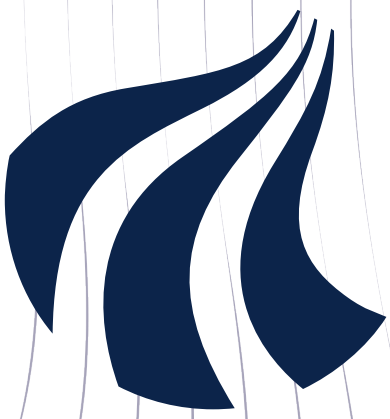
Deep-Q Learning for Adaptive Beamforming in mmWaves Communication using a Centralised System



AALBORG UNIVERSITY

STUDENT REPORT

Master's Thesis

975

Signal Processing and Acoustics

with Specialisation in Signal Processing and Computing

Aalborg University - June 2022

This project was typeset with the online LaTeX editor Overleaf. Figures have been created using draw.io and Python. All simulations have been done in MATLAB and Python.

AALBORG UNIVERSITY
STUDENT REPORT

Department of Electronic Systems
Fredrik Bajers Vej 7B
9220 Aalborg Øst
https://www.es.aau.dk/

Title:

Deep-Q Learning for Adaptive Beamforming in mmWaves Communication using a Centralised System

Project:

Master's Thesis

Project-period:

September 2021 - June 2022

Project-group:

975

Participants:

Victor Mølbach Nissen
Nicolai Almskou Rasmussen

Supervisor:

Carles Navarro Manchon

No. of pages: 143
Appendix: 70 - 143
Date of completion June 2022

Abstract:

Wireless communication in the millimeter-wave (mmWave) band is attractive due to the available bandwidth, but large free-space propagation loss is a challenge. This challenge can be overcome by increasing the signal power using adaptive beamforming techniques in combination with steerable antenna arrays. In this report, we research whether it is possible to construct an adaptive beamforming algorithm for mmWave communication using Reinforcement Learning (RL).
To be able to determine if this is possible data sets has been made using a realistic simulation of an urban environment closely following the 3GPP/5G standard modelling assumptions. Four base stations and one moving user has been simulated in two scenarios a *Car* and a *Pedestrian* for both Line-Of-Sight (LOS) and non-LOS (NLOS).
Extensive research has been done on RL theory and methods to find a suitable algorithm for the urban environment. It was found that a Deep Q-Network (DQN) should be used on a centralised system. The DQN was tuned for the *Car* LOS scenario.
The tuned DQN was then tested on both the *Car* and *Pedestrian* scenario in both LOS and NLOS conditions, and while performance is degraded w.r.t the scenario it was tuned for, we still see acceptable performance in Pedestrian or NLOS scenarios. In all scenarios the tuned DQN outperformed a simple, heuristic non-machine learning algorithm run on the same data sets.

Victor Mølbach Nissen

Nicolai Almskou Rasmussen

Preface

This master's thesis is composed on the 9th and 10th semester of the master program in Signal Processing and Acoustics, specialised in Signal Processing and Computing by group 975.

A basic setup of the environment simulation was done in collaboration with group 974 - Dennis Kjærsgaard Sand (dsand17@student.aau.dk) and Peter Kjær Fisker (pfiske15@student.aau.dk). Specifically, this includes the creation of the mobility model and the setup of the Quadriga simulation tool such that it could simulate one base station and one user equipment moving around. This led to a common worksheet which chapter 2 is based on.

All the code used in this project can be found on the group's GitHub:

https://github.com/Almskou/RL-for-Adaptive-Beamforming

In this report "." is used as decimal separator and angels are in radians if not stated otherwise.

Vectors are represented using lowercase bold whereas matrices are represented using uppercase bold.

Superscript (H) are the Hermitian transpose.

Source references in this report appear according to the standard IEEE referencing method. Sources are referenced by the numerical order in which they appear. This means that the first source is referenced with [1], the second source is referenced with [2] and so on.

The group would like to express their great gratitude to the project supervisor, Carles Navarro Manchon for the excellent and timely feedback during the project.

Contents

Acronyms

AD Automatic Differentiation.

ADC
Analog-Digital Converter.

AoA
Angle of Arrival.

AoD
Angle of Departure.

AWGN
Additive White Gaussian Noise.

BS Base Station.

DAC
Digital-Analog Converter.

DFT
Discrete Fourier Transform.

DQN
Deep Q-Learning.

ECDF
Empirical Cumulative Distribution Function.

FIFO
First in First out.

GD Gradient Descent.

IoT Internet of Things.

LOS
Line of Sight.

MAE
Mean Absolute Error.

MDP
Markov Decision Process.

MIMO
Multiple-Input and Multiple-Output.

ML Machine Learning.

mmWave
Milimeter-Wave.

MSE
Mean Square Error.

NLOS
Non-Line of Sight.

NN Neural Network.

RF Radio Frequency.

RL Reinforcement Learning.

SGD
Stochastic Gradient Descent.

SNR
Signal-to-Noise Ratio.

UE User Equipment.

ULA
Uniform Linear Array.

1 Introduction

The increased number of people who is connected to the internet is rapidly increasing. From 2016 about 3.7 billion people had a smartphone subscription which increased to 6.3 billion in 2021. These numbers doesn't seem to slow down as the prediction from Statista points to an increasement to 7.7 billion subscriptions in 2027 [1]. Its not only the smartphone usage that is increasing but in general the total usage of Internet of Things (IoT) devices is predicted to increase from 11.3 billion in 2021 to 27 billion in 2025 [2]. As with the increasement of IoT devices the demands of more bandwidth for some of the technologies are increasing [3]. Systems that connect autonomous vehicles, augmented reality, remote surgery etc. has a demand on a high data rate and needs a low latency.

So to solve the problem of more users and their demand of a higher throughput, networks using Milimeter-Wave (mmWave) frequencies are developed. The mmWave frequencies range from 30 GHz to 300 GHz, which is higher than the current frequencies used for 3G and 4G where in Denmark frequencies from 800 MHz to 2 600 MHz is used [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 as there is room for broader bandwidths or more spectral channels 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. With these perks the Ericsson Mobility Report predict that in 2024, there will be 1.9 billion 5G subscriptions [5]. As of Q4, 2021 there is 660 million subscriptions to the 5G network [6].

These desirable features using higher frequencies does 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 part will not be looked into in this report as it will be assumed that the hardware can handle it.

1.1 Power attenuation for mmWave Frequencies

This section gives an overview of the different conditions and physical laws that attenuates the signal power to estimate power attenuation for a wireless connection using mmWaves.

1.1.1 Power Loss in Free-Space

By analysing Friis transmission equation[7] (seen on eq. (1.1)) a correlation between how much power is lost over a distance given a frequency can be estimated. 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 experiences a higher degree of power loss over distance travelled compared to a lower frequency when travelling 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	[m^2]
D_t :	The transmitting antenna's directivity with respect to an isotropic antenna	[m^2]
d :	Distance between antennas	[m]
λ :	Wavelength of the radio frequency	[m]

1.1.2 Atmospheric Absorption's Rate

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 given frequency. For this reason other impacts the environment will have on the higher frequencies will also be studied. From a study about mmWave and its behaviour [8], are there several things worth noticing. The study included a figure on how much of the power will be lost due the atmospheric absorption's rate (the figure can be seen on fig. 1.1). Overall the figure shows that the atmospheric absorption's rate increase as the frequency increase, but it also shows areas where the absorption's rate increase drastically compared to nearby frequencies. These frequencies should be avoided transmitting with. The figure also shows some valleys where the absorption's rate is relatively lower and it is at these valleys where it should be preferred to transmit at.

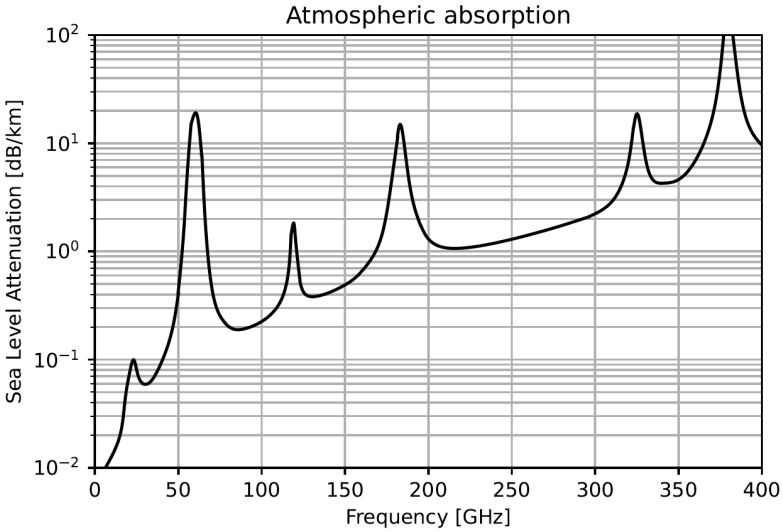


Figure 1.1: Atmospheric absorption across mm-wave frequencies in dB/km. The figure is simplified from the source [8, p. 338].

1.1.3 Penetration Power & Reflection Coefficient

The same study [8] also includes a deeper study on how well a signal transmitted at 28 GHz penetrates different materials often found in buildings. For tinted glass and brick pillars it was found that they respectively causes a penetration loss equal to 40.1 dB and 28.3 dB. Regarding the reflection coefficients it was found that for tinted glass and concrete at an indent angle equal to 10° the coefficients was respectively 0.896 and 0.815 The signal was also tested on different buildings and with different incident angles. Through this study the paper concluded that outdoor-indoor penetration is difficult due to the high penetration loss and high reflections coefficients.

1.1.4 Diffraction

Diffraction refers to the phenomena which occurs when a wave passes through a narrow aperture or across an edge. Due to this it is possible for a wave to bend around corners and propagate onward in the shadow region of a geometric object. The amount of power loss is depending on the material of the object and the frequency. A study on 10 GHz, 20 GHz and 26 GHz show that especially for outdoor materials like marble and stone a high power loss occurs [9, pp. 5–6]. The measured losses shows as much as 50 dB power loss solely due to diffraction [10, p. 5]. The study also show that for increasing frequencies the higher the loss will be. From this it can be deduced than diffraction has a very low impact on mmWave frequencies.

1.2 Alleviating the Consequences

The impact the environment has on using a higher frequency can be summed up as following:

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

4. High reflection coefficients further contributes to the difficulty in transmitting through blockages.
5. High diffraction power loss makes it difficult to transmit around edges and corners.

The low penetration power, high reflection coefficients and high diffraction power loss can be compensated for by either drastically increasing the transmitting power or changing the materials of the blockages. The decrease in communication range can also be compensated with an increased transmitting power.

The increase in transmitting power can either come from supplying the transmitter with more power or make the transmitter directional which will focus the power in one or multiple directions. This is also know as beamforming.

To be able to do beamforming an antenna array is needed. The size of the antenna is dependent on which frequency it operates on as for a larger frequency the antenna becomes smaller. This means for increasing frequencies the physical size of the antenna array becomes smaller. Increasing the frequencies gives a smaller wave length which shortens the needed distance between antennas in an array. This has the effect that complex and effective beamforming can be achieved using physically small arrays with many elements.

Beamforming is quite simple in the cases when the location of the receiver is known and there is Line of Sight (LOS) as it will just be directing the transmitting power in the direction of the receiver. Likewise should the receiver also beamform in the direction of the transmitter as it will increase the received power from that direction. The problem occurs when there is Non-Line of Sight (NLOS) as there the optimum direction to beamforming will not necessarily be in the direction pointing to the transmitter/receiver. This is due to the high penetration power loss the higher frequencies experience. Therefore in the NLOS cases the optimum beamform direction will most likely be a signal which has been reflected several times before it arrives at the receiver from an unknown direction.

To solve this problem an adaptive beamforming algorithm is needed to be implemented. This report will therefore look into how this problem can be solved using Reinforcement Learning (RL). It has been chosen to use RL as it is a machine learning method which is trained in real time and therefore can adapt to environment changes.

1.3 Test Environment

As mentioned before the environment has a large impact on the received power due to higher frequencies being easily reflected and having a poor penetration performance. Because of this the higher frequency has a reduced communication range and thus the need for more Base Stations (BSs).

Due to these reasons it has been chosen to focus on urban environments. It has been chosen that in this project the focus will be on two use cases in an urban environment. First case will be a pedestrian walking around and the second case will be a car driving. The pedestrian case will represent the augmented reality technology and the car case will represent autonomous vehicles technology.

In both cases the user will be moving around inside an urban environment. To simulate a

multi-cell environment it has been chosen to include four BSs and only let the users move inside the BSs communication range. The reason of four BSs has been chosen is to represent a small multi-cell environment without making it too big and complex.

The BS will be placed in a hexagonal structure with an inter-site distance of 200 m [11, p. 21]. This ensures that we do not exceed 100 m range. It has been chosen to use a carrier frequency equal to 28 GHz and a communication range equal to 200 m. 28 GHz has been chosen even though it is just below mmWave frequencies as it is close to the reserved frequency of 26 GHz which Denmark has taken and that it has been researched more thoroughly and therefor there are less uncertainties of its properties.

The placement of the BSs and their communication range can be seen on fig. 1.2.

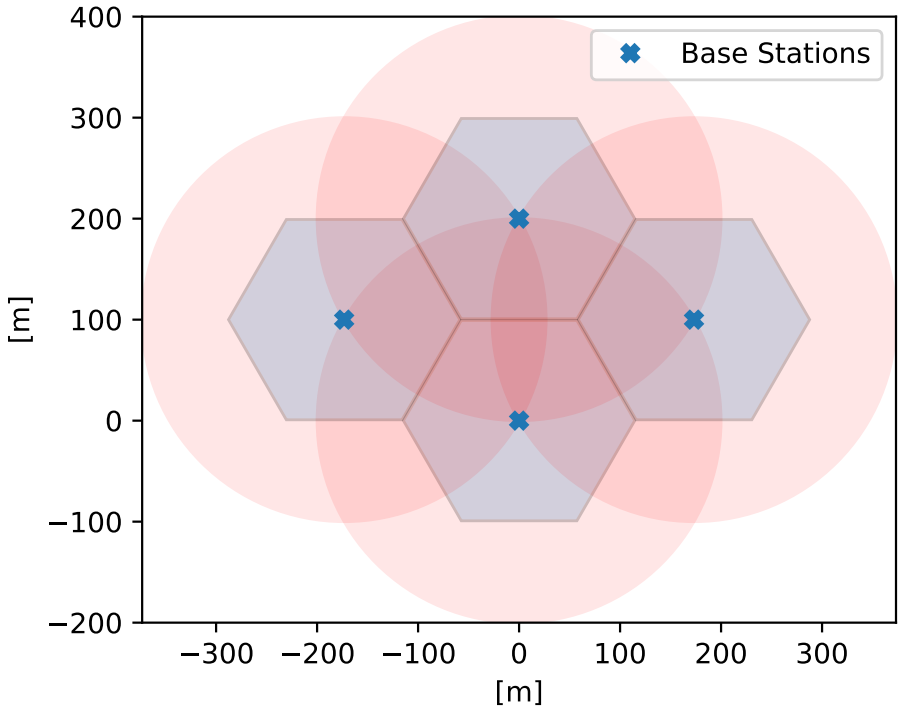


Figure 1.2: Placement of the four BSs in a hexagonal structure.

To simplify the complexity of the problem it has been chosen to focus on 2-dimensional beamforming. This includes using a Uniform Linear Array (ULA) as the antenna structure instead of a planar array which are more used in practice [12][13] [14]. But if the algorithms works well in these conditions it should be simple to implement them in a planner array and to use 3-dimensional beamforming.

1.4 Formulation of the Problem

It is wanted to construct an adaptive beamforming algorithm for the given environment described above using RL.

RL is a machine learning method like supervised and unsupervised learning. There exist

a lot of methods which RL can be realised all with their own pros and cons. It is wanted through an analysis of the environment to find a suitable RL method to be implemented. The focus of the project will then be to research sub categories of RL methods to find a suitable method to see whether it is possible to create an adaptive beamforming algorithm with RL that performs well compared to a reference algorithm.

This leads to the following research questions:

1. Which RL methods are suitable to address the problem of adaptive beamforming in the described environment?
2. What types of information are most important for an RL agent to solve the adaptive beamforming problem?
3. How well does an RL agent perform compared to hand-made heuristic method?

1.5 Report Structure

The structure the rest of the report will be described as follows. In chapter two an in-depth analyse on simulating the environment which includes the channel model, beamforming and the mobility model of the User Equipment (UE) so data can be generated for the RL-agent. In chapter three the general ideas and theory of RL will be described. Here different methods of RL will be analysed and compared where the Deep Q-Learning (DQN) algorithm was chosen. Chapter four will go more in-depth with the DQN and find the different hyper parameters needed to implement the algorithm. The hyper parameters will be split into two catagories; DQN and RL. In chapter five different test will be made to tune the DQN parameters. In chapter six the behaviour of the tuned DQN and the RL will be tested for tuning the RL parameters. Chapter seven will be a discussion of the found results. Chapter eight will be a conclusion on with it is possible for use RL for adaptive beamforming in the described environment. Lastly chapter nine will be about what could be done to improve the results in the future.

Simulating the Environment 2

Before a RL method can be used it is necessary to generate data it can be trained on. This chapter will go through how the environment described in section 1.3 will be simulated.

To simulate the environment a system model of the environment is needed. For this project it has been chosen to use the system model seen on eq. (2.1) [15]. The model is used to find how much power is received 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	$p \in \{0, \ldots, N_{b,t}\}$
q :	Beam index for receiver antenna	$q \in \{0, \ldots, N_{b,r}\}$
$N_{b,r}$:	Number of different beams for the receiver antenna	
$N_{b,t}$:	Number of different beams for the transmitter antenna	
P_t :	Transmission power	[W]
$\boldsymbol{w}_q$:	q'th combiner from codebook - dim$(N_r \times 1)$	
$\boldsymbol{f}_p$:	p'th precoder from codebook - dim$(N_t \times 1)$	
$\boldsymbol{H}$:	Channel matrix - dim$(N_r \times N_t)$	
N_r :	Number of receiver antennas	
N_t :	Number of transmitter antennas	
s :	Transmitted symbol with normalised power	$s \in \mathbb{C}$
$\boldsymbol{n}$:	Complex Gaussian noise vector - dim$(N_r \times 1)$	$\boldsymbol{n} \sim \mathcal{CN}(0, \sigma^2 \mathcal{I})$ [W]

The transmission power (P_t) can be set to an arbitrarily value except if a certain Signal-to-Noise Ratio (SNR) is needed. This is due to that the transmission power scale the received power equally for each pair (p, q). The symbol (s) can be set to one for simplicity. The combiners and precoders are to be defined from a beamforming codebook and a narrowband model is used for the channel matrix.

2.1 Narrowband model

In this section the channel model, H, used in the system model above will be described. It has been chosen to use a narrowband channel model, which is given by eq. (2.2) [15]. This model assumes that both the transmitter and receiver are ULAs, with N_t and N_r array elements respectively.

$$H = \sum_{l=0}^{L} \sqrt{N_r N_t}\, \beta_l\, \boldsymbol{a}_r \left(\theta_r^l\right) \boldsymbol{a}_t^H \left(\theta_t^l\right) \tag{2.2}$$

where

L :	Number of multi paths	
$\boldsymbol{a_t}$:	Steering vector for the transmitter antenna array - dim($N_t \times 1$)	
$\boldsymbol{a_r}$:	Steering vector for the receiver antenna array - dim($N_r \times 1$)	
θ_r^l :	Angle of Arrival (AoA) for the l'th multi path	[rad]
θ_t^l :	Angle of Departure (AoD) for the l'th multi path	[rad]
β_l :	Channel parameter for the l'th multi path	

By assuming the antenna elements are spaced evenly by $\lambda/2$ and are all isotropic, the array steering vectors can be defined as seen on eqs. (2.3) and (2.4). The steering vectors describes the relative phase difference between the antennas and the first antenna in the antenna array. The size of the steering vectors results in H being a $N_r \times N_t$ matrix.

$$\boldsymbol{a}_r\left(\theta_r^l\right) = \frac{1}{\sqrt{N_r}}\left[1, \exp\left(-j\pi\cos\left(\theta_r^l\right)\right), \ldots, \exp\left(-j\pi\left(N_r-1\right)\cos\left(\theta_r^l\right)\right)\right]^T \tag{2.3}$$

$$\boldsymbol{a}_t\left(\theta_t^l\right) = \frac{1}{\sqrt{N_t}}\left[1, \exp\left(-j\pi\cos\left(\theta_t^l\right)\right), \ldots, \exp\left(-j\pi\left(N_t-1\right)\cos\left(\theta_t^l\right)\right)\right]^T \tag{2.4}$$

2.1.1 Radio Channel Simulation

To simulate the wireless communication channel, described by β_l in eq. (2.2), the simulation tool QuaDRiGa is used. QuaDRiGa runs through MATLAB or Octave and is used for "generating realistic radio channel impulse responses for system-level simulations of mobile radio networks" [16]. This involves that QuaDRiGa has the feature to model any mobility pattern and that the channel parameters is underlying a geometric model which ensures spatial consistency. This means that the variations of the channel parameters over time is consistent with the UE mobility pattern. This is very important as it is a necessity to be able to use beamforming and tracking the UE.

QuaDRiGa supports a variety of standardised channel models and simulation settings, for different radio network scenarios [17, pp. 101–102]. In this project the scenarios "3GPP_38.901_UMi_LOS" and "3GPP_38.901_UMi_NLOS" are used for LOS and NLOS cases, respectively. 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, but as the chosen location area seen on fig. 1.2 has a greater radius than this the generated data for each BS outside the 200 m radius cannot be guaranteed to be realistic. This will be assumed not to be a problem as the power of the data generated for the BSs outside their 200 m radius will be a lot lower than the data generated for the BSs inside their radius for the same location. Due to this the optimum choice, for a given UE location, should be a BS where the location is inside their 200 m radius.

For all simulations, QuaDRiGa is set to simulate a multi path propagation, with scatterers placed randomly in the simulation area. The scatterers emulate objects in the area that reflect incoming signals, thus creating NLOS components as seen on fig. 2.1.

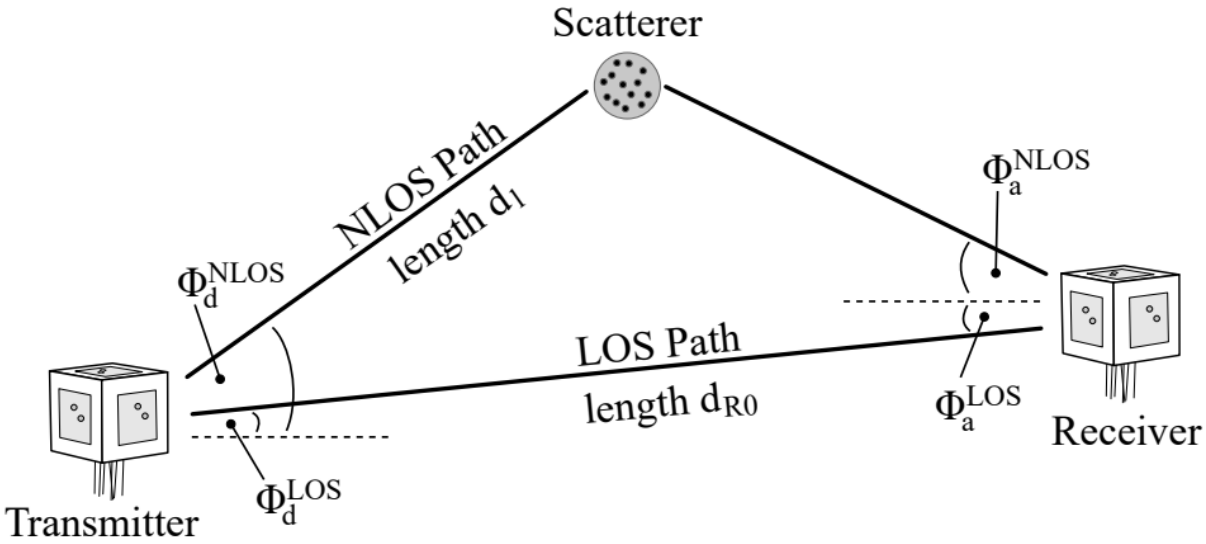


Figure 2.1: Simplified overview of the modelling approach used in QuaDRiGa [17, p. 13].

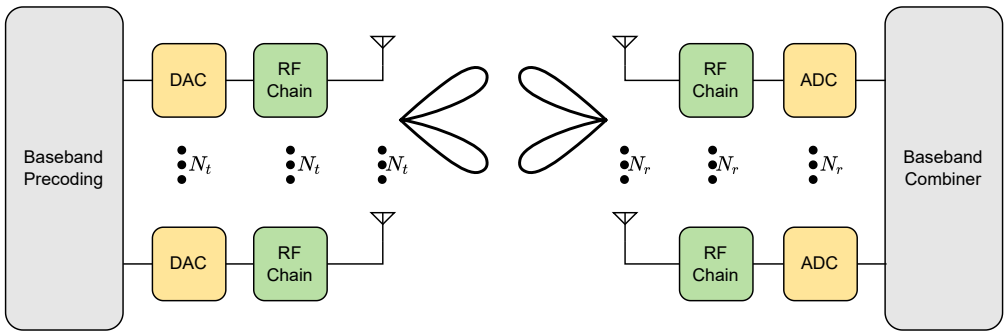
The simulations return the channel parameters (β), the AoA and the AoD for each multi path component at each position given by a mobility pattern for each antenna terminal. Additionally, the movement direction is returned as an orientation parameter. Because of the geometric model QuaDRiga is using these parameters are consistent with the UE mobility pattern. Only one antenna is needed in QuaDRiga as the steering vectors handle the difference en phase values. This is based on the assumption that the arrived signal at each antenna only has a difference in phase and non in power. Isotropic antennas are used for both the transmitter and the receiver such that the parameters only rely on the channel characteristics. Another reason is also that the directivity of the antenna array is handled through the codebooks and that it is a necessity to be able to use the steering vectors eqs. (2.3) and (2.4).

2.2 Beamforming Codebooks

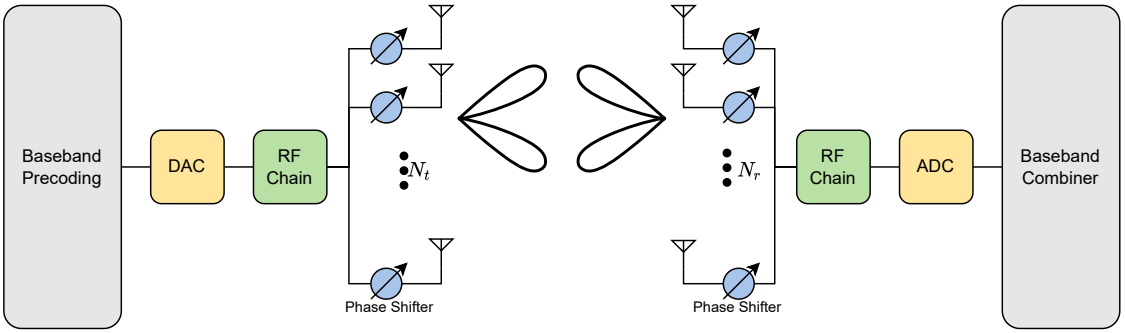
A beamforming codebook is a look-up-table where each entry contains a combination of weights. By changing how the combination of weights is constructed it is possible to make an antenna array beamform in different directions.

These codebooks can be constructed in a lot of different ways depending on the architecture of the receiver and transmitter communication systems. As mention in section 1.2 it is possible for the antenna arrays to be more compact as the frequencies increase. This has the positive consequence that physical antenna arrays does not take up a lot of space, but it has some consequences on what kind of architectures can be used. In a conventional digital Multiple-Input and Multiple-Output (MIMO) architecture there is a Digital-Analog Converter (DAC)/Analog-Digital Converter (ADC) and a Radio Frequency (RF) chain connected to each antennas as seen on fig. 2.2a. This type of architecture is typical only used at frequencies below 6 GHz [3, p. 441]. This is because it becomes very hard to physically be able to place the DAC/ADC and RF chains due to the short distances between antennas at high frequencies. Another reason is the power consumption of the DAC and ADC. The power consumption of these units becomes very power hungry in the mmWave frequencies [3, p. 441] and as there is one device for each antenna it can quite fast require a lot of power. Due to these reasons

at high frequencies an analog MIMO system architecture is used instead. A MIMO analog system can be seen illustrated on fig. 2.2b.



(a) Conventional digital MIMO architecture at frequencies below 6 GHz.



(b) mmWave MIMO system using analog only beamforming.

Figure 2.2: Both figs. 2.2a and 2.2b is based on figure 1. and 2. in [3, p. 441].

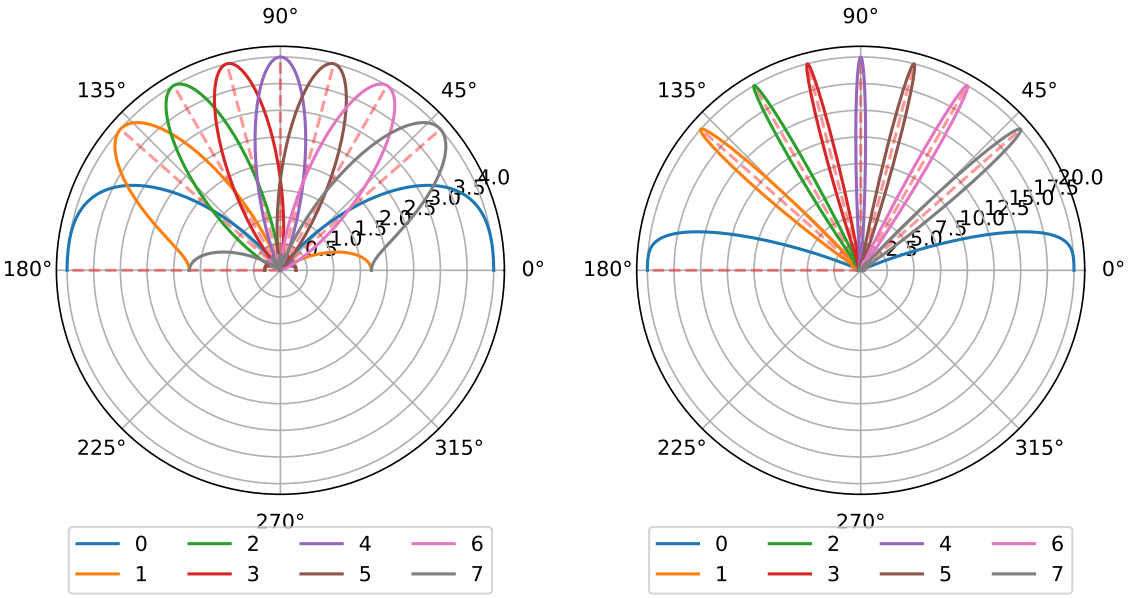
The analog MIMO system only utilises one DAC/ADC and RF chain block and instead introduces phase shifters to construct the beamforming. It does come with some consequences that it is not possible to adjust the amplitude and the control of the beamformning is limited by how quantized the phase shifters are. Due to this it is hard to finely tune the beams and steer the nulls. There exist MIMO architechtures which combines analog and digital MIMO architectures. These systems combines some of the advantages of the two architectures like making it easier to finely tune the beams and steer the nulls without utilising as many DAC/ADC and RF chain blocks as a conventional digital system. It does come with a cost of being more complex and costly than an analog MIMO system. Due to the analog MIMO system being simple it has been chosen to use this type of MIMO architecture in this project.

As mentioned in section 1.3 it has been chosen to use an ULA and for a matching codebook in an analog MIMO architecture it has been chosen to use a Discrete Fourier Transform (DFT)-based codebook [15, p. 2]. Due to using a ULA the array the beams are mirrored around the x-axis assuming the antennas lays on the x-axis. Therefore it is only necessary to look at the half plane. Using this codebook the precoders and combiners are calculated by:

$$\mathbf{f}_p = \frac{1}{\sqrt{N_t}} \left[1, \exp\left(-\mathrm{j}\pi \frac{2p-1-N_{b,t}}{N_{b,t}}\right), \ldots, \exp\left(-\mathrm{j}\pi\left(N_t-1\right)\frac{2p-1-N_{b,t}}{N_{b,t}}\right)\right]^T \tag{2.5}$$

$$\mathbf{w}_q = \frac{1}{\sqrt{N_r}} \left[1, \exp\left(-\mathrm{j}\pi \frac{2q-1-N_{b,r}}{N_{b,r}}\right), \ldots, \exp\left(-\mathrm{j}\pi\left(N_r-1\right)\frac{2q-1-N_{b,r}}{N_{b,r}}\right)\right]^T \tag{2.6}$$

The size of the precoders and combiners corresponds to the size of the antenna array and the amount of them corresponds to the number of beams. It can seen on eqs. (2.5) and (2.6) that the phase shift value is depending on three factors: the number of beams $N_{b,r}/N_{b,t}$, number of antennas N_r/N_t and the chosen beam p/q. Both equations will construct the same codebook if the number of antennas and beams are equal for both the precoder and the combiner. The chosen beam number controls the beam direction, whereas the number of beams controls the resolution in beam directions. This can be seen on fig. 2.3 where the number of beams is equal but the number of antennas differs. By having a higher number of beams it can create an overlap between the beams whereas if the number of beams is less no overlap happens and there may be areas where the gain is so small no signal will be detected. Therefore it is often wanted to have more beams than antennas to ensure that the antenna array can beamform in every direction with no significant loss in the areas between the peaks of the beams.



(a) $N_{b,r} = 8$ with 4 antennas

(b) $N_{b,r} = 8$ with 20 antennas

Figure 2.3: DFT - codebook used for getting the beam pattern of the array when using beam q. The plot is created by combining the DFT-based codebook (eqs. (2.5) and (2.6)) with the antenna array's characteristics (the steering vector in eqs. (2.3) and (2.4)) it is possible to calculate the absolute value of the gain as beam: $D(q,\theta) = \left|\mathbf{w}(q)^H \cdot \sqrt{N_r} \cdot \mathbf{a}_r(\theta)\right|^2$.

The beam pattern with the use of four and 20 antennas with both utilising eight beams can be seen on fig. 2.3. To show the gain difference normalisation was not used. The figure shows a sharper beam (fig. 2.3b) will have more gain in its direction compared to the broader beams

in fig. 2.3a. It can be seen that the gain becomes five times larger when using five times as many antennas. This is because when choosing the best angle for a certain beam the power of $\|W\|_2^2 = 1$ and $\|a_r\|_2^2 = N_r$ when no normalising is applied otherwise $\|W\|_2^2 = 1/\sqrt{N_r}$ and $\|a_r\|_2^2 = N_r/\sqrt{N_r}$. In the non normalised case the power in best case is linearly depending on the number of antennas and therefore when having five times more antennas it is possible to get five time more power in the peak direction. By having sharper beams it is possible to transmit further as the power is more concentrated in one direction than using a boarder beam assuming the total power of the antenna arrays are the same. The downside of a smaller beam width, is that it losses some of its coverage. This may result in it being harder to find or track a signal as the importance of choosing the correct beam increases.

2.2.1 Number of Antennas

With mmWave frequencies the antenna arrays physical size and the distance between the antennas becomes significantly smaller compared to using a lower frequency. This gives some design advantages as a large antenna array can be utilised in a small area. As mmWave is still being developed many different design and array sizes are being tested. [18] utilise 32 antennas for the UE and 64 antennas for the BS. Other 5G systems points towards the use of eight elements for the UE [14]. For this project it has been chosen to use four antennas and eight beams for the UE and eight antennas and 16 beams for the BSs. It has been chosen to limit the number of possible beams there can be chosen from to simplify the problem.

2.3 Mobility Model

As mentioned earlier, QuaDRiGa needs a number of inputs, one of which being the positions of the mobility pattern for each antenna terminal in the environment. Because it is intended that the UE is mobile, simulation of realistic movement patterns is required. The UE track will be simulated such that it is always inside at least one of the four BSs' communication range.

The mobility model used is based on an implementation of the model described in [19].

This model is described both for continuous and discrete time, the implementation for this project is for discrete time. The model is based on stochastic events dictating movement and velocity changes. At each point the model calculates the velocity and direction and produces the next point based on these factors and the sampling period Δt.

2.3.1 Velocity

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

- μ_v [s], mean time between changing target velocity
- v_{max} [m/s], maximum velocity
- v_{min} [m/s], minimum velocity
- **vp** [m/s], list of preferred velocities
- **pvp**, list of probabilities associated with preferred velocities
- 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 velocity is changed when a velocity change event occurs. There is a certain chance at each time step (Δt) that a velocity change event occurs, derived from μ_v as seen on eq. (2.7).

$$pvc = \frac{\Delta t}{\mu_v} \tag{2.7}$$

where

pvc: Probability for velocity change at each time step

If an update velocity event occurs a new target velocity is chosen and the unit accelerates or decelerates to match this velocity. The acceleration and deceleration is taken from a uniform distributing between zero and the maximum value. How it chooses the target velocity can be seen on eq. (2.8).

$$v = \begin{cases} vp_1 & p \leq pvp_1 \\ \vdots & \\ vp_n & 0 < p - \sum_{j=1}^{n-1} pvp_j \leq pvp_n \\ \mathcal{U}[v_{min}, v_{max}] & \text{otherwise} \end{cases} \tag{2.8}$$

The target velocity is drawn based on the list of probabilities given in *pvp*. These probabilities each match an entry in the list of preferred velocities given in *vp*. These entries add up to a number in the range $\sum_{i=1}^{n} \text{pvp}_i \in [0;1]$. The reason they do not necessarily add up to one is because the remaining probability is assigned to a random velocity. If this random velocity is chosen, instead of one of the preferred velocities, then a random velocity between *vmin* and *vmax* is uniformly drawn and set as the target velocity.

2.3.2 Direction

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

- μ_θ [s], mean time between changing target direction
- ct_{max} [s], maximum curve time
- ct_{min} [s], minimum curve time

In a similar manner to a velocity change, when a direction change event occur a new target direction is chosen. Each time step the probability of a direction change event is derived from μ_θ as seen on eq. (2.9).

$$pdc = \frac{\Delta t}{\mu_\theta} \tag{2.9}$$

where

pdc: Probability for direction change at each time step

When such an event happens a target direction is drawn uniformly $\theta_{target} = \mathcal{U}[-\pi, \pi]$. Together with a target direction a curve time is drawn uniformly $ct = \mathcal{U}[ct_{min}, ct_{max}]$. This curve time is converted to a list of time steps, over which the direction is changed to the target direction as seen on the following equation.

$$\theta(t) = \theta(t - \Delta t) + \Delta\theta \tag{2.10}$$

$$\Delta\theta = \frac{\theta_{target} - \theta}{ct/\Delta t} \tag{2.11}$$

where

$\Delta\theta$: How much the direction changes at each time step [rad]

$\Delta\theta$ is only updated when a direction change event occur or when the direction is equal to θ_{target} in which case it is set to 0. The direction must be between $[-\pi, \pi]$ and therefore wrapping has to be implemented to ensure that the direction stays between the limit.

2.3.2.1 Velocity and direction correlation

Both velocity and direction changes currently occur independently, however this is in realty not necessarily the case. A complete stop is often followed by a direction change, such as at intersections and a direction change are also often preceded by a velocity decrease [19]. In order to accommodate the first case a new value is introduced "*psdc*". Whenever the velocity of the unit is changed to zero there is chance to enter the stop-turn-and-go scenario. When entering this scenario *psdc* controls the probability if a turn should be made. When turning there is a 50% chance to turn 90° left and 50% chance to turn 90° right as seen on eq. (2.12).

$$turn = \begin{cases} left & p \leq psdc/2 \\ right & psdc/2 < p \leq psdc \\ straight & \text{otherwise} \end{cases} \tag{2.12}$$

where:

psdc: Probability for taking a 90° turn when stopping

The second case is accommodated by calculating the maximum velocity for each turn and decelerating before turning if current velocity exceed the calculated maximum velocity. The maximum velocity a turn can be navigated at is dependent on the friction between the UE and the ground, the curvetime (*ct*) and target velocity. It can be calculated as seen on eq. (2.13).

$$vturn_{max} = \sqrt{\mu_s \cdot g \cdot rc} \tag{2.13}$$

$$rc = \frac{v_{target} \cdot ct \cdot \Delta t}{\Delta\theta} \tag{2.14}$$

where:

$vturn_{max}$:	Maximum turn velocity	$[m/s]$
μ_s:	Static friction between UE and the ground	
rc:	Curve radius	$[m]$

When the maximum turn velocity ($vturn_{max}$) has been calculated it must be checked whether it is less than the current UE's velocity (v) or not. If this is true then the velocity should decelerate appropriately as seen on eqs. (2.15) and (2.16).

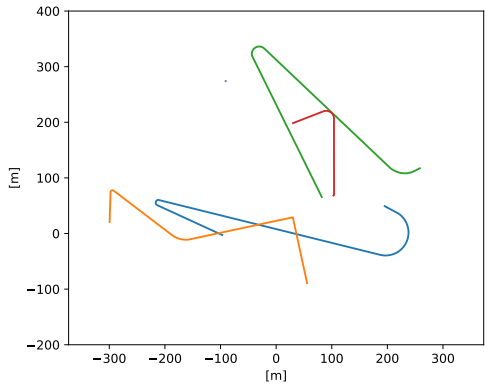
$$v(t) = v(t - \Delta t) + a(t) \cdot \Delta t \tag{2.15}$$

$$a = \begin{cases} -\mathcal{U}[0, dec_{max}] & v > vturn_{max} \\ 0 & \text{otherwise} \end{cases} \tag{2.16}$$

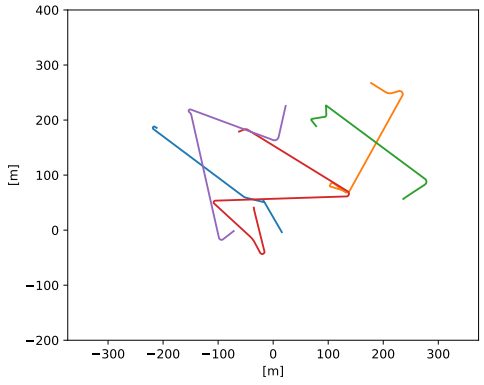
2.3.3 Sample Period

When generating the random walk it is needed to determine how often a sample should be taken. How often the sample should be taken controls how correlated the data generated from the QuaDRiga simulation will be. In section 1.3 it was chosen to use a carrier frequency equal to 28 GHz which corresponds to wavelength equal to 1.07 cm. Due to the short wavelength the channel characteristics can change drastically by moving a few centimetres. It is assumed that to learn which beam direction is most optimum at each sample a correlation between the samples is needed. This restrict how slow samples has to be taken. As the sample period is given in time it also depending on the velocity of the UE. A UE which moves at a high speed requires a lower sample period compared to one which moves slower. Therefor the sample period must be based on the chosen frequency but also the velocity of the UE.

Some examples of the output of the model can be seen in fig. 2.4 given the parameters listed in table 2.1.



(a) Five random mobility traces created by using 10000 samples for the mobility model in the urban car case



(b) Five random mobility traces created by using 50000 samples for the mobility model in the pedestrian case

Figure 2.4: Output of the mobility model using the urban car and pedestrian cases

2.4 Orientation Correction

When beamforming occurs it does so with respect to the direction of the antenna array. This needs to be taken into account as beamforming affects the measured signal power. When calculating the steering vector the local angle is needed but only the global angle is given 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

because of the way the steering vector is calculated 0° isn't the direction the antenna is facing an offset of $\pi/2$ is added. This is also described by equation eq. (2.17).

$$AoA_{local} = \frac{\pi}{2} + AoA_{global} - Orientation_{global} \tag{2.17}$$

The conversion is illustrated in fig. 2.5, where the black arrow is the orientation of the antenna, and the blue arrow is the angle of arrival of the signal.

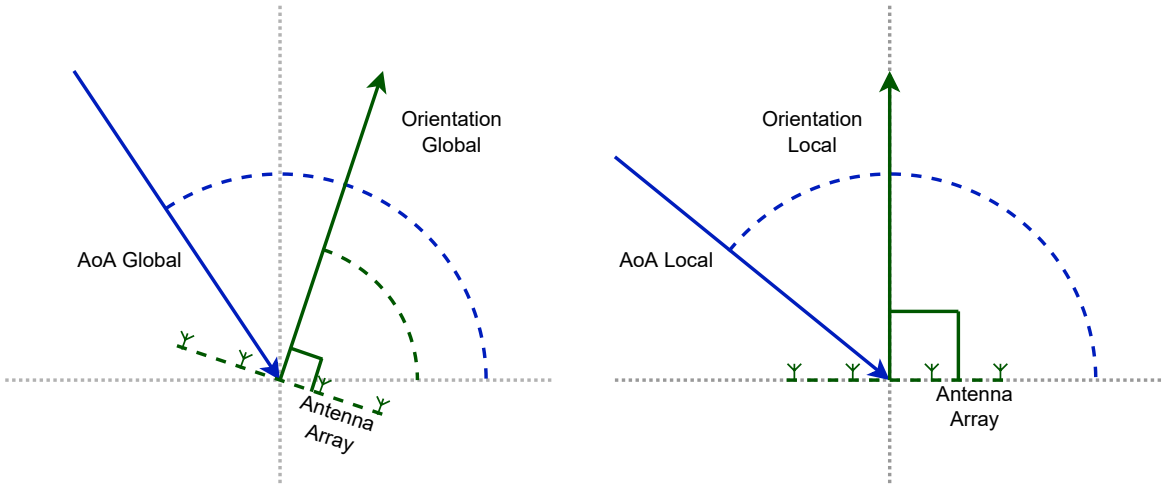


Figure 2.5: Orientation changes from global to local.

2.4.1 Time-Varying Orientation Model

The orientation extracted from QuaDRiga is the same as the direction of which the receiver is travelling. To simulate the behaviour of a pedestrian using a phone, noise is added. A pedestrian will most likely move the phone around while moving which makes changes in the orientation. These changes will be simulated using a filtered random walk from the paper [20, p. 51840]. It should be noted that it is only the angle around the z-axis which is used as the z-coordinate from the mobility model is set to a constant value. Here the noise is calculated by:

$$\bar{\alpha_t} = \bar{\alpha}_{t-1} + \mathcal{N}(0, \sigma^2) \tag{2.18}$$

where:

$\bar{\alpha}$: Random walk

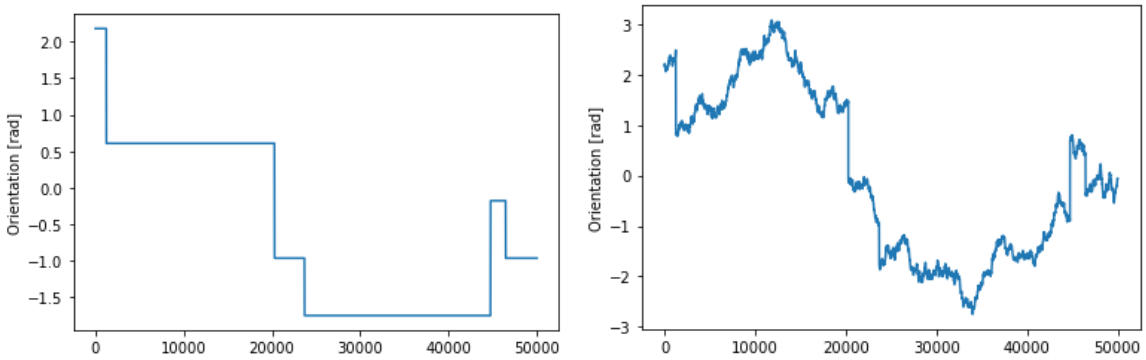
Then the filter walk with a length of K is calculated as a moving average:

$$\alpha_t = \frac{1}{K} \sum_{k=0}^{K-1} \bar{\alpha}_{t-k} \tag{2.19}$$

where:

α: Filtered random walk

where the start value $\bar{\alpha}_0 \sim \mathcal{U}[0, 2\pi]$.



(a) Orientation pedestrian without noise **(b)** Orientation with added noise

Figure 2.6: Orientation of a pedestrian's walk with and without added noise. Implemented with $K = 21$ and $\sigma = 0.5°$.

The smoothness of the filter can be adjusted with the value of K, where a larger K gives a smoother variation of orientation. The output from the filter is then added to the orientation parameter from QuaDRiGa. Afterwards it is wrapped to be within [-π, π]. The orientation before and after the added noise can be seen on figure fig. 2.6a and fig. 2.6b respectively.

The variance of the added noise, σ^2 from eq. (2.18) determines how much of a rotation is added. So with a larger value of σ^2 a larger rotation is to be expected.

2.5 Summary

Through this chapter it has been described how, by creating a realistic random walk and putting it into the described models, it is possible to simulate the environment described in section 1.3. The method used to generate the random walks is parameterised and through these parameters the two cases - pedestrian and car in an urban environment can be realised. The random walks will then be put into the QuaDRiga simulation which generates the realistic channel parameters for both LOS and NLOS paths. The QuaDRiga simulation also calculate the AoA and AoD for the different simulated paths and with all these parameters the channel matrix can be constructed. With a codebook for both the transmitter and receiver and together with the channel matrix it is now possible to calculate the power which the UE will receive for a given beam pair. This means that the received power for a given step in the random walk is only depending on the choice of beam pair.

The next step is then to construct an algorithm using RL which will choose the beam pair which return the highest received power.

2.5.1 Parameter cases

In table 2.1 the parameters for the two cases and their chosen values can be seen. These values are derived from [19], [21, p. iv], [22, §42] and what has already be described in this chapter. It has been decided to use 10 ms as the sample period based on the length of radio

frame used for 5G - New Radio [23]. Thus, it is assumed that the BS and UE adapt their beamforming configuration once per frame. Regarding the values for the mobility model (Velocity, Event, Acceleration and Rotation) a calibration should be made by collecting data of the mobility of real users and then fitting the model parameters to the data observed in the different use-cases. This has been chosen to be be out of scope for this project as the focus lies on the algorithmic aspects of the RL solutions discussed in the next chapter. Therefore, it has been found sufficient to use these parameters as it gives a mobility pattern that seem realistic and contain sufficient diversity of trajectories and orientations.

		Pedestrian	Car
		Urban	Urban
Scenarios	LOS	3GPP_38.901_UMi_LOS	
	NLOS	3GPP_38.901_UMi_NLOS	
Velocity	v_{max} [m/s]	2	14
	v_{min} [m/s]	0	0
	vp [m/s]	[0, 0.7, 1.4]	[0, 8.3, 14]
	pvp	[0.1, 0.1, 0.4]	[0.3, 0.1, 0.4]
Event	μ_v [s]	100	50
	μ_θ [s]	100	50
	$psdc$	0.33	0.33
Acceleration	acc_{max} [m/s^2]	1.44	2.5
	dec_{max} [m/s^2]	0.6	4
Direction Change	ct_{max} [s]	10	10
	ct_{min} [s]	5	1
	μ_s	1	0.7
Sample Period	sample_period [s]	0.01	0.01
Base Station	no_BS	4	4
	no_BS_antennas	8	8
	no_BS_beams	16	16
User Equipment	no_UE	1	1
	no_UE_antennas	4	4
	no_UE_beams	8	8
Noisy Orientation	K	-	21

Table 2.1: Parameters for the two different cases.

Reinforcement Learning 3

RL is based on how humans and animals learns by rewarding desired behaviours and punishing undesired ones. By taking an action the learner receives a reward and by trying different actions the learner discovers which actions yields the best rewards in the long term. This is the idea behind RL. Here the learner is called an agent which interacts with an environment and through these interactions the agent builds up knowledge where the goal is to learn to take the most optimal actions [24].

3.1 Agent-Environment Interaction

By interacting with an environment the agent receives a numerical reward (R_{t+1}) based on the action it took (A_t) and a new interpretation of the environment in the form of a state (S_{t+1}). Based on the given state (S_{t+1}) the agent will take a new action (A_{t+1}) which again will affect the environment and lead to a new state. This circular interaction loop between the agent and the environment can be seen on fig. 3.1.

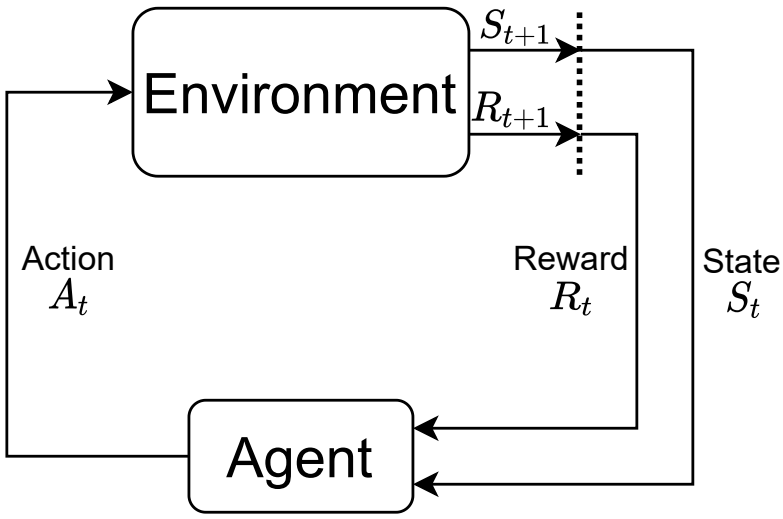


Figure 3.1: The agent-environment interaction in RL

The figure is based on the concept of the Markov Decision Process (MDP) framework where learning can be achieved by breaking it down to three signals from the agent and the environment. One signal that represents the choice taken (action), one that represents the basis of the choice (state) and a signal to achieve learning (reward). In an MDP-process the agent has all the information needed given the state it is in and through this the transition

probabilities is defined as following:

$$p\left(s', r|s,a\right) \doteq Pr\left(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\right) \tag{3.1}$$

where:

- $\mathcal{S}$: Finite set of states.
- $\mathcal{A}$: Finite set of actions
- $\mathcal{R}$: Reward set, $\mathcal{R} \subseteq \mathbb{R}$
- s': Next state taken from $\mathcal{S}$
- s: State taken from $\mathcal{S}$
- r: Reward received from $\mathcal{R}$
- a: Action taken from $\mathcal{A}(s)$

This is the property that defines the structure of the MDP also known as the Markov property of MDPs. Here the state transition and reward function is only dependent on the current state and action and not on previous once. In many problems this is not applicable as the agent does not know everything and therefore it cannot know the transitions probabilities. It is through learning about the environment the agent learns to estimate these probabilities.

An example of a RL problem is the board game chess. Here the goal of the agent is to win the game. The environment for this case is the chess board together with the chess pieces and the state the agent receives is the position of the chess pieces.

This can be represented as a list with an element for each chess piece $\mathcal{S} = S_1 \times S_2 \times \cdots \times S_{32}$. Each element will then contain the included information about the individual chess piece. In this case the information is the position $S_1 = [x, y]$ where x and y can take values from $[0, 7] \in \mathbb{Z}$ which corresponds to the position grid on the chess board. So in total each chess piece can be in $8 \cdot 8 = 64$ different position. This results in the total state space having a size equal to 64^{32}. This is a huge number and not every state is legal as two chess pieces cannot be placed on the same position. A more complex algorithm [25] reported in 2021 that there is about $4.5 \cdot 10^{44}$ legal chess positions.

The actions that the agent can take is all the legal moves the chess pieces has and is dependent on the state ($a \in \mathcal{A}(s)$). The legal moves changes as the game progresses as in the beginning only the pawns and the knights can move, but as the game progress as different states are visited different legal moves becomes available.

By letting the agent play multiple chess games it learns the transition probabilities such that it knows which actions gives the highest probability for winning the game for the given position of the chess pieces. Not all agents learns the transition probabilities directly, most agents learn it indirectly through a policy and a value function.

3.1.1 Value Function and Policy

When the agent chooses an action it is almost always based on a value function [24, p. 58]. There are two popular value functions the state-value function which is a function of states and the action value function which is a function of state-action pair. The first function gives a value that indicates how good it is for the agent to be in a state while the other function value tells the agent how good it is to take an action in a given state. To be able to determine which

action will give the best rewards in the long term, and not only the intermediate reward, the cumulative reward is used which in the RL framework is called the return and can be seen on eq. (3.2) [24, p. 55].

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^k R_{t+k+1} = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.2}$$

where

γ: Forgetting factor $\quad 0 \leq \gamma \leq 1$

The forgetting factor is a variable that tells how much future rewards should impact the cumulative reward. So having a large forgetting factor means that future rewards will have a larger impact compared to a smaller value. This can become very useful as not all RL-problems has a terminating state, meaning that future rewards are infinite. So by doing this the discounting reward becomes finite if the sequence R_k is bounded. But the reason to use cumulative reward is to take future rewards into consideration when the agent must choose an action. An example of the importance of cumulative rewards is made in section 3.3.1.

The value function is also depending on the given policy as the policy controls which action should be taken which affects the received reward.

"Formally, a policy is a mapping from states to probabilities of selecting each possible action." [24, p. 58]. This is a probability distribution for taking an action in a given state and can be described as:

$$\pi(a|s) = Pr(A_t = a|S_t = s) \tag{3.3}$$

The value function is defined as the expected return in the given state for a given policy:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k}|S_t = s\right] \tag{3.4}$$

The action value function is defined as the expected return taking an action in a given state for a given policy:

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k}|S_t = s, A_t = a\right] \tag{3.5}$$

By using these methods the goal is to achieve an agent that uses a value function which is dependent on a policy that can choose actions that gives good cumulative rewards.

There exist a lot of different policies and one of the more common and general ones is the ϵ-greedy policy. This policy chooses the action which return the highest estimated value $100(1-\epsilon)\%$ of the time for $0 \leq \epsilon \leq 1$ and for the rest of the time it will choose a uniformly distributed random action between all possible actions.

When the ϵ-greedy policy chooses the highest estimated value it does an exploitative move and the rest of the time it does an exploratory move. The value of ϵ controls how often the RL agent should do a exploratory move and is also a hyper-parameter which can be adjusted when designing the RL agent.

Often it is wanted to have a highly exploratory agent in the beginning as it operates in a new environment and therefore needs to explore to obtain knowledge about its environment. The reason for utilising exploration is because in RL we do not know the transition probability and therefore we do not know the value functions of a given policy. So to be able to estimate these value functions exploration is used as the agent takes different actions for the same state and uses the observed rewards to calculate estimations of the value functions in order to maximise the the estimated values.

But as it gets more knowledge about the environment it should explore less and exploit more. Ideally it should stop exploring at some point and only use a greedy policy ($\epsilon = 0$) which fully exploit its estimated values. This only works in cases where the environment is stationary which is not the case for the environment explained in this report as it simulates an urban environment where changes can happen. So for this case it would still be preferable to explore a lot in the beginning and less the longer it runs but it should never stop exploring as the environment is not assumed to be stationary.

3.2 Reinforcement Learning Methods

For the agent to make a good choice in deciding the best action it needs to have a good estimate of the possible function values. This can be done in a lot of ways and this section will briefly go through the overall categories and the direction for the rest of the this chapter. A taxonomy of the overall categories and some of the algorithms which belongs to them can be seen on fig. 3.2.

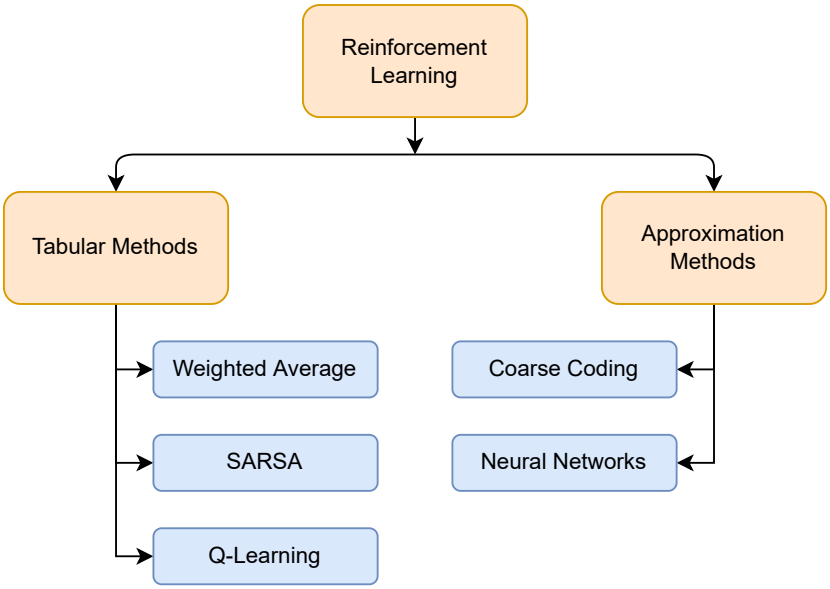


Figure 3.2: Overview of different RL algorithms categorised into either tabular or approximation methods.

The most simple RL methods are called tabular methods. These methods are called tabular

because each state-action pair is assigned a slot in a table and when updating the estimated value it is only done for a single state-action pair. The other methods which are known collectively as function approximation method are based on the same theory as tabular methods but expands further on it. Some function approximation methods still use tables such as coarse coding methods but instead of updating only one state-action pair they update multiple estimation values at the same time. Other approximation methods do not use tables and one of these methods is the DQN which uses a Neural Network (NN) to estimate its state-action pairs' value. Here the state will be the input of the NN and the output will be the estimated value for each possible action. The agent then take the action with the highest estimated value.

Due to tabular methods having a low computation complexity compared to approximation methods it will be preferable if these methods are applicable for this case.

The tabular method does not come without limitations. As the estimated value for each state-action pair is stored in a table, the size of it can become too large. This can limit the choice of parameters used in the state. One of these limits is that a state cannot include continuous features like the position. If the position is needed it has to be discretised or else it will be impossible to store everything in a table. Tabular methods also only update one entry at a time. This has the consequences that the methods cannot see if some states are correlated which some function approximation methods can. Due to this it cannot gain knowledge about states it has not been in. It can also be bad or slow to update the estimation of states which have not or rarely been visited before. Function approximation method can solve a lot these problem but it comes with the cost of an increased complexity. Furthermore it can no longer find the exact estimation for each action-state pair assuming the environment is stationary as can be achieved using tabular methods.

3.2.1 Centralised vs Decentralised

Cases where multiple UEs and/or BSs work in the same environment different kind of RL systems can be realised. One is having an agent for each UE and BS. This is also known as a decentralised system. Here the different agents have their own observation which is used to take an action. The opposite is a centralised system where a single agent executes all actions. As an example looking at this project's case with the four BS and the single UE, a single agent can be constructed to chose which beam each BS and the UE should take. One implementation could be that one BS is getting information from the other BSs and the UE to setup a single agent utilising the knowledge gathered. This can become a very complex and large system. In a decentralised system it can become very difficult for each agent to learn something as every agent is affecting the environment at the same time, but the complexity stays relative low compared to the centralised case. For this project is has been chosen to implement a centralised system.

3.3 Estimating Action Values

As mentioned earlier RL is a machine learning method which is trained on a reward it receives for taking an action in a given state. There exist several tabular methods and in this section there will be looked at three methods, weighted average, SARSA and Q-learning [24, pp. 32, 129, 131].

The weighted average updating method can be seen on eq. (3.6). Here the estimated value in the table Q is updated based on the loss value and a step size. The loss value is calculated as the difference between the received reward and the estimated one. The step size is a hyper-parameter and is one of the parameters which can be adjusted when creating the RL agent.

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[R_{t+1} - Q(S_t, A_t)\right] \tag{3.6}$$

where:

Q:	Function of estimated action values for each state-action pair
A_t:	Chosen action taken at time step t
S_t:	State at time step t
R_{t+1}:	Reward received for the taken action in the given state at time step t
α:	Step size

3.3.1 SARSA & Q-Learning

The weighted average updating method only updates the action value with regards to the intermediate reward. As mentioned this can have some consequences compared to using a cumulative reward.

An example where cumulative reward is useful will be described, where an agent can go either left or right. At the starting point if the agent chooses to go left it moves into a environment where the received reward will be a lot higher than if it went right. The problem lies in that the intermediate reward it gets when it moves to the left is lower than the reward it receives when it moves to the right. This example can been seen fig. 3.3, where the agent always starts in the state $S0$. Due to the intermediate reward towards the bad part of the environment is higher than the intermediate reward towards the good part of the environment the weighted average updating method will estimate the value of $Q(S0, left)$ to be lower than $Q(S0, right)$. If the agent chooses the highest estimated value it will go to the right state even though in the long run the agent will receive better rewards going to the left.

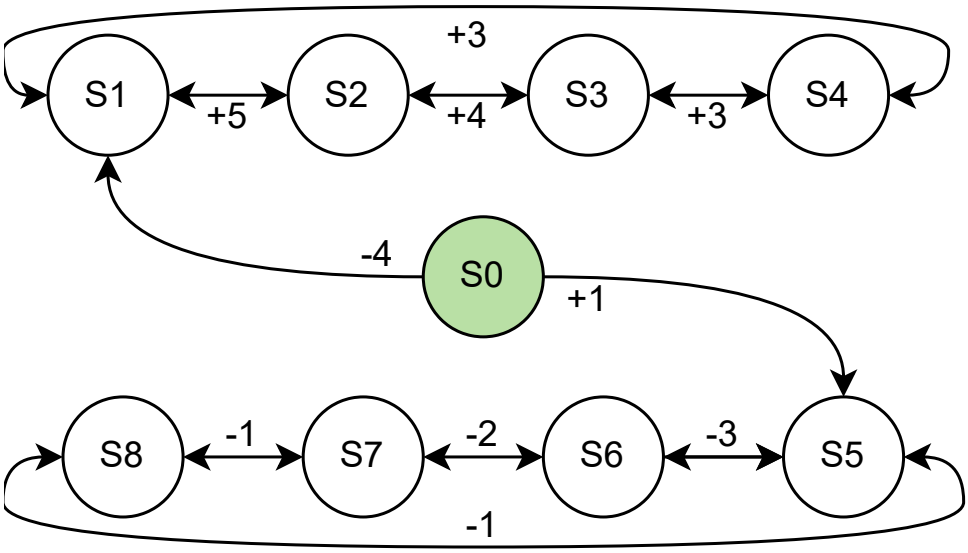


Figure 3.3: Example environment where an agent always start in state $S0$ and its actions is either going left or right. Each time it takes a choice it get an associated reward as seen on the figure.

This example has been implemented with an ϵ-greedy policy where the value of ϵ decreases over time. The step size α is set to be 0.7 and the simulation of the example is set to run 100000 episodes where the agent moves ten steps for each episode before starting at state $S0$ again. The result of the simulation can be seen on table 3.1. Here it can be seen how the weighted average updating method only estimate the intermediate reward. If applying a greedy policy on the Q-table the agent will always start by moving to the right into the bad part of the environment. To avoid the scenario described above the updating methods needs to look at the cumulative reward instead of the intermediate reward.

Weighted Average	Left	Right
S0	-4	1
S1	3	5
S2	5	4
S3	4	3
S4	3	3
S5	-3	-1
S6	-2	-3
S7	-1	-2
S8	-1	-1

Table 3.1: Q-table values after running 100000 episodes using the weighted average updating method, using a decreasing ϵ-greedy policy and a step size equal to 0.7.

Both SARSA and Q-Learning are updating methods which utilises the discounted cumulative reward. Both algorithms can be seen respectively at eqs. (3.7) and (3.8).

SARSA

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right] \tag{3.7}$$

Q-Learning

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

As seen on the equations both methods introduce a new element in the the calculation of the loss. Both methods now include the next estimated action values. SARSA chooses the next estimated action value based on its policy, this is also known as on-policy learning, whereas Q-Learning take the highest estimated action values which is an off-policy learning method. SARSA and Q-Learning is equal if both methods uses a greedy policy. Using both methods on the same example seen on fig. 3.3 it can be observed that they reach different estimated action values compared to the weighted average method. These values can be seen of table 3.2. First thing which differs is that both SARSA and Q-Learning will choose to go left if a greedy policy is applied. From this it can be seen how both methods take future rewards into consideration. Comparing SARSA and Q-Learning it can be observed that they will reach the same choices if a greedy policy is applied, even though the values in the tables differ. This may not always be the case as SARSA uses its own policy instead of the best estimated value for the next estimated action value. If the policy includes some kind of exploration, like the ϵ-greedy policy, it may find that the risk of taking an exploratory move when moving in the wanted direction outweighs the rewards it can get by going that direction. Due to this SARSA will often find a safer route than Q-Learning if exploration is included in the policy [24, p. 132].

SARSA	Left	Right
S0	1.78	-1.13
S1	3.60	8.26
S2	16.00	8.22
S3	4.26	3.73
S4	9.32	9.76
S5	-5.68	-3.04
S6	-3.79	-5.15
S7	-1.70	-4.53
S8	-4.43	-3.30

Q-L	Left	Right
S0	1.78	-1.13
S1	5.24	8.26
S2	16.00	12.24
S3	14.18	3.52
S4	9.33	10.39
S5	-5.92	-3.04
S6	-3.87	-4.96
S7	-1.55	-3.99
S8	-3.21	-2.92

Table 3.2: Q-table values after running 100000 episodes using SARSA and Q-learning, using a decreasing ϵ-greedy policy, a step size equal to 0.7 and a forgetting factor equal to 0.7.

3.4 Analysis of Tabular Methods

Depending on the application there are many different ways to setup the state and affect the state space. For this project different parameters can be used to construct the state such as earlier actions taken by the agent, the position and orientation of the UE.

Combination of these parameters can be used to obtain different state spaces. The goal is to find a state space which has enough information to find a correlation between the states and the received rewards. Each parameter can be represented in different ways e.g. the number of stored earlier actions, positions and orientations. Not only the number of earlier parameters stored in the state but how they are presented can also be adjusted. Looking at the position of the UE it can be represented in many different ways. One way is to use the position to calculate the distance of the UE to the BSs and then discretise it to closest 10 m. This gives less information into the algorithm but reduces the state space. This can be very useful as the agent needs to visit fewer states to be able to construct a table that has converged. This also impact how fast the agent can react to changes in the environment.

But as discretisation is used information is lost so its important to ensure that the state space stays correlated to the reward. This can also be affected by how the reward is constructed. For this project the reward is based on the system model eq. (2.1) which is correlated to the distance and orientation of the UE from the channel model. This means that if only earlier actions are used as the state then the reward can become very different for the same state depending on whether the UE is moving close or further away from a BS. As the same agent is used for different random walks, what it has learned from one walk may no longer be applicable as the walk may start in a new area where the reward is different. This problem can be avoided to some degree by including the position of the UE depending on the discretisation of position or it can be alleviated to some degree by normalising the reward based on the distance to the BS.

In this project the case with four BSs and a single UE is used. This means that if each BS and the UE each has eight different beams the total number of actions, if a centralised agent is being used, becomes $4 \cdot 8 \cdot 8 = 256$. This has to be multiplied to the state space to get the size of the table. So if the distance to the BSs is discretised to be integers with meters as unit and the maximum distance is said to be 200 m then the total table becomes $256 \cdot 200 \cdot 4 = 204800$. And as mentioned earlier orientation could also be used and if only integers with unit degree the total table becomes 360 times bigger. So adding few parameters to the state will affect the size of the table exponentially.

Due to the carrier frequency being transmitted at 28 GHz which corresponds to a wavelength of 1.07 cm the characteristics of the impending wave can change drastically by moving only a couple of centimetres as described in section 1.1. Therefore if it is wanted to include the position while still having a correlation in the state space a dense descretisation is needed. The above example already gave a huge state space where the distance was descretised into meter units so by further increasing resolution of the descretisation the state space will very fast become enormous. Because of this and the restriction it causes it has been chosen to investigate further into function approximation methods.

3.5 Function Approximation

There exists a lot of different function approximations methods and the taxonomy on fig. 3.2 shows only some of the approximation methods which exist. To narrow it further down on which function approximation methods are suited for this project a taxonomy from OpenAI [26] will be used as this taxonomy split the approximation methods into further categories. The taxonomy can be seen on fig. 3.4.

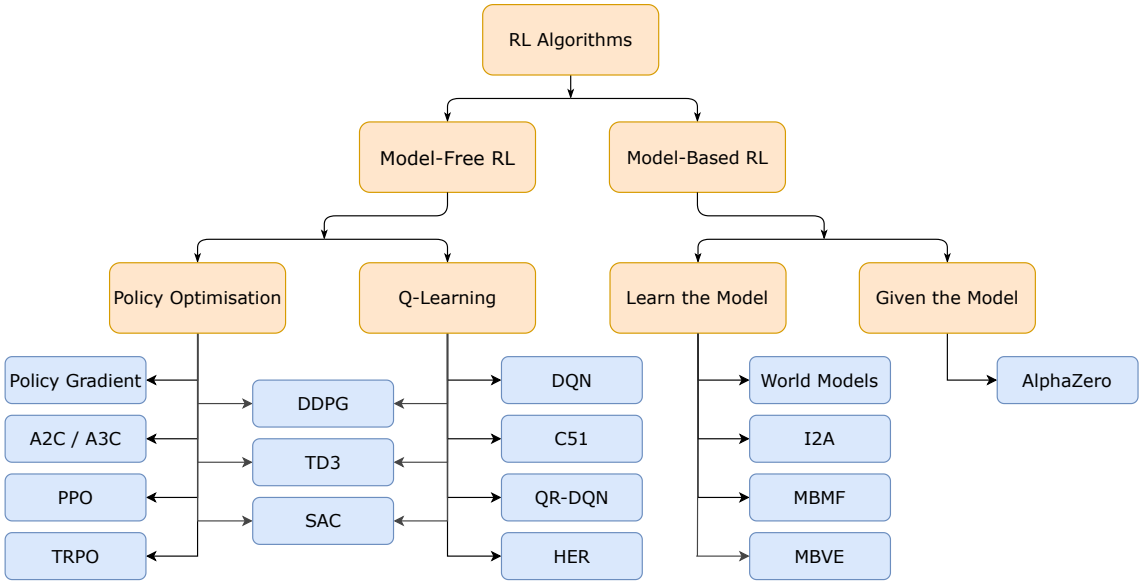


Figure 3.4: Taxonomy of algorithms used in RL [26]

It should be noted that there are many ways of categorising the algorithms but this taxonomy shows some of the most fundamental design categories and what algorithms fit into them. Beginning from the top the model-free RL and model-based RL will be discussed first.

3.5.1 Model-Free and Model-Based

When referring to a model-free RL the agent does not learn a model of the environment. This is not the same as saying no model is used as the action choices are based on a model, but non is implemented to model the environment [26].

On the other hand the model-based RL does use a model of the environment which can be used to predict state transitions and rewards. Some model-based methods need to learn the model whereas others are given a model. Compared to the model-free RL the agent should also find suitable Q-values but also the transition probability if not already given. This can be very useful as it is possible to explore potential future path based on this probability. This can then be used in the equation to update the Q-table without actually getting samples from taken actions in a given state making it more sample efficient. This makes it very fast to readapt if the reward function should change and the environment stays the same. An example where model-based RL is usable is for the game of chess where the agent is fully aware of the rules so it doesn't even need to learn the environment model and can therefore plan future action paths. Even though model-based RL has a lot of advantages over model-free RL it comes with some restrictions. It must be possible for a RL agent to learn a model of the environment. When analysing the environment described in section 1.3 there are some things which has the consequence that a model-based RL agent cannot be used. Here the environment is changing so trying to estimate the environment will be an ongoing task and if a lot of changes is happening it might not be able converge. Also the reward in this environment is based on the position and orientation of the UE and the choice of beam direction. The position and orientation cannot be controlled by the RL agent and is random to a certain degree and thus it is not possible for the agent to plan into the future. The RL

agent has no knowledge and cannot predict where the user will go, especially in the case when it is a pedestrian. It can be argued for that it is possible to predict future actions when the UE is a car as it will not move as random as a pedestrian. Because of these arguments the RL algorithms will be limited to model-free RL methods.

3.5.2 Policy Optimisation and Q-Learning

Q-learning is a value based method. Value based methods estimate either how good it is to be in a certain state or how good it is to take an action from its current state. Q-learning do the last one and based on this it takes an action. This can be seen in section 3.3.1 where the Q-learning algorithm is described. Through the example in that section it can be seen that the estimated value of a state is dependent on the discounted cumulative reward. So even though an action leading to a state gives a bad reward the Q-value of the state can be high as it leads to new states where it can get good rewards. By doing this these methods indirectly learn the transition probabilities from a state to an action (see eq. (3.3)). Policy optimisation methods differ from value based methods as these method directly learn the transition probabilities. These methods take the state as an input and outputs an action, where the action is stochastic [26].

Comparing these two type of methods, policy optimisation have the advantage that it can operate on larger and more complex action spaces. One example would be a continuous action space, here value based methods need to discretise the action space as these methods estimate how good the state-action pair is. Due to needing to discretise the action space a loss is introduced and it is not possible to remove the loss as it will require infinite state-actions. Policy optimisation methods also tend to be more stable and reliable as they directly optimise the policy. Value based methods has the advantage over policy optimising method that they are more sample efficient because they can reuse date more effectively.

Comparing these methods with the described environment in section 1.3 the first thing to evaluate is the action space. In this environment the action space is the combination of beam direction pairs between the BS and UE and therefor the action space is not continuous. This means that either method can be used without utilising discretisation. So the question is whether a more stable and reliable or a sample efficient method is wanted. Value based methods can still be stable and reliable they just tend to be less than the policy methods. Due to this value based methods have been chosen as they are more sample efficient.

3.5.3 Value based methods

There exist a lot of different function approximation value based methods; some are an extension to the already mentioned tabular methods while others uses a new approach. From the methods which are an extension of the tabular methods are the category of the methods called coarse coding.

Coarse coding methods use a table but instead of updating only one entry at a time it updates multiple. The different coarse coding methods differ in what method they use to update several entries at a time. Coarse coding method suffers for some of the same limitations as the tabular methods. Due to updating several entries at a time it has a faster convergence rate and can now estimate values for states it has not visited before. For this reason these methods

can handle larger action-state spaces but like the tabular methods it uses a table to store its estimated values and therefore there becomes a limitation on how large the table can get.

Some function approximation methods which take a new approach are combining the concept behind RL with a NN. So instead of having a table to store its estimated action values, it trains a NN to find its estimated action values based on a state as input. Due to this RL combined with a NN can handle a huge action-state space. By making the NN deeper it is often possible to increase the accuracy but it also takes longer time to run due to the NN becoming more complex. This methods also suffer from all the problems a normal NN has like overfitting which occurs when the NN has been fitted to the training set that it cannot generalise to other similar data sets. This is also one of the reasons it is a challenge to design a NN.

Because a NN can handle a large action-state space and because the group behind this project have some prior experience with NNs it has been chosen to use RL combined with a NN to solve problem. More specifically it has been chosen to use DQN as the project group has prior experience and knowledge about this method. It may not be the best algorithm for this project but based on the arguments in this chapter it has been found suitable. A more thorough analysis and testing is required to find the best algorithm but this is not the goal for this project. A deeper analysis of the DQN algorithm can be found in the following chapter.

Deep Q-Network 4

This chapter analyses the DQN algorithm with the goal of being able to find and understand the algorithm's methods and their hyper parameters. This should lead to an implementation of the algorithm which will be tuned in the following chapters.

DQN is a deep RL method which combines Q-learning with a NN. A NN is a structure which mimics the way that biological neurons signals to one another in a human brain [27]. An example of a model of NN can be seen on fig. 4.1.

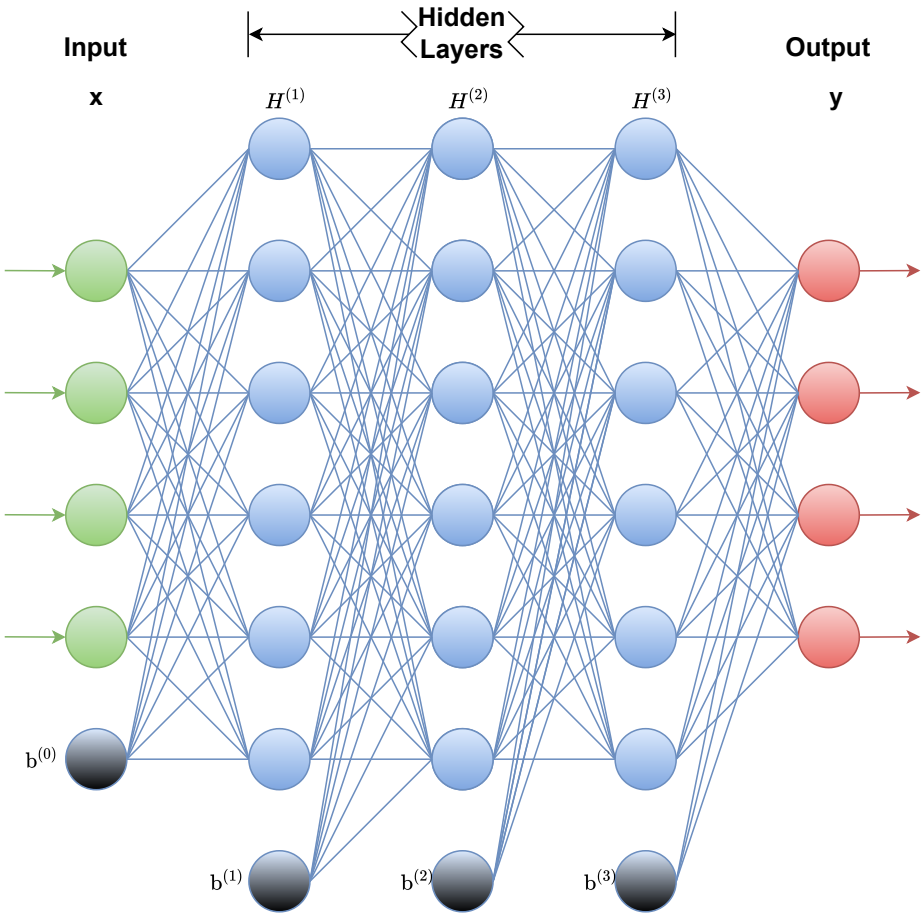


Figure 4.1: Model of a fully connected feed-forward network.

This is known as a feed-forward network as the information only moves in one direction as there is no feedback in the system. It is fully connected as each node is connected to every single node at the next layer. There exist other ways to implement a NN but due to time constraints it has been chosen to use the feed-forward network as the one on fig. 4.1, as its a

network that is simple to implement.

The figure contains three elements - an input, hidden layers and an output. When a NN contains multiple hidden layers it is also known as a deep NN. A NN takes an input vector whose length correspond to the number of nodes in the input layer. Most of NN are based on a chain structure with each output of a layer is fed into the next layer [28, p. 164]:

$$\mathbf{y} = f(\mathbf{x}) = f_{N_L}(\dots f_2(f_1(\mathbf{x}))) \tag{4.1}$$

where

N_L: Number of hidden layers
f_x: Function of layer x

Then the value of a node at each layer is calculated as an affine combination of the values of all neurons in the previous layer and is then applied to an activation function. This can be seen on fig. 4.2 which shows how each of the nodes in a previous layer is multiple with a weight (w) and a bias is added (b) which results in the variable z. z is then used as the input to the activation function (the node). For this project it has been chosen to use the hyperbolic tangent function ($tanh$) as the activation function. There exist a lot of other activation function but it has been chosen to use $tanh$ to limit the number of hyper parameters which needs to be tuned.

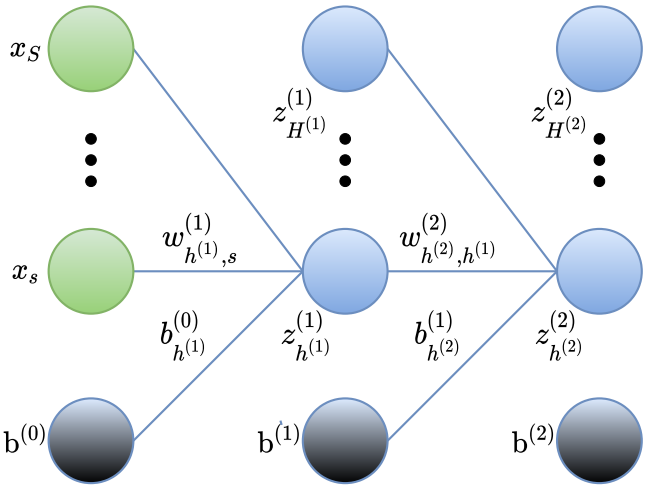


Figure 4.2: Model of a fully connected feed-forward network.

The first layer is calculated by:

$$\mathbf{h}^{(1)} = f_1(\mathbf{x}) = g(\mathbf{z}^{(1)}) = g^{(1)}(\mathbf{w}^{(1)}\mathbf{x} + \mathbf{b}^{(0)}) \tag{4.2}$$

where the second layer is calculated by:

$$\mathbf{h}^{(2)} = f_2(\mathbf{h}^{(1)}) = g^{(2)}(\mathbf{w}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(1)}) \tag{4.3}$$

and so on. where

Superscript (x):	Corresponds to the parameters in the x layer
g:	Activation function
$\mathbf{w}$:	Weight vector
$\mathbf{b}$:	Bias vector
$\mathbf{x}$:	Input vector.

When an input has propagated through the network and generated an output a loss will be calculated. How the loss is calculated is what differentiate different machine learning categories from each other. In supervised learning the loss will be calculated from the difference between the output and the labelled data. In unsupervised learning the loss can be calculated in a lot of different ways depending on the usage of the NN. The loss could be the distance between output and the center of a cluster. Lastly in deep RL the loss will be based on the reward it receives for taking an action. In supervised and unsupervised learning all outputs may be used but in RL only one of the outputs will be used as the number of outputs corresponds to the number of actions it can take. When the loss has been calculated it can then be used to update the weights in the NN.

The input and output layers corresponds respectively to the number of states and actions the agent has. The hidden layers controls how many weights the NN has and how deep it is. How many hidden layers are needed and the size of the individual hidden layers and the design process will be looked into further in the following chapters. First it will be looked into how the loss is calculated for the DQN RL algorithm.

4.1 Loss Function

In section 3.3.1 the Q-learning algorithm is described and the algorithm can be found on eq. (3.8). In that equation the loss is calculated as the error between predicted value and the actually value. This can also be seen on eq. (4.4).

$$\begin{aligned} L_Q &= Q_{obs} - Q_{predicted} \\ Q_{obs} &= R_{t+1} + \gamma \max_a Q(S_{t+1}, a) \\ Q_{predicted} &= Q(S_t, A_t) \end{aligned} \tag{4.4}$$

This loss function could also be used in a NN but this is not very common. It is more common to use either Mean Square Error (MSE), Mean Absolute Error (MAE) or Huber Loss [29]. These loss functions all have a global optimum and no negative loss value. MSE penalise outliers as the error is squared. MAE penalise them less as all error values are weighted linearly. Huber Loss is a combination of the two methods by introducing a new variable δ. If the absolute error value is less than δ it calculate the squared error like MSE and if greater than δ it calculate the absolute error like MAE. The equation for the different methods can be

seen on eqs. (4.5), (4.6) and (4.7) and a figure of them can be seen on fig. 4.3

$$L_{MSE} = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2 \tag{4.5}$$

$$L_{MAE} = \frac{1}{N}\sum_{n=1}^{N}|y_n - \hat{y}_n| \tag{4.6}$$

$$L_{Huber} = \frac{1}{N}\sum_{n=1}^{N}\begin{cases}\frac{1}{2}(y_n - \hat{y}_n)^2 & \text{for } |y_n - \hat{y}_n| \leq \delta, \\ \delta|y_n - \hat{y}_n| - \frac{1}{2}\delta^2 & \text{otherwise}\end{cases} \tag{4.7}$$

where

N: Number of samples
y_n: True value corresponding to Q_{obs}
$\hat{y}_n$: Estimated value corresponding to $Q_predicted$

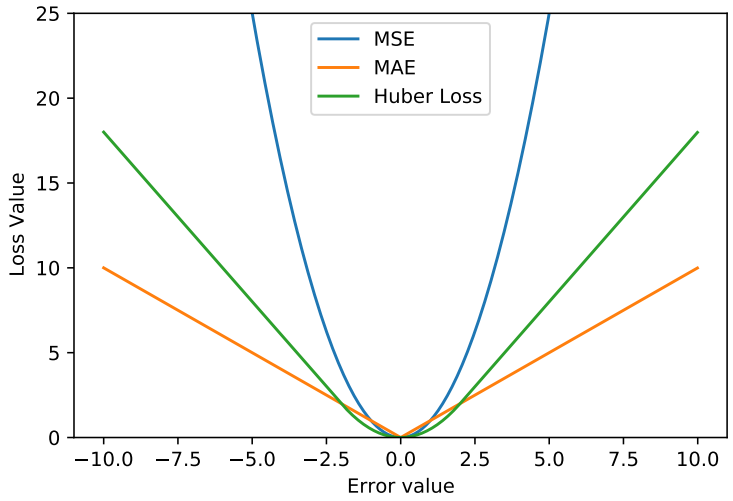


Figure 4.3: Loss functions for MSE, MAE and Huber Loss with $\delta = 2$.

Comparing these methods with the goal of this project it has been chosen to use MSE as the loss function. This is due to the consequences it will have when the agent chooses a wrong action due to a bad estimate. A bad choice of action could result in a drastically drop in the received signal power and may even result in a lost connection. Due to this it is wanted to penalise a high error value a lot. When combing the MSE loss function with Q-learning it result in the following equation:

$$L_{Q,MSE} = \frac{1}{N}\sum_{n=1}^{N}\left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right)^2 \tag{4.8}$$

As this loss function should be used in a NN the Q-values should be taken from the output of the NN therefor the rewriting loss can be seen on eq. (4.9). The Q-values are now dependent on the weights in the NN.

$$L_{DQN,MSE} = \frac{1}{N}\sum_{n=1}^{N}\left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta) - Q(S_t, A_t; \theta)\right)^2 \tag{4.9}$$

where:

θ: Weights in the NN

4.2 Updating the weights

The next step is to update the weights based on the calculated loss value. Updating the weights can be done in several ways, some of the most common methods is using Gradient Descent (GD) or Stochastic Gradient Descent (SGD).

In this project it has been chosen to use TensorFlow[30], which is a popular python package that includes different tools to implement a NN. TensorFlow has implemented several methods for updating the weights [31] and between these methods it has been chosen to use the ADAM algorithm [32][33].

ADAM is a SGD optimising method that is based on adaptive estimation of first-order moment which is the expected value and second-order moment which is the variance. ADAM has been chosen due to it being a computationally efficient algorithm. The ADAM optimising algorithm requires the derivative of the loss function with respect to the weights in the NN. One of the most efficient ways of finding these gradients is to use Automatic Differentiation (AD) which is also what TensorFlow uses [34][35]. The AD techniques is based on the decomposition of the differentials provided by the chain rule. AD can be split into two categories forward accumulation and reverse accumulation. A special case of reverse accumulation is the well known backpropagation method which is often used in machine learning [36]. TensorFlow utilise reverse accumulation to find the gradients. With this it is possible to find the gradients of the NN based on the calculated loss and then updating the weights to minimise the loss.

This makes it possible to construct a deep RL algorithm but as mentioned it is wanted to implement DQN which includes a couple more methods to make it more stable and to be able to better utilise its experience.

4.3 Target Network

To improve the chance of convergence a target network is introduced [37]. The deep RL algorithm now uses two networks, one to predict what action is the best and the other target network is only used when calculating the loss value as seen on eq. (4.10).

$$L_{DQN} = \frac{1}{N}\sum_{n=1}^{N}\left(R_{t+1} + \gamma \max_{a} Q(S_{t+1}, a; \theta_{target}) - Q(S_t, A_t; \theta_{pred})\right)^2 \tag{4.10}$$

where

θ_{target}: Weights in the target NN.
θ_{pred}: Weights in the NN which predicts the action.

Both the predict and target network use the same NN structure although with different values for their trainable weights. The weights for the predict network are updated after each sample whereas the weights for the target network are only updated after a certain number of

samples. When updating the target network its weights become equal to the predict network. The number of samples the target network's weights are frozen for is a hyper parameter which needs to be tuned depending on the case. There are cases where it has been frozen for up 10.000 samples [37].

By freezing the target network for multiple steps it prevents the training of the NN from becoming unstable. It comes at a cost of a slower reaction time as the target network only updates after a certain number of samples. The longer the target network is frozen the more stable it becomes but is also becomes slower, therefore it is important to tune the hyper parameter.

4.4 Experience Replay

Experience replay is a method that utilises a replay buffer of the previous experiences of the agent including (S_t, A_t, R_t, S_{t+1}) which often works as a First in First out (FIFO)-buffer and is updated each time an action is taken [38]. Experience replay then take samples with a fixed size and stores them in a batch from the replay buffer and uses it to update the NN. Which samples to pick from the replay buffer is often based on a uniform distribution, but an algorithm can also be used which exponentially is more likely to chose earlier experiences [39]. Using experience replay means that instead of a single action-state pair the update is now based on a batch of the experience. This means samples are used multiple times. This has the advantage that it usually becomes more efficient and converges faster which can be very useful, if getting samples are very costly. A condition that has to be fulfilled is that the environment must not change rapidly. But if the environment only changes a little, using an algorithm that are more likely to choose newer samples from the batch can be useful. The reason for this is that the longer time the samples are taken apart the more uncorrelated they become with the environment if the environment is not stationary. The cost of using experience replay is only the usage of extra memory.

4.5 Test and Design Procedure

The next step is then to test all the hyper parameters to see how good the DQN can be in the constructed environment. An overview of the hyper parameters can be seen on table 4.1. As seen on the table there exist a lot of different hyper parameters and most of them are unbounded which means there exist infinity possibilities. Therefore it is not possible to do a sweep over every parameter and the combination of them.

	Hyper parameter	Value Space
DQN	Hidden Layers	$([1,\infty[_1,\cdots,[1,\infty[_n), n = [1,\infty[\in \mathbb{N}$
	Learning Rate (ADAM)	$]0, 1] \in \mathbb{R}$
	Memory Size	[Batch Size, $\infty[\in \mathbb{N}$
	Batch Size	[1, Memory Size[$\in \mathbb{N}$
	Target Update	$[1, \infty[\in \mathbb{N}$
RL	Forgetting Factor (γ)	$]0, 1] \in \mathbb{R}$
	Exploring Factor (ϵ)	$]0, 1] \in \mathbb{R}$
	State Space	Undefined

Table 4.1: Hyper parameter for the RL agent based on DQN.

To be able to find a suitable DQN for the RL agent a structured approach is needed to tune these parameters. As seen on the table the hyper parameters has been split into two categories; DQN parameters described in chapter 4 and RL parameters described in chapter 3. It is worth noticing that they cannot be completely separated as every parameter has an impact on each other. For this project is has been chosen to design the parameters described in chapter 4 (DQN) first. This has been chosen due to the importance of the state space. The state space will contain the information about the environment and depending on what is included in the state space the RL agent will learn different things. The following two chapter will go through this process. Chapter 5 will go through the process of finding the parameters found in chapter 4 and chapter 6 will go through to the process of tuning the remaining parameters.

Design of Neural Network 5

This chapter will look into tuning the parameters described in chapter 4 summed up in table 4.1. As it is impossible to do exhaustive searching of all the parameters and their combination to find the best parameter setting another approach will be taken. One parameter will be tuned at a time while keeping the other parameters constant. This can then be repeated to find a local optimum of the parameters. Due to time constraints it has been chosen to only do one iteration of the tuning of the parameters.

For tuning the parameters it has been decided to optimise them in the following order.

1. Hidden Layers
2. Batch and Memory Size
3. Target Update
4. Learning Rate

Batch size and memory size has been combined as they are interconnected with each other.

5.1 Initial Values

To start the tuning of the structure of the hidden layers an initial value for all the other parameters needs to be chosen. Batch size influence how much data it will be trained on for each step whereas memory size control how long a sample will be saved. The environment the agent is in can change quite fast and due to this a large memory size is unwanted as it makes it possible for the NN to be trained on data which is no longer relevant. Therefore it has been chosen to initialise the batch and memory size to be 500 and 1000 respectively which correspond to 5 s and 10 s respectively with the chosen sample period.

How often the target network should be updated is another parameter which influence the stability of the NN but also its response time. Updating the target network often may make it less stable but make it more responsive to changes. As mentioned before the environment the agent is non-stationary and therefore it has been chosen to set the initial value of the target update to ten.

Lastly is the learning rate parameter which the ADAM optimising algorithm uses. This parameter controls how much the NN weights should be updated based on the calculated loss. The learning rate can be compared to the step size when using SGD. A high value (close to 1) will make the system very reactive to changes but becomes difficult to reach an optimum as it may step over it. By making the value lower it becomes less reactive to changes and as a result it can with more accurate find an optimum. Due to this it has been chosen to set the learning rate equal to 0.01.

The parameters used in the RL also need to have an initial value to be able to use DQN. The forgetting factor (γ) controls how important later rewards are. Due to the environment being non stationary it is not wanted to include rewards far into the future as they may no longer be relevant. Therefore is has been chosen to initialise the forgetting factor to be equal to 0.7. For the exploration and exploitation factor (ϵ) it has been chosen to set it equal to a constant with a value of 0.01. It could have been set to vary depending on other parameters. In a stationary environment it could have been wanted to have a decreasing epsilon value so it explore a lot in the start and stops after a certain time. This is not wanted in non-stationary environments as here it should never stop exploring. The exploring factor could also be depending on the loss value. This makes it a little more complex as it needs to be determined how it should depend on it. Should it depend linearly on it or should the loss values be smoothed before it is used. Due to it being a bit more complex it has been chosen to use a constant value.

Lastly is the state space and to keep it simple it has been chosen that it should represent most of the knowledge we know about the UE. This will include the position (x, y), orientation (θ) and its action (a). It has also been chosen that it should include some history of its position, orientation and actions. An overview of all the chosen initial values can be seen in table 5.1.

	Hyper parameter	Value Space
DQN	Hidden Layers	To be tuned
	Learning Rate (ADAM)	0.01
	Memory Size	1000
	Batch Size	500
	Target Update	10
RL	Forgetting Factor (γ)	0.7
	Exploring Factor (ϵ)	0.01
	State Space	$[a_{-1}, a_{-2}, a_{-3}, \theta_0, \theta_{-1}, \theta_{-2}, x_0, x_{-1}, x_{-2}, y_0, y_{-1}, y_{-2}]$

Table 5.1: Initial hyper parameter for the RL agent based on DQN.

5.2 Test preparation

Before testing can be started a training data set needs to be created and which metrics the results should be compared to also needs to be decided. Noise will not be accounted for when tuning the parameters. The reason for this is that the considered noise is a Additive White Gaussian Noise (AWGN) so adding it should not affect the tuning of the NN. Due to it being AWGN there are no correlation in noise and therefore the NN cannot learn the characteristics of the noise thus cannot learn to filter it away. The noise may have some impact on the other hyper parameter tuned in this section but it has been assumed that it will not have as big an impact on these hyper parameter as the RL hyper parameters. Therefore noise will will not be added when tuning these hyper parameters and when tuning the RL parameters in the next chapter the noise will be accounted for and the implementation will be discussed.

As noise is not added when tuning these parameters the value of the transmitted power (p_t) used in the signal model (eq. (2.1)) is just an arbitrary value which scales the rewards. To keep consistency between tuning of the DQN parameter and the RL parameters the transmitted

power will be set to 30 dBm based on simulation parameters used in [40, p. 183].

5.2.1 Training Data set

Regarding the data set there are four possible scenarios - *Pedestrian* and *Car Urban* where either LOS or NLOS is utilised. The parameters for these cases can be seen in table 2.1. To keep it simple it has been chosen to compare the results of the tuning of the individual hyper parameters to only one case and it has been chosen to use *Urban Car - LOS*. The *Car* scenario has been chosen due it to having a longer distance between samples compared to the *Pedestrian* scenario. This is assumed will lower the correlation between samples and thus make it harder for the NN to find the correlation. If it can get good results for this scenario it is assumed it can be quite easily achieve similar results in *Pedestrian* as it has a better correlation between samples. LOS has been chosen due to it being a more simple case and it is wanted to first see whether it is possible to make a good DQN for this environment before going to the more complex NLOS environment. Tuning the DQN on only one scenario will create some bias towards it. This is found acceptable as it is only wanted to see if it is feasible to use RL in this environment. Due to time constraints it is also not possible to tune it against all scenarios.

After the tuning of the DQN hyper parameters the model will be used on every scenario to see how well it performs. These results will also be used as a reference point to see how much the tuning of the RL parameters can improve the performance. The generated training data set for *Car Urban - LOS* scenario and be seen on fig. 5.1.

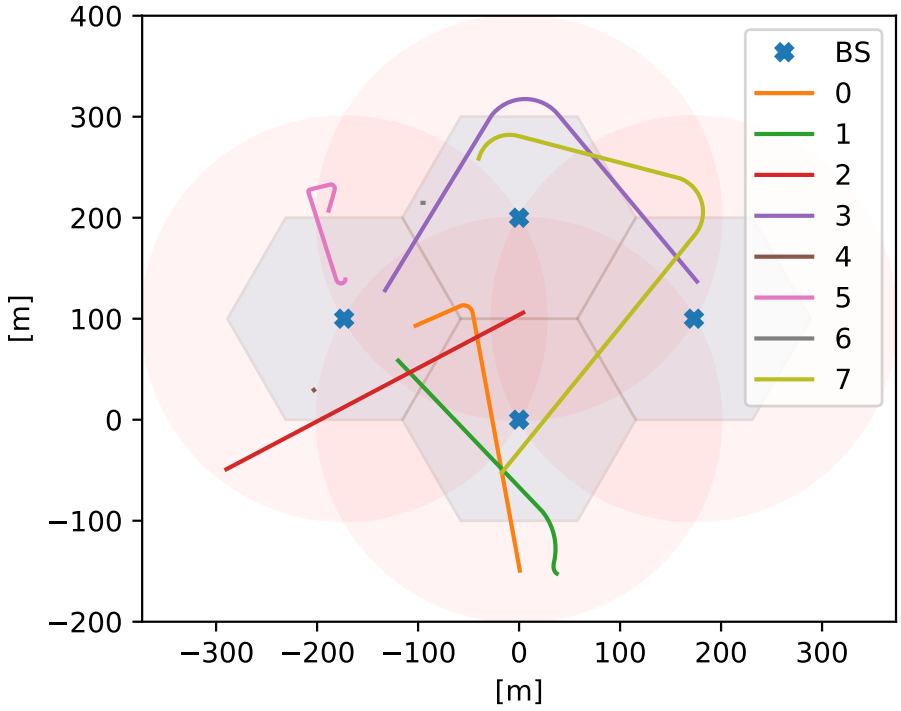


Figure 5.1: Generated data set for eight different users in the same environment made with the parameters in table 2.1. Each user takes 10000 steps. Path 4 is a small dot around [-200, 50] as it is not moving very much. Likewise path 6 can be seen around [-80, 200].

Eight different users have been generated which drives around inside the environment. Each user takes 10000 steps which results in a total of 80000 different samples. As it takes a lot of time to generate the data sets and it requires a lot of available RAM to generate multiple paths in the same environment it has not been possible to generate bigger data sets in the same environment. Therefore to increase the data the NN can be trained on the generated data set will be reused. Instead of using the whole path (10000 steps) a smaller chunk size will be chosen. The chunk size determines how long an episode will be. Instead of having only eight different paths it is possible to having more slightly different episodes by splitting the paths into overlapping chunks as seen in fig. 5.2.

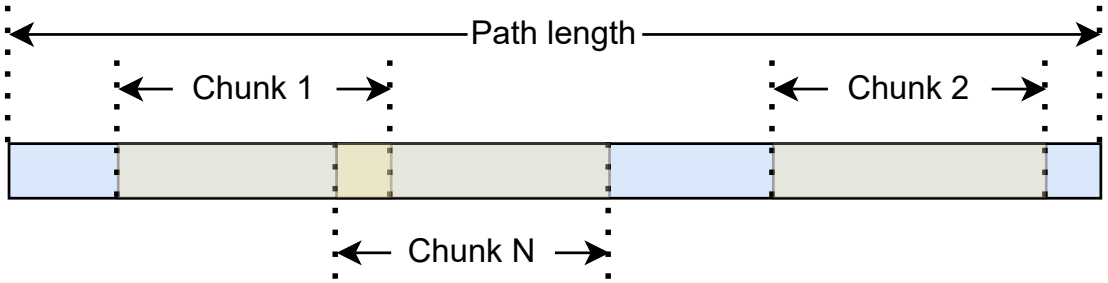


Figure 5.2: Random chunks taken from a path.

To ensure that each test can be compared to each other they will be run on the same chunks which will be drawn from a uniform distribution between all the paths.

Choosing a larger chunk size will ensure that more of the same samples are present at each path. So to keep the episodes long enough and to not use too many of the same samples in every path it has been chosen to set the chunks size equal to 5000.

Based on tests done when creating the simulation tool if was found that for smaller NN it would take around 4-5+ hours whereas for larger it could take over day to complete a simulation with the given hardware when running a total of 1,000,000 steps (2 h 46 min 40 s of driving). The current used hardware does not have a GPU and thus it could have been done a lot faster but it would then not be possible to have multiple instances to do parallel testing. Due to time constraints it has been chosen to simulate 1,000,000 steps which corresponds to 200 episodes. It is assumed 200 episode is enough to see the tendencies of the hyper parameters.

5.2.2 Validation

Regarding the validation it has been chosen to look at two metrics. The first metric is the loss value which the DQN uses to update its weights (see eq. (4.10)). The other metric is the misalignment power loss which will just be referred to as misalignment. The misalignment value will be calculated as the maximum possible reward subtracted with the taken reward as seen on eq. (5.1). The loss value will be used to determine how well the agent adapt to changes whereas the misalignment metric will be used to determine how good the agent is at taking the best choice.

$$mis_t = R_{t,max} - R_{t,taken} \tag{5.1}$$

where

t:	time step
mis_t:	misalignment at time step t
$R_{t,max}$:	Maximum possible reward at time step t
$R_{t,action}$:	Taken reward at time step t

For the loss values there will be a visual inspection of the plotted values to ascertain how fast the agent adapt to changes. There will also be a numerical value which represent the average loss to determine how much the agent's estimated guess differs from the actually values on average.

Misalignment will likewise be inspected visually to ascertain how the misalignment changes over time. The numerical value will be the total average of all the misalignment values to determine how good the agent is in generally at taking the best choices. The misalignment values used in figures will be the average of how close the chosen action's reward is to the maximum reward over the latest 1000 steps (see eq. (5.2)) to make it easier to do a visual inspection of the results.

$$mis_{t,avg} = \frac{1}{1000} \sum_{j=\max(t-1000,0)}^{t} mis_j \tag{5.2}$$

$mis_{t,avg}$: Average misalignment for the latest 1000 steps at time step t

The graphs of the results will also show a smoothed version using an exponential smoothing function with a weight equal to 0.99999. This has been done to make it easier to see the characteristics of the results.

5.3 Hidden Layers

To further limit the search space of number of hidden layers and the sizes of them it has been chosen not to look at NN which are too big. This is due to RL being an ongoing process where the reference for which the estimated value is going to be compared to always changes. Therefore if the NN becomes too big it may overfit and its reaction to changes also slows down which is unwanted.

As mentioned in table 2.1 the environment consist of four BSs and one UE. The BSs each have 16 beams whereas the UE has 8 beams. This results in a total of $4 \cdot 16 \cdot 8 = 512$ different possible actions the agent can take. This means that the output layer will have the same size. The size of the input layer will be equal to state space which for now has been set to 12.

Due to the available computer resources the tests will be done on batches of seven. The number of hidden layers and their size can be seen in table 5.2. The table also show the

results of the tests in numerical values. The full test can be found in appendix A which shows more extensive results.

Test No.	Hidden Layers	Avg. Loss	Avg. Misalignment [dB]
1	[100, 100]	16.117	7.076
2	[200, 200]	16.744	7.265
3	[512, 512]	21.613	7.425
4	[512, 512, 512, 512, 512]	29.454	7.972
5	[500, 1000, 1000, 512]	27.399	7.636
6	[500, 1000, 1500, 1024]	34.937	8.683
7	[500, 1000, 1500, 2000]	75.002	10.723

Table 5.2: Test set 1. Results for testing different hidden layers.

From the first test it can be observed through the average loss and misalignment that the smallest hidden layer achieved the best results on both parameters. When inspecting the graphs for all the results it can be observed that using bigger hidden layers the loss graphs contains more spikes compared using a smaller hidden layer. This can be due to the NN using bigger layers are having a harder time reacting to changes compared to using smaller hidden layers. Due to the results of the tests showing the best performance for smaller layers a new test will be conducted. The new tests will explore hidden layers with sizes around and smaller than the ones used in test one. The number of hidden layers and their size can be seen in table 5.3 and likewise the full test can be found in appendix B.

Test No.	Hidden Layers	Avg. Loss	Avg. misalignment [dB]
1	[20, 20]	16.673	7.480
2	[50, 50]	16.311	6.501
3	[50, 100]	15.262	6.962
4	[20, 30, 40]	18.091	7.394
5	[100, 150]	16.186	7.416
6	[150, 150]	16.638	7.317
7	[75, 100, 125]	18.106	7.220
Prev. Best	[100, 100]	16.117	7.076

Table 5.3: Test set 2. Results for testing different hidden layers.

From the results it can be observed that the best hidden layer combination is [50, 50]. It could be further tested to see if a better combination around [50, 50] exists but it has been chosen to go with [50, 50]. This is due to time constraints and an assessment on to improve the results a lot more it is needed to optimise the other hyper parameters.

5.3.1 Embedding Layer

Due to the way the actions are represented it has been speculated whether an embedding layer should be added. The actions the agent can take is the combination of choosing a BS, choosing which beam to transmit with and lastly which beam the UE should receive the signal with. This gives a total of $4 \cdot 16 \cdot 8 = 512$ different indexes the agent can choose. Some of these indexes may have some similarities between them which can give the NN extra information

and a way to find these similarities is by using an embedding layer. An embedding layer converts a vector of indexes to a dense vector and through this process it is possible to find extra information in the input [41]. Embedding layers are often used in NN where the input are words in a specific language and the goal is to find connections between words. This introduces a new hyper parameter N_{EL} which determine the output dimension of the dense vector which is the output of the embedding layer. The test will be done on the setting that gave the best results from the hidden layers in table 5.3 ([50, 50]) to see if an improvement will be made by using embedding layer and changing the size of the output dimension. The results of the tests can be seen in table 5.4 and the whole test can be found in appendix C.

Test No.	Output dim.	Avg. Loss	Avg. Misalignment [dB]
1	1	18.529	7.236
2	2	18.445	7.044
3	3	18.695	6.677
4	4	18.878	6.937
5	5	17.367	7.114
6	6	19.040	7.255
7	7	19.221	6.995
Prev. Best	0	16.311	6.501

Table 5.4: Test set 3. Results for testing embedding layer with different sizes of the dense vector

Comparing the result with the use of embedding layer with the prior test without the embedding layer it can be observed that adding it does not increase the performance. Therefore it has been chosen to not to include an embedding layer.

5.4 Batch Size + Memory Size

With the found size of the NN the hyper parameters becomes the following:

	Hyper parameter	Value Space
DQN	Hidden Layers	[50, 50]
	Learning Rate (ADAM)	0.01
	Memory Size	To be tuned
	Batch Size	To be tuned
	Target Update	10
RL	Forgetting Factor (γ)	0.7
	Exploring Factor (ϵ)	0.01
	State Space	$[a_{-1}, a_{-2}, a_{-3}, \theta_0, \theta_{-1}, \theta_{-2}, x_0, x_{-1}, x_{-2}, y_0, y_{-1}, y_{-2}]$

Table 5.5: Hyper parameter for the RL agent based on DQN for tuning the batch and memory size.

The batch size affects how many samples should be run through the NN using the updated weights where the loss used to update the weights will be an average over all the losses from the batch. A high batch size will make the loss value more stable as it will be an average over

many samples. The can make the NN more stable but less responsive to changes because the loss value from a new sample may not have a huge influence if the batch size it too big. Memory size also affects how responsive the NN is at changes. The batch is drawn uniformly from the memory and if the memory becomes too big it may contain information which is no longer applicable due to the UE moving around in a new part of the environment. So a big memory size will make the NN less responsive. Likewise if the memory size is many times bigger than batch size it may take several steps before it has chosen to include the newest steps in its batch which will make it less responsive. Based on this seven tests has been designed and the values for the batch and memory size together with the results can be found in table 5.6. The full test can be found in appendix D.

Test No.	Batch	Memory	Avg. Loss	Avg. Misalignment [dB]
1	64	128	9.462	7.050
2	64	256	12.946	7.489
3	100	1000	28.675	8.690
4	250	500	16.225	7.434
5	500	500	15.609	7.672
6	500	2000	39.904	8.787
7	1000	3000	37.448	9.068
Prev. Best	500	1000	16.311	6.501

Table 5.6: Test set 4. Results for testing different batch and memory sizes.

From the results it can be observed that the performance gets worse when the difference between the batch and memory size is big (test no. 3, 6, 7). The best results is when the batch size is around half the value of the memory size (test no. 1 and 4). Comparing these results with test no. 2 in table 5.3 which has the same hyper parameters but a batch and memory size equal 500 and 1000 respectively it can be seen it has a lower average misalignment value but a higher average loss value. Due to it having the lowest average misalignment value it has been chosen to set the batch and memory size equal to 500 and 1000 respectively. A lower average loss value does not guarantee that the agent makes good choices it just means that the value of the actions it has taken has been estimated more accurately.

5.5 Target Update

With the found size of the batch and memory the hyper parameters becomes the following:

	Hyper parameter	Value Space
DQN	Hidden Layers	[50, 50]
	Learning Rate (ADAM)	0.01
	Memory Size	1000
	Batch Size	500
	Target Update	To be tuned
RL	Forgetting Factor (γ)	0.7
	Exploring Factor (ϵ)	0.01
	State Space	$[a_{-1}, a_{-2}, a_{-3}, \theta_0, \theta_{-1}, \theta_{-2}, x_0, x_{-1}, x_{-2}, y_0, y_{-1}, y_{-2}]$

Table 5.7: Hyper parameter for the RL agent based on DQN for tuning target update.

Target update affect the stability of the NN but also how reactive the network is to changes. As mentioned before it is not wanted a NN which is slow to react to changes therefore there will not be tested on target update values like 10.000 as some examples have used. Based on this seven tests have been designed and the values for the target update together with the results can be found in table 5.8. The full test can be found in appendix E.

Test No.	Target Update	Avg. Loss	Avg. Misalignment [dB]
1	1	25.264	8.165
2	5	23.199	7.920
3	20	24.046	8.417
4	50	25.309	7.954
5	100	28.113	9.187
6	250	22.864	8.917
7	500	25.455	10.528
Prev. Best	10	16.311	6.501

Table 5.8: Test set 5. Results for testing different target update values.

From the results it can be seen that when the target update becomes 100 or bigger the misalignment values increases a lot compared to the values beforehand. Because of this no further tests using larger target updates will be made. None of the tests gives better results than the already tested target update of 10. Therefore will the hyper parameter be set to be equal to 10.

5.6 Learning Rate

With the found target update the hyper parameters becomes the following:

	Hyper parameter	Value Space
DQN	Hidden Layers	[50, 50]
	Learning Rate (ADAM)	To be Tuned
	Memory Size	1000
	Batch Size	500
	Target Update	10
RL	Forgetting Factor (γ)	0.7
	Exploring Factor (ϵ)	0.01
	State Space	$[a_{-1}, a_{-2}, a_{-3}, \theta_0, \theta_{-1}, \theta_{-2}, x_0, x_{-1}, x_{-2}, y_0, y_{-1}, y_{-2}]$

Table 5.9: Hyper parameter for the RL agent based on DQN for tuning the learning rate.

The learning rate hyper parameter determine how much the weighs in the NN gets updated based on a new loss value. A high learning rate (close to 1) makes the NN very reactive to the loss value whereas the opposite makes the NN less reactive. In this case it could be assumed that learning rate close to 1 would be optimal as a reactive NN is wanted. But it also comes at the cost of being too reactive. With a high learning rate it is possible to update the weights too much with respect to the loss value at each time step and therefore it is possible for the NN to never converge to a optimum. With a low learning rate the weights only get updated a little each time and it is therefore easier to reach the optimum. This may not be the case in our environment as the optimum is not stationary and when the learning rate is small it may take too long to reach the optimum before the environment has changed. Due to these reasons seven tests has been designed and the values for the learning rate together with the results can be found in table 5.10. The full test can be found in appendix F.

Test No.	Learning Rate	Avg. Loss	Avg. Misalignment [dB]
1	0.0001	255.784	17.338
2	0.001	22.929	9.466
3	0.05	23.039	7.872
4	0.1	23.170	7.620
5	0.3	38.813	8.836
6	0.7	144.604	11.711
7	0.9	257.721	14.378
Prev. Best	0.01	16.311	6.501

Table 5.10: Test set 6. Results for testing different learning rate values.

From the results it can be observed that in both ends the mean loss and misalignment is very high compared to values in the middle. Through this it can be seen that a learning rate around 0.1-0.01 gives the best results for this case. 0.01 achieved the best results so this value is chosen.

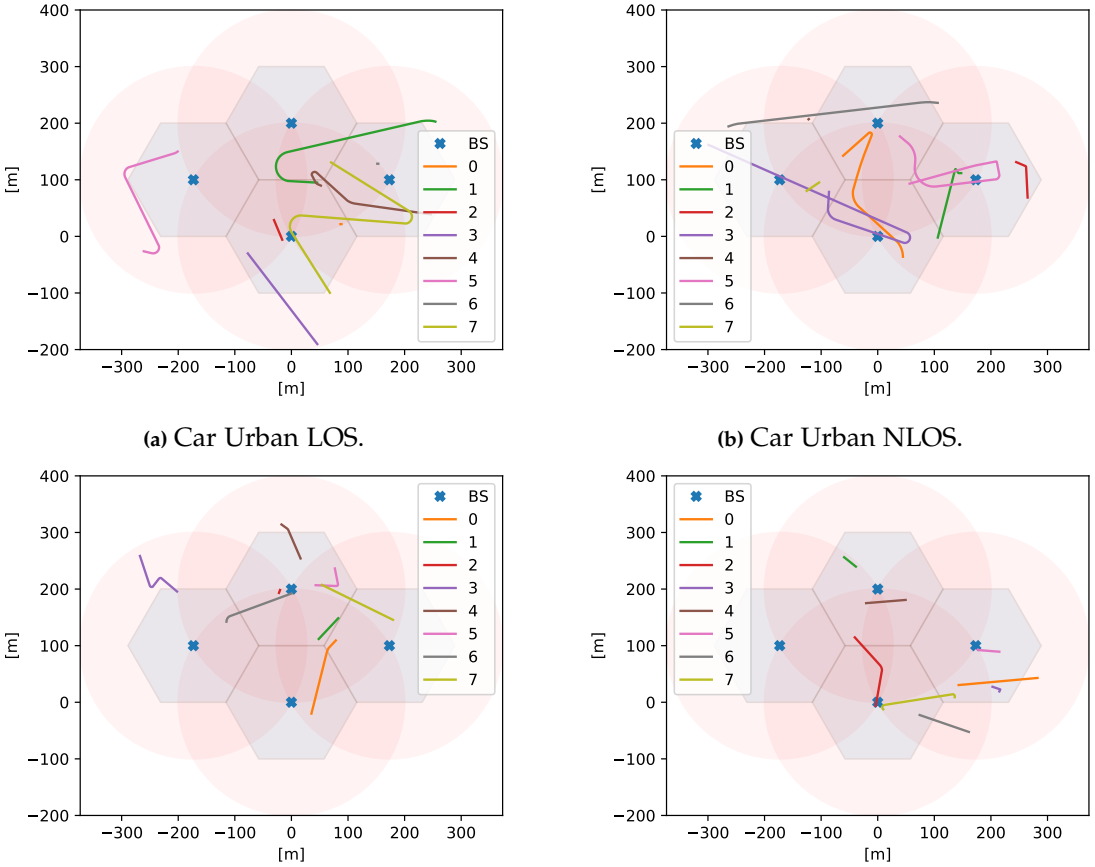
5.7 Performance Test

Based on the presented tests the following parameters for the final NN was found:

	Hyper parameter	Value Space
DQN	Hidden Layers	[50, 50]
	Learning Rate (ADAM)	0.01
	Memory Size	1000
	Batch Size	500
	Target Update	10
RL	Forgetting Factor (γ)	0.7
	Exploring Factor (ϵ)	0.01
	State Space	$[a_{-1}, a_{-2}, a_{-3}, \theta_0, \theta_{-1}, \theta_{-2}, x_0, x_{-1}, x_{-2}, y_0, y_{-1}, y_{-2}]$

Table 5.11: Hyper parameter for the RL agent based on DQN.

A final test will be constructed to see how well the current hyper parameters for the DQN operates on the four different scenarios. A new agent will be created for each of the four new tests sets to be sure that the final tests will be run independently on completely new data. The mobility paths for the test sets can be seen on fig. 5.3.



(a) Car Urban LOS.

(b) Car Urban NLOS.

(c) Pedestrian Urban LOS.

(d) Pedestrian Urban NLOS.

Figure 5.3: Mobility path for the four different scenarios.

The results of the four tests can be seen on table 5.12 with the full test found in appendix G. An Empirical Cumulative Distribution Function (ECDF) graph has also been constructed to make the data more comparable and give an deeper insight in the results. The ECDF graph can be seen on fig. 5.4.

Test No.	Scenario	Avg. Loss	Avg. Misalignment [dB]
1	Car LOS	23.077	7.851
2	Car NLOS	62.052	9.968
3	Pedestrian LOS	25.109	8.522
4	Pedestrian NLOS	49.069	11.165

Table 5.12: Test set 6. Results for testing different Scenarios.

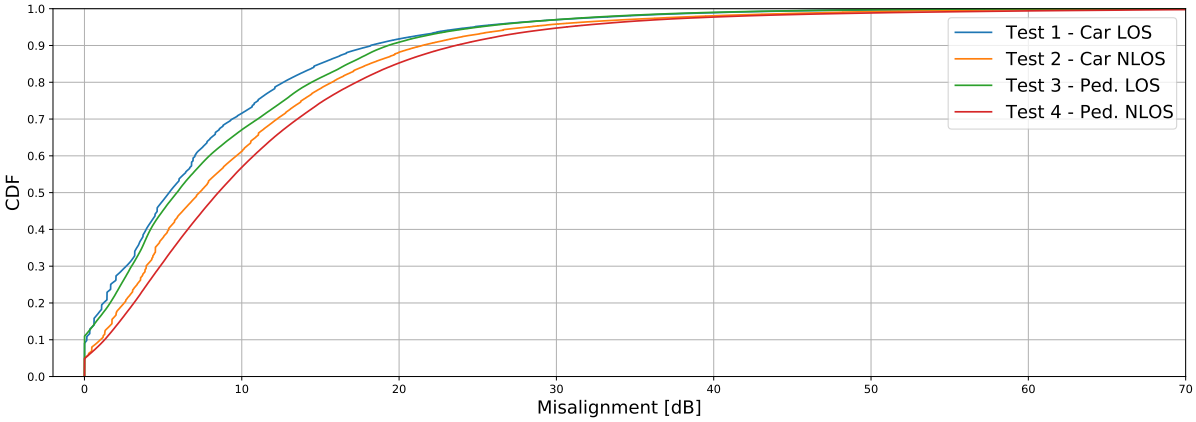


Figure 5.4: ECDF of the misalignment values for the four different tests in table 5.12.

From the results it can be observed that the agent has the best results in the scenario *Car - LOS* which also is predicted as it is the same scenario the hyper parameter has been tuned on. It can be observed that its average misalignment value has increased with a little over 1 dB which is somewhat expected as the hyper parameter has been tuned on another environment and thus it will have the best performance in that specific environment. The results for the *Car - NLOS* has worse results than the LOS scenario which can be expected as the best beam changes more often and therefore has a different behaviour.

Lastly looking at the results for the *Pedestrian* scenarios it can be observed that in both LOS and NLOS the results is worse than the *Car* LOS and NLOS scenarios respectively. This can be due to using the time-varying orientation model on the orientation described in section 2.4.1. This has not been present in the *Car* scenarios and thus the hyper parameters has not been tuned to learn this. This is the most likely hypothesis as the UE moves slower in the *Pedestrian* scenarios and thus the distance is a lot smaller between samples which is assumed would create a greater correlation between the samples if the orientation is fixed and therefore make it easier for the agent to find the best beam.

5.7.1 Reflection of Tuning Procedure

As mentioned in the beginning of this chapter only one parameter will be tuned at a time thus the end result may not necessarily be the optimum. Another thing which affect the tuning of

the parameters is how the results of different hyper parameter values has been compared. All tests have been done on the training data set for a total of 1.000.000 steps over 200 episodes. Due to the way the RL and NN works there are a lot of stochastic elements involved and therefore the results will come with some variation. These stochastic elements comes from the ϵ-greedy method, the optimisation algorithm and noise which is introduced in the next chapter. This can have the consequences that the results a test has achieved may vary such that it seems better than another test. But if both tests have been run an infinite amount of times and the end results has been averaged it could result in the other test being better.

When the size of the NN was chosen in table 5.3 test 2 and 3 the results was similar and as both tests have only been run once it is hard to say which size of the NN is on average actually the best. So with the methods currently used to tune the hyper parameter it is not possible to find the actually best values. One thing which still can be observed using these methods is the tendency of the hyper parameter values. This can easily be seen in the tuning of the learning rate in table 5.10 where a too high or low learning rate is unwanted.

It is not possible to find the best values of the hyper parameters with these methods. But because of of time constraints these methods are enough for the goal of the project which is to see whether RL can be used in this environment and not to find the best RL agent for this environment. The methods still gives an insight in the tendencies for the values of the hyper parameter. These insights can then be used later if a better RL agent needs to be constructed.

Design of Reinforcement Learning Parameters

6

This chapter is about tuning the RL-parameters discussed in chapter 3 and can be seen on table 4.1. Likewise as the previous chapter one parameter will be tuned at a time while keeping the others constant. Due to time constraints it has been chosen to only do one iteration of the tuning of the parameters.

For tuning the parameters it has be decided to optimise them in the following order.

1. Forgetting Factor (γ)
2. Exploring Factor (ϵ)
3. State Space

6.1 Adding Noise

As mention in section 5.2 noise will be added to the following tests as it is assumed that it will have a bigger effect on these hyper parameters than the ones already tuned. The following equation shows the system model described in chapter 2.

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

The noise added here are taken from a Gaussian distribution. Due to using a DFT-codebook the different combiners described by w are orthogonal to each other. This results in the noise being independent for each beam and therefore the noise can be taken from the following distribution $\boldsymbol{n} \sim \mathcal{CN}(0, \sigma^2 \mathcal{I})$. So to be able to add the noise the variance of the Gaussian distribution needs to be found. It will be assumed that the added noise will only consist of thermal noise and thus the noise variance in the equation above is computed as the thermal noise power P_n. The noise power can be calculated as follows:

$$P_n = B \cdot S_n \tag{6.1}$$

where:

P_n:	Noise power	[W]
B:	Bandwidth	[Hz]
S_n:	Noise spectral density	[W/Hz]

Thermal noise at room temperature (300 K) leads to a noise spectral density of $-174\,\frac{\text{dBm}}{\text{Hz}}$ [42]. For the bandwidth it has been chosen to set it equal to 400 MHz [40, p. 183].

$$S_n = -174\text{dBm/Hz} = 3.981 \cdot 10^{-18}\text{mW/Hz} \tag{6.2}$$

$$P_n = 400\text{MHz} \cdot S_n = 1.5924 \cdot 10^{-9}\text{mW} \tag{6.3}$$

With the found noise power the noise can be added to the signal. In the previous tests the misalignment value was calculated as the difference between the taken noiseless reward and the maximum possible noiseless reward. It has been chosen to still use the noiseless reward to calculate the misalignment values due to it given an insight in how often the best beams has been chosen even with noise added. If the misalignment was taken on a the noisy reward it would only be possible to see whether the agent has learned to take the beam with the most received power and not differentiate whether the best received power came from the noise or from the best beam direction. This would only be a problem in cases where the SNR is low. With noise added the tuning of the last hyper parameters can be commenced.

6.2 Forgetting Factor

The forgetting factor will be tuned on the following parameters:

	Hyper parameter	Value Space
DQN	Hidden Layers	[50, 50]
	Learning Rate (ADAM)	0.01
	Memory Size	1000
	Batch Size	500
	Target Update	10
RL	Forgetting Factor (γ)	To be tuned
	Exploring Factor (ϵ)	0.01
	State Space	$[a_{-1}, a_{-2}, a_{-3}, \theta_0, \theta_{-1}, \theta_{-2}, x_0, x_{-1}, x_{-2}, y_0, y_{-1}, y_{-2}]$

Table 6.1: Hyper parameter for the RL agent based on DQN for tuning the forgetting factor.

The forgetting factor controls how important later rewards are when estimating the cumulative reward. Due to the agent not being able to control the movement of the UE it would be detrimental to have a very high forgetting factor (very close to 1). Due to this it is assumed that there will be a lot of uncertainties when taking a lot of future estimates into consideration. This will results in taking a longer time to stabilise the estimates and thus the NN will take longer time to get a good estimate when changes happens in the environment. Based on this seven tests has been designed and the values for the forgetting factor together with the results can be found in table 6.2. The full test can be found in appendix H.

Test No.	Forgetting Factor	Avg. Loss	Avg. Misalignment [dB]
1	0.4	19.706	8.975
2	0.5	21.820	8.697
3	0.6	18.850	8.660
4	0.7	28.215	8.386
5	0.8	29.203	7.193
6	0.9	55.991	6.645
7	0.99	3937.014	14.329

Table 6.2: Test set 7. Results for testing different forgetting factor values.

From the results it can be seen that in the case with a very high forgetting factor (test 7) the loss and misalignment values increased drastically. The increased value of the loss is somewhat expected due to the agent having little control over the visiting sequence of states. Thus when taking a lot of future rewards into consideration a lot of uncertainties can happen which can make the estimate change a lot and therefore give a high loss value. Likewise it can also be seen that when the forgetting factor gets too low the performance fall (test 1, 2, 3, 4). The best cases was test 5 and 6 where test 5 has the lowest loss value of the two and test 6 has the best misalignment value. It has been chosen to set the forgetting factor equal to $\gamma = 0.8$) (test 5) due to it having a much lower loss value and also when doing a visual inspection of the misalignment graphs (figs. H.5b and H.6b) it can be seen that test 5 has less high spikes compared to test 6.

6.3 Exploring Factor

The exploring factor will be tuned on the following parameters:

	Hyper parameter	Value Space
DQN	Hidden Layers	[50, 50]
	Learning Rate (ADAM)	0.01
	Memory Size	1000
	Batch Size	500
	Target Update	10
RL	Forgetting Factor (γ)	0.8
	Exploring Factor (ϵ)	To be tuned
	State Space	$[a_{-1}, a_{-2}, a_{-3}, \theta_0, \theta_{-1}, \theta_{-2}, x_0, x_{-1}, x_{-2}, y_0, y_{-1}, y_{-2}]$

Table 6.3: Hyper parameter for the RL agent based on DQN for tuning the exploring factor.

The exploration factor determine how often the agent should explore instead of exploiting its estimated knowledge. Exploitation is best when the agent's estimate of state-action values is accurate as it will then take the best actions. Likewise if the agent has a bad estimate of its state-action values its estimated best action may not necessarily be the best action and in these cases it is unwanted to exploit its estimation before it has learned more about the environment. The agent learns about the environment when it does exploration and thus the

goal is to find the middle ground between exploration and exploitation. As mentioned in section 5.1 different methods could be used to determine the exploration factor. Due to its simplicity it has been chosen to keep using a constant value for the exploration factor. Based on this seven tests has been designed and the values for the exploring factor together with the results can be found in table 6.4. The full test can be found in appendix I.

Test No.	Exploration factor	Avg. Loss	Avg. Misalignment [dB]
1	0.0005	35.172	10.704
2	0.001	29.056	9.560
3	0.005	29.987	7.907
4	0.05	25.174	6.467
5	0.1	27.661	7.806
6	0.3	24.264	13.081
7	0.5	25.296	19.079
Prev. Best	0.01	29.203	7.193

Table 6.4: Test set 8. Results for testing different exploration values.

From the results it can be observed that for a high exploration factor (Test 6 and 7) the agent has a very poor performance compared to the best results (Test 4). Likewise it can be observed that the performance gets worse when the exploration factor gets too small (Test 1 and 2). From the test it has been found that the middle ground is best and therefore the exploration factor will be set to be $\epsilon = 0.05$.

6.4 State Space

The state space will be tuned on the following parameters:

	Hyper parameter	Value Space
DQN	Hidden Layers	[50, 50]
	Learning Rate (ADAM)	0.01
	Memory Size	1000
	Batch Size	500
	Target Update	10
RL	Forgetting Factor (γ)	0.8
	Exploring Factor (ϵ)	0.05
	State Space	To be tuned

Table 6.5: Hyper parameter for the RL agent based on DQN for tuning the state space.

The last hyper parameter which needs to be tested is the state space. For the initial value it was chosen to include the current position and orientation with a history of two together with the three earlier actions. This was used to make sure the other hyper parameters was tuned to a state space which included a lot of information. For this test it is wanted to see how the individual state features like the position affects the results. It is also wanted to test how less information impacts the results. Based on this ten tests has been designed and the

constructed state space together with the results can be found in table 6.6. The full test can be found in appendix J.

Test No.	State Space	Avg. Loss	Avg. Misalignment [dB]
1	[3, 0, 0]	31.000	6.804
2	[0, 3, 0]	31.437	6.944
3	[0, 0, 3]	21.497	6.442
4	[3, 3, 0]	31.048	6.882
5	[0, 3, 3]	22.486	6.827
6	[3, 0, 3]	24.776	6.633
7	[1, 1, 1]	25.364	6.752
8	[2, 2, 2]	29.138	6.893
9	[3, 3, 3]	28.699	6.795
10	[4, 4, 4]	28.333	6.661

Table 6.6: Test set 9. Results for testing different state spaces. The values inside the bracket represent [No. actions, No. orientations, No. positions]

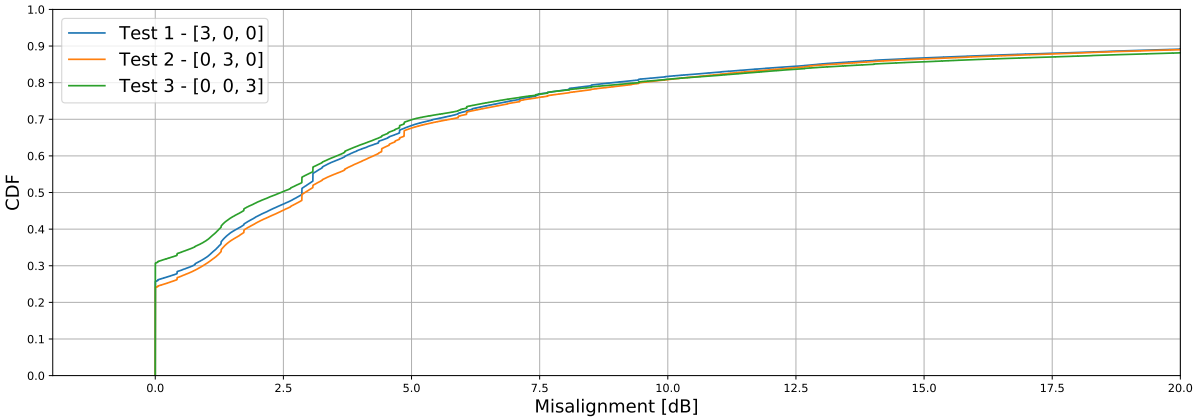
As the results are so close in value the ECDF graphs has been made to better see the performance and can be seen on fig. 6.1. The tests can be split into three parts - test 1-3, 4-6 and 7-10.

Test 1-3 tests how the individual inputs affect the result. Looking at both the results and the ECDF graphs (see fig. 6.1a) it can be observed that it is the position which contribute with most information which makes the agent learn best.

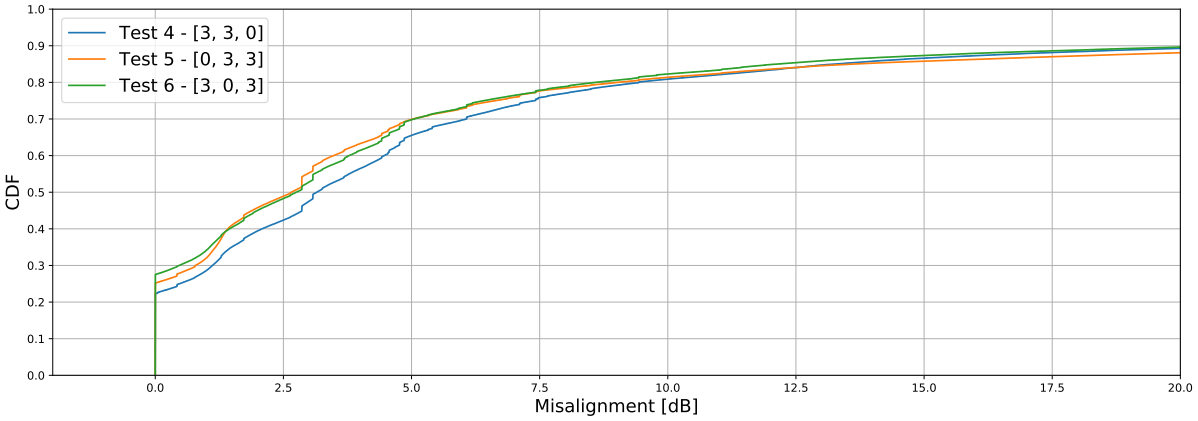
Test 4-6 tests the combination of these individual inputs and from the results it can be observed that the action and position combination (test 6) gives the best average misalignment value but when looking at the ECDF graphs it can be observed that test 5 and 6 fluctuate between having the best performance. Another thing which can be observed on the ECDF graph is that after some time the performance of the three test is almost identical.

Test 7-10 tests the history of the information and from the results and looking at the ECDF on fig. 6.1c it can be seen that the performance is very close to each other. Due to the stochastic elements it is inconclusive to determine which state space is best.

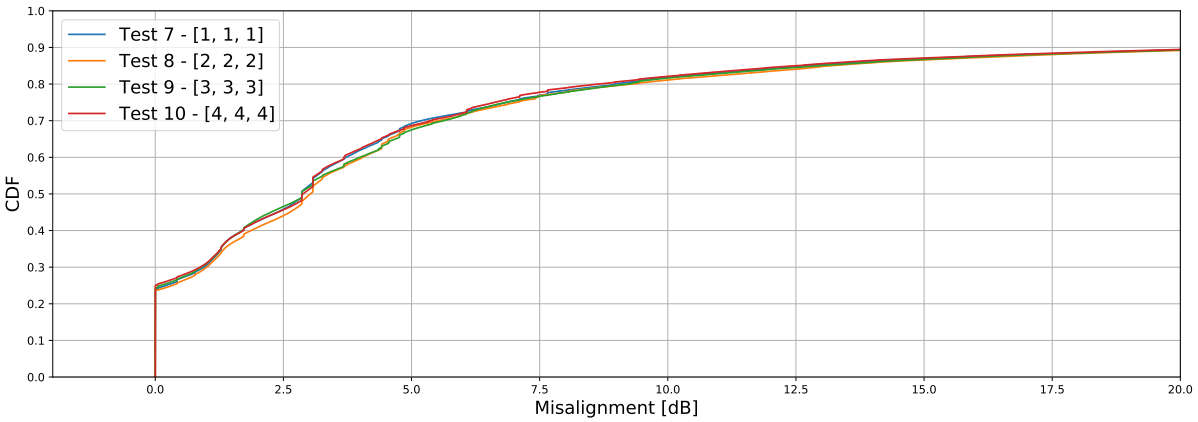
Since it is wanted to see how good performance can be achieved when using less information then it has been chosen to do test 1-6 again with only a history of one. Thus the new test set can be seen on table 6.7 together with the results. The full test can be found in appendix K.



(a) ECDF for test 1-3.



(b) ECDF for test 4-6.



(c) ECDF for test 7-10.

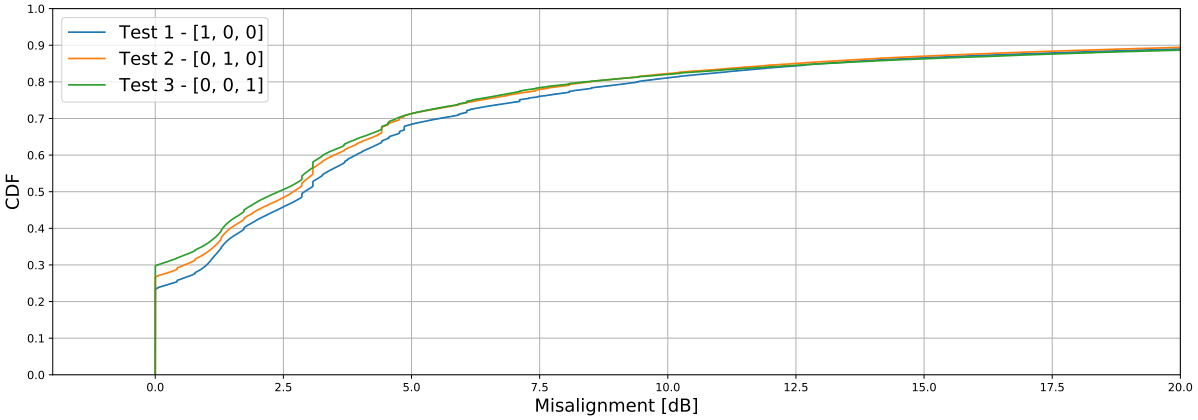
Figure 6.1: ECDF for test set 9.

Test No.	State Space	Avg. Loss	Avg. Misalignment [dB]
1	[1, 0, 0]	31.449	6.915
2	[0, 1, 0]	29.557	6.534
3	[0, 0, 1]	21.760	6.670
4	[1, 1, 0]	31.142	6.747
5	[0, 1, 1]	22.163	6.762
6	[1, 0, 1]	23.452	6.585

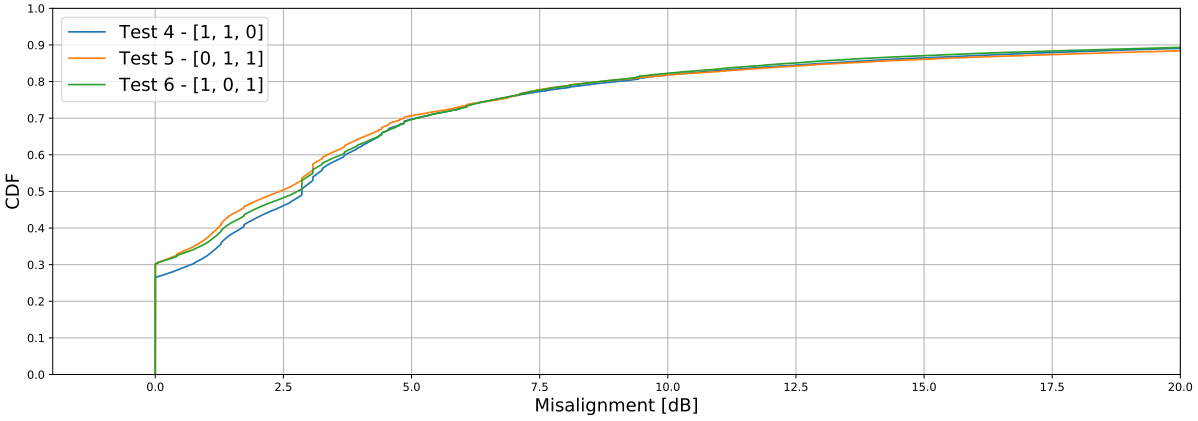
Table 6.7: Test set 10. Results for testing different state spaces. The values inside the bracket represent [No. actions, No. orientations, No. positions]

Due to the similar results an ECDF graph of these results has also been constructed. The ECDF can be seen on fig. 6.2.

Looking at the six different tests it can be observed that the results are very similar to the results in the previous test. Thus it seems that adding history to the state space does not have an influence on the results. From all the tests it can be observed that when utilising the position it results in a lower average loss.



(a) ECDF for test 1-3.



(b) ECDF for test 4-6.

Figure 6.2: ECDF for test set 10.

From all the tests it seems that the results are very similar which comes as a surprise. It has been expected that choosing a different state space would have a bigger impact as different

state space represent different information about the environment. Comparing all the ECDF graphs it can be seen that different state spaces only have an impact on the beginning of the graphs. Around 12+ db misalignment it seems that all test achieve the same results.

The lack on influence from different state spaces can maybe be caused by the environment. In the *Car - LOS* scenario the UE moves in straight lines most of the time and there is no variation in the orientation except when turning. This contributes to the fact that for a lot of samples the best beam pair will be the same. Thus the agent may just learn to keep taking the same beam pair as the one before. If the agent has learned this the input does not have the biggest effect on the results and thus different state spaces will give the same results. The only difference which can be observed from the results is the slight improvement when including the position. In these cases the agent may learn a correlation between sample and rewards and thus it can handle turning better. Due to the stochastic elements the variance in results may only come from this and therefore it is hard to conclude how the different state spaces affects the results.

Due to the similar results it has been chosen to keep using the state space [3, 3, 3] as it is assumed the more information in the state space will not have a huge negative impact on the results.

6.5 Tuning Reference Algorithm

It is wanted to see how the DQN algorithm perform compared to a reference algorithm which is simple, heuristic and does not use Machine Learning (ML).

The reference algorithm uses a reward that is normalised by the free-space received power described by Friis transmission equation which can be seen on eq. (1.1). As there are four BSs the received reward is normalised with the free-space received power calculated to the closest BS.

Then the normalised reward will be compared to a threshold which will determine if a random action should be chosen or if the previous action should be chosen again. To find the value of the threshold the following test and its results can be seen on table 6.8. The full test can be found in appendix L.

Test No.	Threshold	Avg. Misalignment [dB]
1	0.7	25.328
2	0.8	21.106
3	0.9	18.166
4	1	14.51
5	1.1	11.543
6	1.2	8.493
7	1.3	7.197
8	1.4	7.763
9	1.5	12.269
10	1.6	17.083

Table 6.8: Test set 11. Results for different threshold values.

From these results it can be seen that the reference algorithm can achieve very similar results compared to the designed DQN. As a threshold of 1.3 achieves the best results it will henceforth be used. This algorithm may perform well in *Car - LOS* scenario as explained the previous section because the UE moves in straight lines and the best beam pair will therefore be the same for a lot of steps. Thus if the algorithm has chosen a good beam pair it will make a good choice for the next couple of steps. For this reasons it is expected it will perform worse in the *Pedestrian* scenarios.

It is expected that this algorithm will perform worse in the NLOS cases as the reward is normalised to the free-space expected reward. As there is no LOS in the NLOS cases the optimal path from a UE to a BS will be a signal which has been reflected from multiple surfaces and thus its path is longer than the LOS path. This will make the normalised reward lower in the NLOS case than in the LOS and thus the threshold found using the LOS training set is expected to perform worse in the NLOS test sets.

6.6 Performance Test

With a tuned DQN algorithm a performance test will be performed again. The tuned hyper parameter has the following values:

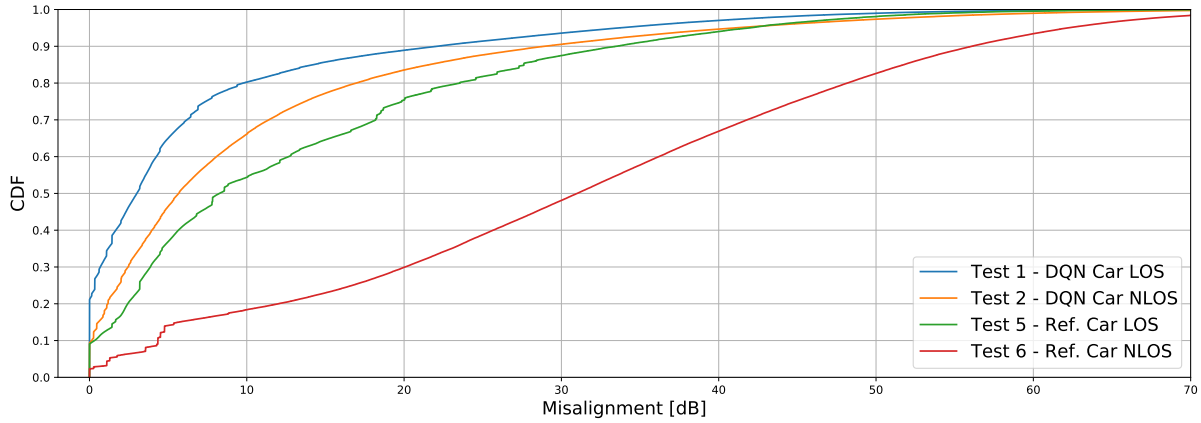
	Hyper parameter	Value Space
DQN	Hidden Layers	[50, 50]
	Learning Rate (ADAM)	0.01
	Memory Size	1000
	Batch Size	500
	Target Update	10
RL	Forgetting Factor (γ)	0.8
	Exploring Factor (ϵ)	0.05
	State Space	$[a_{-1}, a_{-2}, a_{-3}, \theta_0, \theta_{-1}, \theta_{-2}, x_0, x_{-1}, x_{-2}, y_0, y_{-1}, y_{-2}]$

Table 6.9: The final hyper parameter values for the RL agent based on DQN.

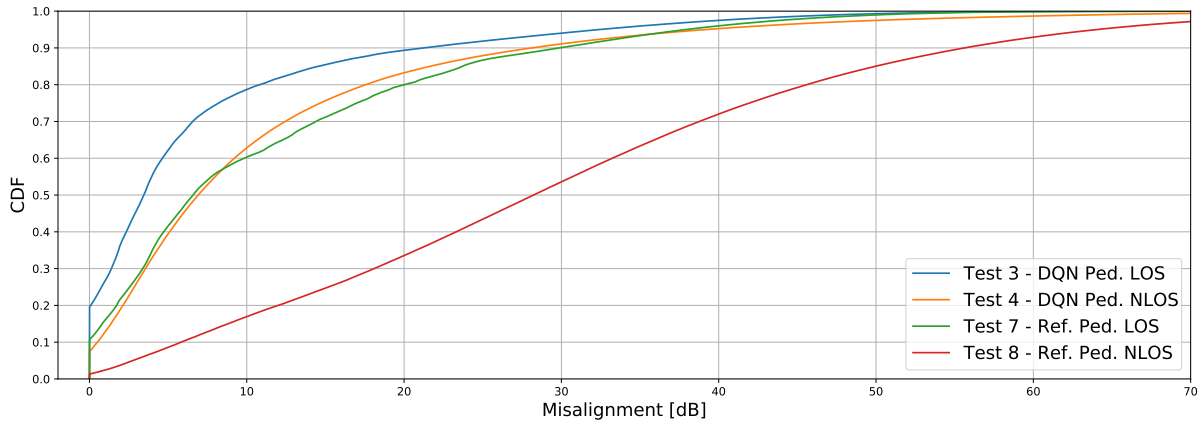
The performance test will run on the same data sets described in section 5.7. The reference algorithm will also be tested on these data sets to see how well the tuned DQN algorithm perform against it. The results of the eight tests can be seen on table 6.10 with the full test found in appendix M. The ECDF graphs can be seen on fig. 6.3.

	Test No.	Scenario	Avg. Loss	Avg. Misalignment [dB]
DQN	1	Car LOS	23.737	7.065
	2	Car NLOS	57.483	10.652
	3	Pedestrian LOS	27.304	7.268
	4	Pedestrian NLOS	53.477	11.356
Reference	5	Car LOS	-	13.437
	6	Car NLOS	-	30.972
	7	Pedestrian LOS	-	11.471
	8	Pedestrian NLOS	-	29.851

Table 6.10: Test set 12. Results for testing different Scenarios.



(a) ECDF graph for the *Car* scenario results.



(b) ECDF graph for the *Pedestrian* scenario results.

Figure 6.3: ECDF graph for the results of the performance test.

Comparing the results for the tuned DQN with the previous results in section 5.7 it can be observed that the performance of the *Car Urban - LOS* scenario has increased whereas for the NLOS it has slightly decreased. The increasement is expected as the rest of the hyper parameters for the DQN has been tuned to this scenario. The descreasement in the *Car Urban - NLOS* is somewhat unexpected. But as it is a slight descrease it can also come from adding noise to the scenario which has a larger impact on NLOS as the received signal power are generally lower in these scenarios, the stochastic elements or just from the different hyper

parameters. The observation also happens in the *Pedestrian* scenarios where a small increase happens in the LOS and a slight decrease in NLOS. As the DQN has been tuned with respect to a LOS scenario it is not unexpected that the *Pedestrian - LOS* likewise see an increase in performance.

For the reference algorithm the results is somewhat expected. It performs poorly in the NLOS scenarios. The performance in the *Car Urban - LOS* is also worse but this can be due to overfitting the threshold to the training data set. It is somewhat unexpected that it perform better in the *Pedestrian - LOS* scenario as there is noise on the orientation. This can be due to the rewards from the mobility track are more similar to the training data set and thus it will perform better to the fitted threshold. Overall the tuned DQN algorithm performs better than the reference algorithm especially in the NLOS scenarios.

6.6.1 Upscaling Antennas and Beams

Currently 4 antennas and 8 beams for the UE are used and 8 antennas and 16 beams for the BSs which are on the low end of the antenna array sizes which are being tested on as explained in section 2.2.1. Therefore an extra test has been run to see how well the current constructed RL algorithm handles larger arrays and thus larger action spaces. The test will increase the number of beams and antennas by a factor of 2 which results in an actions space of 2048. The results from the tests can be seen in table 6.11 and the full test can be found in appendix N.

	Test No.	Scenario	Avg. Loss	Avg. Misalignment [dB]
DQN	1	Car LOS	35.397	13.189
	2	Car NLOS	63.168	16.319
	3	Pedestrian LOS	38.225	13.227
	4	Pedestrian NLOS	61.065	17.099
Reference	5	Car LOS	-	15.868
	6	Car NLOS	-	34.497
	7	Pedestrian LOS	-	16.359
	8	Pedestrian NLOS	-	32.275

Table 6.11: Test set 13. Results for testing 8 antennas and 16 beams for the UE and 16 antennas and 32 beams for the BSs. Action space equal to 2048.

From the results it can be observed that misalignment has increased on average by 6 dB for the tuned DQN. The reference algorithm still perform worse but the increase in misalignment is not as much. As both the UE and the BSs has its number of antennas increased by a factor two the received power will be increased by a factor of 4 which corresponds to an increase by 6 dB. With this in mind and comparing the results it can be observed that even with an increased number of antennas the tuned DQN still on average received the same amount of power. From this it can be speculated that the tuned DQN will not perform well on larger action spaces as results from this test shows that it cannot exploit the increase number of antennas and beams to get an better performance.

7 Discussion

It was found that the DQN algorithm perform slightly worse in the *Pedestrian* scenarios due to it being tuned to an environment where UE moves in straight lines without using the time-varying orientation model on the orientation. It has then been speculated than the tuned DQN has learned to find a good beam and then keep taking this beam until changes happens similar to solving a multi-armed bandit problem. This way is similar to the reference algorithm but instead of having a normalised reward and a threshold it has a NN. It can be observed in the final tests when using the training data sets these two methods had similar results but when used on the test data sets the difference became obvious. It seems that the DQN is more robust on the difference in the rewards as long as the UE moves in the same patterns.

This is also the reason it is assessed that the tuned DQN perform slightly worse in the *Pedestrian* scenarios. This may have been avoided if the DQN was tuned on the worst case scenario *Pedestrian - NLOS* as it is assumed it is easier going from a more noisy environment to a more simple one. It may not get as good of a performance on the *Car* scenarios as it does now but it could be more robust to a UE moving in a urban environment and not just a car driving around in the environment. Likewise if the DQN was tuned to a scenario with more beams and antennas it could maybe have learned to take more advantage of having more beams. In either case the DQN has been tuned on either a purely LOS or NLOS environment and it can assumed it would have been preferable if it was trained on a combination of these two cases such that it may learn the abrupt transition from a LOS to NLOS area and vice versa.

For most cases when using RL the agent has a way to create an impact on the environment. An example of this is an agent which needs to go from A to B and its actions controls the path it goes. Here the agent impacts the environment by controlling how it moves and by controlling this it can predict what it will do in the future. This is not the case for this environment. This can have negative consequences which has been discussed in section 6.2. Both the validation and tests data sets are finite examples and due to time constrains and limitations on hardware it has not been possible to create bigger data sets. Thus by reusing the data by splitting the paths in smaller chunks with overlap more samples has been created. But it is still the same path the UE moves on and therefore it is somewhat possible for the agent to get a good estimation on the cumulative rewards even though the agent does not control the path.

In the real world the UE moves a lot more erratically and the agent cannot control it. This makes it harder for the agent find the expected cumulative reward as there is a lot more possibilities to where the UE can move to than in the simulated data sets where the paths are fixed. Therefore it is hard to say from these simulation results how exactly the forgetting

factor affect the agent in real life as the current created data sets cannot represent this erratic behaviour. It can also be discussed using the cumulative reward when the state space does not include a history about the actions will improve the performance. The reason for this is that the actions the agent takes does not impact the next state so when choosing a beam it should only take the largest intermediate reward.

Currently it has been chosen to simulate a total of 1,000,000 steps equal to 200 episodes. 200 episode is not much compared to other RL experiments which has run over 100,000 episodes. It was assumed that 200 episodes would be enough to see the tendencies of the hyper parameters, but when tuning the size of the NN there may be a bias towards smaller NN. Smaller NN react faster to changes and thus can very fast reach it maximum potential whereas for larger NN it takes a lot longer. Therefore if the smaller NN reach acceptable results fast it will get a better average result compared to a larger which reach the same or better results towards the end of the training. Even if a larger NN may achieve better results in the long run on the training set it may not be the best on the real environment. If the environment changes too fast and the larger NN cannot learn some generalisation between them it may never reach a better result than a smaller NN. All in all it comes down to the stationary of the environment and the size of the training sets.

Summary, Conclusion and Future Work 8

8.1 Summary

To be able to achieve the results shown in section 6.6 a lot of steps were needed. Before anything could be accomplished a realistic simulation of the environment was needed. In chapter 1 an analysis of the environment was done to understand problems occurring when communicating using mmWaves in an urban environment. In chapter 2 a system model was set up which included a lot of different parameters.

First were the need to model the channel matrix to which it was chosen to use a narrowband model. The narrowband model could be realised using two steering vectors one for the UE and one for the BSs and lastly the model needed realistic channel parameters. It was found that these channel parameters could be simulated using a geometric based simulation tool called QuaDRiga which only needed the position of the BSs and the UE to simulate the parameters. The last thing the system model needed was the precoder and combiner of the transmitter and receiver respectively. To design these a DFT-codebook was constructed as it could be used on ULA in mmWaves.

The next thing was then to create a realistic mobility pattern for the different scenarios and in this case it was a *Car* and a *Pedestrian*. To do this a mobility model was created based on [19]. A time-varying orientation model was added to the *Pedestrian* case to simulate a more accurately orientation behaviour of a pedestrian. With all this combined it was possible to create different data sets with a lot different parameters where a algorithm could either be trained or tested on.

To be able to find a suitable RL algorithm for this environment the theory behind RL was analysed in chapter 3. This also included researching different RL methods to understand if they were suitable for this kind of environment. From this it was found that a DQN should be used. Because it includes a NN and methods like experienced replay and a target network a deeper analysis of the methods was done in chapter 4.

With an understanding of the chosen method the last step was to tune the different hyper parameters which was done in chapters 5 and 6. To be able to compared the results a simple, heuristic non-ML algorithm was constructed as a reference algorithm.

8.2 Conclusion

Through research on RL theory and methods it was found that there exist a lot of different methods which are suitable for using adaptive beamforming in an urban environment. Due to the project group having prior knowledge about NN and the pros and cons of using it together with RL the specific DQN algorithm was chosen.

When tuning what information should be included in the state space it was found there was no dominant feature which clearly provided better results than others. This has been speculated that this is due to how the NN was constructed. The DQN was tuned on the *Car LOS* scenario where it moves mostly in straight line and thus the best beam will consist for a lot of samples. Therefore the NN may have been tuned to learn to fast find a good beam when changes happens and thus is not very dependent on input to the states if it can update its weights fast enough. Therefore it has been speculated that a completely different NN will be constructed if it was used on a more complex environment.

From the results of the final test it could be observed that the tuned DQN received better results than the reference algorithm in all scenarios and especially in the NLOS cases. Based on this it can be concluded it is possible to construct an adaptive beamforming algorithm in the *Car Urban* environment using RL that outperforms a simple, heuristic non-ML algorithm.

The constructed algorithm perform slightly worse in the *Pedestrian* scenarios and this has been speculated that this is due the *Pedestrian* scenario which include a time-varying orientation model on the orientation feature. From the test where the number of beams and antennas was increased the tuned DQN achieved the same performance as before. From this it is speculated that the tuned DQN will have a hard time handle larger antenna arrays without changes or improvements to the RL agent like tuning it to a scenario where more beams and antennas are used.

8.3 Future Work

Some of the things which can be changed to maybe improve the results has already been discussed in chapter 7. Other things that changes the fundamental of the constructed NN can also be done.

8.3.1 Machine Learning Methods

Some of the minor things will be changing how the layers in the NN are interconnected. At the moment the NN is a fully feed-forward network with only dense layers. It could be worthwhile to research whether other types of NN could improve the results. This could be a NN with feedback elements and different type of layers like convolution and recurrent layers.

8.3.2 Reinforcement Learning Parameters

Currently the exploring factor (ϵ) is a constant such that it keeps exploring. If it was possible to know when major changes happens to the environment it could be worthwhile to increase the exploration factor to let the agent learn more about the changed environment and then decrease it after some time.

The current way of the batch is chosen from the memory is from a uniform distribution but it may be more preferable to use another distribution which has a higher change of choosing newer samples. This may improve the algorithms reaction to changes as it is more likely to include the new samples when calculating its loss.

8.3.3 Hierarchical Codebooks

One of the major things would be looking at the codebook again. Currently all the beams in the codebook have almost the same width. For the constructed codebook it was chosen to use 8 beams for the UE and 16 beams for the BSs. Using more antennas are also being investigated in the industry which results in the beams' width becoming smaller but gaining more directivity. More and sharper beams may make it harder to find the optimal beams especially if UE orientation changes fast and from section 6.6.1 it was shown that tuned DQN performance did not improve when increasing the number of antennas and beams. In these cases it could be preferable to use broader beams to ensure that it always has a connection even at the cost of some received power compared to the best beam. Therefore it could be worthwhile to research whether a hierarchical codebook could improve the performance of the algorithm.

8.3.4 Multi-Agent Reinforcement Learning

In the beginning of the project it was chosen to simulate a centralised RL agent which controls four BSs and a single UE and with the current number of beams the total action space becomes equal to 512. The currently chosen number of beams for the UE and the BSs is in the low end of what is being tested on. Just by increasing the number of beams by two it will result in the action space becoming four times bigger. Currently everything is done in two dimensions and if a third dimension is added the action space will increase even faster. From this it can be seen that the action space increases fast which will make it harder for the agent to choose the best action. Therefore it could be worthwhile to research whether extra knowledge could be given to the agent such that its action space is limited. One could be to limit the action space to only include beams from the closest BS. This will already result in reducing the action space by 4 (if 4 BS is used).

In the centralised version the action space will always be somewhat big as it will consist of the combination of the beams between the BSs and a UE. Therefore if huge arrays with a lot of beams is used and it is not possible to reduce the action space in the centralised version to a sensible number a decentralised system could work. In a decentralised system every BS and UE has its own RL agent and thus the action space will be limited to the number of beams the individual agent can take. It also comes with some other problems but it could be worthwhile to research whether a centralised or decentralised systems can handle this example best.

Bibliography

[1] statista. 'Number of smartphone subscriptions worldwide from 2016 to 2027.' Downloaded: 02-03-2022. (2022), [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/.

[2] I. Analytics. 'State of iot 2021: Number of connected iot devices growing 9% to 12.3 billion globally, cellular iot now surpassing 2 billion.' Downloaded: 02-03-2022. (2022), [Online]. Available: https://iot-analytics.com/number-connected-iot-devices/.

[3] R. W. Heath, N. Gonzalez-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, ISSN: 1941-0484. DOI: 10.1109/jstsp.2016.2523924. [Online]. Available: http://dx.doi.org/10.1109/JSTSP.2016.2523924.

[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] Ericsson. 'Mobility reports.' Downloaded: 02-03-2022. (2019), [Online]. Available: https://www.ericsson.com/en/press-releases/7/2019/ericsson-mobility-report-5g-uptake-even-faster-than-expected.

[6] ——, 'Mobility reports.' Downloaded: 02-03-2022. (2022), [Online]. Available: https://www.ericsson.com/en/reports-and-papers/mobility-report/reports.

[7] 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.

[8] 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.

[9] S. Deng, G. R. MacCartney and T. S. Rappaport, 'Indoor and outdoor 5g diffraction measurements and models at 10, 20, and 26 ghz,' in *2016 IEEE Global Communications Conference (GLOBECOM)*, 2016, pp. 1–7. DOI: 10.1109/GLOCOM.2016.7841898.

[10] T. S. Rappaport, G. R. MacCartney, S. Sun, H. Yan and S. Deng, 'Small-scale, local area, and transitional millimeter wave propagation for 5g communications,' *IEEE Transactions on Antennas and Propagation*, vol. 65, no. 12, pp. 6474–6490, 2017. DOI: 10.1109/TAP.2017.2734159.

[11] 3GPP, '5g; study on channel model for frequencies from 0.5 to 100 ghz (3gpp tr 38.901 version 17.0.0 release 17),' ETSI, Tech. Rep., Apr. 2022, Downloaded: 26-04-2022. [Online]. Available: https://portal.etsi.org/webapp/workprogram/Report_WorkItem.asp?WKI_ID=65119.

[12] N. O. Parchin, H. J. Basherlou, I. A. Yasir Al-Yasir, M. Sajedin, J. Rodriguez and R. A. Abd-Alhameed, 'Multi-mode smartphone antenna array for 5g massive mimo applications,' in *2020 14th European Conference on Antennas and Propagation (EuCAP)*, 2020, pp. 1–4. DOI: 10.23919/EuCAP48036.2020.9135754.

[13] electronicdesign. 'It's all about the antennas for 5g.' Downloaded: 04-03-2022. (2020), [Online]. Available: https://www.electronicdesign.com/communiqu/article/21135305/its-all-about-the-antennas-for-5g.

[14] C. Di Paola, K. Zhao, S. Zhang and G. F. Pedersen, 'Siw multibeam antenna array at 30 ghz for 5g mobile devices,' *IEEE Access*, vol. 7, pp. 73 157–73 164, 2019. DOI: 10.1109/ACCESS.2019.2919579.

[15] S. Rezaie, C. N. Manchón 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.

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

[17] ——, '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] 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.

[19] 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.

[20] 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.

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

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

[23] ShareTechnote. '5g/nr - frame structure.' Downloaded: 31-05-2022. (), [Online]. Available: https://www.sharetechnote.com/html/5G/5G_FrameStructure.html.

[24] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[25] J. Tromp. 'John's chess playground.' Downloaded: 19-04-2022. (2021), [Online]. Available: https://tromp.github.io/chess/chess.html.

[26] OpenAI. 'Kinds of rl algorithms.' Downloaded: 15-03-2022. (2020), [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.

[27] I. C. Education. 'Neural networks.' Downloaded: 16-03-2022. (2020), [Online]. Available: https://www.ibm.com/dk-en/cloud/learn/neural-networks.

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

[29] G. Seif. 'Understanding the 3 most common loss functions for machine learning regression.' Downloaded: 17-03-2022. (2019), [Online]. Available: https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d3.

[30] TensorFlow. 'Tensorflow.' Downloaded: 21-03-2022. (2022), [Online]. Available: https://www.tensorflow.org/.

[31] ——, 'Tensorflow module: Tf.keras.optimizers.' Downloaded: 17-03-2022. (2022), [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers.

[32] Keras. 'Embedding layer.' Downloaded: 02-05-2022. (2022), [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam.

[33] D. Kingma and J. Ba, 'Adam: A method for stochastic optimization,' *International Conference on Learning Representations*, Dec. 2014.

[34] TensorFlow. 'Tensorflow api: Tf.gradienttape.' Downloaded: 17-03-2022. (2022), [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/GradientTape.

[35] ——, 'Introduction to gradients and automatic differentiation.' Downloaded: 17-03-2022. (2022), [Online]. Available: https://www.tensorflow.org/guide/autodiff.

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

[37] Manning. 'Concept target network in category reinforcement learning.' Downloaded: 18-03-2022. (2022), [Online]. Available: https://livebook.manning.com/concept/reinforcement-learning/target-network.

[38] J. TORRES.AI. 'Deep q-network (dqn)-ii.' Downloaded: 18-03-2022. (2020), [Online]. Available: https://towardsdatascience.com/deep-q-network-dqn-ii-b6bf911b6b2c.

[39] L.-J. Lin, 'Self-improving reactive agents based on reinforcement learning, planning and teaching,' *Mach. Learn.*, vol. 8, no. 3–4, pp. 293–321, May 1992, ISSN: 0885-6125. DOI: 10.1007/BF00992699. [Online]. Available: https://doi.org/10.1007/BF00992699.

[40] M. Giordani, M. Polese, A. Roy, D. Castor and M. Zorzi, 'A tutorial on beam management for 3gpp nr at mmwave frequencies,' *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 173–196, 2019. DOI: 10.1109/COMST.2018.2869411.

[41] TensorFlow. 'Tensorflow api: Tf.keras.optimizers.adam.' Downloaded: 17-03-2022. (2022), [Online]. Available: https://keras.io/api/layers/core_layers/embedding/.

[42] electronics-notes. 'Thermal noise formulas and calculator.' Downloaded: 06-05-2022. (), [Online]. Available: https://www.tensorflow.org/.

Test: Hidden Layers Part 1 A

This appendix will document the test of different hidden layers. The test will be done on created data set seen on fig. 5.1 described in section 5.2 with the parameters seen in table 2.1.

A.1 Static Parameters

The static parameters for this test can been seen in table A.1.

	Hyper parameter	Value Space
DQN	Hidden Layers	To be tuned
	Learning Rate (ADAM)	0.01
	Memory Size	1000
	Batch Size	500
	Target Update	10
RL	Forgetting Factor (γ)	0.7
	Exploring Factor (ϵ)	0.01
	State Space	$[a_{-1}, a_{-2}, a_{-3}, \theta_0, \theta_{-1}, \theta_{-2}, x_0, x_{-1}, x_{-2}, y_0, y_{-1}, y_{-2}]$

Table A.1: Hyper parameter for the RL agent based on DQN.

A.2 Test Parameters

This test will test seven different hidden layers combination as seen in table A.2.

Test No.	Hidden Layers
1	[100, 100]
2	[200, 200]
3	[512, 512]
4	[512, 512, 512, 512, 512]
5	[500, 1000, 1000, 512]
6	[500, 1000, 1500, 1024]
7	[500, 1000, 1500, 2000]

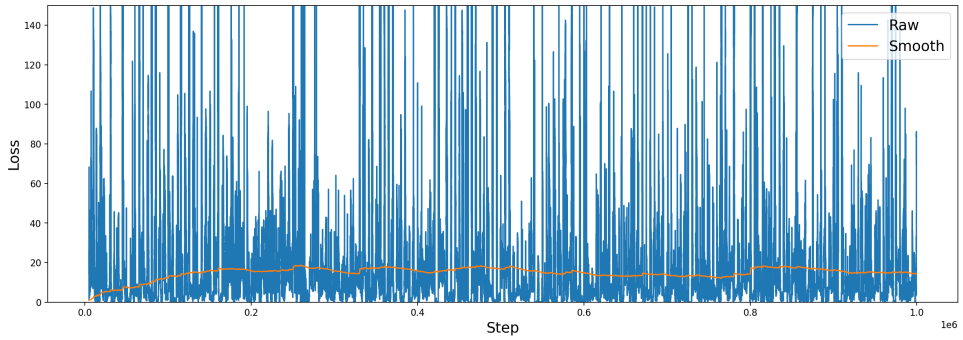
Table A.2: Different hidden layer sizes tested.

A.3 Results

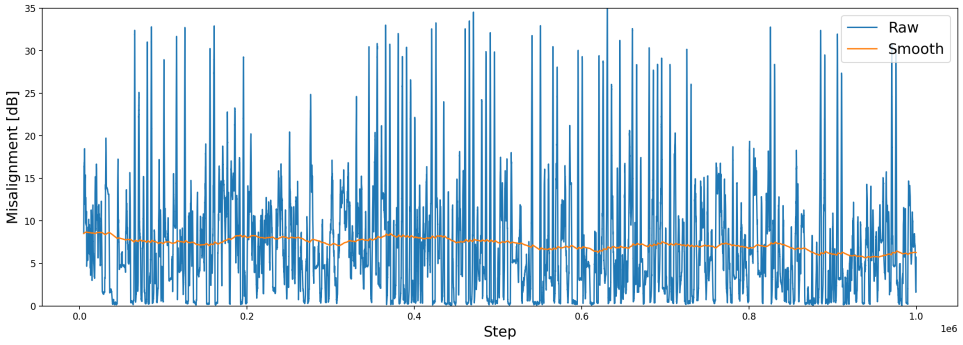
Test No.	Avg. Loss	Avg. Misalignment [dB]
1	16.117	7.076
2	16.744	7.265
3	21.613	7.425
4	29.454	7.972
5	27.399	7.636
6	34.937	8.683
7	75.002	10.723

Table A.3: Results for testing different hidden layers.

A.3.1 Test No. 1



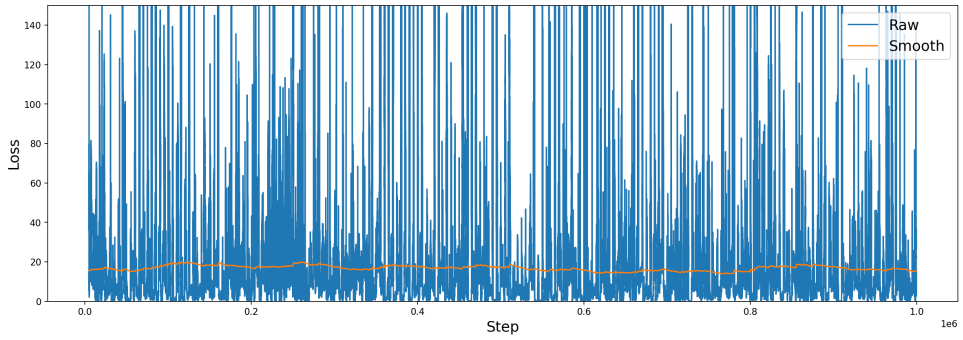
(a) Loss values.



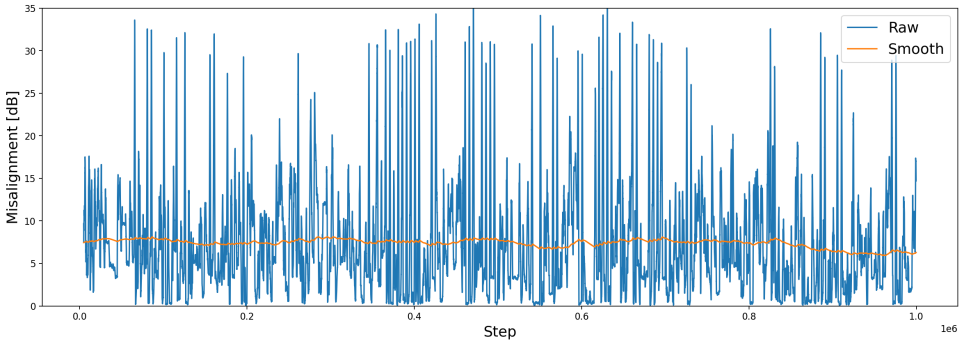
(b) Average misalignment for last 1000 steps.

Figure A.1: Results for test 1. Test parameter: Hidden Layers = [100, 100].

A.3.2 Test No. 2



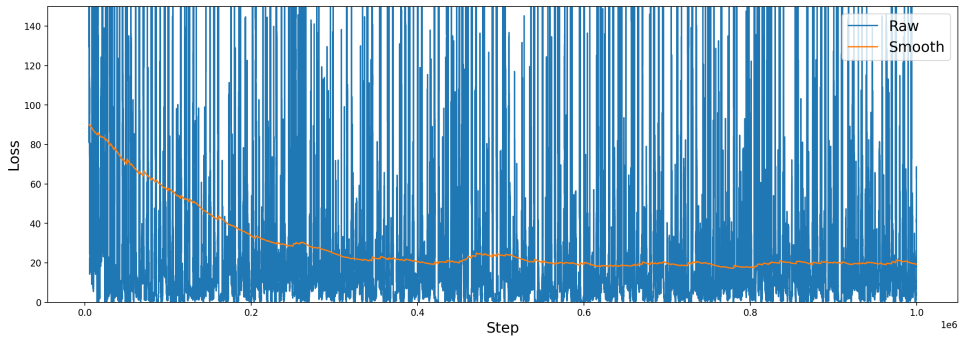
(a) Loss values.



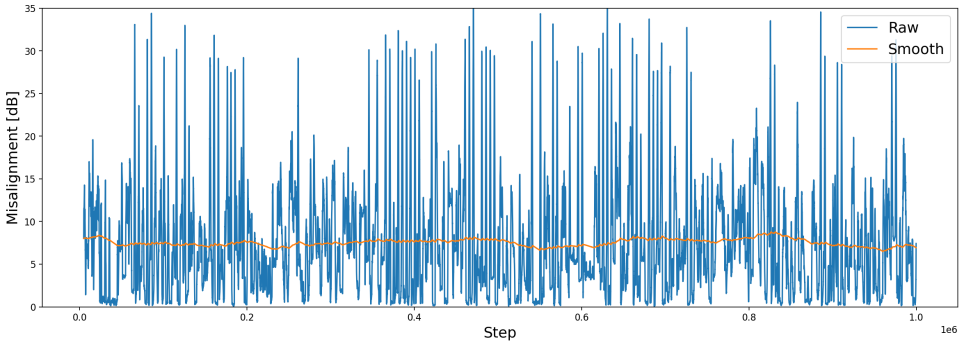
(b) Average misalignment for last 1000 steps.

Figure A.2: Results for test 2. Test parameter: Hidden Layers = [200, 200].

A.3.3 Test No. 3



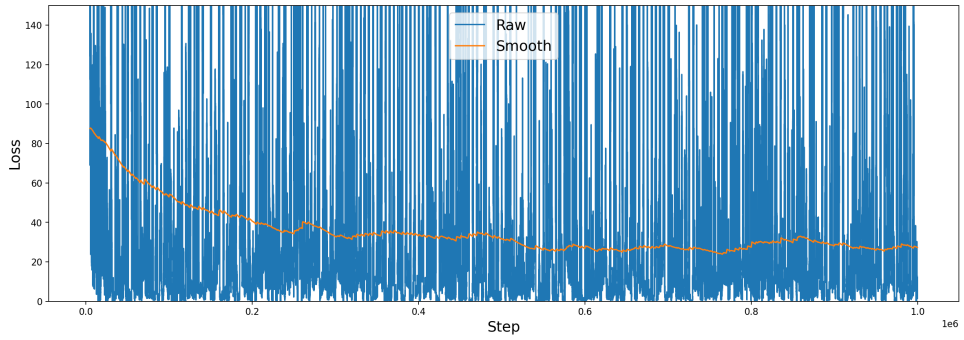
(a) Loss values.



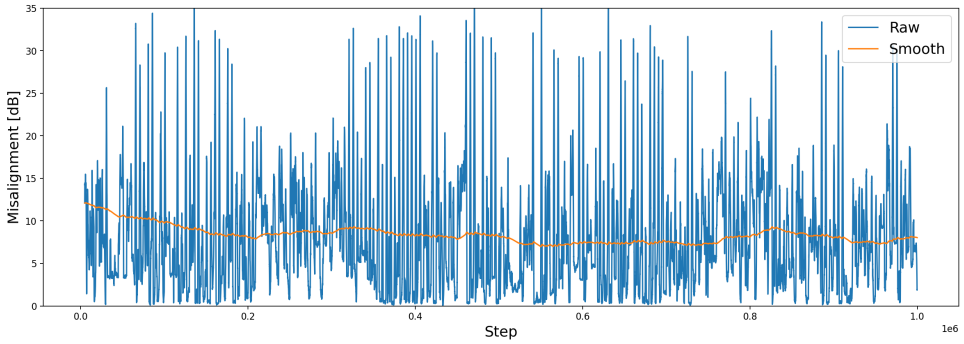
(b) Average misalignment for last 1000 steps.

Figure A.3: Results for test 3. Test parameter: Hidden Layers = [512, 512].

A.3.4 Test No. 4



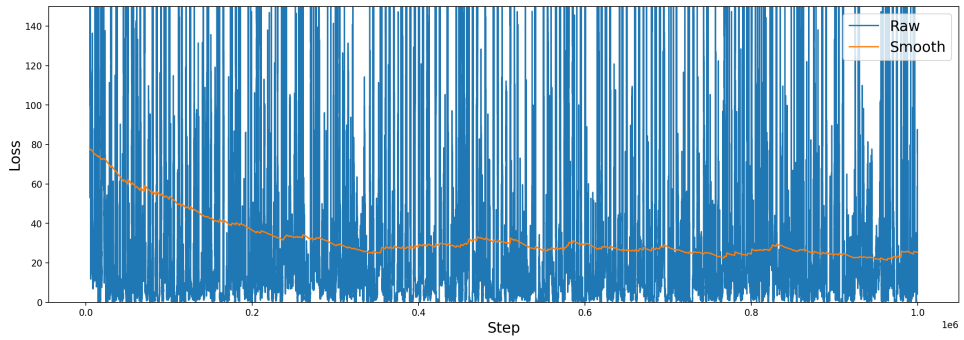
(a) Loss values.



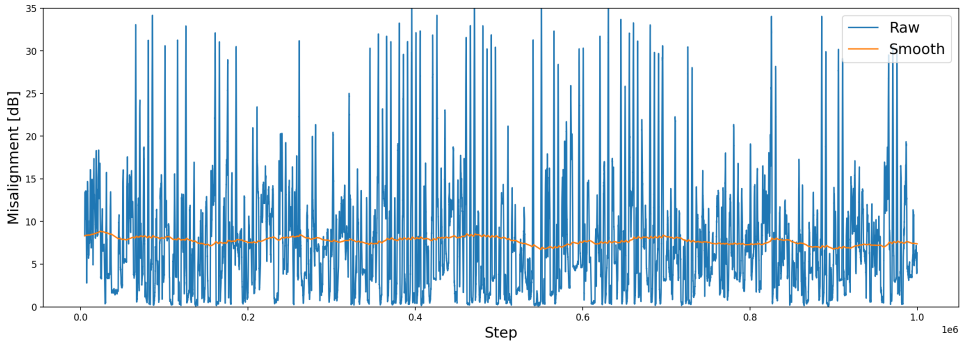
(b) Average misalignment for last 1000 steps.

Figure A.4: Results for test 4. Test parameter: Hidden Layers = [512, 512, 512, 512, 512].

A.3.5 Test No. 5



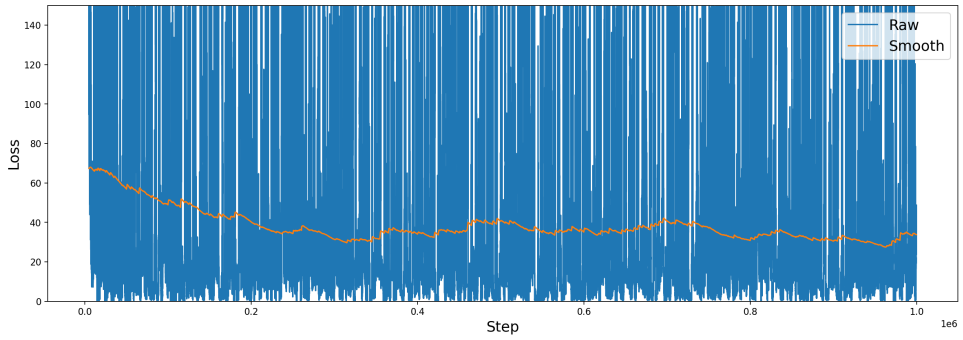
(a) Loss values.



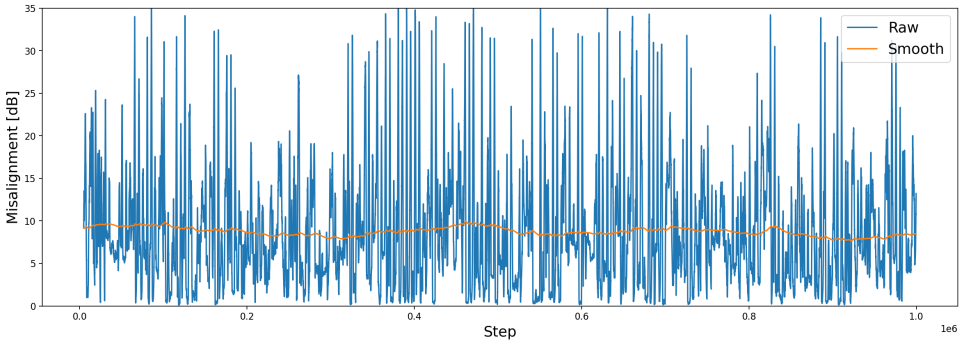
(b) Average misalignment for last 1000 steps.

Figure A.5: Results for test 5. Test parameter: Hidden Layers = [500, 1000, 1000, 512].

A.3.6 Test No. 6



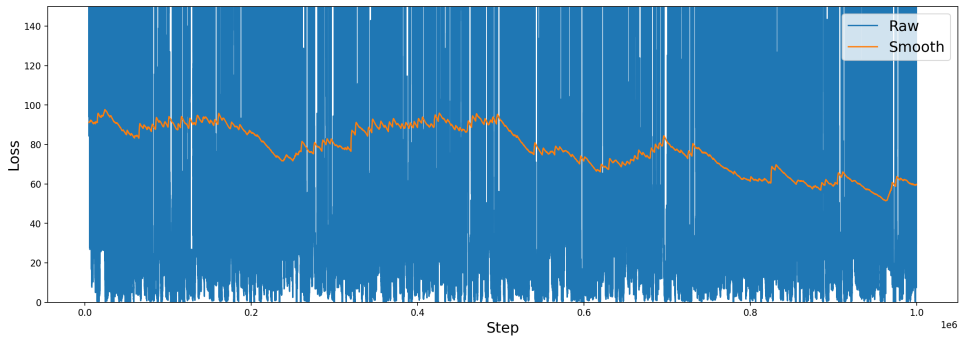
(a) Loss values.



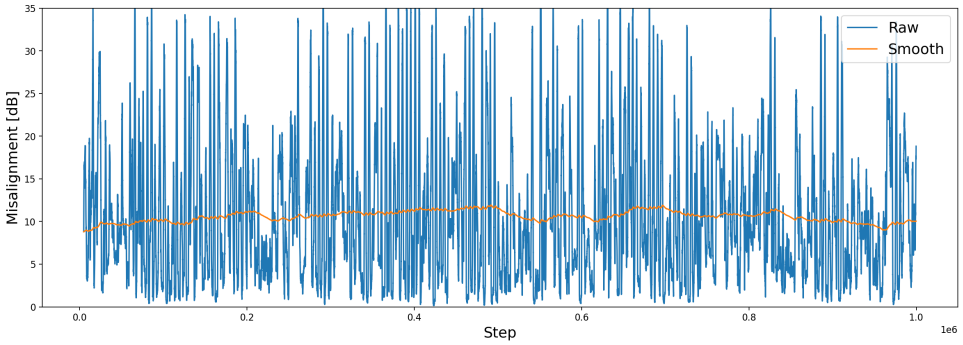
(b) Average misalignment for last 1000 steps.

Figure A.6: Results for test 6. Test parameter: Hidden Layers = [500, 1000, 1500, 1024].

A.3.7 Test No. 7



(a) Loss values.



(b) Average misalignment for last 1000 steps.

Figure A.7: Results for test 7. Test parameter: Hidden Layers = [500, 1000. 1500, 2000].

Test: Hidden Layers Part 2 B

This appendix will document the test of different hidden layers for the second test. The test will be done on created data set seen on fig. 5.1 described in section 5.2 with the parameters seen in table 2.1.

B.1 Static Parameters

The static parameters for this test can been seen in table B.1.

	Hyper parameter	Value Space
DQN	Hidden Layers	To be tuned
	Learning Rate (ADAM)	0.01
	Memory Size	1000
	Batch Size	500
	Target Update	10
RL	Forgetting Factor (γ)	0.7
	Exploring Factor (ϵ)	0.01
	State Space	$[a_{-1}, a_{-2}, a_{-3}, \theta_0, \theta_{-1}, \theta_{-2}, x_0, x_{-1}, x_{-2}, y_0, y_{-1}, y_{-2}]$

Table B.1: Hyper parameter for the RL agent based on DQN.

B.2 Test Parameters

This test will test seven different hidden layers combination as seen in table B.2.

Test No.	Hidden Layers
1	[20, 20]
2	[50, 50]
3	[50, 100]
4	[20, 30, 40]
5	[100, 150]
6	[150, 150]
7	[75, 100, 125]

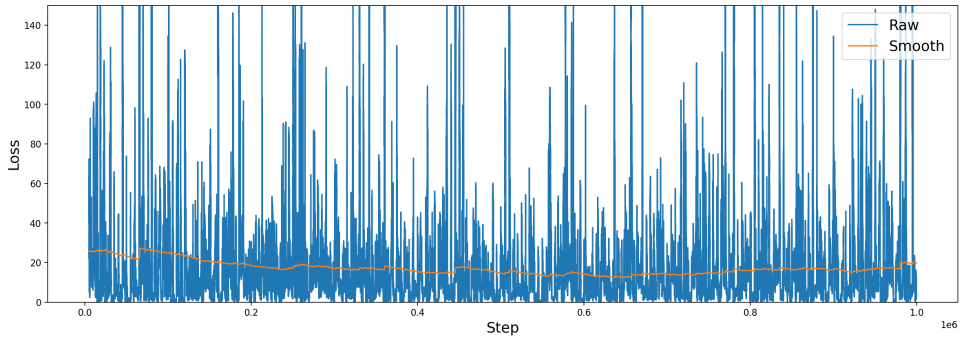
Table B.2: Different hidden layer sizes tested.

B.3 Results

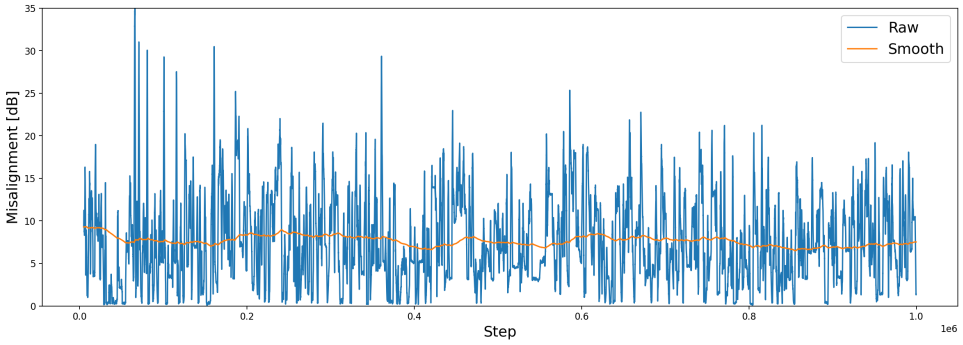
Test No.	Avg. Loss	Avg. Misalignment [dB]
1	16.673	7.480
2	16.311	6.501
3	15.262	6.962
4	18.091	7.394
5	16.186	7.416
6	16.638	7.317
7	18.106	7.220

Table B.3: Results for testing different hidden layers.

B.3.1 Test No. 1



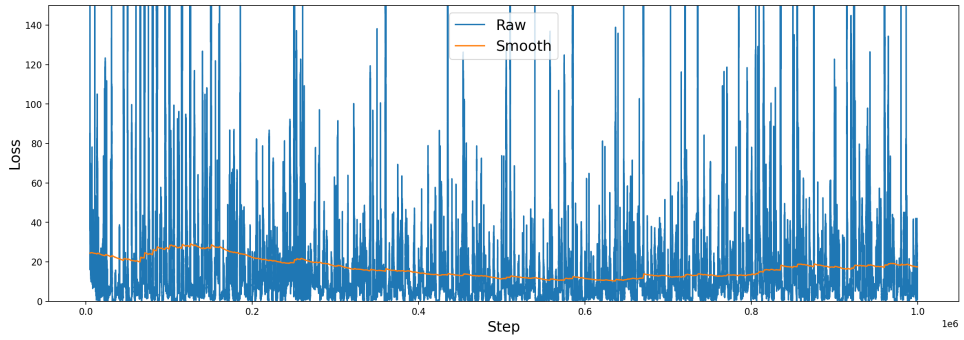
(a) Loss values.



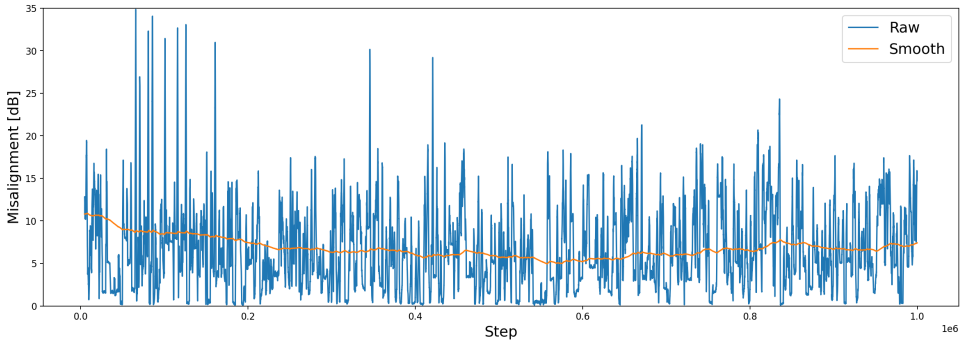
(b) Average misalignment for last 1000 steps.

Figure B.1: Results for test 1. Test parameter: Hidden Layers = [20, 20].

B.3.2 Test No. 2



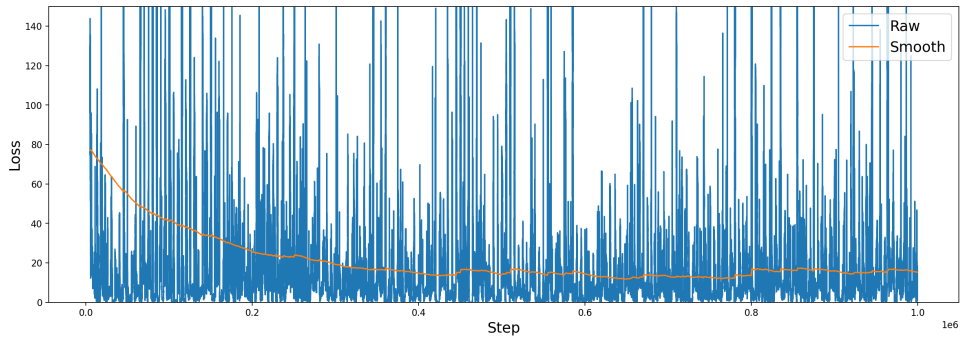
(a) Loss values.



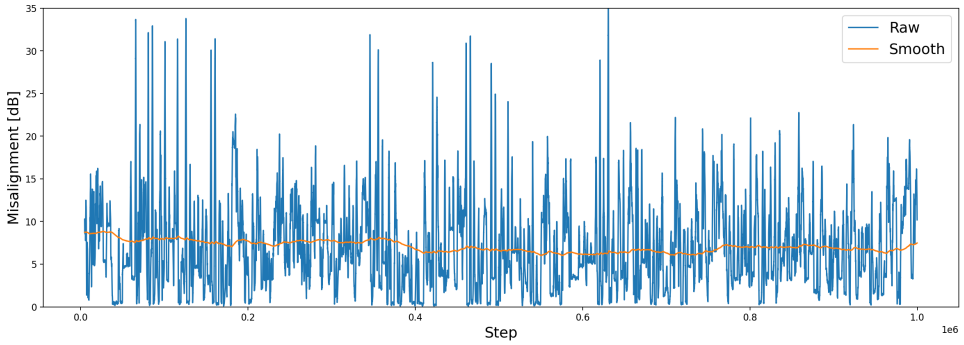
(b) Average misalignment for last 1000 steps.

Figure B.2: Results for test 2. Test parameter: Hidden Layers = [50, 50].

B.3.3 Test No. 3



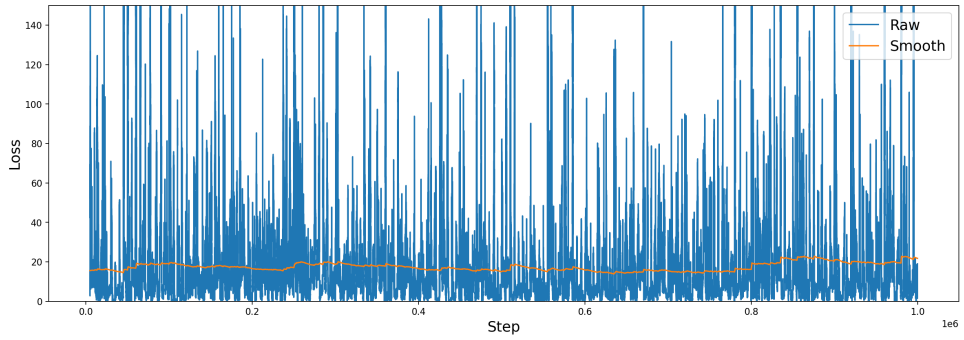
(a) Loss values.



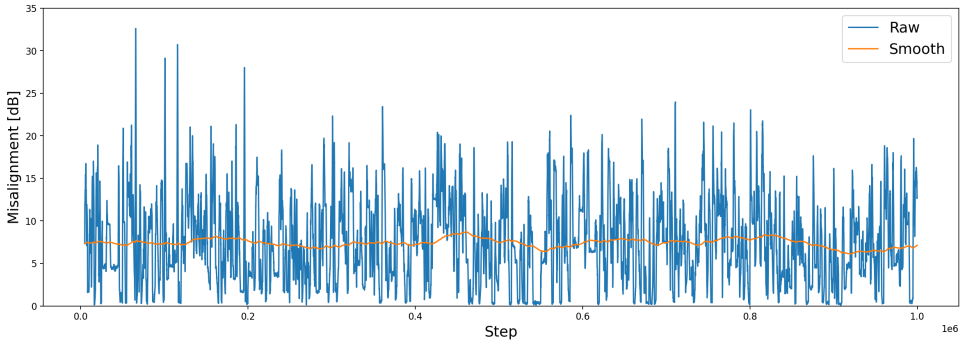
(b) Average misalignment for last 1000 steps.

Figure B.3: Results for test 3. Test parameter: Hidden Layers = [50, 100].

B.3.4 Test No. 4



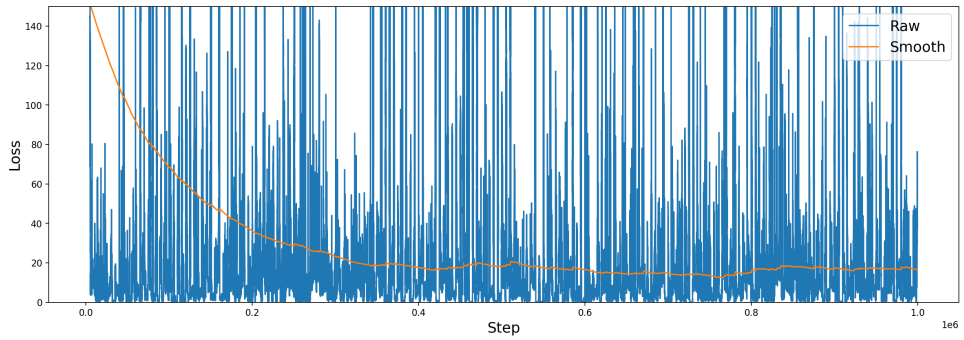
(a) Loss values.



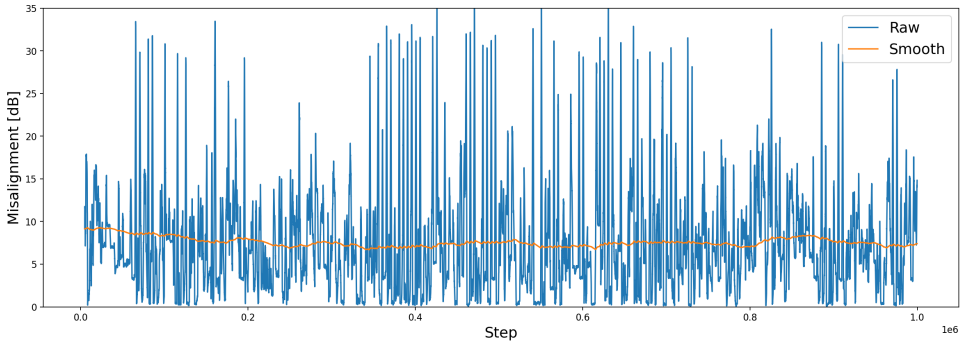
(b) Average misalignment for last 1000 steps.

Figure B.4: Results for test 4. Test parameter: Hidden Layers = [20, 30, 40].

B.3.5 Test No. 5



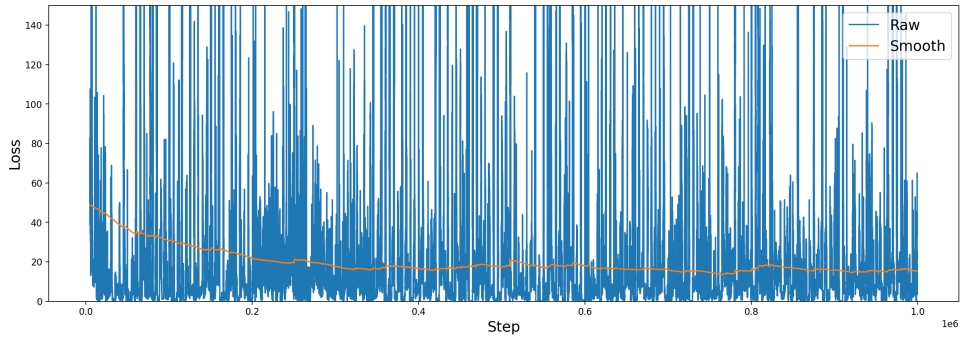
(a) Loss values.



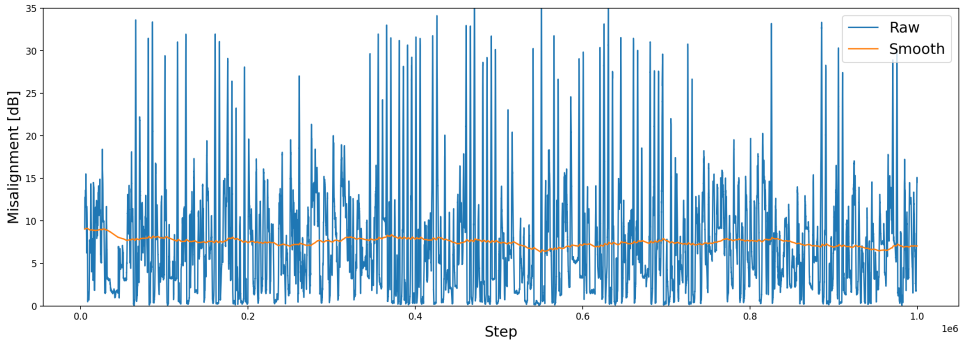
(b) Average misalignment for last 1000 steps.

Figure B.5: Results for test 5. Test parameter: Hidden Layers = [100, 150].

B.3.6 Test No. 6



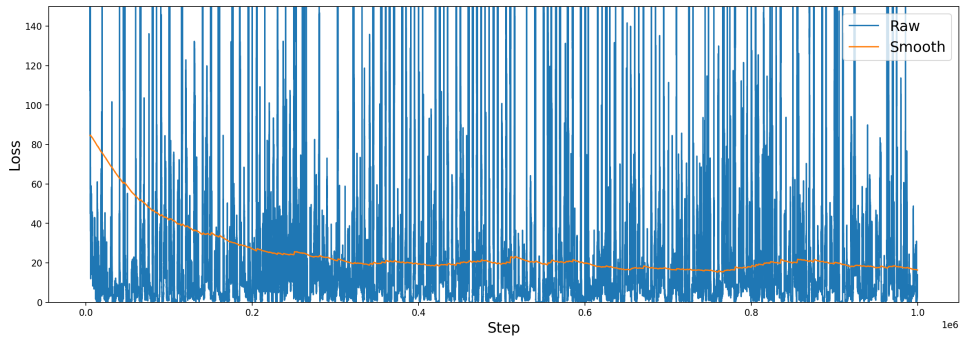
(a) Loss values.



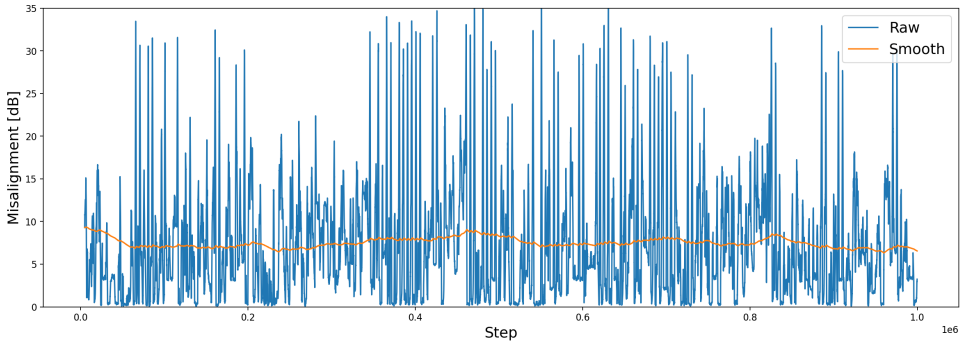
(b) Average misalignment for last 1000 steps.

Figure B.6: Results for test 6. Test parameter: Hidden Layers = [150, 150].

B.3.7 Test No. 7



(a) Loss values.



(b) Average misalignment for last 1000 steps.

Figure B.7: Results for test 7. Test parameter: Hidden Layers = [75, 100, 125].

Test: Embedding Layer C

This appendix will document the test of adding an embedding layer to the NN. The test will be done on created data set seen on fig. 5.1 described in section 5.2 with the parameters seen in table 2.1.

C.1 Static Parameters

The static parameters for this test can been seen in table C.1.

	Hyper parameter	Value Space
DQN	Hidden Layers	[50, 50]
	Learning Rate (ADAM)	0.01
	Memory Size	1000
	Batch Size	500
	Target Update	10
RL	Forgetting Factor (γ)	0.7
	Exploring Factor (ϵ)	0.01
	State Space	$[a_{-1}, a_{-2}, a_{-3}, \theta_0, \theta_{-1}, \theta_{-2}, x_0, x_{-1}, x_{-2}, y_0, y_{-1}, y_{-2}]$

Table C.1: Hyper parameter for the RL agent based on DQN.

C.2 Test Parameters

This test will test seven different output dimensions for the added embedding layer. The output dimensions can be seen in table C.2.

Test No.	Output Dim.
1	1
2	2
3	3
4	4
5	5
6	6
7	7

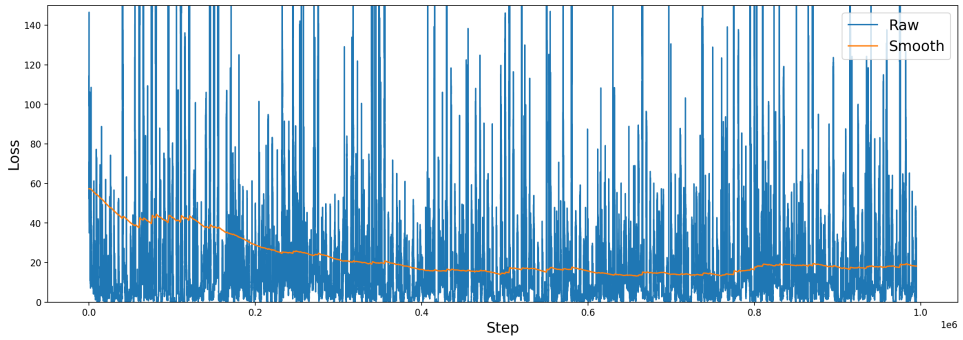
Table C.2: Different output dimension sizes for the dense vector in the embedding layer is tested on.

C.3 Results

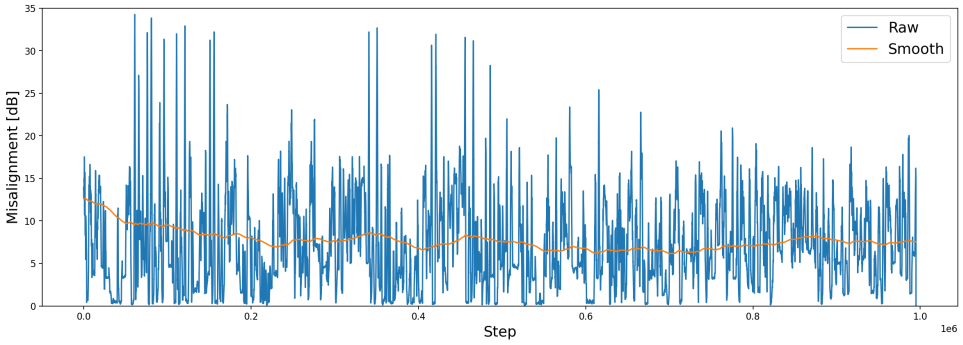
Test No.	Avg. Loss	Avg. Misalignment [dB]
1	18.529	7.236
2	18.445	7.044
3	18.695	6.677
4	18.878	6.937
5	17.367	7.114
6	19.040	7.255
7	19.221	6.995

Table C.3: Results for testing different output dimensions for the added embedding layer.

C.3.1 Test No. 1



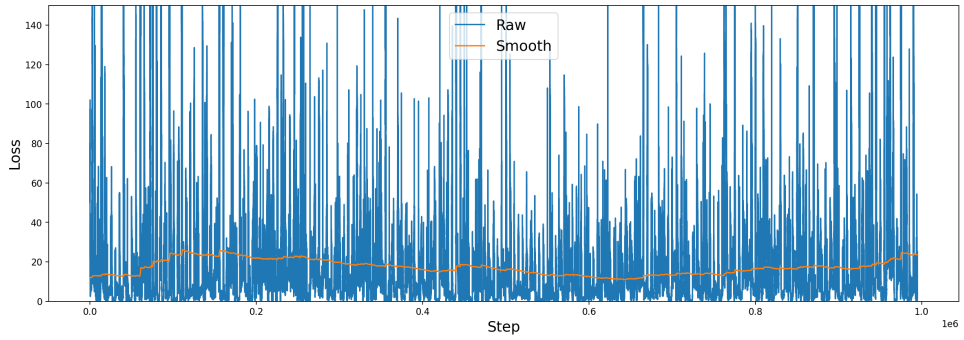
(a) Loss values.



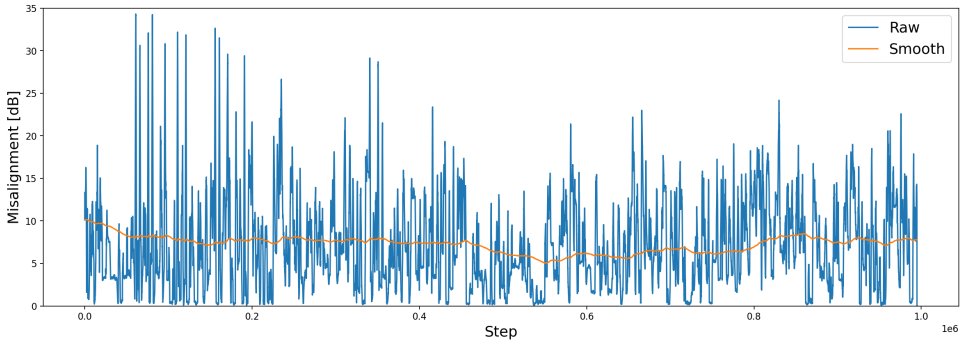
(b) Average misalignment for last 1000 steps.

Figure C.1: Results for test 1. Test parameter: Output dimension = 1.

C.3.2 Test No. 2



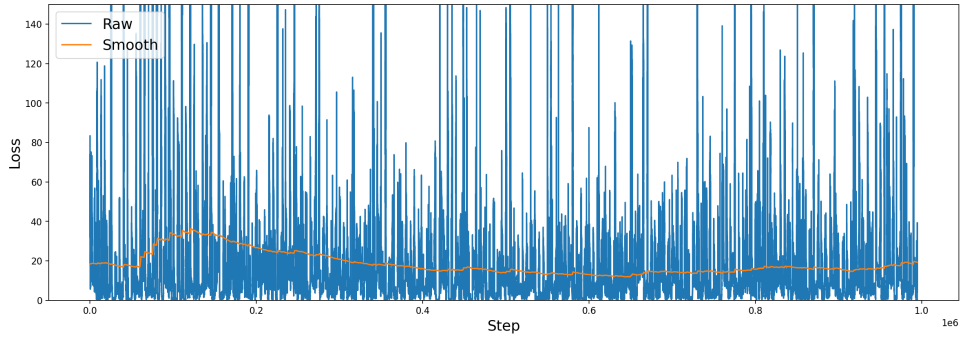
(a) Loss values.



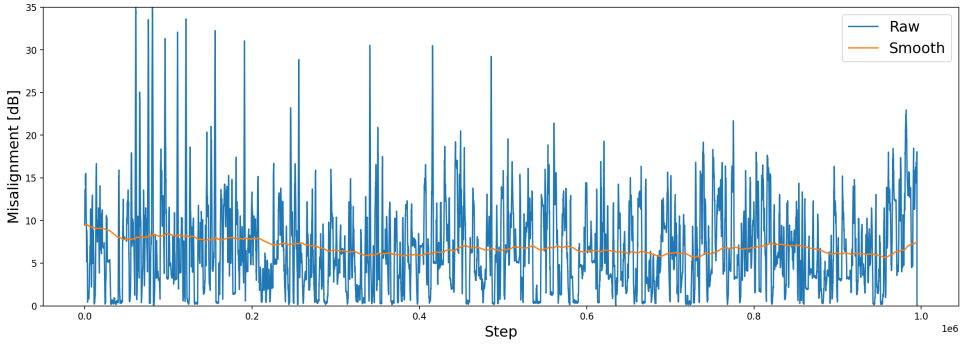
(b) Average misalignment for last 1000 steps.

Figure C.2: Results for test 2. Test parameter: Output dimension = 2.

C.3.3 Test No. 3



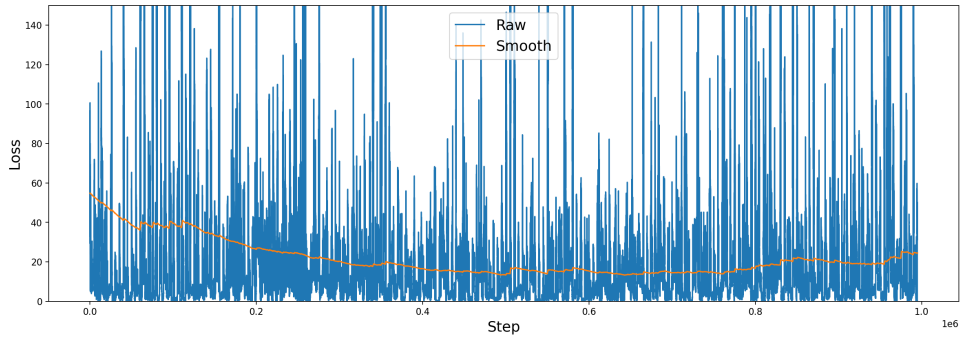
(a) Loss values.



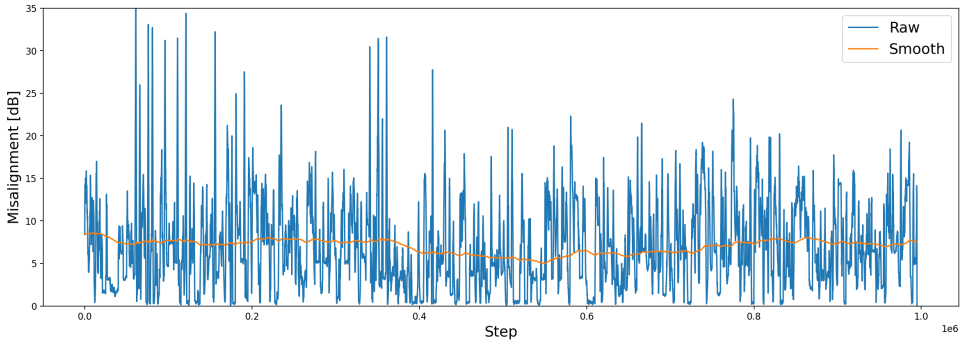
(b) Average misalignment for last 1000 steps.

Figure C.3: Results for test 3. Test parameter: Output dimension = 3.

C.3.4 Test No. 4



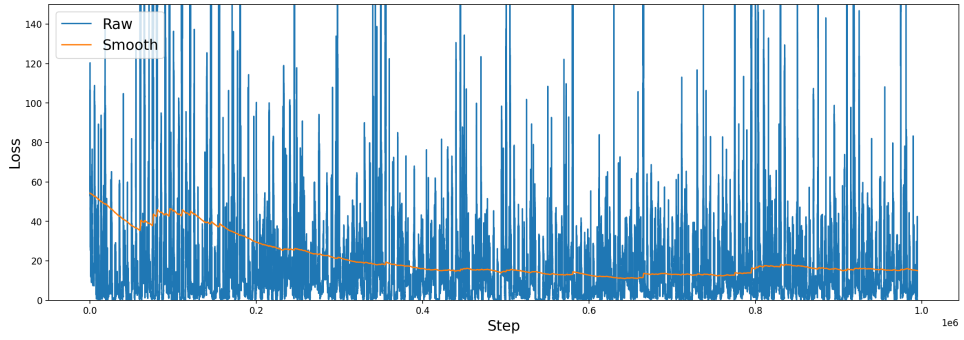
(a) Loss values.



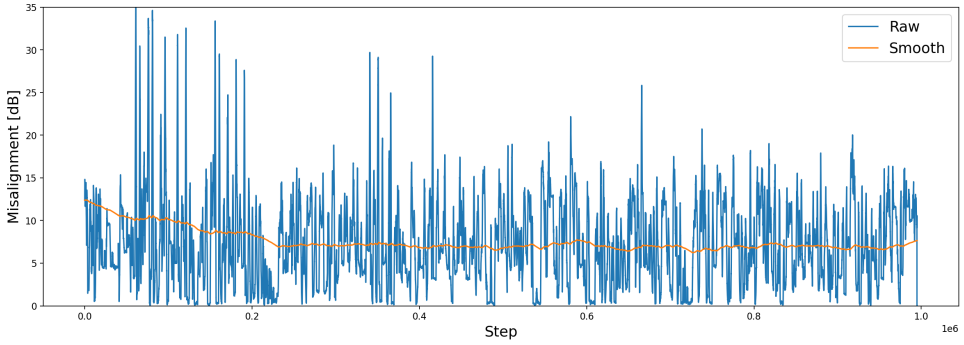
(b) Average misalignment for last 1000 steps.

Figure C.4: Results for test 4. Test parameter: Output dimension = 4.

C.3.5 Test No. 5



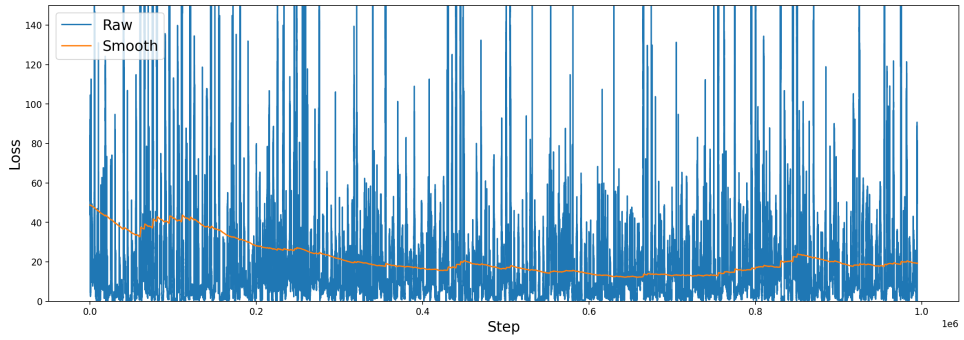
(a) Loss values.



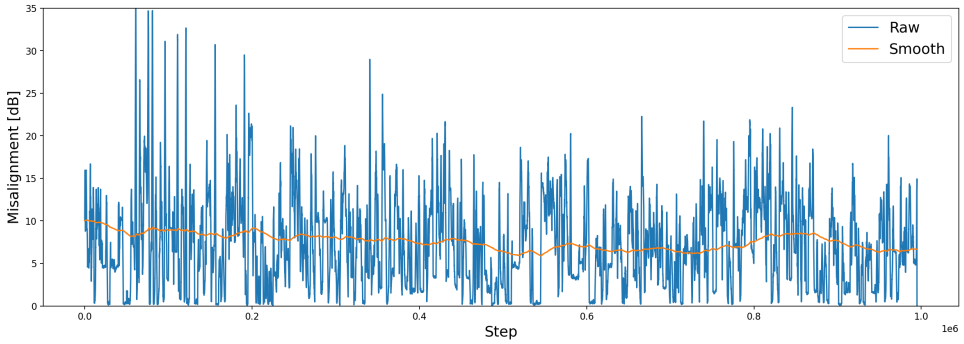
(b) Average misalignment for last 1000 steps.

Figure C.5: Results for test 5. Test parameter: Output dimension = 5.

C.3.6 Test No. 6



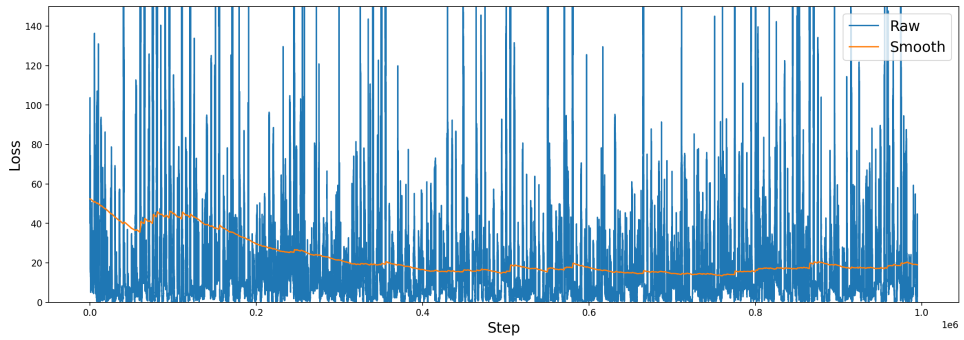
(a) Loss values.



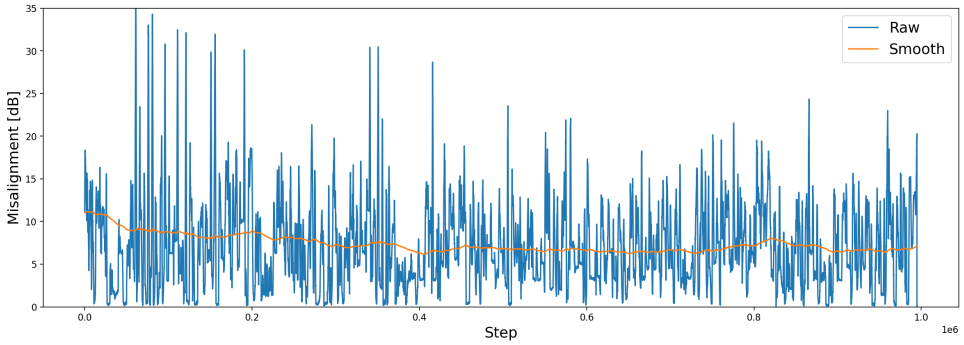
(b) Average misalignment for last 1000 steps.

Figure C.6: Results for test 6. Test parameter: Output dimension = 6.

C.3.7 Test No. 7



(a) Loss values.



(b) Average misalignment for last 1000 steps.

Figure C.7: Results for test 7. Test parameter: Output dimension = 7.

Test: Batch and Memory Size

D

This appendix will document the test of different batch and memory size values. The test will be done on created data set seen on fig. 5.1 described in section 5.2 with the parameters seen in table 2.1.

D.1 Static Parameters

The static parameters for this test can been seen in table D.1.

	Hyper parameter	Value Space
DQN	Hidden Layers	[50, 50]
	Learning Rate (ADAM)	0.01
	Memory Size	To be tuned
	Batch Size	To be tuned
	Target Update	10
RL	Forgetting Factor (γ)	0.7
	Exploring Factor (ϵ)	0.01
	State Space	$[a_{-1}, a_{-2}, a_{-3}, \theta_0, \theta_{-1}, \theta_{-2}, x_0, x_{-1}, x_{-2}, y_0, y_{-1}, y_{-2}]$

Table D.1: Hyper parameter for the RL agent based on DQN.

D.2 Test Parameters

This test will test seven different combination of batch and memory sizes as seen in table D.2.

Test No.	Batch	Memory
1	64	128
2	64	256
3	100	1000
4	250	500
5	500	500
6	500	2000
7	1000	3000

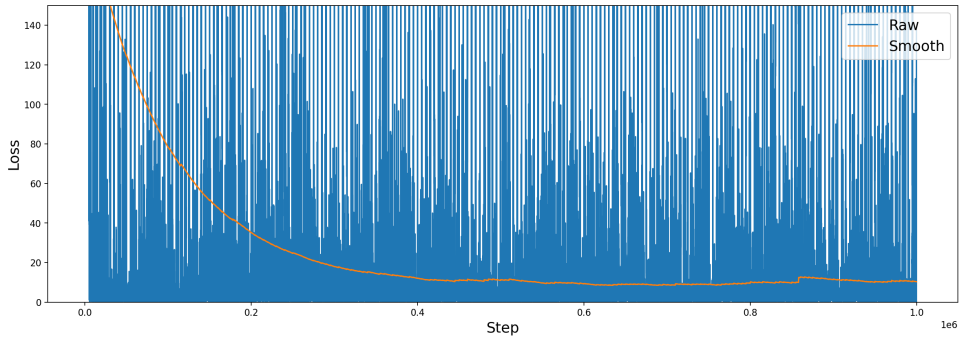
Table D.2: Different batch and memory sizes tested.

D.3 Results

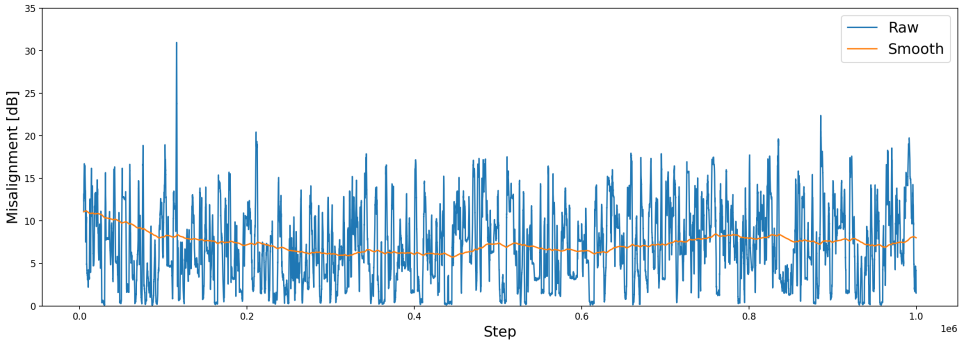
Test No.	Avg. Loss	Avg. Misalignment. [dB]
1	9.462	7.050
2	12.946	7.489
3	28.675	8.690
4	16.225	7.434
5	15.609	7.672
6	39.904	8.787
7	37.448	9.068

Table D.3: Results for testing different batch and memory sizes.

D.3.1 Test No. 1



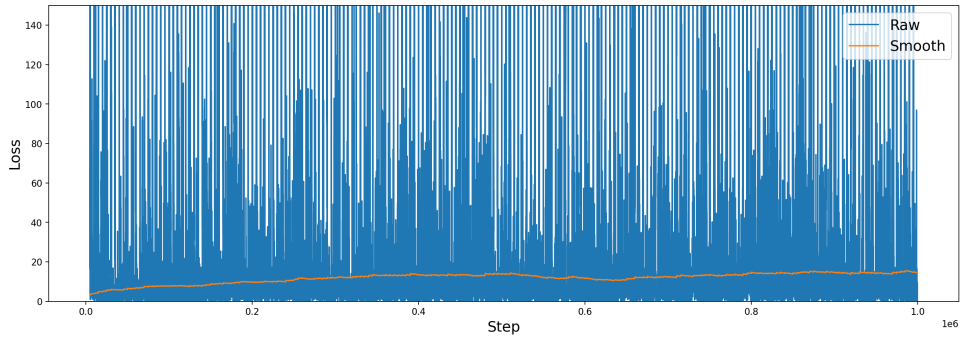
(a) Loss values.



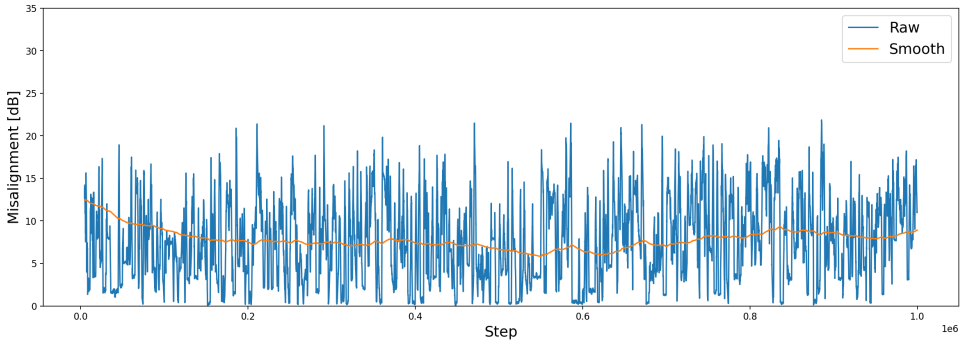
(b) Average misalignment for last 1000 steps.

Figure D.1: Results for test 1. Test parameter: Batch = 64, Memory = 128.

D.3.2 Test No. 2



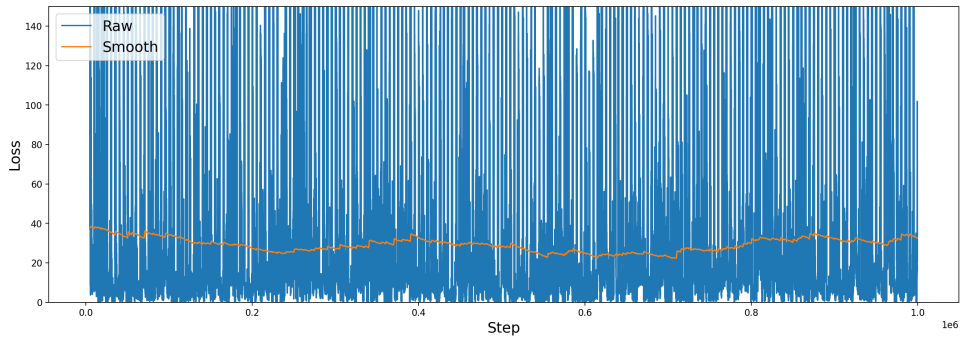
(a) Loss values.



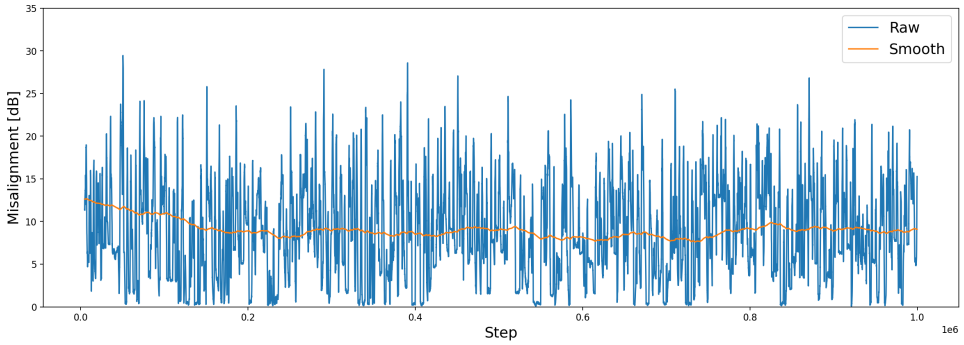
(b) Average misalignment for last 1000 steps.

Figure D.2: Results for test 2. Test parameter: Batch = 64, Memory = 256.

D.3.3 Test No. 3



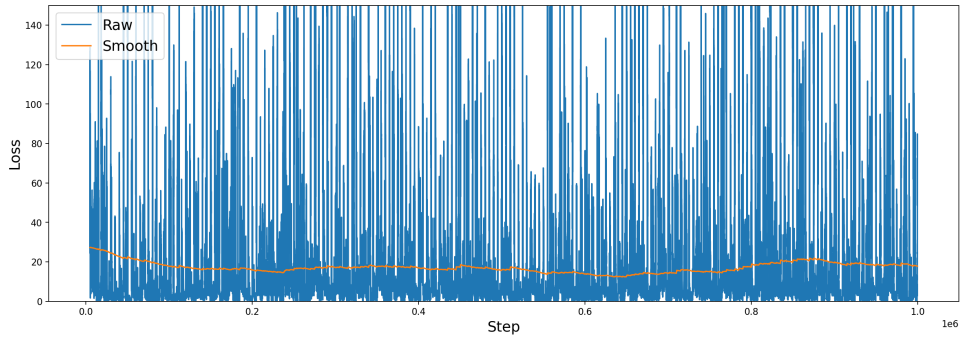
(a) Loss values.



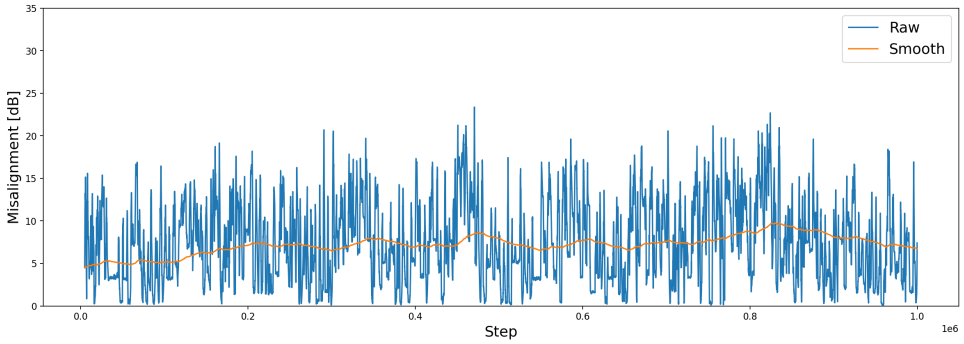
(b) Average misalignment for last 1000 steps.

Figure D.3: Results for test 3. Test parameter: Batch = 100, Memory = 1000.

D.3.4 Test No. 4



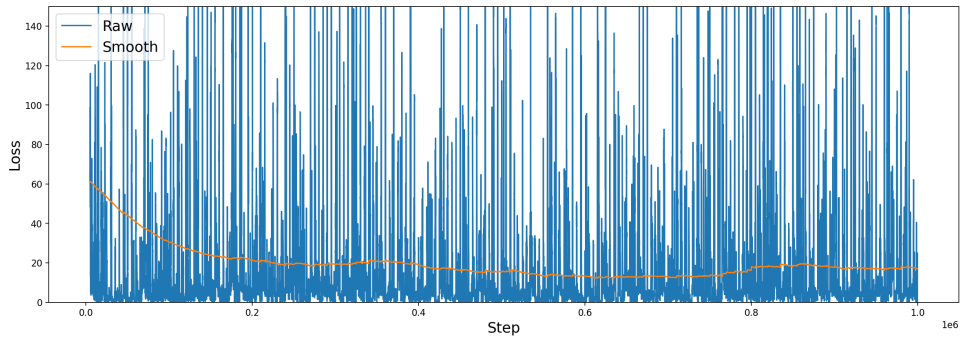
(a) Loss values.



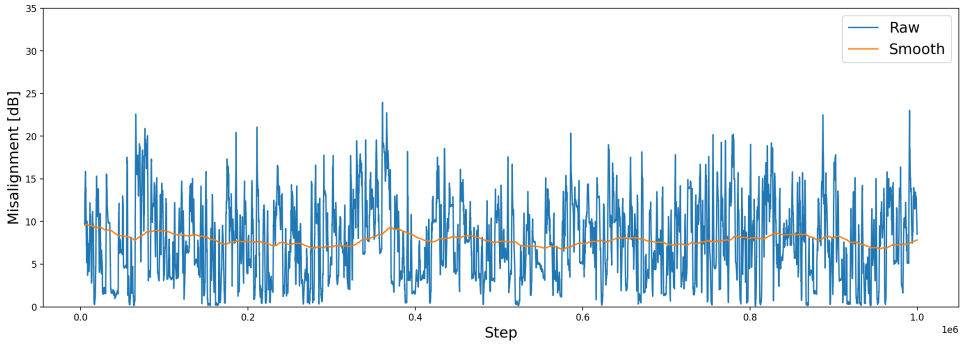
(b) Average misalignment for last 1000 steps.

Figure D.4: Results for test 4. Test parameter: Batch = 250, Memory = 500.

D.3.5 Test No. 5



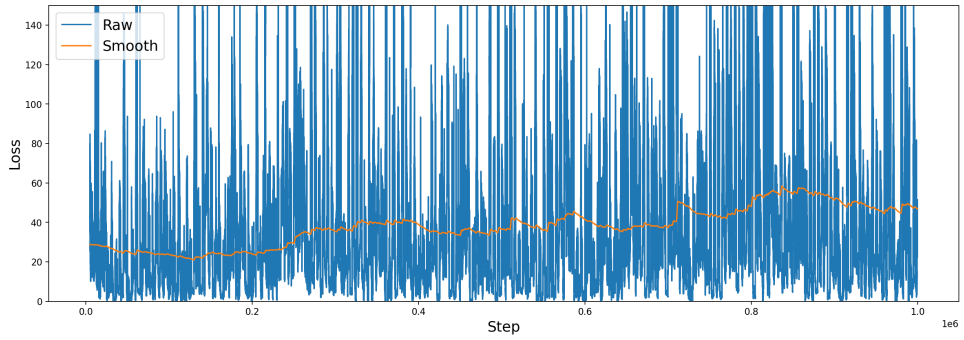
(a) Loss values.



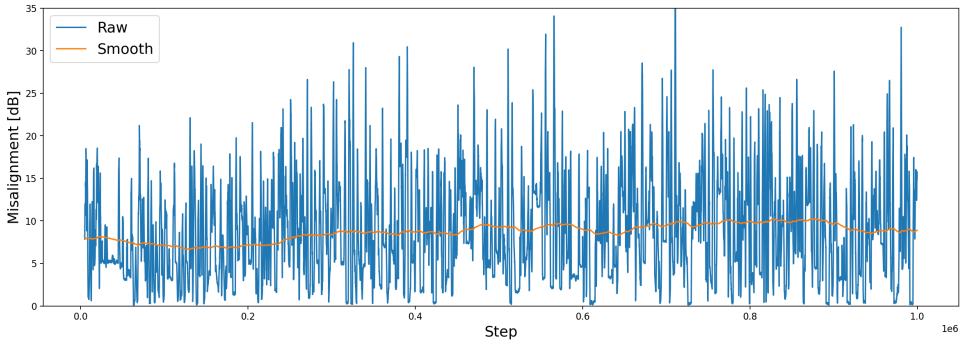
(b) Average misalignment for last 1000 steps.

Figure D.5: Results for test 5. Test parameter: Batch = 500, Memory = 500.

D.3.6 Test No. 6



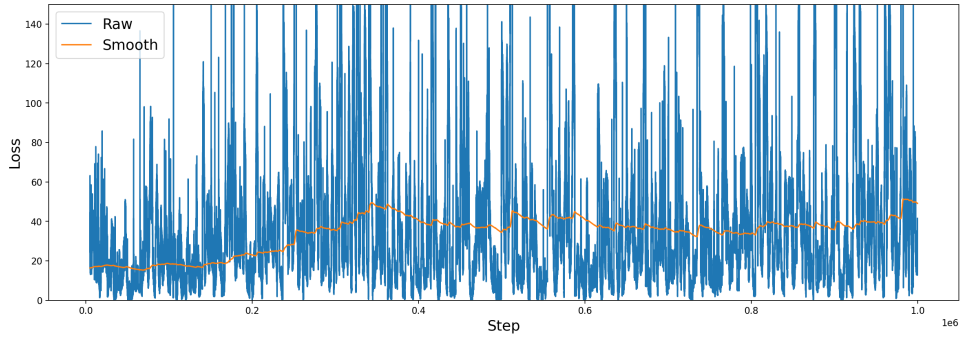
(a) Loss values.



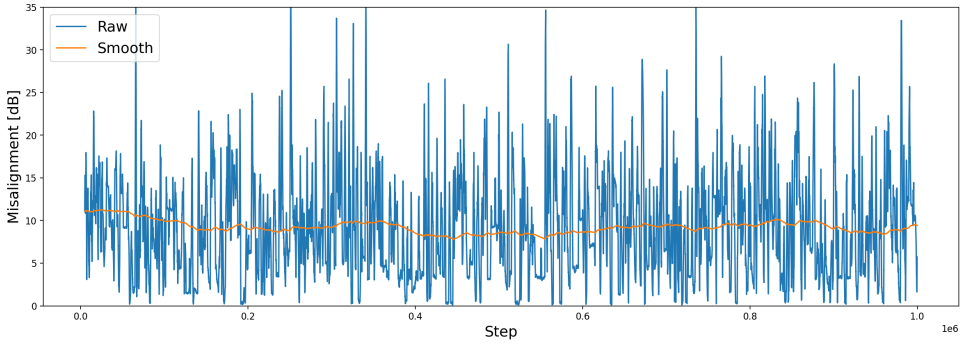
(b) Average misalignment for last 1000 steps.

Figure D.6: Results for test 6. Test parameter: Batch = 500, Memory = 2000.

D.3.7 Test No. 7



(a) Loss values.



(b) Average misalignment for last 1000 steps.

Figure D.7: Results for test 7. Test parameter: Batch = 1000, Memory = 3000.

Test: Target Update

This appendix will document the test of different target update values. The test will be done on the created data set seen on fig. 5.1 described in section 5.2 with the parameters seen in table 2.1.

E.1 Static Parameters

The static parameters for this test can been seen in table E.1.

	Hyper parameter	Value Space
DQN	Hidden Layers	[50, 50]
	Learning Rate (ADAM)	0.01
	Memory Size	1000
	Batch Size	500
	Target Update	To be tuned
RL	Forgetting Factor (γ)	0.7
	Exploring Factor (ϵ)	0.01
	State Space	$[a_{-1}, a_{-2}, a_{-3}, \theta_0, \theta_{-1}, \theta_{-2}, x_0, x_{-1}, x_{-2}, y_0, y_{-1}, y_{-2}]$

Table E.1: Hyper parameter for the RL agent based on DQN.

E.2 Test Parameters

This test will test seven different target update as seen in table E.2.

Test No.	Target Update
1	1
2	5
3	20
4	50
5	100
6	250
7	500

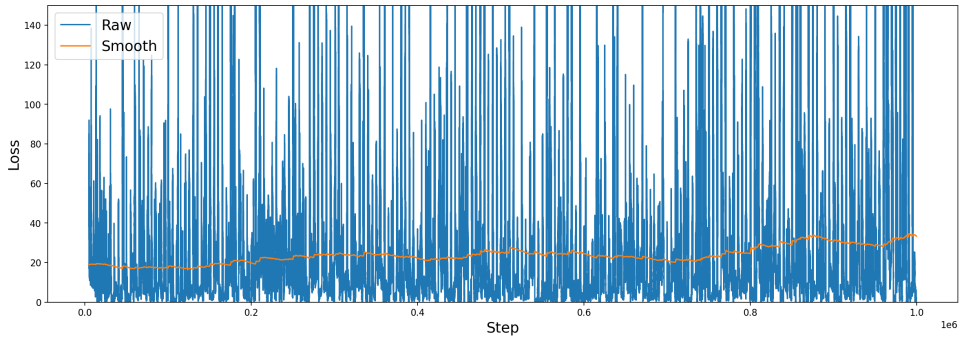
Table E.2: Different target updates tested.

E.3 Results

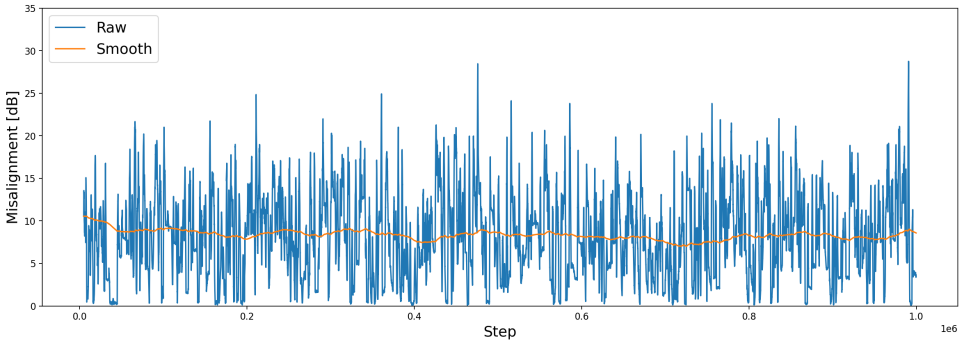
Test No.	Avg. Loss	Avg. Misalignment [dB]
1	25.264	8.165
2	23.199	7.920
3	24.046	8.417
4	25.309	7.954
5	28.113	9.187
6	22.864	8.917
7	25.455	10.528

Table E.3: Results for testing different target update frequencies.

E.3.1 Test No. 1



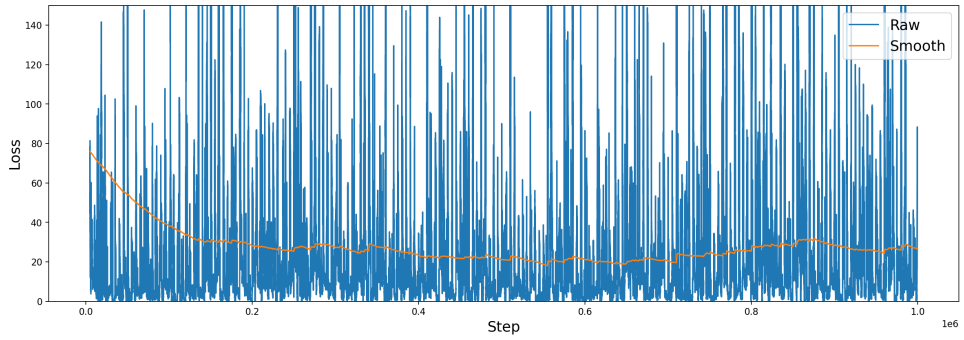
(a) Loss values.



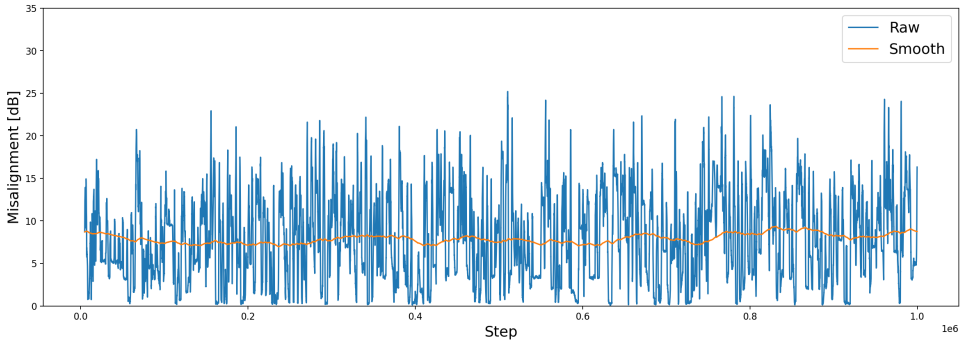
(b) Average misalignment for last 1000 steps.

Figure E.1: Results for test 1. Test parameter: Target Update = 1.

E.3.2 Test No. 2



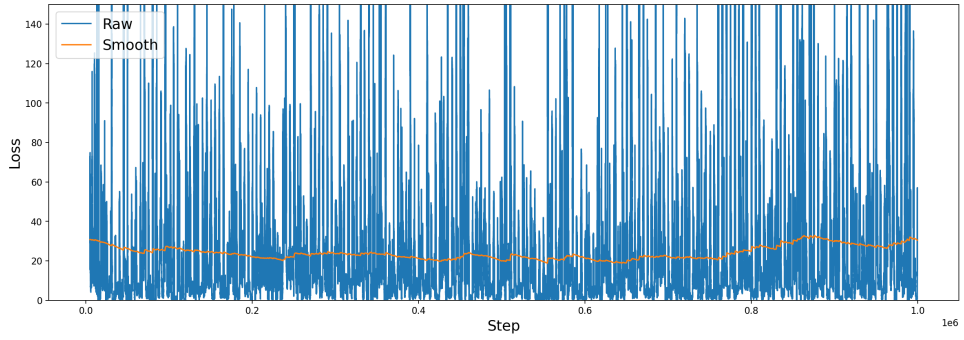
(a) Loss values.



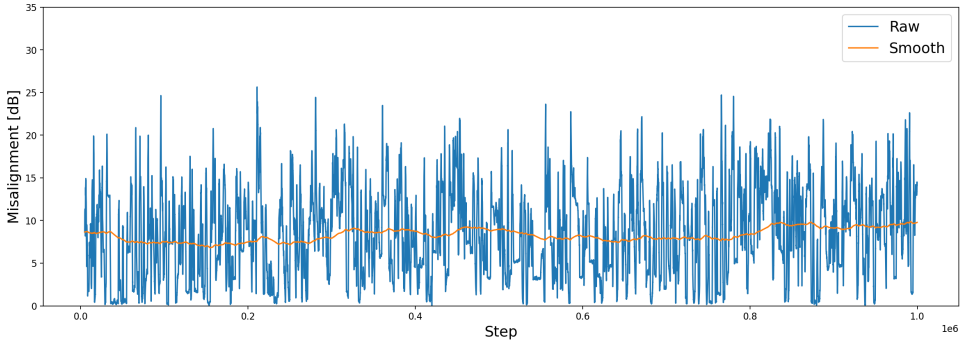
(b) Average misalignment for last 1000 steps.

Figure E.2: Results for test 2. Test parameter: Target Update = 5.

E.3.3 Test No. 3



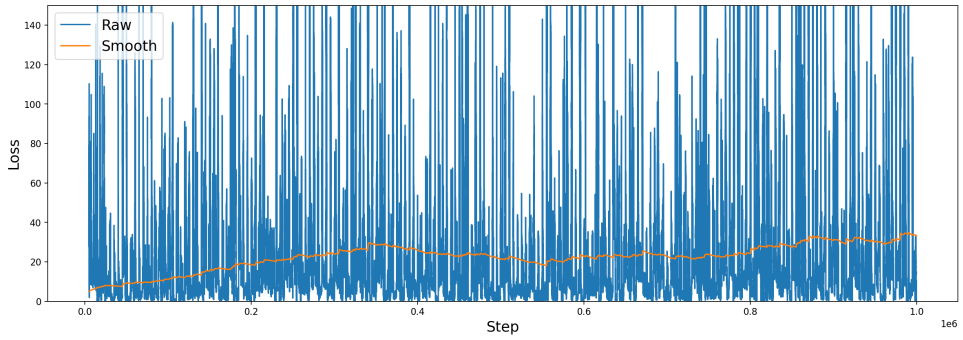
(a) Loss values.



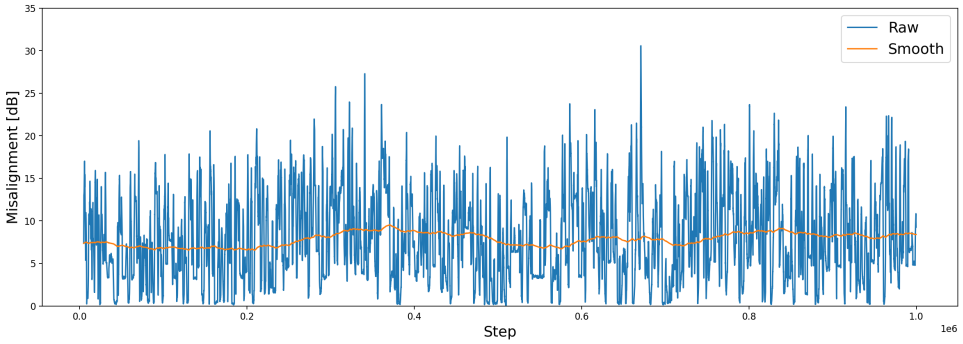
(b) Average misalignment for last 1000 steps.

Figure E.3: Results for test 3. Test parameter: Target Update = 20.

E.3.4 Test No. 4



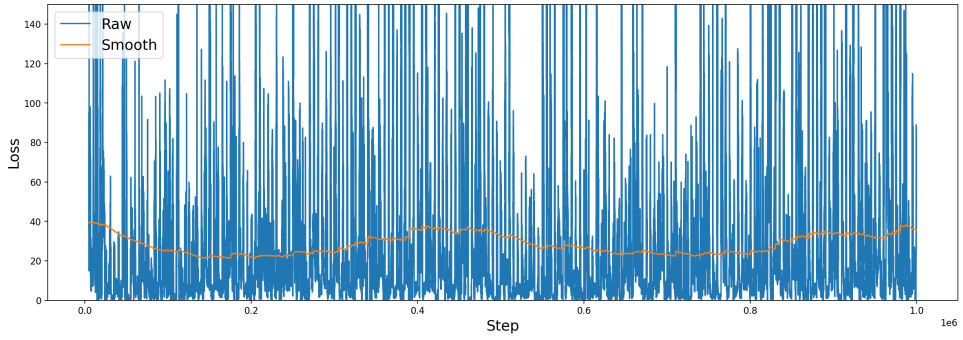
(a) Loss values.



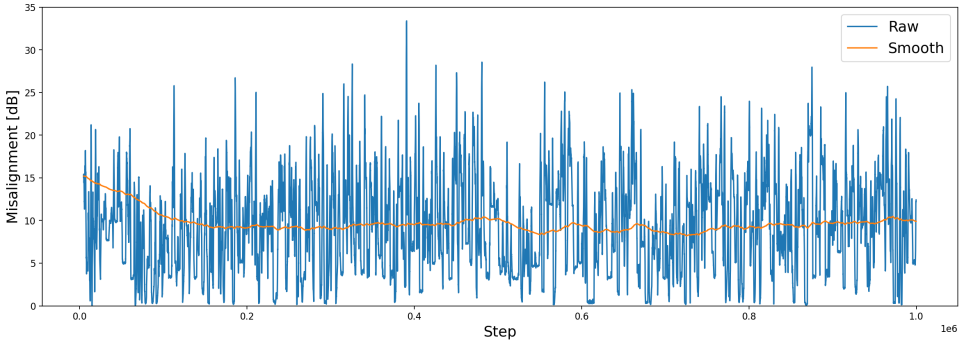
(b) Average misalignment for last 1000 steps.

Figure E.4: Results for test 4. Test parameter: Target Update = 50.

E.3.5 Test No. 5



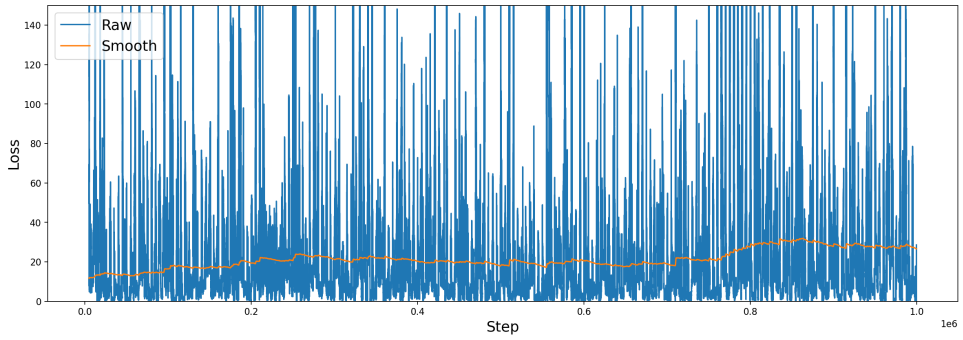
(a) Loss values.



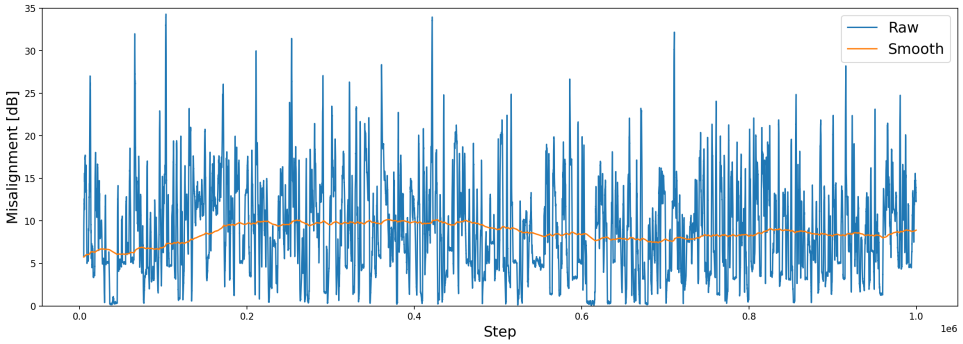
(b) Average misalignment for last 1000 steps.

Figure E.5: Results for test 5. Test parameter: Target Update = 100.

E.3.6 Test No. 6



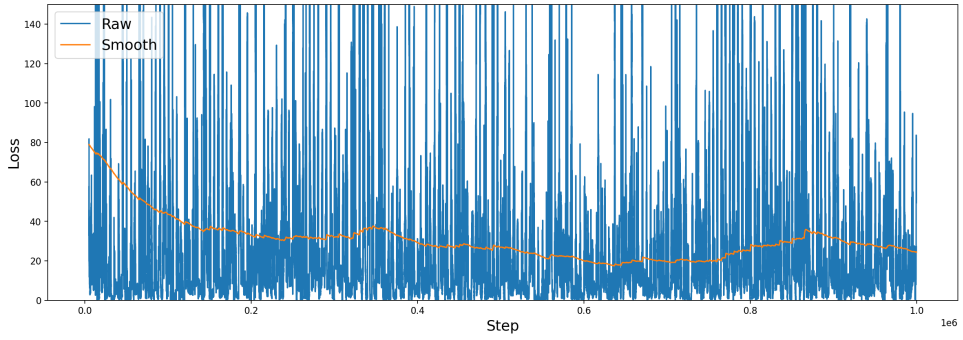
(a) Loss values.



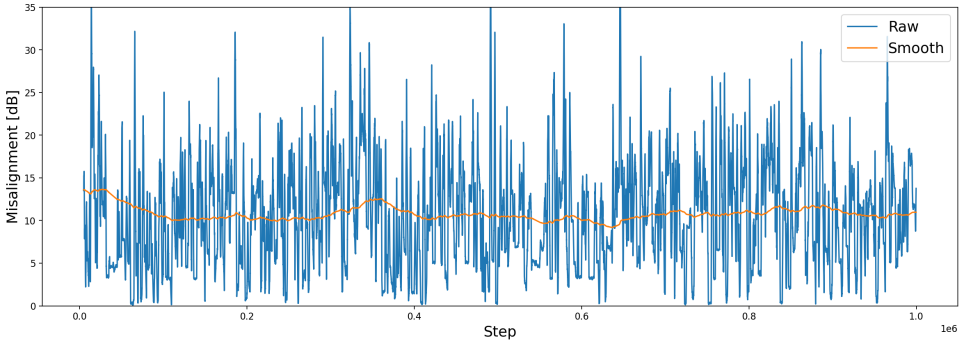
(b) Average misalignment for last 1000 steps.

Figure E.6: Results for test 6. Test parameter: Target Update = 250.

E.3.7 Test No. 7



(a) Loss values.



(b) Average misalignment for last 1000 steps.

Figure E.7: Results for test 7. Test parameter: Target Update = 500.

Test: Learning Rate F

This appendix will document the test of different Learning rates. The test will be done on the created data set seen on fig. 5.1 described in section 5.2 with the parameters seen in table 2.1.

F.1 Static Parameters

The static parameters for this test can been seen in table F.1.

	Hyper parameter	Value Space
DQN	Hidden Layers	[50, 50]
	Learning Rate (ADAM)	To be tuned
	Memory Size	1000
	Batch Size	500
	Target Update	10
RL	Forgetting Factor (γ)	0.7
	Exploring Factor (ϵ)	0.01
	State Space	$[a_{-1}, a_{-2}, a_{-3}, \theta_0, \theta_{-1}, \theta_{-2}, x_0, x_{-1}, x_{-2}, y_0, y_{-1}, y_{-2}]$

Table F.1: Hyper parameter for the RL agent based on DQN.

F.2 Test Parameters

This test will test seven different learning rates as seen in table F.2.

Test No.	Learning Rate
1	0.0001
2	0.001
3	0.05
4	0.1
5	0.3
6	0.7
7	0.9

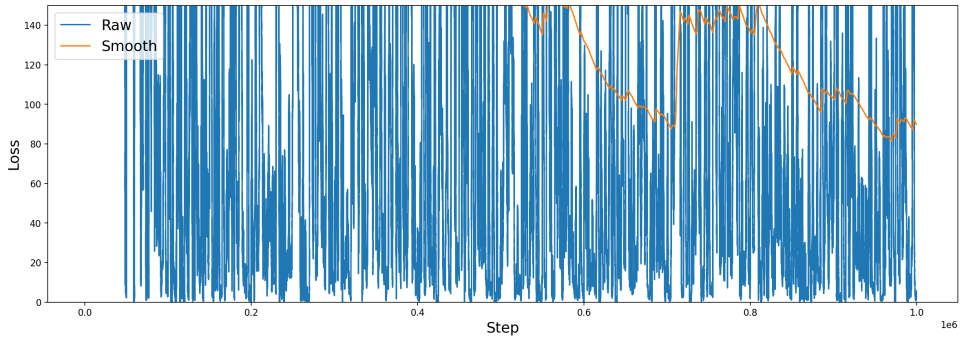
Table F.2: Different learning rates tested.

F.3 Results

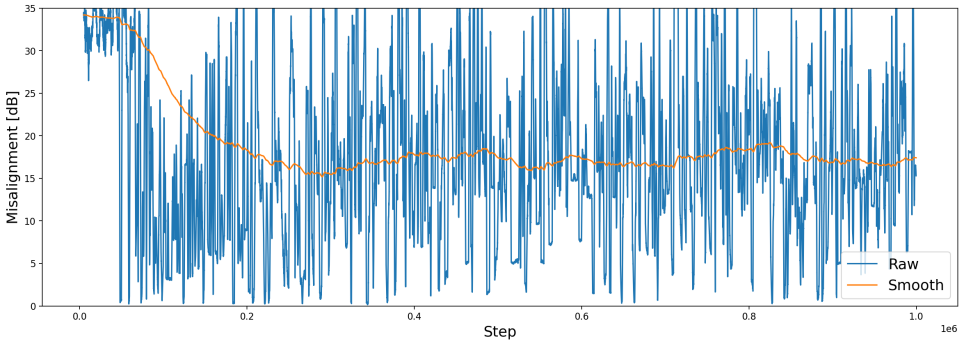
Test No.	Avg. Loss	Avg. Misalignment [dB]
1	255.784	17.338
2	22.929	9.466
3	23.039	7.872
4	23.170	7.620
5	38.813	8.836
6	144.604	11.711
7	257.721	14.378

Table F.3: Results for testing different learning rates.

F.3.1 Test No. 1



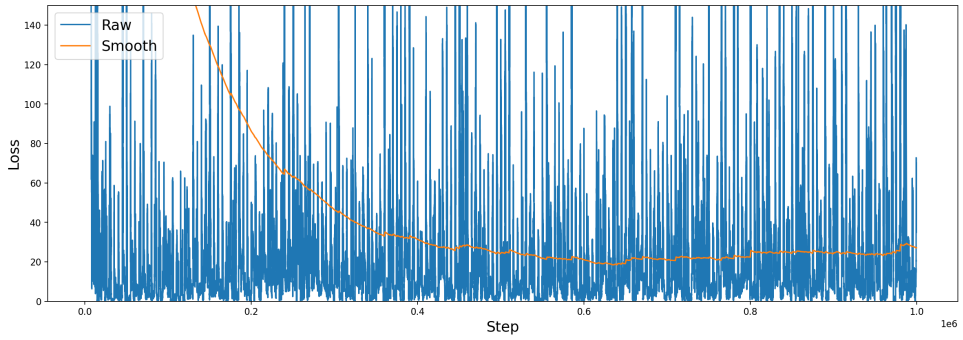
(a) Loss values.



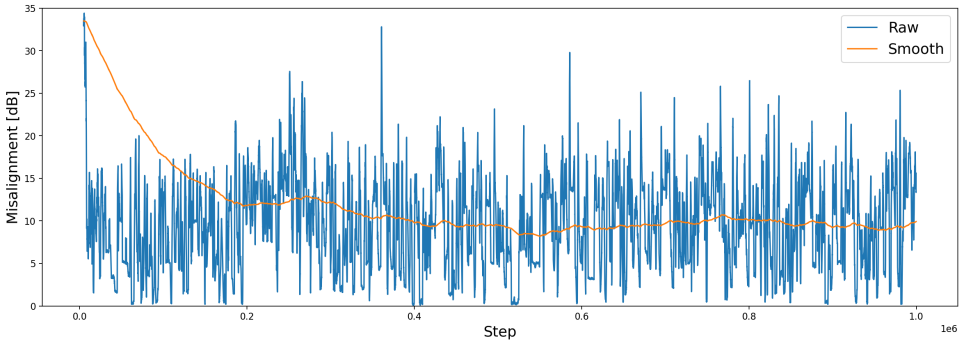
(b) Average misalignment for last 1000 steps.

Figure F.1: Results for test 1. Test parameter: Learning Rate = 0.0001.

F.3.2 Test No. 2



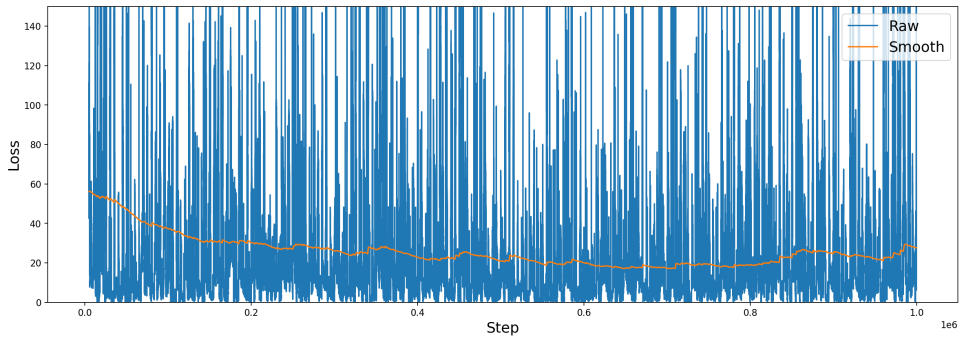
(a) Loss values.



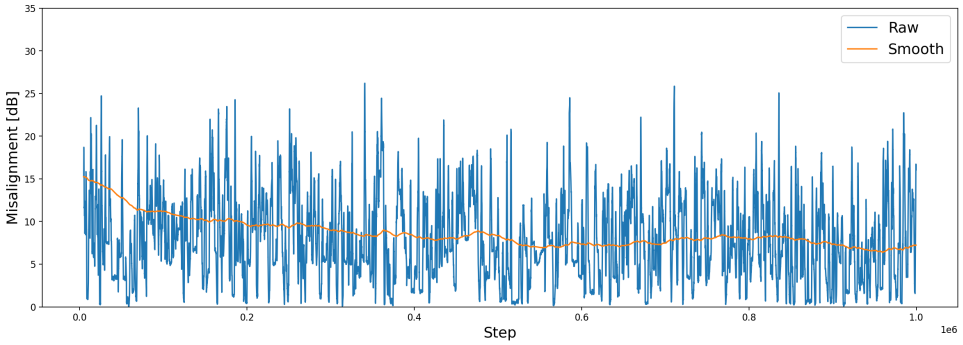
(b) Average misalignment for last 1000 steps.

Figure F.2: Results for test 2. Test parameter: Learning Rate = 0.001.

F.3.3 Test No. 3



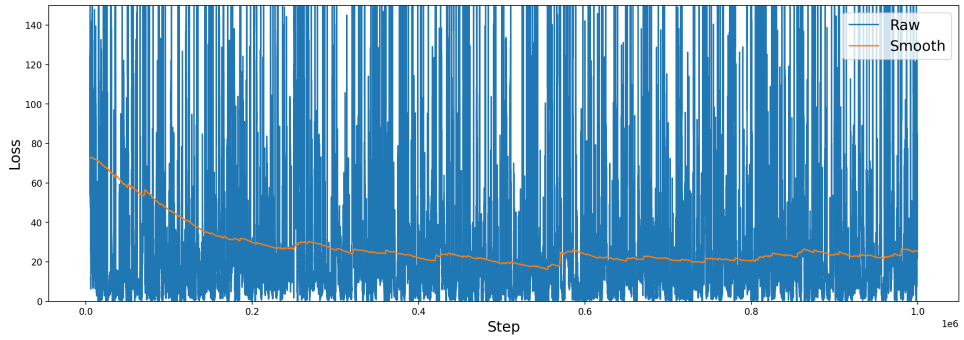
(a) Loss values.



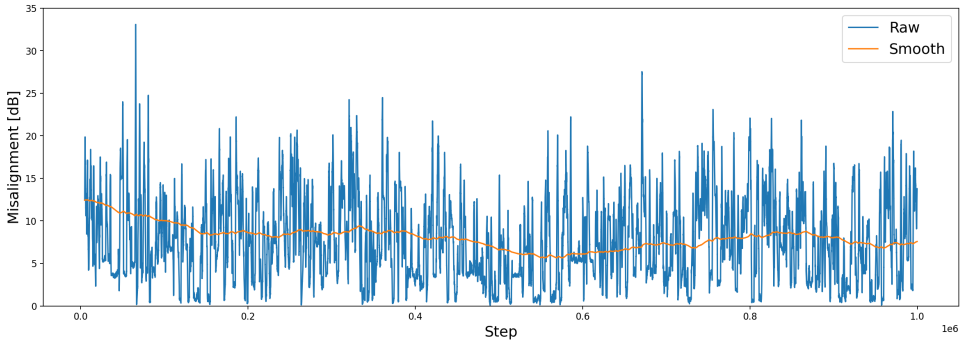
(b) Average misalignment for last 1000 steps.

Figure F.3: Results for test 3. Test parameter: Learning Rate = 0.01.

F.3.4 Test No. 4



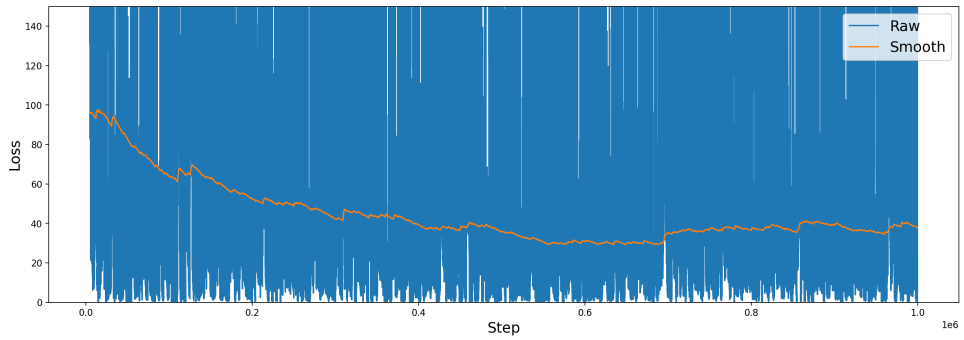
(a) Loss values.



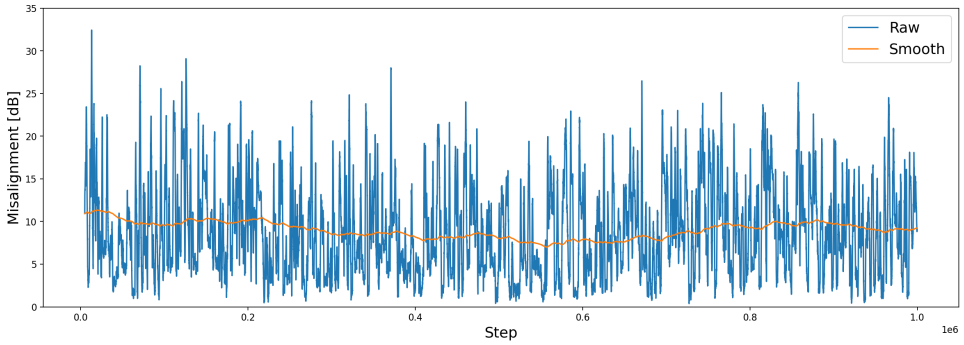
(b) Average misalignment for last 1000 steps.

Figure F.4: Results for test 4. Test parameter: Learning Rate = 0.1.

F.3.5 Test No. 5



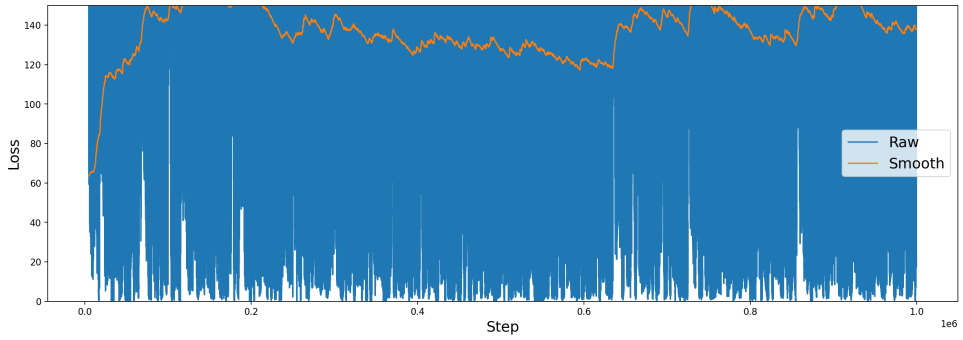
(a) Loss values.



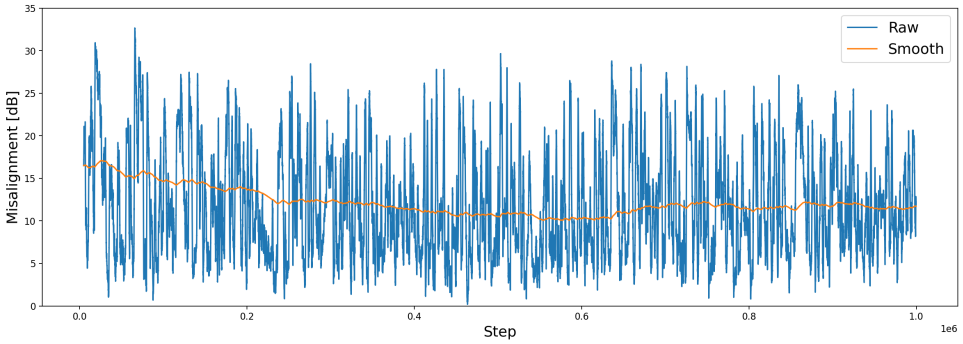
(b) Average misalignment for last 1000 steps.

Figure F.5: Results for test 5. Test parameter: Learning Rate = 0.3.

F.3.6 Test No. 6



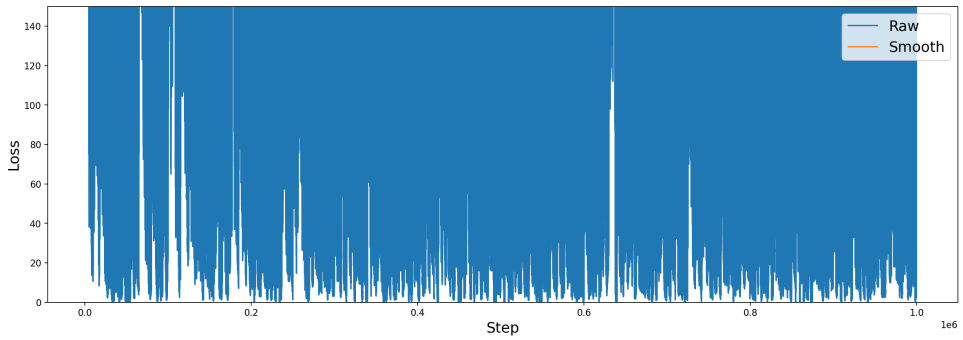
(a) Loss values.



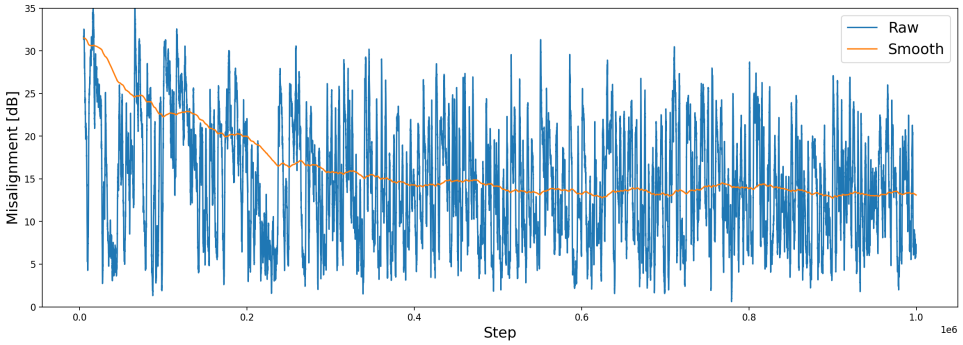
(b) Average misalignment for last 1000 steps.

Figure F.6: Results for test 6. Test parameter: Learning Rate = 0.7.

F.3.7 Test No. 7



(a) Loss values.



(b) Average misalignment for last 1000 steps.

Figure F.7: Results for test 7. Test parameter: Learning Rate = 0.9.

Test: DQN G

This appendix will document the test of the tuned NN parameters on the four different scenarios. The tests will be done on the created data sets seen on fig. 5.3 described in section 5.2 with the parameters seen in table 2.1.

G.1 Static Parameters

The static parameters for the DQN can been seen in table G.1.

	Hyper parameter	Value Space
DQN	Hidden Layers	[50, 50]
	Learning Rate (ADAM)	0.01
	Memory Size	1000
	Batch Size	500
	Target Update	10
RL	Forgetting Factor (γ)	0.7
	Exploring Factor (ϵ)	0.01
	State Space	$[a_{-1}, a_{-2}, a_{-3}, \theta_0, \theta_{-1}, \theta_{-2}, x_0, x_{-1}, x_{-2}, y_0, y_{-1}, y_{-2}]$

Table G.1: Hyper parameter for the RL agent based on DQN.

G.2 Test Parameters

This test will use four different scenarios as seen in table G.2.

Test No.	Scenario
1	Car LOS
2	Car NLOS
3	Pedestrian LOS
4	Pedestrian NLOS

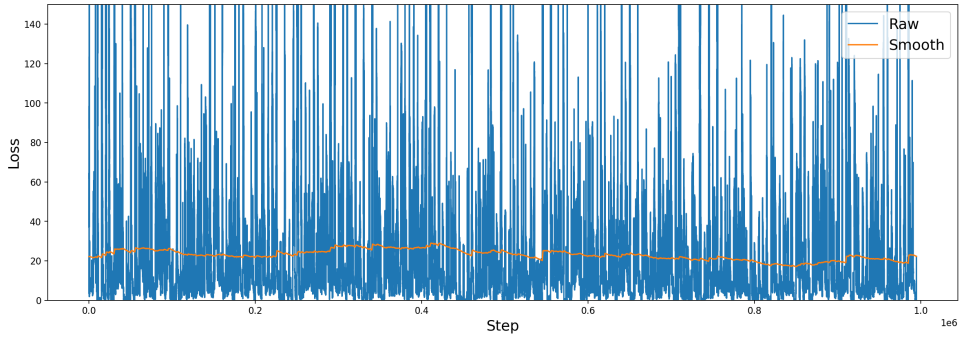
Table G.2: Scenarios which will be tested.

G.3 Results

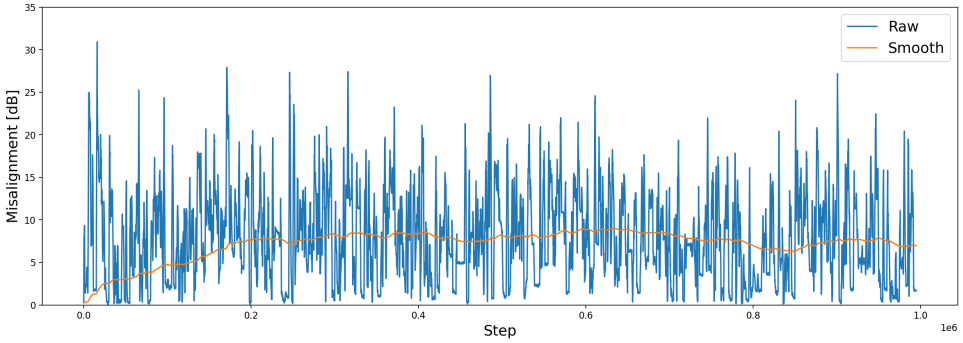
Test No.	Avg. Loss	Avg. Misalignment [dB]
1	23.077	7.851
2	62.052	9.968
3	25.109	8.522
4	49.069	11.165

Table G.3: Results for testing different scenarios.

G.3.1 Test No. 1



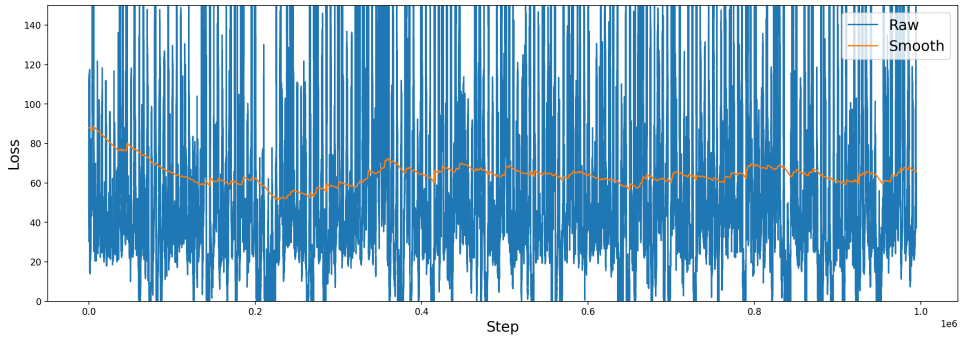
(a) Loss values.



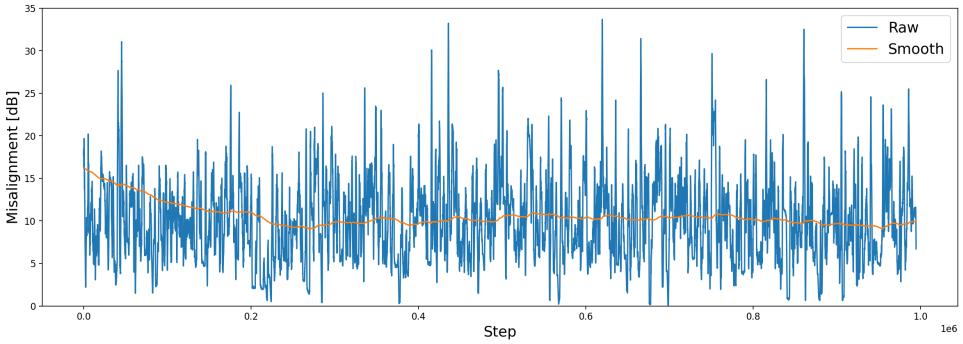
(b) Average misalignment for last 1000 steps.

Figure G.1: Results for test 1. Test parameter: Scenario = Car LOS.

G.3.2 Test No. 2



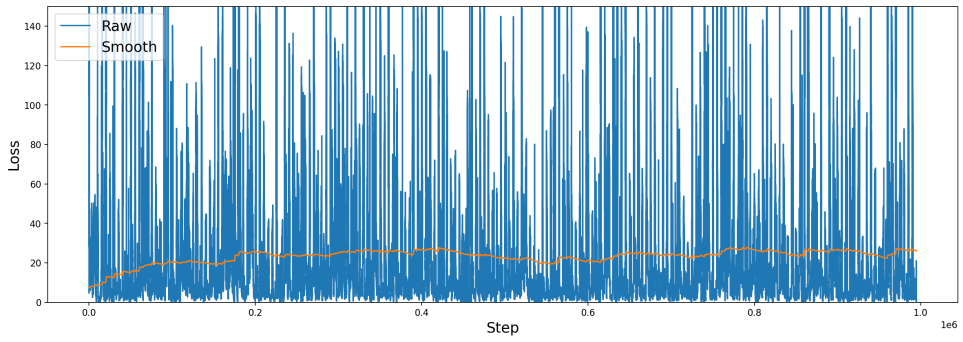
(a) Loss values.



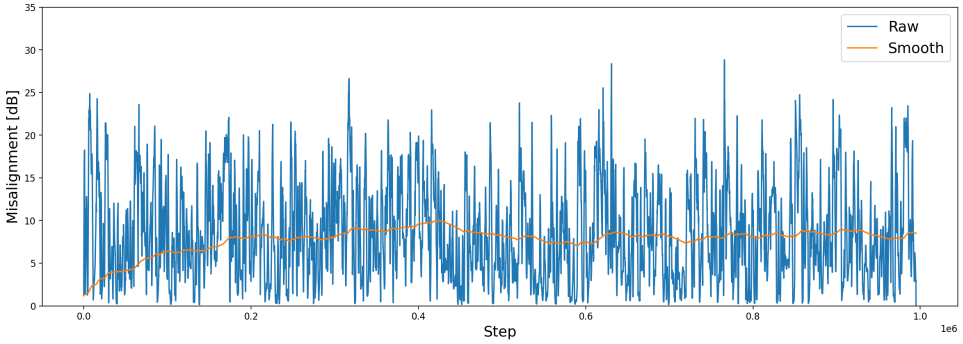
(b) Average misalignment for last 1000 steps.

Figure G.2: Results for test 2. Test parameter: Scenario = Car NLOS.

G.3.3 Test No. 3



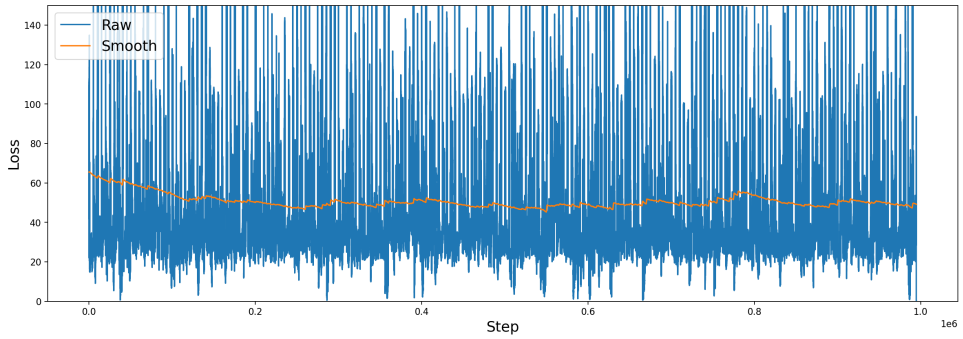
(a) Loss values.



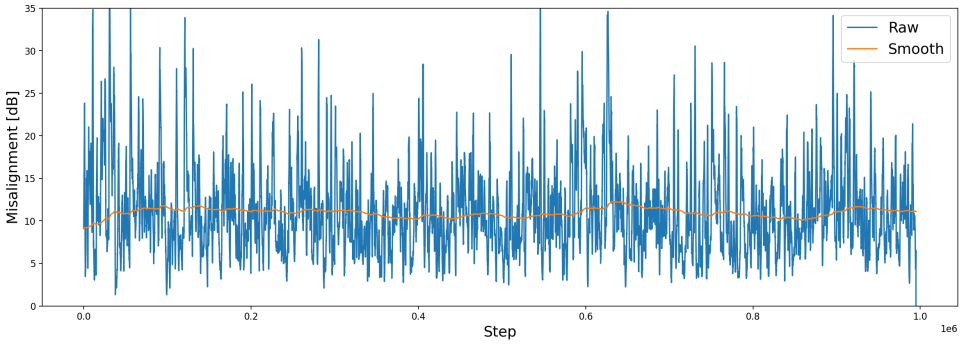
(b) Average misalignment for last 1000 steps.

Figure G.3: Results for test 3. Test parameter: Scenario = Pedestrian LOS.

G.3.4 Test No. 4



(a) Loss values.



(b) Average misalignment for last 1000 steps.

Figure G.4: Results for test 4. Test parameter: Scenario = Pedestrian NLOS.

Test: Forgetting factor H

This appendix will document the test of different forgetting factors. The test will be done on the created data set seen on fig. 5.1 described in section 5.2 with the parameters seen in table 2.1.

H.1 Static Parameters

The static parameters for this test can been seen in table H.1.

	Hyper parameter	Value Space
DQN	Hidden Layers	[50, 50]
	Learning Rate (ADAM)	0.01
	Memory Size	1000
	Batch Size	500
	Target Update	10
RL	Forgetting Factor (γ)	To be tuned
	Exploring Factor (ϵ)	0.01
	State Space	$[a_{-1}, a_{-2}, a_{-3}, \theta_0, \theta_{-1}, \theta_{-2}, x_0, x_{-1}, x_{-2}, y_0, y_{-1}, y_{-2}]$

Table H.1: Hyper parameter for the RL agent based on DQN.

H.2 Test Parameters

This test will use seven different forgetting factors as seen in table H.2.

Test No.	Forgetting factor
1	0.4
2	0.5
3	0.6
4	0.7
5	0.8
6	0.9
7	0.99

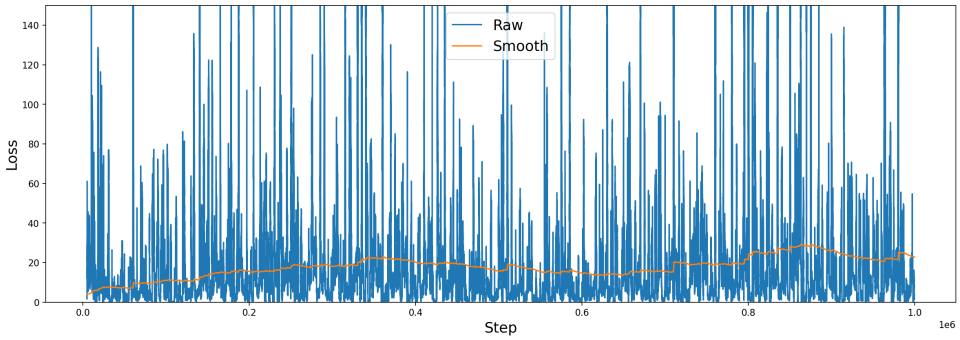
Table H.2: Forgetting factors that will be tested.

H.3 Results

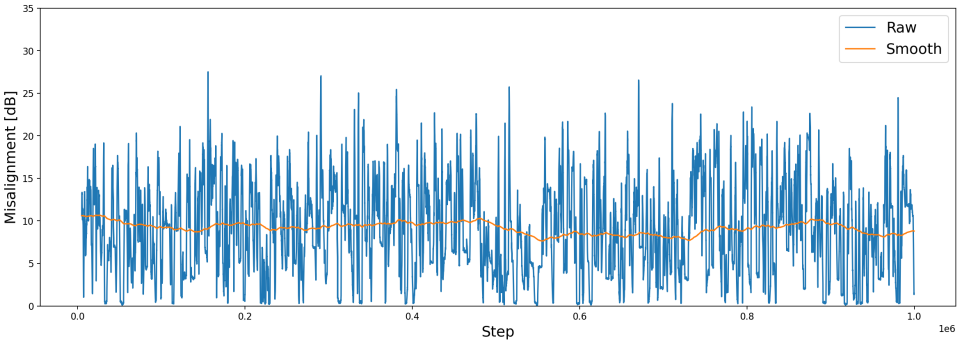
Test No.	Avg. Loss	Avg. Misalignment [dB]
1	19.706	8.975
2	21.820	8.697
3	18.850	8.660
4	28.215	8.386
5	29.203	7.193
6	55.991	6.645
7	3937.014	14.329

Table H.3: Results for testing different forgetting factors.

H.3.1 Test No. 1



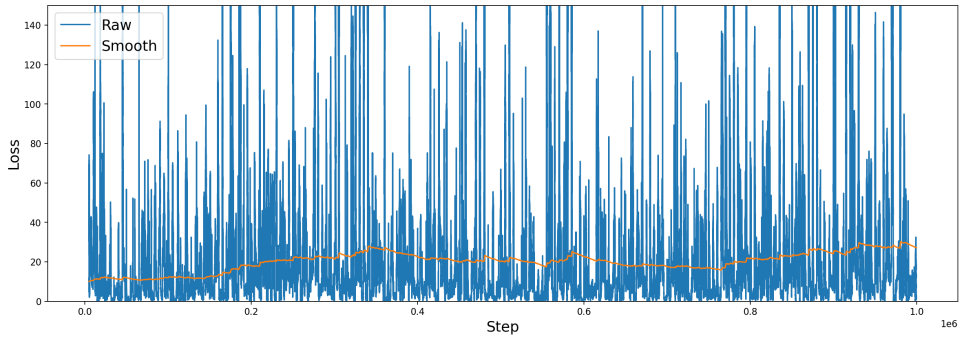
(a) Loss values.



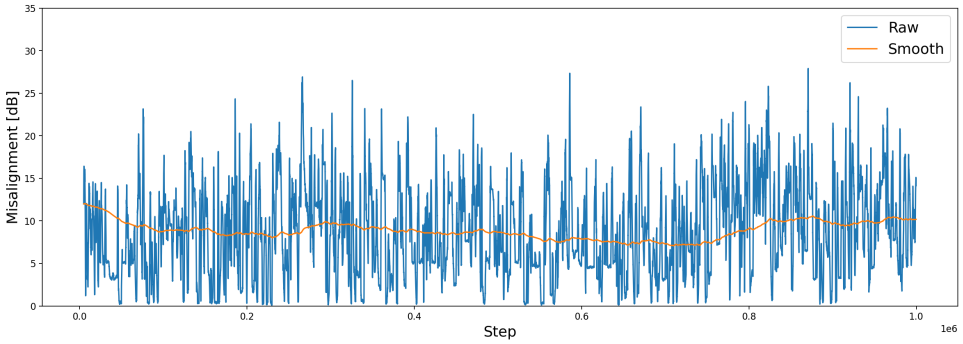
(b) Average misalignment for last 1000 steps.

Figure H.1: Results for test 1. Test parameter: Forgetting Factor = 0.4.

H.3.2 Test No. 2



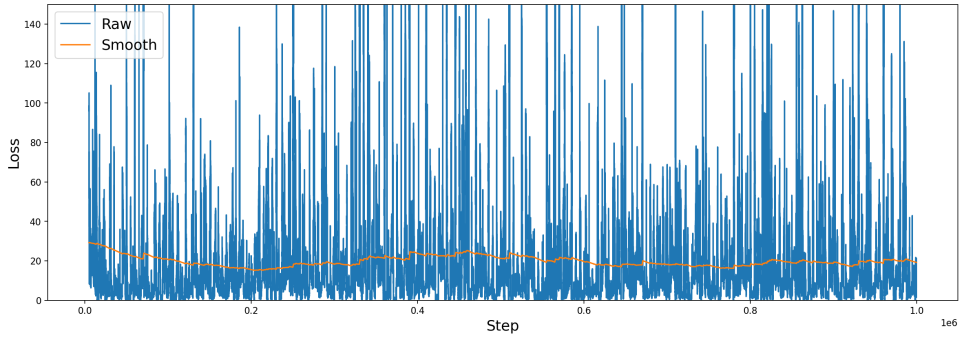
(a) Loss values.



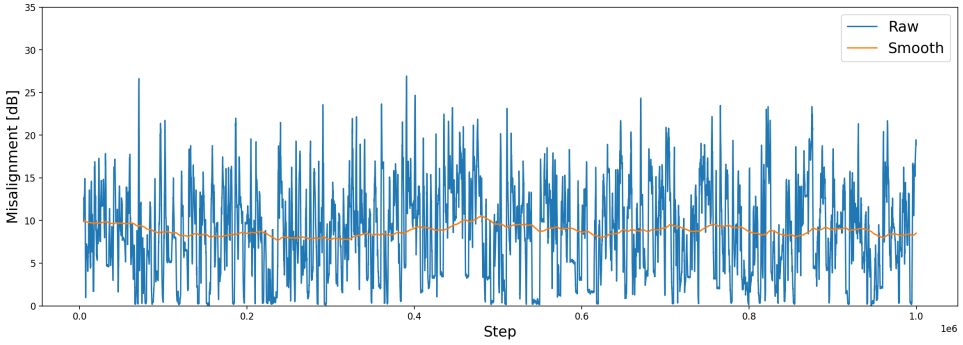
(b) Average misalignment for last 1000 steps.

Figure H.2: Results for test 2. Test parameter: Forgetting Factor = 0.5.

H.3.3 Test No. 3



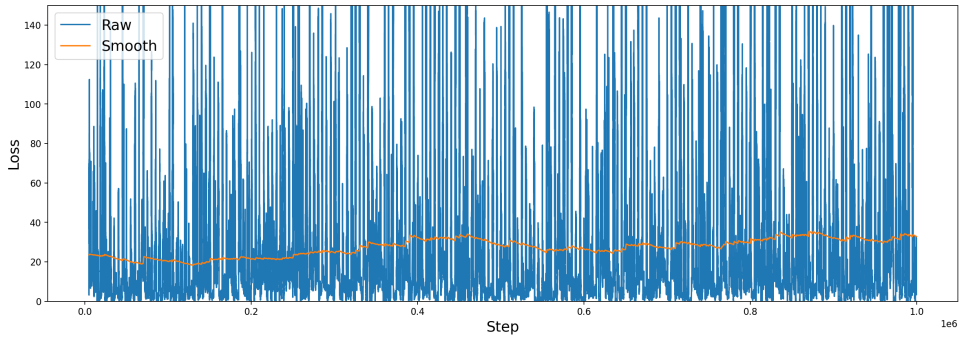
(a) Loss values.



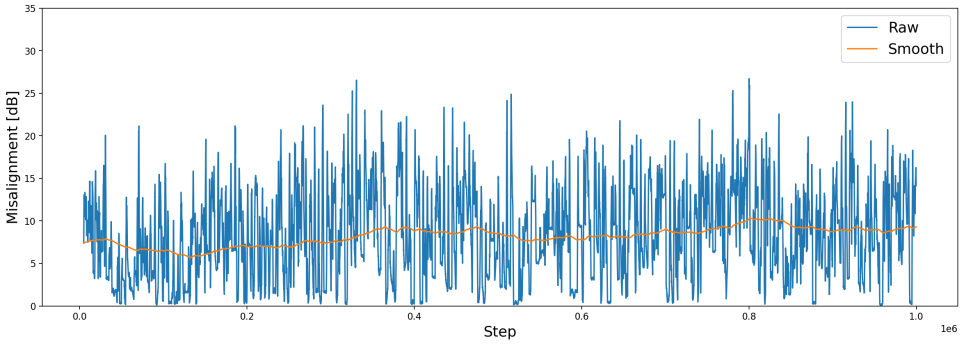
(b) Average misalignment for last 1000 steps.

Figure H.3: Results for test 3. Test parameter: Forgetting Factor = 0.6.

H.3.4 Test No. 4



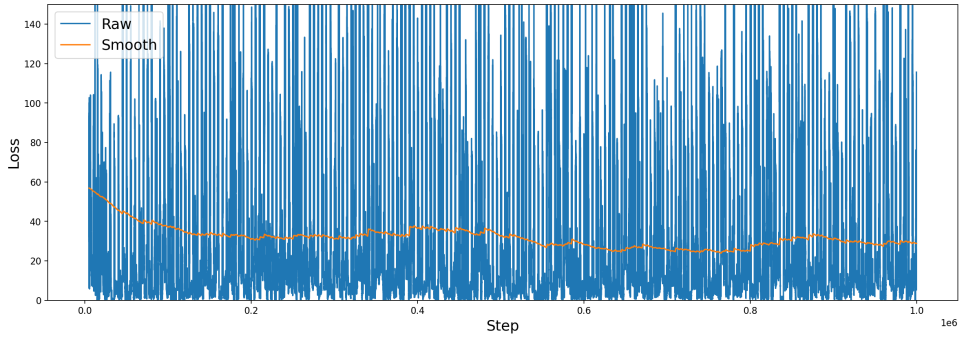
(a) Loss values.



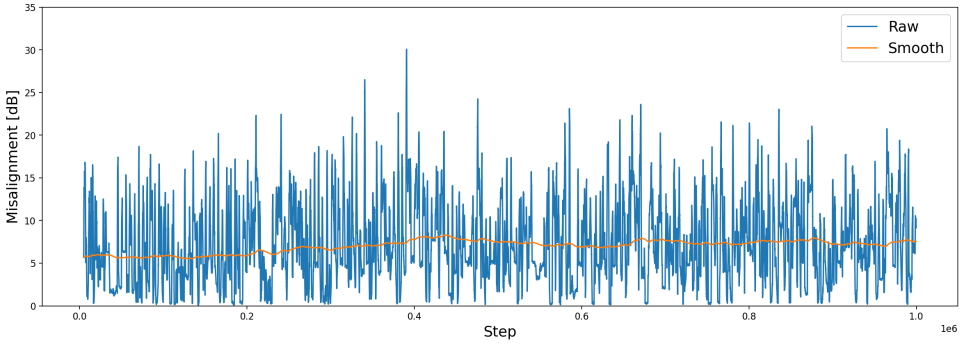
(b) Average misalignment for last 1000 steps.

Figure H.4: Results for test 4. Test parameter: Forgetting Factor = 0.7.

H.3.5 Test No. 5



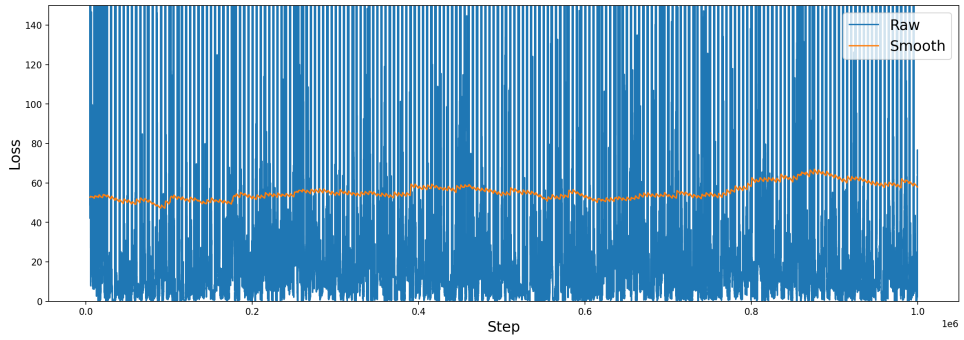
(a) Loss values.



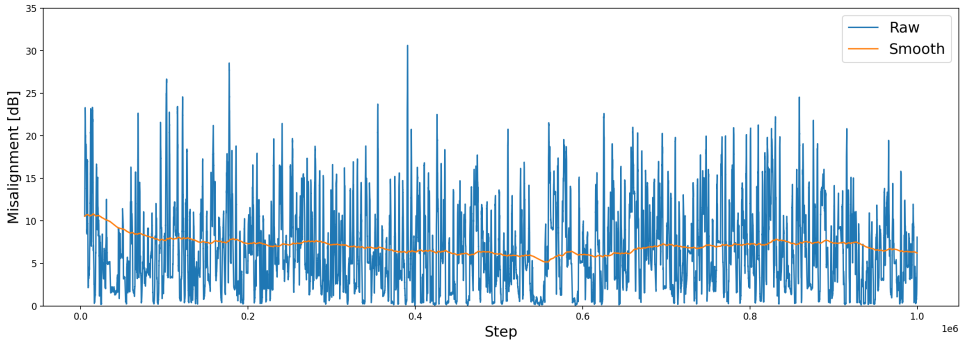
(b) Average misalignment for last 1000 steps.

Figure H.5: Results for test 5. Test parameter: Forgetting Factor = 0.8.

H.3.6 Test No. 6



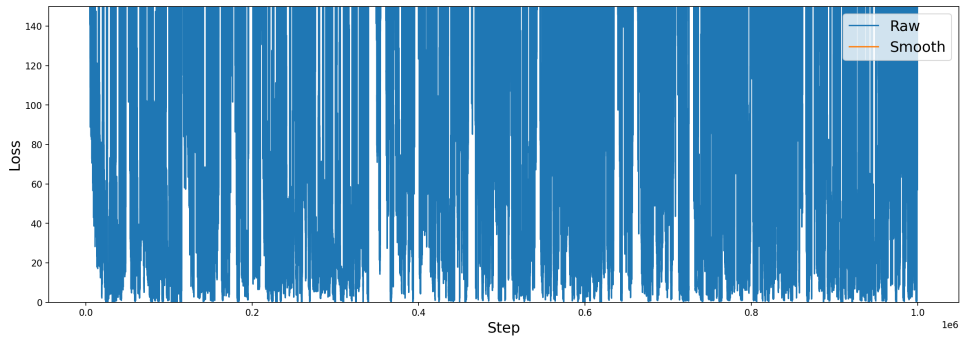
(a) Loss values.



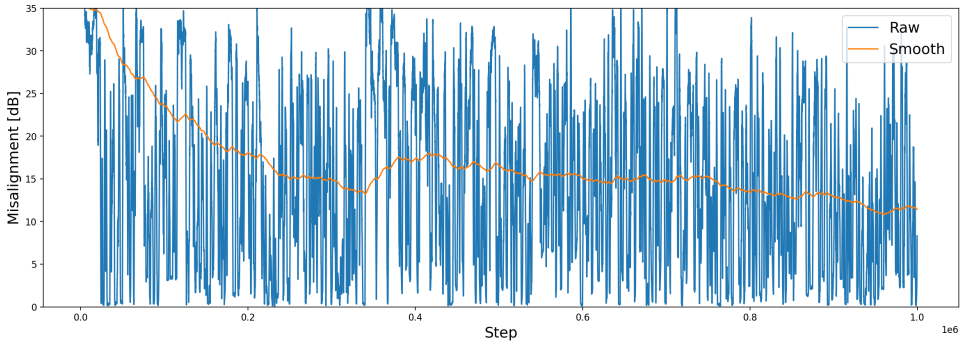
(b) Average misalignment for last 1000 steps.

Figure H.6: Results for test 6. Test parameter: Forgetting Factor = 0.9.

H.3.7 Test No. 7



(a) Loss values.



(b) Average misalignment for last 1000 steps.

Figure H.7: Results for test 7. Test parameter: Forgetting Factor = 0.99.

Test: Exploring Factor I

This appendix will document the test of different Exploring factors. The test will be done on the created data set seen on fig. 5.1 described in section 5.2 with the parameters seen in table 2.1.

I.1 Static Parameters

The static parameters for this test can been seen in table H.1.

	Hyper parameter	Value Space
DQN	Hidden Layers	[50, 50]
	Learning Rate (ADAM)	0.01
	Memory Size	1000
	Batch Size	500
	Target Update	10
RL	Forgetting Factor (γ)	0.8
	Exploring Factor (ϵ)	To be tuned
	State Space	$[a_{-1}, a_{-2}, a_{-3}, \theta_0, \theta_{-1}, \theta_{-2}, x_0, x_{-1}, x_{-2}, y_0, y_{-1}, y_{-2}]$

Table I.1: Hyper parameter for the RL agent based on DQN.

I.2 Test Parameters

This test will use seven different exploring factors as seen in table I.2.

Test No.	Exploration Factor
1	0.0005
2	0.001
3	0.005
4	0.05
5	0.1
6	0.3
7	0.5

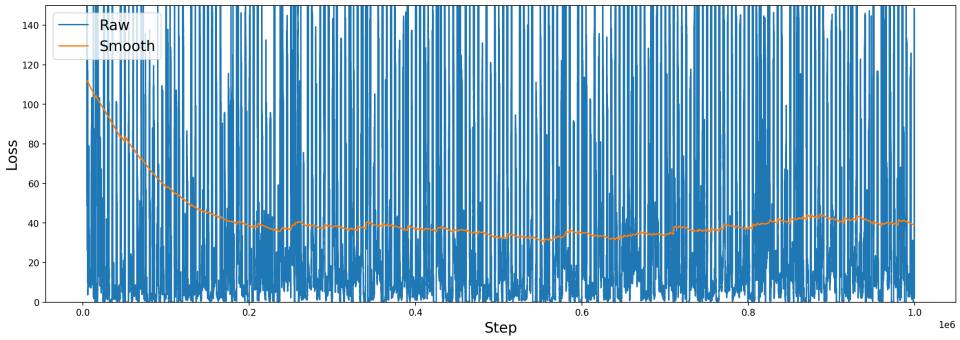
Table I.2: Exploring factors that will be tested.

I.3 Results

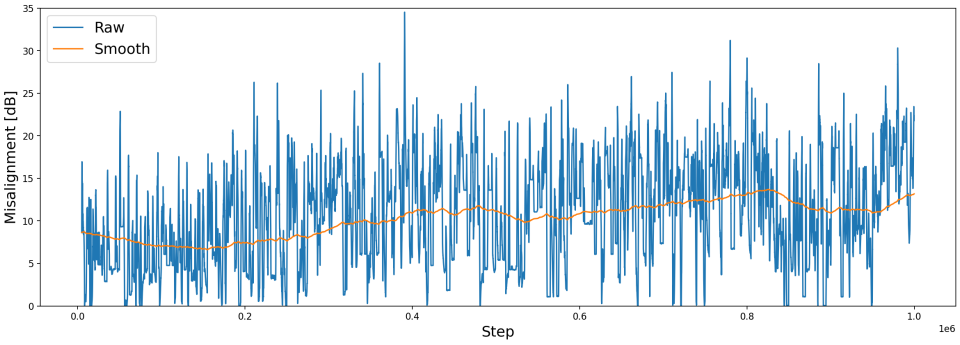
Test No.	Avg. Loss	Avg. Misalignment [dB]
1	35.172	10.704
2	29.056	9.560
3	29.987	7.907
4	25.174	6.467
5	27.661	7.806
6	24.264	13.081
7	25.296	19.079

Table I.3: Results for testing different exploring factors.

I.3.1 Test No. 1



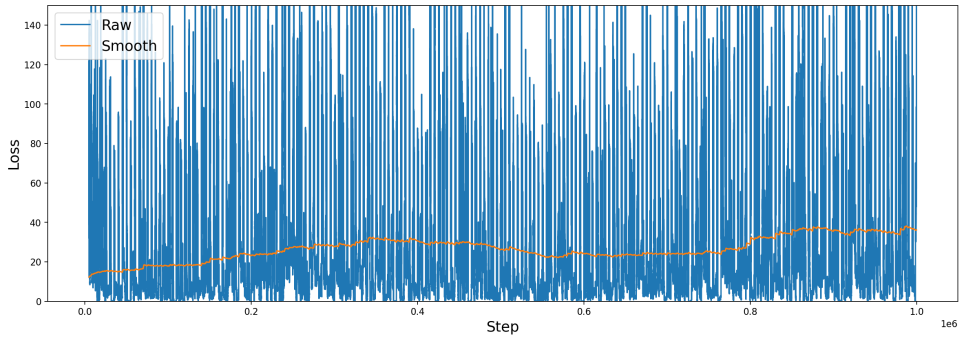
(a) Loss values.



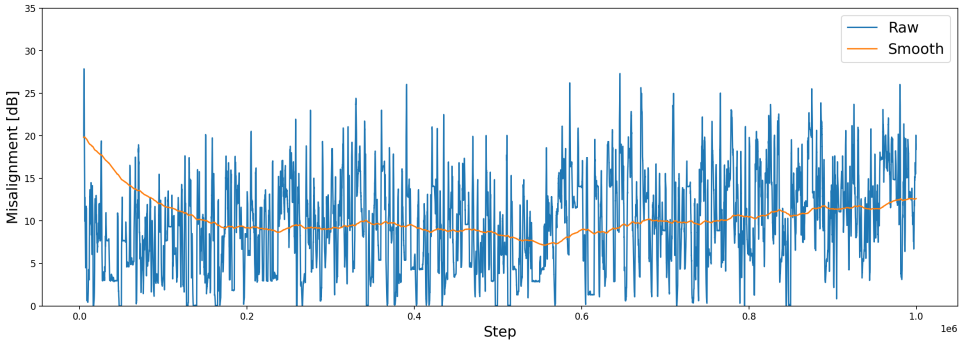
(b) Average misalignment for last 1000 steps.

Figure I.1: Results for test 1. Test parameter: Exploration Factor = 0.0005.

I.3.2 Test No. 2



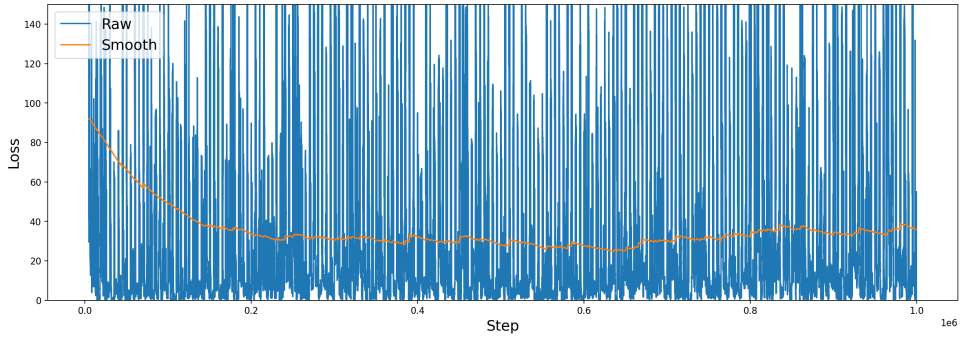
(a) Loss values.



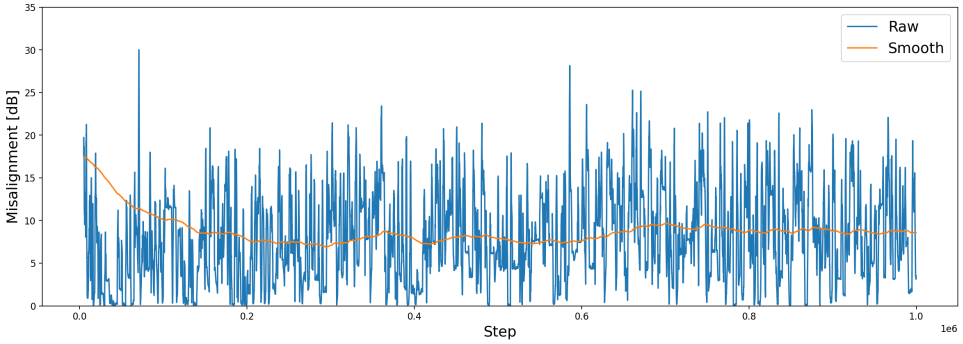
(b) Average misalignment for last 1000 steps.

Figure I.2: Results for test 2. Test parameter: Exploration Factor = 0.001.

I.3.3 Test No. 3



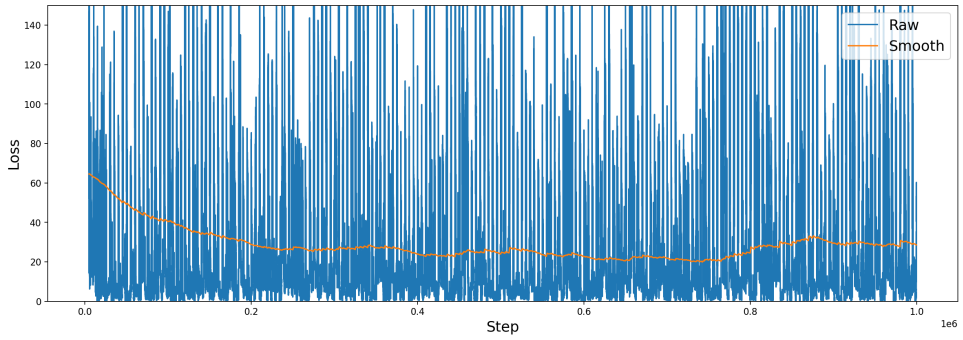
(a) Loss values.



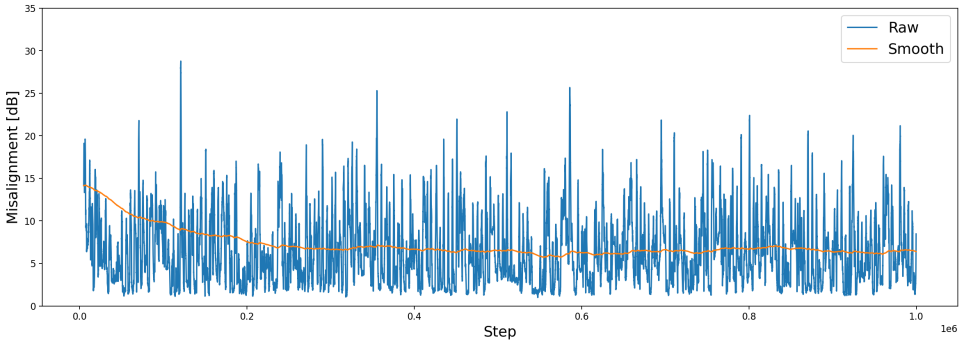
(b) Average misalignment for last 1000 steps.

Figure I.3: Results for test 3. Test parameter: Exploration Factor = 0.005.

I.3.4 Test No. 4



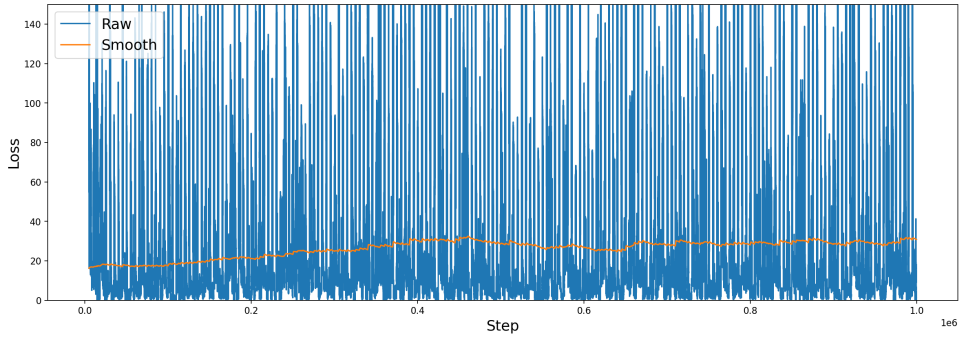
(a) Loss values.



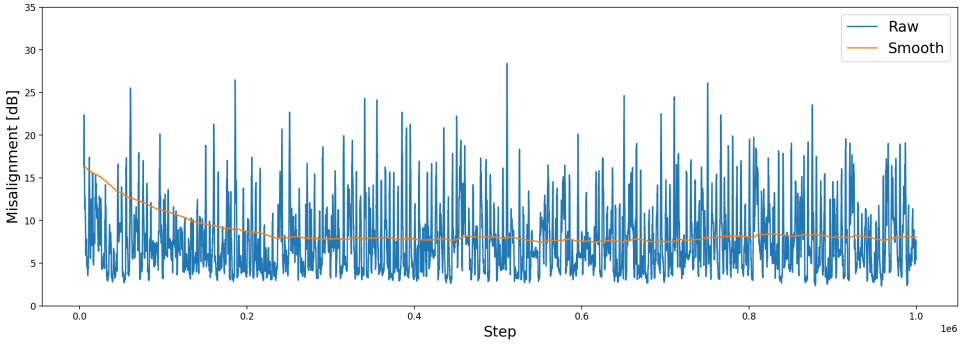
(b) Average misalignment for last 1000 steps.

Figure I.4: Results for test 4. Test parameter: Exploration Factor = 0.05.

I.3.5 Test No. 5



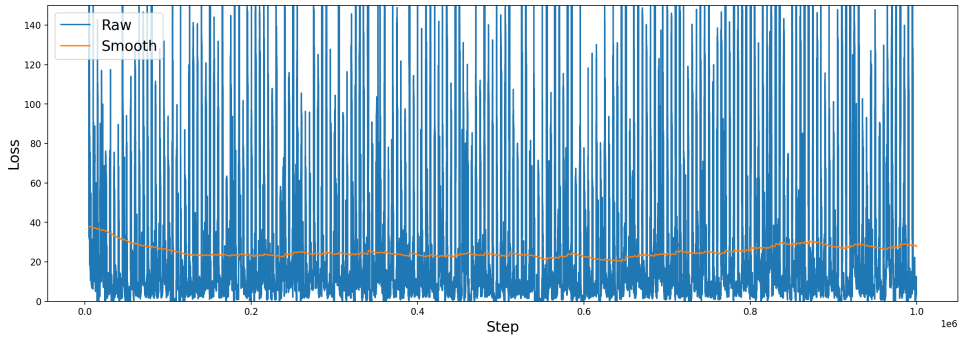
(a) Loss values.



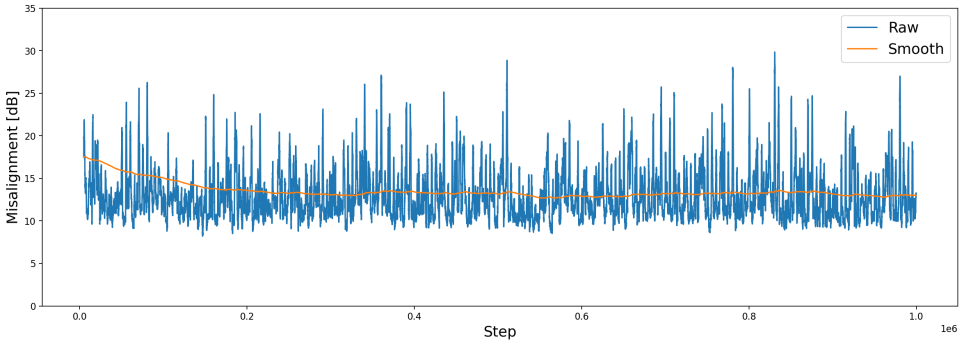
(b) Average misalignment for last 1000 steps.

Figure I.5: Results for test 5. Test parameter: Exploration Factor = 0.1.

I.3.6 Test No. 6



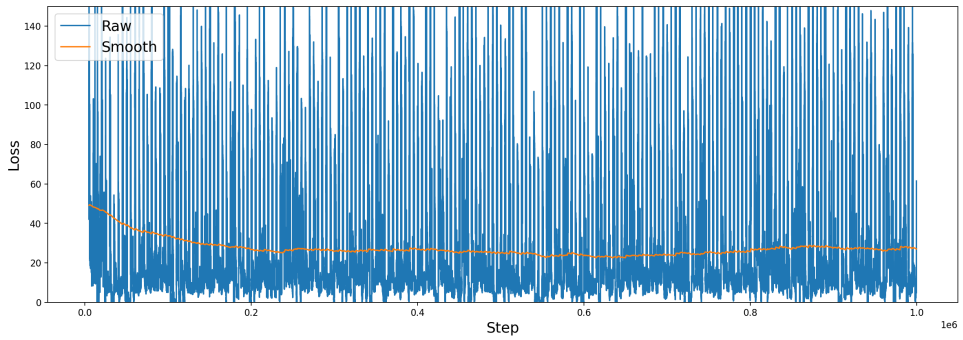
(a) Loss values.



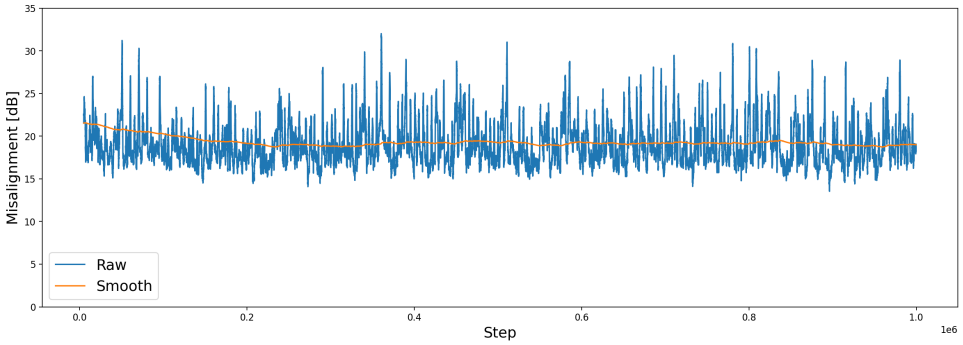
(b) Average misalignment for last 1000 steps.

Figure I.6: Results for test 6. Test parameter: Exploration Factor = 0.3.

I.3.7 Test No. 7



(a) Loss values.



(b) Average misalignment for last 1000 steps.

Figure I.7: Results for test 7. Test parameter: Exploration Factor = 0.5.

Test: State Space Part 1 J

This appendix will document the test of different state spaces. The test will be done on the created data set seen on fig. 5.1 described in section 5.2 with the parameters seen in table 2.1.

J.1 Static Parameters

The static parameters for this test can been seen in table J.1.

	Hyper parameter	Value Space
DQN	Hidden Layers	[50, 50]
	Learning Rate (ADAM)	0.01
	Memory Size	1000
	Batch Size	500
	Target Update	10
RL	Forgetting Factor (γ)	0.8
	Exploring Factor (ϵ)	0.01
	State Space	To be tuned

Table J.1: Hyper parameter for the RL agent based on DQN.

J.2 Test Parameters

This test will use ten different state spaces as seen in table J.2.

Test No.	State Space
1	[3, 0, 0]
2	[0, 3, 0]
3	[0, 0, 3]
4	[3, 3, 0]
5	[0, 3, 3]
6	[3, 0, 3]
7	[1, 1, 1]
8	[2, 2, 2]
9	[3, 3, 3]
10	[4, 4, 4]

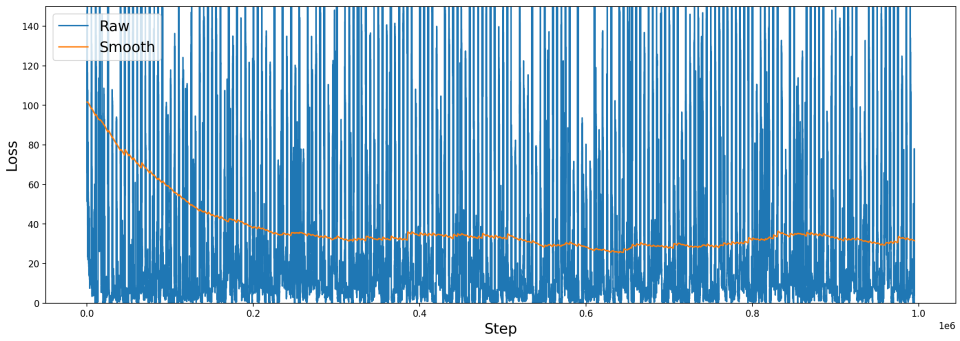
Table J.2: Different state space that will be tested. The values inside the bracket represent [No. actions, No. orientations, No. positions]

J.3 Results

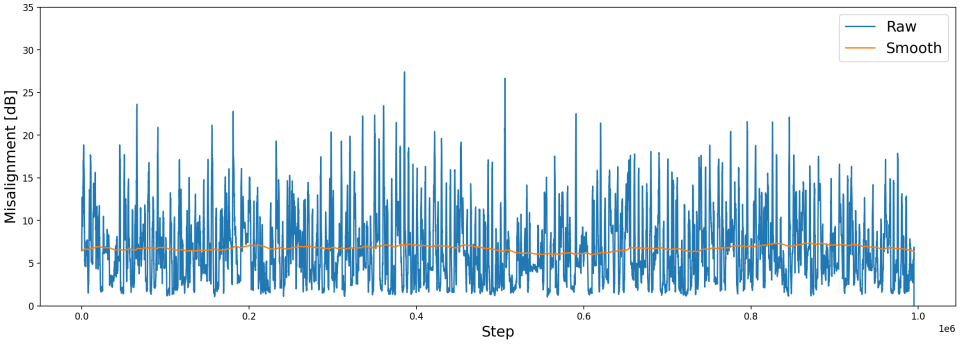
Test No.	Avg. Loss	Avg. Misalignment [dB]
1	31.000	6.804
2	31.437	6.944
3	21.497	6.442
4	31.048	6.882
5	22.486	6.827
6	24.776	6.633
7	25.364	6.752
8	29.138	6.893
9	28.699	6.795
10	28.333	6.661

Table J.3: Results for testing different state spaces.

J.3.1 Test No. 1



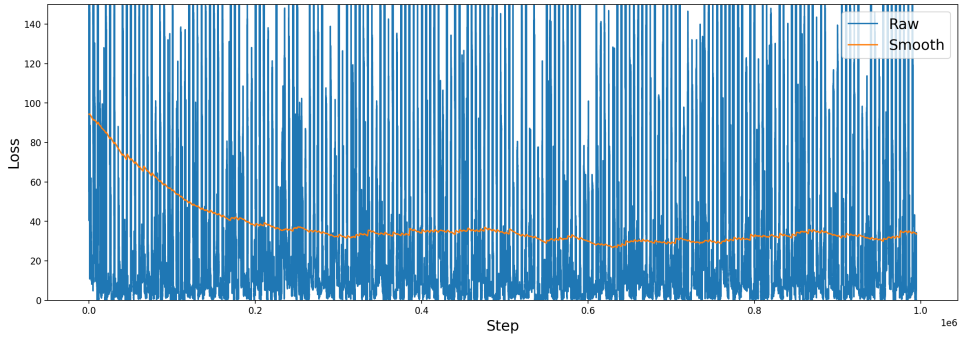
(a) Loss values.



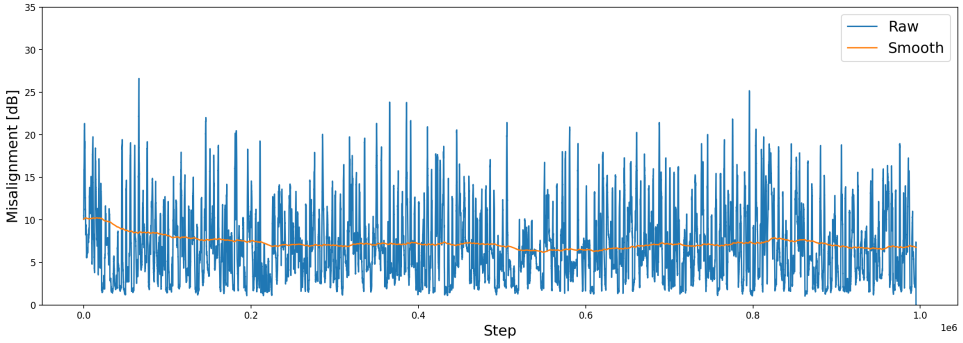
(b) Average misalignment for last 1000 steps.

Figure J.1: Results for test 1. Test parameter: State Space = [3, 0, 0].

J.3.2 Test No. 2



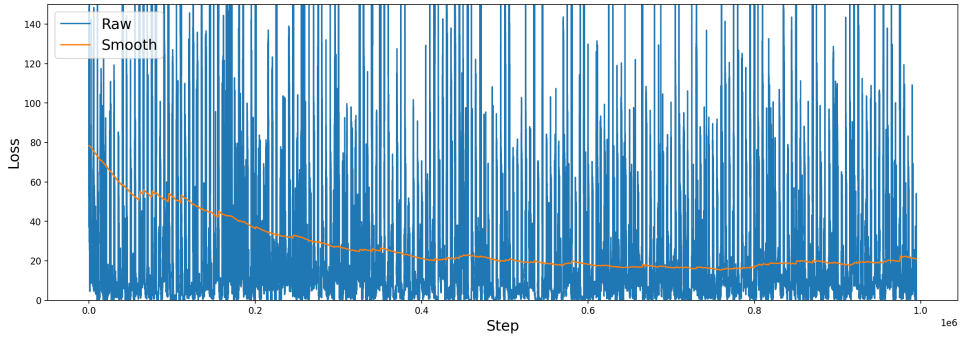
(a) Loss values.



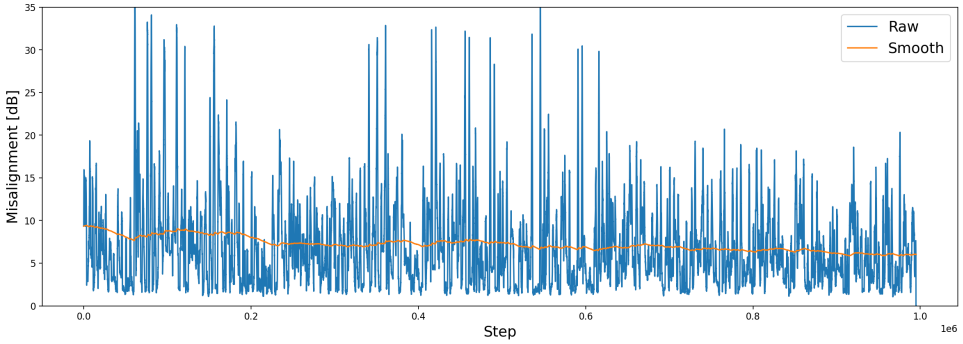
(b) Average misalignment for last 1000 steps.

Figure J.2: Results for test 2. Test parameter: State Space = [0, 3, 0]

J.3.3 Test No. 3



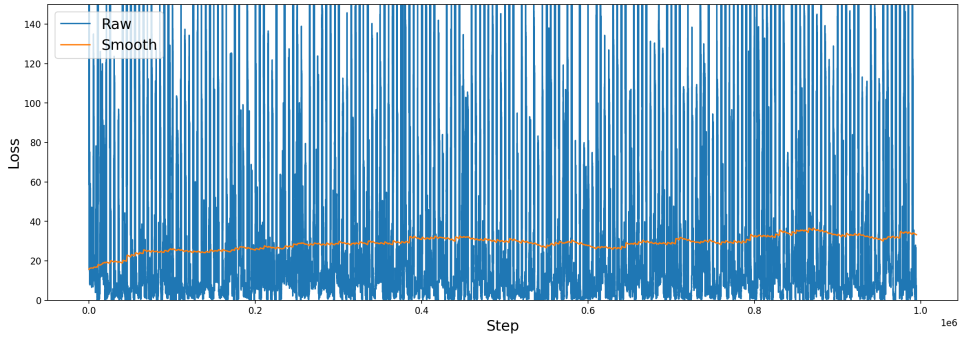
(a) Loss values.



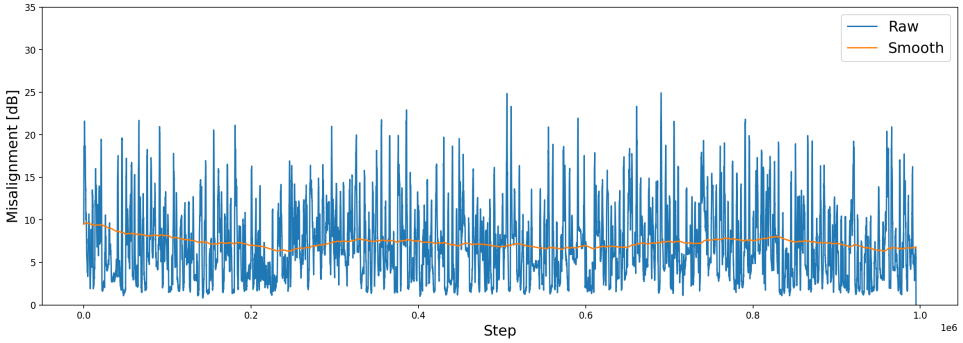
(b) Average misalignment for last 1000 steps.

Figure J.3: Results for test 3. Test parameter: State Space = [0, 0, 3]

J.3.4 Test No. 4



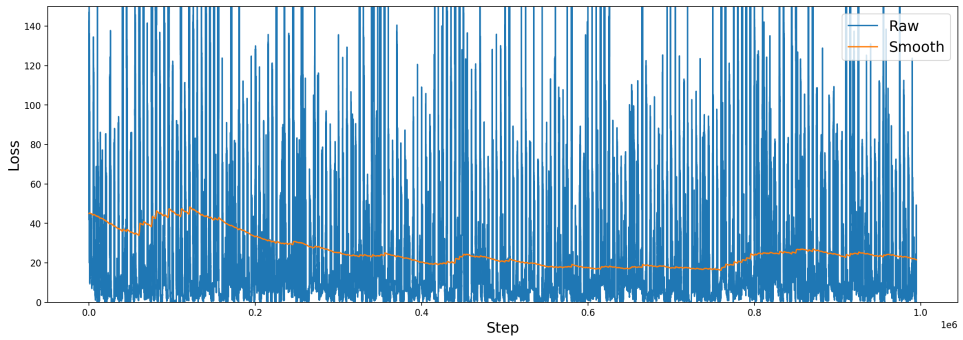
(a) Loss values.



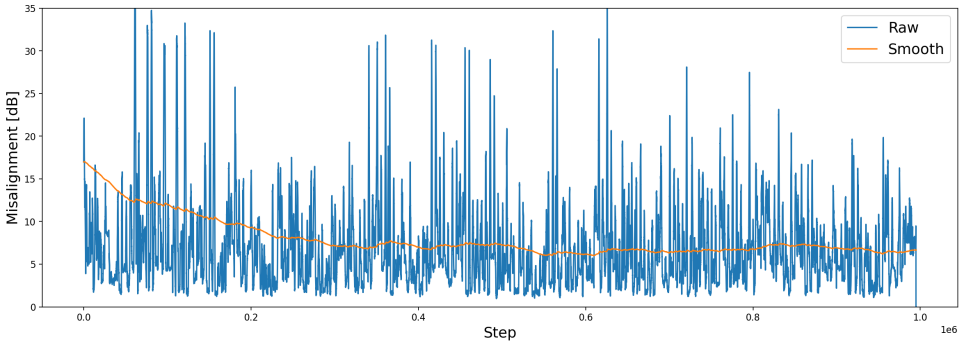
(b) Average misalignment for last 1000 steps.

Figure J.4: Results for test 4. Test parameter: State Space = [3, 3, 0]

J.3.5 Test No. 5



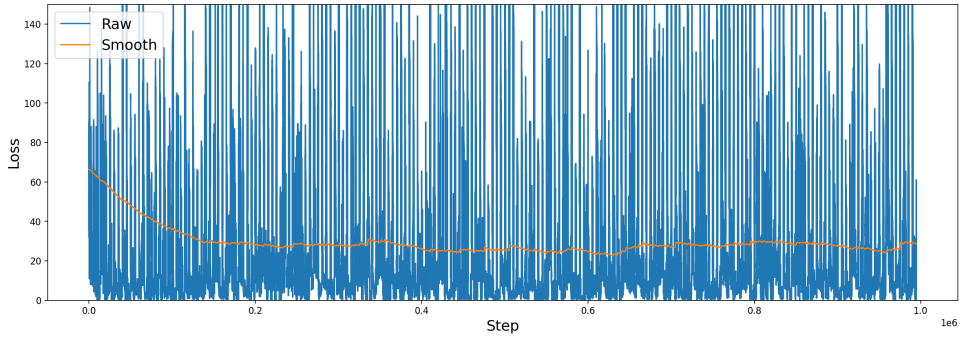
(a) Loss values.



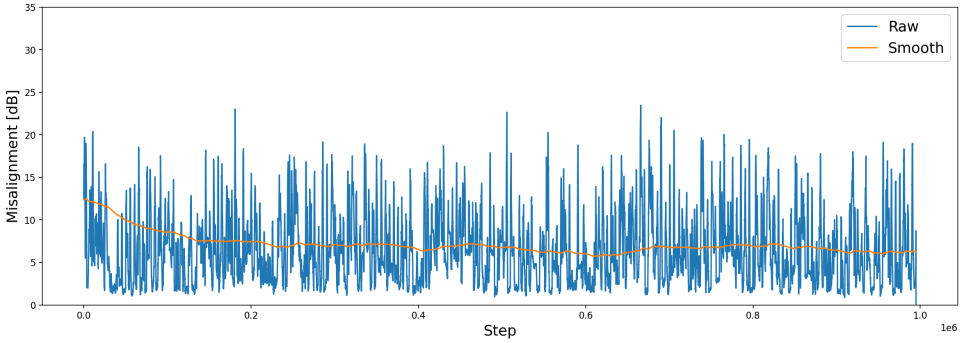
(b) Average misalignment for last 1000 steps.

Figure J.5: Results for test 5. Test parameter: State Space = [0, 3, 3]

J.3.6 Test No. 6



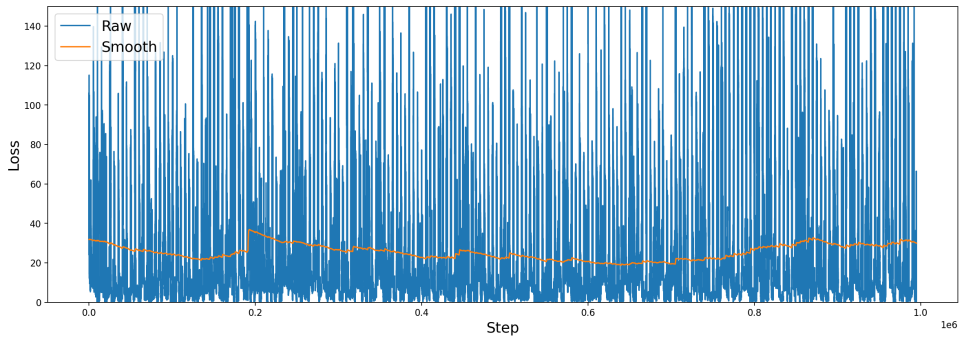
(a) Loss values.



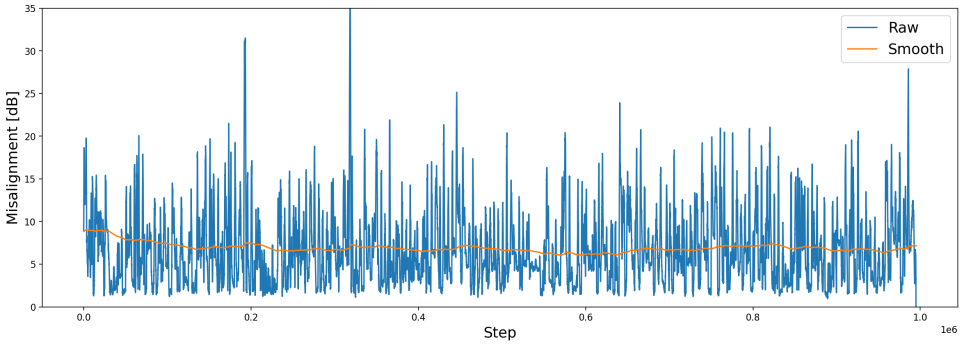
(b) Average misalignment for last 1000 steps.

Figure J.6: Results for test 6. Test parameter: State Space = [3, 0, 3]

J.3.7 Test No. 7



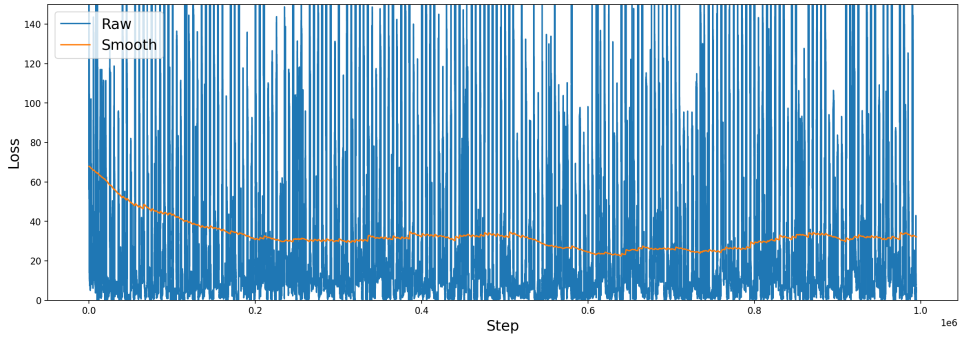
(a) Loss values.



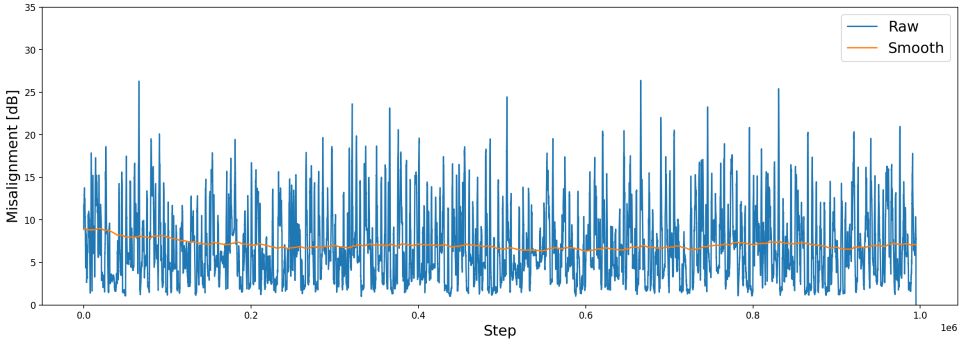
(b) Average misalignment for last 1000 steps.

Figure J.7: Results for test 7. Test parameter: State Space = [1, 1, 1]

J.3.8 Test No. 8



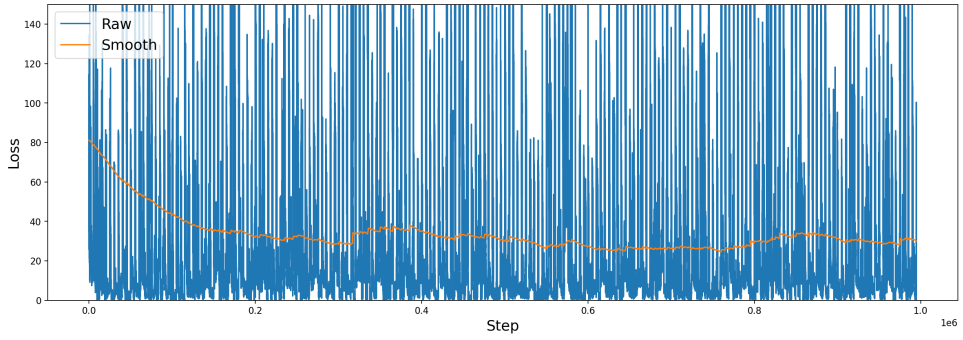
(a) Loss values.



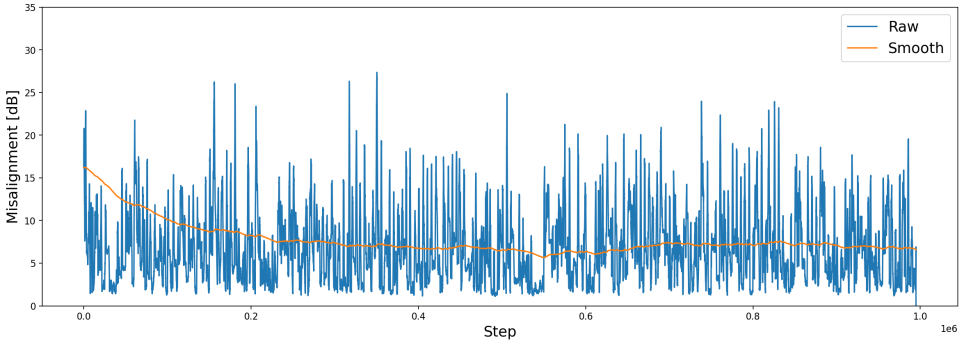
(b) Average misalignment for last 1000 steps.

Figure J.8: Results for test 8. Test parameter: State Space = [2, 2, 2]

J.3.9 Test No. 9



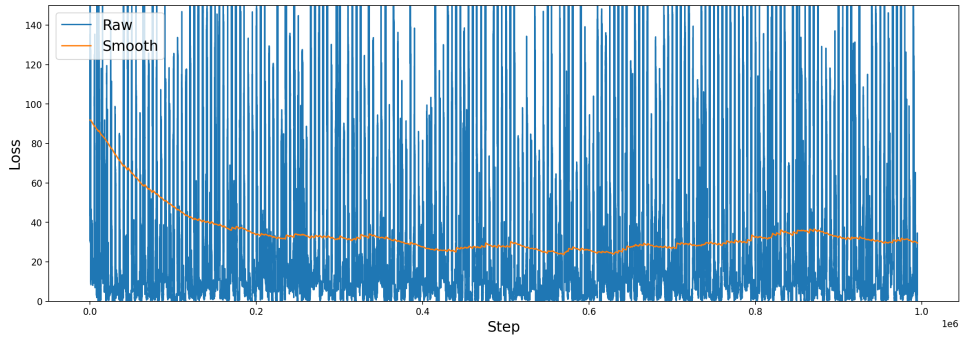
(a) Loss values.



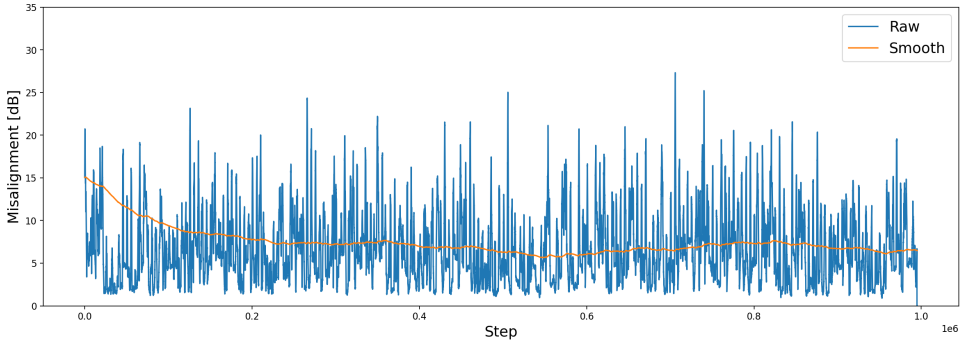
(b) Average misalignment for last 1000 steps.

Figure J.9: Results for test 9. Test parameter: State Space = [3, 3, 3]

J.3.10 Test No. 10



(a) Loss values.



(b) Average misalignment for last 1000 steps.

Figure J.10: Results for test 10. Test parameter: State Space = [4, 4, 4]

Test: State Space Part 2 K

This appendix will document the test of different state spaces. The test will be done on the created data set seen on fig. 5.1 described in section 5.2 with the parameters seen in table 2.1.

K.1 Static Parameters

The static parameters for this test can been seen in table K.1.

	Hyper parameter	Value Space
DQN	Hidden Layers	[50, 50]
	Learning Rate (ADAM)	0.01
	Memory Size	1000
	Batch Size	500
	Target Update	10
RL	Forgetting Factor (γ)	0.8
	Exploring Factor (ϵ)	0.01
	State Space	To be tuned

Table K.1: Hyper parameter for the RL agent based on DQN.

K.2 Test Parameters

This test will use six different state spaces as seen in table K.2.

Test No.	Learning Rate
1	[1, 0, 0]
2	[0, 1, 0]
3	[0, 0, 1]
4	[1, 1, 0]
5	[0, 1, 1]
6	[1, 0, 1]

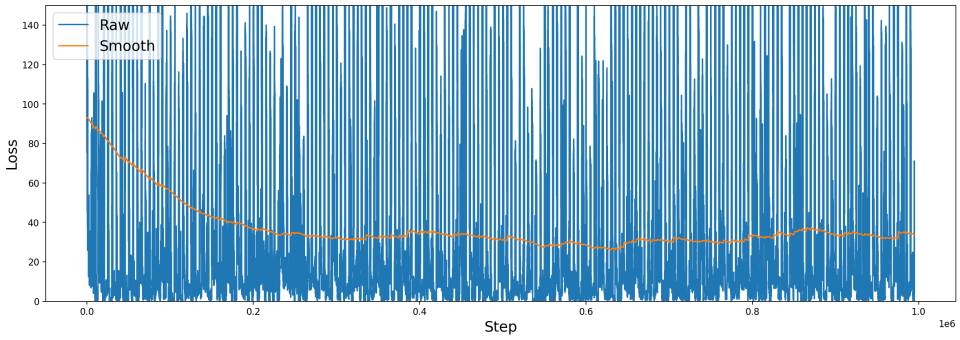
Table K.2: Different state space that will be tested. The values inside the bracket represent [No. actions, No. orientations, No. positions]

K.3 Results

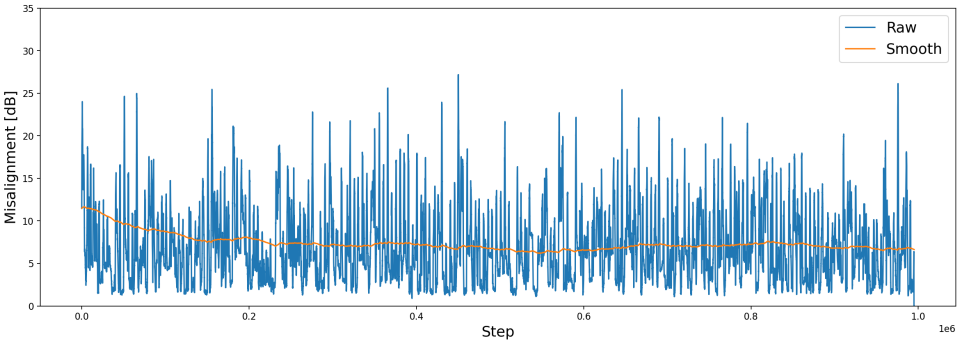
Test No.	Avg. Loss	Avg. Misalignment [dB]
1	31.449	6.915
2	29.557	6.534
3	21.760	6.670
4	31.142	6.747
5	22.163	6.762
6	23.452	6.585

Table K.3: Results for testing different state spaces.

K.3.1 Test No. 1



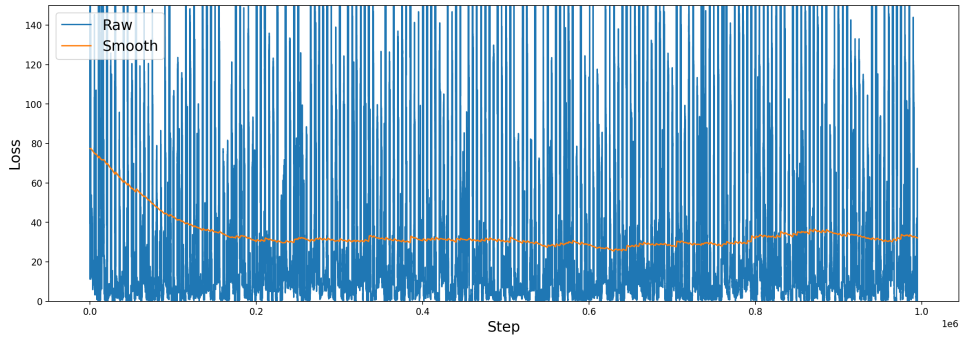
(a) Loss values.



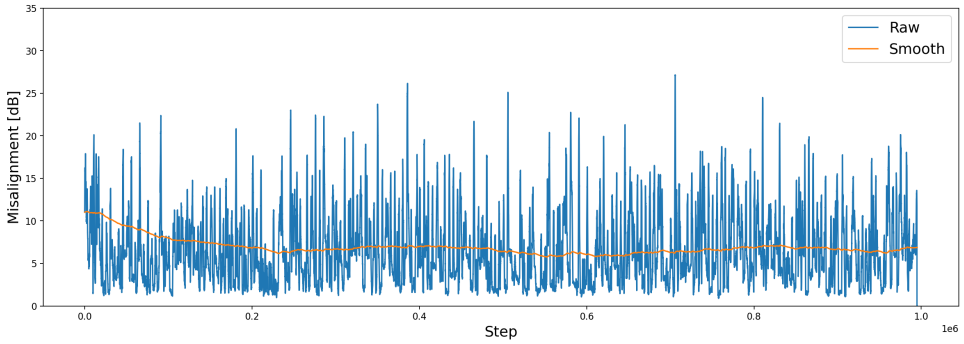
(b) Average misalignment for last 1000 steps.

Figure K.1: Results for test 1. Test parameter: State Space = [1, 0, 0].

K.3.2 Test No. 2



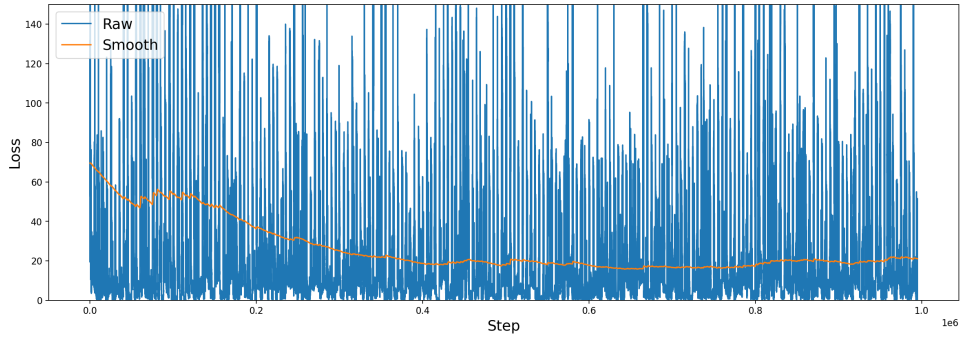
(a) Loss values.



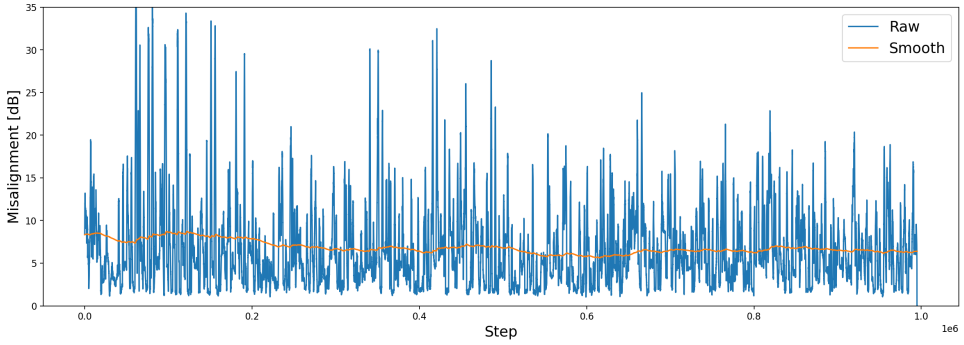
(b) Average misalignment for last 1000 steps.

Figure K.2: Results for test 2. Test parameter: State Space = [0, 1, 0].

K.3.3 Test No. 3



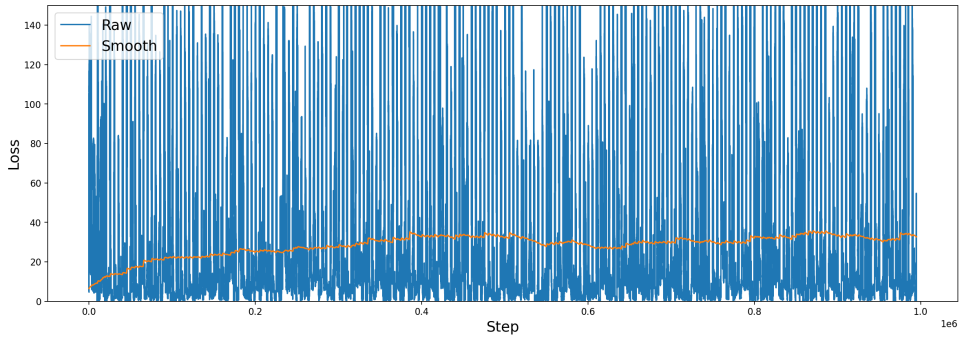
(a) Loss values.



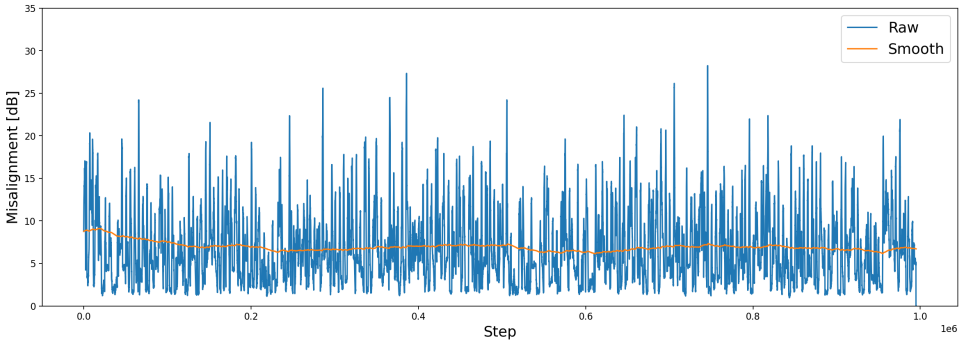
(b) Average misalignment for last 1000 steps.

Figure K.3: Results for test 3. Test parameter: State Space = [0, 0, 1].

K.3.4 Test No. 4



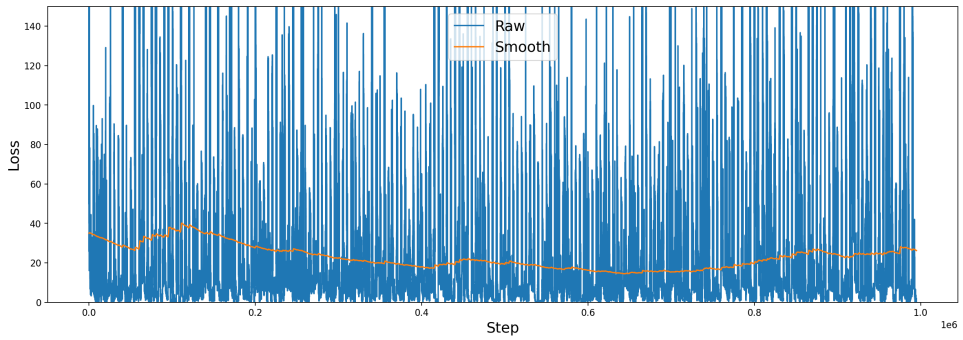
(a) Loss values.



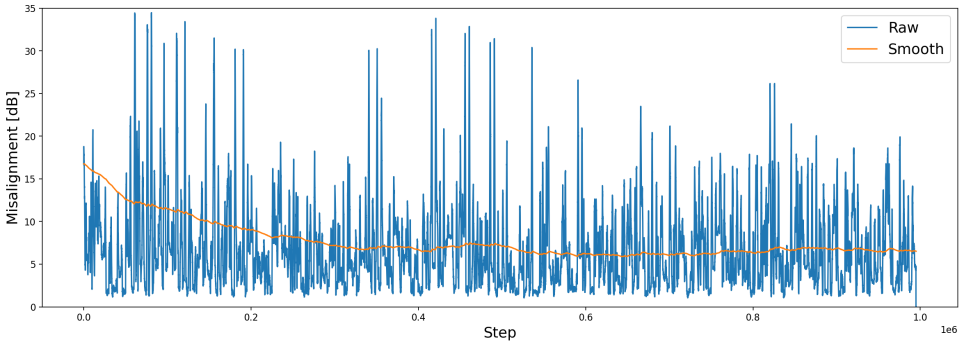
(b) Average Misalignment for last 1000 steps.

Figure K.4: Results for test 4. Test parameter: State Space = [1, 1, 0].

K.3.5 Test No. 5



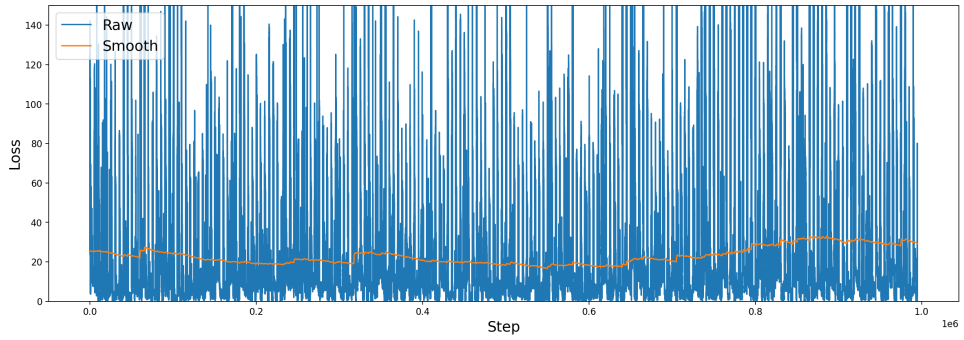
(a) Loss values.



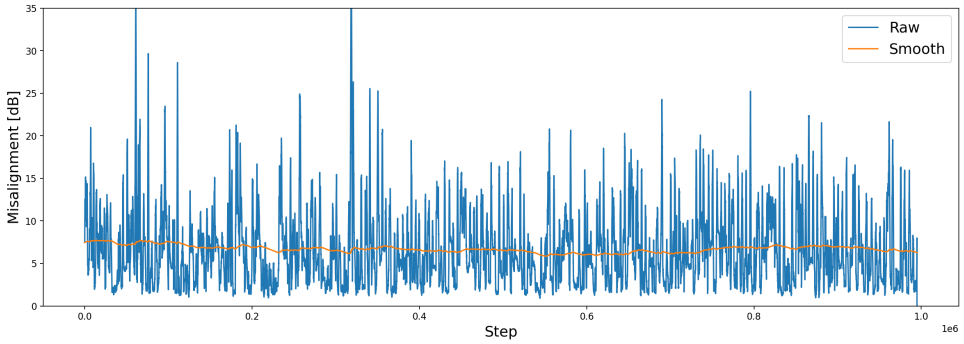
(b) Average misalignment for last 1000 steps.

Figure K.5: Results for test 5. Test parameter: State Space = [0, 1, 1].

K.3.6 Test No. 6



(a) Loss values.



(b) Average misalignment for last 1000 steps.

Figure K.6: Results for test 6. Test parameter: State Space = [1, 0, 1].

Test: Reference L

This appendix will document the test of different threshold values for the reference algorithm.

L.1 Test Parameters

This test will use different thresholds as seen in table L.1.

Test No.	Threshold
1	0.7
2	0.8
3	0.9
4	1.0
5	1.1
6	1.2
7	1.3
8	1.4
9	1.5
10	1.6

Table L.1: Different threshold values that will be tested.

L.2 Results

Test No.	Threshold	Avg. Misalignment [dB]
1	0.7	25.328
2	0.8	21.106
3	0.9	18.166
4	1	14.51
5	1.1	11.543
6	1.2	8.493
7	1.3	7.197
8	1.4	7.763
9	1.5	12.269
10	1.6	17.083

Table L.2: Results for different threshold values.

L.2.1 Test No. 1

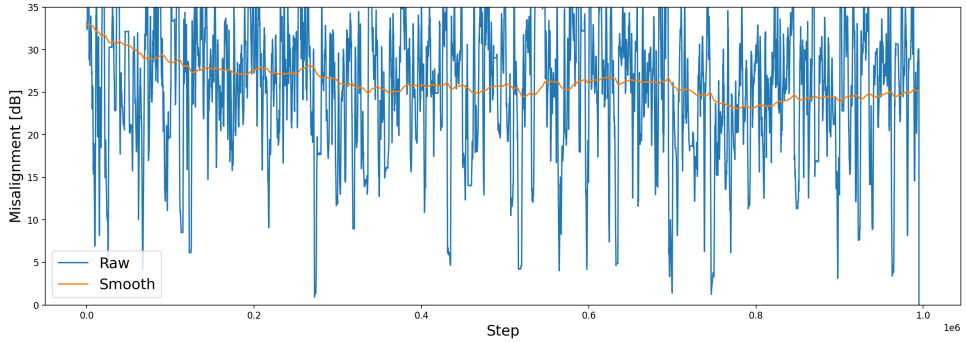


Figure L.1: Results for test 1. Test parameter: Threshold = 0.7.

L.2.2 Test No. 2

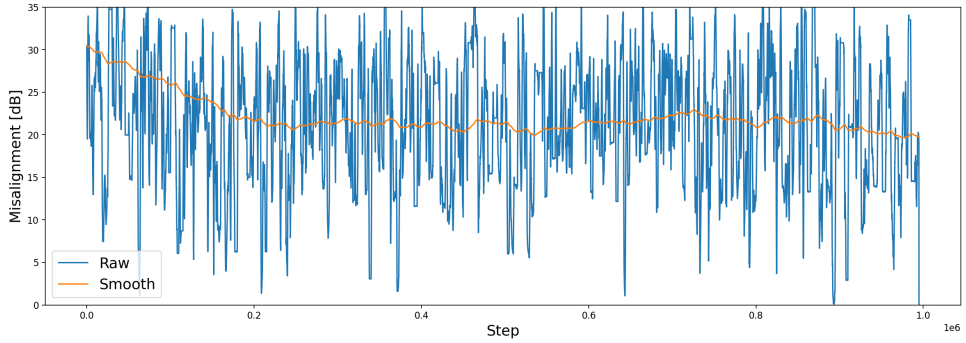


Figure L.2: Results for test 1. Test parameter: Threshold = 0.8.

L.2.3 Test No. 3

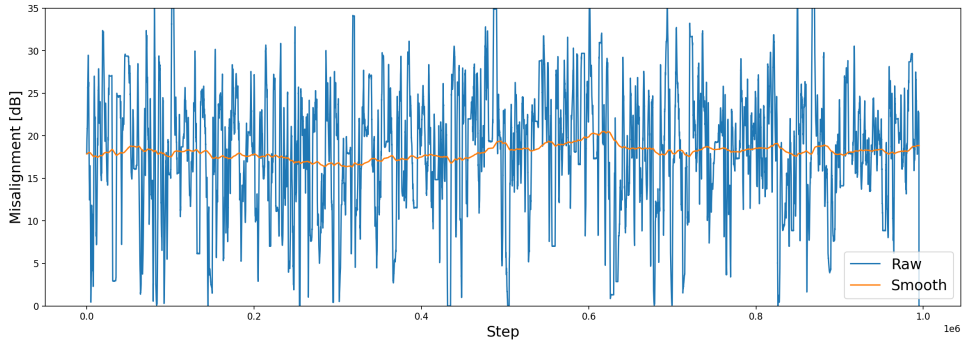


Figure L.3: Results for test 1. Test parameter: Threshold = 0.9.

L.2.4 Test No. 4

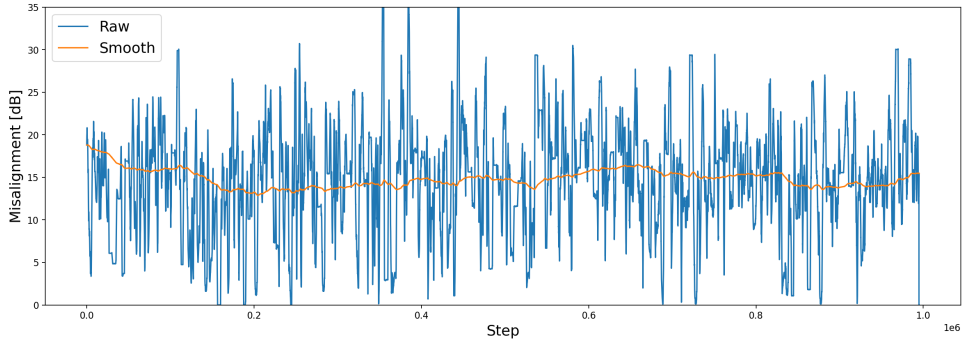


Figure L.4: Results for test 1. Test parameter: Threshold = 1.0.

L.2.5 Test No. 5

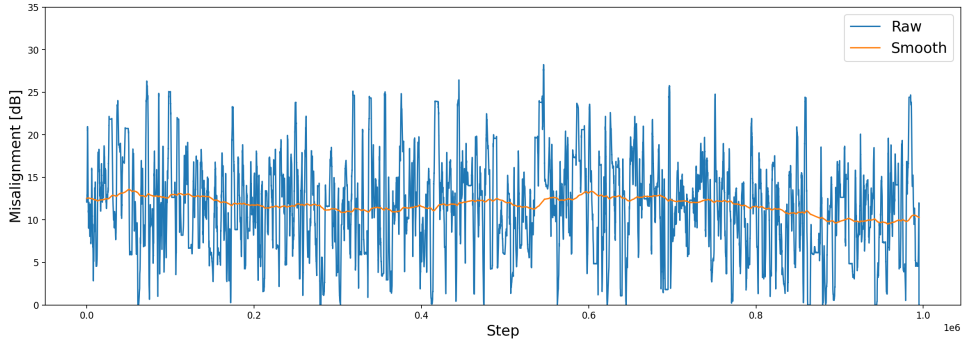


Figure L.5: Results for test 1. Test parameter: Threshold = 1.1.

L.2.6 Test No. 6

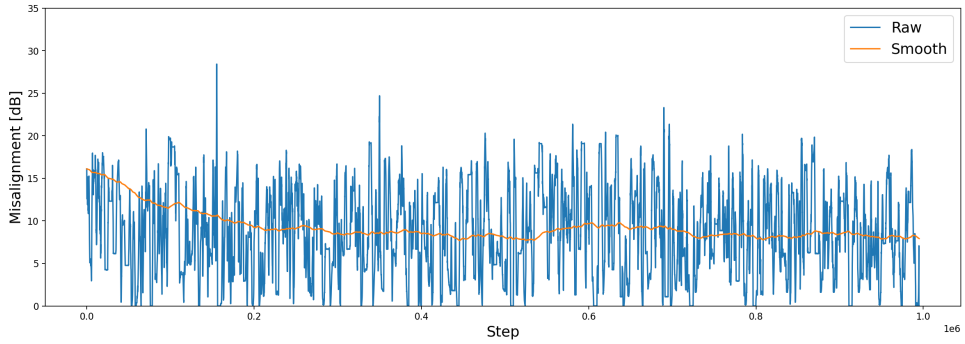


Figure L.6: Results for test 1. Test parameter: Threshold = 1.2.

L.2.7 Test No. 7

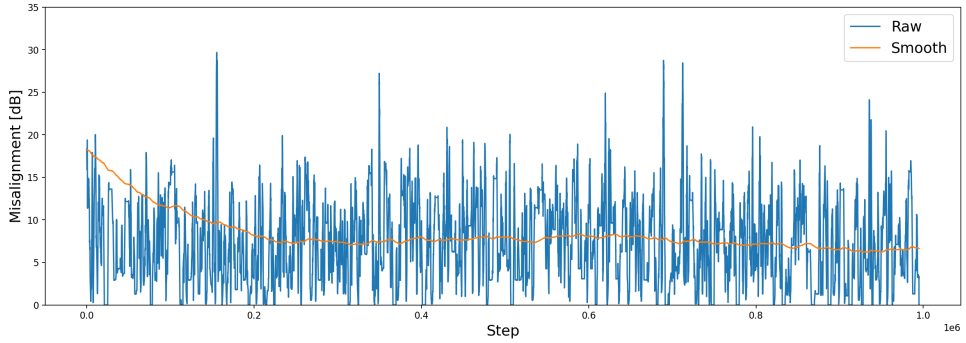


Figure L.7: Results for test 1. Test parameter: Threshold = 1.3.

L.2.8 Test No. 8

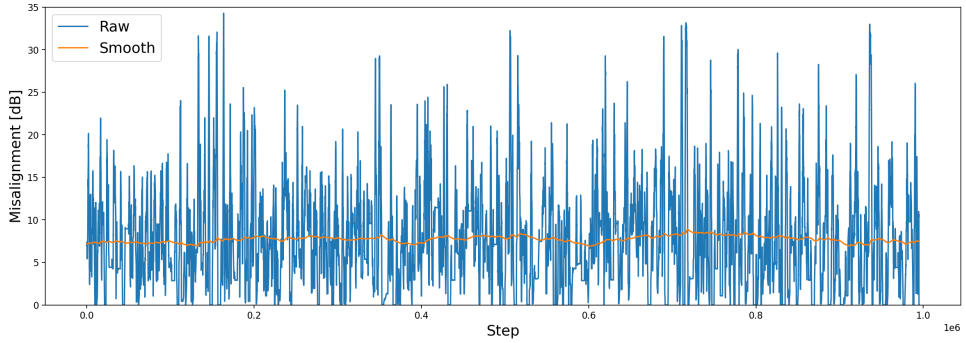


Figure L.8: Results for test 1. Test parameter: Threshold = 1.4.

L.2.9 Test No. 9

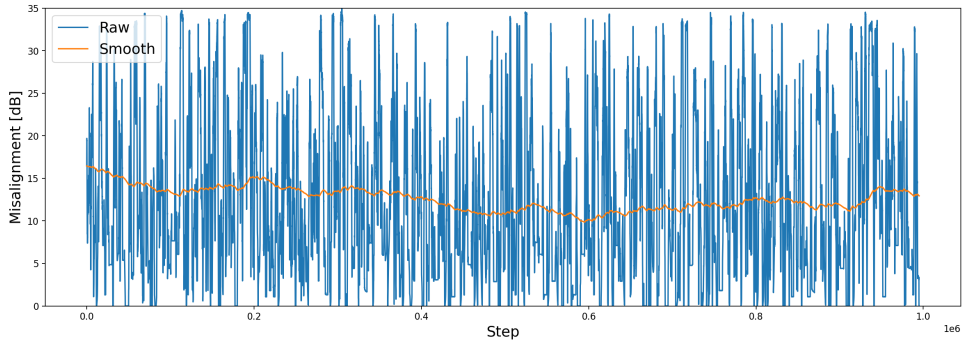


Figure L.9: Results for test 1. Test parameter: Threshold = 1.5.

L.2.10 Test No. 10

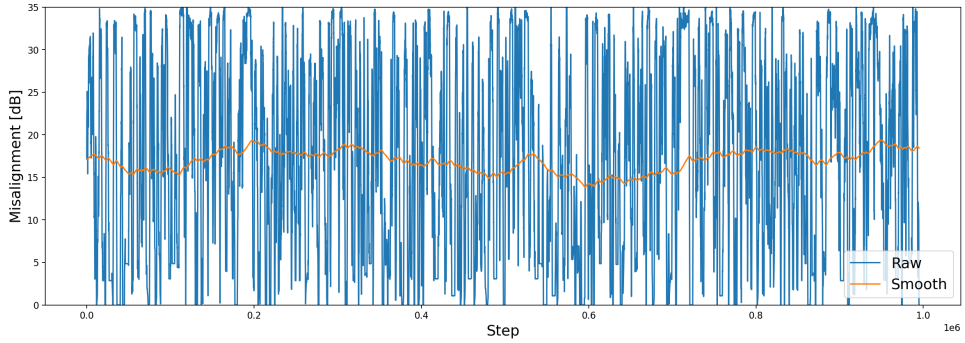


Figure L.10: Results for test 1. Test parameter: Threshold = 1.6.

Test: DQN and Reference algorithm

M

This appendix will document the test of the final DQN tuned for *Car - LOS* and the reference algorithm on the four different scenarios. The test will be done on the created data sets seen on fig. 5.3 described in section 5.2 with the parameters seen in table 2.1.

M.1 Static Parameters

The static parameters for this test can been seen in table M.1 and for the reference algorithm a threshold value of 1.3 as found in section 6.5 will be used.

	Hyper parameter	Value Space
DQN	Hidden Layers	[50, 50]
	Learning Rate (ADAM)	0.01
	Memory Size	1000
	Batch Size	500
	Target Update	10
RL	Forgetting Factor (γ)	0.8
	Exploring Factor (ϵ)	0.05
	State Space	$[a_{-1}, a_{-2}, a_{-3}, \theta_0, \theta_{-1}, \theta_{-2}, x_0, x_{-1}, x_{-2}, y_0, y_{-1}, y_{-2}]$

Table M.1: The final hyper parameter values for the DQN

M.2 Test Parameters

This test will use four different scenarios as seen in table G.2.

	Test No.	Scenario
DQN	1	Car LOS
	2	Car NLOS
	3	Pedestrian LOS
	4	Pedestrian NLOS
Reference	5	Car LOS
	6	Car NLOS
	7	Pedestrian LOS
	8	Pedestrian NLOS

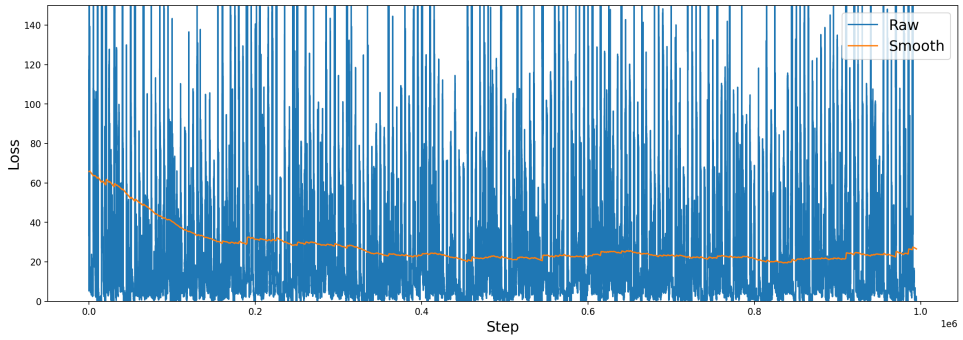
Table M.2: Scenarios which will be tested.

M.3 Results

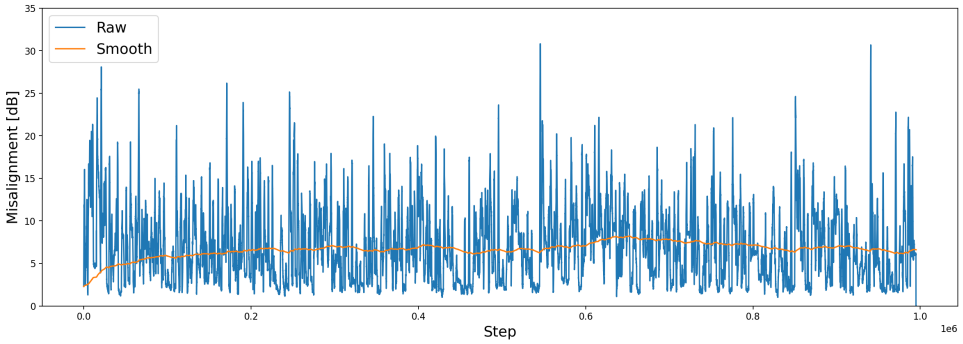
	Test No.	Avg. Loss	Avg. Misalignment [dB]
DQN	1	23.737	7.065
	2	57.483	10.652
	3	27.304	7.268
	4	53.477	11.356
Reference	5	-	13.437
	6	-	30.972
	7	-	11.471
	8	-	29.851

Table M.3: Results for testing different scenarios.

M.3.1 Test No. 1



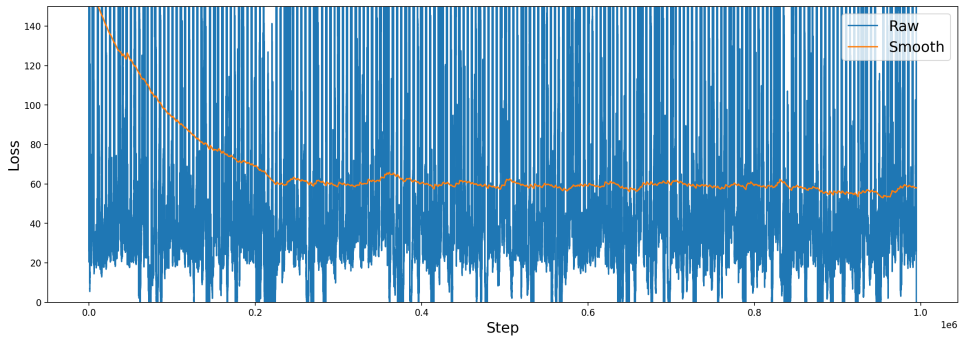
(a) Loss values.



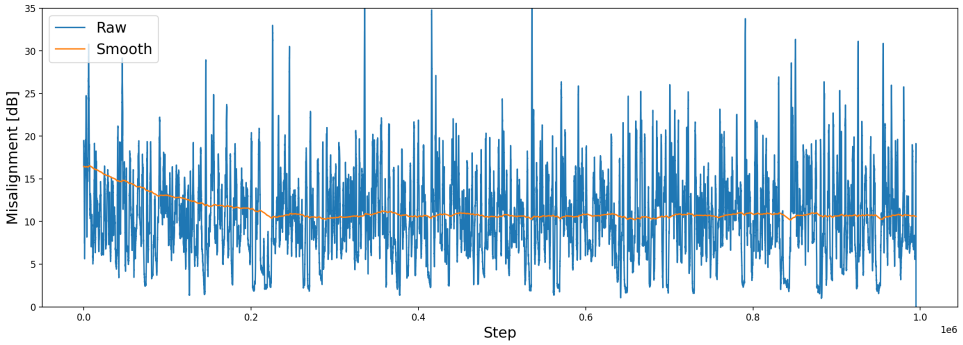
(b) Average misalignment for last 1000 steps.

Figure M.1: Results for test 1. Test parameter: DQN - Scenario = Car LOS.

M.3.2 Test No. 2



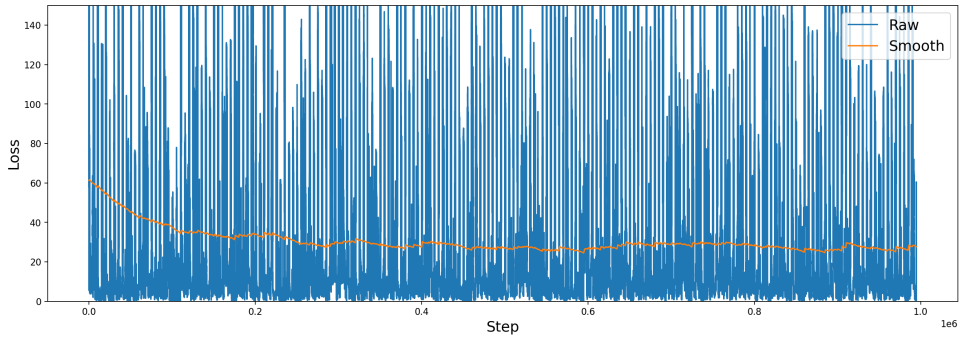
(a) Loss values.



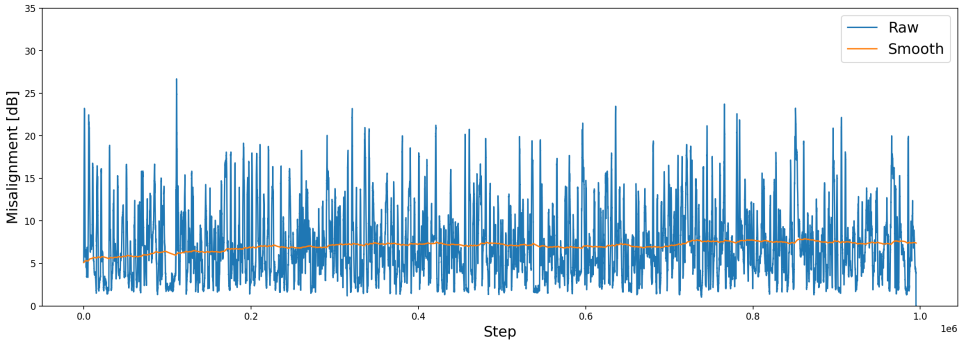
(b) Average misalignment for last 1000 steps.

Figure M.2: Results for test 2. Test parameter: DQN - Scenario = Car NLOS.

M.3.3 Test No. 3



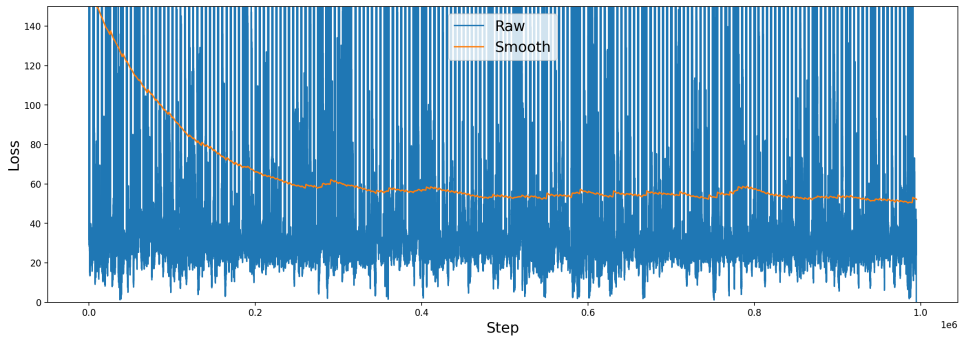
(a) Loss values.



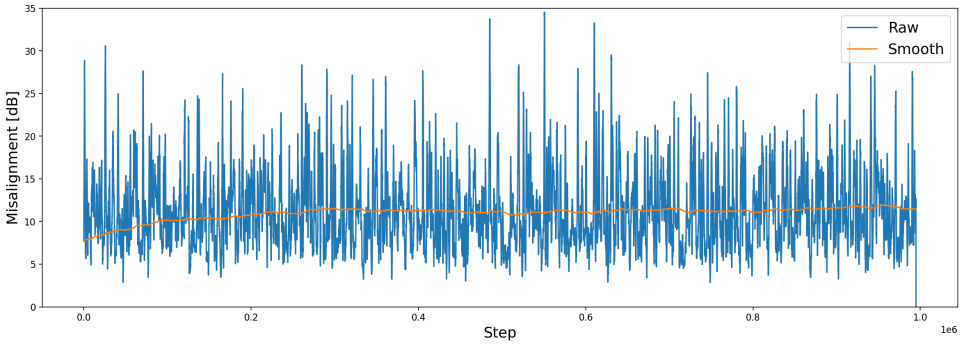
(b) Average misalignment for last 1000 steps.

Figure M.3: Results for test 3. Test parameter: DQN - Scenario = Pedestrian LOS.

M.3.4 Test No. 4



(a) Loss values.



(b) Average misalignment for last 1000 steps.

Figure M.4: Results for test 4. Test parameter: DQN - Scenario = Pedestrian NLOS.

M.3.5 Test No. 5

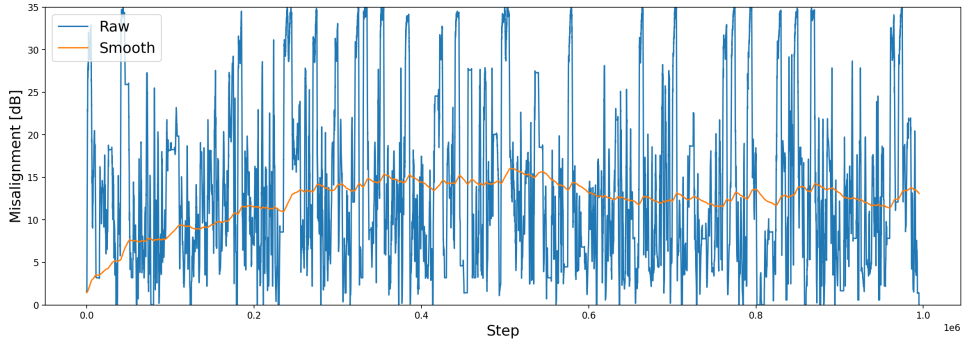


Figure M.5: Results for test 1. Test parameter: Reference - Scenario = Car LOS

M.3.6 Test No. 6

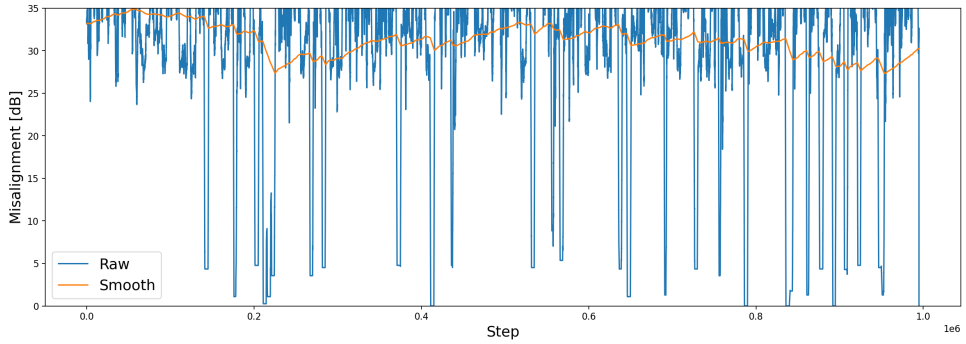


Figure M.6: Results for test 1. Test parameter: Reference - Scenario = Car NLOS

M.3.7 Test No. 7

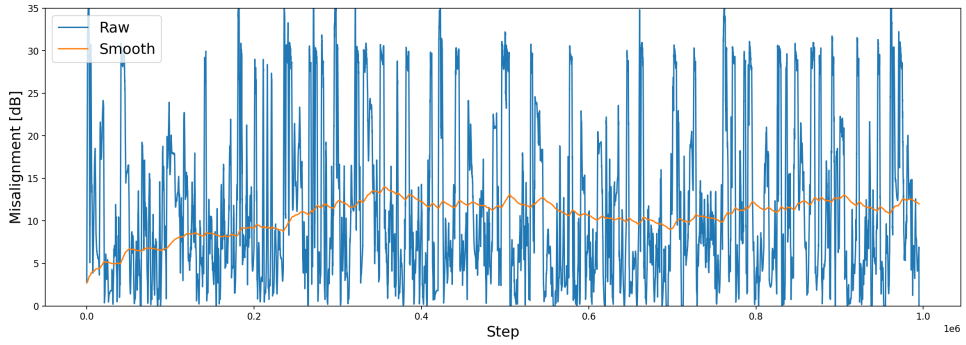


Figure M.7: Results for test 1. Test parameter: Reference - Scenario = Pedestrian LOS

M.3.8 Test No. 8

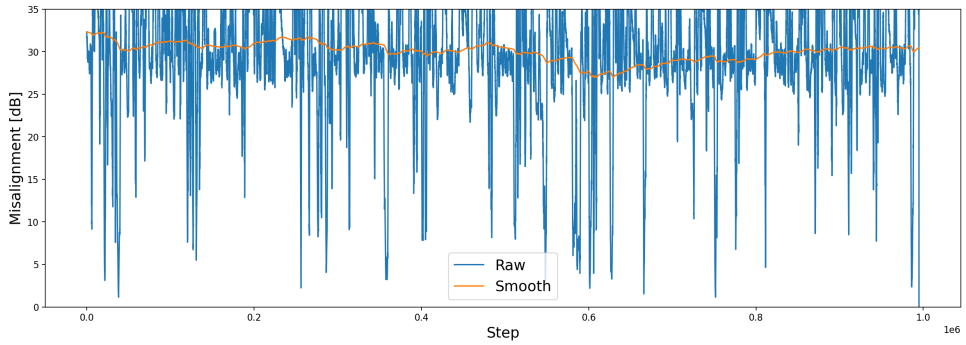


Figure M.8: Results for test 1. Test parameter: Reference - Scenario = Pedestrian NLOS.

Test: Upscaling Antennas and Beams

N

This appendix will document the test of the final DQN tuned for *Car - LOS* and the reference algorithm on the four different scenarios with an upscaled number of antennas and beams equal to 8 antennas and 16 beams for the UE and 16 antennas and 32 beams for the BSs. The test will be done on the created data sets seen on fig. 5.3 described in section 5.2 with the parameters seen in table 2.1.

N.1 Static Parameters

The static parameters for this test can been seen in table N.1 and for the reference algorithm a threshold value of 1.3 as found in section 6.5 will be used.

	Hyper parameter	Value Space
DQN	Hidden Layers	[50, 50]
	Learning Rate (ADAM)	0.01
	Memory Size	1000
	Batch Size	500
	Target Update	10
RL	Forgetting Factor (γ)	0.8
	Exploring Factor (ϵ)	0.05
	State Space	$[a_{-1}, a_{-2}, a_{-3}, \theta_0, \theta_{-1}, \theta_{-2}, x_0, x_{-1}, x_{-2}, y_0, y_{-1}, y_{-2}]$

Table N.1: The final hyper parameter values for the DQN

N.2 Test Parameters

The tests will be using the upscaled number of antennas and beams on the four different scenarios as seen in table N.2.

	Test No.	Scenario
DQN	1	Car LOS
	2	Car NLOS
	3	Pedestrian LOS
	4	Pedestrian NLOS
Reference	5	Car LOS
	6	Car NLOS
	7	Pedestrian LOS
	8	Pedestrian NLOS

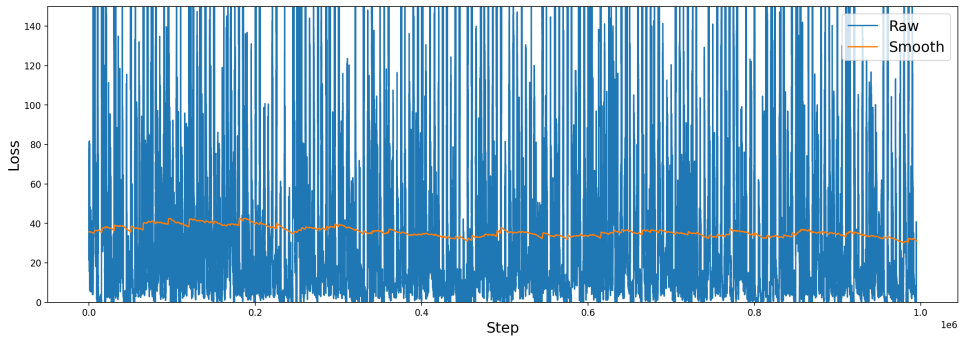
Table N.2: Scenarios which will be tested.

N.3 Results

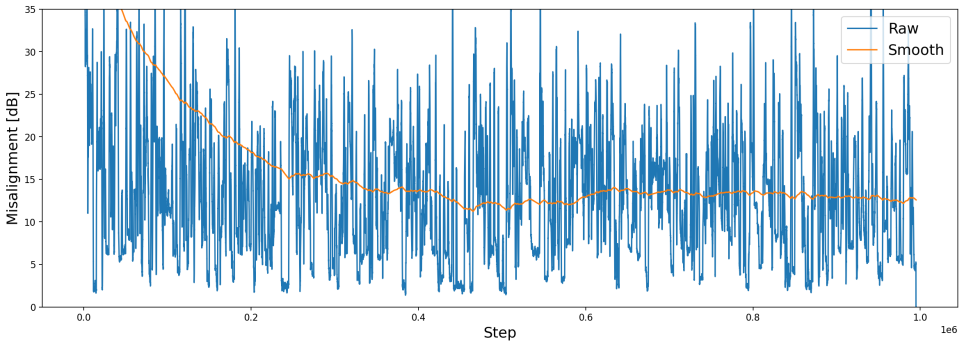
	Test No.	Avg. Loss	Avg. Misalignment [dB]
DQN	1	35.397	13.189
	2	63.168	16.319
	3	38.225	13.227
	4	61.065	17.099
Reference	5	-	15.868
	6	-	34.497
	7	-	16.359
	8	-	32.275

Table N.3: Results for testing different scenarios.

N.4 Test No. 1



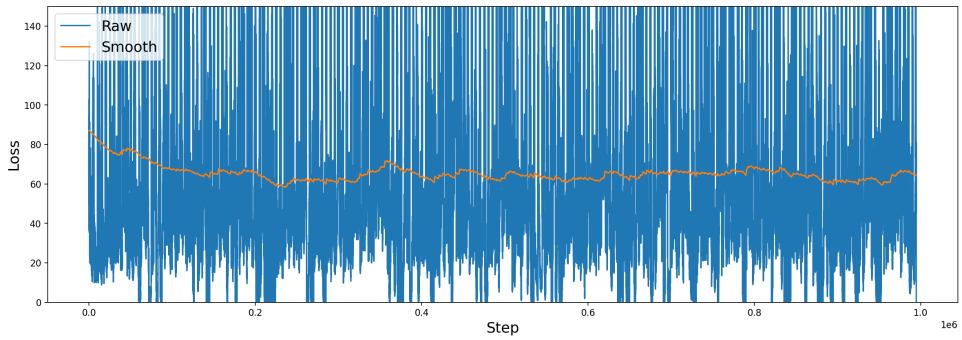
(a) Loss values.



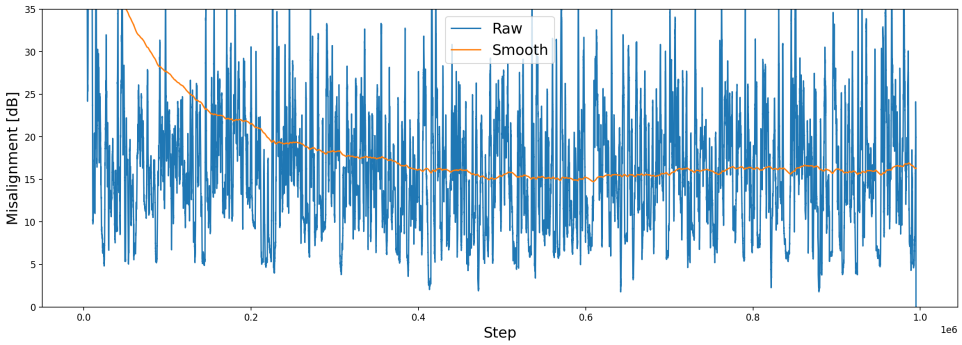
(b) Average misalignment for last 1000 steps.

Figure N.1: Results for test 1. Test parameter: DQN - Scenario = Car LOS.

N.5 Test No. 2



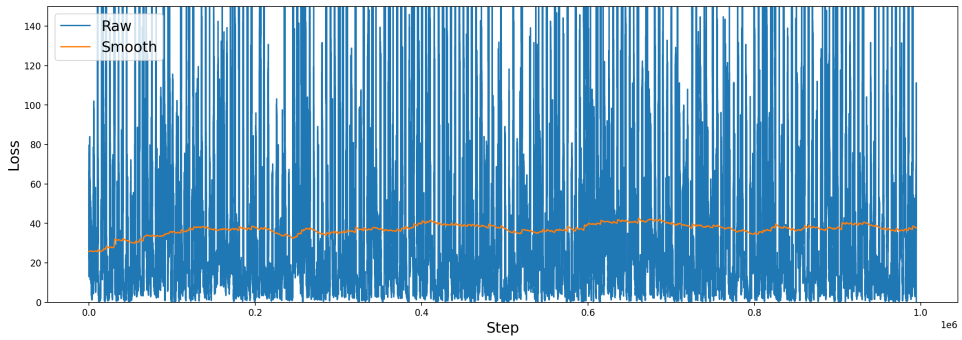
(a) Loss values.



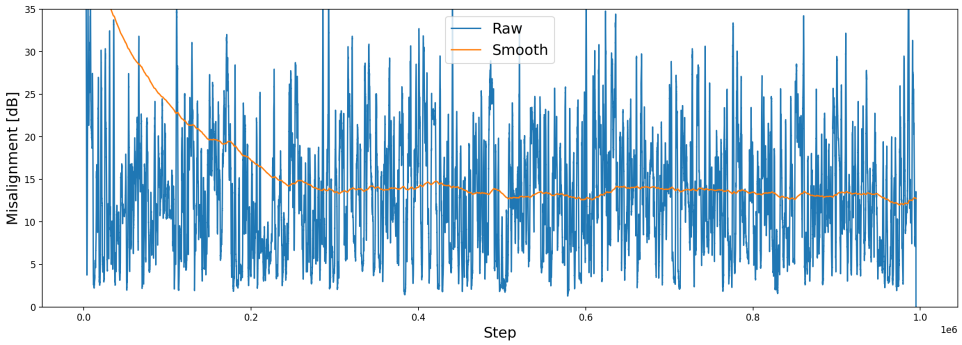
(b) Average misalignment for last 1000 steps.

Figure N.2: Results for test 2. Test parameter: DQN - Scenario = Car NLOS.

N.6 Test No. 3



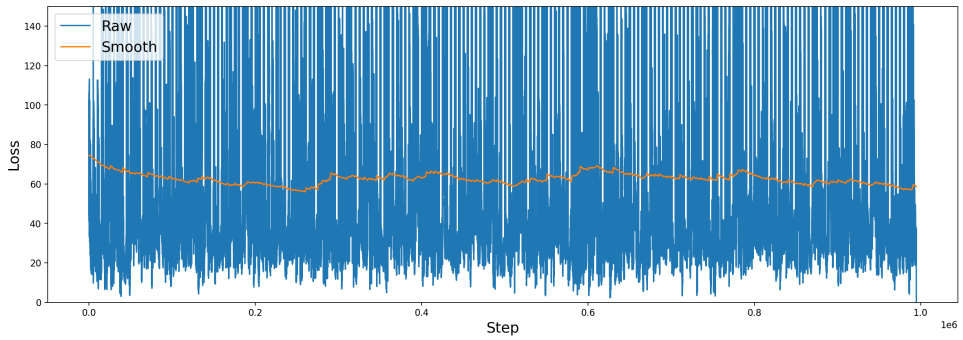
(a) Loss values.



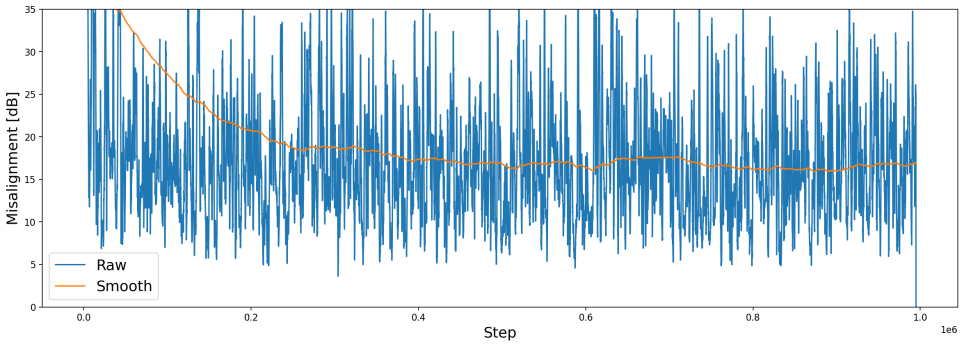
(b) Average misalignment for last 1000 steps.

Figure N.3: Results for test 3. Test parameter: DQN - Scenario = Pedestrian LOS.

N.7 Test No. 4



(a) Loss values.



(b) Average misalignment for last 1000 steps.

Figure N.4: Results for test 4. Test parameter: DQN - Scenario = Pedestrian NLOS.

N.8 Test No. 5

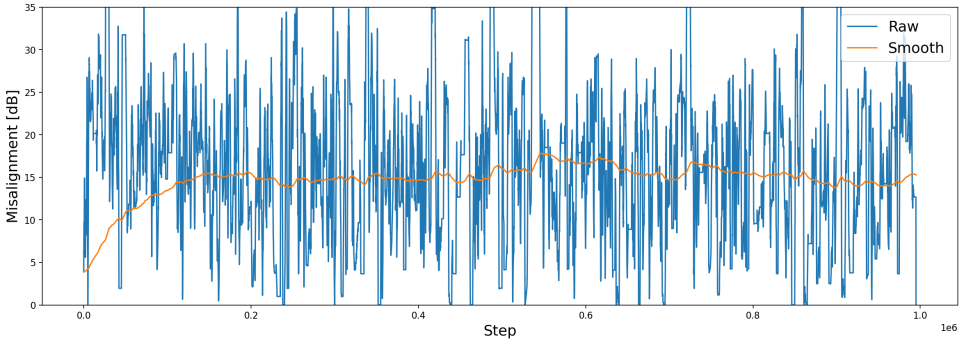


Figure N.5: Results for test 1. Test parameter: Reference - Scenario = Car LOS

N.9 Test No. 6

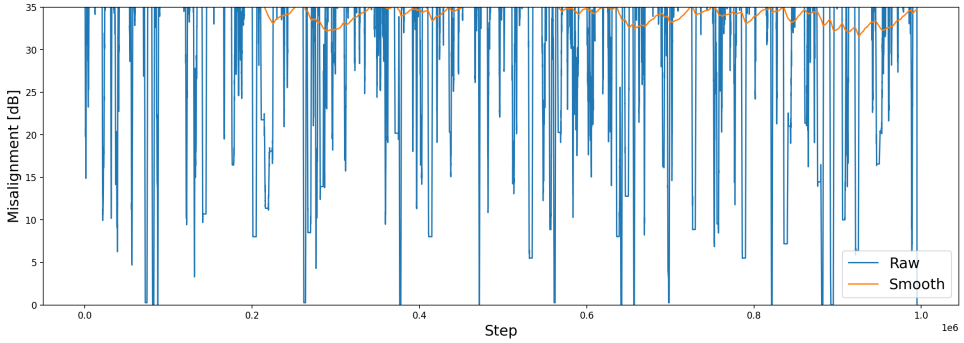


Figure N.6: Results for test 1. Test parameter: Reference - Scenario = Car NLOS

N.10 Test No. 7

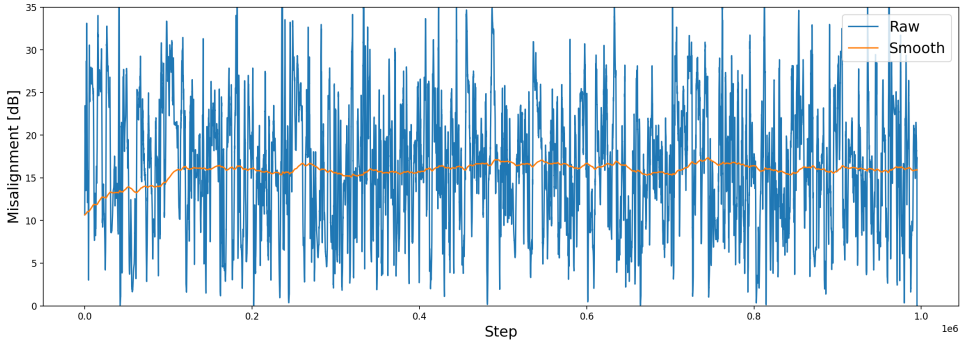


Figure N.7: Results for test 1. Test parameter: Reference - Scenario = Pedestrian LOS

N.11 Test No. 8

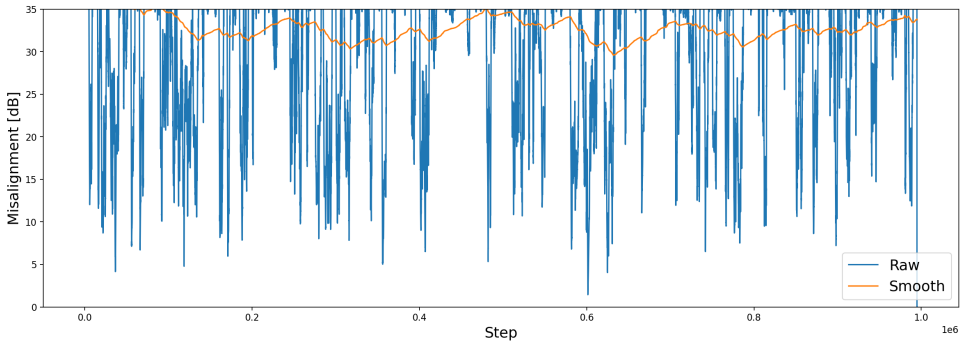


Figure N.8: Results for test 1. Test parameter: Reference - Scenario = Pedestrian NLOS.