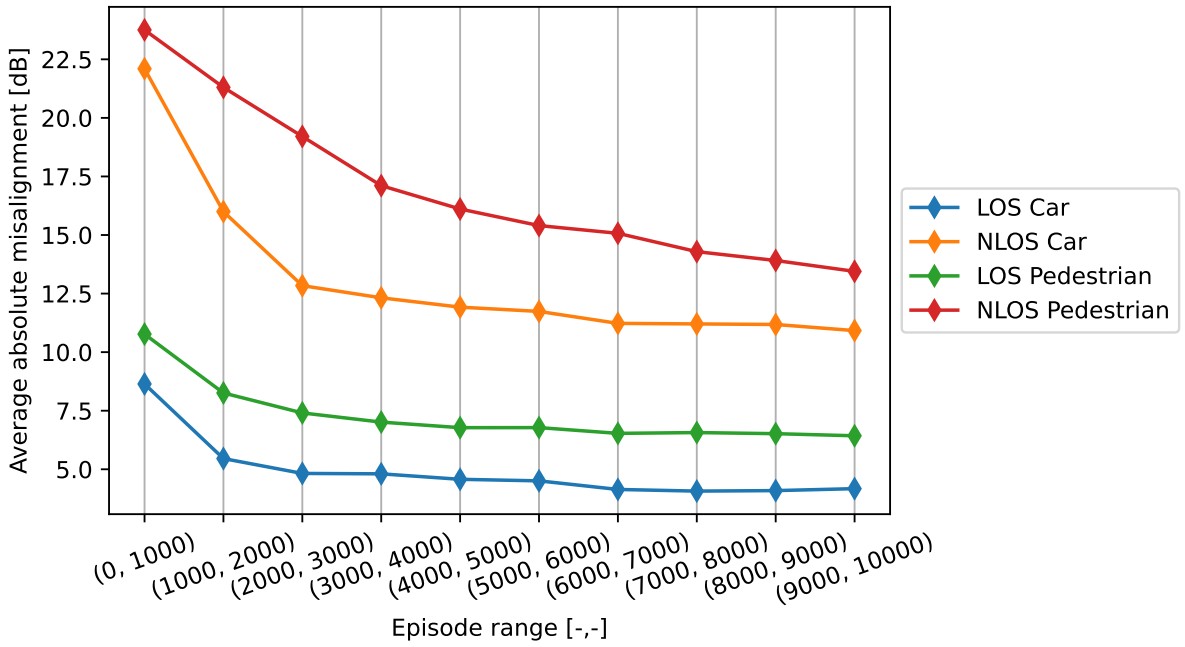


Figure 6.31: Performance of multiple distributed agents with the simple update rule in different environments.

In Fig. 6.31 four curves are given, all for the simple update rule in different environments. In the LOS pedestrian case, the multi agent Sarsa algorithm had an average absolute misalignment of 10.6 dB, where it can be seen, in Table 6.11, that the simple update outperforms it by about 4 dB. The case is the same in the NLOS pedestrian case where the difference is 4.4 dB. This difference becomes larger, relatively, in the car case. Here, the multi agent Sarsa algorithm obtains an average absolute misalignment of 7.9 dB, where the simple update rule has a misalignment almost half as large. The difference is less staggering in car NLOS where Sarsa obtains 12.7 dB and simple 10.9 dB. The simple update rule outperforming Sarsa, but only in multi agent environments, can be partially caused by the environment being more dynamic than in the single agent case. In the multi agent case, an additional random element has been added to the environment for each agent, namely the other agent. Thus it might make less sense for the agent to look very far ahead when estimating the action-values, as even though it follows its own policy, the policy of the other agent may change and thus predicting the expected return becomes harder. However, this still does not account for not looking into the future at all resulting in such good performance.

6.7 Summary

In this chapter the tuned parameter values from Chapter 5 have been tried in various different scenarios to see how well they generalize. From Section 6.3 it was proven that the values in general seem to generalize to other update rules, as both Q-learning and the simple update rule has good performance. In the multi agent case, the simple update rule even outperformed all other methods. For all update rules, the learned behavior of the agent(s) seems to be a lot of the time in the edge beams of the codepages. This makes intuitively good sense, as these beams are broader than the others in a codepage.

In Section 6.4 the tuned agents were put into environments drastically different than those used

during tuning. When used on movement tracks corresponding to the car case, the performance overall gets a bit better, which is probably because the orientation of UE is more stationary. Additionally, these tracks are on average more straight and with softer curves, meaning that the best beam-pairs only change slowly over time, making tracking easier.

When used in NLOS conditions, the centralized agent seems to chose codewords more at random, likely because direction with best signal changes rapidly and the agent tries to keep up with this. Thus the action-value estimates quickly becomes outdated or maybe they just become very similar for all actions, making the agent choose actions seemingly at random. However, the case with distributed agents the results were something else entirely. Here the agents succeeded in learning a similar behavior as in the LOS conditions, leading to a significant increase in performance compared to the centralized agent. It is speculated that because of the smaller state space, it is possible to obtain more accurate action-value estimates, even in NLOS conditions.

In Section 6.5 the tuned agents were placed in a noisy environment, to investigate how robust the RL-algorithm is against noise. Two levels of noise were tested, one consisting only of thermal noise and one with a noise figure of 9 dB added as well. Overall, the performance didn't change much in the noisy environments compared with the noise-free one, but this might just be because the added amount of noise was too low. However, as the total noise added in the environment with both thermal noise and the noise figure was around -38 dBm, it is concluded that the RL-algorithm is somewhat robust against noise.

Finally, based on the experiments in this chapter, it is concluded that to obtain the best performance an implementation with distributed agents using the simple update rule should be chosen. Compared with all other methods it is clearly the one with best performance, and in addition to this it also uses a very small state space, which should comfortably fit into a UE such as a phone. To further examine the performance and behavior of this implementation, the ECDF of the four tests in Fig. 6.31 have been plotted below. In the LOS scenario, for both the pedestrian and car cases, an alignment of 0 dB have been obtained at least one fourth of the time. This is expected as the optimal pair of codewords almost always correspond to the LOS which doesn't change erratically. In the NLOS a single pair of codewords are less likely, than LOS, to outperform every other pair by a large amount, which makes it more difficult to distinguish. Strong paths in NLOS are also more likely to have similar signal strength, and with the adjacent action encoding it may not be optimal to try to navigate to the strongest path every time.

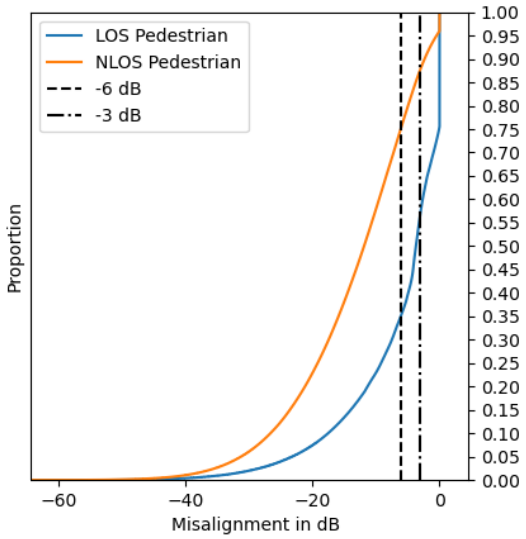


Figure 6.32: ECDF of multiple distributed agents with the simple update rule in different environments for the pedestrian case. Results are from the last 1000 episodes in each test.

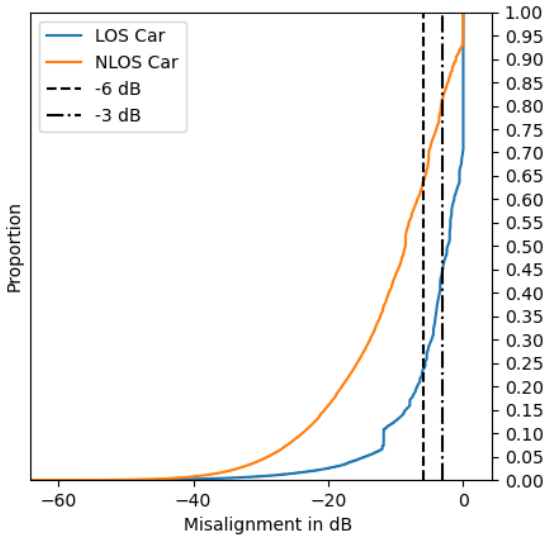


Figure 6.33: ECDF of multiple distributed agents with the simple update rule in different environments for the car case. Results are from the last 1000 episodes in each test.

7 Discussion

In this chapter the process of designing, tuning and evaluating a RL algorithm will be discussed further than was deemed appropriate in the respective chapters.

The design of the algorithm was from the beginning a very complex task, with many degrees of freedom for design choices. It is worth noting that many different choices may have given even better results than what was obtained in this report, especially regarding the design of the state parameters. While the general information put into the state parameters is very valid, the way to encode this information could have been done in many other ways. Especially the context information comes to mind, where many different encodings of the orientation and location information are possible. Two encodings which would have been interesting to investigate would to encode orientation not relative to a fixed coordinate frame, but instead as relative to the previous orientation and to encode the location not with polar coordinates but rectangular coordinates instead. This could probably benefit from being the subject future work.

7.1 Assumptions during tuning and evaluation

During tuning of the designed algorithm, a lot of different assumptions were made in order to make such a task feasible. In hindsight it is worth looking at some of these assumptions again, and discuss their validity further. Firstly, it was assumed that it is possible to separate the tuning of state parameters and hyper-parameters, as the optimal values for one should not affect the other. This might not be entirely valid, especially in the case where two different state parameter combinations have very different size state space. When the state space is very large, a larger step size might be better as the same state might not be visited very often, however this is again assuming that the action-value targets stays more or less the same between consecutive visits to the same state.

Next it was assumed that the individual state parameters identified in Chapter 4 could be tuned separately, as codeword information and context information added independent information to the state. This assumption is probably still mostly valid, supported by the results from tuning the state parameters in the multi agent case. Here tests with and without orientation information at the receiver, combined with the tests with full communication, show that at least the result that adding information about the other agents actions are still unnecessary regardless of the context information added. However, it was also assumed that codeword information would be the most important parameter, which is why tuning always started with this parameter. In hindsight, it might have produced different results if instead the location information was tuned first, as this seemed to add the most in LOS conditions.

Finally it was assumed that the single agent and multi agent cases are so similar that the results from tuning the single agent would carry over to the multi agent case. This was partly done

to save time during tuning, as a complete sweep of especially the state space would have been very time and resource consuming. The similarity was mostly credited to the fact that both cases had the same environment, why the same parameters should work in each case. However, this fails to acknowledge that in the multi agent case, the environment, seen from each agent, now also consists of the other agent acting on its own. Thus it would have been more correct to have the broad sweeps again as in the single agent case, instead of the more narrow ones that were performed. This became visible during evaluation, where in the multi agent case, having a discounting factor of 0 proved to be much better than the tuned value of 0.6. In fact, the discounting factor proved to be a very important hyper-parameter, and it has been the cause of much debate during the project. As such this subject will be discussed later in this chapter.

Something else which might have had a big impact on the results, was the decision to make conclusions based on single simulations. When simulating an RL-agent, there are a lot of random aspects, which means that running a simulation twice with the same settings will produce small differences in the results. In the beginning it was chosen that due to very long simulation times, between 4 and 8 hours, resources would not be allocated to run each simulation multiple times and instead focus would be on testing a variety of different settings. While this makes it hard to draw any definite conclusions when the results are close, in many cases the differences between settings were larger than what could be expected from the variance of the random processes.

7.2 Flawed validation method

In Chapter 5 a series of tests for evaluation of hyper-parameters were conducted. As admitted in the summary of the same chapter, this method does not produce results which accurately depicts the behavior and learning of a new agent with the chosen hyper-parameters. Neither does the result provide insight into whether the parameters are overfitted or not. The performance of a new agent with the given hyper-parameters was unknown as during the training period the agent did not have the given hyper-parameters. Overfitting wasn't as the same parameters were not tested in two independent environments, as thus generalization wasn't tested.

As mentioned, first a training period with a set of initial hyper-parameters was simulated. This was followed by the agent continuing in a identically distributed but independently draw environment with the hyper-parameters changed to the designated test values. While the result does not work for their intended purpose, a useful test with the same methodology could be constructed. If this system were to be implemented the Q-tables are likely to be pre-trained to some extent. If any general unchanging knowledge is gained there's no reason for this cost to be occurred by each user if this could be embedded in the device at the factory. The hyper-parameters need not be constant, end even if this is the case if a set of more optimal parameters is found through further experimentation these could be downloaded at the device. The performance of such a circumstance could be tested by the flawed testing methodology. However as the training period wasn't examined no comments will be made around such circumstances. While the test methodology, for specific purposes, is useful the question of performance and overfitting of the hyper-parameters remain. For this purpose another test report was constructed, as described in Appendix I. This test was only conducted for the multiple agent case in the interest of time. For these test a new agent with a default Q-table values is used for the training period, then another agent is initialized and used for the validation period. Each agent also posses the same hyper-parameters. The tested hyper-parameters remain equal

across the flawed and this test, which are given in Table 7.1.

γ	0.6	0.7	0.8	
α	0.005	0.01	0.05	
$\epsilon constant$	0.001	0.005	0.01	
ϵ *decaying weight*	150	300	600	900
ϵ *adaptive weight*	0.3	1	10	15

Table 7.1: Value of hyper-parameters to be tested

First the training period will be shown, were conclusions about performance and convergence times can be drawn. Hereafter the validation results will be presented to show whether the training performance generalizes.

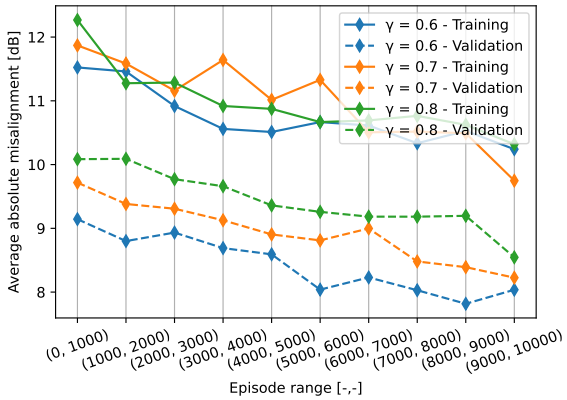


Figure 7.1: Average absolute misalignment of multiple agents with different discounting factors.

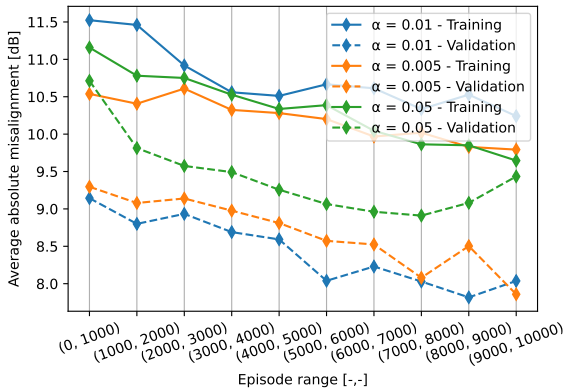


Figure 7.2: Average absolute misalignment of multiple agents with different step sizes.

When inspecting Fig. 7.1 $\gamma = 0.6$ still performs best. In Fig. 7.2 of the training results the best would be for $\alpha = 0.05$. This would have changed the settings for the rest of the simulations were they performed properly the first time. Nevertheless the impact of the step size still seems minimal.

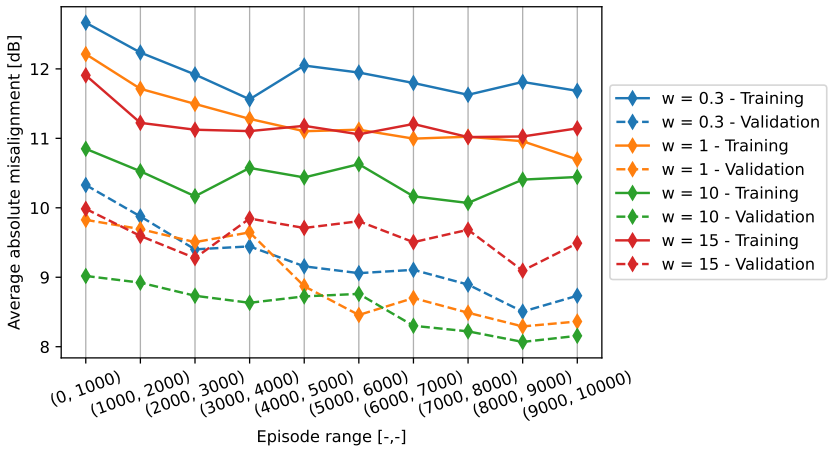


Figure 7.3: Average absolute misalignment of multiple agents with different exploration rates. Dashed lines are for validation, and training otherwise

From Fig. 7.3 the weight of $w = 10$ would still be concluded the best, although if this still would be the case with $\alpha = 0.05$ is unknown and this result is of limited value. While the optimal results likely wouldn't have changed drastically even if the tests were conducted properly the performance would likely be concluded to be worse across the board. An interesting observation is the fact that not only does the validation simulations perform as well as training, but better. The fact that validation obtains better performance so consistently would likely point to some difference between the data sets. Even if the validation set is less challenging it can still be concluded that none of the parameters overfit.

7.3 Discounting factor

The most distinctive result is probably that given in Fig. 6.5. Here the multi agent case was shown with different update rules. It was also shown that the simple update rule, which is equivalent to $\gamma = 0$, outperformed every other method, which was expanded upon in Section 6.6. The fact that simple performed the best suggests that future states and actions have very little impact on what actions should be chosen in any given state. One factor which could cause this in the multi agent case, but not single centralized, is the fact that action are no longer coordinated. When this is the case a large factor impacting the reward is an unknown. Trying to account for future states, and their values, without having knowledge about a large factor impacting that value might be less useful than not trying at all. The consequences of an agents actions are also different in the multi and single agent case. For the multi agent at the BS actions only have the impact of restricting the reachable codewords within as few as seven steps and even less for the UE. While this is still technically true for the single agent the number of possible routes from a pair of codewords to any other arbitrary pair is much larger, as the routes have to be considered together. A single centralized agent with adjacent action encoding is therefore not expected to navigate the codebook better than the multi agents. The discounting factors impact in general have been difficult to get an intuition about as it can be examined through different lenses. Depending on how this value is interpreted different conclusions can be drawn about the underlying cause of different behaviors observed. As mentioned in the paragraphs above it can be interpreted as how much the agent considers the future. It can also be interpreted as how much the agent trust already established values estimate from experiences ultimately from the past. Given the dynamic environment the latter carries more weight than in the idealised RL framework.

7.4 Future work

For further work, if not resource or time constrained, the test methodology should be corrected such that each test includes multiple simulations of agents with the same parameters. This would reduce the what variance is present within each test, and produce a more accurate picture based on the results. The agent itself have only been tested with one state encoding. To gain insight into how the encoding itself affects performance, tests where the same information is given to the agent in different formats should be conducted. As the state is a large factor in the agents performance any results in this area would be valuable for further development. The simulation scenario itself also require more development to reflect reality more accurately. In the simulations only a single BS and UE is considered. In reality each UE would face the problem of identifying the best BS to connect to, and communicate with. The results obtain in this report are only

indicative of performance if another system took on this task. Designating the RL algorithm as entirely responsible would mean a more, but realistic simulation, which also affords the RL algorithm the opportunity to find and exploit information or structure in the problem which isn't obvious.

Conclusion 8

In this report the challenges of communicating in the mmWave spectrum were investigated and analyzed. It was found that a major challenge in this field is the propagation behavior of the electromagnetic waves. Because of the inherent differences between mmWave and previous generations of the cellular network, mmWave signals generally produce lower signal power. This is because of several factors; greater free space propagation loss, lower penetration power and lower diffraction power, which all contribute to worse signal power and coverage. It was concluded that the high attenuation had to be alleviated, where beamforming was proposed. This was deemed suitable because of the small size of the antennas used in the mmWave spectrum. Beamforming however introduced a new problem, that of beams having to align along a path, preferably of low attenuation. The mechanism proposed for this was RL because of its adaptability. This lead to the following research question, which underlie the rest the work, which is

> Can reinforcement learning be implemented as a suitable solution for beam alignment and tracking in a mmWave communication system, consisting of both stationary and mobile units?

To begin answering this question, a detailed preliminary work was conducted with modelling the mmWave communication system to be able to realistically simulate the problems described in Chapter 1. This consisted of first modelling the wireless communication aspects of the problem, including modeling of the wireless channel, the actual signal passed through this channel and the design of a codebook of beamformers. Then a model for how the UE would realistically move around in a physical environment was implemented.

Following this, the subject of RL was researched thoroughly leading to an analysis of how the general framework of RL can fit in the context of the mmWave communication problem. Based on this analysis two RL algorithms, a centralized version with a single agent and a distributed version with two agents were implemented, both utilizing tabular methods. Following this their parameters were tuned through some broad sweeps of the parameter spaces. Finally, the tuned algorithms were evaluated with the intention of coming closer to answering the proposed research question. To determine the suitability of the developed algorithms, they were tested under various different conditions, such as; LOS and NLOS conditions and noisy environments. To act as a reference a simple heuristic algorithm was implemented as well and tested in the same scenarios.

On the basis of these tests it is concluded that RL could definitely be a suitable solution for beam alignment and tracking. While a single agent implementation might not be the best choice, the multi agent implementation showed promising results, with a simple algorithm outperforming the heuristic algorithm in both LOS and NLOS conditions.

Bibliography

[1] companiesmarketcap. 'Largest companies by market cap.' (2022), [Online]. Available: https://companiesmarketcap.com/ (visited on 03/08/2022).

[2] Cisco. 'Cisco visual networking index: Forecast and trends 2017-2022.' (2019), [Online]. Available: https://twiki.cern.ch/twiki/pub/HEPIX/TechwatchNetwork/HtwNetworkDocuments/white-paper-c11-741490.pdf (visited on 03/07/2022).

[3] T. W. Bank, *DATA FOR BETTER LIVES*, First. International Bank for Reconstruction and Development/The World Bank, 2021, pp. 227–265, ISBN: 978-1-4648-1607-9. DOI: 10.1596/978-1-4648-1607-9.

[4] D. E. Agency, '5g action plan for denmark,' Danish Energy Agency, Tech. Rep., 2019. [Online]. Available: https://ens.dk/en/our-responsibilities/telecom/5g-denmark.

[5] H. Friis, 'A note on a simple transmission formula,' *Proceedings of the IRE*, vol. 34, no. 5, pp. 254–256, 1946. DOI: 10.1109/JRPROC.1946.234568.

[6] T. S. Rappaport, S. Sun, R. Mayzus *et al.*, 'Millimeter wave mobile communications for 5g cellular: It will work!' *IEEE Access*, vol. 1, pp. 335–349, 2013. DOI: 10.1109/ACCESS.2013.2260813.

[7] K.-W. Kim, M.-D. Kim, J.-J. Park, J. Lee, J. Liang and K.-C. Lee, 'Diffraction loss model based on 28 ghz over-rooftop propagation measurements,' in *2017 IEEE 86th Vehicular Technology Conference (VTC-Fall)*, 2017, pp. 1–5. DOI: 10.1109/VTCFall.2017.8287881.

[8] M. Giordani, M. Mezzavilla and M. Zorzi, 'Initial access in 5g mmwave cellular networks,' *IEEE Communications Magazine*, vol. 54, no. 11, pp. 40–47, 2016. DOI: 10.1109/MCOM.2016.1600193CM.

[9] M. Li, C. Liu, S. V. Hanly, I. B. Collings and P. Whiting, 'Explore and eliminate: Optimized two-stage search for millimeter-wave beam alignment,' *IEEE Transactions on Wireless Communications*, vol. 18, no. 9, pp. 4379–4393, 2019. DOI: 10.1109/TWC.2019.2924368.

[10] S. Rezaie, C. Manchon and E. De Carvalho, 'Location- and orientation-aided millimeter wave beam selection using deep learning,' English, in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, ser. IEEE International Conference on Communications, 2020 IEEE International Conference on Communications, ICC 2020 ; Conference date: 07-06-2020 Through 11-06-2020, United States: IEEE, Jul. 2020, ISBN: 978-1-7281-5090-1. DOI: 10.1109/ICC40277.2020.9149272. [Online]. Available: https://icc2020.ieee-icc.org/.

[11] S. Rezaie, C. N. Manchon and E. de Carvalho, 'Location- and orientation-aided millimeter wave beam selection using deep learning,' in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6. DOI: 10.1109/ICC40277.2020.9149272.

[12] R. W. Heath, N. González-Prelcic, S. Rangan, W. Roh and A. M. Sayeed, 'An overview of signal processing techniques for millimeter wave mimo systems,' *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 3, pp. 436–453, 2016. DOI: 10.1109/JSTSP.2016.2523924.

[13] Z. Xiao, T. He, P. Xia and X.-G. Xia, 'Hierarchical codebook design for beamforming training in millimeter-wave communication,' *IEEE Transactions on Wireless Communications*, vol. 15, no. 5, pp. 3380–3392, 2016. DOI: 10.1109/TWC.2016.2520930.

[14] A. Alkhateeb, O. El Ayach, G. Leus and R. W. Heath, 'Channel estimation and hybrid precoding for millimeter wave cellular systems,' *IEEE Journal of Selected Topics in Signal Processing*, vol. 8, no. 5, pp. 831–846, 2014. DOI: 10.1109/JSTSP.2014.2334278.

[15] F. I. for Telecommunications. 'Quadriga webpage.' Downloadet: 04-02-2022. (2022), [Online]. Available: https://quadriga-channel-model.de/.

[16] 3GPP. 'Technical specification group radio access network; study on channel model for frequencies from 0.5 to 100 ghz (release 17).' Last visited: 31-05-2022. (2022), [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/38_series/38.901/38901-h00.zip.

[17] F. I. for Telecommunications. 'Quasi deterministic radio channel generator user manual and documentation.' Last visited: 09-02-2022. (2021), [Online]. Available: https://quadriga-channel-model.de/wp-content/uploads/2021/07/quadriga_documentation_v2.6.1-0.pdf.

[18] C. Bettstetter, 'Smooth is better than sharp: A random mobility model for simulation of wireless networks,' in *Proceedings of the 4th ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, ser. MSWIM '01, Rome, Italy: Association for Computing Machinery, 2001, pp. 19–27, ISBN: 1581133782. DOI: 10.1145/381591.381600. [Online]. Available: https://doi.org/10.1145/381591.381600.

[19] K. Teknomo, 'Microscopic pedestrian flow characteristics: Development of an image processing data collection and simulation model,' Ph.D. dissertation, Mar. 2002.

[20] Transportministeriet. 'Bekendtgorelse af faerdselsloven.' Last visited: 07-02-2022. (2021), [Online]. Available: https://www.retsinformation.dk/eli/lta/2021/1710.

[21] A. Ali, J. Mo, B. L. Ng, V. Va and J. C. Zhang, 'Orientation-assisted beam management for beyond 5g systems,' *IEEE Access*, vol. 9, pp. 51 832–51 846, 2021. DOI: 10.1109/ACCESS.2021.3070275.

[22] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html.

[23] C. Nwankpa, W. Ijomah, A. Gachagan and S. Marshall, 'Activation functions: Comparison of trends in practice and research for deep learning,' Dec. 2020.

[24] H. M. Schwartz and H. M. Schwartz, *Multi-agent machine learning: a reinforcement approach.* WILEY, 2014, ISBN: 9781118362082.

[25] M. Tokic, 'Adaptive ϵ-greedy exploration in reinforcement learning based on value differences,' 2010.

[26] A. dos Santos Mignon and R. L. de Azevedo da Rocha, 'An adaptive implementation of ϵ-greedy in reinforcement learning,' *Procedia Computer Science*, vol. 109, pp. 1146–1151, 2017. DOI: https://doi.org/10.1016/j.procs.2017.05.431. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050917311134.

[27] 3GPP. 'User equipment (ue) radio transmission and reception; part 2: Range 2 standalone.' Last visited: 31-05-2022. (2022), [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/38_series/38.101-2/38101-2-h50.zip.

Analysis of state parameters for a single agent with adjacent actions A

A.1 Purpose

In this test report multiple states, for a single centralized agent with adjacent action scheme, will be tested to evaluate performance. The states will be tested by simulating centralized agents and recording performance metrics. Evaluation and comparison will be based on the recorded metrics, and will focus on general performance. These tests are conducted to gain insight into what constitutes a state with good performance. In order to compare the states and find a good one, the performance metric is defined in the next section.

A.2 Test metric(s)

The performance metrics used in this report are all based on the misalignment between the received signal power and the highest achievable signal power at each time step. This metric is used instead of the received signal power alone, as this value is expected to vary significantly as the agent moves around the environment, regardless of the choices made. This is due to phenomena like free space loss and absorption, and thus a low received signal power might not necessarily correspond to bad performance of the agent.

Thus the misalignment is used instead, which is calculated as the difference between the received signal power and maximum achievable signal power in dB scale. The maximum signal power is defined as the maximum found when looking at all possible combinations of codewords in the entire codebook and not just those available to the agent at the current time step.

$$M = S_m - S_{max}$$

where

M :	$[dB]$	Misalignment	$RM \in \mathcal{R}$
S_m :	$[dB]$	Received signal power	$S_m \in \mathcal{R}$
S_{max} :	$[dB]$	Maximum achievable signal power	$S_{max} \in \mathcal{R}$

Both the received and maximum power used for calculating the misalignment are without added noise, to better allow the metric to measure the quality of the choices made. This is based on the notion that choices should be made based on the quality of the channel, which is assumed

to be independent of the thermal noise in the equipment. It should be noted that this metric is only obtainable because the test is simulated, as normally the maximum achievable signal power is not directly available.

The metrics used to compare the different agent configurations are:

- Average misalignment
- Median misalignment
- 1^{st} quartile misalignment
- 3^{rd} quartile misalignment

A.3 Test procedure

The test setup is generally the same for tuning of state parameters and hyper-parameters. The general setup is as follows:

Table A.1: Technical parameters used when tuning agent parameters.

Parameter	Value	Description
B	400 MHz	Total bandwidth
f_c	28 GHz	mmWave carrier frequency
P_{TX}	40 dBm	Transmission power
r_c	200 m	Cell radius
N_r	8	Number of receive antennas
N_t	16	Number of transmit antennas
M	10000	Number of episodes pr. simulation
N	7000	Number of steps pr. episode

A simulation consists of several episodes, with each episode consisting of one or more agents following a path, taking an action at each step. A track is chosen at random from a collection of 16 tracks at the beginning of each episode. Each track in the pedestrian collection is 20000 steps long, however the path used in each episode is only 7000 steps long. Thus a random starting point on the track is chosen as well at the beginning of each episode. By keeping the episodes shorter than the length of the tracks, it is possible to get more unique paths for the agent to follow in different episodes, along with a lot of paths with differing amounts of overlap. This means that it is not just the same 16 paths which are traversed over and over again in the simulation. 7000 steps was chosen as a fitting trade-off between path length and number of unique paths in total. The agent keeps the Q-table values between episodes, and thus learns across all episodes.

In order to ensure that the test metrics are statistically accurate, and to give the agent a chance to explore the entire state space, many episodes are simulated. Both the number of episodes and number of steps in each episode, are derived based on an intermediately sized state space. The total number of steps in the simulation should be enough for an agent to explore all state-action pair, to give accurate value estimates. The state given below, based on a single agent implementation, is assessed to be an intermediate state, and was used to derive the total number of steps.

- A history of the 2 previous beams at the UE

- A history of the 2 previous beams at the BS
- A distance between the UE and BS with a resolution of 4
- An angle between the UE and the BS with a resolution of 8

The size of the combined state-action space of this state was determined in Section 4.4.2 to be $6.05 \cdot 10^6$ state-action pairs. In order to allow proper exploration, the total number of steps for a simulation is set such that each state-action pair can be visited approximately ten times, resulting in a total step number of $70 \cdot 10^6$ steps. This gives a total of 10000 episodes in a simulation. While this allows an agents performance to converge, an even better performance measure would be an average over a statistically significant amount of agents. However, as each simulation takes approximately four hours to run, this is sadly not viable because of time and resource constraints. Thus the results presented in this chapter will be judged with this uncertainty in mind.

A.3.1 Variables

The algorithm consists of a number of variables, where every variable except the state will be kept constant throughout this test to ensure valid comparisons. The constant variables are the following.

γ :	$[-]$	Forgetting factor	$\gamma = 0.7$
α :	$[-]$	Action value adjustment amplitude	$\alpha = 0.05$
ϵ :	$[-]$	Exploration rate	$\epsilon = 0.05$
A :	$[-]$	Action space	$A = adjacent$

The action space A is the adjacent action space as described in Section 4.1.2. ϵ is chosen to be 5% or 1 out of 20 steps is an exploration step. The optimal value of this is highly dependant on the problem, and if the environment is dynamic, how quickly it changes. How often states are visited also influences the choice, if a certain amount of exploration in each state is wanted. In the interest of time this value is based on preliminary test, where 0.05 lead to promising results. α is chosen to be 0.05 which, according to [22, p. 222], means the the action value estimate converges in approximately 20 steps, assuming the target is unchanged. As an initial value this is deemed an appropriate value. For the discounting factor no good heuristic could be found, and a value of 0.7 was chosen at random.

A.3.2 States to be tested

It is infeasible to test every and all state combinations and therefore a subset will be identified. Certain assumptions are made on the problem to simplify, and reduce the search over the possible states. It is assumed the performance impact of adjusting the action/beam history is independent of the impact of adjusting the rest of the state. Therefore the first two parameters of the state can be investigated independently from the rest of the state. When an optimal value for these two parameters have been found, test will be conducted on states consisting of the optimal beam/action history, and a combination of UE orientation, or location and both. Independence of the impact on performance by including the UEs orientation and relative location cannot be guaranteed. However in the interest of time these parts of the state are going to be examined independently. This means that a total five types of states are going to be tested. The first, with only beam/action history. The second with beam/action history and UE orientation.

The third with beam/action history and the distance between the two terminals. The fourth with beam/action history and UE relative position. The fifth with beam/action history, UE orientation and relative location. The action/beam history is kept as part of the state through all types of states, as without it the adjacent action encoding in itself is not enough to accurately know which way is being beamformed. It it therefore being judged important enough to be included in all states being tested. In general the range of values are chosen by finding the lowest sensible value. Then subsequent values are powers of two times the starting value to ensure a difference in performance can be observed.

The first parameters to be adjusted are the length of the UE and BS action/beam history. To simplify further the lengths of both terminals beam/action histories are kept equal, and adjusted simultaneously. As such the first group of tests will be performed on the states presented in Table A.2.

	Nb_t	Nb_r	N_o	R_o	R_d	R_a
State 1	1	1	0	0	0	0
State 2	2	2	0	0	0	0
State 3	3	3	0	0	0	0
State 4	4	4	0	0	0	0

Table A.2: States for testing optimal action/beam history length

Where Nb_t is the codeword history of the UE, Nb_r the codeword history of the BS, N_o the orientation history, R_o the orientation resolution, R_d the distance resolution and R_a the angle resolution. These values, and the encoding, are also explained in Section 4.4.2. The test of which is found best will form a basis for the four other types of states to be tested, because they all contain action/beam histories.

The second type of state, one with orientation is going to be explored, with four different history lengths, and four different resolutions. The first four of which can be seen in Table A.3

	Nb_t	Nb_r	N_o	R_o	R_d	R_a
State 1	N_s	N_s	1	4	0	0
State 2	N_s	N_s	1	8	0	0
State 3	N_s	N_s	1	16	0	0
State 4	N_s	N_s	1	32	0	0

Table A.3: States for testing optimal orientation settings, where N_s is the optimal result from Table A.2

The rest of the state combinations differ by replacing $N_o = 1$ with $N_o = 1, 2$ & 3. Here N_s is the optimal value found from the tests of the previous type of state. The next type of state includes distance between the terminals, which is going to be tested with the values given in Table A.4.

	Nb_t	Nb_r	N_o	R_o	R_d	R_a
State 1	N_s	N_s	0	0	2	0
State 2	N_s	N_s	0	0	4	0
State 3	N_s	N_s	0	0	8	0
State 4	N_s	N_s	0	0	16	0
State 5	N_s	N_s	0	0	32	0

Table A.4: States for testing optimal distance resolution, where N_s is the optimal results from Table A.2

The fourth type of state to be analyzed contains the location of the UE with respect to the BS. The first four of this type are given in Table A.5.

	Nb_t	Nb_r	N_o	R_o	R_d	R_a
State 1	N_s	N_s	0	0	2	4
State 2	N_s	N_s	0	0	2	8
State 3	N_s	N_s	0	0	2	16
State 4	N_s	N_s	0	0	2	32

Table A.5: States for testing optimal location settings, where N_o is the optimal result from Table A.2

The location is going to be explored with four different angle resolutions and five different distance resolutions. The distance resolutions are 2, 4, 8, 16 and 32, given as R_d in the table.

The fifth type of state contains action/beam history, UE orientation and relative location. Instead of testing a multitude of different combinations for this type of state, the values found best in the previous test are going to be used here. That is the best action/beam history length from tests of states type one. The best UE orientation resolution and history length from tests of states of type two. The best relative location resolution from tests of states of type four.

A.4 Data processing

As mentioned in Appendix A.3 each test consists of 70.000.000 steps, however plotting the relative misalignment for all steps would not result in a graph indicative of the general performance of the state. To remove the variation within each episode, the misalignment is calculated based on averages. First the linear misalignment is calculated, and transformed to logarithmic scale.

$$MA = 10 \cdot log_{10}(R) - 10 \cdot log_{10}(R_{max})$$

This misalignment is then averaged across all steps within each episodes, resulting in a reduction from 70.000.000 data points to 10.000.

$$RM_{pe} = \frac{1}{7000} \cdot \sum_{i=1}^{7000} RM_i$$

where

RM_m	[−]	Average reward for a given episode
RM_i	[−]	Reward for a the i'th step in a given episode

In order to remove the influence of a specific track/path on the performance multiple episodes are averaged to get the final performance measure. The 10.000 episodes are averaged in ten intervals of 1000 episodes each.

A.5 Results/Conclusion

In the following section the results of the simulations of agents with the states given in Appendix A.3.2 are presented. The legend on every figure describes the state an agent had when generating the data. This state is encoded in the same way, and order as Table 4.1.

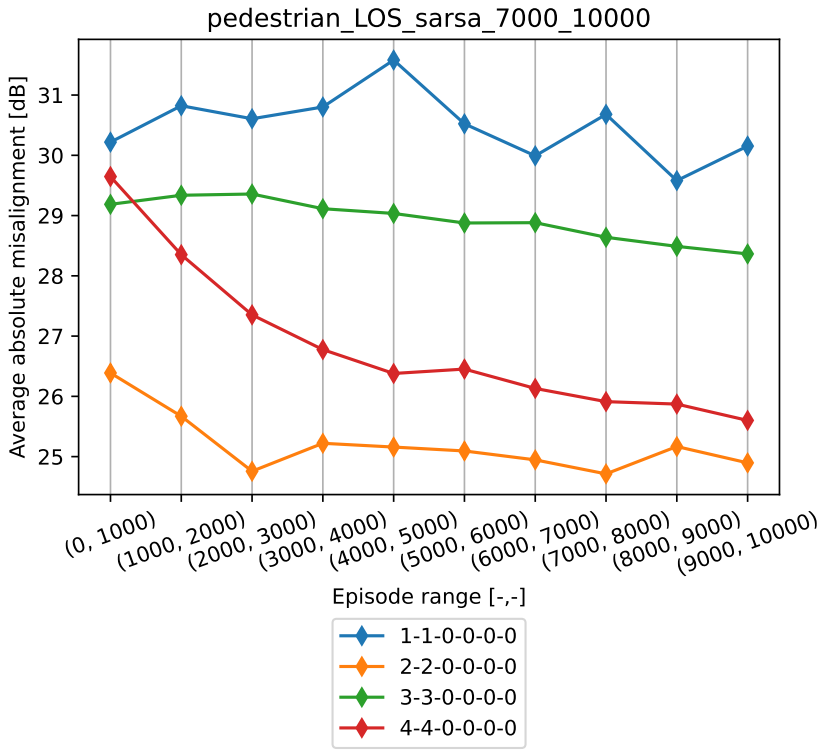


Figure A.1: Performance of states, from Table A.2, only containing action/beam history.
Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a

On Fig. A.1 the performance of several states can be seen. The best state is with a length of two previous beams for both terminals, not only in obtained performance, but also convergence time.

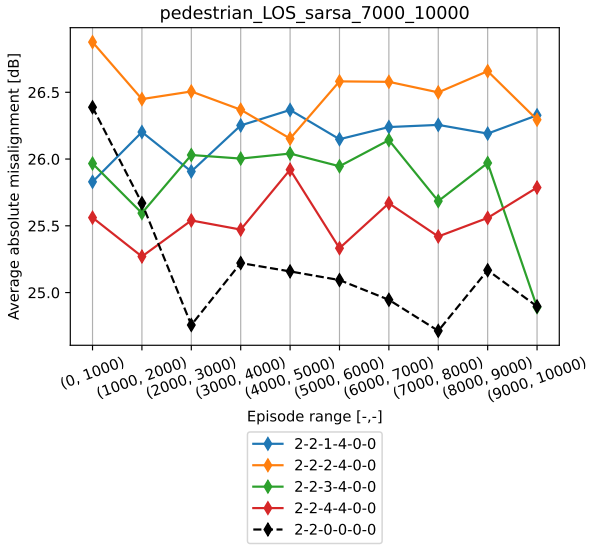


Figure A.2: Performance of states, from Table A.3, with orientation resolution of four

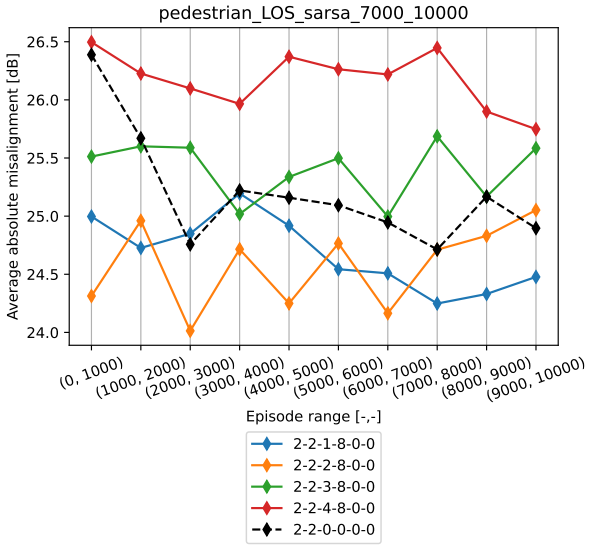


Figure A.3: Performance of states, from Table A.3, with orientation resolution of eight

Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a

Fig. A.2 through Fig. A.5 plots of states of the second type from Table A.3 can be seen. Note that here the state "2-2-0-0-0-0" from Fig. A.1 is also plotted to provide a point of comparison. While more information gives the opportunity to distinguish more situations from each other it also means more states, that needs values assigned. That means, if the information is not valuable with regards to choosing good actions it becomes a hindrance, as likely demonstrated on Fig. A.2.

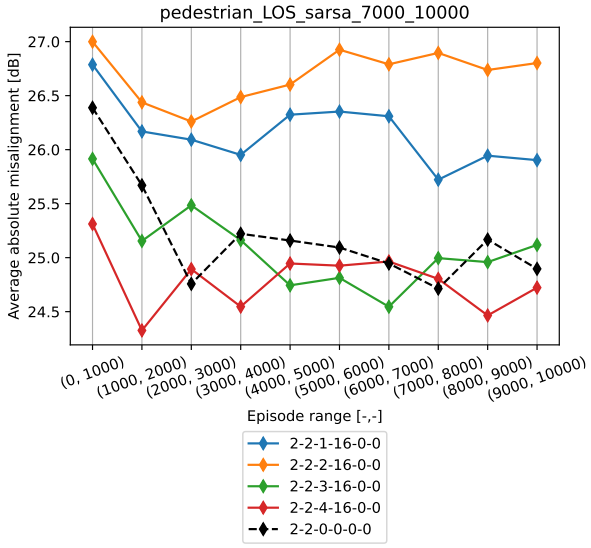


Figure A.4: Performance of states, from Table A.3, with orientation resolution of 16

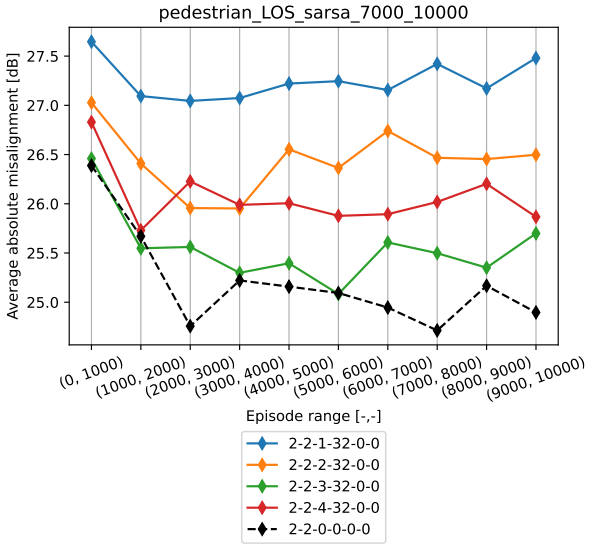


Figure A.5: Performance of states, from Table A.3, with orientation resolution of 32

Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a

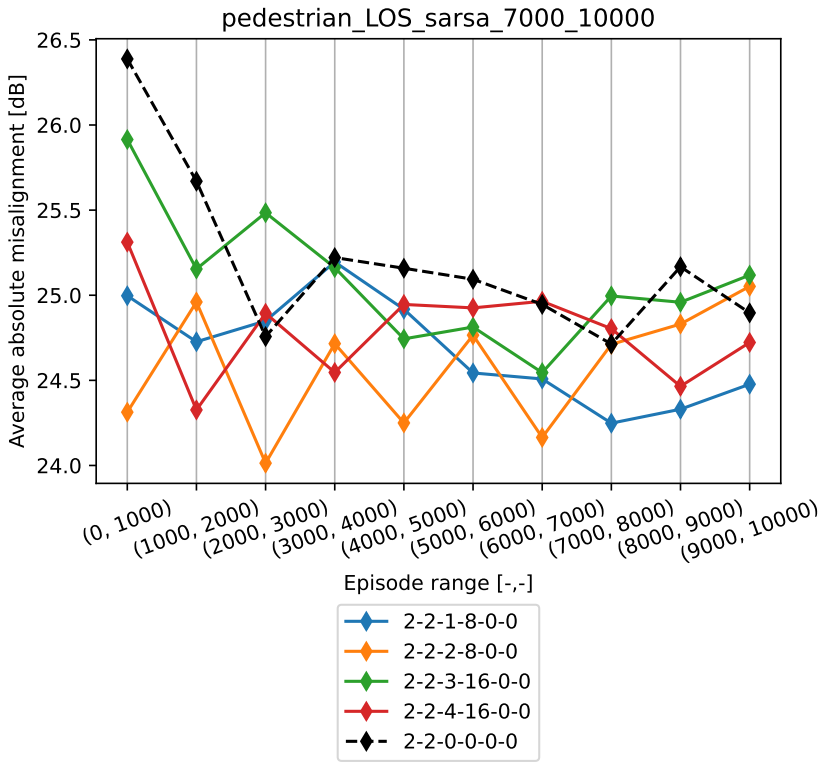


Figure A.6: Performance of the best states from Fig. A.2 to Fig. A.5
Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a

Figure A.6 plots the best performing states from the previous orientation states. It can be seen that the orientation only provide a marginal benefit, if at all, even among the best states. From the figure it can be seen that the best orientation provides around a $\frac{1}{2}$ dB increase in performance.

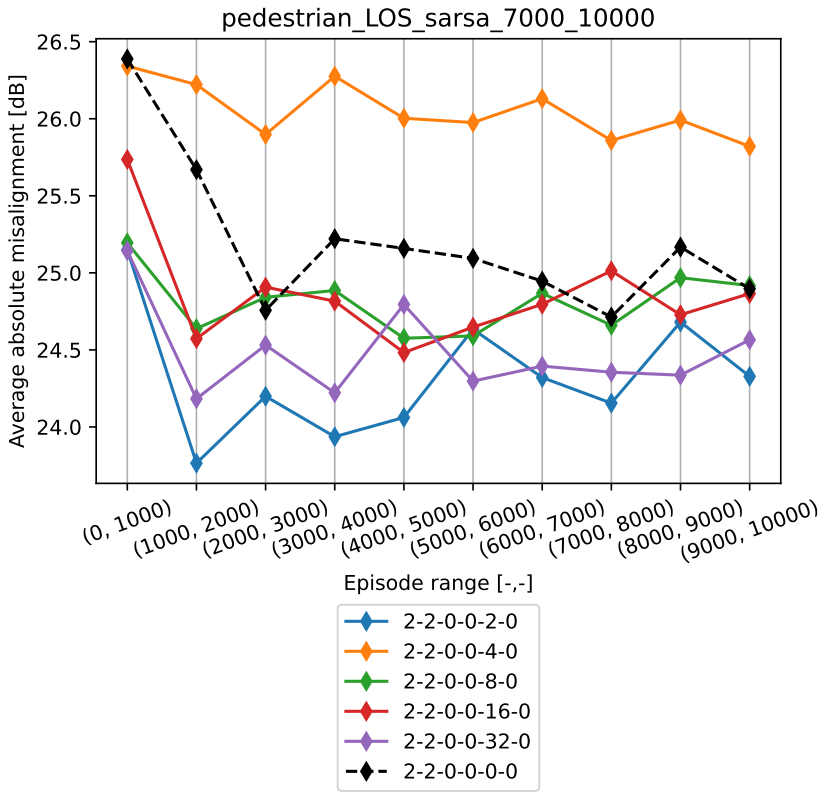


Figure A.7: Performance of states, from Table A.4, with different distance resolutions
Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a

Fig. A.7 shows the states containing the distance between the UE and BS. It can be seen that in general the distance does improve the performance, but only marginally. That is with the exception of the state "2-2-0-0-4-0" which is strictly worse and does not follow the pattern of the rest of the states. Why this is the case is unknown.

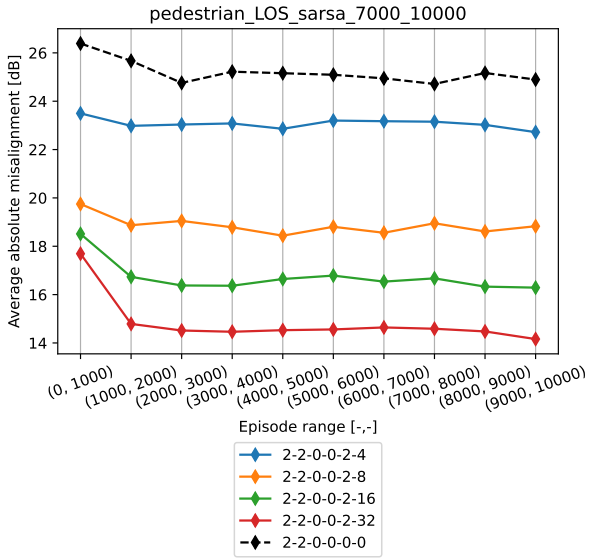


Figure A.8: Performance of states, from Table A.5, with distance resolution of two

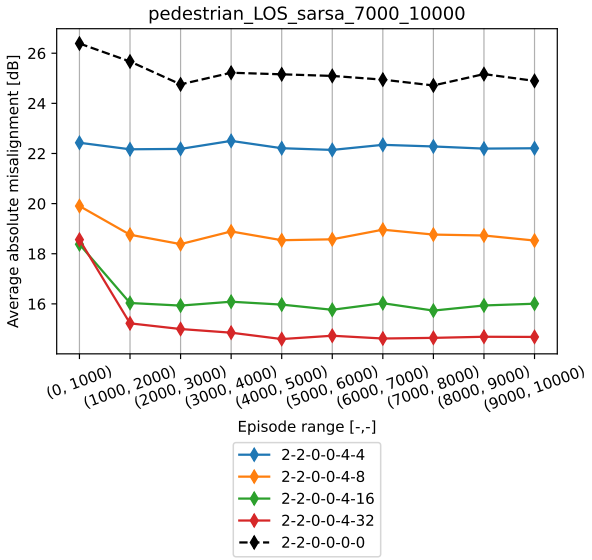


Figure A.9: Performance of states, from Table A.5, with distance resolution of four

Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a

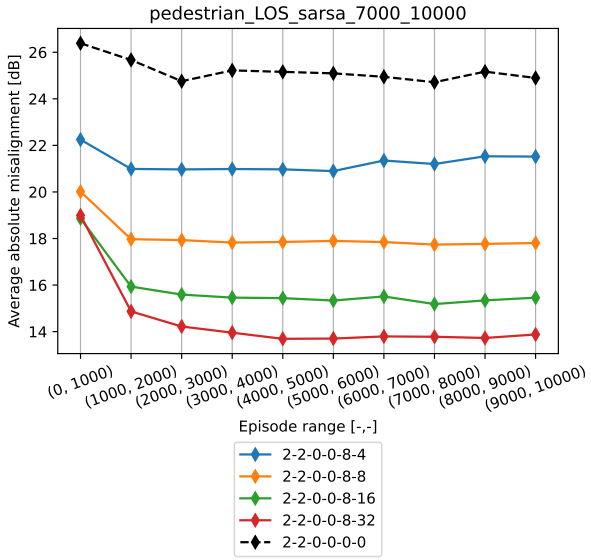


Figure A.10: Performance of states, from Table A.5, with distance resolution of eight

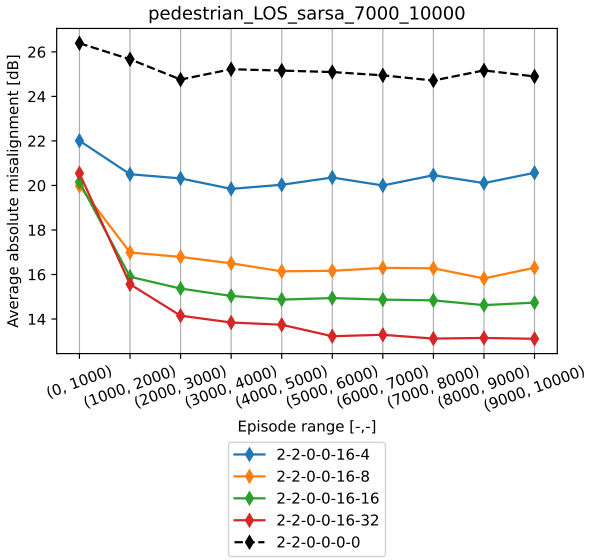


Figure A.11: Performance of states, from Table A.5, with distance resolution of 16

Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a

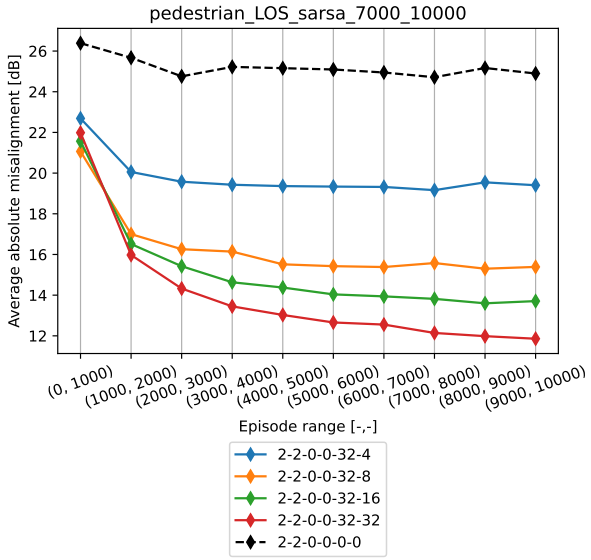


Figure A.12: Performance of states, from Table A.5, with distance resolution of 32

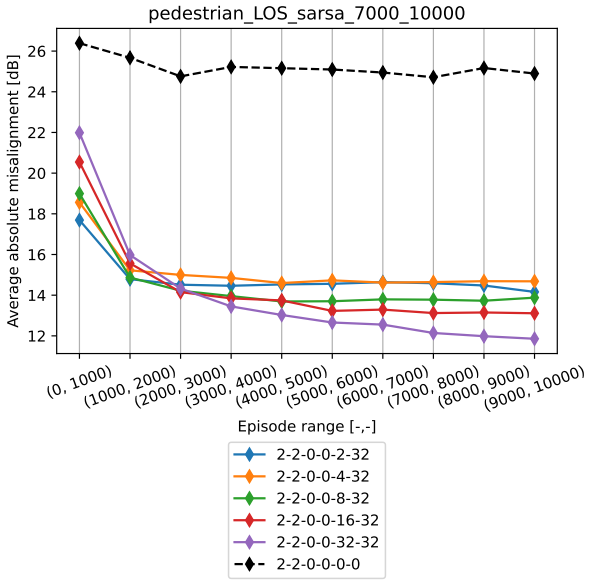


Figure A.13: Performance of the best states from Fig. A.8 to Fig. A.12

Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a

Figure A.8 through Fig. A.12 plots states of the fourth type with different angle and distance resolutions. It can be seen that the location provides a significant performance increase, and this increase is monotonically increasing with respect to the angle resolution. Fig. A.13 Shows the best states from each graph. The best fourth type of state provide a performance improvement of 10 to 13 dB.

While UE orientation did not provide a significant performance increase by itself, this and the relative location may not be independent, and therefore their combined performance cannot be

predicted from their individual performance. The fifth type of state which contains both have been plotted in Fig. A.14

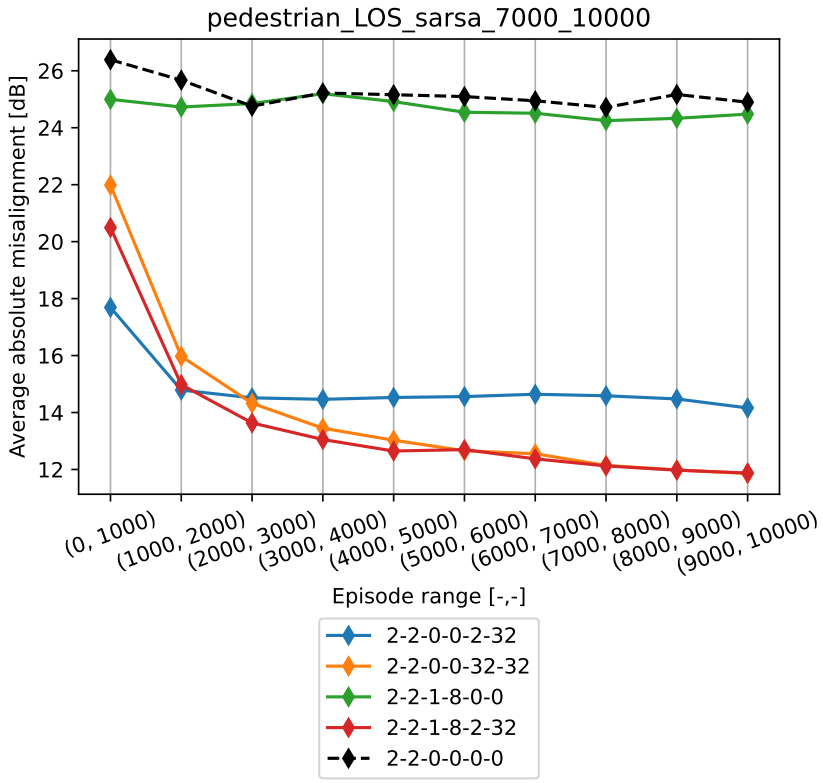


Figure A.14: Performance of a combined state
Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a

The state "2-2-1-8-2-32" is plotted along with the two states, "2-2-1-8-0-0" and "2-2-0-0-2-32", which it is a combination of. A state comparable to it in performance is also depicted. It can be seen that the combined state performs better than its parts. The performance gained from adding orientation information to the initial state, and to the location state is different. It can be seen that the orientation is more valuable if the location is known. The states "2-2-0-0-32-32" and "2-2-1-8-2-32" are comparable in performance, however they are not equal. When comparing the size of the action value space of the two, it becomes clear that the size of the latter is half that of the former, which would result in less memory usage.

This leads to the conclusion that the optimal state for a single centralized agent, is one with a state defined by the parameters given in the table below

	Nb_r	Nb_t	N_o	R_o	R_d	R_a
Optimal parameter	2	2	1	8	2	32

Table A.6: Optimal state parameters, for a single centralized agent

Analysis of state parameters for multiple agents with adjacent actions

B

B.1 Purpose

In this test report multiple states, for multiple agents with adjacent action scheme, will be tested to evaluate performance. The states will be tested by simulating multiple agents and recording performance metrics. Evaluation and comparison will be based on the recorded metrics, and will focus on general performance. These test are conducted to gain insight into what constitutes a state with good performance. In order to compare the states and find a good one, the performance metric is defined in the next section.

B.2 Test metric(s)

The performance metrics used in this report are all based on the misalignment between the received signal power and the highest achievable signal power at each time step. This metric is used instead of the received signal power alone, as this value is expected to vary significantly as the agent moves around the environment, regardless of the choices made. This is due to phenomena like free space loss and absorption, and thus a low received signal power might not necessarily correspond to bad performance of the agent.

Thus the misalignment is used instead, which is calculated as the difference between the received signal power and maximum achievable signal power in dB scale. The maximum signal power is defined as the maximum found when looking at all possible combinations of codewords in the entire codebook and not just those available to the agent at the current time step.

$$M = S_m - S_{max}$$

where

M :	$[dB]$	Misalignment	$RM \in \mathcal{R}$
S_m :	$[dB]$	Received signal power	$S_m \in \mathcal{R}$
S_{max} :	$[dB]$	Maximum achievable signal power	$S_{max} \in \mathcal{R}$

Both the received and maximum power used for calculating the misalignment are without added noise, to better allow the metric to measure the quality of the choices made. This is based on the notion that choices should be made based on the quality of the channel, which is assumed

to be independent of the thermal noise in the equipment. It should be noted that this metric is only obtainable because the test is simulated, as normally the maximum achievable signal power is not directly available.

The metrics used to compare the different agent configurations are:

- Average misalignment
- Median misalignment
- 1^{st} quartile misalignment
- 3^{rd} quartile misalignment

B.3 Test procedure

The test setup is generally the same for tuning of state parameters and hyper-parameters. The general setup is as follows:

Table B.1: Technical parameters used when tuning agent parameters.

Parameter	Value	Description
B	400 MHz	Total bandwidth
f_c	28 GHz	mmWave carrier frequency
P_{TX}	40 dBm	Transmission power
r_c	200 m	Cell radius
N_r	8	Number of receive antennas
N_t	16	Number of transmit antennas
M	10000	Number of episodes pr. simulation
N	7000	Number of steps pr. episode

A simulation consists of several episodes, with each episode consisting of one or more agents following a path, taking an action at each step. A track is chosen at random from a collection of 16 tracks at the beginning of each episode. Each track in the pedestrian collection is 20000 steps long, however the path used in each episode is only 7000 steps long. Thus a random starting point on the track is chosen as well at the beginning of each episode. By keeping the episodes shorter than the length of the tracks, it is possible to get more unique paths for the agent to follow in different episodes, along with a lot of paths with differing amounts of overlap. This means that it is not just the same 16 paths which are traversed over and over again in the simulation. 7000 steps was chosen as a fitting trade-off between path length and number of unique paths in total. The agent keeps the Q-table values between episodes, and thus learns across all episodes.

In order to ensure that the test metrics are statistically accurate, and to give the agent a chance to explore the entire state space, many episodes are simulated. Both the number of episodes and number of steps in each episode, are derived based on an intermediately sized state space. The total number of steps in the simulation should be enough for an agent to explore all state-action pair, to give accurate value estimates. The state given below, based on a single agent implementation, is assessed to be an intermediate state, and was used to derive the total number of steps.

- A history of the 2 previous beams at the UE

- A history of the 2 previous beams at the BS
- A distance between the UE and BS with a resolution of 4
- An angle between the UE and the BS with a resolution of 8

The size of the combined state-action space of this state was determined in Section 4.4.2 to be $6.05 \cdot 10^6$ state-action pairs. In order to allow proper exploration, the total number of steps for a simulation is set such that each state-action pair can be visited approximately ten times, resulting in a total step number of $70 \cdot 10^6$ steps. This gives a total of 10000 episodes in a simulation. While this allows an agents performance to converge, an even better performance measure would be an average over a statistically significant amount of agents. However, as each simulation takes approximately four hours to run, this is sadly not viable because of time and resource constraints. Thus the results presented in this chapter will be judged with this uncertainty in mind.

B.3.1 Variables

The algorithm consists of a number of variables, where every variable except the state will be kept constant throughout this test to ensure valid comparisons. The constant variables are the following.

γ :	$[-]$	Forgetting factor	$\gamma = 0.7$
α :	$[-]$	Action value adjustment amplitude	$\alpha = 0.05$
ϵ :	$[-]$	Exploration rate	$\epsilon = 0.05$
A :	$[-]$	Action space	$A = adjacent$

The action space A is the adjacent action space as described in Section 4.1.2. ϵ is chosen to be 5% or 1 out of 20 steps is an exploration step. The optimal value of this is highly dependant on the problem, and if the environment is dynamic, how quickly it changes. How often states are visited also influences the choice, if a certain amount of exploration in each state is wanted. In the interest of time this value is based on preliminary test, where 0.05 lead to promising results. α is chosen to be 0.05 which, according to [22, p. 222], means the the action value estimate converges in approximately 20 steps, assuming the target is unchanged. As an initial value this is deemed an appropriate value. For the discounting factor no good heuristic could be found, and a value of 0.7 was chosen at random.

B.3.2 States to be tested

It is infeasible to test every and all state combinations and therefore a subset will be identified. The problem of the search space is even worse in the multi agent case than in the single agent case from Appendix A, as two states are present in any test. In the interest of time a partial assumption is made. The assumption is that the optimal values of the context information, ie. the UE orientation and relative location, is equal for the multi and single agent case, as the physical environment is the same. This assumption does not cover whether the information is useful, only that if it is, the optimal value for the parameters are the same. Furthermore two different cases will be tested, one with communication between the two terminals, and one without. In the case of communication each terminal can only know their own beam/action history and only the UE knows its orientation. The relative position is assumed obtainable without further communication overhead and can therefore be included in both cases.

At first a set of states consisting of only the codeword history for each agents own actions will be tested. After this the codeword history of the other terminal will be introduced to allow for differing amount of information about the two agents. Then context information will then be added to the state, with values that gave the best results in the tests with a single agent, as mentioned earlier. Additionally, some information might now not be useful for both agents, and thus different combinations of state parameters might be needed for the receiver agent and the transmitter agent respectively.

The first type of state to be investigated are with the length of the UE and BS action/beam history. In Table B.2 the states for the BS for can be seen. The UE has a set of corresponding states where the values of Nb_t and Nb_r are interchanged.

	Nb_t	Nb_r	N_o	R_o	R_d	R_a
State 1	0	1	0	0	0	0
State 2	0	2	0	0	0	0
State 3	0	3	0	0	0	0
State 4	0	4	0	0	0	0

Table B.2: States for testing optimal action/beam history length for transmitter

Where Nb_t is the codeword history of the UE, Nb_r the codeword history of the BS, N_o the orientation history, R_o the orientation resolution, R_d the distance resolution and R_a the angle resolution. These values, and the encoding, are also explained in Section 4.4.2. In continuation of the case with no communication the second type of states is where only the UE knows it's orientation. These states are given in Table B.3 and Table B.4. Nb_o is the optimal value found from the test for the states in Table B.2.

	Nb_t	Nb_r	N_o	R_o	R_d	R_a
State 1	0	Nb_o	0	0	2	32
State 2	0	Nb_o	1	8	2	32

Table B.3: States for testing optimal action/beam history length for UE

	Nb_t	Nb_r	N_o	R_o	R_d	R_a
State 1	Nb_o	0	0	0	2	32
State 2	Nb_o	0	0	0	2	32

Table B.4: States for testing optimal action/beam history length for transmitter

To further develop the case with communication another set of states regarding the action/beam history are to be tested, which are given below: The third type of state, where communication is allowed, an optimal amount of information about the other terminals beam/action history is investigated with the states below. These states are for the BS, where the corresponding states for the UE are where the values of Nb_t and Nb_r are interchanged.

	Nb_t	Nb_r	N_o	R_o	R_d	R_a
State 1	1	Nb_o	0	0	0	0
State 2	2	Nb_o	0	0	0	0
State 3	3	Nb_o	0	0	0	0
State 4	4	Nb_o	0	0	0	0

Table B.5: States for testing optimal action/beam history length for transmitter

The fourth types of state is a further development of the case with communication, where both terminals know location of the UE, and it's relative location. Nb_{oo} is the optimal value of action/beam history length found in the test from Table B.5. The states in Table B.6 belongs to the BS, and Table B.7 belongs to the UE.

	Nb_t	Nb_r	N_o	R_o	R_d	R_a
State 1	Nb_{oo}	Nb_o	0	0	2	32
State 2	Nb_{oo}	Nb_o	0	0	2	32
State 3	Nb_{oo}	Nb_o	1	8	2	32

Table B.6: States for testing optimal action/beam history length for transmitter

	Nb_t	Nb_r	N_o	R_o	R_d	R_a
State 1	Nb_o	Nb_{oo}	0	0	2	32
State 2	Nb_o	Nb_{oo}	1	8	2	32
State 3	Nb_o	Nb_{oo}	1	8	2	32

Table B.7: States for testing optimal action/beam history length for UE

B.4 Data processing

As mentioned in Appendix B.3 each test consists of 70.000.000 steps, however plotting the relative misalignment for all steps would not result in a graph indicative of the general performance of the state. To remove the variation within each episode, the misalignment is calculated based on averages. First the linear misalignment is calculated, and transformed to logarithmic scale.

$$MA = 10 \cdot log_{10}(R) - 10 \cdot log_{10}(R_{max})$$

This misalignment is then averaged across all steps within each episodes, resulting in a reduction from 70.000.000 data points to 10.000.

$$RM_{pe} = \frac{1}{7000} \cdot \sum_{i=1}^{7000} RM_i$$

where

RM_m	[−]	Average reward for a given episode
RM_i	[−]	Reward for a the i'th step in a given episode

In order to remove the influence of a specific track/path on the performance multiple episodes are averaged to get the final performance measure. The 10.000 episodes are averaged in ten intervals of 1000 episodes each.

B.5 Results/Conclusion

The legend on every figure describes the states the agents had when generating the data. This state is encoded in the same way, and order as Table 4.1, where the first states is for the UE, and the second state is for the BS.

Tuning of codeword information w.o. communication between agents

With no knowledge of each others actions, the amount of information an agent has about codewords is limited to its own actions. A sweep of different amounts of codeword history was run, with the history ranging from only the current codeword, to a history of four codewords. As the state space is smaller than in the single agent case, a bigger range is chosen. The results from the test is shown in Table B.8 and Fig. B.1.

Table B.8: Table of different misalignment measurements of multiple distributed agents with different length codeword history. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
History of 1 codeword each	32.2 dB	32.2 dB	24.7 dB	39.5 dB
History of 2 codewords each	21.3 dB	20.8 dB	10.7 dB	30.4 dB
History of 3 codewords each	23.7 dB	23.6 dB	14.1 dB	32.7 dB
History of 4 codewords each	25.6 dB	25.7 dB	17.2 dB	33.9 dB

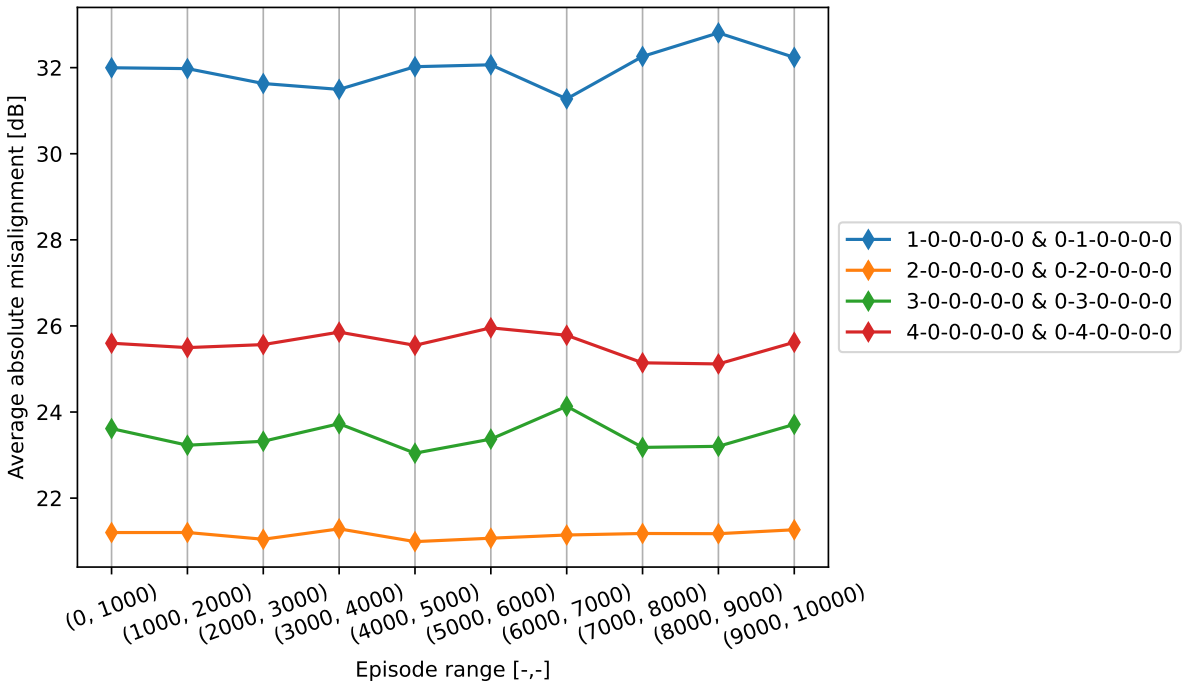


Figure B.1: Average absolute misalignment from tests with multiple distributed agents having different length codeword history.
Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a for UE & BS

As in the centralized implementation, it proves useful to have some knowledge of past actions, however keeping track of too many probably expands the state space with not enough information to compensate. Precisely as for the single agent, a history of two is optimal.

Tuning of codeword information w. communication between agents

To investigate whether information about the other agents actions is useful, a follow-up test is performed, where each agent gets a history of the other agents actions. It is assumed that the best amount of history of the other agents actions is independent from the history of ones own. Thus each agent has a history of two codewords based on the previous test. The results from the test is shown in Table B.9 and Fig. B.2.

Table B.9: Table of different misalignment measurements of multiple distributed agents with different length codeword history of the other agents actions. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
History of 1 codeword from the other	27.6 dB	27.6 dB	20.0 dB	35.3 dB
History of 2 codewords from the other	26.2 dB	26.3 dB	18.1 dB	34.4 dB
History of 3 codewords from the other	26.6 dB	26.6 dB	18.9 dB	34.4 dB
History of 4 codewords from the other	25.6 dB	25.7 dB	17.7 dB	33.7 dB

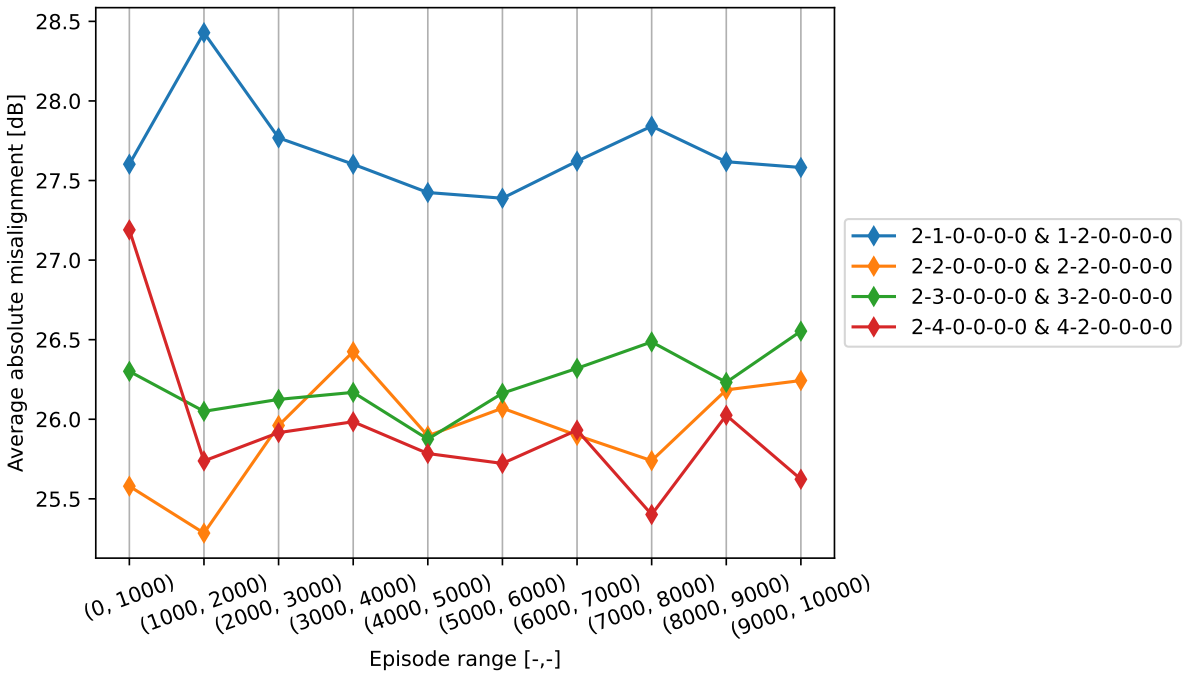


Figure B.2: Average absolute misalignment from tests with multiple distributed agents having different length codeword history of the other agent's actions. Results are from the last 1000 episodes in each test.
Legend values indicate: Nb_r-Nb_t-N_r-R_o-R_d-R_a for UE & BS

By comparing the results from this test with the results on Fig. B.1, it is clear that the performance gets worse by adding information about the other agents actions. This might be a sign that the information added by including the actions of both agents in their states, does

not outweigh the increase in state space which makes it harder for the agents to keep accurate action-value estimates.

Tuning of context information w. partial and full communication

While adding communication about codewords did not improve performance, this might not be the case with context information. Thus, it was investigated what happens when communication is gradually added regarding context information. First only partial communication is added, by giving both agents access to the relative position of the UE. Thus the orientation information about the UE is only available to the UE. This leads to two tests; one where the UE does not use orientation information and one where it does. The results from these tests are shown in Table B.10 and Fig. B.3, with the results from the test with no communication is added as a reference in black.

Table B.10: Table of different misalignment measurements of multiple distributed agents with and without orientation information in the state of the UE. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Reference, no comm.	21.3 dB	20.8 dB	10.7 dB	30.4 dB
Without orientation information	13.6 dB	11.9 dB	4.2 dB	20.6 dB
With orientation information	10.5 dB	7.2 dB	2.4 dB	16.5 dB

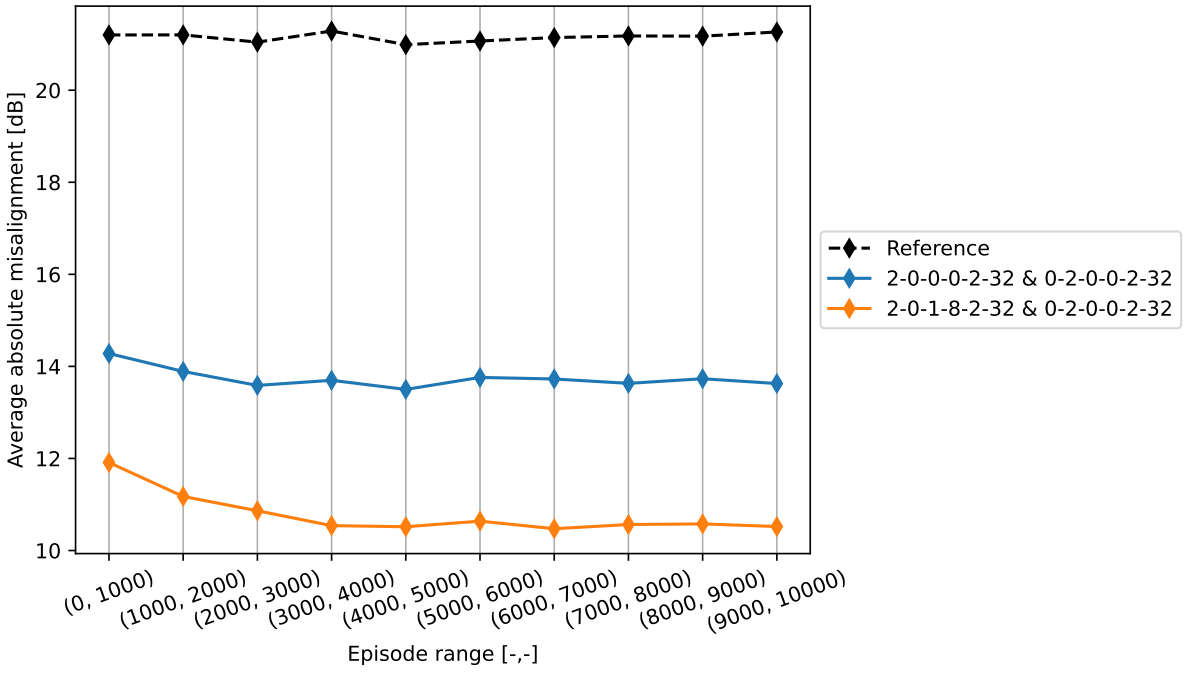


Figure B.3: Average absolute misalignment from tests with multiple distributed agents with and without orientation information in the state of the UE.
Legend values indicate: Nb_r-Nb_t-N_r-R_o-R_d-R_a for UE & BS

As for the single agent case the relative location by itself provides a considerable improvement in performance. Adding the orientation information as well results in a larger performance improvement than in the single case. This can be caused by the fact the this information is

largely useless for the BS, as the best direction to beamform in a LOS scenario is always the direction pointing at UE, regardless of the orientation of the UE. However, for the UE the best direction to beamform depends very much on its current orientation. Thus it makes sense that when the two actions are decentralized the addition of orientation information only where it is useful has a large effect.

Following these tests, the case with full communication was investigated. Here both agents has access to all information, as in the centralized case. As with the tests with only partial communication, a few combinations are possible, where the agents utilize different amounts of the available information. Even though it was proven earlier that codeword information should not be shared between agents, these tests include it anyway to act as a comparison with the single agent case. The results from these tests are shown in Table B.11 and Fig. B.4, with the results from the test with only codeword communication is added as a reference in black.

Table B.11: Table of different misalignment measurements of multiple distributed agents with full communication available between agents. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Reference, no context information.	26.2 dB	26.3 dB	18.1 dB	34.4 dB
Without orientation information	14.6 dB	13.1 dB	4.9 dB	21.9 dB
With orientation information, UE	11.1 dB	8.1 dB	2.9 dB	17.3 dB
With orientation information, both	11.7 dB	8.7 dB	2.9 dB	18.4 dB

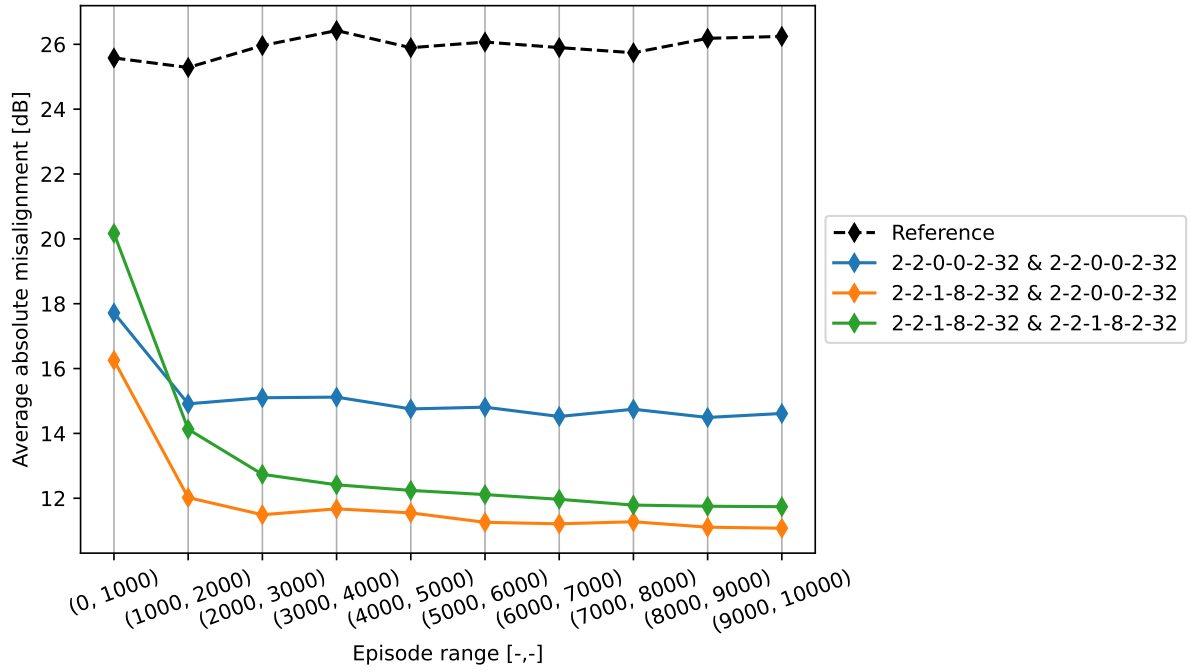


Figure B.4: Average absolute misalignment from tests with multiple distributed agents with full communication available between agents.
Legend values indicate: Nb_r-Nb_t-N_r-R_o-R_d-R_a for UE & BS

The location still provides a large increase in performance by itself. A more interesting result is the fact that the state pair where both agents know the orientation of the UE, performs slightly worse than when only the UE knows. This supports the theory presented for the results in the

case with partial communication. Additionally, even with context information, it is still worse for an agent to use knowledge about the other agents actions. This would probably point to the size of the state space being a problem to some degree, and this information being of fairly little importance.

Thus in summary, for the multi agent case the best performance is found when there is partial communication between the agents, with location information being available to both agents. The optimal combinations of state parameters are shown below for the UE agent and BS agent respectively.

	Nb_r	Nb_t	N_o	R_o	R_d	R_a
Optimal parameter for UE	2	0	1	8	2	32

Table B.12: Optimal state parameters, for an agent on the UE

	Nb_r	Nb_t	N_o	R_o	R_d	R_a
Optimal parameter for the BS	0	2	0	0	2	32

Table B.13: Optimal state parameters, for an agent on the BS

Analysis of hyper parameters for a single agent with adjacent actions

C

C.1 Purpose

In this test report multiple values of hyper parameters, for a single centralized agent with adjacent action scheme, will be tested to evaluate performance. The hyper parameters will be tested by simulating centralized agents and recording performance metrics. Evaluation and comparison will be based on the recorded metrics, and will focus on general performance. These tests are conducted to gain insight into the optimal value of the hyper parameters which leads to best performance. In order to compare values and find a good one, the performance metric is defined in the next section.

C.2 Test metric(s)

The performance metrics used in this report are all based on the misalignment between the received signal power and the highest achievable signal power at each time step. This metric is used instead of the received signal power alone, as this value is expected to vary significantly as the agent moves around the environment, regardless of the choices made. This is due to phenomena like free space loss and absorption, and thus a low received signal power might not necessarily correspond to bad performance of the agent.

Thus the misalignment is used instead, which is calculated as the difference between the received signal power and maximum achievable signal power in dB scale. The maximum signal power is defined as the maximum found when looking at all possible combinations of codewords in the entire codebook and not just those available to the agent at the current time step.

$$M = S_m - S_{max}$$

where

M :	$[dB]$	Misalignment	$RM \in \mathcal{R}$
S_m :	$[dB]$	Received signal power	$S_m \in \mathcal{R}$
S_{max} :	$[dB]$	Maximum achievable signal power	$S_{max} \in \mathcal{R}$

Both the received and maximum power used for calculating the misalignment are without added noise, to better allow the metric to measure the quality of the choices made. This is based on

the notion that choices should be made based on the quality of the channel, which is assumed to be independent of the thermal noise in the equipment. It should be noted that this metric is only obtainable because the test is simulated, as normally the maximum achievable signal power is not directly available.

The metrics used to compare the different agent configurations are:

- Average misalignment
- Median misalignment
- 1^{st} quartile misalignment
- 3^{rd} quartile misalignment

C.3 Test procedure

The test setup is generally the same for tuning of state parameters and hyper-parameters, with one important difference which will be discussed later. The general setup is as follows:

Table C.1: Technical parameters used when tuning agent parameters.

Parameter	Value	Description
B	400 MHz	Total bandwidth
f_c	28 GHz	mmWave carrier frequency
P_{TX}	40 dBm	Transmission power
r_c	200 m	Cell radius
N_r	8	Number of receive antennas
N_t	16	Number of transmit antennas
M	10000	Number of episodes pr. simulation
N	7000	Number of steps pr. episode

A simulation consists of several episodes, with each episode consisting of one or more agents following a path, taking an action at each step. A track is chosen at random from a collection of 16 tracks at the beginning of each episode. Each track in the pedestrian collection is 20000 steps long, however the path used in each episode is only 7000 steps long. Thus a random starting point on the track is chosen as well at the beginning of each episode. By keeping the episodes shorter than the length of the tracks, it is possible to get more unique paths for the agent to follow in different episodes, along with a lot of paths with differing amounts of overlap. This means that it is not just the same 16 paths which are traversed over and over again in the simulation. 7000 steps was chosen as a fitting trade-off between path length and number of unique paths in total. The agent keeps the Q-table values between episodes, and thus learns across all episodes.

In order to ensure that the test metrics are statistically accurate, and to give the agent a chance to explore the entire state space, many episodes are simulated. Both the number of episodes and number of steps in each episode, are derived based on an intermediately sized state space. The total number of steps in the simulation should be enough for an agent to explore all state-action pair, to give accurate value estimates. The state given below, based on a single agent implementation, is assessed to be an intermediate state, and was used to derive the total number of steps.

- A history of the 2 previous beams at the UE
- A history of the 2 previous beams at the BS
- A distance between the UE and BS with a resolution of 4
- An angle between the UE and the BS with a resolution of 8

The size of the combined state-action space of this state was determined in Section 4.4.2 to be $6.05 \cdot 10^6$ state-action pairs. In order to allow proper exploration, the total number of steps for a simulation is set such that each state-action pair can be visited approximately ten times, resulting in a total step number of $70 \cdot 10^6$ steps. This gives a total of 10000 episodes in a simulation. While this allows an agents performance to converge, an even better performance measure would be an average over a statistically significant amount of agents. However, as each simulation takes approximately four hours to run, this is sadly not viable because of time and resource constraints. Thus the results presented in this chapter will be judged with this uncertainty in mind.

Additionally, the simulation has the following settings:

- The update rule used is SARSA.
- The codebooks used are a 3 layer and a 4 layer hierarchical codebook for the UE and BS respectively.
- Transmit power at the BS is 40 dBm.
- The frequency of the carrier wave is 28 GHz
- The actions are encoded with the adjacent actions only encoding.

As mentioned before, the test methodology is slightly different for the hyper-parameters, than for the states. When tuning the hyper-parameters, first a training period is undergone, followed by validation period. The training and validation periods both follow the general setup above, except during the validation period, the tracks are chosen from a validation collection which consists of all new tracks. In the training period the hyper-parameters will take on a fixed set of default values regardless of the test. Then in the validation period the hyper-parameters switch to the values to be tested. This extra step is to ensure the parameters are tested on similar agents, and to investigate if there is overfitting, which is tested by having the environment in the validation period be independent from the one in the training period.

All the results from hyper-parameter tuning in this chapter are from tests following this methodology. However, after the tests were performed, further reflection on the method found that it does not make a lot of sense to do it this way, if the intention is to investigate overfitting. Further comments about the effect of this mistake and a discussion about what should have been done instead can be found in Chapter 7.

C.3.1 Variables

The algorithm consists of a number of variables, where every variable except the hyper parameters will be kept constant throughout this test to ensure valid comparisons. The constant variables are the following.

$\mathcal{S}$: [−] State space $\mathcal{S} =' 2-2-1-8-2-32'$
A : [−] Action space $A = adjacent$

The action space A is the adjacent action space as described in Section 4.1.2.

The state space is chosen as the best performing, from the tests described in A.

C.3.2 Values to be tested

It is infeasible to test every and all value combinations and therefore a subset of values will be identified for testing. As resources are limited the search is conducted over each variable separately. That is optimization will be performed with respect to a single variable at a time, instead of over them jointly. The optimization will be performed in the order given below. The value the variable takes before being optimized is also given.

γ :	[−]	Forgetting factor	$\gamma = 0.7$
α :	[−]	Action value adjustment amplitude	$\alpha = 0.05$
ϵ *constant* :	[−]	Exploration rate	ϵ *constant* $= 0.05$
ϵ *decaying* :	[−]	Exploration rate	ϵ *decaying* $= N/A$
ϵ *adaptive* :	[−]	Exploration rate	ϵ *adaptive* $= N/A$

A thorough explanation of these variables can be found in Section 4.4.4. γ is chosen to be 0.7 which means the series of weights becomes, 0.7 0.49 0.34 0.24 0.17 0.12 and 0.08 and so on. An actions only impact on the state, is on the beam/action history, as the other factors cannot be influenced directly. This means the actions only have an impact on the possible codeword selection in the codebook. This impact does not extend infinitely however as the codebook is of finite state, and any codeword can be reached in a limited amount of actions. Specifically for a codebook with four layers any codeword can be reached in at least seven actions, and therefore values beyond this point should not have great effect.
α is chosen to be 0.05 which, according to [22, p. 222], means the the action value estimate converges in approximately 20 steps, assuming the target is unchanged. As an initial value this is deemed an appropriate value.
ϵ *constant* is chosen to be 5% or 1 out of 20 steps is an exploration step. The optimal value of this is highly dependant on the problem, and if the environment is dynamic, how quickly it changes. How often states are visited also influences the choice, if a certain amount of exploration in each state is wanted. In the interest of time this value is based on preliminary test, where 0.05 lead to promising results.
ϵ*decaying* is not directly chosen but calculated based on the equation given below

$$\epsilon decaying = exp(-\frac{t}{w_d}) \tag{C.1}$$

Where w_d is a weight factor controlling the rate of decay of the logarithmic decreasing function.
ϵ*adaptive* is not directly chosen but calculated based on a set of equations from [25]. A separate ϵ belongs to each state which is adjusted based on the following equation

$$\epsilon = \delta \cdot \epsilon + (\delta - 1) \cdot \epsilon_{adj} \tag{C.2}$$

where δ is a value weighting how much ϵ should be influenced by the calculated ϵ_{adj} value. ϵ_{adj} is a value calculated based on the TD-error as given below

$$\epsilon_{adj} = \frac{1 - exp(\frac{-abs(\alpha \cdot TD - error)}{\sigma})}{1 + exp(\frac{-abs(\alpha \cdot TD - error)}{\sigma})} \tag{C.3}$$

where α is the step size and σ is a weight adjusting how sensitive ϵ_{adj} is to changes in the TD-error.

These values are varied in a neighbourhood around their default values given above. The different values being tested are given in Table C.2.

γ	0.25	0.5	0.7	0.95		
α	0.005	0.01	0.05	0.1	0.15	0.25
ϵ *constant*	0.005	0.01	0.05	0.1		
ϵ *decaying weight*	300	600	900	1200		
ϵ *adaptive weight*	0.1	3	15	20		

Table C.2: Value of variables to be tested

The values of ϵ decaying and ϵ adaptive are unlike that of ϵ constant, in the sense that the latter simply describes the value of ϵ while the two former describe a weight, which has different impact depending on the method.
The values of ϵ *decaying* are that of w_d in Eq. (C.1). They are chosen in such a range that ϵ converges to near zero at steps approximately equally space throughout the 7000 steps in a test, and is resat at the end of an episode.
The values of ϵ *adaptive* are that of $\sigma \cdot 10^3$ in Eq. (C.3). They are chosen such that the average TD-error obtained in the case of a ϵ *constant* results in an exploration rate in the neighborhood of that same constant ϵ.

C.4 Results/Conclusion

In the following section the results of the simulations of agents with the states given in Appendix C.3.2 are presented. The legend on every figure describes the state and the hyper parameters an agent had when generating the data. This state is encoded in the same way, and order as Table 4.1. The hyper parameters are encoded as, starting value of ϵ, value of α, value of γ and if applicable a weight factor, respectively.

Table C.3: Table of different misalignment measurements of single centralized agents with different discounting factors. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
$\gamma = 0.50$	11.8 dB	7.2 dB	1.9 dB	19.3 dB
$\gamma = 0.70$	10.4 dB	6.5 dB	1.4 dB	16.5 dB
$\gamma = 0.25$	15.6 dB	12.5 dB	2.8 dB	26.4 dB
$\gamma = 0.95$	10.8 dB	7.8 dB	2.5 dB	17.0 dB

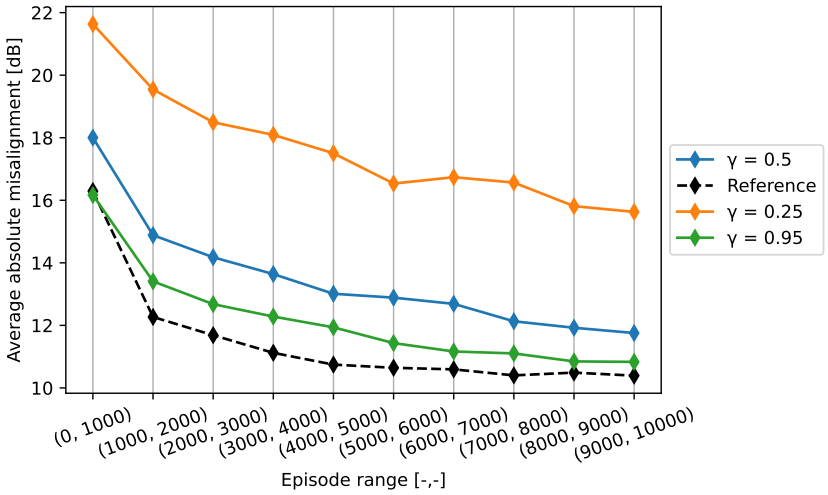


Figure C.1: Forgetting factor
Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a & ϵ α γ

It can be seen that changing the forgetting factor significantly in either direction worsens the performance. The initial choice, and it's reasoning is therefore supported.

Table C.4: Table of different misalignment measurements of single centralized agents with different step sizes. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
$\alpha = 0.01$	9.9 dB	5.8 dB	1.7 dB	15.5 dB
$\alpha = 0.005$	10.9 dB	7.3 dB	2.7 dB	17.2 dB
Reference	10.4 dB	6.5 dB	1.4 dB	16.5 dB
$\alpha = 0.15$	10.6 dB	6.3 dB	0.9 dB	17.4 dB
$\alpha = 0.33$	11.5 dB	7.8 dB	1.9 dB	18.4 dB

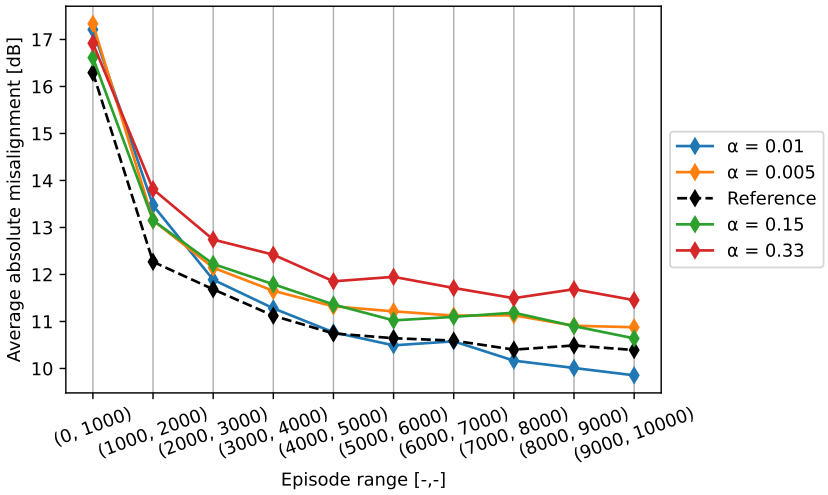


Figure C.2: Stepsize
Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a & ϵ α γ

The step size does not have a large impact on performance. The best value is 0.01 however only

by a small margin. The performance of the non-optimized state is included for comparison.

Table C.5: Table of different misalignment measurements of single centralized agents with different constant ϵ-values. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Constant: $\epsilon = 0.01$	9.4 dB	5.9 dB	2.4 dB	14.3 dB
Constant: $\epsilon = 0.1$	10.7 dB	6.8 dB	2.2 dB	17.0 dB
Constant: $\epsilon = 0.005$	9.3 dB	5.9 dB	2.3 dB	13.7 dB
Constant: $\epsilon = 0.05$	9.9 dB	5.8 dB	1.7 dB	15.5 dB
Reference	10.4 dB	6.5 dB	1.4 dB	16.5 dB

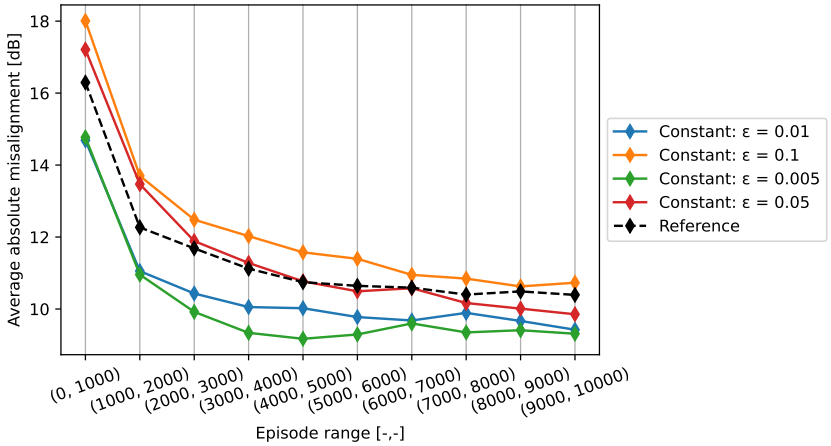


Figure C.3: Constant epsilon
Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a & ϵ α γ

Table C.6: Table of different misalignment measurements of single centralized agents with different decaying ϵ-values. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Reference	10.4 dB	6.5 dB	1.4 dB	16.5 dB
Decaying: $w = 300$	10.9 dB	7.0 dB	3.2 dB	16.3 dB
Decaying: $w = 600$	11.0 dB	7.1 dB	2.8 dB	17.0 dB
Decaying: $w = 900$	11.7 dB	7.3 dB	2.7 dB	18.6 dB
Decaying: $w = 1200$	13.2 dB	9.2 dB	3.4 dB	21.1 dB

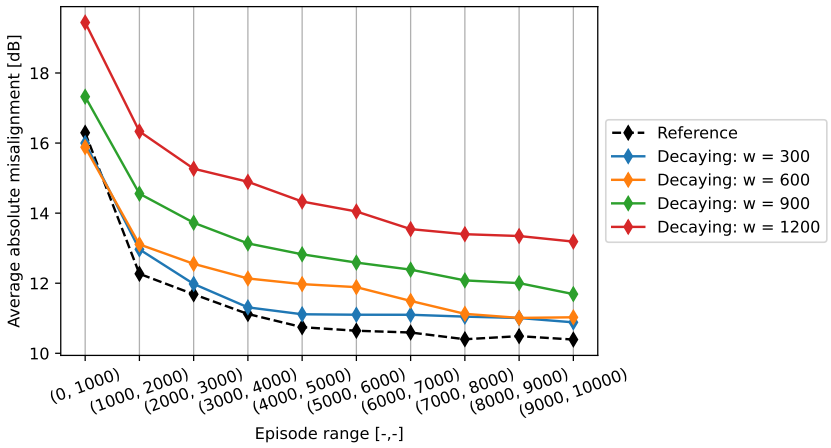


Figure C.4: Decaying epsilon
Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a & ϵ α γ w_d

The decaying ϵ is consequently worse, this is not however unexpected. In a dynamic environment completely getting rid of the exploration rate would likely result in outdated value estimate. Keeping the exploration rate high seems worse, which would result in less exploitation of knowledge.

Table C.7: Table of different misalignment measurements of single centralized agents with different adaptive ϵ-values. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Decaying: $w = 0.1$	11.7 dB	8.6 dB	3.3 dB	17.9 dB
Decaying: $w = 3$	10.6 dB	7.0 dB	2.8 dB	16.4 dB
Decaying: $w = 15$	8.6 dB	5.1 dB	2.2 dB	12.3 dB
Decaying: $w = 20$	10.0 dB	6.5 dB	2.8 dB	14.9 dB
Reference	10.4 dB	6.5 dB	1.4 dB	16.5 dB

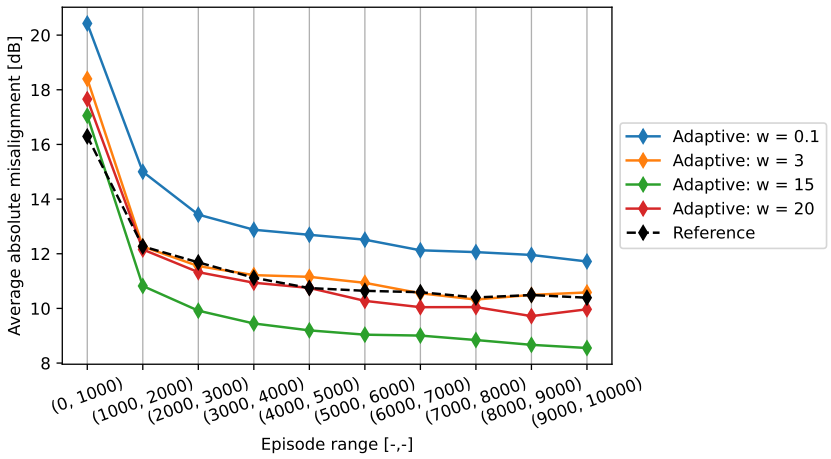


Figure C.5: Adaptive epsilon
Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a & ϵ α γ $\sigma \cdot 10^3$

The adaptive ϵ results in even better performance than either of the other methods, however the

price is the increase in complexity as the value has to be calculated. It should be noted more values have been tested for this method, however to prevent cluttered graphs only a subset were plotted.

This leads to the conclusion that the optimal values for the hyper parameters for a single centralized agent are given in the list below

γ :	[−]	Forgetting factor	$\gamma = 0.7$
α :	[−]	Action value adjustment amplitude	$\alpha = 0.01$
ϵ *adaptive* :	[−]	Exploration rate	ϵ *adaptive* $= N/A$, $\sigma = 15 \cdot 10^{-3}$

Analysis of hyper parameters for multiple agents with adjacent actions

D

D.1 Purpose

In this test report multiple values of hyper parameters, for a multiple agents with adjacent action scheme, will be tested to evaluate performance. The hyper parameters will be tested by simulating agents and recording performance metrics. Evaluation and comparison will be based on the recorded metrics, and will focus on general performance. These tests are conducted to gain insight into the optimal value of the hyper parameters which leads to best performance. In order to compare value and find a good one, the performance metric is defined in the next section.

D.2 Test metric(s)

The performance metrics used in this report are all based on the misalignment between the received signal power and the highest achievable signal power at each time step. This metric is used instead of the received signal power alone, as this value is expected to vary significantly as the agent moves around the environment, regardless of the choices made. This is due to phenomena like free space loss and absorption, and thus a low received signal power might not necessarily correspond to bad performance of the agent.

Thus the misalignment is used instead, which is calculated as the difference between the received signal power and maximum achievable signal power in dB scale. The maximum signal power is defined as the maximum found when looking at all possible combinations of codewords in the entire codebook and not just those available to the agent at the current time step.

$$M = S_m - S_{max}$$

where

M :	$[dB]$	Misalignment	$RM \in \mathcal{R}$
S_m :	$[dB]$	Received signal power	$S_m \in \mathcal{R}$
S_{max} :	$[dB]$	Maximum achievable signal power	$S_{max} \in \mathcal{R}$

Both the received and maximum power used for calculating the misalignment are without added noise, to better allow the metric to measure the quality of the choices made. This is based on the notion that choices should be made based on the quality of the channel, which is assumed to be independent of the thermal noise in the equipment. It should be noted that this metric is only obtainable because the test is simulated, as normally the maximum achievable signal power is not directly available.

The metrics used to compare the different agent configurations are:

- Average misalignment
- Median misalignment
- 1^{st} quartile misalignment
- 3^{rd} quartile misalignment

D.3 Test procedure

The test setup is generally the same for tuning of state parameters and hyper-parameters, with one important difference which will be discussed later. The general setup is as follows:

Table D.1: Technical parameters used when tuning agent parameters.

Parameter	Value	Description
B	400 MHz	Total bandwidth
f_c	28 GHz	mmWave carrier frequency
P_{TX}	40 dBm	Transmission power
r_c	200 m	Cell radius
N_r	8	Number of receive antennas
N_t	16	Number of transmit antennas
M	10000	Number of episodes pr. simulation
N	7000	Number of steps pr. episode

A simulation consists of several episodes, with each episode consisting of one or more agents following a path, taking an action at each step. A track is chosen at random from a collection of 16 tracks at the beginning of each episode. Each track in the pedestrian collection is 20000 steps long, however the path used in each episode is only 7000 steps long. Thus a random starting point on the track is chosen as well at the beginning of each episode. By keeping the episodes shorter than the length of the tracks, it is possible to get more unique paths for the agent to follow in different episodes, along with a lot of paths with differing amounts of overlap. This means that it is not just the same 16 paths which are traversed over and over again in the simulation. 7000 steps was chosen as a fitting trade-off between path length and number of unique paths in total. The agent keeps the Q-table values between episodes, and thus learns across all episodes.

In order to ensure that the test metrics are statistically accurate, and to give the agent a chance to explore the entire state space, many episodes are simulated. Both the number of episodes and number of steps in each episode, are derived based on an intermediately sized state space. The total number of steps in the simulation should be enough for an agent to explore all state-action pair, to give accurate value estimates. The state given below, based on a single agent

implementation, is assessed to be an intermediate state, and was used to derive the total number of steps.

- A history of the 2 previous beams at the UE
- A history of the 2 previous beams at the BS
- A distance between the UE and BS with a resolution of 4
- An angle between the UE and the BS with a resolution of 8

The size of the combined state-action space of this state was determined in Section 4.4.2 to be $6.05 \cdot 10^6$ state-action pairs. In order to allow proper exploration, the total number of steps for a simulation is set such that each state-action pair can be visited approximately ten times, resulting in a total step number of $70 \cdot 10^6$ steps. This gives a total of 10000 episodes in a simulation. While this allows an agents performance to converge, an even better performance measure would be an average over a statistically significant amount of agents. However, as each simulation takes approximately four hours to run, this is sadly not viable because of time and resource constraints. Thus the results presented in this chapter will be judged with this uncertainty in mind.

As mentioned before, the test methodology is slightly different for the hyper-parameters, than for the states. When tuning the hyper-parameters, first a training period is undergone, followed by validation period. The training and validation periods both follow the general setup above, except during the validation period, the tracks are chosen from a validation collection which consists of all new tracks. In the training period the hyper-parameters will take on a fixed set of default values regardless of the test. Then in the validation period the hyper-parameters switch to the values to be tested. This extra step is to ensure the parameters are tested on similar agents, and to investigate if there is overfitting, which is tested by having the environment in the validation period be independent from the one in the training period.

All the results from hyper-parameter tuning in this chapter are from tests following this methodology. However, after the tests were performed, further reflection on the method found that it does not make a lot of sense to do it this way, if the intention is to investigate overfitting. Further comments about the effect of this mistake and a discussion about what should have been done instead can be found in Chapter 7.

D.3.1 Variables

The algorithm consists of a number of variables, where every variable except the hyper parameters will be kept constant throughout this test to ensure valid comparisons. The constant variables are the following.

$\mathcal{S}_r$:	[−]	State space of the UE	$\mathcal{S}_r =' 2-0-1-8-2-32'$
$\mathcal{S}_t$:	[−]	State space of the BS	$\mathcal{S}_t =' 0-2-0-0-2-32'$
A :	[−]	Action space	$A = adjacent$

The action space A is the adjacent action space as described in Section 4.1.2.

The state space is chosen as the best performing, from the tests described in A.

D.3.2 Values to be tested

It is infeasible to test every and all value combinations and therefore a subset of values will be identified for testing. As resources are limited the search is conducted over each variable separately. That is optimization will be performed with respect to a single variable at a time, instead of over them jointly. Furthermore the values of the hyper parameters will be equal for both agents. The optimization will be performed in the order given below. The value the variable takes before being optimized is also given.

γ :	[−]	Forgetting factor	$\gamma = 0.7$
α :	[−]	Action value adjustment amplitude	$\alpha = 0.01$
ϵ *constant* :	[−]	Exploration rate	ϵ *constant* $= 0.01$
ϵ *decaying* :	[−]	Exploration rate	ϵ *decaying* $= N/A$
ϵ *adaptive* :	[−]	Exploration rate	ϵ *adaptive* $= N/A$

A thorough explanation of these variables can be found in Section 4.4.4. γ is chosen to be 0.7, as it was found to be optimal in Appendix C and is therefore used as a starting point. α is chosen to be 0.01, as it was found to be optimal in Appendix C and is therefore used as a starting point. ϵ is chosen to be 1% or 1 out of 100 steps is an exploration step. This value was found to be optimal in Appendix C and is therefore used as a starting point.

$\epsilon decaying$ is not directly chosen but calculated based on the equation given below

$$\epsilon decaying = exp(-\frac{t}{w_d}) \tag{D.1}$$

Where w_d is a weight factor controlling the rate of decay of the logarithmic decreasing function.

$\epsilon adaptive$ is not directly chosen but calculated based on a set of equations from [25]. A separate ϵ belongs to each state which is adjusted based on the following equation

$$\epsilon = \delta \cdot \epsilon + (\delta - 1) \cdot \epsilon_{adj} \tag{D.2}$$

where δ is a value weighting how much ϵ should be influenced by the calculated ϵ_{adj} value. ϵ_{adj} is a value calculated based on the TD-error as given below

$$\epsilon_{adj} = \frac{1 - exp(\frac{-abs(\alpha \cdot TD - error)}{\sigma})}{1 + exp(\frac{-abs(\alpha \cdot TD - error)}{\sigma})} \tag{D.3}$$

where α is the step size and σ is a weight adjusting how sensitive ϵ_{adj} is to changes in the TD-error.

These values are varied in a neighbourhood around their default values given above. The different values being tested are given in Table D.2.

γ	0.6	0.7	0.8	
α	0.005	0.01	0.05	
$\epsilon constant$	0.001	0.005	0.01	
ϵ *decaying weight*	150	300	600	900
ϵ *adaptive weight*	0.3	1	10	15

Table D.2: Value of variables to be tested

The values of ϵ decaying and ϵ adaptive are unlike that of ϵ constant, in the sense that the latter simply describes the value of ϵ while the two former describe a weight, which has different impact depending on the method.
The values of ϵ *decaying* are that of w_d in Eq. (D.1). They are chosen in a range around the optimal value found in C, and is resat at the end of an episode.
The values of ϵ *adaptive* are that of $\sigma \cdot 10^3$ in Eq. (D.3). They are chosen such that the average TD-error obtained in the case of a ϵ *constant* results in an exploration rate in the neighborhood of that same constant ϵ.

D.4 Results/Conclusion

In the following section the results of the simulations of agents with the states given in Appendix D.3.2 are presented. The legend on every figure describes the state and the hyper parameters an agent had when generating the data. This state is encoded in the same way, and order as Table 4.1, where the first states is for the UE, and the second state is for the BS. The hyper parameters are encoded as, starting value of ϵ, value of α, value of γ and if applicable a weight factor, respectively.

Table D.3: Table of different misalignment measurements of multi agents with different γ-values. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Reference	10.5 dB	7.2 dB	2.4 dB	16.5 dB
$\gamma = 0.1$	7.3 dB	3.9 dB	0.1 dB	10.7 dB
$\gamma = 3$	7.7 dB	4.3 dB	0.7 dB	11.4 dB
$\gamma = 15$	8.8 dB	5.5 dB	1.9 dB	13.6 dB

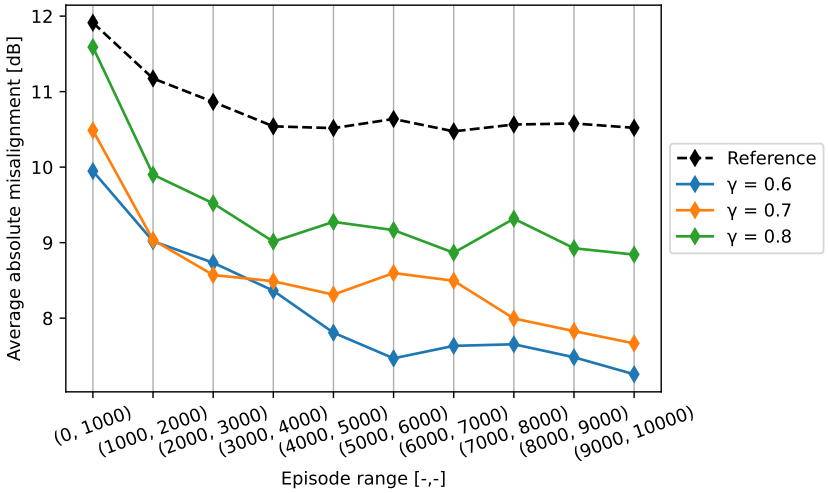


Figure D.1: Forgetting factor
Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a for UE & BS ϵ α γ

It can be seen that the optimal forgetting factor is slightly smaller than in the single agent case. This slightly smaller value may favor the UE more as it's codebook is smaller. The performance of the non-optimized state pair is included for comparison.

Table D.4: Table of different misalignment measurements of single centralized agents with different adaptive ϵ-values. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Reference	10.5 dB	7.2 dB	2.4 dB	16.5 dB
$\alpha = 0.01$	7.3 dB	3.9 dB	0.1 dB	10.7 dB
$\alpha = 0.005$	8.1 dB	4.3 dB	0.9 dB	12.6 dB
$\alpha = 0.05$	8.2 dB	4.6 dB	1.0 dB	12.4 dB

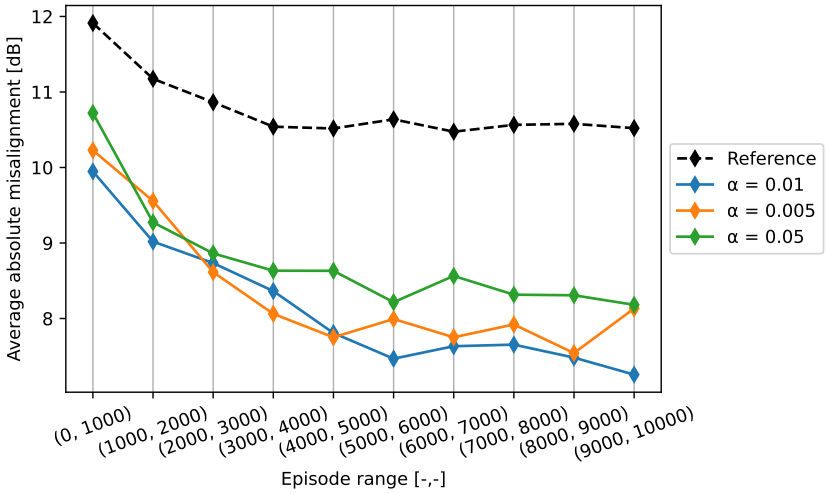


Figure D.2: Step size
Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a for UE & BS ϵ α γ

Table D.5: Table of different misalignment measurements of single centralized agents with different adaptive ϵ-values. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Reference	10.5 dB	7.2 dB	2.4 dB	16.5 dB
Constant: $\gamma = 0.001$	7.7 dB	4.2 dB	0.4 dB	11.5 dB
Constant: $\gamma = 0.01$	7.3 dB	3.9 dB	0.1 dB	10.7 dB
Constant: $\gamma = 0.005$	8.6 dB	5.1 dB	1.7 dB	13.2 dB

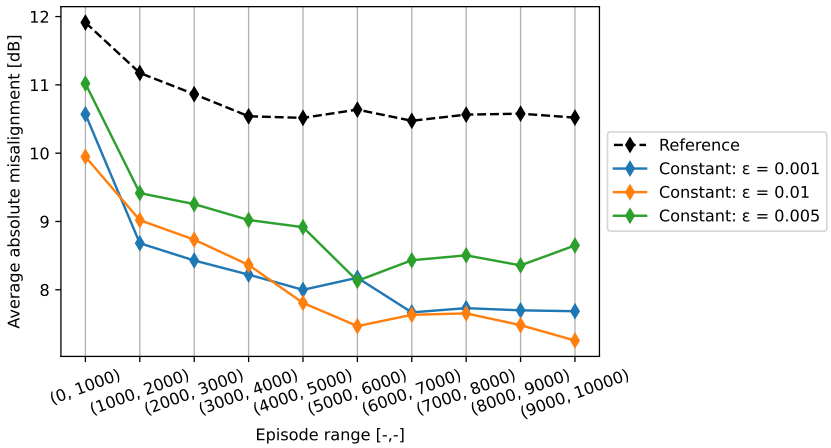


Figure D.3: Constant epsilon
Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a for UE & BS ϵ α γ

The tuning of the constant epsilon has a marginal if any effect on the performance, best seen when comparing Fig. D.2 and Fig. D.3.

Table D.6: Table of different misalignment measurements of single centralized agents with different adaptive ϵ-values. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Reference	10.5 dB	7.2 dB	2.4 dB	16.5 dB
Decaying: $\gamma = 150$	9.1 dB	5.5 dB	2.0 dB	14.0 dB
Decaying: $\gamma = 300$	8.9 dB	5.1 dB	1.3 dB	13.8 dB
Decaying: $\gamma = 600$	9.6 dB	5.6 dB	1.9 dB	15.0 dB
Decaying: $\gamma = 900$	9.9 dB	5.8 dB	1.7 dB	15.6 dB

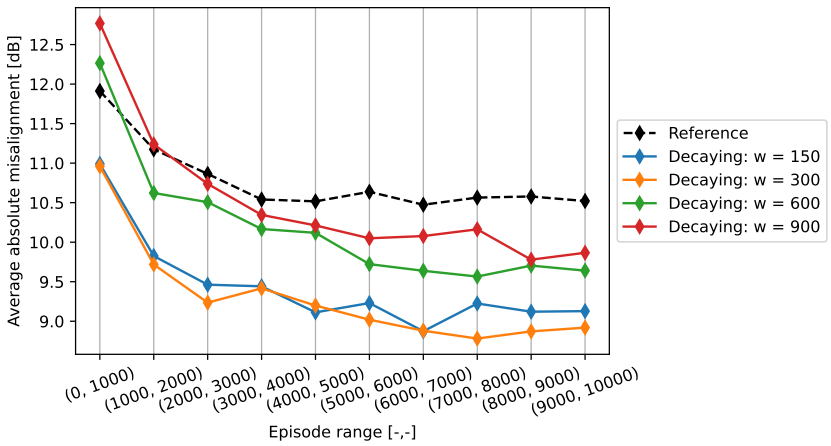


Figure D.4: Decaying epsilon
Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a for UE & BS ϵ α γ w_d

The decaying ϵ delivers poor performance, in agreement with the single agent results from Appendix C.

Table D.7: Table of different misalignment measurements of single centralized agents with different adaptive ϵ-values. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Reference	10.5 dB	7.2 dB	2.4 dB	16.5 dB
Adaptive: $\gamma = 0.3$	9.2 dB	5.2 dB	1.0 dB	14.4 dB
Adaptive: $\gamma = 1$	8.8 dB	5.0 dB	1.3 dB	13.6 dB
Adaptive: $\gamma = 10$	7.8 dB	4.3 dB	0.4 dB	11.8 dB
Adaptive: $\gamma = 15$	8.4 dB	5.0 dB	1.9 dB	12.4 dB

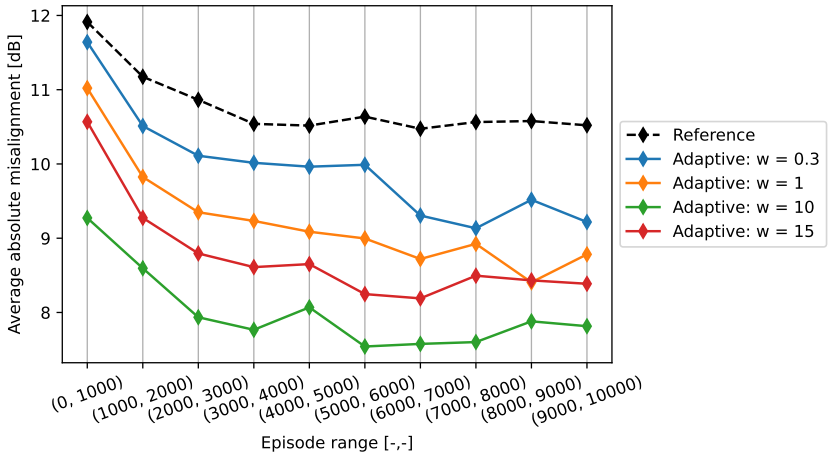


Figure D.5: Adaptive epsilon
Legend values indicate: Nb_r-Nb_r-N_r-R_o-R_d-R_a for UE & BS ϵ α γ $\sigma \cdot 10^3$

An optimal value of an adaptive ϵ delivers improved performance with regard to convergence time, however no performance gain is seen in the final average misalignment.

This leads to the conclusion that the optimal values for the hyper parameters for an agent at each terminal are given in the list below

γ :	[−]	Forgetting factor	$\gamma = 0.6$
α :	[−]	Action value adjustment amplitude	$\alpha = 0.01$
ϵ *adaptive* :	[−]	Exploration rate	ϵ *adaptive* $= N/A$, $\sigma = 10 \cdot 10^{-3}$

Update rule analysis of agent(s) with adjacent actions

E

E.1 purpose

In this test report multiple action value estimate update methods, for agent(s) with adjacent action scheme, will be tested to evaluate performance. The update methods will be tested by simulating agents and recording performance metrics. Both the case of a single centralized agent, and of an agent at each terminal will be simulated. Evaluation and comparison will be based on the recorded metrics, and will focus on general performance. These tests are conducted to gain insight into which update method performs the best. In order to compare the update methods, the performance metric is defined in the next section.

E.2 Test metric(s)

The performance metrics used in this report are all based on the misalignment between the received signal power and the highest achievable signal power at each time step. This metric is used instead of the received signal power alone, as this value is expected to vary significantly as the agent moves around the environment, regardless of the choices made. This is due to phenomena like free space loss and absorption, and thus a low received signal power might not necessarily correspond to bad performance of the agent.

Thus the misalignment is used instead, which is calculated as the difference between the received signal power and maximum achievable signal power in dB scale. The maximum signal power is defined as the maximum found when looking at all possible combinations of codewords in the entire codebook and not just those available to the agent at the current time step.

$$M = S_m - S_{max}$$

where

M :	$[dB]$	Misalignment	$RM \in \mathcal{R}$
S_m :	$[dB]$	Received signal power	$S_m \in \mathcal{R}$
S_{max} :	$[dB]$	Maximum achievable signal power	$S_{max} \in \mathcal{R}$

Both the received and maximum power used for calculating the misalignment are without added noise, to better allow the metric to measure the quality of the choices made. This is based on

the notion that choices should be made based on the quality of the channel, which is assumed to be independent of the thermal noise in the equipment. It should be noted that this metric is only obtainable because the test is simulated, as normally the maximum achievable signal power is not directly available.

The metrics used to compare the different agent configurations are:

- Average misalignment
- Median misalignment
- 1^{st} quartile misalignment
- 3^{rd} quartile misalignment

E.3 Test procedure

The test setup is generally the same as for tuning of state parameters. The general setup is as follows:

Table E.1: Technical parameters used when running tests with different update rules.

Parameter	Value	Description
B	400 MHz	Total bandwidth
f_c	28 GHz	mmWave carrier frequency
P_{TX}	40 dBm	Transmission power
r_c	200 m	Cell radius
N_r	8	Number of receive antennas
N_t	16	Number of transmit antennas
M	10000	Number of episodes pr. simulation
N	7000	Number of steps pr. episode

A simulation consists of several episodes, with each episode consisting of one or more agents following a path, taking an action at each step. A track is chosen at random from a collection of 16 tracks at the beginning of each episode. Each track in the pedestrian collection is 20000 steps long, however the path used in each episode is only 7000 steps long. Thus a random starting point on the track is chosen as well at the beginning of each episode. By keeping the episodes shorter than the length of the tracks, it is possible to get more unique paths for the agent to follow in different episodes, along with a lot of paths with differing amounts of overlap. This means that it is not just the same 16 paths which are traversed over and over again in the simulation. 7000 steps was chosen as a fitting trade-off between path length and number of unique paths in total. The agent keeps the Q-table values between episodes, and thus learns across all episodes.

In order to ensure that the test metrics are statistically accurate, and to give the agent a chance to explore the entire state space, many episodes are simulated. Both the number of episodes and number of steps in each episode, are derived based on an intermediately sized state space. The total number of steps in the simulation should be enough for an agent to explore all state-action pair, to give accurate value estimates. The state given below, based on a single agent implementation, is assessed to be an intermediate state, and was used to derive the total number of steps.

- A history of the 2 previous beams at the UE
- A history of the 2 previous beams at the BS
- A distance between the UE and BS with a resolution of 4
- An angle between the UE and the BS with a resolution of 8

The size of the combined state-action space of this state was determined in Section 4.4.2 to be $6.05 \cdot 10^6$ state-action pairs. In order to allow proper exploration, the total number of steps for a simulation is set such that each state-action pair can be visited approximately ten times, resulting in a total step number of $70 \cdot 10^6$ steps. This gives a total of 10000 episodes in a simulation. While this allows an agents performance to converge, an even better performance measure would be an average over a statistically significant amount of agents. However, as each simulation takes approximately four hours to run, this is sadly not viable because of time and resource constraints. Thus the results presented in this chapter will be judged with this uncertainty in mind.

E.3.1 Variables

The algorithm consists of a number of variables, where every variable except the update method will be kept constant throughout this test to ensure valid comparisons. Two sets of variables will be presented, the first for the case of a single centralized agent, and the next for an agent at each terminal. The first set of constant variables are the following.

γ :	[−]	Forgetting factor	$\gamma = 0.7$
α :	[−]	Action value adjustment amplitude	$\alpha = 0.01$
ϵ *adaptive* :	[−]	Exploration rate	ϵ *adaptive* $= N/A$
$\mathcal{S}$:	[−]	State space	$\mathcal{S} =' 2-2-1-8-2-32'$
A :	[−]	Action space	$A =$ *adjacent*

γ is chosen as the best performing, from the tests described in Appendix C. α is chosen as the best performing, from the tests described in Appendix C. ϵ *adaptive* is chosen as it is the best performing ϵ method and it's weight value is chosen as the best performing, from the tests described in Appendix C. The state space is chosen as the best performing, from the tests described in A.

The second set of variables for an agent at each terminal is:

γ :	[−]	Forgetting factor	$\gamma = 0.6$
α :	[−]	Action value adjustment amplitude	$\alpha = 0.01$
ϵ *adaptive* :	[−]	Exploration rate	ϵ *adaptive* $= N/A$
$\mathcal{S}_r$:	[−]	State space of the UE	$\mathcal{S}_r =' 2-0-1-8-2-32'$
$\mathcal{S}_t$:	[−]	State space of the BS	$\mathcal{S}_t =' 0-2-0-0-2-32'$
A :	[−]	Action space	$A =$ *adjacent*

Note here that the agents at both terminals have the same hyper parameters. γ is chosen as the best performing, from the tests described in Appendix D. α is chosen as the best performing, from the tests described in Appendix D. ϵ *adaptive* is chosen as it is the best performing ϵ method and it's weight value is chosen as the best performing, from the tests described in Appendix D. The state space is chosen as the best performing, from the tests described in B.

E.3.2 Methods to be tested

A limited number of different methods for updating the value estimate will be tested. The methods to be tested are SARSA, Q-learning and weighted average update. All these methods share a similar update rule as given below

$$Q(s_1,q_1) = Q(s_1,q_1) + \alpha \cdot (R + \text{method term} - Q(s_1,q_1)) \tag{E.1}$$

where the method term differs by the method, as given below

Weighted average	$= 0$
SARSA	$= Q(s_2,a_2)$
Q-learning	$= \max_{a_o} Q(s_2,a_o)$

A more thorough explanation of these update rules can be seen in Section 3.2.1.

Additionally for the multi agent case another method, which isn't strictly an update method, WoLf-PHC will also be tested.

E.4 Data processing

As mentioned in Appendix A.3 each test consists of 70.000.000 steps, however plotting the relative misalignment for all steps would not result in a graph indicative of the general performance of the state. To remove the variation within each episode, the misalignment is calculated based on averages. First the linear misalignment is calculated, and transformed to logarithmic scale.

$$MA = 10 \cdot log_{10}(R) - 10 \cdot log_{10}(R_{max})$$

This misalignment is then averaged across all steps within each episodes, resulting in a reduction from 70.000.000 data points to 10.000.

$$RM_{pe} = \frac{1}{7000} \cdot \sum_{i=1}^{7000} RM_i$$

where

RM_m	[−]	Average reward for a given episode
RM_i	[−]	Reward for a the i'th step in a given episode

In order to remove the influence of a specific track/path on the performance multiple episodes are averaged to get the final performance measure. The 10.000 episodes are averaged in ten intervals of 1000 episodes each.

E.5 Results/Conclusion

In the following section the results of the simulations of agents with the update rules given in Appendix E.3.2 are presented. The legend on every figure describes the state and the hyper

parameters an agent had when generating the data. This state is encoded in the same way, and order as Table 4.1. If the test is for a single agent the state is for that centralized agent, if not then the first states is for the UE, and the second state is for the BS. The hyper parameters are encoded as, starting value of ϵ, value of α, value of γ and if applicable a weight factor, respectively.

E.6 Update rule

The update rules that will be tested are SARSA, Q-learning, which are discussed in Section 3.2.1, and a simple update rule that has the immediate reward as target. The simple update rule approximates the action-value as the average immediate reward received when taking that action in that state, and is implemented by modifying SARSA with a discounting faction of $\gamma = 0$. The three updates will be compared with the simple heuristic algorithm to gauge whether this RL solution is worthwhile. The update rules will be judged based on their performance in terms of the average misalignment. They will further be analyzed based on the actions the agent take, with a certain update rule, and based on how often each specific beam was visited.

E.6.1 Single centralized agent

Table E.2: Table of different misalignment measurements of single centralized agents with different update rules. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Heuristic	9.1 dB	6.5 dB	3.7 dB	10.8 dB
Q-learning	10.1 dB	7.1 dB	3.3 dB	14.7 dB
SARSA	11.3 dB	7.7 dB	2.6 dB	17.6 dB
Simple	12.6 dB	6.9 dB	1.8 dB	21.3 dB

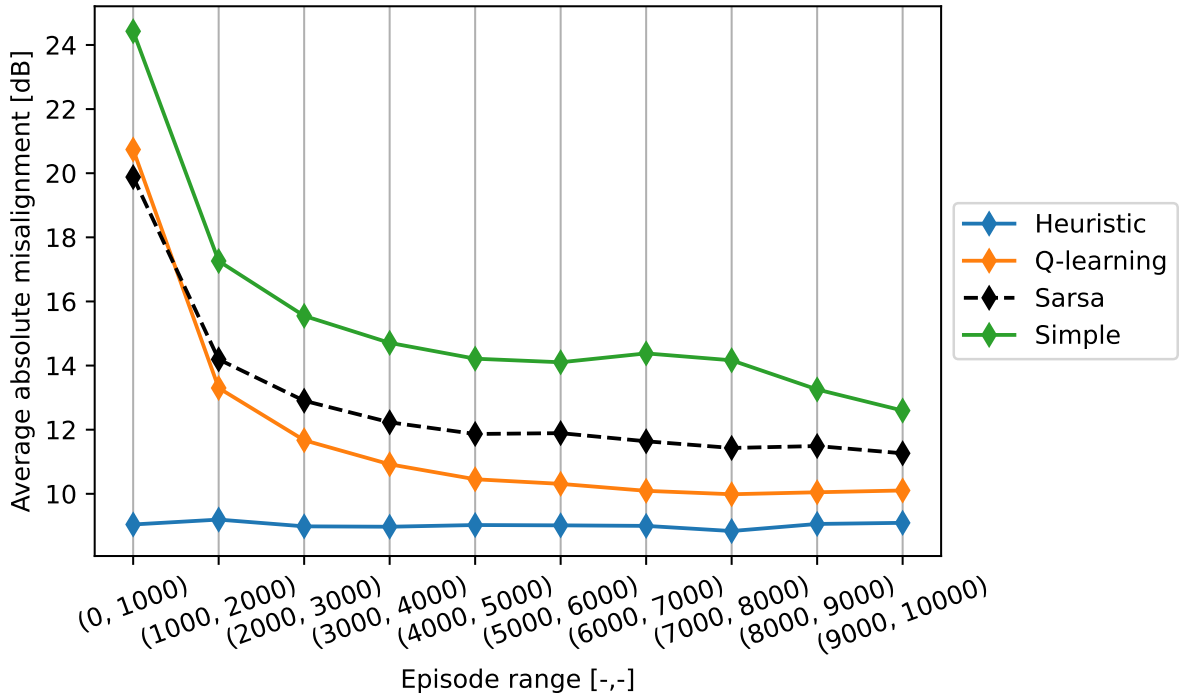


Figure E.1: Absolute misalignment of single centralized agents with different update rules.

In Figure E.1 graphs for three different update rules are plotted. Sarsa, which was used during state and hyper parameters optimization, is plotted in black. The performance of a heuristic algorithm has been plotted for reference. The heuristic algorithm performs well despite it's simplicity, some of this performance can be assigned to the adjacent action encoding. When this algorithm chooses to change beam, an adjacent is chosen. If the currently chosen beam was good, while not all, one of the adjacent beams are likely to be good. The Sarsa algorithm gets outperformed by the heuristic by around 2 dB which is interesting, as the scenario is LOS and should be fairly simple. The size of the value space, and the dynamic environment might be to blame for this. Q-learning outperforms Sarsa slightly, their performance being close isn't unexpcted as when ϵ is small the update is the same most of the time.

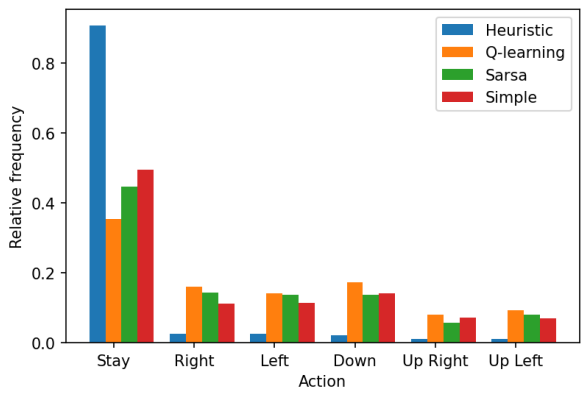


Figure E.2: Histogram of actions of a single centralized agent with different update rules for UE

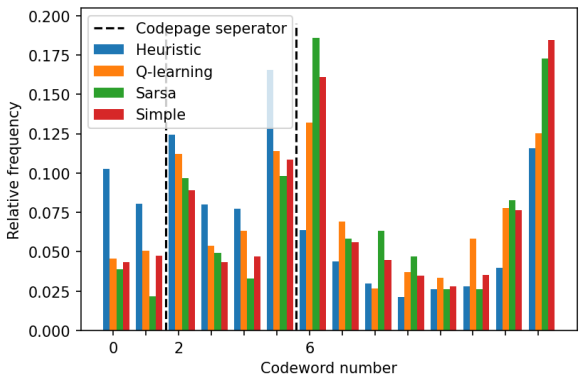


Figure E.3: Histogram of UE codewords chosen for a single centralized agent with different update rules

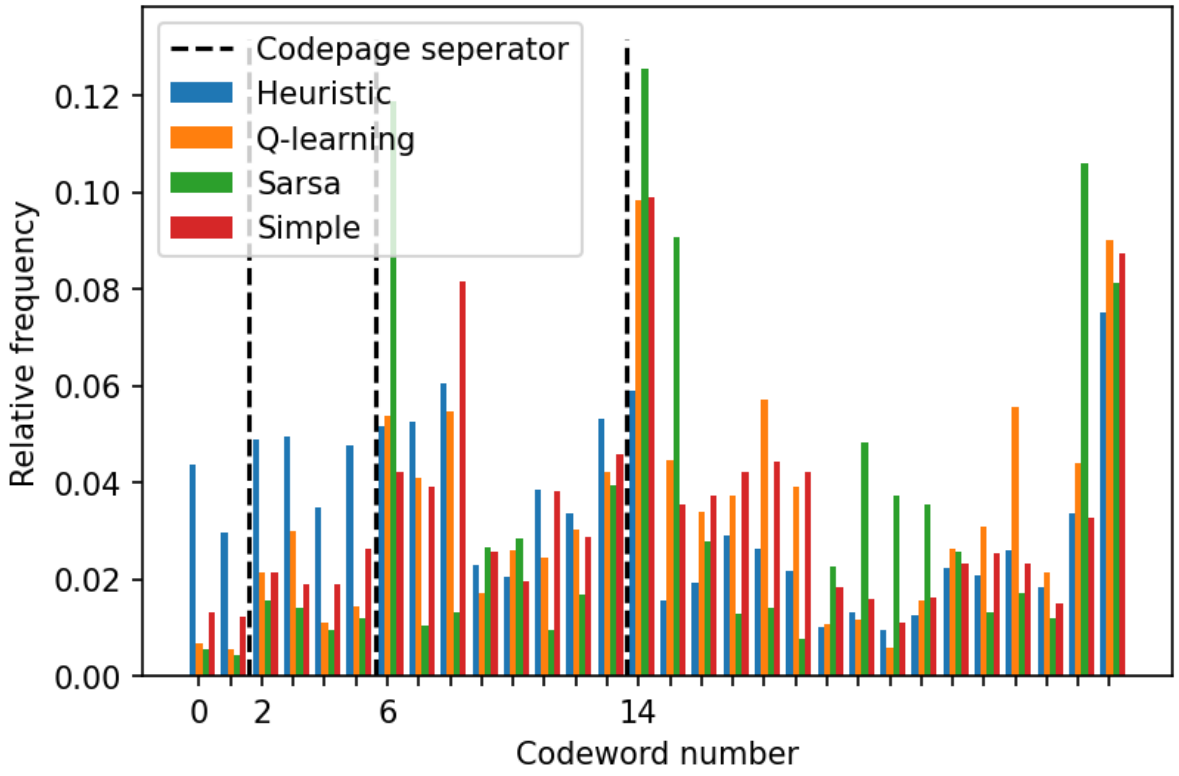


Figure E.4: Histogram of BS codewords chosen for a single centralized agent with different update rules

When inspecting the actions, given in Fig. E.2, as expected the heuristic algorithms chooses the stay most often. The behavior for the three update rules are similar. From Fig. E.3 and Fig. E.4 it can be seen that the heuristic algorithm has the greatest propensity to stay in the broader beams, and lower layers of the codebook. A likely explanation is during simulation if the UE is close to the BS if a broad beam has sufficient power the heuristic algorithm will stick with it, even more so because of it's large coverage. For the UE and to a lesser extent for the BS all update rules tend to gravitate towards beams which lie at the edge of layers in the codebook, see Fig. 4.1. This is not unexpected as these beams, while having broader coverage have similar gain to other beams in the same layer, and are generally better to chose. This is the case even more so for Sarsa, which is sensible as it's an update which values states that are close to low value state less, unlike Q-learning. The beam choices of Q-learning and simple being more similar than Q-learning and Sarsa is peculiar as the two latter are more similar.

E.6.2 Multi agent

Table E.3: Table of different misalignment measurements of multi agents with different update rules. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Heuristic	9.1 dB	6.5 dB	3.7 dB	10.8 dB
Q-learning	10.6 dB	7.6 dB	3.2 dB	16.0 dB
SARSA	10.6 dB	7.2 dB	2.5 dB	16.7 dB
Simple	6.4 dB	3.7 dB	0.1 dB	9.2 dB

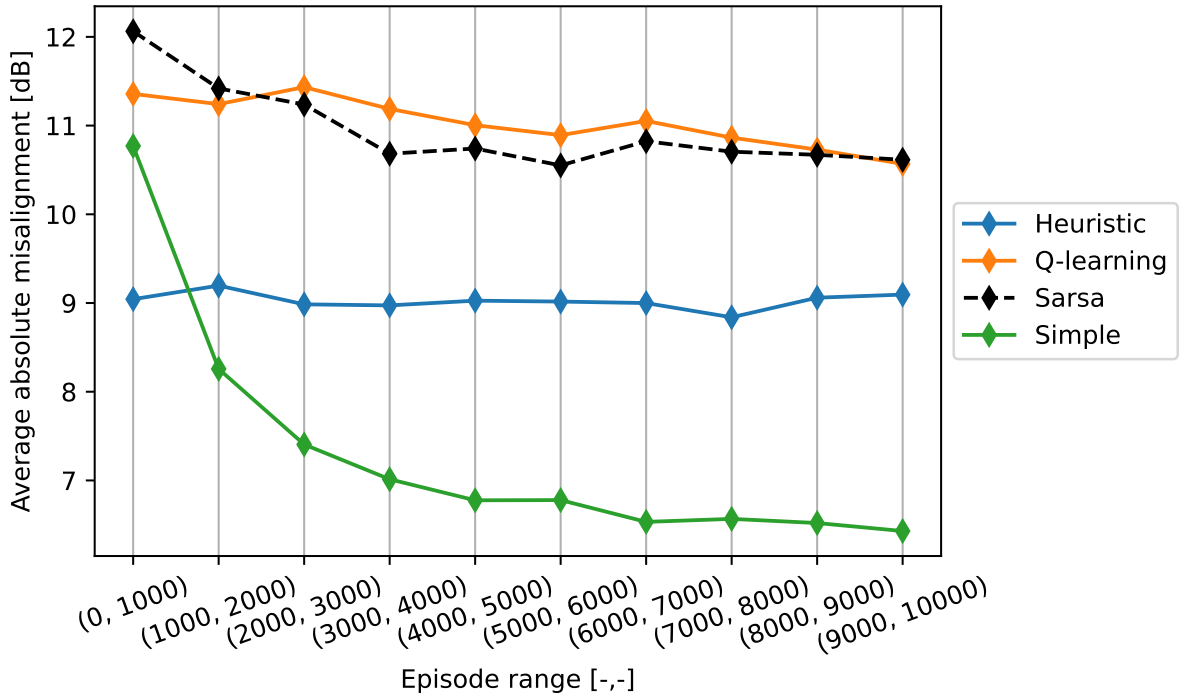


Figure E.5: Absolute misalignment of multi agents with different update rules.

In Fig. E.5 graphs for three different update rules are plotted. Sarsa, which was used during state and hyper parameters optimization, is plotted in black. The performance of a heuristic algorithm has been plotted for reference. Here Sarsa has every comparable performance to Q-learning which is expected, where both get outperformed by the heuristic. Based on the single agent results this is expected. Unlike in the single agent case Q-learning and Sarsa does not require a large amount of episodes to converge to stable performance which is expected as the value space is smaller. The simple converges to a considerably better performance than both the heuristic and the other two update methods quickly. This is surprising considering both it's a simpler method, but also the results for the single agent case. The larger value space could have impaired the algorithm in the single agent case. The method clearly learns over the entire test, an quiet aggressively

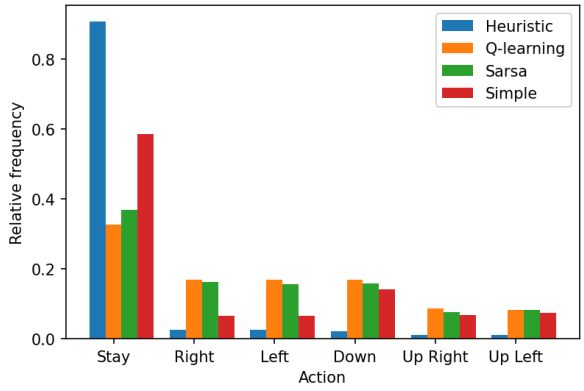


Figure E.6: Histogram of actions for multi agents with multiple update rules for UE

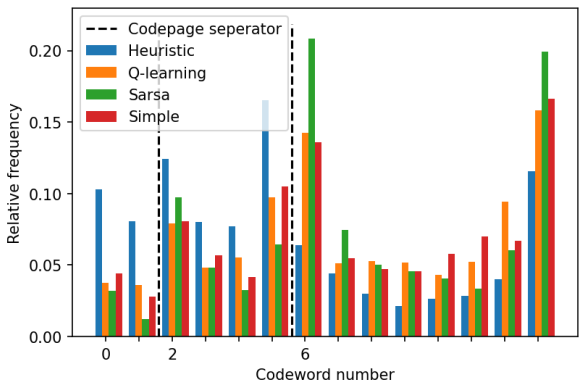


Figure E.7: Histogram of UE codewords chosen for multi agents with different update rules

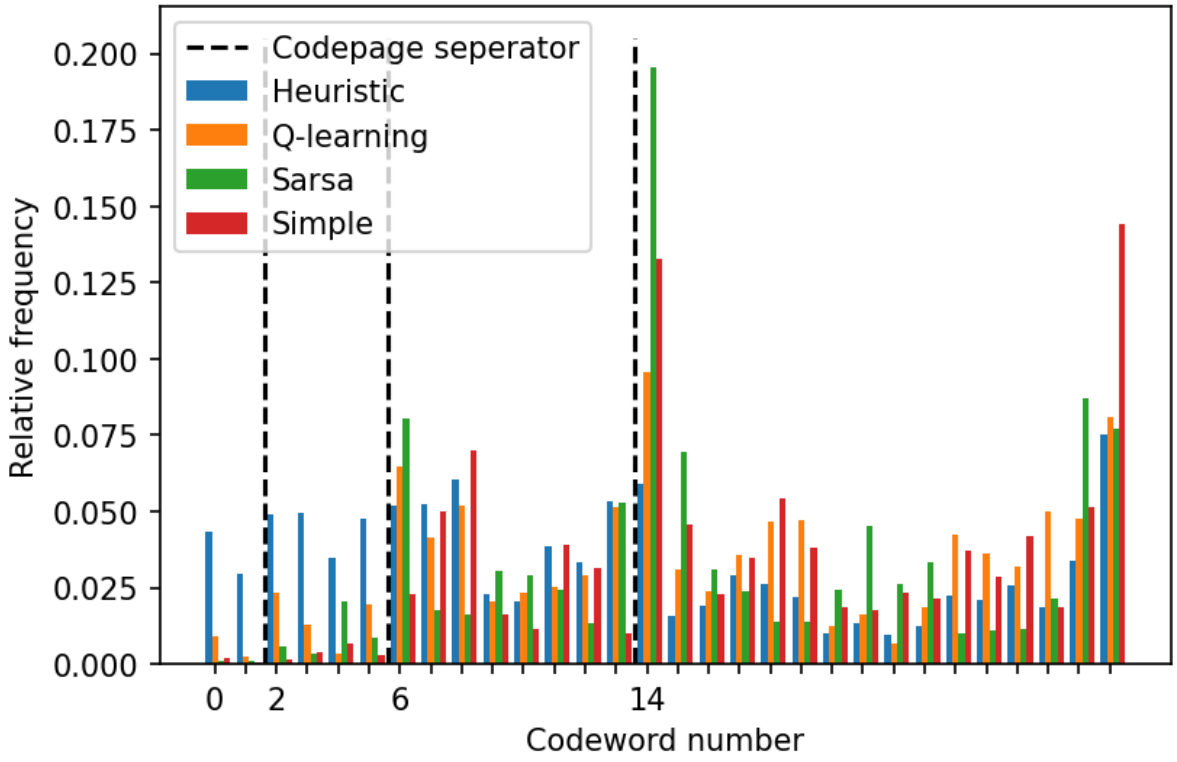


Figure E.8: Histogram of BS codewords chosen for multi agents with different update rules

The heuristic algorithm still favors the stay option more than the update rules. Simple does favor the stay option meaningfully more than the other update rules, unlike in the single agent case. While the simple algorithm moves up and down layers with a similar frequency as the other update rules it chooses left and right less often, likely meaning the simple algorithm is more likely to stick with a good beam. The beam choices, in Fig. E.7 and Fig. E.8 are largely similar to those for the single agent case. The simple algorithm behaves less like Q-learning than in the single agent case, while still being somewhat similar. Again broader beams are largely preferred.

An additional series of graphs have been illustrated below for the multi agent case. This is to prove the WoLF-PHC algorithm with a point of comparison, while technically using the Q-learning update rule, this was deemed the most appropriate place to include it, as it has a similar effect on the policy as the other update rules.

Table E.4: Table of different misalignment measurements of multi agents with different update rules and the WoLf-PHC algorithm. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Q-learning	10.6 dB	7.6 dB	3.2 dB	16.0 dB
SARSA	10.6 dB	7.2 dB	2.5 dB	16.7 dB
WOLF	11.0 dB	7.9 dB	2.9 dB	17.1 dB

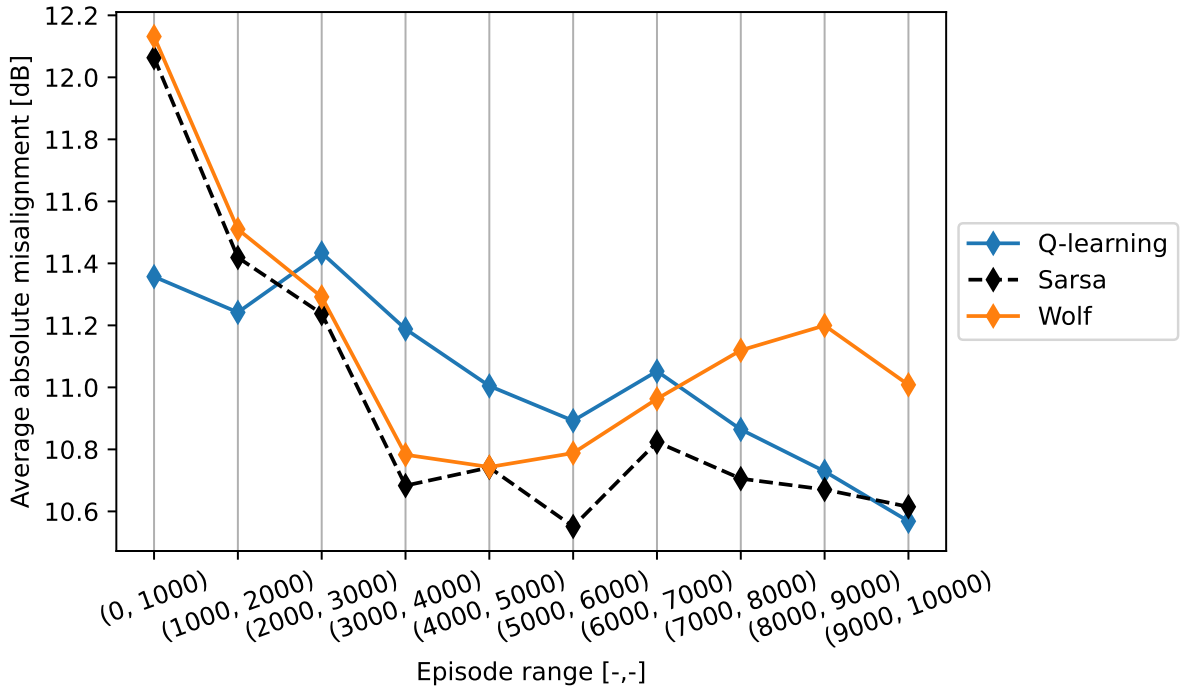


Figure E.9: Absolute misalignment of multi agents with different update rules and the WoLf-PHC algorithm.

In Fig. E.9 graphs for two different update rules are plotted. Sarsa, which was used during state and hyper parameters optimization, is plotted in black. The performance of the WoLF-PHC algorithm has been plotted for comparison. The performance WoLF-PHC is similar to the two update methods. This isn't unexpected as WoLF-PHC uses the Q-learning update rule, which was previously shown to perform similarly to Sarsa. The fact that it does not improve the performance despite being an algorithm developed specifically for multi agent environments is interesting. A cause could be the actions limited influence on the environment.

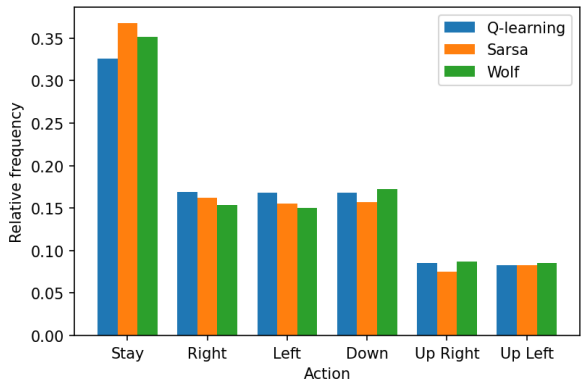


Figure E.10: Histogram of actions for multi agents with multiple update rules and the WoLf-PHC algorithm. for UE

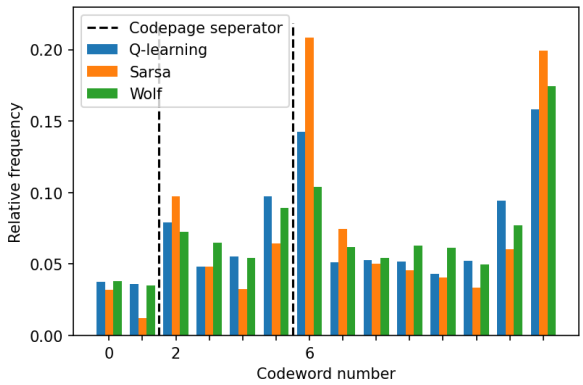


Figure E.11: Histogram of UE codewords chosen for multi agents with different update rules and the WoLf-PHC algorithm.

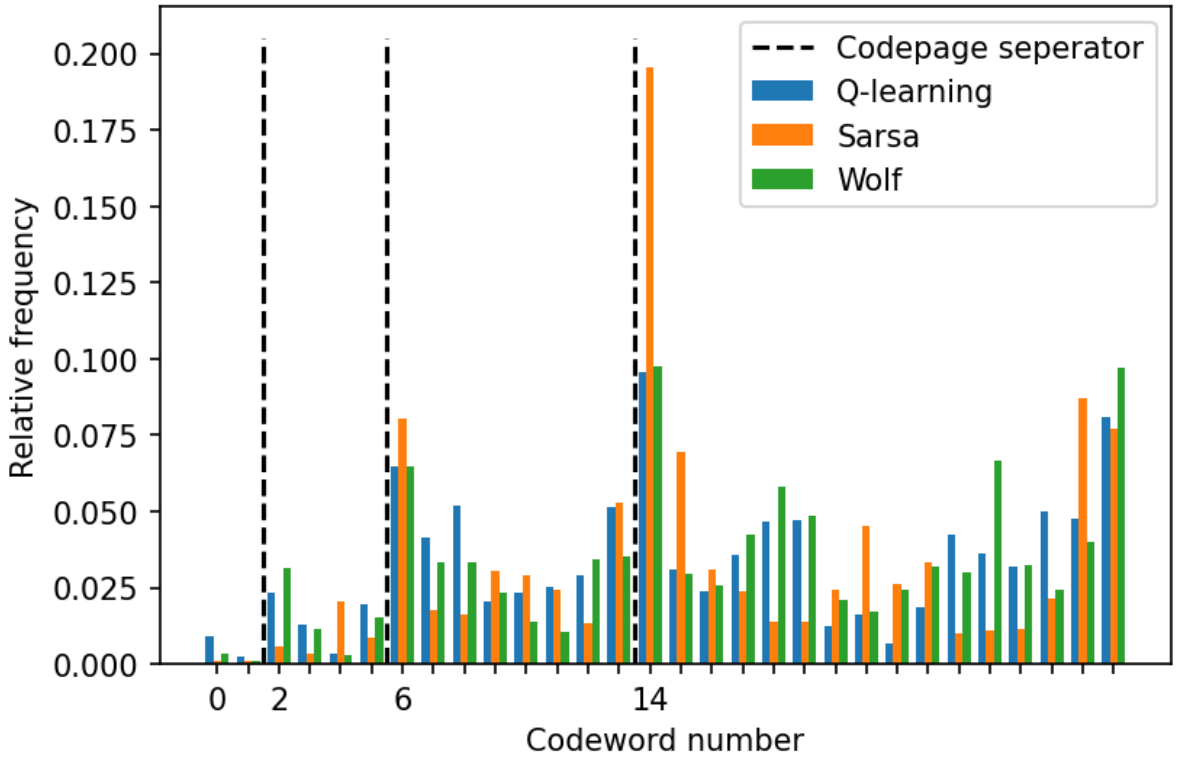


Figure E.12: Histogram of BS codewords chosen for multi agents with different update rules and the WoLf-PHC algorithm.

The distribution of the actions are almost indistinguishable between the methods. The beams chosen are very similar between Q-learning and WoLF-PHC which isn't surprising given their similarity in both their update rule and performance.

Impact of environment on agent(s) with adjacent actions F

F.1 purpose

In this test report agent(s) with adjacent action scheme will be tested in different environments to evaluate performance. The performance will be tested by simulating agents and recording performance metrics. Both the case of a single centralized agent, and of an agent at each terminal will be simulated. Evaluation and comparison will be based on the recorded metrics, and will focus on general performance. These tests are conducted to gain insight into how well the agent(s) perform in different circumstances. In order to compare the agents, the performance metric is defined in the next section.

F.2 Test metric(s)

The performance metrics used in this report are all based on the misalignment between the received signal power and the highest achievable signal power at each time step. This metric is used instead of the received signal power alone, as this value is expected to vary significantly as the agent moves around the environment, regardless of the choices made. This is due to phenomena like free space loss and absorption, and thus a low received signal power might not necessarily correspond to bad performance of the agent.

Thus the misalignment is used instead, which is calculated as the difference between the received signal power and maximum achievable signal power in dB scale. The maximum signal power is defined as the maximum found when looking at all possible combinations of codewords in the entire codebook and not just those available to the agent at the current time step.

$$M = S_m - S_{max}$$

where

M :	$[dB]$	Misalignment	$RM \in \mathcal{R}$
S_m :	$[dB]$	Received signal power	$S_m \in \mathcal{R}$
S_{max} :	$[dB]$	Maximum achievable signal power	$S_{max} \in \mathcal{R}$

Both the received and maximum power used for calculating the misalignment are without added noise, to better allow the metric to measure the quality of the choices made. This is based on

the notion that choices should be made based on the quality of the channel, which is assumed to be independent of the thermal noise in the equipment. It should be noted that this metric is only obtainable because the test is simulated, as normally the maximum achievable signal power is not directly available.

The metrics used to compare the different agent configurations are:

- Average misalignment
- Median misalignment
- 1st quartile misalignment
- 3rd quartile misalignment

F.3 Test procedure

The test setup is generally the same as for tuning of state parameters. The general setup is as follows:

Table F.1: Technical parameters used when running tests in different environments and scenarios.

Parameter	Value	Description
B	400 MHz	Total bandwidth
f_c	28 GHz	mmWave carrier frequency
P_{TX}	40 dBm	Transmission power
r_c	200 m	Cell radius
N_r	8	Number of receive antennas
N_t	16	Number of transmit antennas
M	10000	Number of episodes pr. simulation
N	7000	Number of steps pr. episode

A simulation consists of several episodes, with each episode consisting of one or more agents following a path, taking an action at each step. A track is chosen at random from a collection of 16 tracks at the beginning of each episode. Each track in the pedestrian collection is 20000 steps long, however the path used in each episode is only 7000 steps long. Thus a random starting point on the track is chosen as well at the beginning of each episode. By keeping the episodes shorter than the length of the tracks, it is possible to get more unique paths for the agent to follow in different episodes, along with a lot of paths with differing amounts of overlap. This means that it is not just the same 16 paths which are traversed over and over again in the simulation. 7000 steps was chosen as a fitting trade-off between path length and number of unique paths in total. The agent keeps the Q-table values between episodes, and thus learns across all episodes.

In order to ensure that the test metrics are statistically accurate, and to give the agent a chance to explore the entire state space, many episodes are simulated. Both the number of episodes and number of steps in each episode, are derived based on an intermediately sized state space. The total number of steps in the simulation should be enough for an agent to explore all state-action pair, to give accurate value estimates. The state given below, based on a single agent implementation, is assessed to be an intermediate state, and was used to derive the total number of steps.

- A history of the 2 previous beams at the UE
- A history of the 2 previous beams at the BS
- A distance between the UE and BS with a resolution of 4
- An angle between the UE and the BS with a resolution of 8

The size of the combined state-action space of this state was determined in Section 4.4.2 to be $6.05 \cdot 10^6$ state-action pairs. In order to allow proper exploration, the total number of steps for a simulation is set such that each state-action pair can be visited approximately ten times, resulting in a total step number of $70 \cdot 10^6$ steps. This gives a total of 10000 episodes in a simulation. While this allows an agents performance to converge, an even better performance measure would be an average over a statistically significant amount of agents. However, as each simulation takes approximately four hours to run, this is sadly not viable because of time and resource constraints. Thus the results presented in this chapter will be judged with this uncertainty in mind.

F.3.1 Variables

The algorithm consists of a number of variables, where every variable except the environment will be kept constant throughout this test to ensure valid comparisons. Two sets of variables will be presented, the first for the case of a single centralized agent, and the next for an agent at each terminal. The first set of constant variables are the following.

γ :	[−]	Forgetting factor	$\gamma = 0.7$
α :	[−]	Action value adjustment amplitude	$\alpha = 0.01$
ϵ *adaptive* :	[−]	Exploration rate	ϵ *adaptive* $= N/A$
$\mathcal{S}$:	[−]	State space	$\mathcal{S} =' 2-2-1-8-2-32'$
A :	[−]	Action space	$A =$ *adjacent*

γ is chosen as the best performing, from the tests described in Appendix C. α is chosen as the best performing, from the tests described in Appendix C. ϵ *adaptive* is chosen as it is the best performing ϵ method and it's weight value is chosen as the best performing, from the tests described in Appendix C. The state space is chosen as the best performing, from the tests described in A.

The second set of variables for an agent at each terminal is:

γ :	[−]	Forgetting factor	$\gamma = 0.6$
α :	[−]	Action value adjustment amplitude	$\alpha = 0.01$
ϵ *adaptive* :	[−]	Exploration rate	ϵ *adaptive* $= N/A$
$\mathcal{S}_r$:	[−]	State space of the UE	$\mathcal{S}_r =' 2-0-1-8-2-32'$
$\mathcal{S}_t$:	[−]	State space of the BS	$\mathcal{S}_t =' 0-2-0-0-2-32'$
A :	[−]	Action space	$A =$ *adjacent*

Note here that the agents at both terminals have the same hyper parameters. γ is chosen as the best performing, from the tests described in Appendix D. α is chosen as the best performing, from the tests described in Appendix D. ϵ *adaptive* is chosen as it is the best performing ϵ method and it's weight value is chosen as the best performing, from the tests described in Appendix D. The state space is chosen as the best performing, from the tests described in B.

F.3.2 Environments to be tested

A limited number of different environments will be tested. The environments are limited by the possibilities of the simulation tools. The environments that are being tested are

- Pedestrian in LOS conditions
- Pedestrian in NLOS conditions
- Car in urban environment in LOS conditions

An explanation of what pedestrian and car being in an urban environment means for the simulation can be found in Section 2.6 and Section 2.6.4. The differences between LOS and NLOS are explained in Section 2.5. These environments will be tested for both a single centralized agent, and for each terminal possessing their own agent.

F.4 Data processing

As mentioned in Appendix A.3 each test consists of 70.000.000 steps, however plotting the relative misalignment for all steps would not result in a graph indicative of the general performance of the state. To remove the variation within each episode, the misalignment is calculated based on averages. First the linear misalignment is calculated, and transformed to logarithmic scale.

$$MA = 10 \cdot log_{10}(R) - 10 \cdot log_{10}(R_{max})$$

This misalignment is then averaged across all steps within each episodes, resulting in a reduction from 70.000.000 data points to 10.000.

$$RM_{pe} = \frac{1}{7000} \cdot \sum_{i=1}^{7000} RM_i$$

where

RM_m	[−]	Average reward for a given episode
RM_i	[−]	Reward for a the i'th step in a given episode

In order to remove the influence of a specific track/path on the performance multiple episodes are averaged to get the final performance measure. The 10.000 episodes are averaged in ten intervals of 1000 episodes each.

F.5 Results/Conclusion

In the following section the results of the simulations of agents in different environments given in Appendix F.3.2 are presented. The agent settings has been tuned using a line-of-sight scenario, with pedestrian movement tracks. It is thus of interest how the agent performs when the environment changes to a non-line-of-sight scenario and if the performance seen in pedestrian movement tracks is replicated when car movement tracks are used instead. In Chapter 2 it is mentioned that one of the differences between the two movement models are that the car tracks

move over longer distances and has larger turn radii. Additionally, for the car the orientation of the UE is fixed to the direction of movement, where in pedestrian movement the orientation is more erratic.

F.5.1 Single centralized agent

Table F.2: Table of different misalignment measurements of a single centralized agent in different environments in the pedestrian case. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
LOS Heuristic	9.1 dB	6.5 dB	3.7 dB	10.8 dB
LOS Sarsa	11.3 dB	7.7 dB	2.6 dB	17.6 dB
NLOS Heuristic	24.6 dB	24.7 dB	16.3 dB	32.3 dB
NLOS Sarsa	24.1 dB	24.1 dB	15.8 dB	31.8 dB

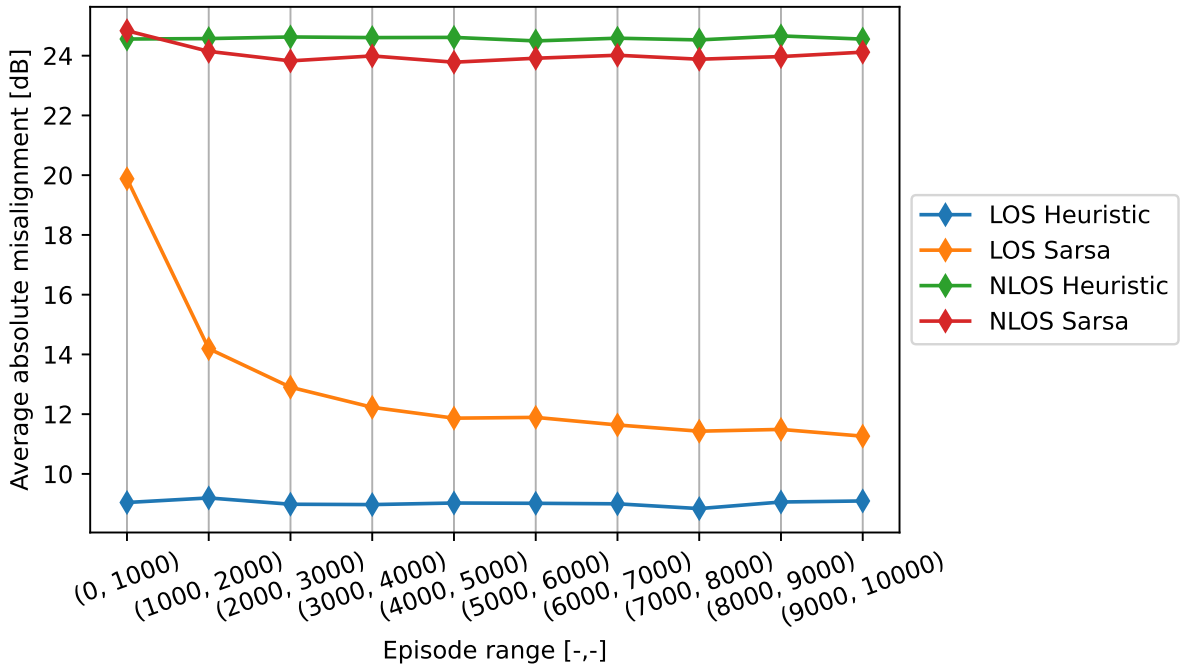


Figure F.1: Absolute misalignment of a single centralized agent in different environments in the pedestrian case.

In Fig. F.1 the performance of the optimized centralized Sarsa agent is shown, together with the heuristic for comparison. The performance of the two methods in the LOS scenario is significantly better, however this is expected. The signal strength should be less erratic and the best signal direction shouldn't change drastically over short periods in LOS compared to NLOS. The Sarsa performing worse in the NLOS scenario is also partially explained by the fact the state lends itself better to the LOS scenario rather than NLOS. Pieces of information that would be useful in the NLOS scenario like information about the geometry of the environment is considerably more costly to obtain and process.

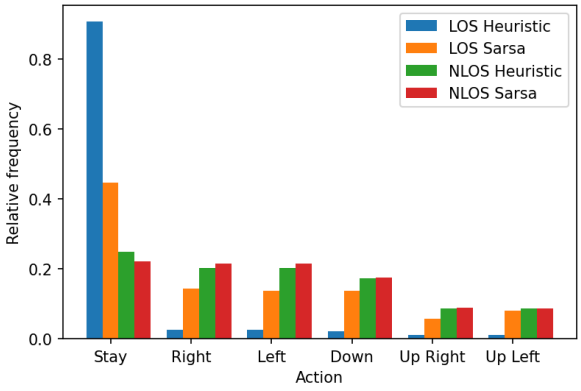


Figure F.2: Histogram of actions for a single centralized agent in different environments for UE in the pedestrian case

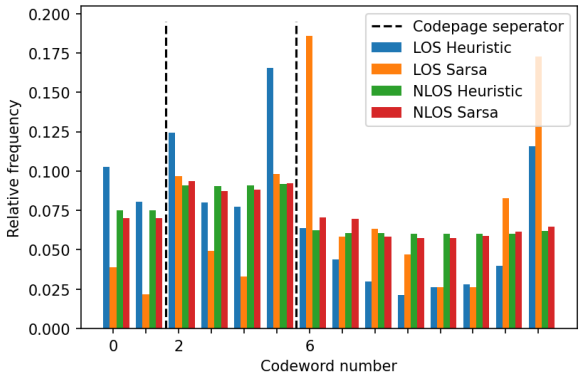


Figure F.3: Histogram of UE codewords chosen for a single centralized agent in different environments in the pedestrian case

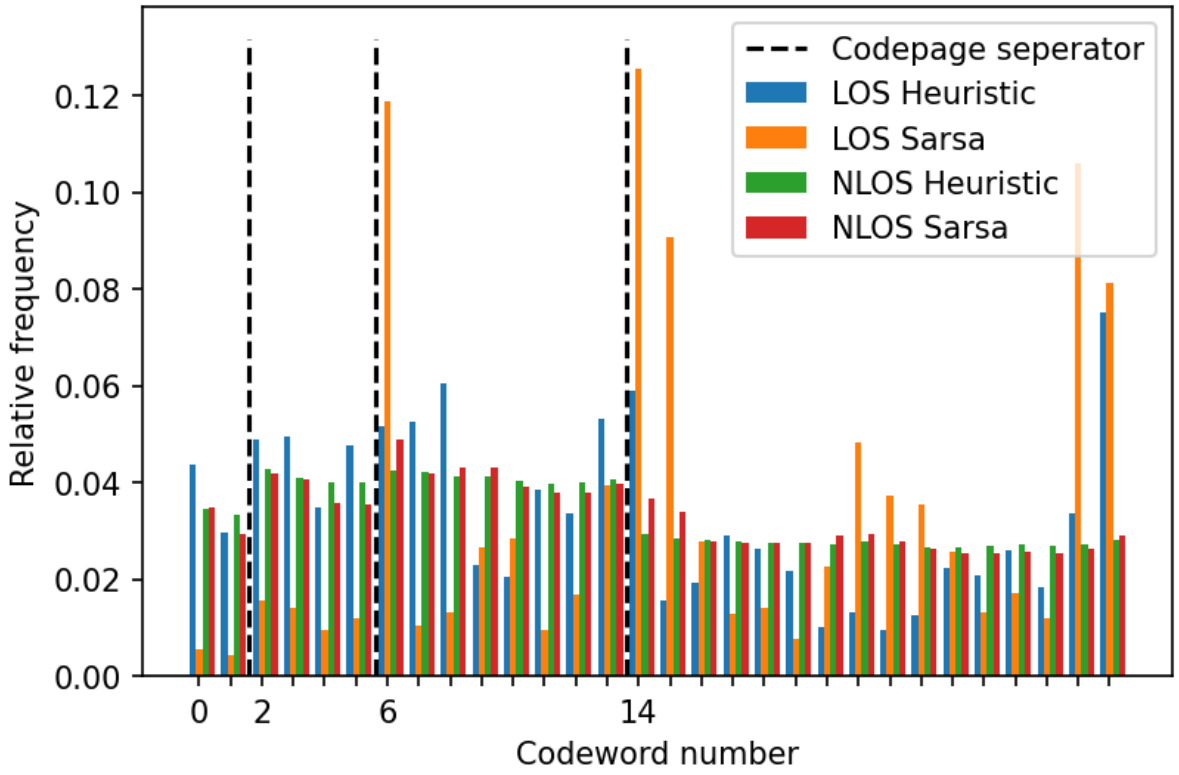


Figure F.4: Histogram of BS codewords chosen for a single centralized agent in different environments in the pedestrian case

Seen on Fig. F.2 through Fig. F.4 is the actions taken, and codewords used for the LOS and NLOS scenario for Sarsa and the heuristic algorithm. In contrast with the LOS scenario the actions taken in the NLOS scenarios are more evenly distributed for both the heuristic algorithm and Sarsa. It's sensible the action stay would be less useful as the NLOS scenario is more dynamic, which likely would result in good codeword choices to shift around more. The codewords used in NLOS drastically different, as here neither algorithm is inclined to used codewords close to the edge of the codepages. All codewords in any given codepage seems chosen equally often. One would still expect the codewords at the edges of codepages to be more valuable by virtue

of their larger coverage, this might still be the case as these choices aren't optimal as seen on Fig. F.1.

Table F.3: Table of different misalignment measurements of a single centralized agent in different environments in the car_urban case. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
LOS Heuristic	8.0 dB	6.1 dB	3.0 dB	10.7 dB
LOS Sarsa	7.9 dB	5.2 dB	1.5 dB	11.5 dB
NLOS Heuristic	23.6 dB	23.4 dB	14.8 dB	31.9 dB
NLOS Sarsa	23.1 dB	22.8 dB	14.0 dB	31.4 dB

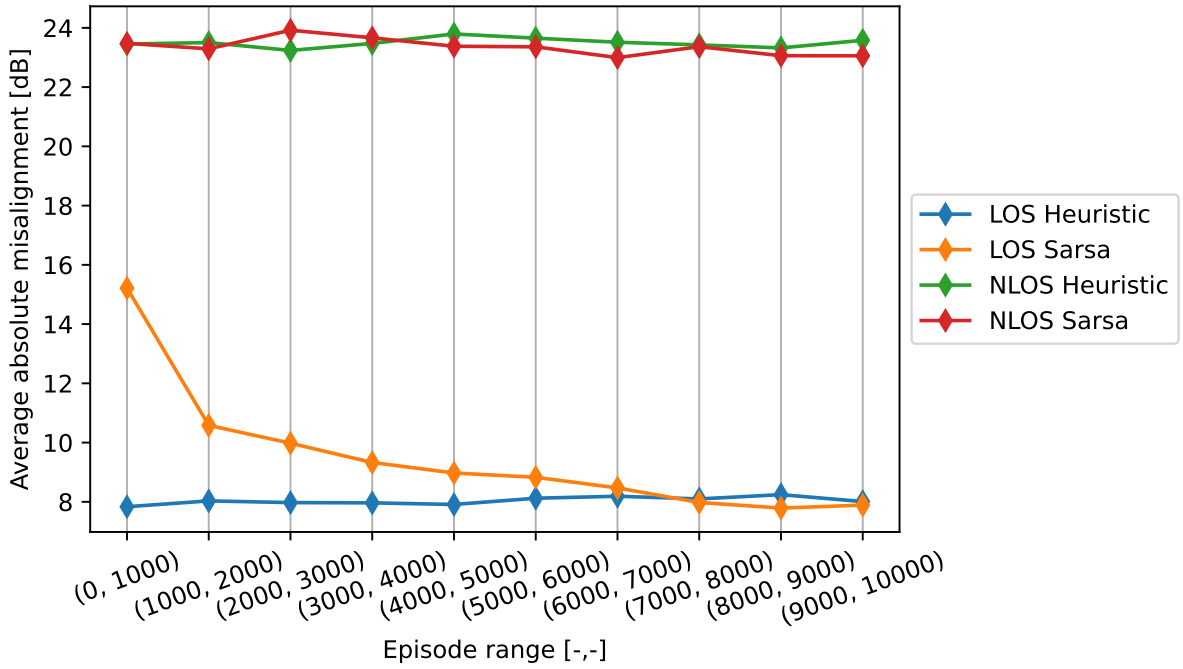


Figure F.5: Absolute misalignment of a single centralized agent in different environments in the car_urabn case for.

In Fig. F.5 the performance of the optimized centralized Sarsa agent is shown, together with the heuristic for comparison. The performance in the car_urban scenario is similar to those of pedestrian across the board, with a small improvement. This isn't unexpected as both the pedestrian and car_urban case are somewhat similar. The small performance improvement likely comes from the fact that the car_urban case has a less random orientation, because it's simply the moving direction of the vehicle. This has also resulted in the Sarsa agent reaching similar performance as the heuristic, which means the agent was capable of exploiting the simpler environment more than the heuristic algorithm.

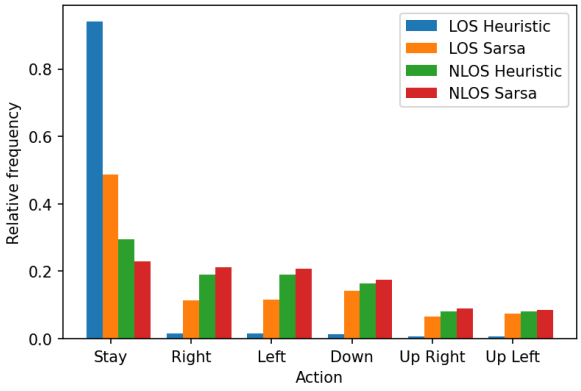


Figure F.6: Histogram of actions for a single centralized agent in different environments in the car_urban case for the UE

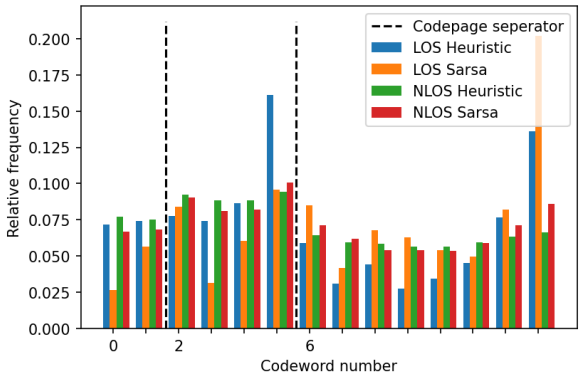


Figure F.7: Histogram of UE codewords chosen for a single centralized agent in different environments in the car_urban case.

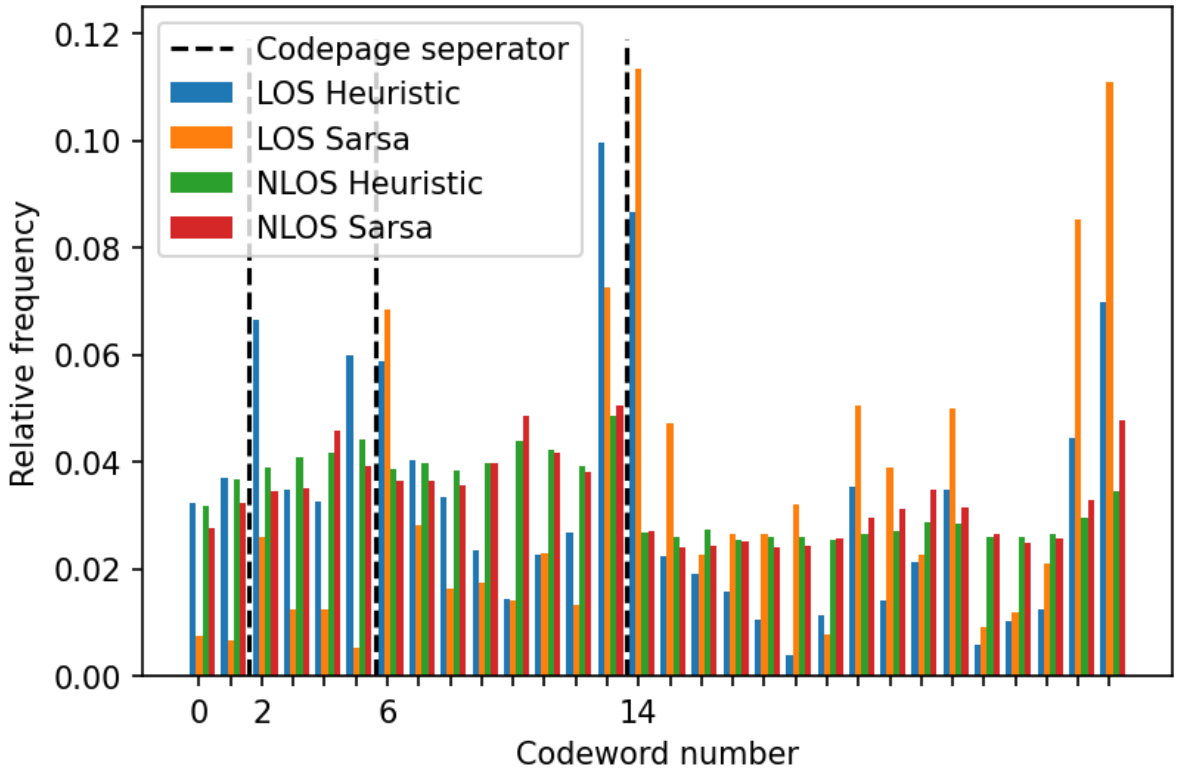


Figure F.8: Histogram of BS codewords chosen for a single centralized agent in different environments in the car_urban case.

The actions, see in Fig. F.6, largely resemble those for the pedestrian case, which isn't unexpected, the cases similarity have already been remarked on. The codeword choices for the NLOS scenario are close to being equally distributed, with a slight preference toward the edges of codepages. For the BS some codewords close to the middle of codepages have been used more often, why this is the case is unknown. Again the performance isn't impressive, so any information interpreted from the graphs can only be used as example of what not to do.

F.5.2 Multi agent

Table F.4: Table of different misalignment measurements of multi agents in different environments for the pedestrian case. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
LOS Heuristic	9.1 dB	6.5 dB	3.7 dB	10.8 dB
LOS Sarsa	10.6 dB	7.2 dB	2.5 dB	16.7 dB
NLOS Heuristic	24.6 dB	23.4 dB	16.3 dB	32.3 dB
NLOS Sarsa	17.7 dB	10.5 dB	9.4 dB	24.8 dB

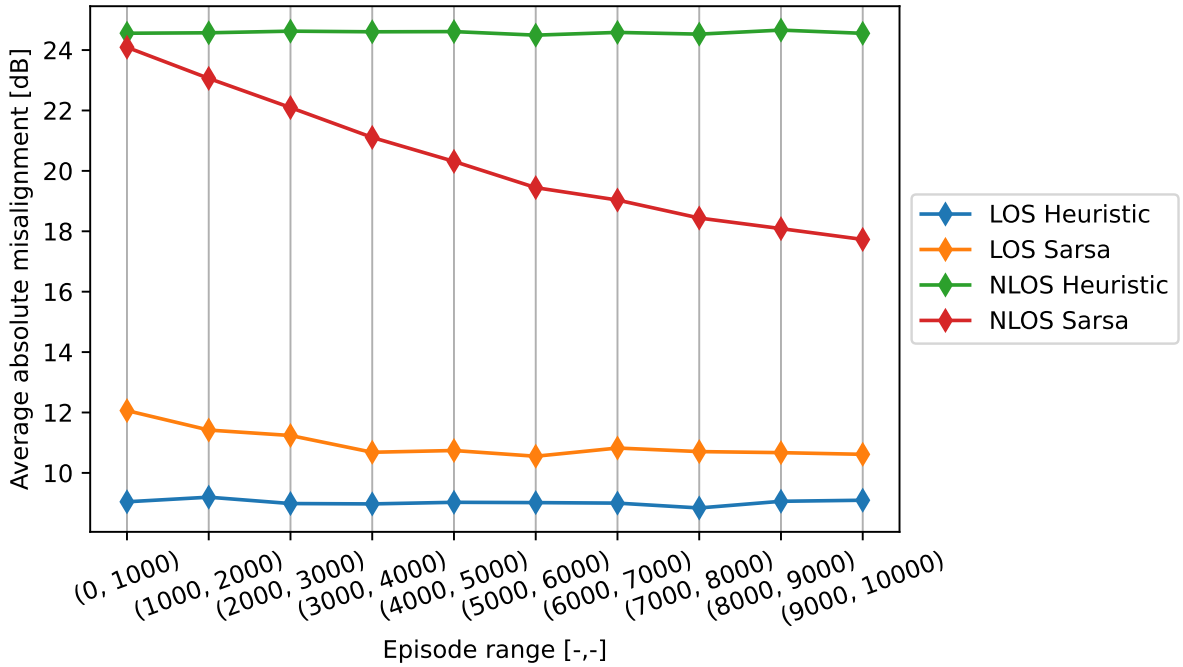


Figure F.9: Absolute misalignment of multi agents in different environments for the pedestrian case.

In Fig. F.9 the performance of the optimized Sarsa multi agents is shown, together with the heuristic for comparison. Unlike in the single agent case, learning provides large improvement in performance for the NLOS scenario. While learning naturally would be slower in the single agent case because of the Q-tables' size, it's performance seemed like it has converged. The fact that multi agent not only beats it in NLOS but does so, convincingly is interesting as the single agent would have an easier time coordinating a pair of beams along a strong reflected path. Regardless, the multi agent case outperforms the heuristic algorithm by quiet a wide margin in NLOS.

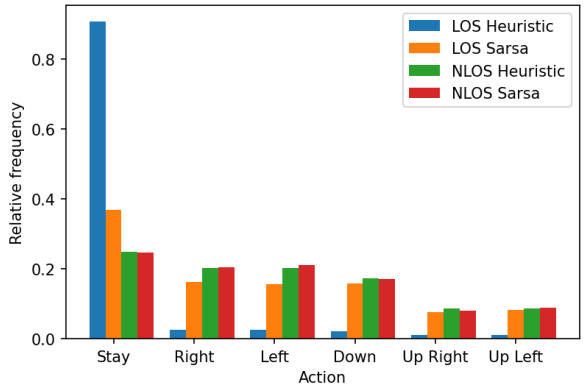


Figure F.10: Histogram of actions for multi agents in different environments for the pedestrian case, for the UE

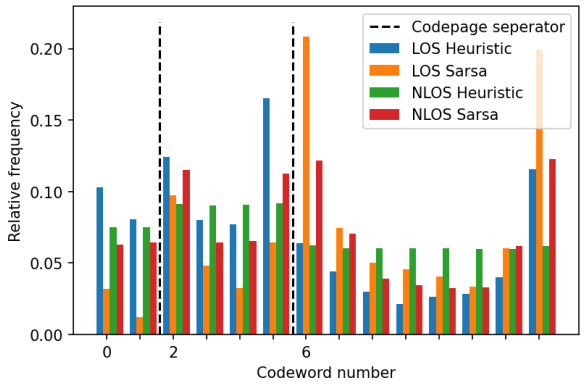


Figure F.11: Histogram of UE codewords chosen for multi agents in different environments for the pedestrian case.

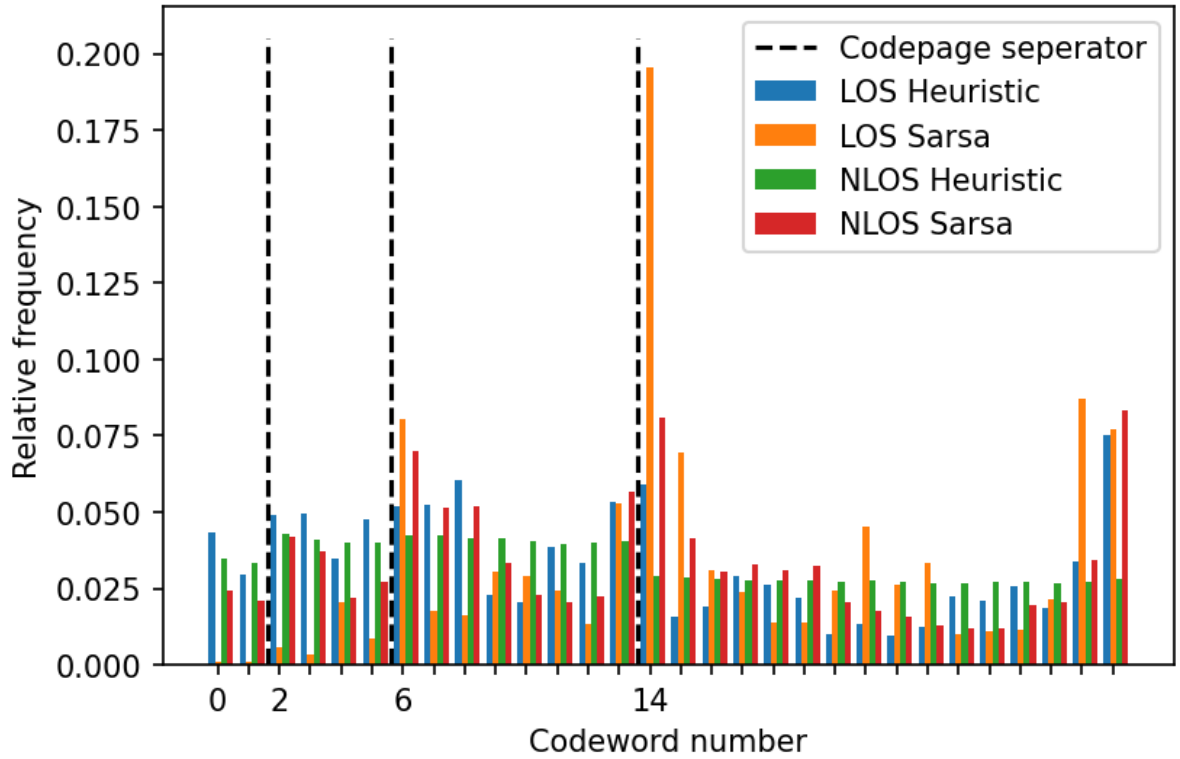


Figure F.12: Histogram of BS codewords chosen for multi agents in different environments for the pedestrian case

Like in the single agent case the choices for NLOS, as reflected in Fig. F.10, are more evenly distributed. This means an equal action distribution is not exclusively a trait of bad performance, at least not in the NLOS scenario. When inspecting the codeword distributions behavior for Sarsa in NLOS it's much close to that of LOS where the edges of codepages are preferred. This would support having this codeword distribution correlates with good performance. This behavior is still slightly less pronounced for NLOS than LOS, however the performance is also worse in NLOS. The slight preference for codewords in the lower half of codepages is interesting, and might point toward trends in the dataset, however the actual cause is unknown.

Table F.5: Table of different misalignment measurements of multi agents in different environments for the car_urban case. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
LOS Heuristic	8.0 dB	6.1 dB	3.0 dB	10.7 dB
LOS Sarsa	7.9 dB	4.8 dB	1.6 dB	11.5 dB
NLOS Heuristic	23.6 dB	23.4 dB	14.8 dB	31.9 dB
NLOS Sarsa	12.7 dB	10.5 dB	4.0 dB	19.5 dB

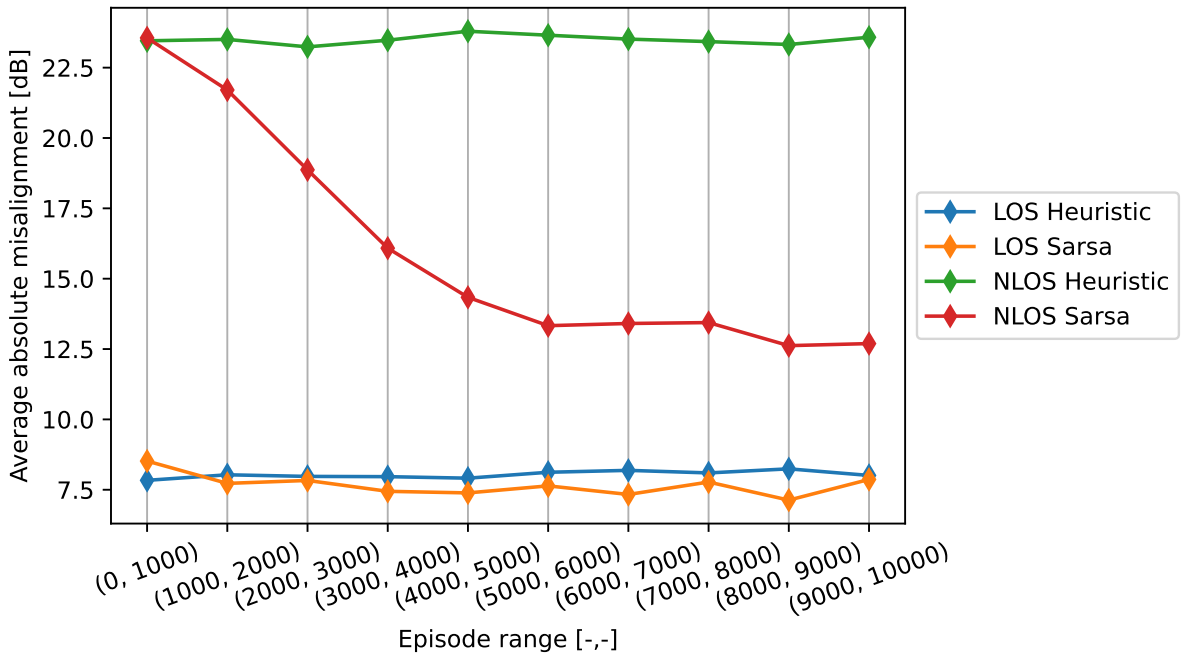


Figure F.13: Absolute misalignment of multi agents in different environments for the car_urban case

In Fig. F.13 the performance of the optimized Sarsa multi agents is shown, together with the heuristic for comparison. This results is not unexpected based on the pedestrian multi agent and car_urban single agent results. This performance closely mirrors that of the pedestrian case, although slightly better, with the exception of the NLOS Sarsa which is significantly better. The less challenging environment of the car_urban case is likely the cause.

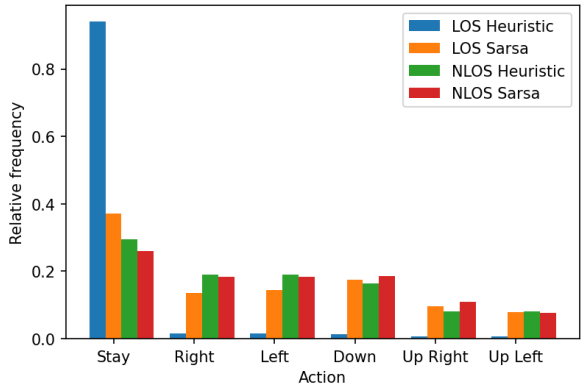


Figure F.14: Histogram of actions for multi agents in different environments for the car_urban case for UE

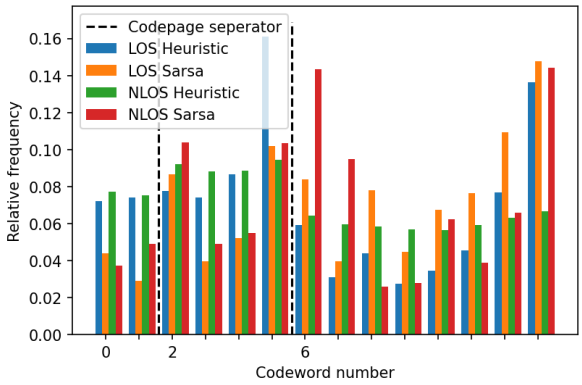


Figure F.15: Histogram of UE codewords chosen for multi agents in different environments for the car_urban case.

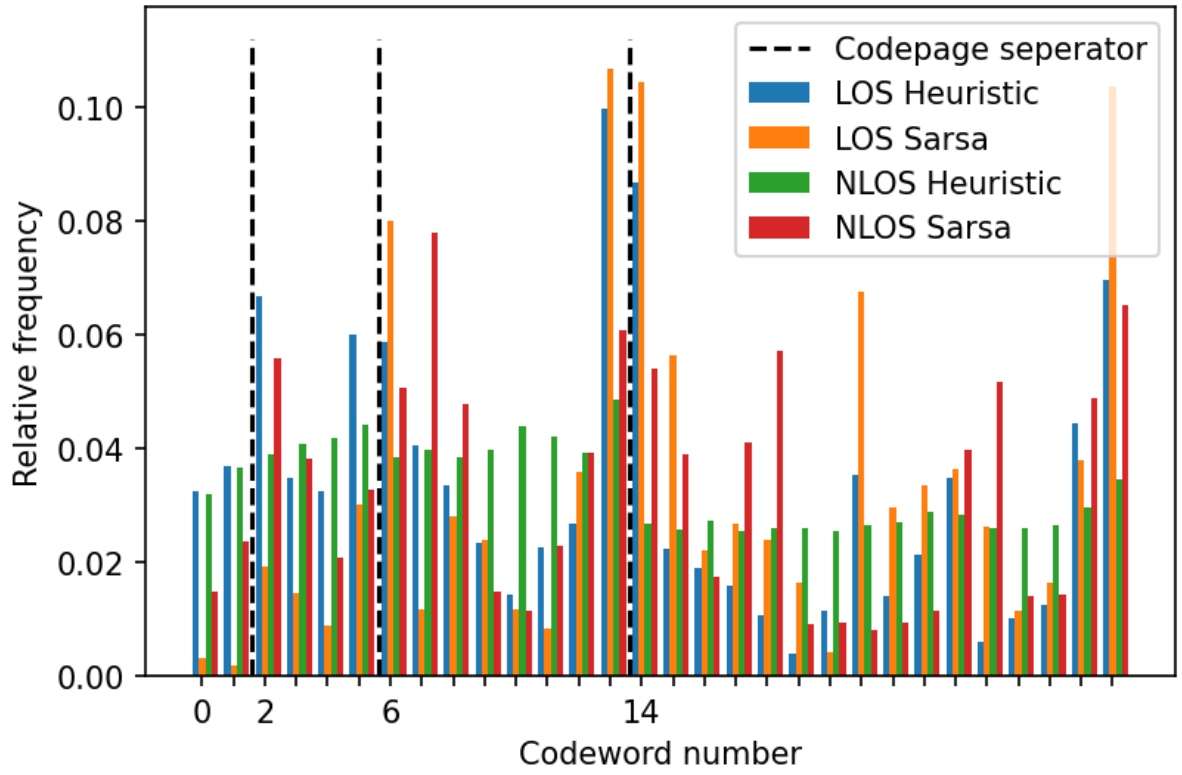


Figure F.16: Histogram of BS codewords chosen for multi agents in different environments for the car_urban case.

The actions, in Fig. F.14, resemble those of the pedestrian case. The behavior for Sarsa in car_urban NLOS posses the same preference for the edges of codepages, especially for the UE in Fig. F.15. The behavior for the BS is slightly different as it also prefers a couple of codewords somewhere between the edges and middle of the codepages. While previously unseen behavior it's evidently very efficient in the NLOS scenarios in comparison to both the heuristic and single agents behavior.

Impact of noise on agent(s) with adjacent actions

G

G.1 purpose

In this test report agent(s) with adjacent action scheme will be tested with differing amount of noise added during the simulation. The performance will be tested by simulating agents and recording performance metrics. Both the case of a single centralized agent, and of an agent at each terminal will be simulated. Evaluation and comparison will be based on the recorded metrics, and will focus on general performance. These tests are conducted to gain insight into how well the agent(s) perform when influenced by differing levels of noise. In order to compare the agents, the performance metrics are defined in the next section.

G.2 Test metric(s)

The performance metrics used in this report are all based on the misalignment between the received signal power and the highest achievable signal power at each time step. This metric is used instead of the received signal power alone, as this value is expected to vary significantly as the agent moves around the environment, regardless of the choices made. This is due to phenomena like free space loss and absorption, and thus a low received signal power might not necessarily correspond to bad performance of the agent.

Thus the misalignment is used instead, which is calculated as the difference between the received signal power and maximum achievable signal power in dB scale. The maximum signal power is defined as the maximum found when looking at all possible combinations of codewords in the entire codebook and not just those available to the agent at the current time step.

$$M = S_m - S_{max}$$

where

M :	$[dB]$	Misalignment	$RM \in \mathcal{R}$
S_m :	$[dB]$	Received signal power	$S_m \in \mathcal{R}$
S_{max} :	$[dB]$	Maximum achievable signal power	$S_{max} \in \mathcal{R}$

Both the received and maximum power used for calculating the misalignment are without added noise, to better allow the metric to measure the quality of the choices made. This is based on the notion that choices should be made based on the quality of the channel, which is assumed to be independent of the thermal noise in the equipment. It should be noted that this metric is

only obtainable because the test is simulated, as normally the maximum achievable signal power is not directly available.

The metrics used to compare the different agent configurations are:

- Average misalignment
- Median misalignment
- 1^{st} quartile misalignment
- 3^{rd} quartile misalignment

G.3 Test procedure

The test setup is generally the same as for tuning of state parameters. The general setup is as follows:

Table G.1: Technical parameters used when running tests with different noise amounts.

Parameter	Value	Description
B	400 MHz	Total bandwidth
f_c	28 GHz	mmWave carrier frequency
P_{TX}	40 dBm	Transmission power
P_{TN}	-78 dBm	Thermal noise power
P_{NF}	9 dB	Noise figure power
r_c	200 m	Cell radius
N_r	8	Number of receive antennas
N_t	16	Number of transmit antennas
M	10000	Number of episodes pr. simulation
N	7000	Number of steps pr. episode

A simulation consists of several episodes, with each episode consisting of one or more agents following a path, taking an action at each step. A track is chosen at random from a collection of 16 tracks at the beginning of each episode. Each track in the pedestrian collection is 20000 steps long, however the path used in each episode is only 7000 steps long. Thus a random starting point on the track is chosen as well at the beginning of each episode. By keeping the episodes shorter than the length of the tracks, it is possible to get more unique paths for the agent to follow in different episodes, along with a lot of paths with differing amounts of overlap. This means that it is not just the same 16 paths which are traversed over and over again in the simulation. 7000 steps was chosen as a fitting trade-off between path length and number of unique paths in total. The agent keeps the Q-table values between episodes, and thus learns across all episodes.

In order to ensure that the test metrics are statistically accurate, and to give the agent a chance to explore the entire state space, many episodes are simulated. Both the number of episodes and number of steps in each episode, are derived based on an intermediately sized state space. The total number of steps in the simulation should be enough for an agent to explore all state-action pair, to give accurate value estimates. The state given below, based on a single agent implementation, is assessed to be an intermediate state, and was used to derive the total number of steps.

- A history of the 2 previous beams at the UE
- A history of the 2 previous beams at the BS
- A distance between the UE and BS with a resolution of 4
- An angle between the UE and the BS with a resolution of 8

The size of the combined state-action space of this state was determined in Section 4.4.2 to be $6.05 \cdot 10^6$ state-action pairs. In order to allow proper exploration, the total number of steps for a simulation is set such that each state-action pair can be visited approximately ten times, resulting in a total step number of $70 \cdot 10^6$ steps. This gives a total of 10000 episodes in a simulation. While this allows an agents performance to converge, an even better performance measure would be an average over a statistically significant amount of agents. However, as each simulation takes approximately four hours to run, this is sadly not viable because of time and resource constraints. Thus the results presented in this chapter will be judged with this uncertainty in mind.

G.3.1 Variables

The algorithm consisst of a number of variables, where every variable expect the noise level will be kept constant throughout this test to ensure valid comparison. Two sets of variables will be presented, the first for the case of a single centralized agent, and the next for an agent at each terminal. The first set of constant variables are the following:

γ :	[−]	Forgetting factor	$\gamma = 0.7$
α :	[−]	Action value adjustment amplitude	$\alpha = 0.01$
ϵ *adaptive* :	[−]	Exploration rate	ϵ *adaptive* $= N/A$
$\mathcal{S}$:	[−]	State space	$\mathcal{S} =' 2-2-1-8-2-32'$
A :	[−]	Action space	$A =$ *adjacent*
case	[−]	The mobility case	*case* = *pedestrian*
scenario	[−]	The LOS status	*scenario* = *LOS*

γ is chosen as the best performing, from the tests described in Appendix C. α is chosen as the best performing, from the tests described in Appendix C. ϵ *adaptive* is chosen as it is the best performing ϵ method and it's weight value is chosen as the best performing, from the tests described in Appendix C. The state space is chosen as the best performing, from the tests described in A.

The second set of variables for an agent at each terminal is:

γ :	[−]	Forgetting factor	$\gamma = 0.6$
α :	[−]	Action value adjustment amplitude	$\alpha = 0.01$
ϵ *adaptive* :	[−]	Exploration rate	ϵ *adaptive* $= N/A$
$\mathcal{S}_r$:	[−]	State space of the UE	$\mathcal{S}_r =' 2-0-1-8-2-32'$
$\mathcal{S}_t$:	[−]	State space of the BS	$\mathcal{S}_t =' 0-2-0-0-2-32'$
A :	[−]	Action space	$A =$ *adjacent*
case	[−]	The mobility case	*case* = *pedestrian*
scenario	[−]	The LOS status	*scenario* = *LOS*

Note here that the agents at both terminals have the same hyper parameters. γ is chosen as the best performing, from the tests described in Appendix D. α is chosen as the best performing,

from the tests described in Appendix D. ϵ *adaptive* is chosen as it is the best performing ϵ method and it's weight value is chosen as the best performing, from the tests described in Appendix D. The state space is chosen as the best performing, from the tests described in B.

G.3.2 Noise levels to be tested

As mentioned in Chapter 4, only the thermal noise and the noise factor from the circuits at the transceivers are considered. The noise is modelled as additive white circular Gaussian noise:

$$n \sim \mathcal{CN}(0, P_n)\,, \tag{G.1}$$

where
P_n is the noise power. As the noise is considered to be thermal noise, the noise power can calculated as:

$$P_n = k_\mathrm{B} \cdot T \cdot B\,, \tag{G.2}$$

where

k_B :	$[J/K]$	Boltzmann constant
T :	$[K]$	Temperature
B :	$[Hz]$	Bandwidth

Under normal room temperature the temperature is 290 K, and the bandwidth used for mmWave systems is here assumed to be 400 MHz, which results in a noise power of approximately -88 dBm.

In addition to the thermal noise, which can't be changed, it is also interesting to investigate the performance if using less ideal equipment that add further noise due to its noise figure. The noise figure is based on the assumptions made in the 3GPP channel model specifications, of 9 dB [16]. However, as the previous tests were all performed with a transmission power of 40 dBm, which is 10 dBm higher than in usual mmWave scenarios, an additional 10 dBm is added to the noise in both cases.

G.4 Data processing

As mentioned in Appendix A.3 each test consists of 70.000.000 steps, however plotting the relative misalignment for all steps would not result in a graph indicative of the general performance of the state. To remove the variation within each episode, the misalignment is calculated based on averages. First the linear misalignment is calculated, and transformed to logarithmic scale.

$$MA = 10 \cdot log_{10}(R) - 10 \cdot log_{10}(R_{max})$$

This misalignment is then averaged across all steps within each episodes, resulting in a reduction from 70.000.000 data points to 10.000.

$$RM_{pe} = \frac{1}{7000} \cdot \sum_{i=1}^{7000} RM_i$$

where

RM_m	[–]	Average reward for a given episode
RM_i	[–]	Reward for a the i'th step in a given episode

In order to remove the influence of a specific track/path on the performance multiple episodes are averaged to get the final performance measure. The 10.000 episodes are averaged in ten intervals of 1000 episodes each.

G.5 Results/Conclusion

In the following section the results of the simulations of agents in different environments given in Appendix F.3.2 are presented.

Table G.2: Table of different misalignment measurements of multi agents with different update rules. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Heuristic: No noise	9.1 dB	6.5 dB	3.7 dB	10.8 dB
Heuristic: Thermal noise	9.1 dB	6.6 dB	3.7 dB	10.7 dB
Heuristic: Noise factor	8.9 dB	6.5 dB	3.6 dB	10.7 dB
Sarsa: No noise	11.3 dB	7.7 dB	2.6 dB	17.6 dB
Sarsa: Thermal noise	10.2 dB	7.4 dB	3.5 dB	14.6 dB
Sarsa: Noise factor	9.9 dB	6.9 dB	3.2 dB	14.2 dB

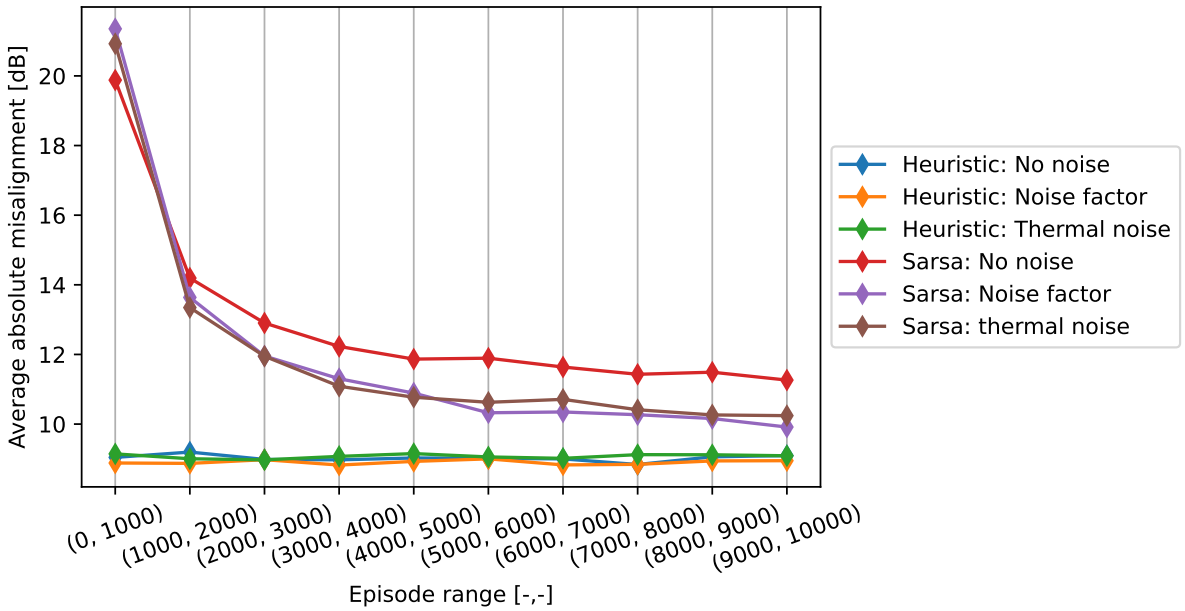


Figure G.1: Performance of a single centralized agent in a noisy environment, compared with a heuristic algorithm

In Fig. G.1 curves for the performance of a single centralized agent in a noisy environment is plotted. From this it's clear that the heuristic algorithm suffers no degradation of performance

in noisy environments, at least for the LOS scenario. The agents performance actually improves when noise is added which unexpected. Noise would be likely to make the value estimates worse which would normally result in worse performance. This difference could be because of the variations within each test, however the fact that the two noise tests produce practically identical results would imply otherwise. The exact reason is unknown. The fact that the performance is equal between the two noise levels is unexpected. A possible explanation is that noise of this level only meaningfully affect choice with already poor signal power. If the performance of the agent in general results in high signal power then the noise would only affect the small amount of actions in which the signal power is low, and as such would only result in a slight degradation of performance.

Table G.3: Table of different misalignment measurements of multi agents with different update rules. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Heuristic: No noise	9.1 dB	6.5 dB	3.7 dB	10.8 dB
Heuristic: Thermal noise	9.1 dB	6.6 dB	3.7 dB	10.7 dB
Heuristic: Noise factor	8.9 dB	6.5 dB	3.6 dB	10.7 dB
Sarsa: No noise	10.6 dB	7.2 dB	2.5 dB	16.7 dB
Sarsa: Thermal noise	10.7 dB	7.4 dB	3.2 dB	16.3 dB
Sarsa: Noise factor	10.7 dB	7.7 dB	3.1 dB	16.2 dB

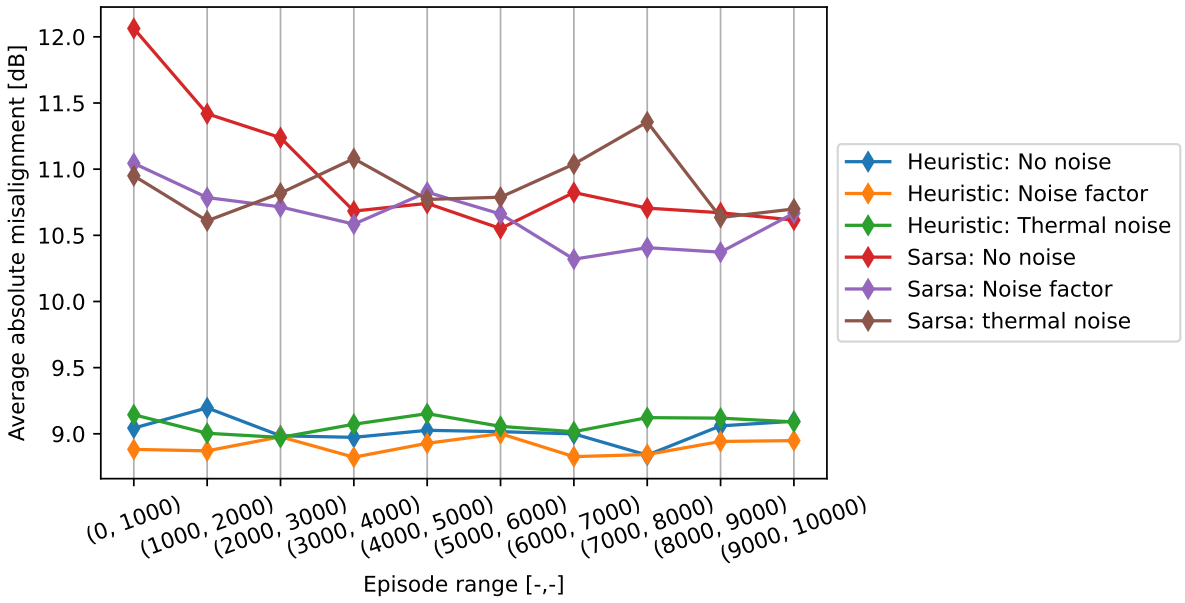


Figure G.2: Performance of multi agents in a noisy environment, compared with a heuristic algorithm

In Fig. G.2 in can be seen the noise affects neither the heuristic algorithm nor the Sarsa agent. While unexpected this is still more reasonable than for the single agent case where noise improved performance. The step size could be so low to the point where the levels of noise added does not matter, which would mean the agent is resilient to noise.

Work sheets: 12-05-2022 H

H.1 Determining the parameters for adaptive epsilon algorithm

Following the algorithm in [25] we wanted to find values for the parameter δ and the inverse sensitivity σ. In the paper they recommend a δ equal to the inverse of the number of actions possible at a given state, which we approximate as $1/5$ as 5 is the expected amount of choices for the agent.

For the value of σ we have tried to base our analysis on investigation of a plot similar to the one from the paper, seen below.

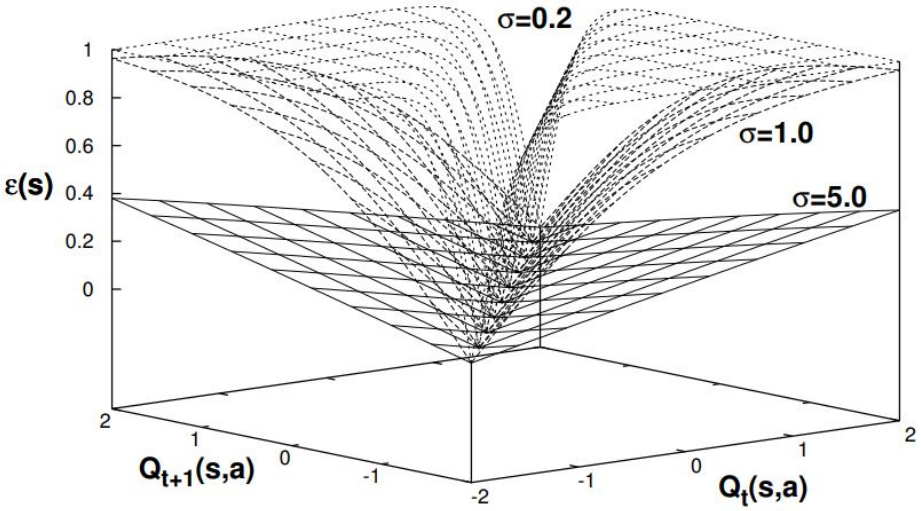


Figure H.1: ϵ_{adj} as a function of two consecutive Q values

In our case we thought it would make a bit more sense to just plot it in 2D with the x-axis being the absolute value of the TD-error and the y-axis the value of equation (6) from [25]. We removed the state and action from the equation, by just looking at the TD-error and then finding out how this is distributed in general, independent of state and action.

We thus ran a small test where we logged the TD-error for each step for 2000 episodes and then looked at how it was distributed. Both in the beginning and in the final episodes, to see if the values change a lot after some training.

In the plots below are some cut-outs of some KDE-plots showing the distribution of $|TD-error|$. It should be noted that TD-errors of upwards of 0.3 are observed as well, but this is so rare that we chose only to look in the intervals shown in the plots. We note as well, that the distribution

of the TD-error seems to be a bit more flat after training, but we still see a definite peak at 0.0003.

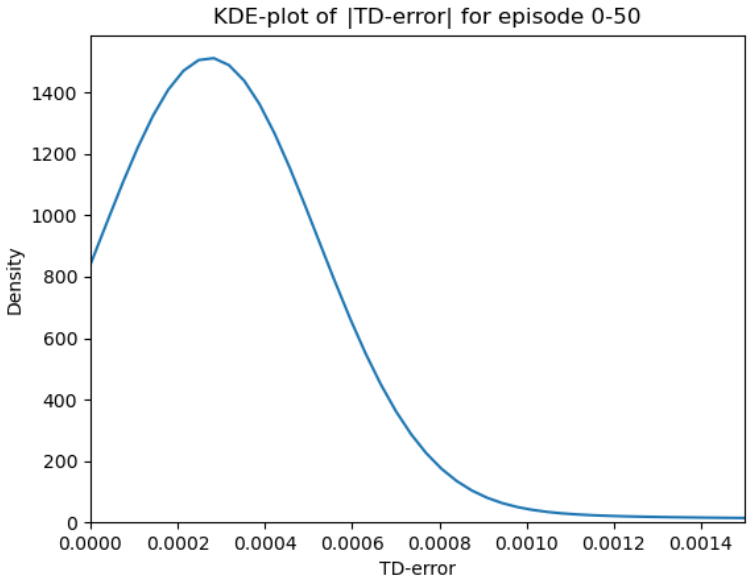


Figure H.2: KDE-plot of $|TD - error|$ for the first 50 episodes [0-50]

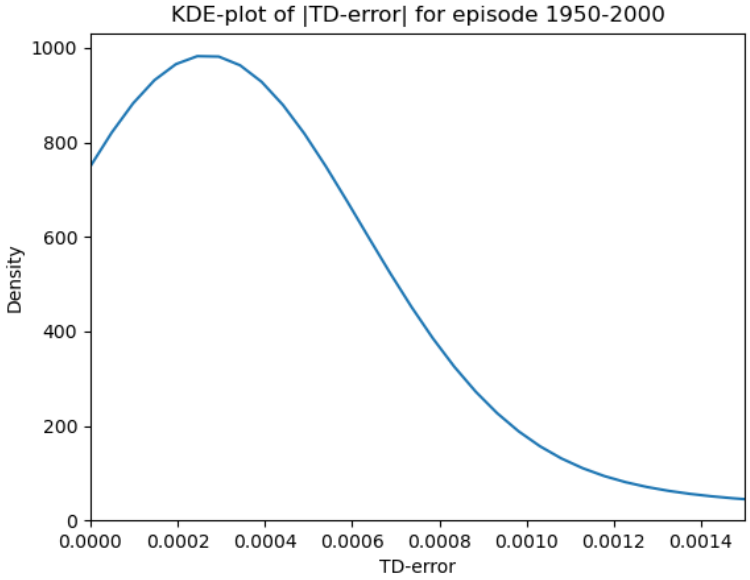


Figure H.3: KDE-plot of $|TD - error|$ for the last 50 episodes [1950-2000]

From the first plot we see that most of the time the TD-error is around 0.0003 and very rarely above 0.001. Thus we thought to look at the plot for $f(TD\_Error, \sigma)$, which we will call the "exploration indicator" for now, in this region, with the goal of having high exploration when above 0.001 and lower when closer to the mode.

The level of exploration we want, we base on our initial results from the test where we tried a constant epsilon value and the test where we use a exponential decaying epsilon value. From

these tests we found that a constant epsilon of around $[0.001 - 0.01]$ gives faster learning and better end results, than a decaying epsilon value. The results from these two tests are shown in figures Fig. H.4 and Fig. H.5.

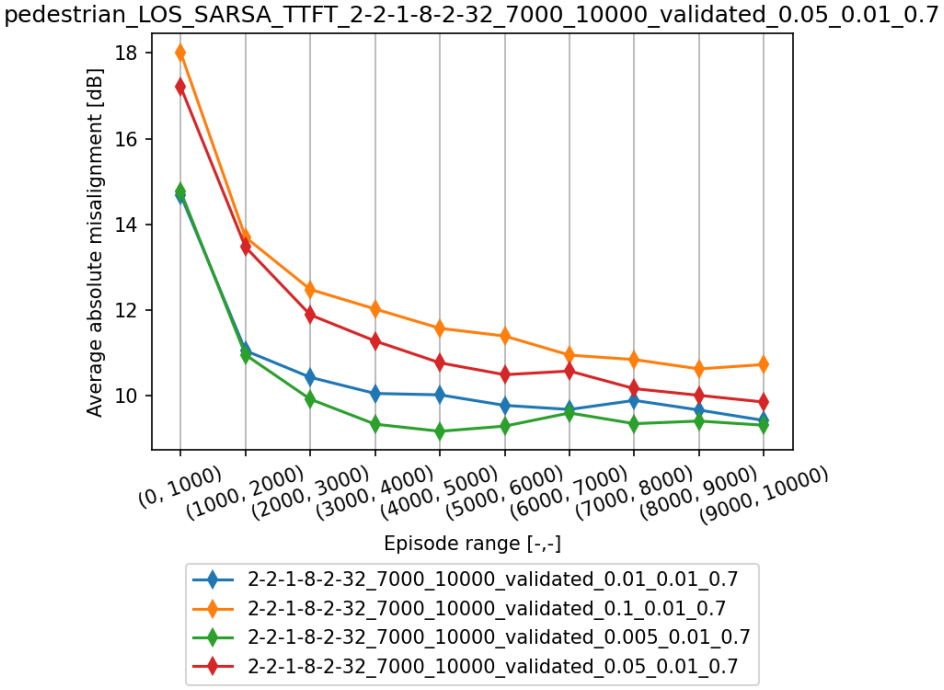


Figure H.4: constant epsilon values

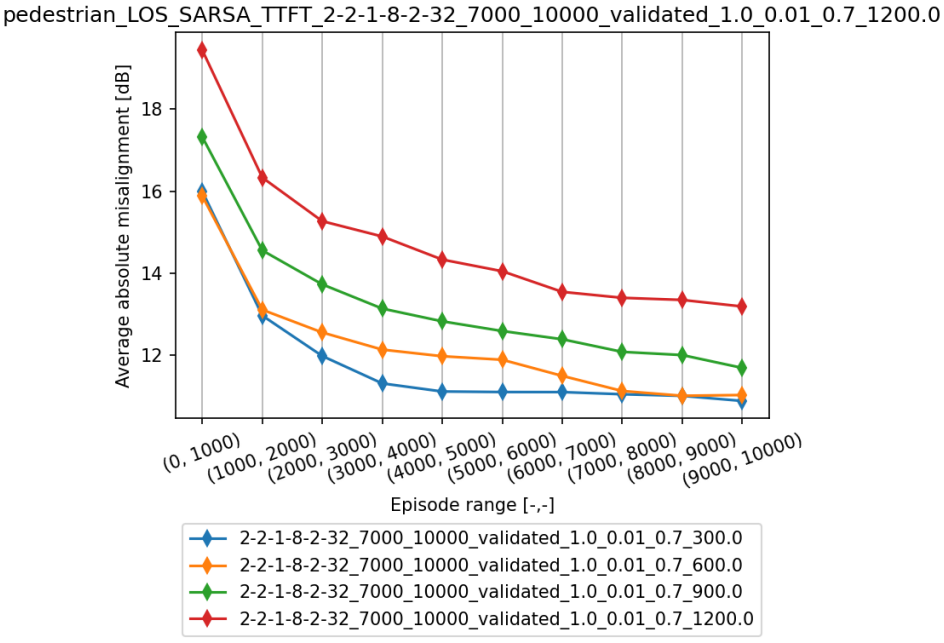


Figure H.5: decaying epsilon values

Thus it seems like a too small epsilon value is not desired, but it might be possible to augment the performance of the constant epsilon by exploring a bit more some times. We then think that a good point of reference for our tests with adaptive epsilon would be to have $f(TD\_Error)$

be around 0.005 at the mode, as this should be similar to having our constant epsilon, when a state has a good estimate and has been visited many times. We can then shift the σ value a bit up and down and compare the results.

H.2 Developing our own version with a sigmoid function

Now, it has proven a bit hard to have an exploration indicator of 0.005 at a TD-error of 0.0003 but also high at a TD-error of 0.001, by using the equation in [25]. An example of how $f(TD\_Error)$ looks like at these regions is shown in Fig. H.6 and Fig. H.7 as the red line. We would like that the exploration indicator is at least above 0.2 when it is to be deemed "high", although we have no good argumentation for this.

Figure H.6: Plot of the exploration indicator in the range around the edge of the KDE-plot. Red plot is function from [25], blue is Sigmoid.

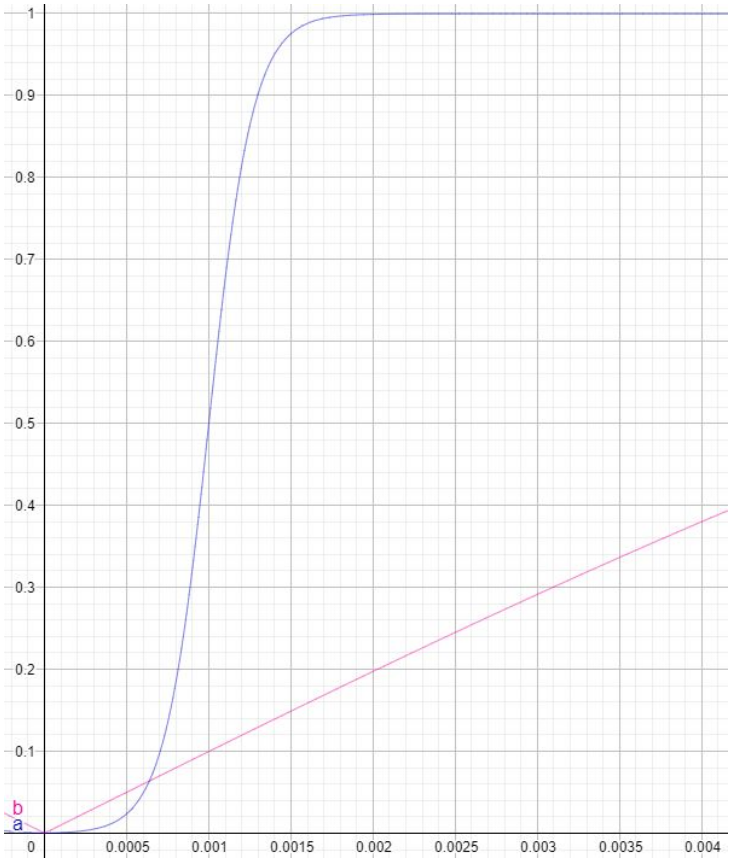


Figure H.7: Plot of the exploration indicator in the range around the mode. Red plot is function from [25], blue is Sigmoid.

After tweaking the σ value for some time, we thought of something else that might get us closer to what we want. Rather than using the function (H.1) from [25], we modify it to be a similar sigmoid function as seen in (H.2). This new function has approximately $f_s(0.001) = 0.5$ and $f_s(0.0003) = 0.005$ as wanted.

$$f_s(TD\_Error) = \frac{1 - exp(\frac{-|\alpha \cdot TD\_Error|}{\sigma})}{1 + exp(\frac{-|\alpha \cdot x|}{\sigma})} \tag{H.1}$$

$$f_s(TD\_Error) = \frac{1}{1 + exp(\frac{-0{,}01 \cdot (|TD\_Error| - 0{,}001)}{\sigma})} - \frac{1}{1 + exp(\frac{-0{,}01 \cdot (|0| - 0{,}001)}{\sigma})} \tag{H.2}$$

Now, we would like to hear your thoughts about both our logic when finding a range for the value of σ and also about using the other function that we don't really have any citation for?

Analysis of hyper parameters with corrected validation method

I

I.1 Purpose

A series of test have been conducted to find the optimal values of the hyper-parameters. The tests themselves are described in Appendix C and Appendix D, and the results are further analyzed in Chapter 5. In that chapter and in Chapter 7 the shortcomings of the test methodology was discussed. This test report employs a test methodology which produces results that more accurately represents the intention behind the hyper-parameters analysis. In the interest of time these tests will only be used with multiple distributed agents.

I.2 Test metric(s)

The performance metrics used in this report are all based on the misalignment between the received signal power and the highest achievable signal power at each time step. This metric is used instead of the received signal power alone, as this value is expected to vary significantly as the agent moves around the environment, regardless of the choices made. This is due to phenomena like free space loss and absorption, and thus a low received signal power might not necessarily correspond to bad performance of the agent.

Thus the misalignment is used instead, which is calculated as the difference between the received signal power and maximum achievable signal power in dB scale. The maximum signal power is defined as the maximum found when looking at all possible combinations of codewords in the entire codebook and not just those available to the agent at the current time step.

$$M = S_m - S_{max}$$

where

M :	$[dB]$	Misalignment	$RM \in \mathcal{R}$
S_m :	$[dB]$	Received signal power	$S_m \in \mathcal{R}$
S_{max} :	$[dB]$	Maximum achievable signal power	$S_{max} \in \mathcal{R}$

Both the received and maximum power used for calculating the misalignment are without added noise, to better allow the metric to measure the quality of the choices made. This is based on the notion that choices should be made based on the quality of the channel, which is assumed

to be independent of the thermal noise in the equipment. It should be noted that this metric is only obtainable because the test is simulated, as normally the maximum achievable signal power is not directly available.

The metrics used to compare the different agent configurations are:

- Average misalignment
- Median misalignment
- 1^{st} quartile misalignment
- 3^{rd} quartile misalignment

I.3 Test procedure

The test setup is generally the same as for tuning of state parameters. The general setup is as follows:

Table I.1: Technical parameters used when running tests with different noise amounts.

Parameter	Value	Description
B	400 MHz	Total bandwidth
f_c	28 GHz	mmWave carrier frequency
P_{TX}	40 dBm	Transmission power
P_{TN}	-78 dBm	Thermal noise power
P_{NF}	9 dB	Noise figure power
r_c	200 m	Cell radius
N_r	8	Number of receive antennas
N_t	16	Number of transmit antennas
M	10000	Number of episodes pr. simulation
N	7000	Number of steps pr. episode

A simulation consists of several episodes, with each episode consisting of one or more agents following a path, taking an action at each step. A track is chosen at random from a collection of 16 tracks at the beginning of each episode. Each track in the pedestrian collection is 20000 steps long, however the path used in each episode is only 7000 steps long. Thus a random starting point on the track is chosen as well at the beginning of each episode. By keeping the episodes shorter than the length of the tracks, it is possible to get more unique paths for the agent to follow in different episodes, along with a lot of paths with differing amounts of overlap. This means that it is not just the same 16 paths which are traversed over and over again in the simulation. 7000 steps was chosen as a fitting trade-off between path length and number of unique paths in total. The agent keeps the Q-table values between episodes, and thus learns across all episodes.

In order to ensure that the test metrics are statistically accurate, and to give the agent a chance to explore the entire state space, many episodes are simulated. Both the number of episodes and number of steps in each episode, are derived based on an intermediately sized state space. The total number of steps in the simulation should be enough for an agent to explore all state-action pair, to give accurate value estimates. The state given below, based on a single agent implementation, is assessed to be an intermediate state, and was used to derive the total number of steps.

- A history of the 2 previous beams at the UE
- A history of the 2 previous beams at the BS
- A distance between the UE and BS with a resolution of 4
- An angle between the UE and the BS with a resolution of 8

The size of the combined state-action space of this state was determined in Section 4.4.2 to be $6.05 \cdot 10^6$ state-action pairs. In order to allow proper exploration, the total number of steps for a simulation is set such that each state-action pair can be visited approximately ten times, resulting in a total step number of $70 \cdot 10^6$ steps. This gives a total of 10000 episodes in a simulation. After initial simulation the agent will be transferred to a identically distributed but independently drawn environment. A simulation of the same length will then be run again. The agent keeps all value between the simulation. The second simulation is to ensure the performance shown for a specific combination of hyper-parameters generalizes. This test period is the validation period, while the first simulation is the training period. While this allows an agents performance to converge, an even better performance measure would be an average over a statistically significant amount of agents. However, as each simulation takes approximately four hours to run, this is sadly not viable because of time and resource constraints. Thus the results presented in this chapter will be judged with this uncertainty in mind.

I.3.1 Variables

The algorithm consisst of a number of variables, where every variable expect the noise level will be kept constant throughout this test to ensure valid comparison. Two sets of variables will be presented, the first for the case of a single centralized agent, and the next for an agent at each terminal. The first set of constant variables are the following:

γ :	[−]	Forgetting factor	$\gamma = 0.7$
α :	[−]	Action value adjustment amplitude	$\alpha = 0.01$
ϵ *adaptive* :	[−]	Exploration rate	ϵ *adaptive* $= N/A$
$\mathcal{S}$:	[−]	State space	$\mathcal{S} =' 2-2-1-8-2-32'$
A :	[−]	Action space	$A = adjacent$
case	[−]	The mobility case	$case = pedestrian$
scenario	[−]	The LOS status	$scenario = LOS$

γ is chosen as the best performing, from the tests described in Appendix C. α is chosen as the best performing, from the tests described in Appendix C. ϵ *adaptive* is chosen as it is the best performing ϵ method and it's weight value is chosen as the best performing, from the tests described in Appendix C. The state space is chosen as the best performing, from the tests described in A.

The second set of variables for an agent at each terminal is:

γ :	[−]	Forgetting factor	$\gamma = 0.6$
α :	[−]	Action value adjustment amplitude	$\alpha = 0.01$
ϵ *adaptive* :	[−]	Exploration rate	ϵ *adaptive* $= N/A$
$\mathcal{S}_r$:	[−]	State space of the UE	$\mathcal{S}_r =' 2-0-1-8-2-32'$
$\mathcal{S}_t$:	[−]	State space of the BS	$\mathcal{S}_t =' 0-2-0-0-2-32'$
A :	[−]	Action space	$A = adjacent$
case	[−]	The mobility case	$case = pedestrian$
scenario	[−]	The LOS status	$scenario = LOS$

Note here that the agents at both terminals have the same hyper parameters. γ is chosen as the best performing, from the tests described in Appendix D. α is chosen as the best performing, from the tests described in Appendix D. ϵ *adaptive* is chosen as it is the best performing ϵ method and it's weight value is chosen as the best performing, from the tests described in Appendix D. The state space is chosen as the best performing, from the tests described in B.

I.3.2 Hyper-parameters to be tested

It is infeasible to test every and all value combinations and therefore a subset of values will be identified for testing. As resources are limited the search is conducted over each variable separately. That is optimization will be performed with respect to a single variable at a time, instead of over them jointly. Furthermore the values of the hyper parameters will be equal for both agents. The optimization will be performed in the order given below. The value the variable takes before being optimized is also given.

γ :	[−]	Forgetting factor	$\gamma = 0.7$
α :	[−]	Action value adjustment amplitude	$\alpha = 0.01$
ϵ *constant* :	[−]	Exploration rate	ϵ *constant* $= 0.01$
ϵ *decaying* :	[−]	Exploration rate	ϵ *decaying* $= N/A$
ϵ *adaptive* :	[−]	Exploration rate	ϵ *adaptive* $= N/A$

A thorough explanation of these variables can be found in Section 4.4.4. γ is chosen to be 0.7, as it was found to be optimal in Appendix C and is therefore used as a starting point. α is chosen to be 0.01, as it was found to be optimal in Appendix C and is therefore used as a starting point. ϵ is chosen to be 1% or 1 out of 100 steps is an exploration step. This value was found to be optimal in Appendix C and is therefore used as a starting point.

$\epsilon decaying$ is not directly chosen but calculated based on the equation given below

$$\epsilon decaying = exp(-\frac{t}{w_d}) \tag{I.1}$$

Where w_d is a weight factor controlling the rate of decay of the logarithmic decreasing function.

$\epsilon adaptive$ is not directly chosen but calculated based on a set of equations from [25]. A separate ϵ belongs to each state which is adjusted based on the following equation

$$\epsilon = \delta \cdot \epsilon + (\delta - 1) \cdot \epsilon_{adj} \tag{I.2}$$

where δ is a value weighting how much ϵ should be influenced by the calculated ϵ_{adj} value. ϵ_{adj} is a value calculated based on the TD-error as given below

$$\epsilon_{adj} = \frac{1 - exp(\frac{-abs(\alpha \cdot TD - error)}{\sigma})}{1 + exp(\frac{-abs(\alpha \cdot TD - error)}{\sigma})} \tag{I.3}$$

where α is the step size and σ is a weight adjusting how sensitive ϵ_{adj} is to changes in the TD-error.

These values are varied in a neighbourhood around their default values given above. The different values being tested are given in Table I.2.

γ	0.6	0.7	0.8	
α	0.005	0.01	0.05	
$\epsilon constant$	0.001	0.005	0.01	
ϵ *decaying weight*	150	300	600	900
ϵ *adaptive weight*	0.3	1	10	15

Table I.2: Value of variables to be tested

The values of ϵ decaying and ϵ adaptive are unlike that of ϵ constant, in the sense that the latter simply describes the value of ϵ while the two former describe a weight, which has different impact depending on the method.
The values of ϵ *decaying* are that of w_d in Eq. (I.1). They are chosen in a range around the optimal value found in C, and is resat at the end of an episode.
The values of ϵ *adaptive* are that of $\sigma \cdot 10^3$ in Eq. (I.3). They are chosen such that the average TD-error obtained in the case of a ϵ *constant* results in an exploration rate in the neighborhood of that same constant ϵ.

I.4 Training

In the following section the results of the simulations of multiple distributed agents with the hyper-parameters given in Table I.2 are presented. The training period is examined in this section, and the validation period in the next.

I.4.1 Forgetting factor

Table I.3: Table of different misalignment measurements of training of multi agents with different γ values. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
$\gamma = 0.6$	10.2 dB	7.3 dB	2.9 dB	15.6 dB
$\gamma = 0.7$	9.7 dB	6.6 dB	2.8 dB	14.8 dB
$\gamma = 0.8$	10.3 dB	7.2 dB	3.0 dB	15.8 dB

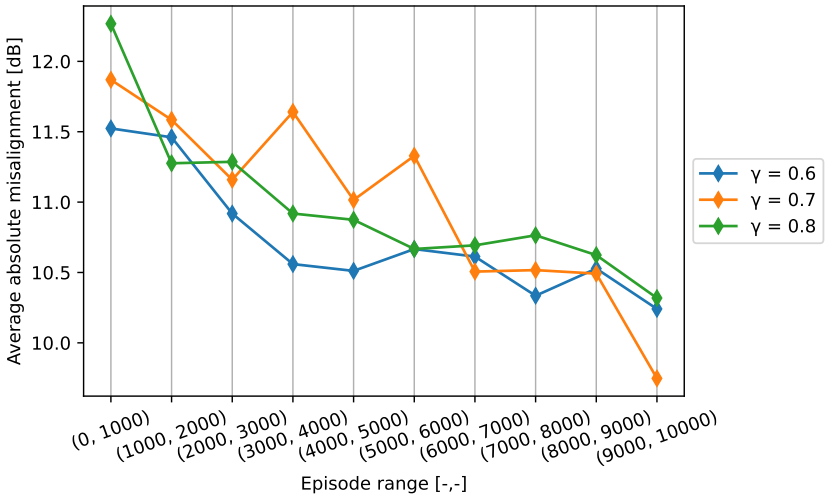


Figure I.1: Absolute misalignment of training of multi agents with different γ values.

I.4.2 Step size

Table I.4: Table of different misalignment measurements of training of multi agents with different α values. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
$\alpha = 0.01$	10.2 dB	7.3 dB	2.9 dB	15.6 dB
$\alpha = 0.005$	9.8 dB	6.7 dB	2.8 dB	15.0 dB
$\alpha = 0.05$	9.6 dB	5.9 dB	2.2 dB	15.0 dB

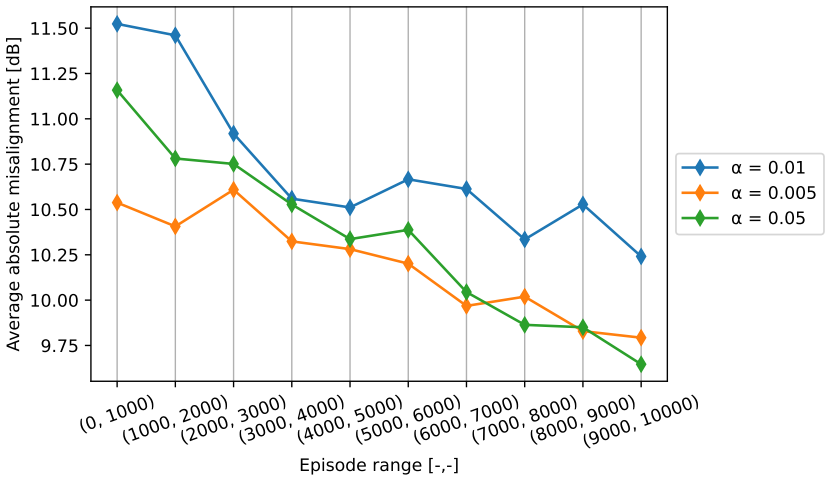


Figure I.2: Absolute misalignment of training of multi agents with different α values.

I.4.3 Exploration rate

Table I.5: Table of different misalignment measurements of training of multi agents with different adaptive ϵ values. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Adaptive: $\epsilon = 0.3$	11.7 dB	8.8 dB	3.3 dB	18.1 dB
Adaptive: $\epsilon = 1$	10.7 dB	7.5 dB	2.7 dB	16.6 dB
Adaptive: $\epsilon = 10$	10.4 dB	7.3 dB	2.7 dB	15.9 dB
Adaptive: $\epsilon = 12$	11.1 dB	8.4 dB	3.4 dB	17.0 dB

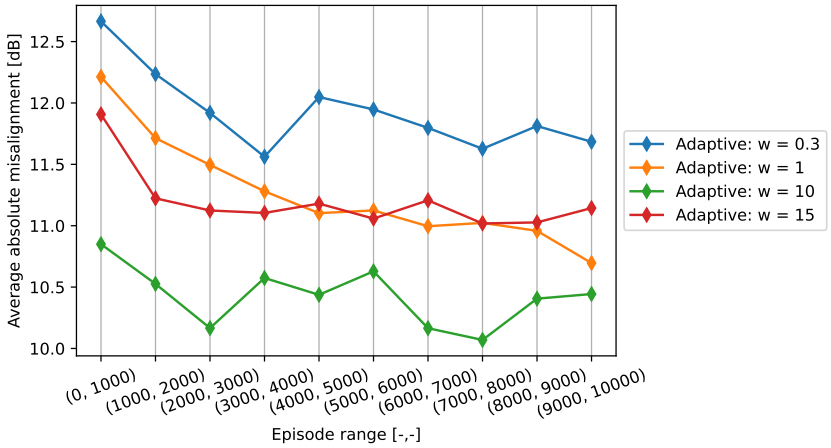


Figure I.3: Absolute misalignment of training of multi agents with different adaptive ϵ values.

I.5 Validation

I.5.1 Forgetting factor

Table I.6: Table of different misalignment measurements of validation of multi agents with different γ values. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
$\gamma = 0.6$	8.0 dB	4.7 dB	1.1 dB	12.2 dB
$\gamma = 0.7$	8.2 dB	4.5 dB	1.2 dB	12.4 dB
$\gamma = 0.8$	8.5 dB	5.4 dB	2.0 dB	13.0 dB

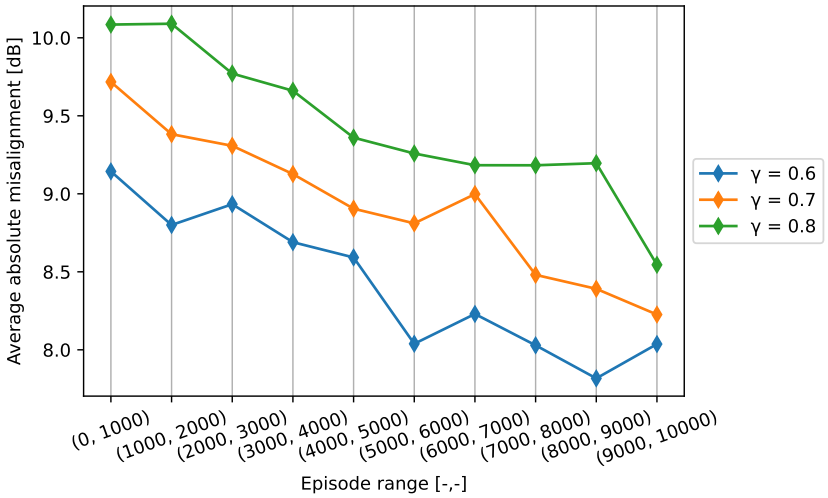


Figure I.4: Absolute misalignment of validation of multi agents with different γ values.

I.5.2 Step size

Table I.7: Table of different misalignment measurements of validation of multi agents with different α values. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
$\alpha = 0.01$	8.0 dB	4.7 dB	1.1 dB	12.2 dB
$\alpha = 0.005$	7.9 dB	4.5 dB	1.1 dB	11.7 dB
$\alpha = 0.05$	5.9 dB	5.9 dB	2.0 dB	14.8 dB

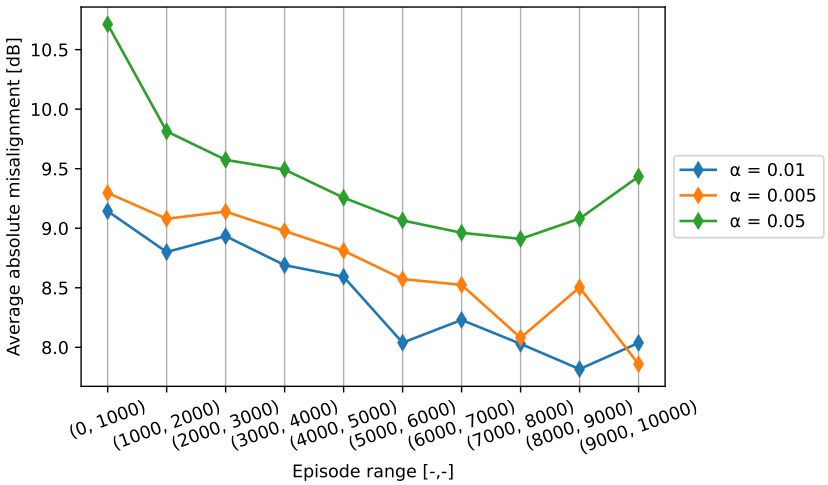


Figure I.5: Absolute misalignment of validation of multi agents with different α values.

I.5.3 Exploration rate

Table I.8: Table of different misalignment measurements of validation of multi agents with different adaptive ϵ values. Results are from the last 1000 episodes in each test.

	Absolute Misalignment			
Test config.	Average	Median	1st quartile	3rd quartile
Adaptive: $\epsilon = 0.3$	8.7 dB	4.9 dB	1.2 dB	13.6 dB
Adaptive: $\epsilon = 1$	8.4 dB	4.9 dB	1.4 dB	12.5 dB
Adaptive: $\epsilon = 10$	8.2 dB	4.9 dB	1.6 dB	12.0 dB
Adaptive: $\epsilon = 12$	9.5 dB	6.5 dB	2.7 dB	14.3 dB

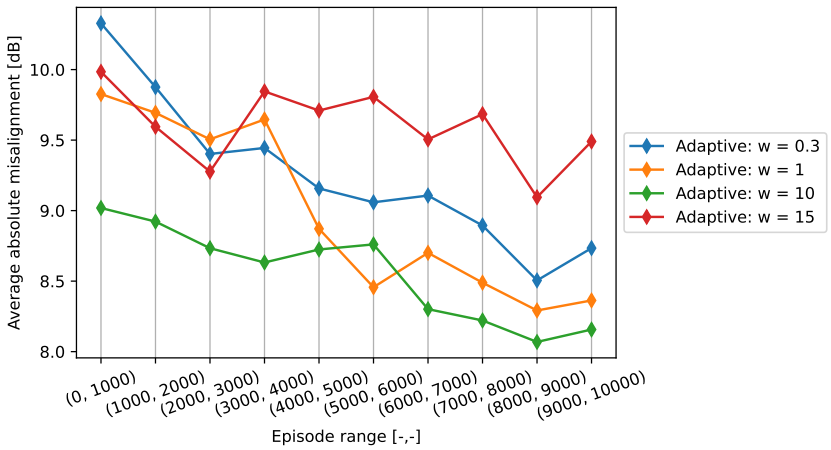


Figure I.6: Absolute misalignment of validation of multi agents with different adaptive ϵ values.