# Prosthetic vision for the blind: Intelligent optimization of limited vision Testing trained computer agents for phosphene vision in a realistic environment

Berfu Karaca[1]

**Supervisors:**
Marcel van Gerven[1], Umut Güçlü[1], Burcu Küçükoglu[1], Jaap de Ruyter van Steveninck[1]

**Affiliations:**
[1]Donders Institute for Brain, Cognition and Behaviour, Radboud University Nijmegen, The Netherlands

**Corresponding author:**
Berfu Karaca, berfu.karaca@ru.nl

**Abstract:**

Blindness is a common societal problem that affects day-to-day functioning. Even though there is no effective treatment yet, there are some alternative ways such as neuroprosthetic visual implants. Although prosthetic vision does not provide normal vision, it does provide a rudimentary form of the environment through point-like flashes known as phosphenes which might still help basic activities like navigation. However, due to biological limitations, the current implants have low resolution. The limited capacity of resolution increases the need for optimal information extraction from the scene for efficient understanding of the environment. In line with this, there is a need for better image pre-processing techniques.

One limitation of the studies testing phosphene vision is the necessary surgical operation for implants. For this reason, researchers found other techniques to test phosphene vision. One solution is testing sighted participants with wearable head-mounted displays (e.g., VR) that convert the real scene to processed phosphene vision. However, studies suggested that different image pre-processing techniques should be used for different contexts and scenes which requires an optimization adaptive to the scene. The variability in parameters to be tested and the need for optimization raise other challenges such as significantly increased number of tests for optimization problems which also means increased cost.

This project will contribute to our knowledge on the potential use of trained Deep Reinforcement Learning agents to test the performance in particular tests such as navigation rather than human participants which is likely to minimize the cost and accelerate the process of optimization. Additionally, as these models will be used in real life, using a realistic virtual environment to test the behaviour of the trained agent and to optimize the parameters of simulated phosphene vision will provide more applicable results for future studies on prosthetic implants.

**Keywords:** Phosphene vision, visual loss, visual prosthesis, deep learning, artificial intelligence, realistic virtual environment, computer agent

# Introduction

## 1.1 Background

The 2020 reports of the International Agency for the Prevention of Blindness demonstrates that globally 1.1 billion people are suffering from vision loss and researchers expect that the number of people with vision loss will rise to 1.7 billion until 2050. Additionally, more than 500 million people live with mild and severe vision impairments ("Updated vision atlas shows 1.1 billion people have vision loss," 2021).

Relying on these statistics, we can say vision loss is a non-negligible societal problem especially when we think about the fact that we significantly rely on visual cues to accomplish most of the daily activities like navigation, object recognition or face recognition. Studies support that, vision loss might cause several other problems such as reduced life quality; dependence to another during casual activities like reading, writing, shopping, driving; fractures or injuries; cognitive impairments; higher risk for depression or mental health disorders; mobility and even mortality (Grover, 2017). There is no successful treatment for blindness, yet. However, through the development of technology, acquiring some form of vision is possible for visually impaired individuals.

As a result of the loss of daily functionalities mentioned above such as walking around avoiding obstacles, there is an increased demand in studies to provide vision. One of the promising solutions for providing vision is visual prostheses studies which are implantable electronic micro-systems. The implants convert the images of the outside world into a meaningful representation through point-like flashes known as phosphenes.

### 1.1.1 Retinal and Cortical Implants

Only in the early 20$^{th}$ century did artificial vision experience its greatest breakthrough. However, the restoration of vision has been a well-focused area of research for many more years. In 1929, Foerster discovered that electrically stimulating visual cortex induces perception of small points of lights or in other words phosphenes, which has provided the scientific basis to develop prosthetic vision for the blind (Fernández et al., 2020).

In 1968, researchers at the University of Cambridge attempted to implant radio receivers connected to the electrodes under the scalp of the blind people and sent pulses to these electrodes with the aid of a cap. The pulses activated the electrons and created phosphenes in the patients' visual fields. However, only a few electrodes were activated and even some have caused pain in patients. Despite the limitations, the researchers suggested that, with the improvements made to their prototype, the prosthetic vision will enable the blind people to walk by avoiding obstacles, to read and write. Despite the many other studies conducted, the research area of visual prosthetics showed significant progress after this first implementation study in the 1960s (Brindley & Lewin, 1968).

In 1979, scientists at the University of Utah published another study relying on findings of Brindley and Lewin. They started their experiments in 1969, with blind participants who already needed a surgical operation for the removal of tumors/lesions to minimize the risk. They have tested their implant with 37 participants in 4 years. After these trials, in 1973, they were satisfied with the experiences of successful implants and started to try temporary implants with healthy blind volunteers. The results of their previous experiments showed recognition of basic patterns such as triangles or some of the letters. The promising results led the scientists to try chronic implantation. Before starting the experiments on chronic implantation, they have developed the methods used in previous studies to get rid of some limitations such as eliminating crosstalk between different electrodes and providing more reliable results. Additionally, they have restricted the blind volunteers to individuals totally blind but with the memory of visual sensations. At the end of their attempts, their results showed that subjects were able to perceive the image without special training and the speed of reading the braille was much faster than reading by using the finger (Dobelle et al., 1979). Even with more promising results, successful implants are still not available yet to use in humans. In a recent study, researchers also tried these implants using trained monkeys with several tests as participants and demonstrated the stimulation of the electrodes in V1 provides a successful perception of the shapes which again demonstrates the promising capacity for creating phosphene vision (Chen et al., 2020).

An obvious challenge in this area of research is the need for surgical operations to test and develop the implants which has high risk and cost. The technological developments allowed researchers to test the vision provided by the implants in an easier and less costly way. For example, there are many studies that benefit from wearable head-mounted displays (e.g., VR glasses) to simulate phosphene vision for sighted subjects to test in various tasks. This development in the area also facilitates the research to test and develop the parameters of the prosthetic vision. This is one of the most significant aspects in this area of research because the resolution or in other words the number of phosphenes that can be simulated is

limited. Additionally, as the resolution is limited, there is a need for better image pre-processing techniques which allow researchers to extract the meaningful visual cues for particular tasks.

### 1.1.2 Resolution

It is known that there is a compromise between the resolution and the size of the implant which must be optimized. In other words, there is a need for optimization of the visual perception generated with limited resolution. Researchers tried to find the balance between the size of the implants and the resolution to get meaningful information. In one of the studies, Hayes et al. (2003) tried different resolutions to test the functionality of the simulated vision. Researchers tested the performance of sighted participants wearing head-mounted display by using pixelized images during various tasks such as object recognition/discrimination, symbol recognition, pouring, reading etc. The results demonstrated increased performance with higher resolution in all tests. Some of the subjects were able to recognize simple objects and symbols with lower resolution. Relying on that, researchers suggested that using fewer pixels of light is also enough to perform tasks. However, one of the limitations of the study is the simplicity of the images used during these tasks (Hayes et al., 2003).

The resolution or the number of phosphenes needed to understand the environment might depend on other factors like the complexity of the environment or the amount of time the participant practiced.
Dagnelia et al. (2007) tested wayfinding with limited visual resources to find out minimal requirements for visual resolution. They found out the inexperienced participants needed higher resolution for adequate performance. Similarly, Srivastava et al. (2009) focused on the amount of experience gained by the participant and the adaptation to the phosphene vision. As a result of their surgical studies, they suggested the possibility of implanting electrodes up to 650. They have tested the performance in 3 different conditions with different percentages of dropouts as 0%, 25%, 50%. They claimed that there is an improvement in the performance with practice and it is possible to adapt to low-resolution images with 50% dropout or in other words with only 325 phosphenes.

Researchers have focused on phosphene vision for many different capabilities such as object segmentation or facial recognition (Bollen et al., 2019; Lu et al., 2013; Sanchez-Garcia, 2018; Thompson et al., 2003) However, many researchers found mobility/navigation and obstacle avoidance ability more important for daily functioning. McCarthy et al. (2014) made a study by using obstacle avoidance tasks with low to medium resolution and used different luminances for surface-obstructions segmentation. They believed that a better representation of the environment is required. In this study they sought to improve the segmentation of ground surface and obstructions by depth-perception. Results demonstrated that depth-based representations of obstacles provided more efficient visual navigation performance with low resolution. As a significant limitation of this study, they have only used static images (McCarthy et al., 2014).

In summary, studies demonstrate that different tasks or environments might require different pre-processing techniques to provide adequate and meaningful information with limited resolution. Phosphene vision is not as informative as normal vision as a result of the lack of spatial resolution, colours, and contrast. That's why there is a need for simplification of the scene. However, the simplification of the scene requires efficient reduction of information which leads to the need for better image pre-processing techniques to extract more precise visual cues that helps the interpretation of the environment (Sanchez-Garcia et al., 2020).

### 1.1.3 Different Image Pre-Processing Techniques

To meet that demand of simplification, many researchers focused on different image pre-processing techniques for transmitting an image to phosphene vision. They have developed different filters for edge detection, texture detection or to extract the information of depth (Dowling, 2007).

Boyle et al. (2001) tested the object recognition of 174 sighted participants by using different resolutions with static images and compared the effect of different image pre-processing techniques. They found increased performance when they used 3-level gray images than black and white images. In line with this, they suggested that 3-level gray images provided higher spatial resolution, and this allowed higher object recognition even though this is highly correlated with other parameters such as the size of the object. Still, results demonstrated that higher resolution was more important than increased gray scale. Researchers also supported the importance of the context of the scene for the recognition and emphasized the need of adaptive image pre-processing techniques for efficient processing of the scenes. Some of the many limitations of this study is the lack of realistic environment, dynamic stimuli, and lack of diversity in conditions in terms of resolution and colour as they didn't have conditions with high resolution and did not test with coloured stimuli (Boyle et al., 2001).

As a next step, Boyle et al. (2002) investigated the effect of different factors to enhance the visual information like brightness, resolution, contrast, edges, the distance to the object/depth and importance mapping. Researchers showed the significant role of edge detection for meaningful information especially with high resolution images. Additionally, they concluded that for the optimization to transfer the image to artificial visuals as phosphenes, importance ranking is important. Because with limited resolution the displayed features like edges must be extracted depending on the importance of the information (Boyle et al., 2002).

With a follow-up study, Boyle et al. (2003) suggested different parameters to consider according to the types of the scene by relying on their previous findings. They matched different descriptors like motion, colour, and edges with different types of scenes to gain the most efficient information from the pre-processed scene. For example, as the working range is about 1 meter in an office, a low visual range is enough. However, for an outdoor environment, the required visual range might be significantly higher. By the same token, a street environment includes moving obstacles like people or cars; while motion is a highly descriptive for a street environment, it is lower for an indoor environment (Boyle et al., 2003). Other researchers also contributed similar results about the significance of filtering for the task performance (Hallum et al., 2005).

Vergnieux et al. (2017) also studied prosthetics and concentrated on navigational tasks. They hypothesized that low resolution implants with no special processing cause overcrowding and provide insufficient visual cues for navigation tasks. They tested the effect of various renderings which simplify visual information on the performance in navigation tasks. As one strategy, they tried to limit the viewing distance of the participants. This strategy helped to achieve better performance while reducing the cognitive load during the task. A key finding of this study was that having only the edges present in the environment lessened the cognitive load and led to improved performance in the navigation task. However, they have not assessed that under real conditions (Vergnieux et al., 2017).

Researchers at the Donders Institute are also working in this area of research to further understand prosthetic vision. Researchers are testing sighted participants by simulating phosphene vision generated via the implants by using VR glasses in real world experimental settings. In a previous study, Ruyter van Steveninck et al. (2022) claimed that scene simplification obtained as a result of image pre-processing such as the extraction of contours may improve the performance in navigation tasks. In this study, they tested the performance with different resolutions of phosphenes by keeping the light intensity constant. Also, to test the effect of complexity of the environment, they have used a real corridor environment with different complexities by adding background and surface textures. Relying on their findings, they showed that 26x26 resolution provided sufficient visual cues for the participants with contour-based simulation. However, the performance was lowered in the condition of a complex environmental setting with textures.

In addition, researchers supported that an efficient technique of image pre-processing is required to simplify the scene which prevents overcrowding in the scene (De Ruyter van Steveninck et al., 2022). In conclusion, there is a need for simplification of scenes to extract the meaningful cues from the scene with limited resolution. Yet, there is a need for balance to avoid eliminating necessary visual cues which may differ depending on the context of the scene.

### 1.1.4 Reinforcement Learning

Sam Danen et al. (2021) tried to use reinforcement learning to test prosthetic vision and automatically optimize the parameters of image pre-processing in his thesis project. They attempted to compare the performances of human participants and trained computer agents to evaluate usability of computer agents instead of human participants. They have used wayfinding task in the experiment. In this project, they had 2 different experimental settings. In the first experiment, they tried to monitor the change in performance in different experimental settings with different visual complexities and different phosphene resolutions. In the second experiment, they let the reinforcement learning agent find the optimum threshold for canny edge detection model in different conditions. They found a significant change in model learning as they altered the light intensity of the environment. When the light intensity was the lowest, the performance of the agent was improved. However, they are suspicious about whether the lowest light intensity of the environment was also better for human participants' performances. Still, by relying on the similar graphs of performances with human participants, they argue that there are many similarities between the performance of human participants and computer agents which makes the use of computer agents as opposed to human participants a promising method. Even though this study provided a comparison for the performances of human participants and computer agents, some of the limitations of the previous studies were still present. Firstly, the virtual environment used was an unrealistic basic corridor environment. Additionally, the task used was a simple

navigation task that the agent only walked forward through the hallway.

There have also been more complicated studies conducted by using indoor visual navigation task (Krantz et al., 2021; Savva et al., 2019; Zhu et al., 2017). In one of the studies, Savva et al. (2018) used AI-Habitat which is a simulation platform to train AI-agents. In their experimental setting, the agent was using different detectors such as depth, RGB camera and GPS + Compass sensor. With the GPS + Compass sensor, the agent had the input of relevant location and orientation of the goal for its current location. Rather than only defining three actions to move forward and 10° rotations, they also included a stop action. With this additional motion, they wanted to ensure that the agent knows that it reached the target but not randomly came to that location. They found significant improvements with the sensory input of depth. Also, they supported that the RGB images were not performing significantly better than blind agents that chooses random actions during the training. They gave GPS input for both RGB agents and blind agents, and researchers suggested that this sensory input given both might be the reason for that (Savva et al., 2018).

### 1.2 Research Aim

To deal with limitations of previous studies' methods that included blind or sighted participants in basic, unrealistic environments that explained above, in this project we are training computer agents with Deep Reinforcement Learning in a realistic 3D indoor environment to reach human-level performance. The main aim of this study is investigating the usability of RL agents to develop simulated phosphene vision. By this way, we can provide an alternative for behavioural experiments with human participants to optimize phosphene models. Additionally, realistic virtual environments will allow more significant comparisons between the performances of RL agents and humans.

In summary, the aims of this research project with following provisional implications are: **1) Using a realistic environment to train and test the RL agent with phosphene vision.** This will clear the way for studies with more complex environments and tasks such as dynamic environments. In line with this, using a realistic virtual environment will widen the choices of experimental designs. Additionally, the realistic environment developed in this study will provide a baseline testing environment for future studies. **2) Comparing different parameters to train the agent, which contributes to the aim of training equivalent Deep RL agents to human participants.** Achieving this might decrease the costs significantly and facilitate studies in this area of research. **3) Comparing different image pre-processing techniques** which might allow us to see the effect of extracting visual cues and the contribution of different visual cues in a realistic setting for a navigation task.

### 1.2 Technical Background

#### 1.3.1 Reinforcement Learning

To optimize the simulated vision, testing sighed participants provides a better alternative than testing blind participants with implants in terms of the risk of the surgical operation and the cost. However, even when we test simulated vision with sighted participants, the cost still remains high and takes significant amount of time. For this reason, reinforcement learning may serve to find better ways to try and develop phosphene vision. The use of artificial agents as participants accelerates testing while decreasing costs.

The use of Reinforcement Learning is encouraged because of the similarity of the learning process with humans. The learning process of people includes learning the environment from the experiences via trial and error. Rational people seek to find the optimum decision by maximizing the reward while minimizing the penalty during life. We start to explore the environment and relying on the positive or negative feedback, we develop knowledge about our environment. Similarly, in machine learning, reinforcement learning (RL) is used as a decision-making concept that maximizes the long-term cumulative reward/return even in complex problems. With this feedback information which decreases the uncertainties in the environment, the agent can choose the optimal action in that state and move to the next state.

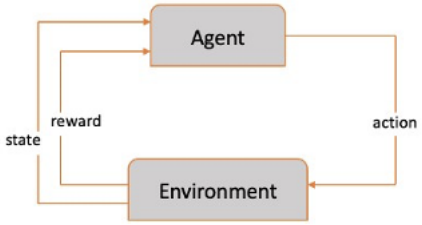



Figure 1.1. Reinforcement Learning Cycle

In a typical RL problem, there is an interaction between the agent and the environment. The agent chooses an action, and the environment provides rewards and the next state based on the chosen action of the agent. To formulate this problem, we need to understand the **Markov Decision Process (MDP)**.

To describe a Markov Decision Process, there are 5 elements should be captured (Garg et al., 2022).

$$m = (S, A, P_a, R_a, \gamma) \tag{1.1}$$

In this formula, the S stands for the **set of states** and A stands for the **set of actions**. $P_a$ or P(s,a,s') is the **transition probability** or in other words the probability that the chosen action a at current state will lead to state s' at the next state. $R_a$ or R(s,a,s') is the received **reward** after the transition after the action chosen from the current state to next state. Lastly, $\lambda$ is the **discount factor** to generate a discounted reward. It determines the significance of the future reward in different time steps. It changes between 0 and 1.

Fundamentally, the Markov Decision Process assumes that the environment and the agent interact at discrete time steps. In other words, at a specific discrete time-point or state, the agent performs an action and moves into its next state based on the transition function; this state and action provides the agent a reward. By this way, the agent learns the decision-making process based on the current state which shows the current state is already enough to choose the optimal action with the information retained via previous states.

More technically, the MDP aims to find the optimal policy **($\pi$)** to yield the optimal maximized long-term reward. **Policy** is basically a method of mapping from states to actions. Constructing the optimal policy is important to maximize the rewards corresponding to the actions. To reach this aim, the information of the **value of the action at a specific state** is required. Q-Values serve for this aim. Q(s, a) or state-action values gives the value or expected return for taking an action a in a state s in policy $\pi$.

$$Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\{(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a)\} \tag{1.2}$$

Briefly, the state-value functions allow the agent to predict the future rewards. However, it is not preferred to wait until observing all the future rewards; as an alternative for that, the value functions can be defined recursively in terms of a **Bellman Equation** for the value of selecting the next action for the next state.

$$Q^{\pi}(s,a) = \sum_{s'} P^a_{ss'} \left[ R^a_{ss'} + \gamma \sum_{a'} \pi(s',a') Q(s',a') \right] \tag{1.3}$$

As the main goal is finding the best policy, the optimal policy $\pi^*$ should lead to the optimal value function Q*(s,a) with the maximum value or maximum reward.

$$\mathbf{Q}^*(\mathbf{s},\mathbf{a}) = \max_{\pi} Q^{\pi}(s,a) \tag{1.4}$$

$$\begin{aligned} Q^*(s,a) &= E\{r_{t+1} + \gamma max_{a'} Q^* s_{t+1}, a' | s_t = s, a_t = a)\} \\ &= \sum_{s'} P^a_{ss'} \left[ R_{\mathbf{ss'a}} + \gamma max_{a'} Q^*(s',a') \right] \end{aligned} \tag{1.5}$$

### 1.3.2 Q-Learning

Q-learning is the process of updating the Q-values for each state-action pairs with the Bellman Equation with the reward received after the action until it converges to optimal value function. During Q-learning, Q-values are stored in **Q-table** which has each possible state as rows and each possible actions as columns. In Q-table, the Q-values are initialized to zero at the first implementation. However, when the agent starts to explore the environment, it updates the Q-table with the new observations. Each cell of this table contains the reward estimated to be received with that state-action pair. An optimal Q-table contains the values that allows the RL agent to choose the best action with the highest return in any possible state.

As the agent does not have enough knowledge about the environment at first, it starts with the exploration process. During Q-learning, the agent uses the Q-value estimations to choose the optimal action and uses **ε-greedy policy** to decide. With this policy, agent might explore, or exploit based on the ε. The epsilon determines the probability of whether the agent is going to explore or exploit. The agent chooses the optimal action with the probability of 1-ε and otherwise it chooses a random action and explores the environment. The ε is set to 1 at the beginning and it slowly decreases towards 0 as the agent gains more knowledge.

During the exploration, the agent selects a random action from the possible actions. After a while, the agent starts to use the experience so the knowledge about the environment to choose the actions. During this exploitation process, the agent chooses the optimal action with maximum Q-values.

### 1.3.3 Deep Q-Networks

However, even though Q-learning is working well with basic environments, Q-tables are impractical for complex problems with higher state space. For complex environments, tabular Q-learning is infeasible.

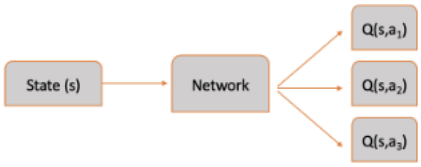



Figure 1.2. Deep Q-Learning

Mnih et al. (2015) solved this problem by inventing the **Deep Q-Networks (DQN)** algorithm. With DQN, they provided a solution by applying neural networks to estimate the optimal Q-function by using function approximators. By this way, DQN can learn policies also from high-dimensional sensory inputs directly using end-to-end RL. In that study, they gave input of stacked 4 frames to the network and tested the performance of the agent on 2600 Atari games. The performance of the algorithm was better than the previous algorithms on 43 of the games and it outperformed a human expert on 49 of the games (Mnih et al., 2015).

**DQN** algorithm is the combination of Reinforcement Learning and Artificial Neural Networks which provides the RL agent artificial intelligence capabilities. Mnih et al. (2015) demonstrated using DQN offer promising results by integrating **Convolutional Neural Networks**, **replay memory** and a **target network**. At the end, this integration provides human compatible behavioural results with the trained agents.

**Convolutional Neural Networks** takes an input of stacked frames and generates output of Q-values for each possible action.

Training with the DQN algorithm benefits from **experience replay** technique. With the experience replay, the agent's previous experiences for each state, action, reward, and the next states at a time point is stored in the **replay memory** as tuples.

$$Experience_{agent} = (s_t, a_t, r_{t+1}, s_{t+1}) \tag{1.6}$$

As a significant advantage, using a replay memory allow us to use a sample that not only use the last consecutive experiences. This technique called **replay buffer** or **experience replay** is a better way to benefit from previous experiences because it avoids the high correlation between consecutive samples by using a batch of random experiences to update the Q-network.

Lastly, **target network** is another integrated element of DQN to make the training more stable. The target network has the same architecture with the online network. It is used to calculate the target values. Also, the parameters of the target network are copied from the online network. However, the trick is, the parameters of the target network is updated less frequently. By this way, the target values are calculated with the same parameters until the next update which provides stability during training.

$$TargetValue = r + \lambda max_{a'} Q(s', a') \tag{1.7}$$

A Q-network can be trained by minimizing the **loss function** which is the difference between the **target values** and the **predicted values**.
There are different functions to minimize the loss. Huber Loss function is one of these functions which is related with this study.

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & for |a| \le \delta \\ \delta(|a| - \frac{1}{2}\delta) & otherwise \end{cases} \tag{1.8}$$

In this formula, a refers to the difference between the predicted and observed values. **Huber loss** or **SmoothL1** is a robust function to the outliers. With the formula above, it is more linear with larger values and quadratic with smaller values. Additionally, Huber Loss applies error clipping in range of (-1,1) and avoids the exploding gradients which might cause large changes in the weights of the neural network during the training and make the model highly unstable (Mnih et al., 2015).

## Methods

### 2.1 Navigation Task

For the mobility learning, we have used a navigation task. In this navigation task, the goal of the agent was reaching the target. There are different ways of goal identification such as ObjectGoal, PointGoal or AreaGoal (Anderson et al., 2018). In our project, we selected the navigation task of **PointGoal**. The agent was generated at the same location at the beginning of each episode and the task of the trained agent was to freely navigate in the environment towards the target by avoiding the wall and object collisions and by using the shortest path. The episodes were successfully terminated when the agent reached the target. However, for the unsuccessful episodes, the episodes were terminated when the agent reached the maximum number of steps. The

maximum number of steps that the agent can use was 500 steps in our experiments. By using this task as a baseline in all experiments, we have conducted different experiments by manipulating some parameters like the reward and the location of the target generated in different training sessions.

Even though we are working with computer agents to develop the models to provide optimized and efficient phosphene vision models, the important thing to remember is that these models will be used in real world settings. That's why it is important to design these experimental settings as much as realistic to have correlational results with real life situations.

For that reason, we used the behavioural navigation studies as a reference. When a rational person walks towards a location, s/he prefers the closest route and avoids the obstacles. In a recent study De Ruyter van Steveninck et al. (2022) conducted a study by using a real hallway setting to test sighted people with simulated phosphene vision with VR glasses. Then researchers compared the performances of the participants with the walking speed, avoidance strategy and their subjective reports (De Ruyter van Steveninck et al., 2022).

In our experimental setting, we also implemented the avoidance strategy. For the avoidance strategy, the aim was avoiding the obstacles while walking towards the target. With negative feedback, we aimed to teach the agent to minimize the collisions. Additionally, as rational people would prefer to use the shortest path to reach a location, we counted the number of steps agent used in one episode and manipulated the rewards to teach the agent to use minimum number of steps to reach the target location. Briefly, both the efficiency of the navigation and reaching the goal were important for the evaluations of the navigation task.

### 2.2 Realistic Virtual Environment

One of the limitations of the previous studies was the lack of a realistic environment. In this project, we aimed to use a realistic indoor environment.

#### 2.2.1 Unity

Unity is one of the most preferred platforms to create interactive and real-time 3D environments in many industries such as video games, movies, or architecture. It has some benefits like providing wide range of assets and a really good graphics that makes the motions more natural.

A major strength of Unity Platform is that Unity is also a Physics Engine. We benefit from multiple physics rules in this study.

For a realistic environment, we have developed and used **ArchVizPro Interior Vol.1** 3D Environment suitable for our project aim. The environment consists of high-quality furniture/props and HD textures. The environment includes 3 different furnished rooms that the agent can freely navigate. The props, player, or components of the scene or technically the GameObjects are physical entities in the environment and there are some components assigned for each physical entity such as Mesh Components, Renderers, Colliders.

Figure 2.1. Unity realistic virtual environment from different perspectives.

**Collisions**

The most important physics component we benefit was the **colliders**. In Unity virtual environment, there are colliders assigned to each object in the environment. Colliders are used to detect a collision between GameObjects in the environment. There are different types of colliders and physic materials of colliders. One type of colliders is the Box Colliders. The components of the objects allow us to use the geometrical information of the GameObjects like the size or bounds of the collider. If the collider has the same physical entities with the GameObject, then the colliders provide the geometrical information of the object. However, Unity allows the user to change the parameters of the components.

**Target Generation**

As we adopted the **PointGoal navigation task**, the agent navigates towards the target to reach a specific location. At the beginning of every episode a new target is generated at a random location. The agent is considered to reach the target when it gets in range of distance below the threshold of 'agent's body size x 2' which is used as a default threshold recommended by reviews (Anderson et al., 2018). To check if the agent reached the target, we have used the Physics information we got from Collider properties such as

the collider size of the agent and the physical position of the agent in the dimensions of -x, -y and -z.

In the current setup, we also implemented the **ObjectGoal navigation task**. For that case, we collect the information of the center of the objects and their sizes to reach a kind of mapping about the environment. By using this list, we are also able to use the objects in the environment as targets. The parameters provided with colliders such as the size, boundaries, coordinates, or the center of the objects are used while generating the target at a location not colliding with any other objects.

There were some critical points to be ensured during the target generation.

1. **No collision with other objects:**

- Agent - Object Collision

A significant aspect of generating the target was ensuring that when the agent reached the target, it is not going to collide with any other objects. To achieve that, rather than generating a target location, we have generated a **target object** with the collider size of the agent. In this way, when we assured that the generated target is not colliding with any other objects, we also assured the agent is not going to be colliding when it reached to target location or the center of the target object. Also, we disabled the physics components of the object such as the colliders to avoid the negative feedback caused by collisions with the target object.

- Target Object - Object Collision

As there are many pieces of furniture in the environment, we ensured that the target location was not created within the boundaries of any of the objects. Before generating a target location within the indoor environment, it is checked with the condition of "If the agent reaches the target, would it collide with any objects?" and kept generating another target location until the conditions are provided.

2. **Selecting Floors:**

As the target should be inside the environment, we needed to use the information of floors to draw the boundaries of the environment. However, the floors were also grouped with other objects in the environment. For this reason, there was a need for differentiating the floors from other GameObjects in the scene. To solve this problem, we have used the labels to reach the information of the floors specifically. In other words, we labelled the floors with 'floor' label to differentiate from other objects.

3. **Visibility**

Since, we are providing the frames as an input to the agent, the targets should not be visible in the frames provided to the agent. For this reason, we disabled the visibility of the target object by disabling the Mesh Renderer.

**Physics Rules**

When we used only the physic rules of the Unity Engine, the agent was able to go through the objects and walls. That was the case especially when we used a bigger step size. We wanted to keep it realistic, so changing the step size was not an optimal solution. That's why we needed to implement **additional physic rules** to the environment. After this implementation, when the agent was collided with an object, the agent moved back to the location before the last step. However, the negative feedback for the collision was still provided. That was necessary because we aim to teach the agent minimizing the collisions and also want to keep the environment with realistic conditions.

**Algorithm 1:** Double-DQN Algorithm with experience replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights $\theta$
Initialize Target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
**foreach** *episode* **do**
  Initialize sequence $s_1 = x_1$ and pre-processed sequence $\phi_1 = \phi(s_1)$
  **foreach** *step of episode* **do**
    With probability $\epsilon$ select a random action
    Otherwise select an $a_t = argmax_a Q(\phi(s_t), a; \theta)$
    Take action $a_t$ and observe reward $r_t$ and frame $x_{t+1}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and pre-process $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t_1}$ in D
    Sample random minibatch of transitions $\phi_j, a_j, r_j, \phi_{J+1}$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j \gamma max'_a \hat{Q}(\phi_{j+1}, a'; \theta^{-} & otherwise \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters $\theta$
    Every C steps reset $\hat{Q} = Q$
  **end foreach**
**end foreach**

### 2.3 Learning Algorithm

Studies demonstrated that the DQN algorithm has a limitation of overestimation (Hasselt et al., 2016). DQN algorithm combines a deep neural network with Q-learning. For more detailed explanation you can refer to the part explained Q-learning and DQN. However, in summary, overestimation problem is mainly the result of Q-Learning which uses argmax () function which basically always chooses the maximum value to estimate the values. Additionally, it uses the same weights to choose the action and evaluate the action which is a really optimistic approach. Another algorithm suggested that can be used to avoid these limitations is the **Double DQN algorithm**. With D-DQN, rather than this overoptimistic approach, we can use a more realistic evaluation by using different networks to choose and evaluate an action chosen.

Hasselt et al. (2016) analysed the performance and the estimated values of DQN and D-DQN with various Atari games and they demonstrated that the value estimations of DQN was significantly higher than the true values. Researchers defined the true values as *Q**(s,a) = V*(s) in their evaluation. Their results demonstrated that, the overestimation problem was also affecting the performance in a negative way (Hasselt et al., 2016).

Similar to DQN, the D-DQN algorithm chooses the optimal action for the current state by using the online network. However, to fairly evaluate this action, it uses another network with different weights. By this way, it reduces the overestimation and gives closer results with real values. In line with this, it results in better performance.

$$Y_t^{Q} = \mathbf{R_{t+1}} + \gamma Q(S_{t+1}, argmax_a Q(S_{t+1}, a; \theta_t); \theta_t)$$
$$Y_t^{DoubleQ} \equiv \mathbf{R_{t+1}} + \gamma Q(S_{t+1}, argmax_a Q(S_{t+1}, a; \theta_t); \theta_t^{'}) \quad (2.1)$$

By relying on the recent evaluations of DQN and Double-DQN, we preferred to use the Double-DQN algorithm. With the D-DQN algorithm, we chose the actions for the current state by using the online network. Then, we have used the target network to fairly evaluate the estimated values with different set of weights.

## 2.4 Vision Processing

The size of the frames returned from the Unity environment was 128x128. These RGB frames converted into gray-scale images. Then we applied different image pre-processing techniques in different experiments to compare their effect on performance of the Deep RL agent.

### Phosphene Vision with Canny Edge Detection

Canny edge detection is a commonly used algorithm to detect edges. By using Canny Edge Detection algorithm on gray-scaled images, we detected the edges in the image. We have used **cv2** package to implement Canny edge detection. The algorithm uses 2 threshold parameters to generate clear edges as an upper and lower threshold. The algorithm smooths the image and decides to the edges by checking upper and lower threshold for all pixels. Any pixels with higher intensity than the maximum threshold is considered as sure-edge pixels. If there are pixels connected to sure-edge pixels with intensity between maximum threshold and minimum threshold, they are also accepted as a part of the edges. However, the pixels that has lower intensity then the minimum threshold and pixels that has higher intensity than the minimum threshold but not connected to a sure-edge pixel are discarded. By this way, the algorithm specifies the edges and creates a canny edge mask. In theory, it is suggested that the canny algorithm satisfies some criteria such as low error rate, good localization and minimal response which help to detect only the real edges and only one detection for per edge.

We have implemented the maximum threshold as 70 and minimum threshold as 35 in our experimental settings. Later, we have fed these images into phosphene simulator. Phosphene simulator converts the image into simulated phosphene vision. The phosphene resolution applied in our experimental settings were 30.

Figure 2.2. An example of input image. From left to right 1) Original Image. 2) The Object Contour Segmentation Image. 3) Canny Edge Detection Image. 4) Phosphene Simulator Image.

### Phosphene Vision with Object Contour Segmentation and Canny Edge Detection

Even though Canny Edge Detection is a commonly used and efficient algorithm, this environment was too complex and realistic. It contained so many objects, textures, realistic elements like lights or shadows. That's why we also implemented another algorithm to test the efficiency of more complex image pre-processing algorithms for complex environments.

Rather than using gray-scaled images to create Canny masks, we have used Unity's Object Contour segmentation algorithm. This algorithm uses the Renderer Unity components to select each object in the scene and segment them with different colours. This algorithm basically selects object by drawing a bounding box around them. As a next step, we have used these images to create Canny edge mask and then fed into phosphene simulator with the same procedure mentioned above.

Even though the computational load of this process was higher, the images generated with only Canny Edge Detection algorithm were noisier. By this additional pre-processing step, we created an input with less noise. We believe this is important because as mentioned before, because of the biological limitations, we need maximum information with lower resolution.

Figure 2.3. An example conversion of input image. From left to right 1) Original Image. 2) The Object Contour Segmentation Image. 3) Canny Edge Detection Image. 4) Phosphene Simulator Image.

## 2.5 Neural Network

The convolutional neural network is composed of 3 convolutional layers followed by ReLU activation function and batch normalization. Additionally, the model has a fully connected Linear output layer that provides 3 outputs as there are 3 possible actions the agent can choose. The outputs are basically the Q-values for each possible action

which provides a comparison between actions to judge the values then selection of the action based on this comparison.

### 2.6 Input to the Neural Network

For the first experiments, we have used frame stacking technique by stacking last 4 frames. The stacked and pre-processed frames formed the **states** as inputs.

However, in some of the experimental settings we added another frame to the state to give an additional information of 'distance to target'. Then, we gave this input to the convolutional neural network.

To add the information of 'distance to target', we generated an array with the size of the frames; all pixel values were filled with the information of 'distance to target'. As pixels can only have a value between 0-255, we manipulated the information of distance to target. We multiplied the distance to target information by 10 then rounded it to the closest integer. We took this information via PyClient which provides the communication between Unity and Python. Then we normalized the distance with 255 which is the maximum value a pixel can get. By this way, we combined the additional distance knowledge with states.

### 2.7 Action Selection

The agent was able to move forward only with forward step. However, the agent was also able to rotate 90° to the right and left. During training, agent chose the action with the ε-greedy strategy as explained above. However, during the validation session, agent chose the action with maximum value.

### 2.8 Training Procedure

#### 2.8.1 Training Duration

The trainings lasted 1000 episodes in general. However, in some cases there have been some changes because of different reasons which will be explained later in detail in related parts. The maximum number of steps that agent can act was 500 in all the experiments. Each episode terminated when the agent reached the target or when the agent reached the limit of maximum steps.

#### 2.8.2 Update of target network

As mentioned before, Double-DQN algorithm also use a target network. In the first experiment, the target network was updated with the parameters of the policy network every 10 episodes. However, to stabilize the model training, we have changed it and updated the target network every 100 episodes in later experiments.

#### 2.8.3 Training Parameters

As a part of the D-DQN, we have used replay memory component to store the experiences of the agent. In all experimental settings, the memory capacity was 12000. The batch size we have used to sample was 128.

To update the weights of the network at each iteration, we have used Adaptive Momentum or Adam optimizer. Some advantages of using the Adam optimizer are the minimum memory requirement and its efficiency also with complex problems (Yi et al., 2020). As mentioned before, to reach improved performance of RL agents, we want to minimize the prediction error. Adam optimizer uses an adaptive learning rate. We used 0.0001 as the initial learning rate. The maximum optimization step in a training session was 1e6.

#### 2.8.4 Target

In different experimental settings, we have used different strategies to generate the targets.

1. **Target generated at a random location in the environment**

   In some of the environments, the target object was generated at a random location inside the environment.

2. **Target generated at a further location as training progresses**

   During the experimental settings, the results demonstrated that the capacity of the RL agent to reach the target differs with the initial distance to target. As an alternative technique, we manipulated the area that the target object can be generated in the environment. In these setups, for the first 500 episodes the area that target can be generated increased every 100 episodes. At the end of the first 500 episodes, the area that target object generated reached the same size with the room the agent generated. After 500 episodes, the area target object can be generated was like the first strategy, so it was generated at a random location in one of the 3 rooms. In these experimental settings, the training lasted 1000 episodes.

### 2.8.5 Distance Calculations

In different experimental settings, we have used different types of distance calculations.[1]

**Distance approached to the target**

For the purposes of statistical analysis, we have calculated the distance approached to the target at the beginning of each episode and at each step.

$$\text{Distance approached to the target} = \text{Distance to the target after step} - \text{Distance to the target before step} \quad (2.2)$$

We explored different approaches to calculate the distance.

1. **Euclidean Distance**

   Euclidean distance is basically the magnitude of the distance between two location points as a straight line.

2. **Shortest Distance**

   To calculate the shortest path between two locations, Unity provides a component of NavMeshPath. To calculate the distance, it uses the corner properties and calculates the distance between two corner points. As an advantage over the other distance calculation methods, the NavMeshComponent finds the shortest path between two locations by also considering the obstacles between two locations. The path was not an input to the RL agent, we just used it for statistical evaluation.

3. **Manhattan Distance**

   Even though we did not prefer to use in our experimental settings, the Manhattan distance type is also implemented. It is the sum of absolute distance between two coordinations.

### 2.8.6 Rewards

In different experimental settings, we have used different reward sets until the 4th experiment. But in general, the goals of the rewards were the same. For specific values, please refer to Table 3.1.

- **Collision Reward**

To achieve a better avoidance strategy, we have used negative reward for object and wall collisions.

- **Forward Step Reward**

The reward selection was more complicated to satisfy a successful navigation task. For the forward steps, we did not keep the reward constantly negative or positive. We have manipulated the reward with the distance approached to the target. More specifically, the agent got negative feedback when it moved further from the target, and it got a positive reward when it approached to the target. In the first experimental setting the reward was scaled by the distance change between the agent and the target with the current step. For the second experiment, we didn't use a reward for forward step because the rewards in this experimental setting were normalized and we only gave feedback when the agent reached the target or collided with an object. However, for the rest of the experiments, we kept the forward step reward fixed and gave +1 for the conditions agent moved towards the target and -1 for the forward steps getting further from the target. The aim behind these rewards was teaching the agent to walk towards the target location by avoiding steps going further from the target.

- **Rotation Reward**

We have used a negative reward for the rotations in all experimental settings other than the Experiment 2. As we evaluated the navigation task with also the efficiency of the travel, the agent was expected to reach the target in shortest time. For this aim, we gave negative feedback to the agent for losing a timestep. Only in the 2nd experimental setting, we have used a basic reward setting by just giving a positive reward for target reached conditions and a negative reward for collisions. So, there was no other rewards, including the rotation.

- **Target Reached Reward**

In all experimental settings, we gave positive feedback for the episodes that the agent reached to the target location. Reaching the target was the main navigation task, so the positive reward was significantly higher than the other rewards.

[1] 1 Unity unit corresponds to 1 meter in reality.

• **Maximum Steps Reached Reward**

Only in the first experiment, we gave negative feedback if the episode terminated with the condition of the agent reached the maximum number of steps without reaching the target. The goal behind this reward was again teaching the agent to reach the target. However, using so many rewards with high values made the model unstable, so we changed our strategy.

#### 2.8.7 Agents

We can classify agents in different training sessions according to the image pre-processing technique we have used or types of models we implemented.

1. **Double-DQN Agent and Random Agent**

These agents were mainly different with their action selection strategies. While the **Double-DQN Agent** chose the action with the output of the Q-Network (Please refer to Algorithm 1 for details.), the random agent chose the actions randomly. Then we have used these different model behaviours to see the effect of training.

2. **Sighted-Agent, Canny-Agent, and Segmentation-Agent**

We have used different image pre-processing techniques in different experimental settings. To compare the performances of the agents trained with different inputs, we have tried different image pre-processing techniques in different training sessions. The **Sighted-Agents** were the agents we have trained by using gray-scaled images as inputs. The **Canny-Agents** were the agents we have used only Canny Edge Detection algorithm to pre-process the input. Lastly, the **Segmentation-Agent** was the agent trained with the input pre-processed with both Object Contour Segmentation Algorithm and Canny Edge Detection Algorithm.

### 2.9 Model Evaluation

#### 2.9.1 Validation Session

After every 50 episodes, we have run a validation session with specific 5 different seeds. Because of the seeds, for each validation episode, the target location was the same for all the validation sessions. By this way, we were able to test the model with the same validation session every time.

During the training, in validation loops we have saved the models to test the performance of the agent with the policy network parameters and create videos to also observe the session visually. For this aim, we have saved different models. One model was the **best model** which was the model used in validation session with the highest average return. The second model was the **recent model**, which is the final version of the trained model. Lastly, as the best model was selected with the information of the maximum reward, we have saved a third model as target-reached model. **target-reached model** was selected based on the number of episodes that the agent reached the target during a validation session.

Lastly, the mean initial distance was the same for all the validation sessions because of the seeds. Also, even in cases we used different strategies for target generation, the validation session was the same for all the experiments; the target objects in the validation session always generated at a random location in the house environment.

## Experiments and Results

| Experiments | 1 | 2 | 3 | 4,5,6,7,8 |
|---|---|---|---|---|
| # of Episodes | 1000 | 400 | 405 | 1000 |
| Maximum # of Steps | 500 | | | |
| Forward Step Reward | ± distance approached to the target | 0 | ±1 | ±1 |
| Rotation Reward | -10 | 0 | -1 | -1 |
| Target Reached Reward | +90 | +1 | +50 | +50 |
| Maximum Steps Reached Reward | -90 | 0 | 0 | 0 |
| Wall/Object Collisions Reward | -30 | -1 | -2 | -2 |
| Gamma | 0.99 | | | |
| # of Episodes to Update Target Network | 10 | 100 | | |
| Distance Type | NavMeshPath | | | Euclidean |

Table 3.1. A table to summarize all the parameters used in experimental settings.

### 3.1 Input to Neural Network

In the first three experiments, the input of the Neural Network was the frames converted to phosphene. We stacked 4 pre-processed phosphene frames as states and fed the network with this input.

### 3.2 Double-DQN Canny-Agents

#### 3.2.1 Experiment 1

**Rewards** In this experimental setting, we gave positive feedback if the agent was moving towards the target or reaching the target and we gave negative feedback for the actions not moving towards the target, collisions, and

terminated episodes without reaching the target. For values of the rewards, please refer to Table 3.1. For a general explanation for the rewards, please refer to subsection 2.8.6.

**Target** At the beginning of each episode, a target was generated at a random location in the environment.

**Training Performance** We have assessed the performance by relying on different parameters such as the cumulative reward, total collisions, distance approached to the target, the initial distance at the beginning of the episode and the final distance to the target at the end of the episode etc.

During the training, the cumulative rewards for episodes were increasing. However, even though it was increasing, the reward was highly negative.

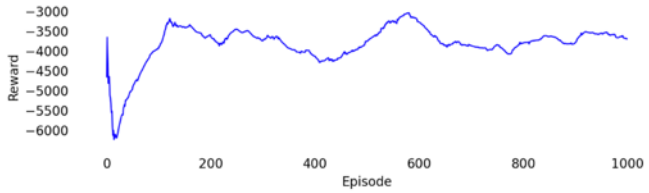



Figure 3.1. Episode-Reward Graph for Experiment 1.

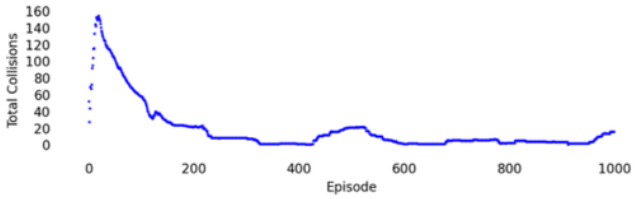



Figure 3.2. Episode-Total Collisions Graph for Experiment 1.

The Figure 3.2 demonstrates that, the agent was learning obstacle avoidance during the training session. This is a possible reason for increasing rewards.

However, as it can be seen from the Figure 3.3 for the percentage of actions chosen during the training session and the percentage of conditions for episode termination, the RL agent was avoiding the collisions by avoiding forward steps. As the agent was only able to move forward with the forward step, it was colliding only when it chose the forward step. When the agent moved forward, the rewards for the collisions were remarkably higher than the rotation reward. We believe, that's why the agent was more prone to rotation rather than a forward step. The action selection of the Canny-Agent almost looks like a Random-Agent that chooses the actions randomly at each step. Also, during the training session, the Deep RL agent almost did not reach the target location. The Figure 3.3 demonstrates that almost all the episodes terminated unsuccessfully when the agent reached the maximum number of steps.

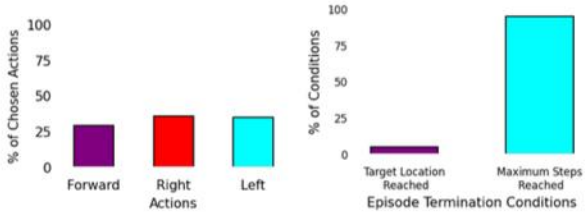



Figure 3.3. A) Percentage of actions chosen during the training for Experiment 1. B) Percentage of termination conditions in a training session for Experiment 1.

Also, the positive reward scaled with distance approached was not compensating the negative reward of the collisions. For this reason, the cumulative rewards were highly negative, and the agent was not colliding with the objects because it was learning to rotate around rather than moving towards the target.

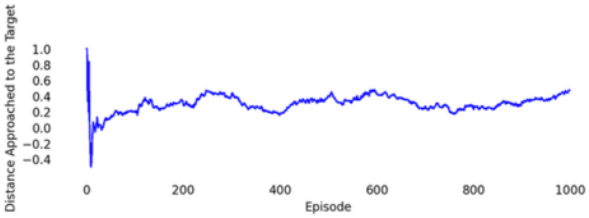



Figure 3.4. Cumulative distance approached to the target graph for episodes in a training session for Experiment 1.

The Figure 3.4 also supports that the agent did not learn to reach the target during the training as it was almost the same in all episodes.

**Validation Performance** Even though the increase of the reward was more stable during training, for the validation sessions it was unstable. The Figure 3.5 demonstrated that there was not a development in the learning across episodes.

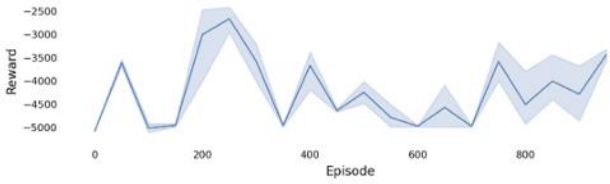



Figure 3.5. Rewards for the validation session for Experiment 1.

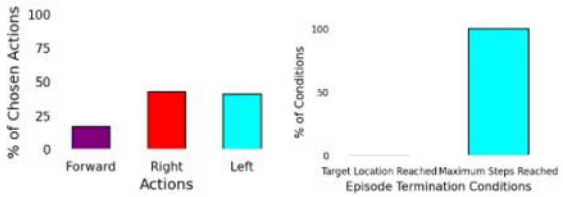



Figure 3.6. A) Graph for percentage of actions chosen during validation sessions. B) Graph for percentage of episode termination conditions as target reached or maximum number of steps reached.

Also, during the validation sessions, the RL agent mostly preferred to rotate around itself. As the agent was not learning to move towards the target, it was not able to reach the target in any of the validation sessions.

In general, the analysis shows that the RL agent did not learn to move towards the target, but it preferred to rotate around itself to avoid collisions and minimize the negative reward. In addition to the graphs demonstrating the model was not serving for our aim, the GIFs created with the models saved during the validation sessions also showed that the models were not successfully trained.

### 3.2.2 Experiment 2

To deal with the problems in the first experiment, we wanted an experimental setup to reach more stable results. For this reason, in this experiment, we have used normalized rewards.

**Rewards** In this experimental setup, we only used rewards for the conditions of target reached and collisions. By this way, we aimed to see the behaviour of the agent with a more simplistic design and normalized reward set. For values of the rewards, please refer to Table 3.1. For a general explanation for the rewards, please refer to subsection 2.8.6.

**Target** Similar to the first experiment, the target location generated at a random location in the environment.

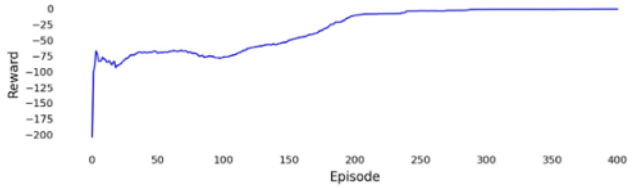



Figure 3.7. Episode-Reward Graph for Experiment 2.

**Training Performance** Figure 3.7 demonstrates that using normalized rewards also results in normalized cumulative reward. As the aim of this experiment was reaching more stable results, we reached the aim in 400 episodes. Similarly, the Figure 3.8 for total collisions showed that the agent learned to minimize the collisions in 400 episodes.

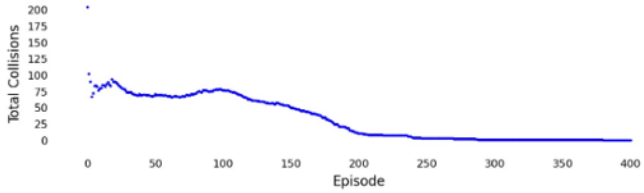



Figure 3.8. Episode-Total Collisions Graph for Experiment 2.

Interestingly, the percentage of the actions chosen during the training and the percentage of the conditions for the episode termination was the same with the first experiment (Please refer to Figure 3.3). However, in this experimental setting, only positive feedback agent got was reaching the target. In other words, there was no reward implemented for walking towards the target. In line with this, the increasing cumulative reward can be explained by avoidance strategy rather than reaching the target location. As also can be seen from the Figure 3.10, there was no development for the performance of navigation task, so the agent did not learn to move towards the target during this training session. As mentioned before, since there were no

implemented rewards other than reaching to target, it was expected that the agent was not learning this task.

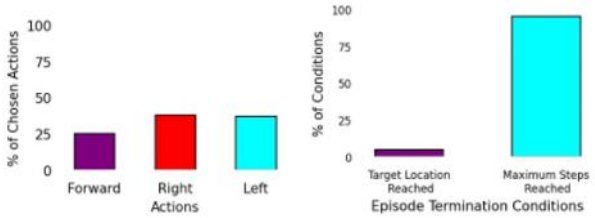



Figure 3.9. A) Percentage of actions chosen during the training for Experiment 2. B) Percentage of termination conditions in a training session for Experiment 2.

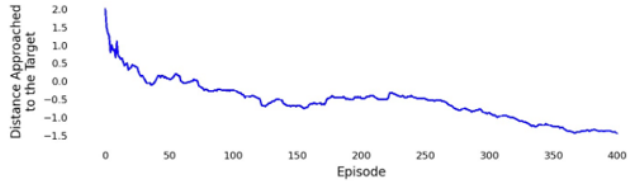



Figure 3.10. Cumulative distance approached to the target graph for episodes in a training session for Experiment 2.

In the first experimental setting, there were no improvements across episodes in terms of distance approached to the target location (Please refer to Figure 3.4). In this experimental setting, the agent was even getting further from the target location. For conclusion, even though the model was not working with the parameters specified in the Experiment 1, the reward implementation for the distance approached to the target had some positive implications in the results.

**Validation Performance** To evaluate the model and the training, we investigated the validation performance. Like the training session, the cumulative reward reached to 0 after a short training session. In contrast with the first experiment, in this experimental setting, the increase of the reward was stable in validation sessions. The reward graphs 3.11 and 3.12 shows that after 150 episodes, the mean reward was 0 for every seed or 5 different target locations.

This reward setting did not increase the percentage of forward steps remarkably. Also, there were no episodes that terminated due to target reached condition.

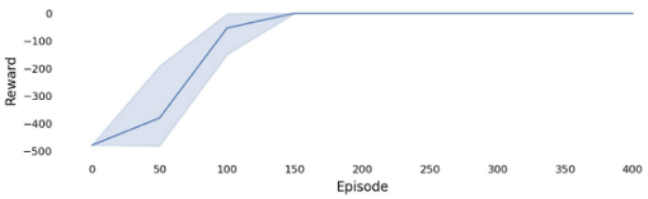



Figure 3.11. Rewards for the validation session for Experiment 2.

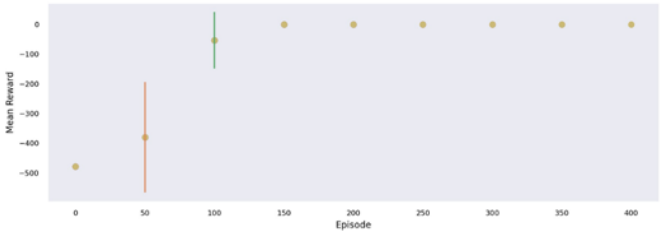



Figure 3.12. Graph for the rewards during validation sessions and the standard deviation.

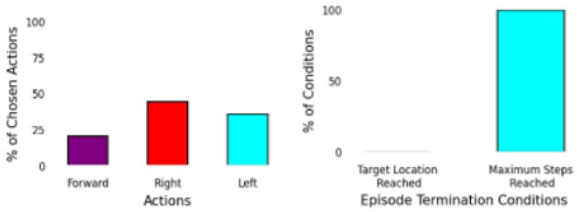



Figure 3.13. A) Graph for percentage of actions chosen during validation sessions. B) Graph for percentage of episode termination conditions as target reached or maximum number of steps reached.

The Figure 3.14 also proves that the agent did not learn to move towards the target. The average final distance to the target in the last validation sessions were even higher than the average initial distance.

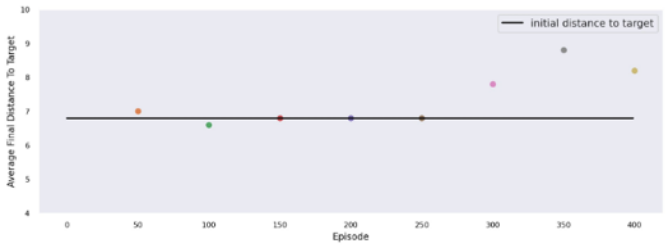



Figure 3.14. Graph for average initial and final distances to the target in validation sessions.

### 3.2.3 Experiment 3

The RL agent was able to learn the avoidance strategy with Experiment 2. However, in this experimental setting we have added rewards to teach the agent to move towards the target to reach the target.

During this experiment, there were some limitations that affected the results. The Unity crashed multiple times during the training. To deal with this problem, we uploaded the policy network and optimizer parameters and continued the training. However, loading the model affected the visualized graphs and training procedure. For this reason, in this experimental setting, the training duration is 405 episodes.

**Rewards** In this experimental setup, we aimed to implement the rewards to teach the navigation task of reaching the target. We added the negative step reward again for the rotations. Additionally, we gave a negative step reward if the agent was going further from the target. However, we gave a positive reward for walking towards the target.

To choose these rewards, we have made some tests and checked the performance for different reward settings. For example, in one of the experimental settings we scaled the positive reward for walking towards the target with the distance approached to the target. The results of that trial showed that the positive reward was too low. That's why we changed the positive reward fixed to +1 if it is getting closer. Also, we have tried a setting with +10 reward for the conditions the agent reached the target. This reward was not enough so we converted it to +50 in this setting. For values of the rewards, please refer to Table 3.1. For a general explanation for the rewards, please refer to subsection 2.8.6.

**Target** The target location generated at a random location in the environment.

**Training Performance** In this training session, the Figure 3.15 for the cumulative reward was demonstrating increase in reward like the reward graph (Figure 3.1) of the training session in Experiment 1, even though it was still negative. It was still normal because for every step not moving towards the target, we gave negative reward.

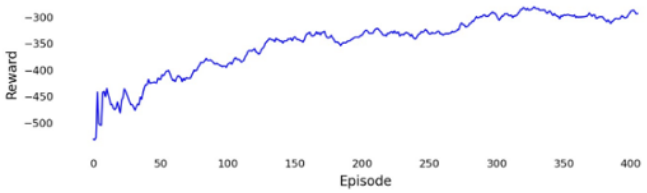
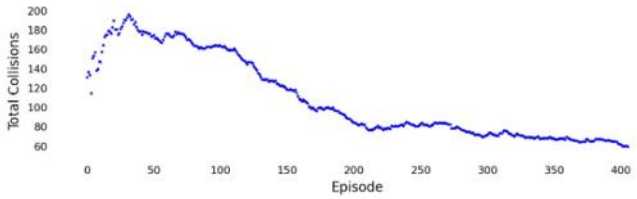



Figure 3.15. Episode-Reward Graph for Experiment 3.

The Figure 3.16 shows that for the avoidance strategy, the performance got better during the training session; agent learned to avoid collisions. Additionally, the Figure 3.17 demonstrates that with this reward setting the agent started to prefer forward step rather than rotating around. In this experimental setting, the RL agent was getting equally negative reward for every step it was not moving towards the target which made it prefer to move towards the target location which was the main navigation task. With this development in the action selection of the RL agent, the




episodes terminated due to target reached condition also increased.

Figure 3.16. Episode-Total Collisions Graph for Experiment 3.

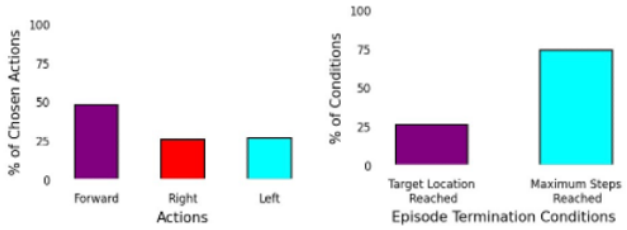



Figure 3.17. A) Percentage of actions chosen during the training for Experiment 3. B) Percentage of termination conditions in a training session for Experiment 3.

The Figure 3.18 for distance approached to the target also supports the claim that the agent started to learn to move towards the target within this experimental setting.

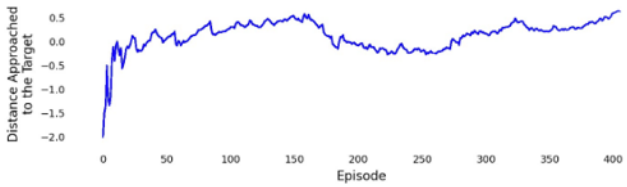



Figure 3.18. Cumulative distance approached to the target graph for episodes in a training session for Experiment 3.

**Validation Performance** The mean reward for the validation sessions was not as good as the mean reward in the validation session of the Experiment 2 (Please refer to Figure 3.1.1). Still, as the reward setting was completely different and more basic in that setting, the difference between the cumulative graphs of two experiments is expected.

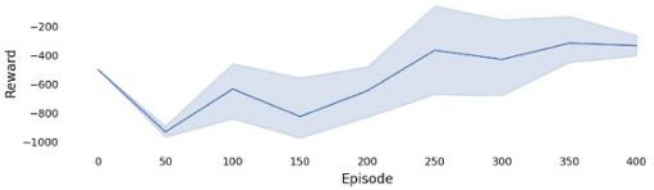



Figure 3.19. Rewards for the validation session for Experiment 3.

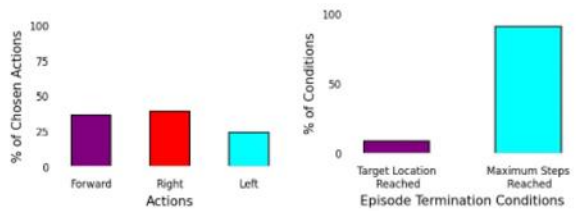



Figure 3.20. A) Graph for percentage of actions chosen during validation sessions. B) Graph for percentage of episode termination conditions as target reached or maximum number of steps reached.

This experimental setting was the first experimental setting we have seen the agent reached the target during the validation sessions. Similarly, the Figure 3.21 comparing the initial distance and the final distance showed that, the agent started to show development in navigation task. However, when we visualized the models to check the behaviour of the agent, the model was still not performing efficiently. For conclusion, this experimental setting showed that this training method was not enough for this complex problem and environment.

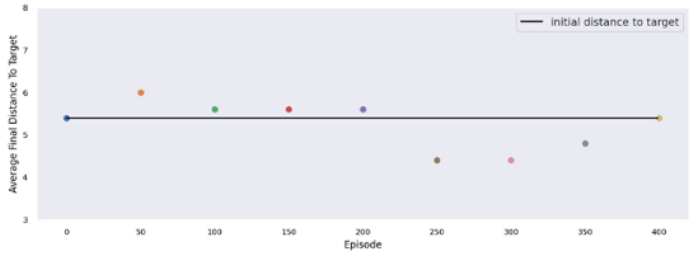



Figure 3.21. Graph for average initial and final distances to the target in validation sessions.

### 3.3 Random Agent

For the next experiments, we kept the reward setting the same. This provided us with a means to compare performance of agents with different visual and behavioural capacities under the same conditions. The random agent acts as a baseline in this sense.

#### 3.3.1 Experiment 4

**Rewards** Rewards kept the same. For values of the rewards, please refer to Table 3.1. For a general explanation for the rewards, please refer to subsection 2.8.6.

**Target** The target object was getting further during the training. There were some problems with the ShortestPath, in some of the steps it was not possible to find the shortest path. As we implemented the distance to target parameter as an input to the target, we have changed to distance type Euclidean.

**Training Performance** As this was the Random-Agent, the percentage of actions chosen was the same. However, agent was still able to reach the target while randomly moving during the training. As the target was generated in closer areas during the early episodes, it was easily reachable while exploring the environment randomly.

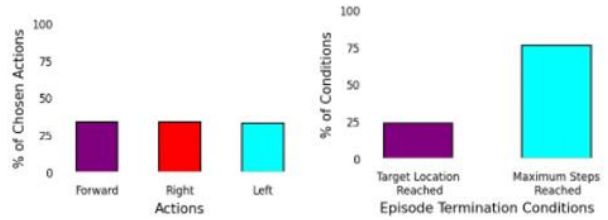



Figure 3.22. A) Percentage of actions chosen during the training for Experiment 4. B) Percentage of termination conditions in a training session for Experiment 4.

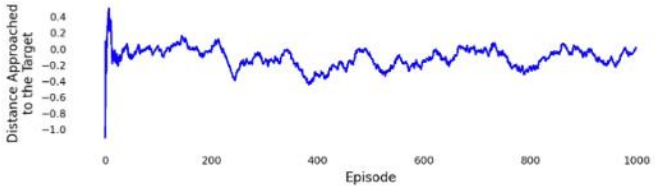



Figure 3.23. Cumulative distance approached to the target graph for episodes in a training session for Experiment 4.

Still, in average, the distance approached to the target across episodes were almost didn't change. In other words, the agent did not show development for the navigation task.

The most obvious difference was in the performance for avoidance strategy. The Figure 3.24 for total collisions across episodes was significantly different from the results of other experiments as expected. This was the only experimental setting the total collisions across episodes were increasing.

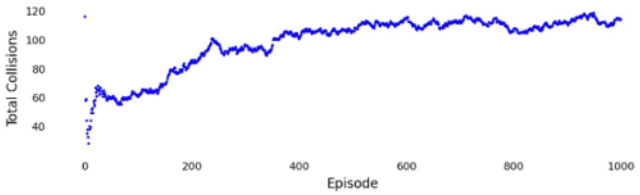



Figure 3.24. Episode-Total Collisions Graph for Experiment 4.

**Validation Performance** During the validation sessions, the model was not trained. Additionally, during the validation, agent did not reach to the target in any of the episodes. As also shown with the Figure 3.26, the initial distance and the final distance to the target was the same for all the validation sessions as the agent only chose to rotate.

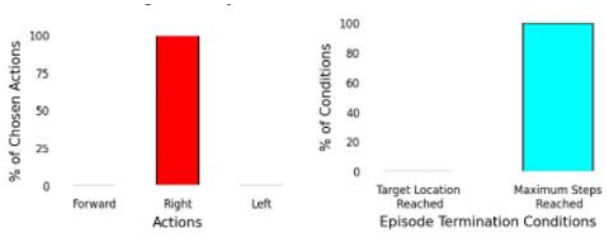



Figure 3.25. A) Graph for percentage of actions chosen during validation sessions. B) Graph for percentage of episode termination conditions as target reached or maximum number of steps reached.

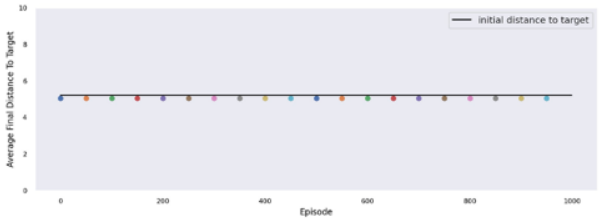



Figure 3.26. Graph for average initial and final distances to the target in validation sessions.

### 3.4 Input to Neural Network

After Experiment 3, we have changed the input given to the neural network. In the next experiments, the states given as input to the neural network had an additional information of distance to target. For more detailed information, please refer to the section on Input to the Neural Network (2.6)

### 3.5 Double-DQN Canny-Agents

#### 3.5.1 Experiment 5

**Rewards** As we achieved to increase the number of forward steps and decrease the collisions during the training and validation sessions, we have kept the rewards the same. For values of the rewards, please refer to Table 3.1. For a general explanation for the rewards, please refer to subsection 2.8.6.

**Target** Target generated at a random location in the environment.

**Training Performance** During the training, rewards increased until some point.

Then, the cumulative reward across episodes were more stable.

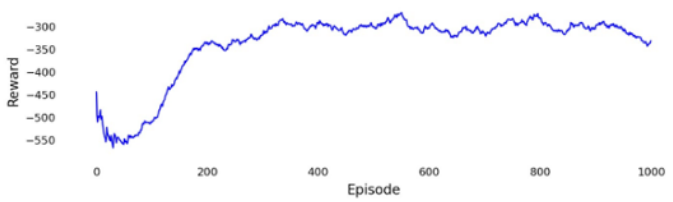



Figure 3.27. Episode-Reward Graph for Experiment 5.

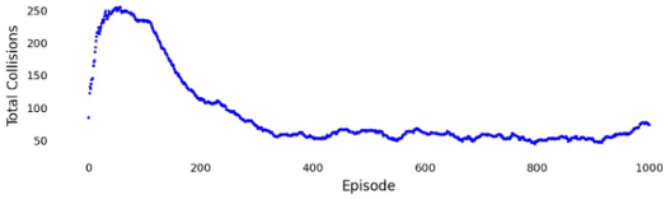



Figure 3.28. Episode-Total Collisions Graph for Experiment 5.

The Figure 3.28 for total collisions across episodes showed that, the agent was learning the avoidance strategy. Also, the Figure 3.28 for total collisions and the Figure 3.27 for the cumulative reward were almost reversed version of each other.

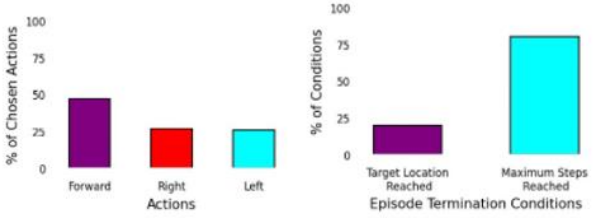
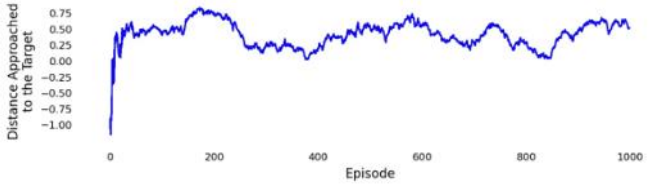



Figure 3.29. A) Percentage of actions chosen during the training for Experiment 5. B) Percentage of termination conditions in a training session for Experiment 5.

The percentage of the episodes terminated successfully and the percentage of the forward action in training session was almost the same with the training performance in Experiment 3 (Please refer to Figure 3.17).

However, the Experiment 3 lasted in 405 episodes so we cannot talk about the differences reliably.

Figure 3.30. Cumulative distance approached to the target graph for episodes in a training session for Experiment 4.




The total approach to the target across episodes was slightly better in this experimental setting than previous experiments. This demonstrates that giving another input to the network for the distance approached to the target was helping for a more efficient learning.

We also checked the effect of initial distance for the conditions that agent reached the target. The Figure 3.31 shows that, when the initial distance was higher than 3, the conditions agent reached the target was less.

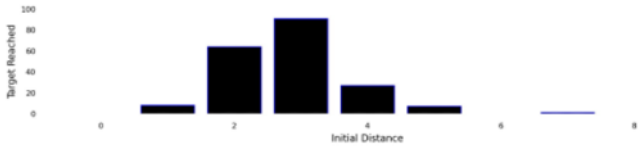



Figure 3.31. The graph for initial distances for the conditions agent reached the target during the training session in Experiment 4.

**Validation Performance** The mean reward in validation sessions increased until about 700 episodes. However, after that, it was again unstable. This shows that, the model was not developing efficiently during the training session for 1000 episodes.

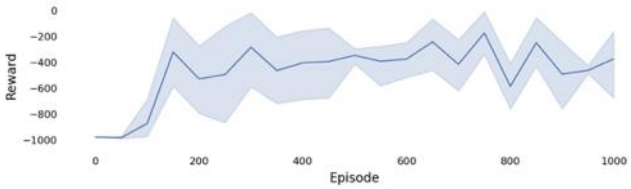



Figure 3.32. Rewards for the validation session for Experiment 5.

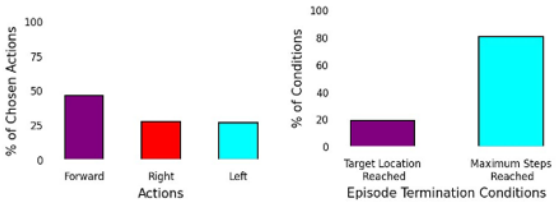



Figure 3.33. A) Graph for percentage of actions chosen during validation sessions. B) Graph for percentage of episode termination conditions as target reached or maximum number of steps reached.

The percentage of forward actions was remarkably higher than the validation sessions in previous experiments. Also, there was an increase in the percentage of episodes terminated successfully. However, the Figure 3.34 does not show a stable development across validation sessions when we compare the average initial distance and the final distances.

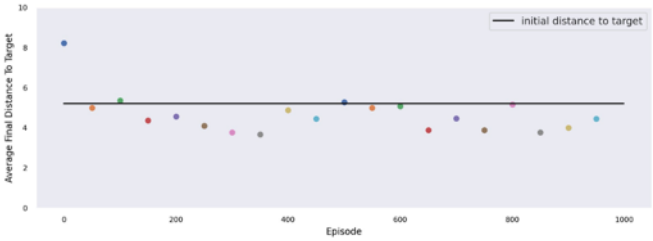



Figure 3.34. Graph for average initial and final distances to the target in validation sessions.

### 3.5.2 Experiment 6

**Rewards** Rewards kept the same. For values of the rewards, please refer to Table 3.1. For a general explanation for the rewards, please refer to subsection 2.8.6.

**Target** In training session, the initial distance to target location increased across episodes (For more detailed information, please refer to subsection 2.8.4).

**Training Performance** As the distance to target was increasing, the training session was not informative enough to compare.

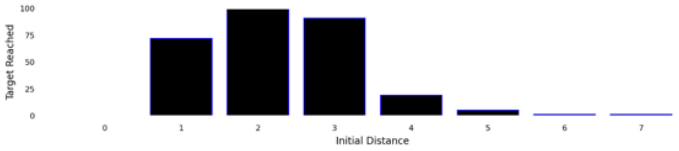



Figure 3.35. Graph for the initial distances and the number of episodes the agent reached the target.

However, as shown in the Figure 3.35 and also shown in the previous experimental settings (Please refer to Figure 3.31, the agent was able to learn to reach the target when the target generated at a closer location.

**Validation Performance** The conditions were the same for all the validation sessions. In other words, even though the target location was getting further during the training, for the validation session the target generated at a random location.

In this experimental setting, the graph for cumulative rewards across validation sessions (Figure 3.36) was better than the experimental settings that target generated at a random location in the environment in terms of cumulative reward in the last validation sessions.

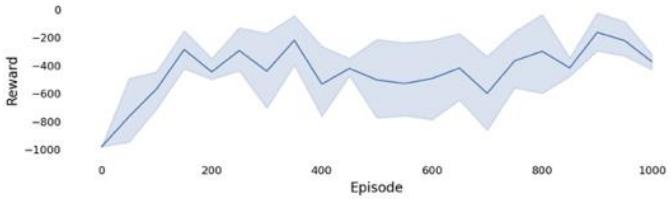



Figure 3.36. Rewards for the validation session for Experiment 6.

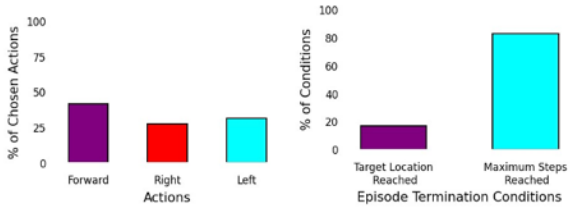



Figure 3.37. A) Graph for percentage of actions chosen during validation sessions. B) Graph for percentage of episode termination conditions as target reached or maximum number of steps reached.

The percentage of episodes terminated when the agent reached the target was the same with Experiment 5 (Please refer to Figure 3.33).

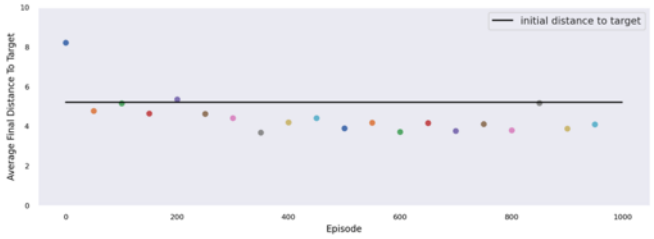



Figure 3.38. Graph for average initial and final distances to the target in validation sessions.

However, as we can see from the Figure 3.38, the average initial distance in our validation sessions was about 6. The Figure 3.35 demonstrates that, the conditions agent reached the target is remarkably lower for the conditions the target generated at a location with the initial distance higher than 3. As the average initial distance to target in validation sessions higher than 3, this is a possible reason for that the successful validation episodes are not increasing.

### 3.6 Double-DQN Segmentation-Agent

#### 3.6.1 Input

The input of the network was states as 4 frames stacked and an additional frame combined to give the information of distance approached to the target. The input was converted to phosphene. However, as the input created by only using Canny Edge Detector was still noisy, this time before we fed the input to phosphene simulator, we pre-processed the images with Object Contour Segmentation and Canny Edge Detector.

#### 3.6.2 Experiment 7

**Rewards** Rewards kept the same. For values of the rewards, please refer to Table 3.1. For a general explanation for the rewards, please refer to subsection 2.8.6.

**Target** As the observed performance of the agent and the graph of reward across validation sessions (Figure 3.36) was better, we kept generating the target at a further location during the training.

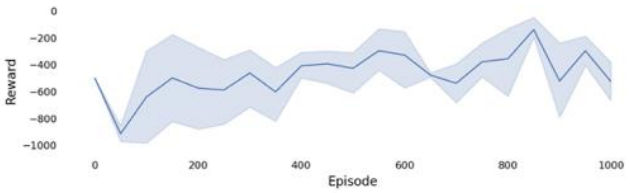



Figure 3.39. Rewards for the validation session for Experiment 7.

**Validation Performance** The reward graph for validation sessions was similar to the reward graph in Experiment 6 (Please refer to Figure 3.36). There was no continuous development in performance in terms of rewards in validation sessions.

With this image pre-processing technique, the images were less noisy. For this reason, we expected a better performance. However, in this experimental setting, the percentage of episodes terminated successfully was less than the episodes Canny-Agent reached the target (3.37).

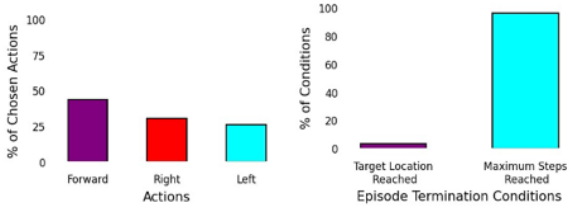



Figure 3.40. A) Graph for percentage of actions chosen during validation sessions. B) Graph for percentage of episode termination conditions as target reached or maximum number of steps reached.

As explained before during image pre-processing part, it is important to extract required visual cues during simplification of the scene. In comparison to Canny-Agent, the episodes terminated with the condition reaching the target was decreased. In other words, even though the simulated phosphene images generated with only Canny Edge Detection algorithm were looking noisier, the evaluations demonstrated that the model was still better. When we used an additional image pre-processing technique, the visualized images were clearer. However, results show that we lost some part of the required information for the agent.

### 3.7 Double-DQN Sighted-Agent

#### 3.7.1 Input

The input of the network was states as 4 frames stacked and an additional frame combined to give the information of distance approached to the target. The input was gray-scaled images.

#### 3.7.2 Experiment 8

**Rewards** Rewards kept the same. For values of the rewards, please refer to Table 3.1. For a general explanation for the rewards, please refer to subsection 2.8.6.

**Target** During the training session, the initial distance between the agent and the target increased.

**Training Performance** During the training session, agent was able to reach the target in almost half of the episodes.

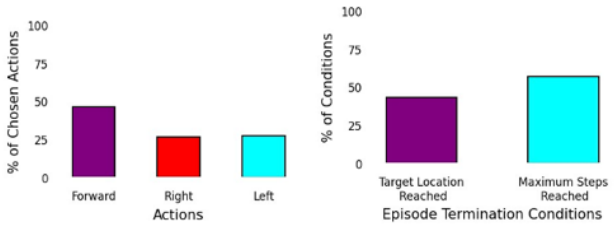



Figure 3.41. A) Percentage of actions chosen during the training for Experiment 8. B) Percentage of termination conditions in a training session for Experiment 8.

**Validation Performance** As the Sighted-Agent was getting gray-scaled input without a pre-processing step, we expected a better performance than other experimental settings. Unfortunately, the reward graph (3.42) for the reward was not better as expected.

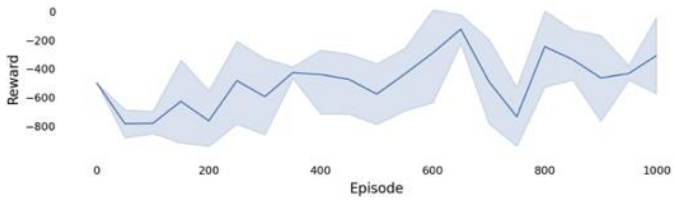



Figure 3.42. Rewards for the validation session for Experiment 8.

Additionally, the percentage of episodes terminated successfully in the validation sessions (3.43) was not as high as the training sessions (3.41). In line with this, also the Figure 3.44 does not show a stable development for the navigation task.

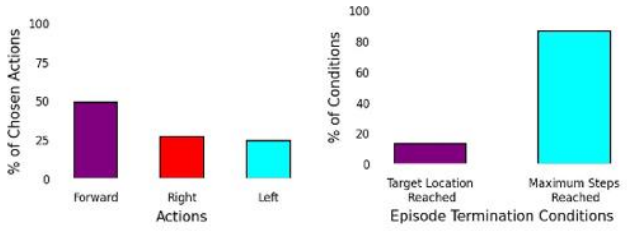



Figure 3.43. A) Graph for percentage of actions chosen during validation sessions. B) Graph for percentage of episode termination conditions as target reached or maximum number of steps reached.

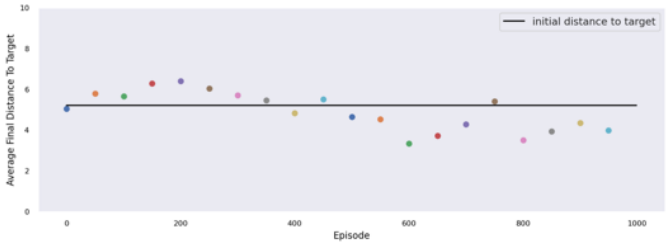



Figure 3.44. Graph for average initial and final distances to the target in validation sessions.

## Discussion and Conclusion

We have tried multiple experimental settings to develop a model to reach successful performances in the navigation task and avoidance strategy. We have evaluated the performances of the models with different experimental settings to reach the best model.

For target generation strategy, when we compare the performances of two Double-DQN Canny-Agents, the performance of the RL agent was better in terms of the cumulative reward in last validation sessions. For this reason, for the next experiments, we used this target generation method.

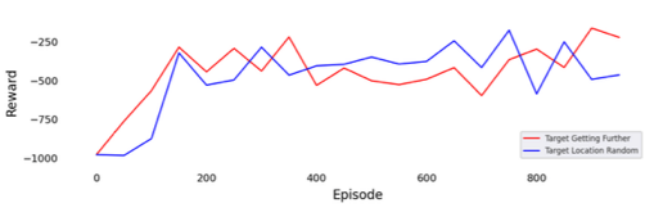



Figure 4.1. Validation Reward-Episode graph of Double-DQN Agents with different target generation strategies

In Experiments 6,7 and 8, all the conditions other than the visual input was the same. That's why we mainly compared the effect of different visual inputs by comparing these agents' performances. The Figure 4.2 of cumulative reward in validation sessions demonstrated that, for all the trained RL-Agents the performance was better than the performance of the Random-Agent. The performance of the Canny Agent was slightly better than the other agents. However, for general evaluation with rewards, the differences between trained models were not remarkably different.

The environment was highly complex and there were situations that when the agent walked into narrow areas, it could not get out of those areas. This caused to increase of the collisions in some cases.

For the comparison of total collisions of the agents with different visual inputs, the segmentation agent's performance for avoidance strategy seems the best. However, the difference is really low and in general all the models in different experimental settings showed development for obstacle avoidance.

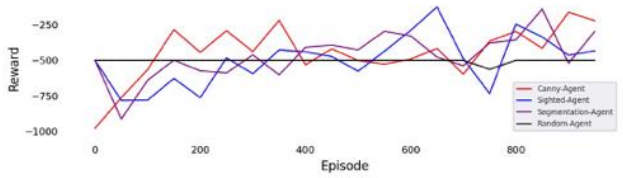



Figure 4.2. Validation Reward-Episode graph for models trained with different inputs


Figure 4.3. Distribution of the total collisions in final validation session and the best validation session in terms of rewards of Canny-Agent, Segmentation-Agent, and the Sighted Agent.

As mentioned in detail before, the best model saved in terms of the highest cumulative reward in validation session. Still, as the agent got positive reward for each step moving towards the target, the final distance to target and the cumulative reward are correlated. However, the negative reward for the collisions was double of the positive reward for distance moving towards the target. That's why, we cannot consider the best model also the best model for the navigation task. The Figure 4.5 demonstrates that, the recent model of the Segmentation-Agent was better than other models.

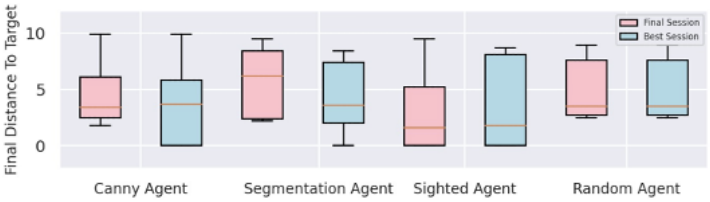



Figure 4.4. Distribution of the final distance to the target in final validation session and the best validation session in terms of rewards.

Still, when we consider the percentage of the episodes that the agent reached the target during the validation sessions, Canny-Agent was the best model. The percentage of episodes that the Segmentation-Agent reached the target when we evaluate all the validation sessions was less than the Sighted-Agent and the Canny-Agent.

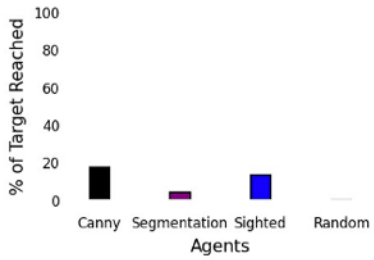



Figure 4.5. Distribution of the final distance to the target in final validation session and the best validation session in terms of rewards.

Even though the models are not working efficiently for the aim of reaching the target, the evaluations still demonstrate development in different experimental settings. Especially, for the avoidance strategy, the performance of the agents was developed. Additionally, the comparisons with the performance of the Random-Agent shows remarkable differences in terms of both the navigation task and avoidance strategy.

There are also some implications for using different image pre-processing techniques. As explained in detail in the image pre-processing section, there is a trade-o between simplification and the informativity of the scene. Our results for the Segmentation-Agent demonstrated that, even though the visualized scenes were better for our observation, it might not be informative enough for the RL agent. This should be investigated further in different experimental

settings. A visualization of simulated phosphene visions with different image pre-processing techniques can be found in the Figure 4.6.

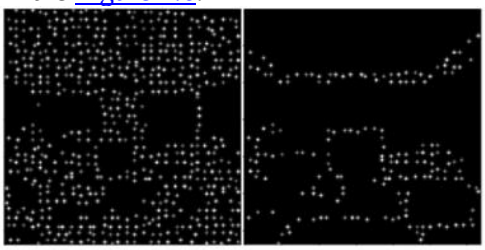


Figure 4.6. A) Input with Canny Edge Detection fed into phosphene simulator. B) Input with Object Contour Segmentation and Canny Edge Detection fed into phosphene simulator.

## Limitations and Future Work

There were some limitations in this project. As mentioned briefly before, Unity crashed multiple times during some of the training sessions. Even though we tried to load the policy network and the optimizer parameters as suggested in Pytorch Website, we were not able to continue without causing a difference in the model performance. That's why we needed to evaluate the model trained before the loaded data. That caused limitations of time and comparability of data.

Another limitation for Unity part was the distance calculations. Normally, we aimed to use the NavMeshPath to calculate the distance in the environment. That would be the best option because it generates the shortest path by considering the objects in the environment. In other words, the generated path is the path that the RL agent preferably used for the shortest path by avoiding collisions. However, the environment was highly complex and for some locations it was not possible to generate a path between two locations.

There were also problems caused by the methods we used. The target location was changing at the beginning of each episode. For memory replay, the model uses the experiences stored as ($s_t$, $a_t$ , $r_{t+1}$, $s_{t+1}$). However, when the target location changed, the return of the state-action pairs was also changing.

Additionally, as a limitation for the image pre-processing part, while generating the input for Segmentation-Agent, we have used Object-Contour Segmentation. With the Unity virtual environment, we were able to use an implemented ML algorithm that gives a unique colour to each object in the scene. With this algorithm, it uses the Renderer components of each object in the scene for segmentation. For this reason, the results for the Segmentation-Agent are generated under the assumption of perfect segmentation. There are other methods for Object Contour Segmentation. However, still there are still gaps in comparison to human vision (Yang et al., 2022). This raises the question "Is this really applicable in real-life scenarios?".

Still, this project provides a baseline for more realistic and complex future studies. Even though we have tried multiple experimental settings, we were just able to provide some developments rather than a working model for the navigation task. Future studies might benefit from different sets of parameters related with developments in the models to generate a working model. As this is a highly complex task, it is also important to test with more complex algorithms and network architectures in the future.

Additionally, in future, it is important to test the simulated vision in realistic settings with different contexts and tasks. As the optimized models are going to be used in real life, the models should be optimized by using real life experiences. To achieve this, there is a need to test real life contexts such as outdoor environments with dynamic stimuli. It is a scenario that we might benefit the most from the realistic virtual environments. Considering the risk and the ethical issues, we cannot test human participants in dynamic outdoor stimuli with every task such as crossing a road open to traffic. However, realistic virtual environments allow us to optimize the models without considering these kinds of limitations. In future, we should benefit from opportunities like that.

Lastly, in this project, we have tested Canny Edge Detection and Object Contour Segmentation for scene simplification. However, there are many other parameters should be tested for different contexts like depth. Future studies should test various contexts with different image pre-processing techniques to have more knowledge about required visual cues in different contexts.

**Acknowledgements**

Special thanks to Marcel van Gerven, Umut Güçlü, Burcu Küçükoğlu, Jaap de Ruyter van Steveninck, Richard van Wezel and Yağmur Güçlütürk for their great help and guidance. Additionally, I am grateful to Ertunç Erdil for his guidance and my adorable nephew Ege Luca Erdil for the energy and motivation he provided with his big smile every time I call during this process.

References


Anderson, P., Chang, A., Chaplot, D. S., Dosovitskiy, A., Gupta, S., Koltun, V., ... & Zamir, A. R. (2018). On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*.

Bollen, C. J., Van Wezel, R. J., Van Gerven, M. A., & Güçlütürk, Y. (2019). Emotion recognition with simulated phosphene vision. *Proceedings of the 2nd Workshop on Multimedia for Accessible Human Computer Interfaces - MAHCI '19*. https://doi.org/10.1145/3347319.3356836

Boyle, J., Maeder, A., & Boles, W. (2001). Static image simulation of electronic visual prostheses. *The Seventh Australian and New Zealand Intelligent Information Systems Conference, 2001*. https://doi.org/10.1109/anziis.2001.974055

Boyle, J., Maeder, A., & Boles, W. (2003). Scene specific imaging for bionic vision implants. *3rd International Symposium on Image and Signal Processing and Analysis, 2003. ISPA 2003. Proceedings of the*. https://doi.org/10.1109/ispa.2003.1296934

Boyle, J. R., Maeder, A. J., & Boles, W. W. (2002). Image enhancement for electronic visual prostheses. *Australasian Physics & Engineering Sciences in Medicine*, *25*(2), 81-86. https://doi.org/10.1007/bf03178470

Brindley, G. S., & Lewin, W. S. (1968). The sensations produced by electrical stimulation of the visual cortex. *The Journal of Physiology*, *196*(2), 479-493. https://doi.org/10.1113/jphysiol.1968.sp008519

Chen, X., Wang, F., Fernandez, E., & Roelfsema, P. R. (2020). Shape perception via a high-channel-count neuroprosthesis in monkey visual cortex. *Science*, *370*(6521), 1191-1196. https://doi.org/10.1126/science.abd7435

De Ruyter van Steveninck, J., Van Gestel, T., Koenders, P., Van der Ham, G., Vereecken, F., Güçlü, U., Van Gerven, M., Güçlütürk, Y., & Van Wezel, R. (2022). Real-world indoor mobility with simulated prosthetic vision: The benefits and feasibility of contour-based scene simplification at different phosphene resolutions. *Journal of Vision*, *22*(2), 1. https://doi.org/10.1167/jov.22.2.1

Dobelle, W. H., Quest, D. O., Antunes, J. L., Roberts, T. S., & Girvin, J. P. (1979). Artificial vision for the blind by electrical stimulation of the visual cortex. *Neurosurgery*, *5*(4). https://doi.org/10.1097/00006123-197910000-00021

Dowling, J. (2007). Mobility enhancement using simulated artificial human vision.

Fernández, E., Alfaro, A., & González-López, P. (2020). Toward long-term communication with the brain in the blind by Intracortical stimulation: Challenges and future prospects. *Frontiers in Neuroscience*, *14*. https://doi.org/10.3389/fnins.2020.00681

Garg, D., Jagannathan, S., Gupta, A., Garg, L., & Gupta, S. (2022). *Advanced computing: 11th International Conference, IACC 2021, Msida, Malta, December 18–19, 2021, revised selected papers*. Springer Nature.

Grover, L. L. (2017). Making eye health a population imperative: A vision for tomorrow—A report by the committee on public health approaches to reduce vision impairment and promote eye health. *Optometry and Vision Science*, *94*(4), 444-445. https://doi.org/10.1097/opx.0000000000001073

Hallum, L. E., Suaning, G. J., Taubman, D. S., & Lovell, N. H. (2005). Simulated prosthetic visual fixation, saccade, and smooth pursuit. *Vision Research*, *45*(6), 775-788. https://doi.org/10.1016/j.visres.2004.09.032

Van Hasselt, H., Guez, A., & Silver, D. (2016, March). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 30, No. 1).

Hayes, J. S., Yin, V. T., Piyathaisere, D., Weiland, J. D., Humayun, M. S., & Dagnelie, G. (2003). Visually guided performance of simple tasks using simulated prosthetic vision. *Artificial Organs*, *27*(11), 1016-1028. https://doi.org/10.1046/j.1525-1594.2003.07309.x

Jepkoech, J., Mugo, D. M., Kenduiywo, B. K., & Too, E. C. (2021). The effect of adaptive learning rate on the accuracy of neural networks. *International Journal of Advanced Computer Science and Applications*, *12*(8). https://doi.org/10.14569/ijacsa.2021.0120885

Krantz, J., Gokaslan, A., Batra, D., Lee, S., & Maksymets, O. (2021). Waypoint models for instruction-guided navigation in continuous environments. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. https://doi.org/10.1109/iccv48922.2021.01488

Kulhanek, J., Derner, E., De Bruin, T., & Babuska, R. (2019). Vision-based navigation using deep reinforcement learning. *2019 European Conference on Mobile Robots (ECMR)*. https://doi.org/10.1109/ecmr.2019.8870964

Lu, Y., Wang, J., Wu, H., Li, L., Cao, X., & Chai, X. (2013). Recognition of objects in simulated irregular phosphene maps for an Epiretinal prosthesis. *Artificial Organs*, *38*(2), E10-E20. https://doi.org/10.1111/aor.12174

McCarthy, C., Walker, J. G., Lieby, P., Scott, A., & Barnes, N. (2014). Mobility and low contrast trip

hazard avoidance using augmented depth. *Journal of Neural Engineering*, *12*(1), 016003. https://doi.org/10.1088/1741-2560/12/1/016003

Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A. J., Banino, A., ... & Hadsell, R. (2016). Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*.

Mnih, V., Kavukcuoglu, K., Silver, D. *et al.* Human-level control through deep reinforcement learning. *Nature* **518,** 529–533 (2015). https://doi.org/10.1038/nature14236

National Academies of Sciences; Engineering; and Medicine, Health and Medicine Division, Board on Population Health and Public Health Practice, & Committee on Public Health Approaches to Reduce Vision Impairment and Promote Eye Health. (2017). *Making eye health a population health imperative: Vision for tomorrow*. National Academies Press.

Sanchez-Garcia, M., Martinez-Cantin, R., & Guerrero, J. (2019). Indoor scenes understanding for visual prosthesis with fully Convolutional networks. *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. https://doi.org/10.5220/0007257602180225

Sanchez-Garcia, M., Martinez-Cantin, R., & Guerrero, J. J. (2020). Semantic and structural image segmentation for prosthetic vision. *PLOS ONE*, *15*(1), e0227677. https://doi.org/10.1371/journal.pone.0227677

Sanchez-Garcia, M., Martinez-Cantin, R., & Guerrero, J. J. (2018). Structural and object detection for phosphene images. *arXiv preprint arXiv:1809.09607*.

Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., Parikh, D., & Batra, D. (2019). Habitat: A platform for embodied AI research. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. https://doi.org/10.1109/iccv.2019.00943

Sehic, A., Guo, S., Cho, K., Corraya, R. M., Chen, D. F., & Utheim, T. P. (2016). Electrical stimulation as a means for improving vision. *The American Journal of Pathology*, *186*(11), 2783-2797. https://doi.org/10.1016/j.ajpath.2016.07.017

Shang, W., Wang, X., Srinivas, A., Rajeswaran, A., Gao, Y., Abbeel, P., & Laskin, M. (2021). Reinforcement learning with latent flow. *Advances in Neural Information Processing Systems*, *34*, 22171-22183.

Thompson, R. W., Barnett, G. D., Humayun, M. S., & Dagnelie, G. (2003). Facial recognition using simulated prosthetic Pixelized vision. *Investigative Opthalmology & Visual Science*, *44*(11), 5035. https://doi.org/10.1167/iovs.03-0341

*Updated vision atlas shows 1.1 billion people have vision loss*. (2021, April 15). The International Agency for the Prevention of Blindness. https://www.iapb.org/news/updated-vision-atlas-shows-1-1-billion-people-have-vision-loss/

Vergnieux, V., Macé, M. J., & Jouffrais, C. (2017). Simplification of visual rendering in simulated prosthetic vision facilitates navigation. *Artificial Organs*, *41*(9), 852-861. https://doi.org/10.1111/aor.12868

Yang, D., Peng, B., Al-Huda, Z., Malik, A., & Zhai, D. (2022). An overview of edge and object contour detection. *Neurocomputing*, *488*, 470-493. https://doi.org/10.1016/j.neucom.2022.02.079

Yi, D., Ahn, J., & Ji, S. (2020). An effective optimization method for machine learning based on ADAM. *Applied Sciences*, *10*(3), 1073. https://doi.org/10.3390/app10031073

Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., & Farhadi, A. (2017). Target-driven visual navigation in indoor scenes using deep reinforcement learning. *2017 IEEE International Conference on Robotics and Automation (ICRA)*. https://doi.org/10.1109/icra.2017.7989381