PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

Ministry of Higher Education and Scientific Research

UNIVERSITY OF MOHAMED KHIDER – BISKRA

Faculty of Exact Sciences, Science of Nature and Life

Computer Science Department

MASTER'S THESIS

Semi-mind controlled robots based on Reinforcement Learning for Indoor Application

Author:
Walid GUETTALA

Supervisor:
Dr. Ahmed TIBERMACINE

A thesis submitted in fulfillment of the requirements
for the degree of Master's Degree

in the

Artificial Intelligence
Computer Science Department

July 3, 2023

Declaration of Authorship

I, Walid GUETTALA, declare that this thesis titled, "Semi-mind controlled robots based on Reinforcement Learning for Indoor Application" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Abstract

Robots are increasingly used in indoor environments to carry out tasks that assist with everyday living. As these robots are expected to perform more complex tasks in the future, they will need to quickly acquire experience from multiple sources. One promising approach to achieve this is by incorporating interactive feedback. In this approach, a trainer advises the robot on which actions to take to speed up the learning process. However, deep reinforcement learning, which has been successfully applied to robotics, can be time-consuming to learn a task. This thesis proposes a solution to this problem by incorporating interactive feedback from brain signals to train robots using deep reinforcement learning for indoor applications, specifically maze navigation. We aimed to speed up the learning process in a human-robot scenario by using human emotion or attention feedback. To achieve this, we allowed the robot to learn new tasks with a dynamic policy network from human feedback. We also combined this feedback with other sensor feedback, such as LIDAR. We conducted different experiments to compare manual feedback, brain signal feedback, and no brain signal feedback using different Reinforcement Learning models, such as Deep Deterministic Policy Gradient (DDPG), Deep Q-Network, and twin-delayed deep deterministic policy gradient (TD3). Additionally, we studied different models for classifying emotions using Graph Neural Network models and traditional deep learning models and compared the results.

Keywords: Reinforcement learning, Human Feedback, Navigation, BCI, Emotion, P300.

Résumé

Les robots sont de plus en plus utilisés dans les environnements intérieurs pour effectuer des tâches qui facilitent la vie quotidienne. À mesure que ces robots sont censés accomplir des tâches plus complexes à l'avenir, ils devront acquérir rapidement de l'expérience à partir de sources multiples. Une approche prometteuse pour y parvenir consiste à intégrer des retours interactifs. Dans cette approche, un formateur conseille le robot sur les actions à entreprendre pour accélérer le processus d'apprentissage. Cependant, l'apprentissage par renforcement profond, qui a été appliqué avec succès à la robotique, peut prendre du temps pour apprendre une tâche. Cette thèse propose une solution à ce problème en intégrant des retours interactifs à partir des signaux cérébraux pour former des robots à l'aide de l'apprentissage par renforcement profond pour des applications en intérieur, en particulier la navigation dans des labyrinthes. Nous avons cherché à accélérer le processus d'apprentissage dans un scénario homme-robot en utilisant des retours émotionnels ou d'attention humaine. Pour cela, nous avons permis au robot d'apprendre de nouvelles tâches avec un réseau de politique dynamique à partir des retours humains. Nous avons également combiné ces retours avec d'autres retours de capteurs, tels que le LIDAR. Nous avons mené différentes expériences pour comparer les retours manuels, les retours de signaux cérébraux et l'absence de signaux cérébraux en utilisant différents modèles d'apprentissage par renforcement, tels que le Deep Deterministic Policy Gradient (DDPG), le Deep Q-Network et le twin-delayed deep deterministic policy gradient (TD3). De plus, nous avons étudié différents modèles pour classifier les émotions en utilisant des modèles de Graph Neural Network et des modèles d'apprentissage profond traditionnels, et nous avons comparé les résultats.

Mots-clés: Apprentissage par renforcement, rétroaction humaine, navigation, BCI, émotion, P300.

Contents

Acknowledgements

In the name of Allah, the Most Gracious and the Most Merciful, Alhamdulillah. All praises to Allah for the strengths and His blessings in completing this thesis.

First and foremost, I would like to express my heartfelt gratitude to my beloved parents, my mother, and my father, for their unwavering support in my education and throughout my life. I am also grateful to my brothers and sisters for their constant encouragement and presence.

I extend special appreciation to my supervisor, Dr. Ahmed Tibermacine, whose guidance and unwavering support have been instrumental in the successful completion of this research. His invaluable help, constructive comments, and insightful suggestions during the experimental and thesis works have significantly contributed to its success. I am also thankful to Imad Eddine Tibermacine for his support and extensive knowledge on the topic.

Furthermore, I would like to express my deep appreciation to Professor Laid Kahloul for his continuous advice and guidance throughout my years of study. His expertise and dedication have been invaluable. I am equally grateful to Professor Meftah Zouai for his direct and indirect support.

I would also like to thank Professor Abdelhamid Djeffal and Professor Okba Tibermacine for their consistent help whenever I needed it. I am indebted to all the other professors whose names I have not mentioned but who have played a significant role in my education at the University of Biskra.

My sincere acknowledgment also goes to all the technicians and office staff of the School of Computer Science for their cooperation and assistance.

I would like to express my gratitude to all my friends from the Department of Computer Science, as well as my dormitory friends, for their kindness and unwavering moral support during my studies. I am truly grateful for the cherished friendships and memories we have created.

List of Figures

List of Tables

General Introduction

0.1 General Context

Robotics is a vast and influential domain that has made a significant impact across various sectors, ranging from healthcare to industry, and has become an integral part of our everyday lives. The portrayal of advanced robots in fantasy movies exemplifies the fascination with the potential capabilities of robotic systems. In parallel with these advancements, the field of artificial intelligence (AI) has witnessed significant breakthroughs, including the development of sophisticated AI systems such as ChatGPT, which have revolutionized natural language processing tasks.

This thesis explores the intersection of robotics and AI, specifically focusing on the utilization of reinforcement learning and neural feedback to enable semi-mind controlled robots for indoor applications. The aim is to enhance the learning capabilities of robots by incorporating human guidance through neural feedback. One of the key aspects of this research involves training robots to navigate through randomly generated mazes by leveraging the principles of reinforcement learning.

The integration of reinforcement learning techniques and neural feedback has yielded impressive results, as evidenced by existing literature. Through the fusion of AI and robotics, robots have been able to learn and execute complex tasks efficiently. Moreover, this research contributes to the advancement of the robot industry by harnessing the power of simulation and reinforcement learning to reduce the costs associated with prototyping advanced robot systems.

The thesis focuses on two primary areas of investigation. Firstly, it aims to advance and optimize the learning capabilities of robots, making them more adaptable and efficient in various indoor environments. Secondly, it explores the integration of the human sensory system, analogous to the eyes and the brain, as a sensor for robots.

This integration aims to enhance the perceptual capabilities of robots by leveraging human motion feedback, which can be categorized into positive, neutral, and negative feedback signals.

In this research, the effectiveness of the proposed approach is demonstrated through experimental evaluations. Notably, the state-of-the-art models in the field have been surpassed, as exemplified by the achievement of outstanding results in the renowned benchmark named SEEDIV. Additionally, the thesis presents three main case studies within the realm of deep reinforcement learning. These studies involve optimizing reward calculation, comparing different sensor feedback systems (such as lidar, emotion, and hyperdimensional representation), and investigating the impact of human emotion on advancing the learning process of robots.

To facilitate these experiments, advanced tools such as Gazebo and ROS2 are employed, enabling the creation of highly realistic 3D physics simulations that simulate real-world environments. This approach significantly reduces the cost and complexity associated with building physical prototypes, making it more accessible to develop and test robot systems.

0.2 Problematic and Objectives

The field of robotics has witnessed significant advancements and has a profound impact on various domains such as healthcare, industry, and everyday human life. Additionally, the integration of advanced AI systems, including ChatGPT, has made remarkable strides in natural language processing tasks. This thesis addresses the following problematic and objectives:

- **Problematic**

  1. Incorporating human neural feedback into reinforcement learning for robot navigation in indoor applications.
  2. Enhancing the learning capabilities of robots by leveraging the integration of human emotion feedback and sensor feedback.

- **Objectives**

1. To develop a novel method for mapping and navigation of robots in indoor environments using reinforcement learning and neural feedback.
2. To investigate the impact of human emotion feedback as a form of neural feedback in the robot learning process.
3. To optimize the learning process of robots by combining sensor feedback, such as 2D LIDAR data, with human emotion feedback.
4. To compare and evaluate different deep reinforcement learning algorithms, including Deep Deterministic Policy Gradient (DDPG), Deep Q-Network (DQN), and twin-delayed deep deterministic policy gradient (TD3), in the context of robot navigation.
5. To explore different models for classifying human emotions, including Graph Neural Network models and traditional deep learning models, and analyze their effectiveness in improving the learning process.
6. To utilize simulation tools such as Gazebo and ROS2 to create realistic 3D physics simulations that emulate real-world environments and enable cost-effective prototyping for robot systems.
7. To surpass the state-of-the-art models in the SEEDIV benchmark, demonstrating the superiority of the proposed method.
8. To investigate the feasibility and effectiveness of utilizing the proposed method in more general tasks beyond robot navigation.
9. To explore the potential application of the developed method in multi-robot swarm systems, studying its impact on collective behaviors and coordination.

By addressing these objectives, this thesis aims to advance the field of robotics by introducing a novel approach that leverages human neural feedback for reinforcement learning in robot systems. Furthermore, it seeks to explore the wider implications and potential applications of this method, paving the way for future research and advancements in the field of human-robot interaction and AI-guided robotics.

0.3 Structure of the thesis

In this section, we provide an overview of the structure and contents of the thesis. The thesis is organized into the following chapters:

1. **Chapter 2: Reinforcement Learning**
   This chapter introduces the fundamentals of reinforcement learning (RL), including its definition, key concepts, and terminology. It explores different RL algorithms such as Monte Carlo method, Q-learning, and deep reinforcement learning (DRL). Additionally, it discusses reinforcement learning with human feedback (RLHM), its challenges, and applications.

2. **Chapter 3: Brain Computer Interface**
   In this chapter, the theoretical and technical background of Brain Computer Interfaces (BCIs) is presented. It covers the definition of BCI, lobes of the brain, EEG rhythms, event-related potentials, and hybrid BCIs. The technical background includes brain signal acquisition, artifacts removal and preprocessing, feature extraction and selection, classification, and control unit. The chapter also explores the applications of BCIs and discusses related works.

3. **Chapter 4: Contribution and Results**
   This chapter presents the proposed model that integrates reinforcement learning with human feedback and brain signal feedback for robot navigation. It explains the global system design and provides details on the components related to brain signal feedback and deep reinforcement learning. The chapter includes information about the experimental setup, dataset used, hyperparameters, simulation challenges in maze environments, and presents the results of EEG experiments and case studies using DRL. Finally, it concludes with the overall conclusions of the thesis and suggestions for future work.

Reinforcement Learning

1.1 Introduction

Reinforcement Learning (RL) has revolutionized the field of training intelligent agents to make optimal decisions in complex environments. This chapter provides an exploration of RL's fundamental concepts, advancements, and improvements, including Deep Reinforcement Learning (DRL). Despite its successes, RL faces challenges in uncertain and dynamic environments, necessitating a paradigm shift towards Reinforcement Learning with Human Feedback (RLHM) [1]. Section 1.2 delves into RLHM, examining its definition, key components, and various types of human feedback. The RLHM framework, Reinforcement Learning from Human Feedback, is introduced as a structured approach incorporating human guidance. The chapter also discusses different approaches to integrating human feedback, highlighting their potential benefits and limitations. Addressing challenges and showcasing diverse applications, from robotics to game playing, this chapter sets the stage for a comprehensive exploration of RLHM's potential.

1.2 Reinforcement Learning

1.2.1 Definition

In Reinforcement Learning, an agent is supposed to learn a specific behavior by trial and error. The agent interacts with its environment to collect experiences. [Fig. 1.1] illustrates the basic concept behind Reinforcement Learning. At each time step $t = 1, 2, 3, \ldots$, the agent is in a certain state $s_t \in S$ and takes one of the possible available actions $a_t \in A(s)$. The action changes the environment, and the agent ends up in a new state s_{t+1}. Furthermore, it receives a reward R_{t+1} from the environment that serves as feedback about how good it was to take action a_t in state s_t [2].

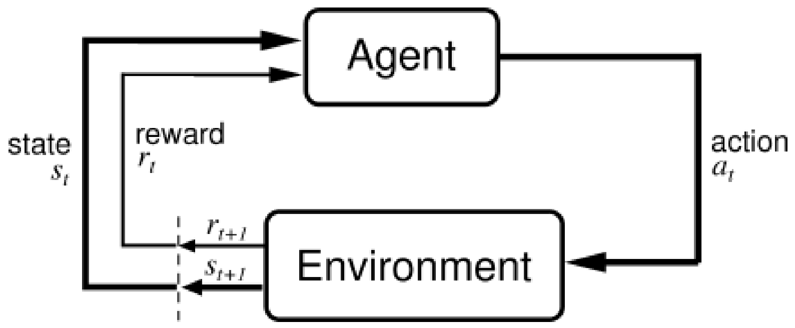


FIGURE 1.1: Agent-environment interaction loop [3]

1.2.2 Key Concepts and Terminology

Reinforcement learning (RL) encompasses a set of key concepts and terminology that form the foundation of RL algorithms and their application [Fig. 1.1].

- **Agent:** An agent in RL refers to an entity that interacts with the environment, making decisions and taking actions based on its observations and goals. The role of the agent involves perceiving the environment, selecting actions, and learning from the rewards received. RL encompasses various types of agents, such as deterministic and stochastic agents [4], which influence the learning process.

- **Environment:** The environment in RL is the external system or world in which the agent operates and interacts. It is represented using states that capture relevant information about the system at a given time. The dynamics of the environment and the transition model describe how the system evolves from one state to another based on the agent's actions.

- **Actions:** The action space in RL defines the set of possible actions that the agent can take in a given state. It can be discrete or continuous, and the choice of action selection policies, such as deterministic or stochastic policies [5], affects the agent's decision-making process.

- **Rewards:** Rewards in RL are numerical signals received by the agent from the environment, indicating the desirability of its actions. They can be immediate or delayed, and designing appropriate reward functions that align with the desired behavior of the agent is a critical consideration.

- **States:** States provide a complete description of the state of the world in reinforcement learning. They encapsulate all the relevant information necessary for the agent to perceive and interact with the environment. Effective state representations, typically as real-valued vectors or tensors, are crucial for capturing essential aspects of the environment while minimizing irrelevant information.

1.2.3 Model-Free vs Model-Based RL

Reinforcement learning (RL) algorithms can be broadly categorized as either model-free or model-based approaches. In model-free RL, the agent learns directly from experience without having access to or explicitly learning a model of the environment. This approach focuses on optimizing the agent's policy or action-value function, often through techniques like policy optimization [6] or Q-learning [7]. Model-free methods are easier to implement and tune but may require more data for effective learning.

On the other hand, model-based RL involves the agent having access to or learning a model of the environment. This model predicts state transitions and rewards, enabling the agent to plan ahead and make informed decisions. By considering a range of possible choices, the agent can distill the results into a learned policy. While model-based methods offer the potential for improved sample efficiency, learning an accurate model from experience poses challenges.

Model-free methods are more popular and extensively developed in the RL field compared to model-based methods [Fig. 1.2]. They forego the potential benefits of using a model but tend to be easier to implement and tune. Model-based methods, on the other hand, require the agent to learn a model and face the risk of exploiting biases in the learned model.

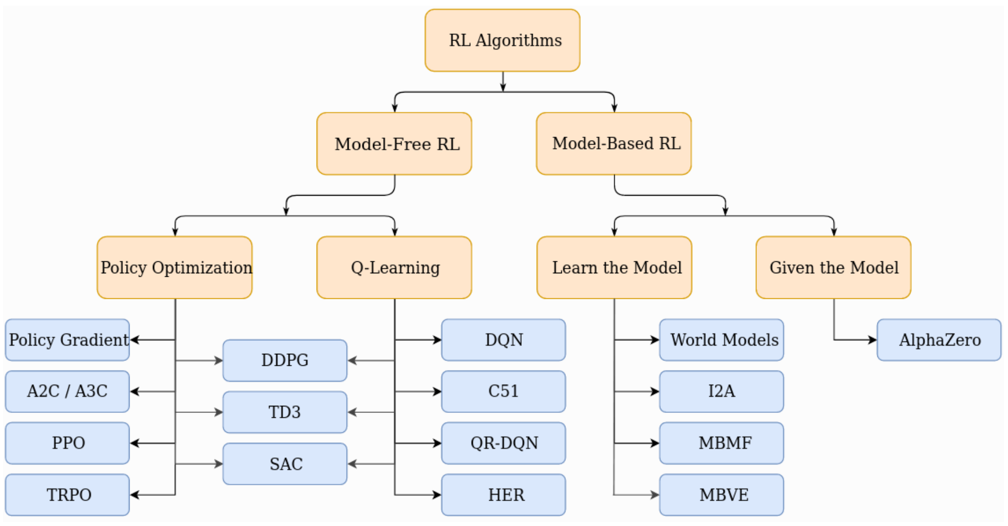


FIGURE 1.2: A Taxonomy of RL Algorithms [8]

1.2.4 Markov Decision Process

The Markov Decision Process (MDP) is based on the Markov Assumption, which assumes that the state and behavior of the environment at a given time step are independent of past agent-environment interactions (actions) prior to that time step [9]. If an RL-task satisfies the Markov Assumption, it can be represented as a five-tuple MDP (S, A, P, R, γ).

Here are the components of the MDP:

- Set of states, denoted as S, represents all possible states of the environment.
- Set of actions, denoted as A, represents the available actions in each state. $A(s)$ specifies the actions available in state s.
- Transition probabilities, denoted as $P(s, a, s')$, describe the probability of transitioning from state s to state s' when taking action a at time step t. It is a function mapping from $(S \times A \times S)$ to the range $[0, 1]$.
- Reward probabilities, denoted as $R(s, a, s')$, define the immediate reward the agent receives after transitioning from state s to state s' by taking action a. It is a function mapping from $(S \times A \times S)$ to the set of real numbers ($\mathbb{R}$).
- Discount factor, denoted as γ, is a value between 0 and 1 that determines the importance of future rewards in the computation of the discounted expected return.

It should be noted that the MDP is considered finite if the sets of states (S) and actions (A) are finite.

1.2.5 Discounted Expected Reward

In order to effectively train an agent, its objective should be to maximize the overall reward over the long term, rather than focusing solely on immediate returns. Let's consider the environment depicted in [Fig. 1.4], which consists of six different rooms. The agent begins in room one and the ultimate goal is to reach room five. The immediate reward is determined by the negative distance between the agent and room five. If the agent only prioritizes maximizing the immediate reward, it would choose to move to room three since the immediate reward there is higher than in rooms one or two. Unfortunately, this decision prevents the agent from reaching room five, and it remains stuck in room three indefinitely. Consequently, the agent never achieves its final goal. On the contrary, if the agent's objective is to maximize the expected return, it is willing to accept a lower immediate reward in room two, followed by higher immediate rewards in rooms four, six, and finally five. This sequence of actions leads to the accumulation of higher overall rewards.

The discounted expected return represents the cumulative sum of potential future rewards and can be calculated using Equation 1.1. The discount factor, γ, which lies within the range of [0, 1], determines the significance of future rewards and specifies how far into the future these rewards are considered. When $\gamma = 1$, all future rewards are given equal weight. In contrast, when $\gamma = 0$, only the immediate reward is taken into account.

The formula for calculating the discounted expected return, denoted as G_t, is as follows:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1.1}$$

Here, R_t represents the immediate reward at time step t, and k is the index used for summation.

We can distinguish between two types of tasks: episodic and continuous.

- **Episodic:** The training procedure can be divided into episodes. An episode concludes when the agent reaches a terminal state, at which point the scene is reset and the agent begins the next episode. The terminal state yields an immediate reward of 0 and can be reached within a finite number of time steps, denoted as T.
- **Continuous:** This type of problem cannot be formulated in episodes, as it is an ongoing, continuous process. In this case, $T = \infty$, indicating an infinite number of time steps.

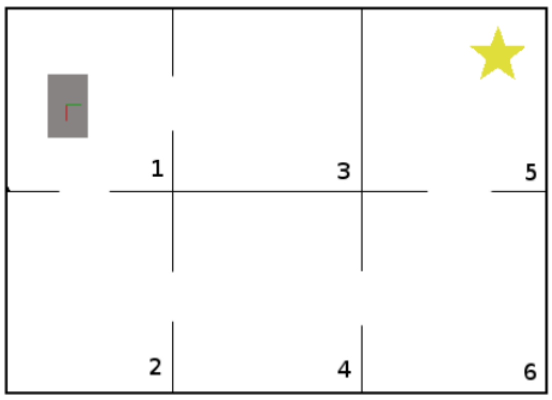


FIGURE 1.3: Immediate reward vs. expected return [10]

1.2.6 Policy and Value Functions

A policy, denoted as π, guides the behavior of the agent. It models a probability distribution $\pi(a|s)$ over the available actions $a \in A(s)$ for each state s. In other words, $\pi(A_t|S_t)$ represents the probability of selecting action A_t in state S_t. The agent samples its next action from this probability distribution, $\pi(a|s)$.

The value function $v_\pi(s)$ provides an estimate of the advantage of being in a particular state s. It represents the expected discounted return of being in state s, assuming the agent follows policy π (see Equation 1.2).

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right], \quad \text{for all } s \in S \tag{1.2}$$

Equation 1.2 can be recursively formulated, leading to the Bellman Equation for v_π (Equation 1.3). In the Bellman equation, the value of a state s depends on the next possible states s_0, weighted by the transition probability P_{s,s_0}. Many Reinforcement Learning approaches approximate the optimal Bellman equation by estimating the value of the next states.

$$\begin{aligned} v_\pi(s) &= E_\pi[G_t|S_t = s] \\ &= E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_{\pi(a|s)} \sum_{P_{s,s_0}} [R_{s,s_0} + \gamma E_\pi[G_{t+1}|S_{t+1} = s']] \\ &= \sum_{\pi(a|s)} \sum_{P_{s,s_0}} [R_{s,s_0} + \gamma v_\pi(s')] \end{aligned} \tag{1.3}$$

The action-value function $q_\pi(s, a)$ estimates the desirability of taking action a in state s. It represents the expected return of selecting action a in state s and subsequently following policy π.

$$q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right] \tag{1.4}$$

Reinforcement Learning aims to discover an optimal policy π^*. A policy is considered better than another policy $\pi \geq \pi_0$ if its value function, $v_\pi(s)$, is superior to $v_{\pi_0}(s)$ for all states $s \in S$. When the state-value function is optimal, it implies that the agent has adopted an optimal policy. It is possible for multiple optimal policies to yield the same optimal state-value function. The optimal state-value function v_* is defined as follows:

$$v_*(s) = \max_\pi v_\pi(s) \quad \text{for all } s \in S \tag{1.5}$$

Moreover, optimal policies correspond to the optimal action-value function q_*.

$$q_*(s,a) = \max_\pi q_\pi(s,a) \quad \text{for all } s \in S \text{ and } a \in A(s) \tag{1.6}$$

$$= E\left[R_{t+1} + \gamma v_*(s_0)|S_t = s, A_t = a\right] \tag{1.7}$$

Finally, the Bellman optimality equation [11] (Equation 1.8) can be derived from the previously introduced equations.

$$v_*(s) = \max_{q\pi^*} q_*(s,a) = \max_{E[P_{s,s_0}]} \left[R_{s,s_0} + \gamma v_*(s')\right] \tag{1.8}$$

1.2.7 Monte Carlo Method

The Monte Carlo method is an iterative approach used to solve reinforcement problems with episodic tasks when there is no existing model of the environment. Its goal is to converge towards the optimal policy as the number of episodes increases [12].

In this method, a Q-table is employed to store the action-value for each possible state-action pair. The value represents the average return collected across all episodes. Whenever the agent encounters a state-action pair, the corresponding entry in the Q-table is updated using the following equation:

$$Q(St, At) = Q(St, At) + \frac{1}{N(St, At)} \cdot (Gt - Q(St, At)) \tag{1.9}$$

where $N(St, At)$ denotes the number of visits to that specific state-action pair.

To weigh different returns, a factor α can be introduced. By adjusting α, recent returns can be given more or less importance, as shown in the following equation:

$$Q(St, At) = Q(St, At) + \alpha \cdot (Gt - Q(St, At)) = (1-\alpha) \cdot Q(St, At) + \alpha \cdot Gt \tag{1.10}$$

The Monte Carlo method proceeds through episodes. During the Evaluation step, the agent follows policy π for one episode. After completing the episode, the agent possesses a sequence of experiences $(S1, A1, R2, S2, ..., ST)$ and updates the Q-table accordingly. For each state-action pair (St, At) in the sequence, the expected return Gt is obtained, and the corresponding entry in the Q-table is updated using equation (1.10). In the Improvement step, the policy π is updated based on the recent Q-table. The next iteration begins with the Evaluation step, utilizing the updated policy. Figure 2.8 illustrates the concept of the Monte Carlo method.

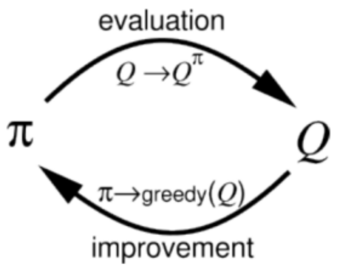


FIGURE 1.4: Concept of the Monte Carlo method [10]

The most conservative policy update is known as the greedy policy. In this case, the agent always selects the action with the highest action-value. However, relying solely on greedy actions can lead to suboptimal policies, as the agent might get stuck. To address this issue, the agent should occasionally take exploratory actions. An e-greedy policy is employed, where the greedy action is chosen with a probability of $(1 - e)$, and a random action is selected with a probability of e to explore the action space. The formula for the e-greedy policy is as follows:

$$\pi(a|s) \leftarrow \begin{cases} 1 - e & \text{if } a = \operatorname{argmax} Q(s,a) \\ \frac{e}{|A(s)|} & \text{otherwise} \end{cases} \tag{1.11}$$

Here, e represents a value between 0 and 1, determining the balance between exploitation and exploration. It is reasonable to emphasize exploration at the beginning when the agent lacks reliable knowledge. As the agent accumulates more experiences, the focus shifts towards exploiting the learned knowledge. A common approach is to gradually decrease e over time until it reaches a minimum value of e_{min}, which reflects this behavior.

1.2.8 Q-Learning

Q-Learning is a popular off-policy TD control method, which means that the policy π is not used for updating the Q-table [7]. The main difference compared to Sarsa is that instead of using the action-value of the next state-action pair $Q(S_{t+1}, A_{t+1})$, only the maximum Q-value of the next state S_{t+1} is taken into account. The action-value update rule for Q-Learning is shown in Equation 1.12:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max Q(S_{t+1}, a) - Q(S_t, A_t)\right] \tag{1.12}$$

In this equation, α represents the learning rate, R_{t+1} is the reward obtained at time $t+1$, γ denotes the discount factor, and $\max Q(S_{t+1}, a)$ signifies the maximum Q-value for state S_{t+1} over all possible actions a. By applying this update rule, the Q-values of the state-action pairs are iteratively adjusted in Q-Learning.

1.2.9 Deep Reinforcement Learning (DRL)

In the classical Reinforcement Learning discussed, all approaches rely on a tabular setting, which presents essential disadvantages. One limitation is that the use of a table restricts classical approaches to tasks with a low number of states and actions. However, real-world problems often involve large state spaces, making it infeasible to visit all possible states to retrieve values for action-state pairs. Additionally, the size of the table is constrained by hardware memory limitations.

Furthermore, classical RL approaches lack knowledge sharing among similar states, which can hinder representation quality and prolong training times.

To overcome these limitations, a common approach is to replace the value table with a Deep Neural Network (DNN) as a function approximator. DNNs have the ability to approximate non-linear functions and extract complex patterns from data. By utilizing a DNN, the drawbacks of the tabular representation can be mitigated. The DNN-based approach enables more efficient handling of large state spaces and facilitates knowledge transfer between similar states, resulting in improved representation quality and reduced training times.

1.2.10 Deep Reinforcement Learning: Advancements and Improvements

In classical Reinforcement Learning (RL), traditional methods that rely on tabular representations are being replaced by Deep Neural Networks (DNNs). This transition allows for more efficient handling of tasks with large state and action spaces, as well as the ability to learn from high-dimensional raw sensory input without hand-crafted features.

One notable improvement in DRL is the introduction of Deep Q-Network (DQN) [13]. DQN combines Q-Learning with DNNs to approximate action values and make decisions. It utilizes an end-to-end architecture, enabling the network to extract relevant features autonomously. The DQN approach overcomes challenges of unstable learning by incorporating mechanisms such as experienced replay and a frozen target network.

Experienced replay involves storing and sampling past experiences to break data correlations and provide independent training samples. The frozen target network, represented by a separate network with fixed weights, helps stabilize learning by reducing policy oscillations.

Several improvements have been made to enhance DQN's performance. Double DQN addresses the overestimation of Q-values by using two separate networks to estimate the TD-target. Prioritized Experienced Replay prioritizes important experiences for more effective learning. Dueling DQN introduces a network architecture that separates the estimation of state value and action advantage, improving the identification of relevant state features.

These advancements in DRL, replacing classical RL with DNNs and incorporating improvements like DQN, have significantly contributed to the success and applicability of RL in complex and high-dimensional environments.

1.3 Reinforcement Learning with Human Feedback (RLHM)

1.3.1 Definition

RLHM is a subfield of Machine Learning that focuses on building algorithms and models capable of learning and making decisions interactively [14]. It incorporates the usage of an entity called an agent, which interacts with an external system known as the environment. The agent perceives the current situation or configuration of the environment, referred to as the state, and takes a specific move or decision, known as an action, in that particular state. The agent's strategy or behavior, represented by a direct map between states and actions, is called the policy. The policy defines how the agent chooses actions based on the current state and can be deterministic or stochastic.

After taking action in a specific condition, the agent receives a scalar feedback signal from the environment, known as a reward. This reward serves to evaluate the quality of the agent's actions and guides it to learn the optimal behavior. The main objective of RLHM is to find an optimal policy that maximizes the expected cumulative reward over time. Typically, the agent achieves this by exploring different actions to gather information about the environment and exploiting its learned knowledge to make better decisions.

1.3.2 Types of Human Feedback

The input HF in an RL model can take various forms, such as reward shaping, demonstrations, or detailed evaluations, and it plays a crucial role in improving the agent's performance and accelerating the learning process. Below we will look into each HFs in detail and try to understand their usability and pros and cons.

- **Reward Shaping**

  It involves providing explicit rewards or penalties to the RL agent based on its actions. Human experts can design reward functions to reinforce the desired rewards and discourage undesired behavior. This feedback form helps the agent learn the optimal policy by maximizing its cumulative compensation.

  Pros:

  - Faster Learning: Providing informative rewards helps converge to the optimal policy more quickly.
  - Guided Exploration: It only allows exploration of the promising regions of the state-action space.

  Cons:

  - Potential Bias: Often, it may introduce biases if implemented improperly, thus influencing the agent's behaviors in the wrong way, leading to suboptimal policies.
  - Incorrect Shaping: Designing the correct shaping functions is challenging and should always be done carefully.

- **Demonstrations**

  This involves human experts or demonstrators showcasing the desired behavior by showcasing the desired actions or trajectories. The RL agent then learns or imitates these behaviors to develop and generalize policies.

  Pros:

  – Efficient Learning: The learning process can be increased incrementally by adding the demonstrator's knowledge to the agent's initial knowledge.

  – Safe Exploration: By imitating expert behavior, the agent can avoid potentially harmful or inefficient actions during the exploration phase.

  Cons:

  – Lack of Exploration: By solely depending on the expert's knowledge, the RL agent may be deprived of its inherent tendency to explore and discover novel solutions, thus limiting its capabilities.

  – Expert Sub-optimality: The availability of high-quality demonstrators is limited and costly, and using imperfect or suboptimal demonstrators might lead the RL agent to inherit the limitations.

- **Critiques and Advice**

  Humans critique or advise the agent's learned policies in this feedback form. They can evaluate the agent's behavior or suggest improvements to enhance performance. This feedback helps iteratively refine the agent's policies and align them more with human preferences.

  Pros:

  – Fine-grained Guidance: Humans can provide specific feedback to help the agent improve its behavior in a targeted manner.

  – Policy Refinement: Iterative feedback and advice can enhance the agent's policies over time.

  Cons:

  - Subjectivity: Human feedback may vary, challenging reconciling conflicting advice or critiques.
  - Feedback Quality: The quality and relevance of human advice can vary, and suboptimal feedback may hinder learning progress.

- **Ranking and Preferences**

  Human experts provide the RL agent with rankings or preferences for the agent's different actions or policies. By comparing the options, the RL agent can develop the optimal moves.

  Pros:

  - Preference Learning: Incorporating human preferences allows the agent to focus on actions or policies more likely to be desired by humans.
  - Fine-grained Control: Humans can communicate nuanced preferences, enabling the agent to optimize for specific criteria.

  Cons:

  - Subjectivity: Human preferences may vary, making it challenging to reconcile conflicting feedback.
  - Limited Feedback Granularity: Assigning precise scores or rankings to actions or policies may be difficult for humans, leading to less informative feedback.

1.3.3 RLHM Framework: Reinforcement Learning from Human Feedback

The RLHM (Reinforcement Learning from Human Feedback) framework, also known as the General Tamer Framework [14], is a method for training agents in sequential decision-making tasks. In traditional RL approaches, agents learn autonomously through interaction with the environment using reinforcement learning algorithms. However, in the RLHM framework, the focus is on learning the reward function through supervised learning, with the help of human trainers [Fig. 1.5].

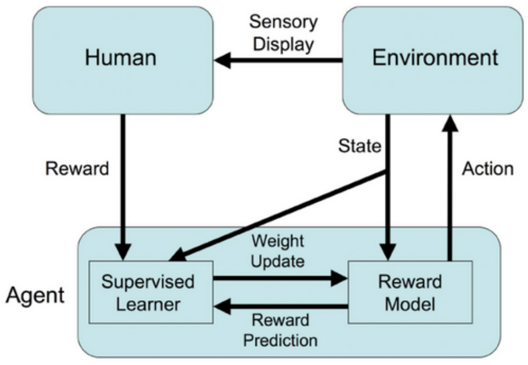


FIGURE 1.5: Immediate reward vs. expected return [14]

In the RLHM framework, a finite Markov decision process (MDP) is defined by the tuple (S, A, T, γ, D, R), where S and A represent the sets of possible states and actions, T is the transition function that specifies the probabilities of transitioning from one state to another, γ is the discount factor for future rewards, D is the distribution of start states, and R is the reward function that assigns rewards based on the current and next states.

Unlike traditional RL, where the agent interacts solely with the environment, the RLHM framework incorporates human trainers who provide feedback to the agent. The reward function R is removed from the task specification, and instead, the human trainer evaluates the agent's performance and provides rewards in the form of expressions that can be mapped to scalar values. The agent's goal is to select actions that maximize the reward provided by the human trainer.

The RLHM framework adopts a supervised learning approach, treating each action as a training sample. The states and the human trainer's reward for each action are considered attributes and labels, respectively. The agent models the human's reward function and greedily selects actions that are expected to yield the highest reward. Once the agent has learned an accurate model of the human's reward, it can continue performing the task even in the absence of the human trainer.

A key challenge in the RLHM framework is that human reward functions are not consistent and may change over time as the trainer's standards and preferences evolve. Therefore, recency-weighted function approximators, such as supervised learning algorithms, are used to accommodate these changing reward functions.

One notable aspect of the RLHM framework is that the reward function comes exclusively from the human trainer, and there is no predetermined "correct" behavior specified by the environment. The agent's behavior is deemed correct or optimal solely based on the trainer's evaluation.

While it is possible for the agent to learn autonomously by combining the human trainer's feedback with environmental rewards using traditional RL algorithms, the RLHM framework focuses on the challenge of learning the human's reward function in isolation. This isolated learning process allows the agent to act greedily based on the learned reward function. Additionally, the RLHM framework is particularly useful when the environmental reward function is difficult to define or when the correct policy depends on specific human preferences.

1.3.4 Approaches to Incorporate Human Feedback into Reinforcement Learning

Incorporating human feedback into reinforcement learning (RL) agents can be achieved through various approaches. Here, we will explore different strategies for integrating human feedback into RL agents and discuss their brief descriptions.

- **Interactive Learning**

  Interactive learning methods involve direct engagement between the learning agent and human experts or users. The agent actively seeks feedback and adapts its behavior based on input. Two approaches commonly used in interactive learning are:

  – Active Learning: The agent selects informative instances or queries humans for feedback on specific data points to accelerate learning.

  – Online Learning: The agent receives real-time feedback from humans, continuously adapting its policy based on the received feedback.

- **Imitation Learning**

  Imitation learning, also known as learning from demonstrations, aims to acquire a policy by emulating expert behavior. Expert humans provide sample trajectories or actions, which the agent tries to mimic. Two popular approaches in imitation learning are:

  - Behavioral Cloning: The agent learns to mimic the demonstrated behavior by mapping observations to actions, aiming to match the expert's actions without considering the underlying reward signal.
  - Inverse Reinforcement Learning: The agent infers the underlying reward function from expert demonstrations, enabling it to learn a policy that aligns with the expert's preferences.

- **Reward Engineering**

  Reward engineering involves modifying the reward signal to guide the agent's learning. Human experts design shaping functions or provide additional rewards that encourage desired behavior or penalize undesirable actions. Two common approaches in reward engineering are:

  - Reward Shaping: Shaped rewards are added to the environment's intrinsic reward signal to provide additional guidance to the agent.
  - Reward Modeling: Human experts explicitly model the reward function based on their preferences or domain knowledge, allowing the agent to learn from the expert's reward model.

- **Preference-based Learning**

  Preference-based learning methods involve gathering comparisons or rankings of different actions or policies from human evaluators. The agent learns to optimize its behavior based on observed preferences. Two approaches commonly used in preference-based learning are:

  - Pair-wise Comparison: Humans provide preferences by comparing pairs of actions or policies and indicating their preferred option.
  - Rank-based Comparison: Humans rank different options based on their desirability, providing a relative ordering of actions or policies.

- **Natural Language Feedback**

Natural language feedback allows humans to communicate with the learning agent using instructions, critiques, or explanations. The agent processes the textual input and adapts its behavior accordingly. Two approaches in natural language feedback are:

– Text-based Reinforcement Learning: The agent incorporates natural language instructions or feedback to guide decision-making.

– Language Grounding: The agent learns to associate textual feedback with specific states or actions to understand and respond to human instructions.

By combining these approaches, RL agents can effectively leverage human feedback to improve their decision-making capabilities in complex real-world scenarios.

1.3.5 Challenges of Reinforcement Learning with Human Feedback

Addressing the challenges associated with reinforcement learning with human feedback is crucial for effectively integrating and utilizing human guidance. Several key challenges arise in this context:

- **Feedback Quality and Consistency**

  Human feedback can be subjective and inconsistent, making it difficult to interpret and effectively utilize. Different individuals may have varying preferences, leading to conflicting guidance. Ensuring high-quality and reliable feedback becomes essential for training accurate and robust reinforcement learning models.

- **Scalability and Cost**

  Collecting and annotating human feedback can be resource-intensive, time-consuming, and costly. As tasks and environments become more complex, obtaining sufficient and diverse feedback becomes increasingly challenging, particularly in large-scale or real-time systems.

- **Exploration-Exploitation Tradeoff**

  Balancing exploration and exploitation in reinforcement learning is critical for learning optimal policies. Incorporating human feedback without compromising exploration poses a challenge. Over-reliance on human guidance can limit the agent's ability to explore and discover novel solutions.

- **Generalization and Transfer Learning**

  Human feedback is often specific to particular tasks or environments, making generalization to new scenarios or domains non-trivial. Ensuring that learned policies and models can transfer knowledge from one context to another presents a significant challenge.

- **Subjectivity and Bias**

  Human feedback can be subjective and influenced by personal preferences, biases, or context-dependent factors. Addressing bias in feedback and ensuring fairness and inclusivity become essential considerations.

- **Feedback Delay and Feedback Inconsistency**

  Obtaining real-time feedback from humans may not always be feasible, leading to feedback delays that can hinder the learning process, especially in dynamic environments. Additionally, inconsistencies or changes in feedback over time can challenge maintaining policy coherence.

Effectively addressing these challenges will contribute to the successful integration of human feedback into reinforcement learning, enabling the development of robust and adaptive learning systems.

1.3.6 Applications of Reinforcement Learning and Reinforcement Learning with Human Feedback

Both reinforcement learning (RL) and reinforcement learning with human feedback (RLHF) have wide-ranging applications across various domains. Let's explore some of these applications in detail:

- **Robotics**

  Reinforcement learning with human feedback can be employed in robotics for tasks such as robot manipulation, object grasping, and locomotion. Human experts can provide demonstrations or critiques to guide the robot's learning process and improve its performance in real-world environments.

- **Game Playing**

  Reinforcement learning with human feedback can be used to train game-playing agents. Human experts can provide demonstrations or rankings to enhance the agent's decision-making, strategy, and overall gameplay, leading to more intelligent and competent game-playing agents.

- **Autonomous Vehicles**

  Applying reinforcement learning with human feedback to autonomous vehicle systems is another valuable application. Human feedback can assist in training the vehicle to navigate complex traffic scenarios, improve safety, and handle challenging driving situations, ultimately enhancing the overall performance of autonomous vehicles.

- **Dialogue Systems**

  Reinforcement learning with human feedback can be utilized to train conversational agents in natural language processing and dialogue systems. Human evaluations, critiques, or preferences can guide the agent's responses, improve dialogue coherence, and enhance user satisfaction in conversational interactions.

- **Healthcare**

  Reinforcement learning with human feedback holds potential in healthcare applications, such as personalized treatment planning, medical diagnosis, and drug discovery. Human feedback can contribute to optimizing treatment decisions, improving the accuracy of medical diagnoses, and facilitating the discovery of more effective drugs.

- **Recommender Systems**

  Employing reinforcement learning with human feedback in recommendation systems enables the system to learn user preferences and provide personalized recommendations. Human feedback in the form of ratings, reviews, or explicit preferences can guide the system's learning process, resulting in more accurate and relevant recommendations for users.

1.4 Conclusion

In conclusion, the field of Reinforcement Learning with Human Feedback (RLHM) offers valuable approaches to enhance machine learning algorithms and models. By incorporating human feedback in the form of reward shaping, demonstrations, critiques and advice, or ranking and preferences, RLHM aims to improve decision-making and policy optimization. These approaches bring advantages such as faster learning, safe exploration, fine-grained guidance, and preference learning. However, challenges like feedback quality, scalability, exploration-exploitation tradeoff, generalization, subjectivity, and feedback delay need to be carefully addressed. Despite these challenges, RLHM holds great potential in various applications such as robotics, game playing, and autonomous vehicles, paving the way for intelligent and adaptive systems.

Brain Computer Interface

2.1 Introduction

Brain-Computer Interfaces (BCIs) enable direct communication between the brain and external devices, relying on the recording and interpretation of brain signals. This chapter provides a theoretical and technical background on BCIs. It begins by discussing the two approaches in BCI design: synchronous and asynchronous BCIs [15], highlighting their respective advantages and challenges. The chapter then explores the categorization of BCIs into invasive and non-invasive methods [16], discussing the benefits and drawbacks of each approach. Furthermore, it delves into the different lobes of the brain and their associated functions, emphasizing that brain functions are integrated and extend beyond individual lobes. The chapter also explains electroencephalography (EEG) rhythms [17] and their significance in understanding brain activity. Event-related potentials (ERPs) are introduced as electrical brain responses that provide insights into cognitive processes [18], with a particular focus on the P300 component. Lastly, the chapter introduces hybrid BCIs, which integrate multiple modalities to enhance accuracy and versatility in capturing and interpreting neural activity. The technical background section covers brain signal acquisition techniques such as EEG, fMRI, MEG, NIRS, and ECoG, discussing their characteristics and applications. Understanding these foundational concepts is crucial for developing effective and reliable BCI systems.

2.2 Theoretical Background

2.2.1 Definition of BCI

A Brain-Computer Interface (BCI) is a system that enables direct communication between the brain and external devices or applications. It translates brain activity into commands or actions without relying on traditional motor outputs. BCIs utilize various technologies to capture and interpret brain signals, offering potential applications in communication, control, and neurofeedback.

- **Synchronous or Asynchronous BCI**

  Synchronous and asynchronous BCIs are two distinct approaches in the field of brain-computer interfaces. Synchronous BCIs provide precise timing control, ensuring that commands are received and executed consistently. However, they impose constraints on the user, requiring them to conform to the BCI's timing constraints. On the other hand, asynchronous BCIs offer more freedom and flexibility as users can initiate commands at their own pace. They can control the timing of their actions and even pause the BCI system as needed. Implementing asynchronous BCIs presents technical challenges due to the need to accurately recognize the user's intention to send or withhold commands. Distinguishing the brain's "idle" state, which is always functioning, from intentional inactivity can be complex. Several methods have been developed to address this challenge, including thresholding, using a "brain-switch," or utilizing muscle movements or blinks as triggers to initiate BCI functions. Ongoing research aims to combine the strengths of both approaches to enhance the usability and performance of BCI systems [19].

- **Non-Invasive BCI vs Invasive BCI**

  BCIs, or brain-computer interfaces, can be categorized into invasive and non-invasive methods. Invasive BCIs involve a surgical procedure to implant electrodes beneath the scalp for direct communication with the brain signals. This method offers the advantage of more accurate readings. However, there are drawbacks to invasive BCIs, such as potential side effects from the surgery itself, including the formation of scar tissue that can weaken brain signals. Moreover, research by Abdulkader [20] highlights the possibility of the body rejecting the implanted electrodes, leading to medical complications.

  On the other hand, non-invasive BCIs utilize neuroimaging technologies as interfaces without the need for surgery. The majority of BCI research focuses on non-invasive EEG-based BCIs. These EEG-based interfaces are convenient to wear and do not require invasive procedures. However, they have limitations in terms of spatial resolution and the utilization of higher-frequency signals [21]. The skull dampens signals, causing dispersion and blurring of the electromagnetic waves generated by neurons. Additionally, EEG-based interfaces require some preparation time and effort prior to each usage session. In contrast, both non-EEG-based and invasive BCIs do not require prior training.

2.2.2 Lobes of The Brain

The lobes of the brain are distinct regions or divisions that are responsible for different functions and processes [Fig. 2.1]. The human brain is typically divided into four main lobes: the frontal lobe, parietal lobe, temporal lobe, and occipital lobe. Each lobe plays a crucial role in various aspects of cognition, perception, and motor control [22]. Here's a brief overview of the functions associated with each lobe:

- **Frontal Lobe:** Located at the front of the brain, the frontal lobe is involved in higher cognitive functions, personality, decision-making, problem-solving, planning, and motor control. It also houses the prefrontal cortex, which is associated with executive functions such as reasoning, judgment, and self-control.
- **Parietal Lobe:** Positioned behind the frontal lobe, the parietal lobe processes sensory information from the body and is responsible for spatial awareness, perception of touch, temperature, pain, and pressure. It also plays a role in integrating sensory information with other cognitive processes.
- **Temporal Lobe:** Situated on the sides of the brain, the temporal lobe is involved in auditory processing, language comprehension, memory formation, and the recognition of faces and objects. It includes the primary auditory cortex and the hippocampus, a structure vital for memory consolidation.
- **Occipital Lobe:** Found at the back of the brain, the occipital lobe is primarily responsible for visual processing and interpretation. It contains the primary visual cortex, which receives and processes visual information from the eyes, allowing us to perceive and understand the surrounding environment.

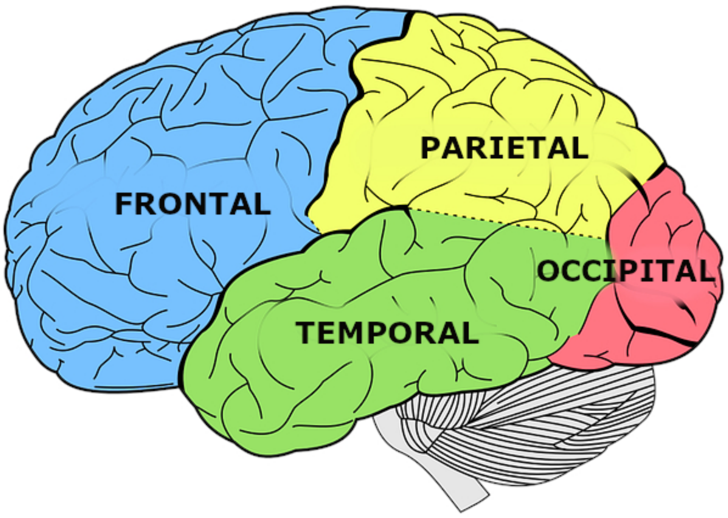


FIGURE 2.1: The lobes of the brain [23]

It's important to note that while these lobes have specialized functions, the brain works as an integrated system, and many cognitive processes involve the collaboration of multiple lobes. Additionally, other regions and structures within the brain also contribute to various functions, and the complexity of brain functions extends beyond the boundaries of individual lobes.

2.2.3 Electroencephalography Rhythms

Electroencephalography (EEG) is a non-invasive technique used to measure the electrical activity of the brain [17]. EEG rhythms refer to the patterns of electrical oscillations observed in the EEG signal [Fig. 2.2]. These rhythmic patterns are associated with different states of brain activity and play a crucial role in understanding brain function.

- **Delta Waves (0.5-4 Hz):** Delta waves are the slowest brain rhythms and are commonly observed during deep sleep. They are characterized by high amplitude and are indicative of a state of unconsciousness or relaxation.
- **Theta Waves (4-8 Hz):** Theta waves are often observed during drowsiness or light sleep. They can also be present during deep meditation or in certain pathological conditions. Theta waves are associated with memory processes and can be seen in states of creativity or daydreaming.

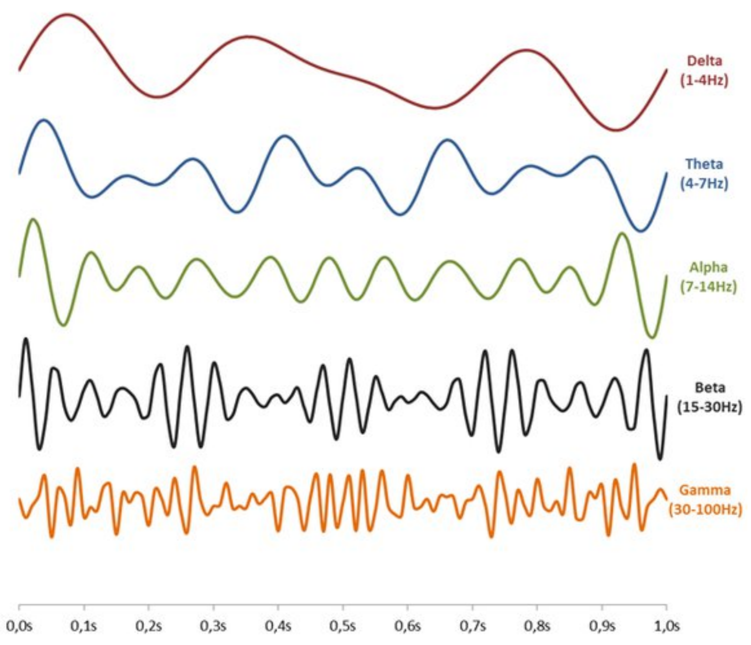


FIGURE 2.2: Electroencephalography Rhythms [24]

- **Alpha Waves (8-13 Hz):** Alpha waves are prominent in awake and relaxed individuals with closed eyes. They are commonly observed over the occipital region of the brain and are associated with a relaxed, calm, and meditative state.
- **Beta Waves (13-30 Hz):** Beta waves are typically observed when the brain is in an active, alert state. They are associated with cognitive processes, concentration, and focused attention. Beta waves can be further divided into low-beta (13-20 Hz) and high-beta (20-30 Hz) frequencies.
- **Gamma Waves (30-100 Hz):** Gamma waves are the fastest brain rhythms and are associated with high-level cognitive processes, perception, and information processing. They have been linked to attention, memory formation, and conscious awareness.

The specific frequency ranges mentioned above are approximate and can vary slightly depending on the source. It's important to note that the brain's electrical activity is highly complex, and these frequency bands often overlap and interact with each other.

2.2.4 Event-related potentials

Event-Related Potentials (ERPs) are electrical brain responses that occur in response to specific stimuli or events. These ERPs provide valuable insights into the underlying cognitive processes and are widely used in research studies [18, 25]. Among the various ERP components, the P300 has garnered significant attention and serves as a prominent focus in many research papers, including the present study. Let's explore the different ERP components and delve deeper into the P300 phenomenon.

- **N100:** The N100 component is a negative waveform that typically peaks around 100 milliseconds after the presentation of a stimulus. It is predominantly observed in auditory tasks and is believed to be associated with the early stages of sensory processing. The N100 reflects the brain's response to the onset of stimuli and contributes to the initial encoding and analysis of sensory information [26].

- **P200:** The P200 component is a positive ERP waveform that emerges approximately 200 milliseconds after stimulus onset. It is commonly linked to early cognitive processing, including perceptual and attentional processes. The P200 represents the brain's engagement in tasks that require the allocation of attention and the processing of relevant stimuli [26, 27].

- **N400:** The N400 component manifests as a negative deflection occurring around 400 milliseconds after stimulus presentation. It is primarily associated with language and semantic memory processes. The N400 is sensitive to the integration and processing of semantic information, reflecting the brain's response to meaningful stimuli in the context of language comprehension [28].

- **Late Positive Component (LPC):** The Late Positive Component, or LPC, is a positive ERP wave that typically emerges around 500 milliseconds post-stimulus and can persist for up to 1 second. The LPC is closely linked to memory processes, particularly recognition memory. It reflects the brain's engagement in evaluating and categorizing stimuli based on previous encounters or memory representations [29].

- **Focus on the P300:** In our research, we specifically focus on the P300 component due to its relevance and significance in cognitive neuroscience and related fields.

The P300 is a positive waveform that occurs approximately 300 milliseconds after stimulus onset. It is associated with cognitive processes related to attention, stimulus evaluation, and decision-making. The P300 is commonly investigated in tasks involving target detection, oddball paradigms, and cognitive assessments. Its amplitude and latency variations have been utilized as indicators of cognitive function, task engagement, and information processing capabilities.

2.2.5 Hybrid Brain-Computer Interfaces

Hybrid Brain-Computer Interfaces (BCIs) represent a cutting-edge approach in the field of neurotechnology. These interfaces combine multiple modalities, such as electroencephalography (EEG), functional near-infrared spectroscopy (fNIRS), and electromyography (EMG), to enable bidirectional communication between the brain and external devices [30]. By integrating different types of signals, hybrid BCIs offer enhanced accuracy, robustness, and versatility in capturing and interpreting neural activity. They hold great potential for various applications, including neurorehabilitation, assistive technologies, and cognitive augmentation [31]. The integration of multiple modalities in hybrid BCIs allows for a more comprehensive understanding of brain activity and facilitates the development of more advanced and adaptive neurotechnologies for improved human-machine interaction. Ongoing research in this field aims to further optimize the performance, usability, and potential applications of hybrid BCIs, paving the way for exciting advancements in the intersection of neuroscience and technology.

2.3 Technical Background

Brain-Computer Interfaces (BCIs) are systems that enable direct communication between the brain and external devices or computers. These interfaces rely on the recording and interpretation of brain signals to translate the user's intentions into commands or actions. Understanding the technical aspects of BCI frameworks [Fig. 2.3] is essential for developing effective and reliable systems.

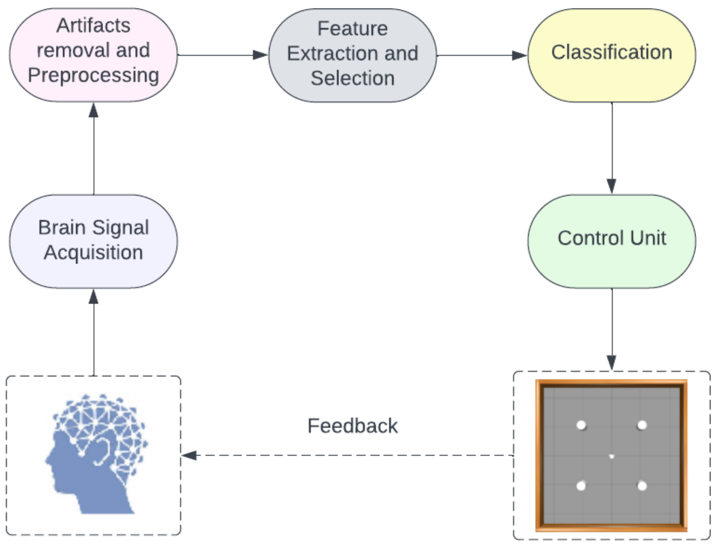

Artifacts removal and Preprocessing
Feature Extraction and Selection
Classification
Brain Signal Acquisition
Control Unit
Feedback

FIGURE 2.3: The framework of brain-computer interface (BCI)

2.3.1 Brain Signal Acquisition

Brain signal acquisition involves capturing electrical, hemodynamic, or magnetic signals generated by the brain. There are several types of techniques commonly used for brain signal acquisition in the field of brain-computer interfaces (BCIs). Here are some of the main types:

- **Electroencephalography (EEG):** EEG measures the electrical activity of the brain by placing electrodes on the scalp. It detects the small voltage fluctuations resulting from neural activity. EEG is non-invasive, portable, and provides a high temporal resolution, making it widely used in BCI research and applications.
- **Functional Magnetic Resonance Imaging (fMRI):** fMRI measures changes in blood oxygenation levels to indirectly infer neural activity. It uses a strong magnetic field and radio waves to map brain activation [32]. fMRI provides excellent spatial resolution but has a lower temporal resolution compared to EEG.
- **Magnetoencephalography (MEG):** MEG measures the magnetic fields generated by neural activity. It uses extremely sensitive sensors to detect the tiny magnetic fields produced by electric currents in the brain. MEG offers excellent temporal and good spatial resolution [33], allowing for precise localization of brain activity.

- **Near-Infrared Spectroscopy (NIRS):** NIRS measures changes in blood oxygenation and hemodynamics in the brain using near-infrared light [34]. It provides an indirect measure of neural activity and is relatively portable and less expensive compared to other techniques. NIRS has lower spatial and temporal resolution but can be suitable for certain BCI applications.

- **Electrocorticography (ECoG):** ECoG involves placing electrodes directly on the surface of the brain, either through invasive surgery or using subdural grids [35]. It provides high spatial and temporal resolution and is commonly used in clinical settings for epilepsy monitoring. ECoG offers insights into neural activity at a finer scale but requires invasive procedures.

  These techniques vary in terms of invasiveness, spatial and temporal resolution, portability, and cost. The choice of brain signal acquisition technique depends on the specific requirements of the BCI application, balancing factors such as the desired resolution, signal quality, and user comfort.

2.3.2 Artifacts Removal and Preprocessing

Artifacts removal and preprocessing are crucial steps in the analysis of EEG signals to enhance the quality and reliability of the data. These steps aim to minimize or eliminate unwanted artifacts that can arise from various sources. Here are some common types of artifacts removal and preprocessing techniques used for EEG signals:

- **Filtering:** Filtering is a fundamental preprocessing technique that involves applying digital filters to EEG signals. It helps to remove unwanted noise and artifacts while preserving the desired frequency components. Common filters used in EEG preprocessing include high-pass filters to remove baseline drift and low-frequency noise, and notch filters to eliminate power line interference [36].

- **Artifact Rejection:** Artifact rejection is the process of identifying and removing EEG segments contaminated by artifacts. This can be done manually by visually inspecting the data or using automated algorithms. Various types of artifacts can be identified and removed, such as eye blinks, eye movements, muscle activity, and electrode pops. Techniques like independent component analysis (ICA) [37] can be employed to separate artifact-related independent components from the EEG data.

- **Baseline Correction:** Baseline correction involves adjusting the EEG signal by subtracting a baseline value. This helps to remove any offset or drift in the signal and ensures that subsequent analyses are based on a consistent baseline . Baseline correction is typically performed by estimating the mean or median value of the baseline period and subtracting it from the entire signal [38].

- **Epoching:** Epoching involves segmenting the continuous EEG signal into shorter epochs or time windows. This segmentation is often based on specific events or experimental conditions. Epoching facilitates the analysis of EEG responses to specific stimuli or tasks and allows for the extraction of event-related potentials (ERPs) and other relevant features.

- **Artifact Correction:** Artifact correction techniques aim to correct or minimize the impact of artifacts on the EEG signal. These techniques can include template matching, regression-based methods [39], or adaptive filtering algorithms [40]. Artifact correction helps to restore the integrity of the EEG data and improve subsequent analysis and interpretation.

These are some of the key techniques used for artifacts removal and preprocessing of EEG signals. The specific techniques employed may vary depending on the characteristics of the data, the nature of the artifacts, and the objectives of the analysis. The goal is to ensure that the EEG data is free from artifacts as much as possible, allowing for accurate and meaningful analysis of brain activity.

2.3.3 Feature Extraction and Selection

Feature extraction and selection are essential steps in analyzing EEG signals to extract relevant information and reduce dimensionality. These steps involve identifying informative features from the raw EEG data that can be used for subsequent analysis and classification. Here are some common types of feature extraction and selection techniques used for EEG signals:

- **Time-domain Features:** Time-domain features are derived directly from the raw EEG signal and provide information about the signal's amplitude, amplitude variations, and temporal characteristics. Examples of time-domain features include mean amplitude, root mean square, variance, skewness, and kurtosis [41]. These features capture the overall signal characteristics and can be used to represent signal dynamics and variability.

- **Frequency-domain Features:** Frequency-domain features are obtained by transforming the EEG signal from the time domain to the frequency domain using techniques such as the Fourier transform or wavelet transform. These features capture the spectral characteristics of the EEG signal and provide information about the power distribution across different frequency bands. Common frequency-domain features include power spectral density, spectral entropy, and peak frequency.

- **Statistical Features:** Statistical features involve the application of statistical measures to the EEG signal or its transformed representation. These features provide information about the statistical properties of the signal and can capture patterns, asymmetries, or regularities. Examples of statistical features include mean, standard deviation, skewness, kurtosis [41], correlation coefficients, and fractal dimensions.

- **Spatial Features:** Spatial features are derived from the spatial distribution of EEG signals recorded from multiple electrodes. These features capture the spatial patterns and relationships between different electrode locations. Common spatial features include topographic maps, spatial covariance matrices, and scalp potential difference measurements.

- **Waveform-based Features:** Waveform-based features focus on specific waveform components or patterns present in the EEG signal [42]. These features can be derived from individual waveforms such as peaks, troughs, or deflections, or from specific waveform patterns such as event-related potentials (ERPs). Examples of waveform-based features include peak amplitudes, latencies, slope measurements, and ERP components such as P300 or N400.

- **Feature Selection:** Feature selection techniques aim to identify the most informative and relevant features from the extracted feature set. These techniques help reduce dimensionality, eliminate redundant or irrelevant features, and improve classification performance. Common feature selection methods include statistical tests (e.g., t-tests, ANOVA) [43], information-theoretic approaches (e.g., mutual information, entropy), and machine learning-based feature selection algorithms (e.g., recursive feature elimination, genetic algorithms).

The choice of feature extraction and selection techniques depends on the specific EEG analysis task, the characteristics of the data, and the desired outcome. It is important to select features that capture the relevant aspects of brain activity and can discriminate between different brain states, events, or conditions.

2.3.4 Classification

Classification is a crucial step in the system design of a Brain-Computer Interface (BCI) framework, as it involves predicting or categorizing the user's intentions or mental states (emotion and P300) based on the extracted features from the brain signals. Machine learning and deep learning techniques are commonly employed for classification tasks in BCI frameworks. Here are some examples of machine learning and deep learning models used in BCI classification:

- **Machine Learning Models:**

  Support Vector Machine (SVM): SVM is a widely used machine learning algorithm for classification. It constructs a hyperplane or set of hyperplanes that separate different classes in the feature space. SVM has been successfully applied to various BCI applications, including motor imagery classification and mental state recognition [44].

  Extreme Gradient Boosting (XGBoost): XGBoost is a gradient boosting framework that utilizes an ensemble of decision trees for classification. It combines the predictions of multiple weak learners to create a strong classifier. XGBoost has shown excellent performance in BCI tasks, particularly in scenarios with high-dimensional feature spaces [45].

Random Forest: Random Forest is an ensemble learning method that builds a collection of decision trees and combines their predictions for classification [46]. It is robust against overfitting and can handle large feature sets efficiently. Random Forest has been employed in BCI applications for motor imagery classification and cognitive task recognition.

- **Deep Learning Models:**

  Convolutional Neural Network (CNN): CNN is a deep learning architecture particularly effective in processing structured data [47], such as images or spatially organized signals. In the context of BCI, CNNs have been applied to analyze electroencephalography (EEG) signals by treating them as images, where the electrodes or channels represent the image's spatial dimensions. CNNs can automatically learn spatial filters and hierarchical representations, making them suitable for tasks like motor imagery classification.

  Long Short-Term Memory (LSTM): LSTM is a type of recurrent neural network (RNN) designed to handle sequential data. LSTM networks [48] excel at capturing temporal dependencies and have been employed in BCI frameworks for tasks involving time-series EEG signals. LSTM models can capture the dynamics and temporal patterns in the EEG data, making them suitable for applications such as mental state recognition or EEG-based gesture recognition.

2.3.5 Control Unit

The Control Unit is a vital component of the BCI framework, responsible for translating users' brain signals into meaningful commands for device control. It also integrates feedback mechanisms to enhance the user experience. The Control Unit consists of the following key functionalities:

- **Signal Decoding and Command Translation:** The Control Unit encompasses the processes of Signal Decoding and Command Translation, which are crucial for accurate device control. Signal Decoding involves extracting relevant information from the user's brain signals, while Command Translation translates these decoded intentions into specific commands or actions. Machine learning and pattern recognition techniques are employed within the Control Unit to develop robust models for precise decoding and translation.

- **Control Device:** The Control Device is an integral part of the Control Unit, establishing the connection between the user and the BCI framework. It captures and interprets the user's intentions, serving as the interface for transmitting commands to the BCI system. The Control Device can take various forms, such as EEG headsets, implanted electrodes, or other devices capable of recording brain signals. Its seamless integration within the Control Unit ensures accurate and efficient transmission of user commands.

- **Feedback:** Feedback is an essential component integrated within the Control Unit, enhancing user engagement and performance in the BCI system. While not extensively discussed in this subsection, feedback plays a significant role within the framework. By providing real-time information about the system's interpretation of the user's intentions and the resulting device control actions, feedback serves purposes such as performance evaluation, system confidence and reliability assessment, and error correction. It enables users to evaluate their device control performance, understand the system's reliability, and make adjustments or adaptations based on the feedback received. The integration of feedback within the Control Unit contributes to an improved user experience and facilitates effective device control.

2.3.6 Applications of Brain-Computer Interfaces (BCIs)

Brain-Computer Interfaces (BCIs) have demonstrated great potential in various fields and have found applications in numerous domains. The ability to translate brain activity into actionable commands has opened up new possibilities for enhancing human capabilities and improving quality of life. Here are some notable applications of BCIs:

- **Assistive Robotics:** BCIs enable individuals with severe motor disabilities to control external devices such as prosthetic limbs, wheelchairs, and robotic exoskeletons using their brain signals. This technology empowers individuals with conditions like spinal cord injuries, amyotrophic lateral sclerosis (ALS), or locked-in syndrome to regain mobility and independence. I have investigated the implementation and effectiveness of BCIs in controlling robotic systems for assistive purposes.

- **Sentiment Analysis and Emotion Recognition:** In my thesis, I have also explored the use of BCIs for analyzing brain signals associated with emotions and sentiments. By decoding brain activity patterns, BCIs can provide insights into an individual's emotional state, allowing for applications in mental health monitoring, affective computing, and personalized user experiences. I have conducted research on developing algorithms and techniques for sentiment analysis and emotion recognition using BCIs.

- **Communication and Augmentation:** BCIs provide a means for individuals with communication impairments, such as those with locked-in syndrome or certain types of motor neuron diseases, to express themselves and communicate with others. By translating their thoughts into text or speech, BCIs enable them to overcome the limitations imposed by their physical disabilities. Furthermore, BCIs can be used to augment existing communication methods by offering faster and more efficient ways to compose and send messages.

- **Neurorehabilitation:** BCIs have shown promise in neurorehabilitation, particularly in stroke recovery and motor rehabilitation. They can facilitate the relearning of motor skills by enabling users to control robotic or virtual reality-assisted therapy systems. BCIs provide real-time feedback and encourage neuroplasticity by promoting the activation of specific brain areas associated with motor functions. This technology has the potential to enhance the effectiveness and efficiency of rehabilitation programs.

- **Cognitive Enhancement and Brain Training:** BCIs are being explored for cognitive enhancement applications, such as improving attention, memory, and mental focus. Neurofeedback-based BCIs can provide users with real-time information about their brain activity and guide them to achieve desired cognitive states. Additionally, BCIs have been used in brain training applications to enhance cognitive abilities in healthy individuals, such as improving working memory or reducing stress levels.

- **Gaming and Entertainment:** BCIs have made their way into the gaming and entertainment industry, offering novel and immersive experiences. They enable users to control characters or actions in virtual reality games using their brain signals, providing a more engaging and interactive gameplay environment. BCIs can also be used to adapt game difficulty levels based on the player's cognitive state or emotional responses, enhancing the overall gaming experience.

2.4 Related Works

2.4.1 EEG Based Emotion Detection of DEAP and SEED-IV Using SVM

In [49] provides a brief overview of related work on EEG-based emotion detection and proposes a method for detecting inner emotion states using publicly available datasets. The proposed approach involves applying Discrete Wavelet Transforms [50] to preprocessed EEG signals from the DEAP [51] and SEED-IV [52] databases to extract five frequency bands. Features such as power, energy, differential entropy, and time domain are then extracted. A supervised machine learning algorithm, specifically a channel-wise SVM classifier [53], is developed for emotion state recognition. The paper reports classification rates of 74%, 86%, 72%, and 84% for the DEAP database and 79%, 76%, 77%, and 74% for the SEED-IV database. The literature review section presents several studies that used different methodologies and achieved accuracy rates ranging from 62.5% to 90%. In this proposed method, the focus is on developing a machine learning algorithm for classifying emotion states into four classes using a channel fusion approach with data from DEAP and SEED-IV databases.

2.4.2 Correlated Attention Networks for Multimodal Emotion Recognition

In [54], the authors propose a novel model called Correlated Attention Network (CAN) for multimodal emotion recognition. The CAN model extends the attention-based recurrent neural network by incorporating correlation calculations of different gated recurrent units to capture the correlation between EEG and eye movement signals. By leveraging coordinated representation with complementary features, the CAN model aims to achieve higher emotion classification accuracy. Experimental results on three real-world datasets demonstrate that the proposed model outperforms state-of-the-art methods, achieving mean accuracies of 94.03% on the SEED dataset, 87.71% on the SEED IV dataset, and 88.51% and 85.62% for four classification and two dichotomies on the DEAP dataset, respectively. The contributions of the paper include the introduction of the CAN model, which combines deep gated recurrent neural networks [55] with canonical correlation and attention mechanism [56], and the exploration of coordinated representation in multimodal emotion recognition tasks.

2.4.3 Robot Navigation

In [57], an algorithm was proposed to solve the Simultaneous Localization and Mapping (SLAM) problem, which is crucial for autonomous navigation of mobile robots in unknown environments. Another lightweight and real-time efficient SLAM algorithm called OrthoSLAM [58] was introduced for simple mobile robots in indoor office-like environments. It simplifies the complexity by assuming parallel or perpendicular lines in the environment structure. A novel method combining Hector SLAM and artificial potential field (APF) controller was proposed in [59] for autonomous indoor navigation of wheeled mobile robots in GPS-denied environments, such as greenhouses. This method utilizes single Light Detection and Ranging (LiDAR) for localization, Hector SLAM for pose estimation, and an APF controller for autonomous navigation.

In [60], a semantically rich graph representation was suggested for indoor robotic navigation, enabling the robot to generate motor commands directly from semantic information, without explicit computation of precise location or environment geometry. However, this method was not implemented using a real robot. Two approaches [61, 62] introduced an end-to-end approach using Convolutional Neural Networks (CNNs) for autonomous mobile robot navigation using RGB-D cameras. One approach [62] focused on training the CNN model to directly generate velocity and angular rates for navigation. However, these approaches required extensive labeling and did not generalize well.

A DRL-based method for mobile robot navigation was proposed in [63], combining deep reinforcement learning and recurrent neural network (RNN) modules. This approach uses DRL to make decisions based on path selection. In [64], deep reinforcement learning based on a Deep Q-Network (DQN) was proposed for autonomous mobile robot navigation using only current visual observations. Limitations of this method include the reliance on RGB-D cameras, which restricts perception of the environment.

[65, 66] presented a deep reinforcement learning approach using a Double Deep Q-Network (DDQN) for autonomous navigation and collision avoidance. Inputs such as obstacle size, position, and target position were used to determine the robot's direction of motion. An asynchronous advantage actor-critic network was proposed in [67] for autonomous mobile robot navigation in an indoor environment, incorporating parallel environment simulation to enhance generalization and reduce learning time.

The paper's objective is to design autonomous mobile robot navigation trained using a DRL algorithm, which optimizes actions to maximize environmental rewards. The DQN and DDQN policies were trained in a Gazebo simulation environment and then evaluated in a testing environment. Finally, the evaluated policies were deployed on a real mobile robot without modifications. The training process required multiple iterations in the Gazebo environment to achieve successful navigation and collision avoidance. The goal is to train DQN and DDQN agents to autonomously navigate mobile robots in unknown environments.

In contrast to SLAM, this paper proposes an extension of the method in [66] and an end-to-end DRL approach for autonomous navigation without collisions in unknown environments. The primary contribution is a model-free DRL approach that doesn't require kinematics and dynamics equations for mobile robots. Additionally, the proposed method extends [66] by incorporating an RGB-D camera for target object detection and deploying the algorithm on a real mobile robot without hyperparameter tuning. Experimental results demonstrate that the proposed DRL algorithm enables autonomous navigation of mobile robots without collisions in unknown environments.

2.5 Conclusion

In conclusion, this chapter provided a comprehensive overview of Brain-Computer Interfaces (BCIs), covering theoretical and technical aspects. The theoretical background included definitions, distinctions, brain lobes, rhythms in EEG signals, and the introduction of Event-Related Potentials (ERPs). The technical background explored BCI frameworks, signal acquisition techniques, artifacts removal, and preprocessing steps. The concept of hybrid BCIs was also highlighted. Overall, this chapter established a foundation for understanding BCI fundamentals, enabling further exploration of applications and advancements in the field.

Design and Implementation of Proposed Model

3.1 Introduction

Robots have become essential in various applications, including rescue operations, medical assistance, and autonomous driving. Autonomous navigation is a key research topic in mobile robotics [68], with traditional methods relying on map-building-based techniques like SLAM [69]. However, mapless navigation has gained popularity, offering a direct mapping between sensory inputs and robot actions. Deep reinforcement learning (DRL) has made significant contributions to autonomous navigation, but real-world training poses challenges [70, 71]. This chapter aims to address practical implementation challenges of DRL in real robotic tasks, focusing on autonomous navigation. The chapter structure includes sections on related work, proposed methodology, experimental results, and conclusions. Additionally, an enhanced neurofeedback-based reinforcement learning model is proposed, incorporating algorithms like DDPG, DQN, and TD3 [72]. Reward functions and global system settings are discussed, providing guidance for the learning process. Overall, this chapter presents an in-depth study on enhanced neurofeedback-based reinforcement learning for robot navigation, leveraging DRL algorithms, reward functions, and global system settings.

3.2 Objectives

- Develop a fast and efficient method for extracting required features from brain waves, aiming to optimize the learning efficiency of Deep Reinforcement Learning (DRL) algorithms by incorporating neural feedback. This will enable faster learning in fewer episodes.

- Achieve accurate learning of robot navigation in complex and dynamic environments, leveraging natural human sensors and minimizing the number of additional sensors required for robot learning.
- Conduct in-depth case studies to identify suitable policies and reward functions that enable accurate learning of robot navigation and improve the generalizability of our novel approach.
- Explore different DRL methods to assess the applicability and effectiveness of our proposed approach in various scenarios.
- Incorporate emotions or attention signals captured when the human is in a relaxed state, thereby eliminating the need for additional effort or control during the robot learning process.
- Reduce the cost of materials required for robot learning by exploring ways to optimize the architecture and design prior to constructing a physical prototype.

3.3 Proposed model

3.3.1 Global System Design

- **Global System Design From AI Perspective**

  This global system design [Fig. 3.1] focuses on an AI perspective and comprises two main components: the Brain Signal part utilizing EEG and the DRL (Deep Reinforcement Learning) part. The system architecture involves an environment consisting of a maze with static obstacles representing maze walls and dynamic obstacles. A robot, controlled by a DRL agent, aims to navigate through the maze to reach a randomly generated red circular goal spot.

  The DRL agent receives two primary feedback signals for observations. The first is the EEG signal, representing emotions such as sadness, happiness, and neutrality. The second feedback is obtained from a 2D LIDAR sensor, providing environmental information to the agent. Human interaction with the system occurs through a BCI (Brain-Computer Interface) setup. Users observe the simulation and collect EEG recordings. The collected EEG data undergoes preprocessing,

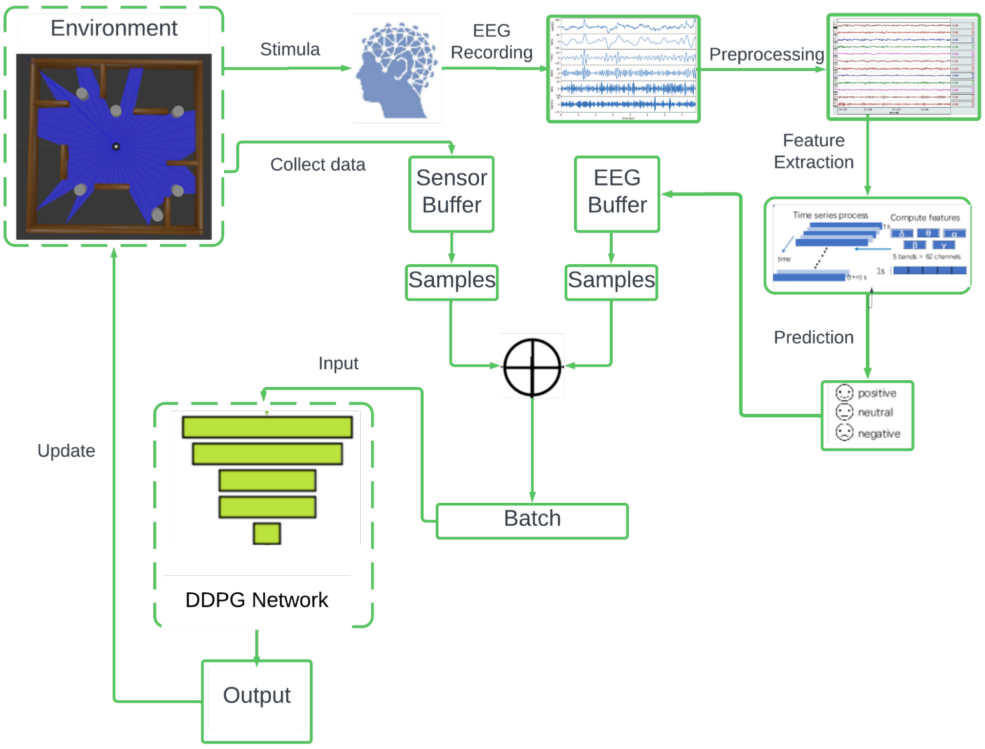


FIGURE 3.1: Global System Design

feature extraction, and model prediction to classify the emotional states into three classes: sad, happy, and neutral.

Simultaneously, the 2D LIDAR sensor data is combined with the EEG data and used as observations for the DRL agent. The agent learns to make optimal decisions and take actions based on the combined feedback to navigate the maze effectively. Multiple versions of the system have been developed, including variations that utilize only the 2D LIDAR sensor or focus solely on emotion feedback. Additionally, different noise types and constant feedback mechanisms have been explored to enhance system performance.

In the subsequent sections, specific DRL algorithms and their detailed functionalities will be explained, providing further insights into the system's operation and learning processes.

- **System Design From Network Perspective**

This figure [Fig. 3.2] represents a global system design from a network perspective, specifically for the Robot Operating System (ROS2+Gazebo). The design aims to illustrate the various nodes, publishers, and subscribers within the system. Nodes are depicted as circles, labeled with their corresponding node names. Publishers and subscribers are represented by solid arrows, while topics are shown as rectangular shapes pointing from the publisher node to the topic.

To simplify the visualization and avoid complexity, servers and clients are not displayed in the figure. The focus is primarily on nodes, topics, and their interactions. The schema used in the figure depicts a maze with six dynamic obstacles, although the same principle applies to different numbers of dynamic obstacles.

In this example, the TD3 agent is utilized. The nodes presented in the figure include:

- `/drl_environment`
- `/drl_gazebo`
- `/gazebo`
- `/obstacle/p3d_base_controller_obstacleX`
- `/robot_state_publisher`
- `/td3_agent`
- `/teleop_twist_keyboard`
- `/turtlebot3_diff_drive`
- `/turtlebot3_imu`
- `/turtlebot3_joint_state`
- `/turtlebot3_laserscan`

The topics depicted in the figure include:

- `/cmd_emotion`
- `/cmd_vel`
- `/goal_pose`
- `/imu`
- `/joint_states`
- `/obstacle/odom`

- `/odom`
- `/parameter_events`
- `/performance_metrics`
- `/robot_description`
- `/rosout`
- `/scan`

Overall, this figure provides an overview of the network architecture and communication flow within the system, emphasizing the nodes and topics involved in the ROS2+Gazebo setup.

3.3.2 Brain Signal Feedback Part

This is a sub-system design from the global system [Fig. 3.2] and it has 2 phases:

- **Phase 1**

  This phase involves training the model, such as the CNN model, using the SEED-IV dataset. We applied the preprocessing and feature extraction techniques as mentioned in the dataset section. For our model, we performed predictions for four classes. However, in our global system design, the classes of sadness and fear were treated as a single class. We then transformed the batches into 2D images to feed them into the CNN model. In addition to the CNN model trained in the train data loader, we also experimented with other models, such as advanced CNN architecture, LSTM, and ViT.

- **Phase 2**

  After finishing the first phase, we take the trained weights and apply them to our system as an offline model. The offline model has a predefined function that takes input from the BCI API and first transforms the features to predict the real-time emotions of the human. These predicted emotions are then stored in the EEG buffer. The EEG buffer is subsequently merged with the buffer of the 2D lidar to create the sensor buffer and form the created batch.

- **Replacmnet of Emotion feedback**

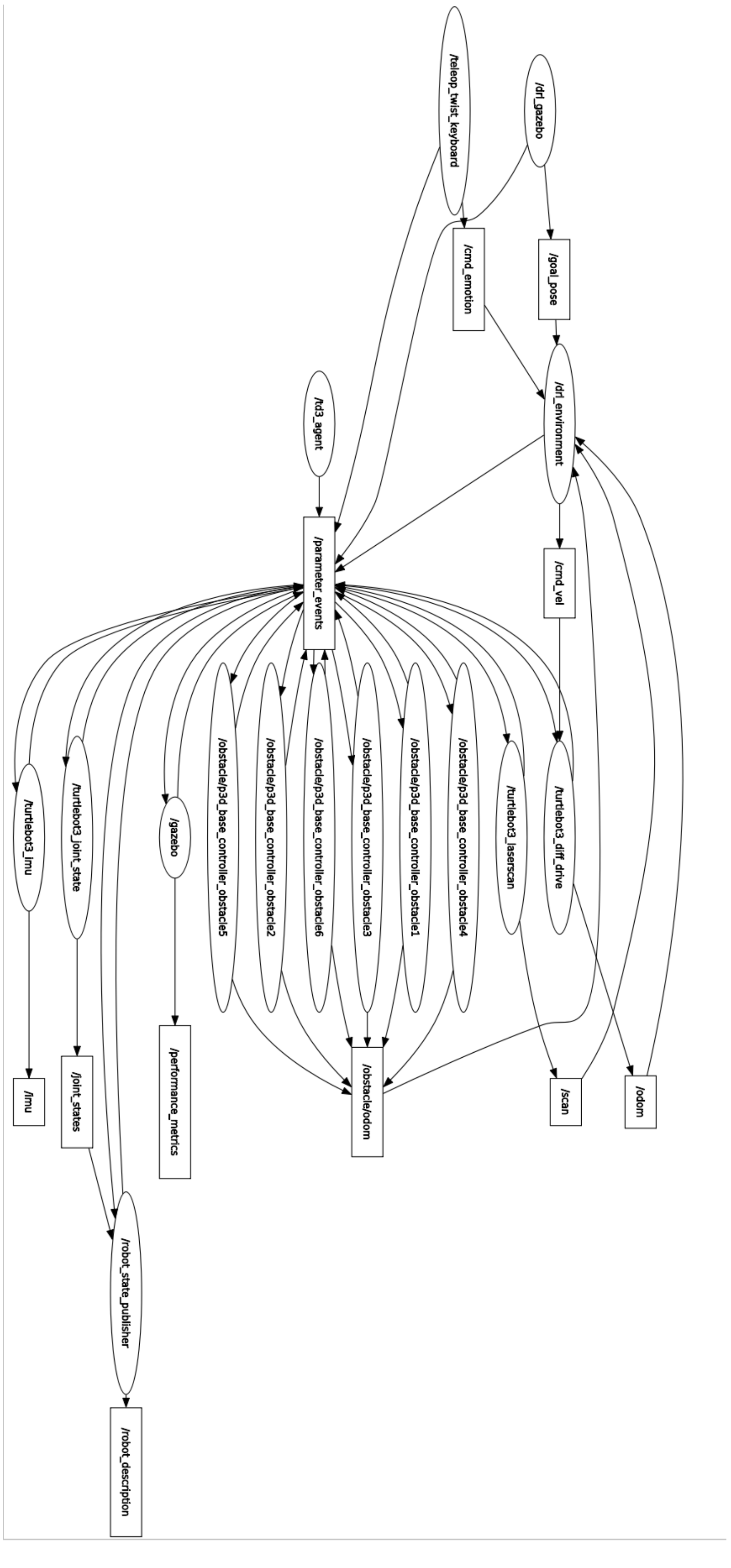


FIGURE 3.2: Network System Design

Regarding the brain signal feedback, it can be replaced with keyboard input using humans as feedback. In this case, the user can provide their emotional state as input, indicating whether they are feeling happy, sad, or neutral. Alternatively, we can replace the brain signal feedback system with generated samples of emotional states following a uniform distribution. In this approach, we have three variables that control the probability of each emotional event: happiness, sadness, and neutrality.

– **Classification Models Used For Emotion Dataset**

We utilized emotion classification as a feedback system and observation for deep reinforcement learning. This feedback provides information about the human's emotional state based on the simulation. To achieve this, we employed various models for classifying brain signals into four classes: angry, sad, happy, or neutral. These models encompassed a range of techniques.

∗ **Time Series Models**

· **GRU**

The GRU (Gated Recurrent Unit) model [73] consists of two GRU layers followed by a linear layer. Each GRU layer processes the input sequentially and updates its hidden state based on the current input and the previous hidden state. The output of the last time step of the second GRU layer is fed into a linear layer, which produces the final prediction. The GRU layers enable the model to capture sequential dependencies and the linear layer maps the learned representations to the desired output. Overall, the model utilizes the GRU architecture to process sequential data and make predictions based on the learned representations.

Layer	Type	Input Shape	Output Shape	Parameters
gru_layer1	GRU	[batch_size, 128, 32]	[batch_size, 128, 64]	12,672
gru_layer2	GRU	[batch_size, 128, 64]	[batch_size, 128, 64]	24,576
-	Dropout	[batch_size, 128, 64]	[batch_size, 128, 64]	0
out	Linear	[batch_size, 64]	[batch_size, num_classes]	130

TABLE 3.1: GRU Model Architecture

- **LSTM**

  The LSTM (Long Short-Term Memory) model [74] consists of two LSTM layers followed by a linear layer. Each LSTM layer processes the input sequentially and updates its hidden state and cell state based on the current input and the previous hidden state and cell state. The output of the last time step of the second LSTM layer is fed into a linear layer, which produces the final prediction. The LSTM layers are designed to capture long-term dependencies and handle the vanishing gradient problem. The linear layer maps the learned representations to the desired output. Overall, the LSTM model utilizes the LSTM architecture to process sequential data and make predictions based on the learned representations.

TABLE 3.2: Architecture of the LSTM model

Layer Name	**Layer Type**	**Output Size**
LSTM_1	LSTM Layer	(batch_size, hid_channels)
LSTM_2	LSTM Layer	(batch_size, hid_channels)
Linear	Linear Layer	(batch_size, num_classes)

* **CNN Models**

  - **CNN**

    The `CNN` class represents a convolutional neural network (CNN) model that consists of multiple blocks. Each block follows a similar structure, comprising the `ZeroPad2d`, `Conv2d`, and `ReLU` layers, details of layer model represented in the table [Tab. 3.3].

TABLE 3.3: Layers and Parameters in the CNN Model

Layer	Type	Input Shape	Output Shape	Number of Parameters
conv1	Conv2d	(batch_size, 20, H, W)	(batch_size, 64, H, W)	5,120
	ZeroPad2d	(batch_size, 64, H, W)	(batch_size, 64, H, W)	0

Continued on next page

Table 3.3 – Continued from previous page

Layer	Type	Input Shape	Output Shape	Number of Parameters
	ReLU	(batch_size, 64, H, W)	(batch_size, 64, H, W)	0
conv2	Conv2d	(batch_size, 64, H, W)	(batch_size, 128, H, W)	131,200
	ZeroPad2d	(batch_size, 128, H, W)	(batch_size, 128, H, W)	0
	ReLU	(batch_size, 128, H, W)	(batch_size, 128, H, W)	0
conv3	Conv2d	(batch_size, 128, H, W)	(batch_size, 256, H, W)	524,544
	ZeroPad2d	(batch_size, 256, H, W)	(batch_size, 256, H, W)	0
	ReLU	(batch_size, 256, H, W)	(batch_size, 256, H, W)	0
conv4	Conv2d	(batch_size, 256, H, W)	(batch_size, 64, H, W)	262,208
	ZeroPad2d	(batch_size, 64, H, W)	(batch_size, 64, H, W)	0
	ReLU	(batch_size, 64, H, W)	(batch_size, 64, H, W)	0
lin1	Linear	(batch_size, 5184)	(batch_size, 1024)	5,306,624
lin2	Linear	(batch_size, 1024)	(batch_size, *num_classes*)	4,100

General Structure of the Block:

1. `ZeroPad2d`: This layer performs zero-padding on the input tensor, adding zeros around the borders to maintain spatial dimensions during convolution operations.
2. `Conv2d`: This layer applies a 2D convolution operation, extracting features using learnable filters.
3. `ReLU`: This activation function introduces non-linearity by applying the rectified linear unit (ReLU) operation element-wise to the output of the convolution layer.

The `CNN` class contains four blocks, denoted as `conv1`, `conv2`, `conv3`, and `conv4` in the code. The number of blocks used can vary based on the specific model design.

The purpose of `ZeroPad2d` is to control the spatial size of feature maps during convolutional operations. By adding zeros around the borders of the input tensor, the spatial dimensions are preserved, ensuring the output feature maps maintain the same size as the input.

In summary, the `CNN` class utilizes a block structure consisting of `ZeroPad2d`, `Conv2d`, and `ReLU` layers, allowing the model to learn hierarchical representations of the input data by progressively extracting more complex features.

- **FBCNet**

  FBCNet model [75] is specifically designed to extract important information from EEG-MI data, which contains valuable spectro-spatial discriminative patterns [Fig. 3.3]. It addresses the challenge of overfitting when working with small datasets. The architecture of FBCNet consists of four main stages:

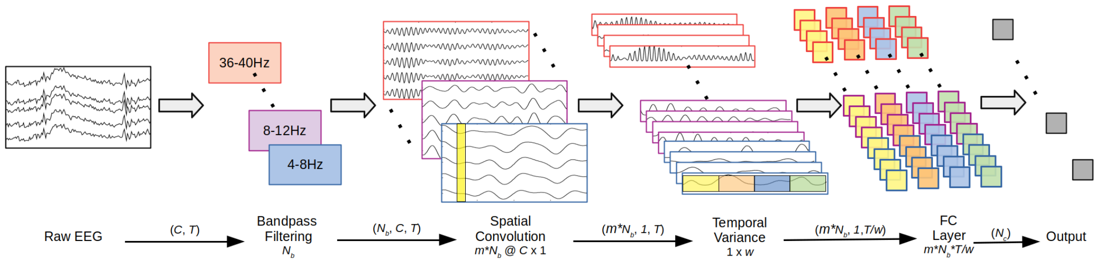


FIGURE 3.3: FBCNet [75]

1. Multi-view data representation: The raw EEG data is transformed into a multi-view representation by applying narrow-band filters that filter specific frequency ranges.
2. Spatial transformation learning: For each view, the model learns spatial discriminative patterns using a Depthwise Convolution layer. This layer helps identify important spatial features within each view.
3. Temporal feature extraction: After the spatial transformation, a novel Variance layer is employed to effectively capture the temporal information. This layer focuses on extracting essential temporal patterns from the data.
4. Classification: Finally, a fully connected (FC) layer is used to classify the features obtained from the Variance layer into the desired classes. This step assigns the input data to specific categories based on the extracted features.

By employing a multi-view EEG representation and spatial filtering, FBCNet is capable of extracting valuable spectro-spatial discriminative features. Furthermore, the variance layer provides a concise representation of the temporal information captured by the model.

* **FBCCNN** The FBCCNN model represents a Frequency Band Correlation Convolutional Neural Network (FBCCNN) model, designed for emotion recognition tasks based on EEG data. This model architecture is described in the paper "Emotion Recognition Based on EEG Using Generative Adversarial Nets and Convolutional Neural Network" by Pan B. and Zheng W. (2021) [76], detailed architecture layers [Tab. 3.4], while the general structure of the FBCCNN model is as follows:

TABLE 3.4: Layers and Parameters of the FBCCNN Model

Layer	Type	Input Shape	Output Shape	Parameters
block1	Conv2d	(batch_size, 4, 9, 9)	(batch_size, 12, 9, 9)	444
	ReLU	(batch_size, 12, 9, 9)	(batch_size, 12, 9, 9)	0
	BatchNorm2d	(batch_size, 12, 9, 9)	(batch_size, 12, 9, 9)	24
block2	Conv2d	(batch_size, 12, 9, 9)	(batch_size, 32, 9, 9)	3,488
	ReLU	(batch_size, 32, 9, 9)	(batch_size, 32, 9, 9)	0
	BatchNorm2d	(batch_size, 32, 9, 9)	(batch_size, 32, 9, 9)	64
block3	Conv2d	(batch_size, 32, 9, 9)	(batch_size, 64, 9, 9)	18,496
	ReLU	(batch_size, 64, 9, 9)	(batch_size, 64, 9, 9)	0
	BatchNorm2d	(batch_size, 64, 9, 9)	(batch_size, 64, 9, 9)	128
block4	Conv2d	(batch_size, 64, 9, 9)	(batch_size, 128, 9, 9)	73,856
	ReLU	(batch_size, 128, 9, 9)	(batch_size, 128, 9, 9)	0
	BatchNorm2d	(batch_size, 128, 9, 9)	(batch_size, 128, 9, 9)	256
block5	Conv2d	(batch_size, 128, 9, 9)	(batch_size, 256, 9, 9)	295,168
	ReLU	(batch_size, 256, 9, 9)	(batch_size, 256, 9, 9)	0
	BatchNorm2d	(batch_size, 256, 9, 9)	(batch_size, 256, 9, 9)	512
block6	Conv2d	(batch_size, 256, 9, 9)	(batch_size, 128, 9, 9)	295,040

Continued on next page

Table 3.4 – Continued from previous page

Layer	Type	Input Shape	Output Shape	Parameters
	ReLU	(batch_size, 128, 9, 9)	(batch_size, 128, 9, 9)	0
	BatchNorm2d	(batch_size, 128, 9, 9)	(batch_size, 128, 9, 9)	256
block7	Conv2d	(batch_size, 128, 9, 9)	(batch_size, 32, 9, 9)	36,896
	ReLU	(batch_size, 32, 9, 9)	(batch_size, 32, 9, 9)	0
	BatchNorm2d	(batch_size, 32, 9, 9)	(batch_size, 32, 9, 9)	64
lin1	Linear	(batch_size, 2592)	(batch_size, 512)	1,328,896
	ReLU	(batch_size, 512)	(batch_size, 512)	0
lin2	Linear	(batch_size, 512)	(batch_size, 128)	65,664
	ReLU	(batch_size, 128)	(batch_size, 128)	0
lin3	Linear	(batch_size, 128)	(batch_size, num_classes)	258

· The input EEG data is expected to have a shape of [n, 4, 9, 9], where n represents the batch size, 4 is the number of feature dimensions for each electrode, and (9, 9) is the spatial grid size of the EEG representation.

· The model consists of several convolutional blocks (block1 to block7), each followed by a ReLU activation function and batch normalization. These blocks perform feature extraction and learn hierarchical representations from the input EEG data.

· The output of the last convolutional block (block7) is flattened to a 1D tensor using the flatten function.

· The flattened tensor is then passed through fully connected layers (lin1, lin2, and lin3) with ReLU activation functions. These layers further process the extracted features and gradually reduce the dimensionality.

· The final fully connected layer (lin3) produces the predicted probabilities for each class, where the number of output units corresponds to the number of classes to predict.

Additionally, the FBCCNN class provides a property `feature_dim` that calculates the output dimensionality of the flattened tensor after passing through the convolutional blocks. This property is useful for downstream tasks that require the feature dimension of the model.

* **EEGNet**

  The EEGNet model represents a compact convolutional neural network (EEGNet) model designed for EEG-based brain-computer interfaces [77]. It is specifically recommended for use in emotion recognition tasks, detailed architecture layers [Tab. refEEGNet].The general structure of the EEGNet model is as follows:

Layer	Type	Input Shape	Output Shape	Parameters
block1	Conv2d	[batch_size, 1, 60, 151]	[batch_size, 8, 60, 151]	64
	BatchNorm2d	[batch_size, 8, 60, 151]	[batch_size, 8, 60, 151]	16
	Conv2d	[batch_size, 8, 60, 151]	[batch_size, 16, 60, 151]	512
	BatchNorm2d	[batch_size, 16, 60, 151]	[batch_size, 16, 60, 151]	32
	ELU	[batch_size, 16, 60, 151]	[batch_size, 16, 60, 151]	0
	AvgPool2d	[batch_size, 16, 60, 151]	[batch_size, 16, 60, 37]	0
	Dropout	[batch_size, 16, 60, 37]	[batch_size, 16, 60, 37]	0
block2	Conv2d	[batch_size, 16, 60, 37]	[batch_size, 16, 60, 37]	64
	Conv2d	[batch_size, 16, 60, 37]	[batch_size, 16, 60, 37]	256
	BatchNorm2d	[batch_size, 16, 60, 37]	[batch_size, 16, 60, 37]	32
	ELU	[batch_size, 16, 60, 37]	[batch_size, 16, 60, 37]	0
	AvgPool2d	[batch_size, 16, 60, 37]	[batch_size, 16, 60, 5]	0
	Dropout	[batch_size, 16, 60, 5]	[batch_size, 16, 60, 5]	0
-	Flatten	[batch_size, 16, 60, 5]	[batch_size, 4800]	0
num_classes	Linear	[batch_size, 4800]	[batch_size, num_classes]	9600

  · The model expects EEG data with a shape of [n, 60, 151], where n represents the batch size, 60 is the number of electrodes, and 151 is the number of data points in each EEG chunk.

- The model consists of two convolutional blocks (block1 and block2), each followed by an ELU activation function, batch normalization, and downsampling.
- The output of the last convolutional block (block2) is flattened to a 1D tensor using the flatten function.
- The flattened tensor is then passed through a linear layer (lin) to produce the predicted probabilities for each class. The number of output units corresponds to the number of classes to predict.

Additionally, the EEGNet class provides a property `feature_dim` that calculates the output dimensionality of the flattened tensor after passing through the convolutional blocks. This property can be useful for downstream tasks that require the feature dimension of the model.

* **ViT**

The Vision Transformer (ViT) [78] is a model architecture that applies the Transformer architecture, originally designed for natural language processing, to image recognition tasks. It has gained popularity and achieved competitive performance on various computer vision benchmarks.

Layer Name	**Layer Type**	**Input Size**	**Output Size**
PatchEmbedding	Sequential	(batch_size, C, H, W)	(batch_size, P, P, D)
Positional Embedding		(batch_size, P*P+1, D)	(batch_size, P*P+1, D)
Transformer Encoder	Transformer	(batch_size, P*P+1, D)	(batch_size, P*P+1, D)
MLPHead	Sequential	(batch_size, P*P+1, D)	(batch_size, num_classes)

TABLE 3.6: ViT Model Layers

The ViT model consists of several key components:

- **Patch Embedding**: The input image is divided into a grid of patches. Each patch is linearly projected to the desired feature dimension, transforming the image into a sequence of patches.
- **Positional Embedding**: Spatial information is added to the patch embeddings by concatenating a positional embedding vector to each

patch embedding. This provides the model with knowledge of the relative positions of the patches.

- **Transformer Encoder**: The patch embeddings, along with the positional embeddings, are processed by a stack of Transformer encoder layers. Each encoder layer comprises two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. The self-attention mechanism captures global dependencies within the image, while the feed-forward network applies non-linear transformations to each patch independently.
- **Pooling**: After the Transformer encoder layers, a pooling operation is applied to obtain a fixed-dimensional representation of the input image. This can be achieved by either selecting the classification-related token (CLS token) or performing average pooling over all the patch embeddings.
- **MLP Head**: The pooled representation is fed into a multi-layer perceptron (MLP) head, consisting of fully connected layers with non-linear activations. The MLP head maps the pooled representation to the output classes.

The self-attention mechanism is a crucial component of the ViT model, as it allows the model to capture global relationships between different patches in the image. By assigning weights to each patch based on its relevance to other patches, the model can attend to informative regions and understand the context and long-range dependencies within the image.

The ViT model also utilizes techniques such as layer normalization, residual connections, dropout regularization, and positional encoding to enhance stability during training and improve generalization capabilities.

∗ **DGCNN**

The DGCNN (Dynamical Graph Convolutional Neural Networks) model [79] is designed for EEG emotion recognition tasks. It leverages graph convolutional operations to capture spatial dependencies between different electrodes in EEG signals. Here is the general structure of the DGCNN model:

· **GraphConvolution:** This module performs graph convolution operations on the input EEG signals. It consists of a graph convolutional layer with learnable weights and an optional bias.

· **Linear:** This module applies a linear transformation to its input.

· **normalize_A:** This function normalizes the adjacency matrix A by applying ReLU activation, calculating the graph Laplacian matrix, and performing normalization based on whether symmetry is desired.

· **generate_cheby_adj:** This function generates a list of Chebyshev polynomials of the normalized adjacency matrix A, up to the specified number of layers.

· **Chebynet:** This module applies multiple graph convolution layers (GraphConvolution) with Chebyshev filters to the input features. It uses the generated Chebyshev polynomials as adjacency matrices for each layer.

· **DGCNN:** This is the main model class that integrates the above components. It takes input EEG signals, applies batch normalization to the signals, performs graph convolutional operations using Chebynet, and passes the resulting features through fully connected layers (Linear) to obtain the final classification output.

Importance of Graph Neural Networks (GNN) in DGCNN:
Graph Neural Networks, such as the DGCNN model, are essential for capturing and modeling relationships between graph-structured data, where nodes represent entities and edges represent connections. In the context of EEG emotion recognition, GNNs enable the DGCNN model to effectively learn and exploit the spatial dependencies between different electrodes. By leveraging graph convolutional operations, the DGCNN model can capture complex patterns and representations from

EEG signals, leading to improved emotion recognition performance.

3.3.3 DRL Part

This is a crucial part of the system design where the learning and creation of the Agent take place. Most DRL algorithms consist of two main parts:

– **Critic Network**

  The critic network estimates the expected future rewards (Q-values) for different state-action pairs. It helps the agent evaluate the quality of its actions.

– **Actor Network**

  The actor network selects actions based on the observed state, aiming to maximize the expected future rewards predicted by the critic network. The actor network takes observations from the lidar and emotion feedback, or one of them, as input.

  Next, we will present the pseudocode for each DRL algorithm [Alg. 1 2 3] in our implementation:

Algorithm 1 Our DDPG

```
1: Initialize actor network actor and critic network critic with random parameters
2: Initialize target actor network target_actor and target critic network target_critic
   with the same parameters as the actor and critic networks
3: Initialize replay buffer buffer
4: for episode = 1 to max_episodes do
5:     Initialize the environment state state
6:     for step = 1 to max_steps do
7:         Select an action action using the actor network and exploration noise
8:         Execute the action in the environment and observe the reward reward and
   the next state next_state
9:         Store the transition (state, action, reward, next_state) in the replay buffer
10:        Sample a random mini-batch of transitions from the replay buffer
11:        Update the critic network by minimizing the mean squared Bellman error
12:        Update the actor network using the sampled policy gradient
13:        Update the target networks by performing a soft update
14:        Set the current state state to the next state next_state
15:    end for
16: end for
```

According to these three algorithms, the actor network used in each algorithm contains two hidden neural network layers followed by an output

Algorithm 2 Our DQN

```
1: Initialize actor network actor with random parameters
2: Initialize target actor network target_actor with the same parameters as the actor network
3: Initialize replay buffer buffer
4: Initialize discount factor $\gamma$, target update frequency target_update_frequency
5: for episode = 1 to max_episodes do
6:     Initialize the environment state state
7:     while not done do
8:         Select an action action using the actor network and exploration strategy
9:         Execute the action in the environment and observe the reward reward and the next state next_state
10:        Store the transition (state, action, reward, next_state) in the replay buffer
11:        Sample a minibatch of transitions from the replay buffer
12:        Compute the target Q-values using the target actor network
13:        Compute the current Q-values using the actor network
14:        Compute the loss between the current Q-values and the target Q-values
15:        Update the actor network using gradient descent
16:        if step is a multiple of target_update_frequency then
17:            Update the target actor network by copying the parameters from the actor network
18:        end if
19:        Set the current state state to the next state next_state
20:    end while
21: end for
```

neural network layer. The activation function used for the hidden layers is relu, and the output activation function is Hyperbolic Tangent. The input size is the state size, and the output size is the action size.

On the other hand, the critic network consists of three hidden layers with the relu activation function, followed by an output layer. There are differences in the critic network among the three algorithms. DDPG has a standard critic network, while DQN does not have a critic network but rather double actor and target actor networks. TD3 has Q1 layers and Q2 layers. For more details, please refer to the provided code on GitHub.

Furthermore, we have an Agent container class for the algorithm represented as a class named DrlAgent. The pseudocode for the class is [Alg. 4]

– **reward Function**

The reward function has five versions named A, B, C, D, and E. Version A uses lidar as the observation, version B uses uniform distribution of emotion, version C combines uniform distribution of emotion with lidar, version D

Algorithm 3 Our TD3

```
Initialize actor network actor and target actor network target_actor with random parameters
Initialize critic network critic and target critic network target_critic with random parameters
Initialize replay buffer buffer
Initialize noise process noise
Initialize hyperparameters: policy noise policy_noise, noise clip noise_clip, policy update frequency policy_freq
for episode = 1 to max_episodes do
    Initialize the environment state state
    while not done do
        Select an action action using the actor network and exploration strategy
        Execute the action in the environment and observe the reward reward and the next state next_state
        Store the transition (state, action, reward, next_state) in the replay buffer
        Sample a minibatch of transitions from the replay buffer
        Compute the target Q-values using the target actor and target critic networks
        Compute the current Q-values using the critic network
        Compute the loss between the current Q-values and the target Q-values for both critics
        Update the critic networks using gradient descent
        if step is a multiple of policy_freq then
            Compute the loss for the actor network using the critic network
            Update the actor network using gradient ascent
            Update the target networks using a soft update
        end if
        Set the current state state to the next state next_state
    end while
end for
```

Algorithm 4 DrlAgent

```
procedure INITIALIZE
    Set algorithm, training, load_session, load_episode, train_stage
    Set is_training as int(training)
    Set episode as int(load_episode)
    Set num_episodes as NUM_EPISODES
    Set device as util.check_gpu()
    Set sim_speed as util.get_simulation_speed(train_stage)
    Initialize model based on algorithm with device and sim_speed
    Set replay_buffer as ReplayBuffer with model.buffer_size
    Set graph as Graph
    Set sm as StorageManager with algorithm, train_stage, load_session, episode, device
    Load model from sm if load_session is not empty, set device, load weights, and load
replay_buffer if is_training is True
    Create session directory in sm if load_session is empty and store model in sm
    Set graph.session_dir as sm.session_dir
    Set logger as Logger with relevant parameters
    Create step_comm_client, goal_comm_client, gazebo_pause, and gazebo_unpause
end procedure
procedure PROCESS
    Pause simulation
    while num_episodes > 0 do
        Set episode_done as False
        Set step, reward_sum, loss_critic, loss_actor as 0
        Set action_past as [0.0, 0.0]
        Set state using util.init_episode
        if ENABLE_STACKING is True then
            Set frame_buffer and next_state
        end if
        Unpause simulation
        Sleep for 0.5 seconds
        Set episode_start as time.perf_counter()
        while episode_done is False do
            Determine action based on is_training and total_steps
            Set action_current based on algorithm
            Take a step using util.step
            Update action_past, reward_sum, and other relevant variables
            if ENABLE_STACKING is True then
                Update frame_buffer and next_state
            end if
        end while
        num_episodes -= 1
    end while
end procedure
```

uses emotion only, and version E combines emotion with lidar. We will focus on the two main reward functions, while the other three are variations of these two main ones.

The first reward function [Alg. 5] is a hybrid function that utilizes lidar and emotion as observations. The second reward function [Alg. 6] employs a uniform distribution of emotion based on a given probability. Additionally, there is a variable called 'constant' that is added to the reward when the robot successfully avoids collisions with obstacles. This constant value varies based on four types: Step-1, Step-2, Episode, and time, which are defined in seconds. The constant is added according to one of these four types.

Algorithm 5 get_reward_E(succeed, action_linear, action_angular, goal_dist, goal_angle, min_obstacle_dist, emotion)

```
1:  $r\_yaw \leftarrow -1 \times \text{abs}(goal\_angle)$
2:  $r\_vangular \leftarrow -1 \times (action\_angular^2)$
3:  $r\_distance \leftarrow 2 \times \left(\frac{2 \times goal\_dist\_initial}{goal\_dist\_initial + goal\_dist} - 1\right)$
4:  $r\_vlinear \leftarrow -1 \times ((0.22 - action\_linear) \times 10)^2$
5:  if $min\_obstacle\_dist < 0.22$ then
6:      $r\_obstacle \leftarrow -4$
7:  else
8:      $r\_obstacle \leftarrow 0$
9:  end if
10: $reward \leftarrow \text{float}(r\_yaw) + \text{float}(r\_distance) + 4.0 \times \text{float}(emotion) + \text{float}(r\_vlinear) + \text{float}(r\_vangular) - 1.0 + \text{float}(r\_obstacle)$
11: if $succeed = SUCCESS$ then
12:     $reward \leftarrow reward + 2500.0$
13: else if $succeed = COLLISION\_OBSTACLE$ or $succeed = COLLISION\_WALL$ then
14:     $reward \leftarrow reward - 2500.0$
15: end if
16: return float(reward)
```

3.3.4 System Settings

The global system settings in the project can be explained as follows. Please refer to [Tab. 3.7] for a comprehensive overview of these settings. The table provides detailed descriptions of each parameter, allowing you to understand their purpose and configuration options.

Algorithm 6 get_reward_C(succeed, action_linear, action_angular, goal_dist, goal_angle, min_obstacle_dist, emotion)

```
1: global cnt, samples_sorted
2: $cnt \leftarrow cnt + 1$
3: $reward \leftarrow samples\_sorted[cnt]$
4: if STEP_1 then
5:     $reward \leftarrow reward + CONSTANT$
6: end if
7: if STEP_5 and (cnt + 1) % 5 == 0 then
8:     $reward \leftarrow reward + CONSTANT$
9: end if
10: if TIME then
11:     global last_increment_time, increment_interval
12:     $current\_time \leftarrow$ time.time()
13:     $time\_since\_last\_increment \leftarrow current\_time - last\_increment\_time$
14:     if $time\_since\_last\_increment \geq increment\_interval$ then
15:         $reward \leftarrow reward + CONSTANT$
16:         $last\_increment\_time \leftarrow current\_time$
17:     end if
18: end if
19: $r\_yaw \leftarrow -1 \times \text{abs}(goal\_angle)$
20: $r\_vangular \leftarrow -1 \times (action\_angular^2)$
21: $r\_distance \leftarrow 2 \times \left(\frac{2 \times goal\_dist\_initial}{goal\_dist\_initial + goal\_dist} - 1\right)$
22: $r\_vlinear \leftarrow -1 \times ((0.22 - action\_linear) \times 10)^2$
23: if $min\_obstacle\_dist < 0.22$ then
24:     $r\_obstacle \leftarrow -4$
25: else
26:     $r\_obstacle \leftarrow 0$
27: end if
28: if $succeed = SUCCESS$ then
29:     $reward \leftarrow reward + 2500$
30: else if $succeed = COLLISION\_OBSTACLE$ or $succeed = COLLISION\_WALL$ then
31:     $reward \leftarrow reward - 2500$
32: end if
33: return $reward + \text{float}(r\_yaw) + \text{float}(r\_distance) + \text{float}(r\_vlinear) + \text{float}(r\_vangular) - 1.0$
```

TABLE 3.7: Parameter Descriptions

Parameter	Description
General Parameters	
ENABLE_BACKWARD	A boolean parameter indicating whether backward movement is enabled.
ENABLE_STACKING	A boolean parameter indicating whether stacking of subsequent frames is enabled.
ENABLE_VISUAL	A boolean parameter indicating whether visualizations are enabled, typically used during evaluation or testing phases.
ENABLE_TRUE_RANDOM_GOALS	A boolean parameter indicating whether goals are randomly selected from a list of known valid positions or not.
MODEL_STORE_INTERVAL	The interval (number of episodes) at which the model weights are stored.
DRL Parameters	
ACTION_SIZE	The size of the action space.
HIDDEN_SIZE	The number of neurons in the hidden layers of the neural network.
NUM_EPISODES	The total number of episodes to run.
BATCH_SIZE	The number of samples per training batch.
BUFFER_SIZE	The maximum number of samples stored in the replay buffer.
DISCOUNT_FACTOR	The discount factor for future rewards in the reinforcement learning algorithm.
LEARNING_RATE	The learning rate for the optimizer used in the training process.
TAU	The rate at which the target network is updated in the DRL algorithm.
DRL Algorithm Parameters	

Continued on next page

Table 3.7 – *Continued from previous page*

Parameter	**Description**
`OBSERVE_STEPS`	The number of initial steps where random actions are taken to explore the environment.
`STEP_TIME`	The delay (in seconds) between each step in the simulation.
`EPSILON_DECAY`	The decay factor for the epsilon-greedy exploration strategy.
`EPSILON_MINIMUM`	The minimum value for the exploration factor epsilon.
DQN Algorithm Parameters	
`DQN_ACTION_SIZE`	The size of the action space specific to DQN.
`TARGET_UPDATE_FREQUENCY`	The frequency at which the target network is updated.
DDPG and TD3 Algorithm Parameters	
`POLICY_NOISE`	The standard deviation of the Gaussian noise added to the actions during training.
`POLICY_NOISE_CLIP`	The maximum absolute value of the noise added to the actions during training.
`POLICY_UPDATE_FREQUENCY`	The frequency at which the actor network is updated.
Environment and Simulation Parameters	
`REWARD_FUNCTION`	The chosen reward function defined in the "reward.py" file.
`EPISODE_TIMEOUT_SECONDS`	The maximum duration (in seconds) of an episode before it times out.
`ENABLE_MOTOR_NOISE`	A boolean parameter indicating whether normally distributed noise is added to motor outputs to simulate hardware imperfections.
Stacking Frames Parameters	
`STACK_DEPTH`	The number of subsequent frames processed per step.
`FRAME_SKIP`	The number of frames skipped between subsequent frames.
Reward Function Parameters	
`SIZE`	The number of samples for the reward function.
`P1`, `P2`, `P3`	The weights for negative, neutral, and positive emotions in the reward function.

Continued on next page

Table 3.7 – *Continued from previous page*

Parameter	**Description**
`EPSILON`	A small value used for calculations.
`CONSTANT`	A constant value added every time the robot avoids collision.
`TIME_REWARD`	The amount of reward added every value seconds.
`STEP_1, STEP_5, EPISODE_1, TIME`	Boolean parameters controlling the type of constant return.

These parameters provide control and configuration options for various aspects of the global system in the project.

3.4 Experiments

3.4.1 Experimental Setup

- **Software Used**
  - **Programming Languages**
    - Python: A versatile and high-level programming language known for its simplicity and readability, widely used in various domains including web development, data analysis, and artificial intelligence.
    - C++: A powerful and efficient programming language commonly used for system programming and performance-critical applications, known for its low-level control and high execution speed.
  - **Framework**
    - Gazebo: An open-source 3D physics-based simulator widely used for robotics and autonomous systems simulations, offering realistic simulation environments and physics modeling [Fig. 3.4].

FIGURE 3.4: Gazebo

- ROS2 (Robot Operating System 2): A flexible framework for developing robotic systems, providing tools and libraries for communication, control, and integration of robot components. It supports multiple programming languages, including Python and C++, allowing developers to write ROS2 nodes using their language of choice [Fig. 3.5].

FIGURE 3.5: ROS2

- **Libraries**
  - PyTorch: An open-source machine learning library known for its dynamic computation graphs and extensive support for deep neural networks. It provides a flexible framework for building and training various machine learning models [Fig. 3.6].

FIGURE 3.6: Pytroch

  - scikit-learn (sklearn): A machine learning library that provides efficient tools for data mining, data analysis, and machine learning algorithms. It offers a wide range of supervised and unsupervised learning techniques, as well as data preprocessing and evaluation tools [Fig. 3.7].

FIGURE 3.7: Sklearn

· NumPy: A fundamental library for scientific computing in Python, providing powerful array manipulation and numerical operations. It is widely used for tasks such as linear algebra, Fourier transforms, and random number generation [Fig. 3.8].

FIGURE 3.8: NumPy

· Pandas: A data analysis and manipulation library for Python, offering versatile data structures (such as DataFrames) and tools for handling structured data. It simplifies tasks like data cleaning, transformation, and analysis [Fig. 3.9].

FIGURE 3.9: Pandas

· MNE: MNE-Python is an open-source Python tool for exploring, displaying, and interpreting human neurophysiological data, including MEG, EEG, sEEG, and ECoG, among others [Fig. 3.10].

FIGURE 3.10: MNE

· Matplotlib: A plotting library for Python, allowing the creation of static, animated, and interactive visualizations. It provides a wide range of plots and customization options for presenting data in a visually appealing manner [Fig. 3.11].

matplotlib

FIGURE 3.11: Matplotlib

– **Material Used**

The following materials were used in the experiments:

∗ **E**EG Headset: We used the Emotiv EPOC+ wireless neuroheadset, which features 14 sensors and two references. It captures raw electrical signals from the scalp, allowing real-time detection of thoughts, emotions, and facial expressions. The EEG signals were recorded at a sampling rate of 128 Hz using the EMOTIV-PRO app [Fig. 3.12].

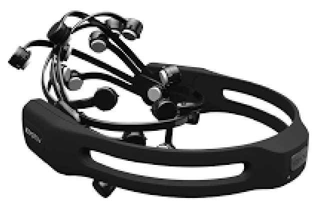


FIGURE 3.12: EEG Headset

∗ **E**EG Electrodes Gel: For optimal electrical conductivity, we utilized Bio True gel (BAUSCH + LOMB) to ensure reliable measurements. This gel was applied between the EEG electrodes and the skin to establish a strong electrical connection [Fig. 3.13].

FIGURE 3.13: EEG Electrodes Gel

∗ **H**ardware:

· RAM: 8 GB

    - CPU: Intel Core i5 8th generation
    - Operating System: Ubuntu 20.04
  - **Kaggle Notebook:**
    - RAM: 16 GB
    - CPU: Intel(R) Xeon(R) CPU @ 2.30GHz
    - Operating System: The Kaggle notebook operates on a cloud-based environment and does not disclose the specific underlying operating system.
    - GPU: NVIDIA GPU P100 was used for training the models in the Kaggle notebook.

3.4.2 Dataset

- **EEG Dataset**

  The SEED IV dataset [80] provides a comprehensive collection of EEG signals, focusing on emotion recognition using EEG and eye movement data. This subsection presents an overview of the EEG part of the SEED IV dataset used in this thesis. It includes information about the number of subjects, the film clips used, and the available classes. Furthermore, details about the preprocessing of the EEG signals and the feature extraction process are provided.

  - **Dataset Overview**

    The EEG dataset consists of recordings from 15 subjects who participated in the experiment. Each subject completed three sessions on different days, with each session comprising 24 trials. The film clips shown during the trials were carefully chosen to induce happiness, sadness, fear, or a neutral emotional state. Thus, the dataset contains four emotional classes: happiness, sadness, fear, and neutrality.

  - **EEG Preprocessing and Feature Extraction**

    In the EEG preprocessing stage, various steps were applied to prepare the data for further analysis. The details of these steps are summarized in [Tab. 3.8]. The steps include downsampling the raw EEG data to a sampling rate of 200 Hz and applying a bandpass filter from 1 Hz to 75 Hz to remove noise.

TABLE 3.8: EEG Preprocessing

Step	**Description**
Downsampling	Reduce the sampling rate of the raw EEG data to 200 Hz
Bandpass Filtering	Apply a bandpass filter (1 Hz - 75 Hz) to remove noise

Following the preprocessing stage, EEG feature extraction was performed to capture relevant information from the data. The features extracted include Power Spectral Density (PSD) and Differential Entropy (DE), as outlined in [Tab. 3.9]. PSD measures the power distribution across five frequency bands, namely Delta (1 Hz - 4 Hz), Theta (4 Hz - 8 Hz), Alpha (8 Hz - 14 Hz), Beta (14 Hz - 31 Hz), and Gamma (31 Hz - 50 Hz). On the other hand, DE calculates the differential entropy within each segment and across the aforementioned frequency bands.

TABLE 3.9: EEG Feature Extraction

Feature	**Description**
Power Spectral Density (PSD)	Compute the power distribution across five frequency bands: Delta (1 Hz - 4 Hz), Theta (4 Hz - 8 Hz), Alpha (8 Hz - 14 Hz), Beta (14 Hz - 31 Hz), and Gamma (31 Hz - 50 Hz)
Differential Entropy (DE)	Calculate the differential entropy within each segment and across the five frequency bands mentioned above

* **Utilization of EEG Feature Data**

  In this thesis, the `eeg_feature_smooth` folder of the SEED IV dataset was utilized for feature extraction. Specifically, the `PSD` and `DE` features were employed. The data from each `.mat` file were loaded and transformed into a format suitable for model training. To enhance the training process, the samples from each `.mat` file were concatenated into batches of size 128, where each batch contained EEG segments from different trials of the same subject. Additionally, one label from the four emotional classes was assigned to each batch. These batches were then used to create a dataloader, which was further split into training, validation, and testing sets with ratios of 0.8, 0.1, and 0.1, respectively. The dataset was shuffled to ensure randomness in the training process.

By utilizing this approach, the thesis aimed to train a model capable of recognizing emotions based on EEG features extracted from the SEED IV dataset.

– **Simulation Feedback Data**

The dataset used in this study case consists of environmental data obtained from a LIDAR sensor and feedback from an EEG (electroencephalogram) device. The LIDAR sensor provides information about the surroundings, such as distances to obstacles and the orientation of the robot. The EEG device measures the brain activity of the operator and provides feedback on their cognitive state.

3.4.3 hyperparameters

– **EEG Training hyperparameters**

In the classification models, there are shared implementation parameters. First and foremost, I would like to mention that the generated experiments for these models are deterministic. This means that I have fixed the seed for generating random numbers during the training of each model, ensuring consistent results if the models are retrained. Additionally, all models utilize the same loss function called CrossEntropy. The batch size is set to 128, and the number of epochs for all models is fixed at 100.

Furthermore, each model undergoes three trials with different initialization weights, based on the seed number. This approach allows us to obtain the average results and their standard deviation for better performance evaluation. For instance, the CNN model is trained three times with varying initialization weights, ensuring a comprehensive analysis.

Considering these settings, several general parameters are fixed across all models. We utilize Adam optimization, which has been theoretically and practically proven to be superior to gradient descent. This optimization algorithm optimizes the gradient descent steps. Additionally, the learning rate for all models is fixed at 0.0001.

Overall, these standardized settings and shared implementation parameters contribute to a robust and consistent evaluation of the models' performance.

– **DRL training hyperparameters**

  In this subsection, we provide the specific values for the implementation settings of the system:

  * **Model Settings:**
    - `ENABLE_BACKWARD`: `False`
    - `ENABLE_STACKING`: `False`
    - `ENABLE_VISUAL`: `False`
    - `ENABLE_TRUE_RANDOM_GOALS`: `True`
    - `MODEL_STORE_INTERVAL`: `100`
  * **DRL Parameters:**
    - `ACTION_SIZE`: `2`
    - `HIDDEN_SIZE`: `512`
    - `NUM_EPISODES`: `1000`
    - `BATCH_SIZE`: `128`
    - `BUFFER_SIZE`: `1000000`
    - `DISCOUNT_FACTOR`: `0.99`
    - `LEARNING_RATE`: `0.003`
    - `TAU`: `0.003`
  * **DQN Parameters:**
    - `DQN_ACTION_SIZE`: `5`
    - `TARGET_UPDATE_FREQUENCY`: `1000`
  * **TD3 Parameters:**
    - `POLICY_NOISE`: `0.2`
    - `POLICY_NOISE_CLIP`: `0.5`
    - `POLICY_UPDATE_FREQUENCY`: `2`

* **drl_environment Settings:**
  - `EPISODE_TIMEOUT_SECONDS`: 30
  - `ENABLE_MOTOR_NOISE`: `True`
* **Stacking Settings:**
  - `STACK_DEPTH`: 3
  - `FRAME_SKIP`: 4

3.4.4 Simulation Challenges in Maze Environment

In this subsection, we discuss the various simulation challenges encountered in the maze environment. These challenges are designed to test the capabilities and performance of the robot in different scenarios. A screenshot depicting the nine different challenges is provided in Figure [Fig. 3.14].

- **Challenge 1: Square Maze**

  The first challenge presents a simple square maze where the robot is placed at the center. This challenge serves as the baseline for evaluating the robot's navigation abilities.

- **Challenge 2: Square Maze with Dynamic Obstacles**

  Challenge 2 builds upon the previous one by introducing four dynamic obstacles within the square maze. The presence of these obstacles adds complexity to the robot's path planning and obstacle avoidance strategies.

- **Challenge 3: Varied Speed and Movement of Dynamic Obstacles**

  In this challenge, the dynamic obstacles exhibit different speeds and movement patterns compared to Challenge 2. The robot must adapt its navigation strategy accordingly to avoid collisions.

- **Challenge 4: Maze with Inner Walls and Two Dynamic Obstacles**

  Challenge 4 incorporates inner walls within the maze structure and introduces two dynamic obstacles. The presence of inner walls further complicates the robot's path planning process.

- **Challenge 5: Maze with Inner Walls and Six Dynamic Obstacles**

Similar to Challenge 4, this challenge includes inner walls, but with an increased number of dynamic obstacles (six in total). The robot's ability to handle a higher density of obstacles is tested here.

– **Challenge 6: Maze with Dynamic Obstacles (No Inner Walls)**

  Challenge 6 removes the inner walls from Challenge 5 while keeping the same configuration of six dynamic obstacles. This challenge evaluates the robot's performance when navigating in an open maze environment.

– **Challenge 7: Larger Maze with More Inner Walls**

  Challenge 7 presents a larger maze with an increased number of inner walls. The robot must navigate through narrower passages and overcome more complex maze structures.

– **Challenge 8: Larger Maze with Inner Walls and Two Dynamic Obstacles**

  Building upon Challenge 7, Challenge 8 adds two dynamic obstacles to the maze. This challenge combines the difficulties of navigating in a larger maze with the presence of moving obstacles.

– **Challenge 9: Larger Maze with Inner Walls and Six Dynamic Obstacles**

  Lastly, Challenge 9 maintains the larger maze and inner walls from Challenge 8 but increases the number of dynamic obstacles to six. This challenge presents a highly demanding scenario, testing the robot's ability to handle complex maze structures and a high density of dynamic obstacles.

Overall, these nine challenges encompass a range of maze configurations and obstacle arrangements, progressively increasing in difficulty. They allow for a comprehensive evaluation of the robot's navigation and obstacle avoidance capabilities. Figure [Fig. 3.14] provides a visual representation of the different challenges discussed.

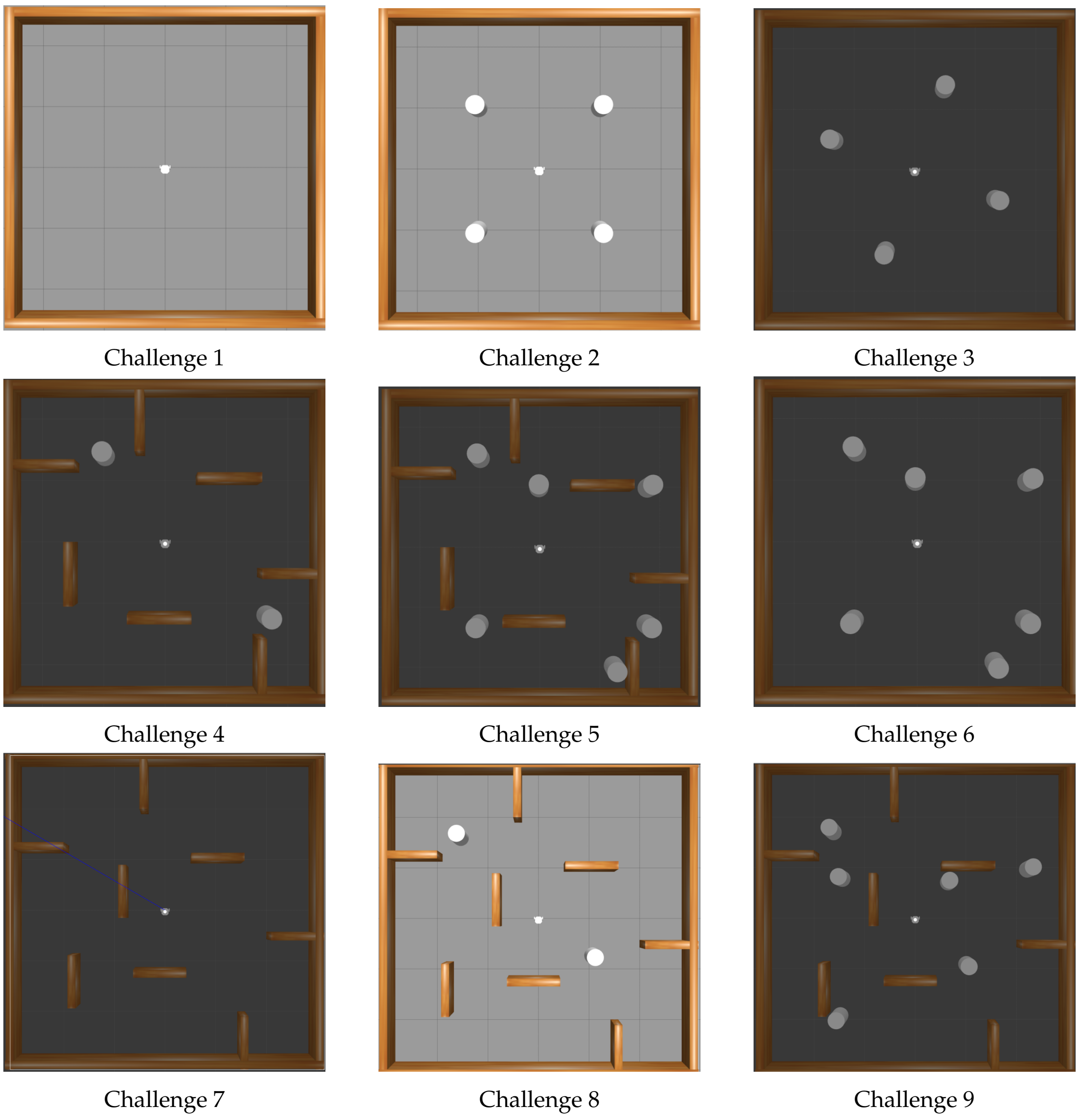


FIGURE 3.14: Simulation Challenges in Maze Environment

3.5 Results

3.5.1 EEG Results

– **Summary Results**

According to the table provided [Tab. 3.10,3.11, which presents the results of the models trained on the SEED IV dataset, each model was trained three times, and the average and variance of the scores were calculated. The best-performing models in terms of accuracy, with results exceeding 90%, are DGCNN and ViT.

DGCNN (Dynamic Graph Convolutional Neural Network) achieved an average training accuracy of 99.52% and an average validation accuracy of 99.03%. ViT (Vision Transformer) achieved an average training accuracy of 99.67% and an average validation accuracy of 98.60%.

The superior performance of DGCNN and ViT can be attributed to their architectural characteristics and capabilities. DGCNN utilizes graph convolutional layers that can effectively capture and model complex relationships and dependencies within the data. This makes it particularly suitable for tasks involving graph-structured data like the SEED IV dataset. Additionally, the high accuracy achieved by DGCNN indicates its ability to generalize well to unseen data.

ViT, on the other hand, is a transformer-based model originally designed for image classification tasks. It utilizes self-attention mechanisms to capture global dependencies and relationships within the input data, which can be advantageous for capturing patterns and features in the SEED IV dataset. The strong performance of ViT suggests that it is able to effectively learn and represent the relevant EEG signals for emotion recognition.

In summary, the DGCNN and ViT models outperformed the other models in terms of accuracy in the SEED IV dataset. The success of these models can be attributed to their architectural design and ability to capture complex relationships and dependencies within the data.

– **Our Models vs State of The Art Models**

In the provided table, we compare the accuracy of models trained by us (marked with $*$) and other referenced models. Certain features contributed to achieving the reported results. Let's analyze the performance and role of features.

Among the models trained by us, the DGCNN and ViT models stood out in terms of accuracy. The DGCNN model achieved a test accuracy of 99.19% (with a standard deviation of 5×10^{-6}), while the ViT model achieved a test accuracy of 99.02% (with a standard deviation of 2×10^{-6}). These models demonstrated superior performance compared to the others.

TABLE 3.10: Model Performance (Part 1) (The models with $*$ we trained them, while the other referenced models are state-of-the-art models taken the accuracy from the referenced papers most of the model trained for 5 to 15 trials)

Model	Train Loss	Train Accuracy (%)	Validation Accuracy (%)	Test Accuracy (%)
GRU$*$	$0.0063 \pm 4 \times 10^{-8}$	$67.15 \pm 8 \times 10^{-8}$	$65.05 \pm 1 \times 10^{-7}$	$65.9 \pm 3 \times 10^{-6}$
LSTM$*$	$0.0062 \pm 2 \times 10^{-8}$	$68.02 \pm 6 \times 10^{-8}$	$65.33 \pm 1 \times 10^{-7}$	$68.2 \pm 2 \times 10^{-6}$
CNN$*$	$0.0108 \pm 5 \times 10^{-8}$	$27.54 \pm 3 \times 10^{-8}$	$26.87 \pm 2 \times 10^{-7}$	$28.32 \pm 3 \times 10^{-6}$
FBCNet$*$	$0.0098 \pm 6 \times 10^{-8}$	$39.645 \pm 6 \times 10^{-8}$	$38.645 \pm 2 \times 10^{-7}$	$38.5 \pm 5 \times 10^{-6}$
FBCCNN$*$	$0.0001 \pm 5 \times 10^{-8}$	$23.28 \pm 3 \times 10^{-8}$	$23.83 \pm 8 \times 10^{-8}$	$28.33 \pm 3 \times 10^{-6}$
ViT$*$	$0.0003 \pm 5 \times 10^{-8}$	$99.67 \pm 5 \times 10^{-8}$	$98.60 \pm 2 \times 10^{-7}$	$\mathbf{99.02} \pm 2 \times 10^{-6}$
DGCNN$*$	$0.0001 \pm 5 \times 10^{-8}$	$99.52 \pm 8 \times 10^{-8}$	$99.03 \pm 8 \times 10^{-8}$	$\mathbf{99.19} \pm 5 \times 10^{-6}$
CAN [24]	/	/	/	87.71 ± 9.74
SVM [81]	/	/	/	75.88 ± 16.14
PR-PL [82]	/	/	/	85.56 ± 4.78
DCCA [83]	/	/	/	87.45 ± 9.23

TABLE 3.11: Model Performance (Part 2)

Model	Precision (%)	Recall (%)	F1 Score (%)	Time per Epoch (seconds)
GRU$*$	$66.0 \pm 2 \times 10^{-7}$	$66.3 \pm 6 \times 10^{-6}$	$65.9 \pm 2 \times 10^{-6}$	$13.303 \pm 1 \times 10^{-6}$
LSTM$*$	$67.7 \pm 1 \times 10^{-7}$	$68.8 \pm 4 \times 10^{-6}$	$68.0 \pm 1 \times 10^{-6}$	$13.370 \pm 9 \times 10^{-7}$
CNN$*$	$28.0 \pm 3 \times 10^{-6}$	$28.0 \pm 1 \times 10^{-6}$	$28.1 \pm 2 \times 10^{-6}$	$18.93 \pm 1 \times 10^{-6}$
FBCNet$*$	$39.6 \pm 2 \times 10^{-7}$	$40.6 \pm 4 \times 10^{-6}$	$38.6 \pm 3 \times 10^{-6}$	$13.8146 \pm 1 \times 10^{-6}$
FBCCNN$*$	$68.0 \pm 3 \times 10^{-6}$	$26.0 \pm 1 \times 10^{-6}$	$26.8 \pm 1 \times 10^{-6}$	$18.8471 \pm 7 \times 10^{-7}$
ViT$*$	$98.5 \pm 3 \times 10^{-6}$	$99.2 \pm 2 \times 10^{-6}$	$98.9 \pm 2 \times 10^{-6}$	$17.6572 \pm 1 \times 10^{-6}$
DGCNN$*$	$98.5 \pm 3 \times 10^{-6}$	$99.2 \pm 1 \times 10^{-6}$	$98.9 \pm 1 \times 10^{-6}$	$13.3278 \pm 9 \times 10^{-7}$

Regarding the influence of features, the use of specific features such as ’de_movingAve’, ’de_LDS’, ’psd_movingAve’, and ’psd_LDS’ for all rhythm waves generally led to better performance.

It’s important to note that the DGCNN and ViT models had better architecture compared to other models trained by us, such as CNN, LSTM, and GRU. However, these models had very low accuracy due to not performing grid search, lacking hyperparameter fine-tuning, and being trained for only 100 epochs.

Comparing our models with the referenced models, our models exhibited higher accuracy. The CAN model achieved an accuracy of 87.71% (with a standard deviation of 9.74), SVM achieved 75.88% (with a standard deviation of 16.14), PR-PL achieved 85.56% (with a standard deviation of 4.78), and DCCA achieved 87.45% (with a standard deviation of 9.23). However, the DGCNN and ViT models trained by us outperformed these referenced models with significantly higher accuracy.

– **Analysis of DGCNN Training and Validation Results**

According to the training accuracy and validation accuracy plots of DGCNN [Fig. 3.15], the plots show that they are moving together, with the training accuracy slightly better than the validation accuracy. However, the validation plot is not significantly different from the training plot. This indicates that our model learns well, can generalize effectively, and demonstrates stable learning in the first epochs. The convergence of the plots starts from epoch 80.

Furthermore, the loss value decreases as the model progresses through epochs, indicating that the model is learning and improving its predictions. The training accuracy increases with each epoch, suggesting that the model is getting better at predicting the training data. It starts at 48.08% and gradually improves to 99.26%.

Similarly, the validation accuracy follows a similar trend to the training accuracy. It starts at 46.58% and increases to 98.67%, indicating that the model is generalizing well and performing consistently on unseen data.

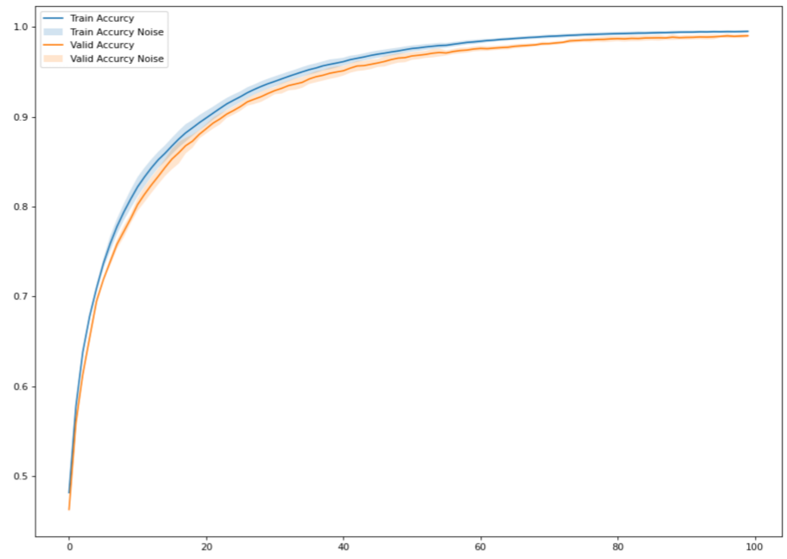


FIGURE 3.15: Train and Validation accuracy plots for 3 DGCNN model representing the average with variance

The final test accuracy also improves over the epochs, starting at 47.63% and reaching 98.64%. This metric provides an overall measure of the model's performance on the test set.

In summary, the training and validation accuracies of the DGCNN model move together, with the training accuracy slightly better. The model demonstrates stable learning, starts converging from epoch 80, and shows improvement in loss, training accuracy, validation accuracy, and final test accuracy over the epochs. These results indicate that the model learns well, can generalize effectively, and achieves high accuracy on both the training and validation datasets.

– **Analysis of ViT Training and Validation Results**

The Vision Transformer (ViT) model [Fig. 3.16] was trained and evaluated over 100 epochs, and the following results were obtained:

During the initial epochs, the model's performance gradually improved, with the training accuracy starting at 29.37% and the validation accuracy at 28.45% in the first epoch. The final test accuracy at this point was 29.06%.

As the training progressed, the model's accuracy continued to increase. By the 50th epoch, the training accuracy reached 69.12%, and the validation and final test accuracies were 67.21% and 66.76% respectively.

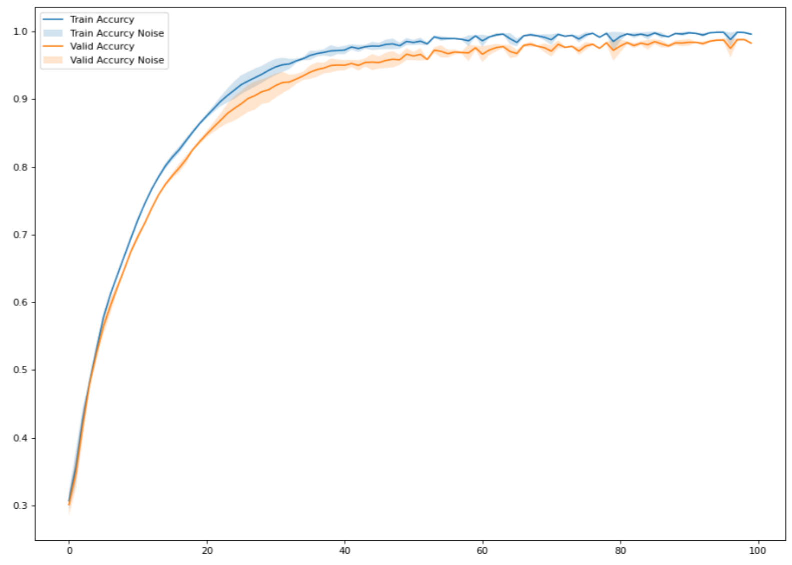


FIGURE 3.16: Train and Validation accuracy plots for 3 ViT model representing the average with variance

Throughout the remaining epochs, the model's accuracy consistently improved. The training accuracy continued to increase and reached 99.97% by the 86th epoch. The validation and final test accuracies also saw significant improvements, reaching 98.88% and 98.88% respectively.

These results indicate that the ViT model effectively learned from the training data and generalized well to the validation and final test sets. The model's high accuracy on the final test set suggests that it has achieved a good level of performance and can make accurate predictions on unseen data.

The loss values decreased steadily during training, indicating that the model was able to minimize the discrepancy between its predictions and the ground truth labels. This reduction in loss signifies the model's increasing ability to fit the training data.

In summary, the ViT model demonstrated consistent improvement in accuracy throughout the training process, achieving high performance on both the validation and final test sets. Its ability to learn complex patterns from image data and make accurate predictions highlights the effectiveness of the Vision Transformer architecture.

– **Analysis of Loss Curves and Convergence**

Throughout the 100 epochs, both the loss curves of DGCNN and ViT exhibit remarkable similarity [Fig. 3.17], with minimal variance observed across multiple trials. This indicates that the models consistently converge towards similar local minima, resulting in highly accurate predictions.

In the case of DGCNN, the loss curve demonstrates a gradual decline, starting from an initial high value and steadily decreasing over the training process. This signifies that the model effectively learns and adjusts its parameters to fit the training data better. After 100 epochs, DGCNN achieves an impressive loss value of 0.0005, indicating a high level of accuracy and strong predictive capability.

Similarly, the loss curve for ViT showcases a similar trend of continuous reduction. As the model progresses through the epochs, the loss steadily decreases, suggesting effective learning and adaptation to the training data. After 100 epochs, ViT attains a final loss of approximately 0.0001, further validating its ability to converge towards highly accurate predictions.

The minimal variance in the loss values across different trials for both models demonstrates the consistency in finding similar local minima. This consistency highlights the stability and reliability of the solutions obtained by both DGCNN and ViT.

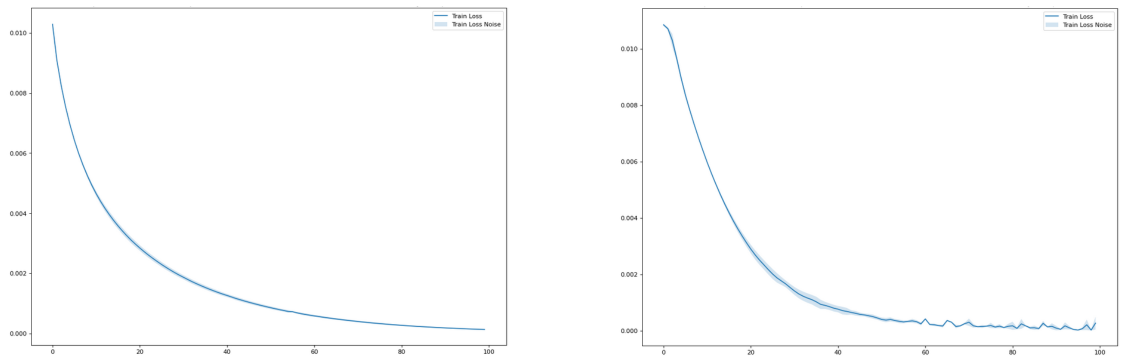
In summary, the analysis of the loss curves for both DGCNN and ViT reveals their remarkable ability to minimize discrepancies between predictions and ground truth values, resulting in highly accurate models. Both models exhibit a strong capacity to learn and converge towards reliable solutions, making them valuable tools for various applications.

– **Analysis of Confusion Matrices of The Best Models**

The confusion matrices provide valuable insights into the performance of classification models. Let's analyze the results for both ViT and DGCNN models based on the given confusion matrices [Fig. 3.18].

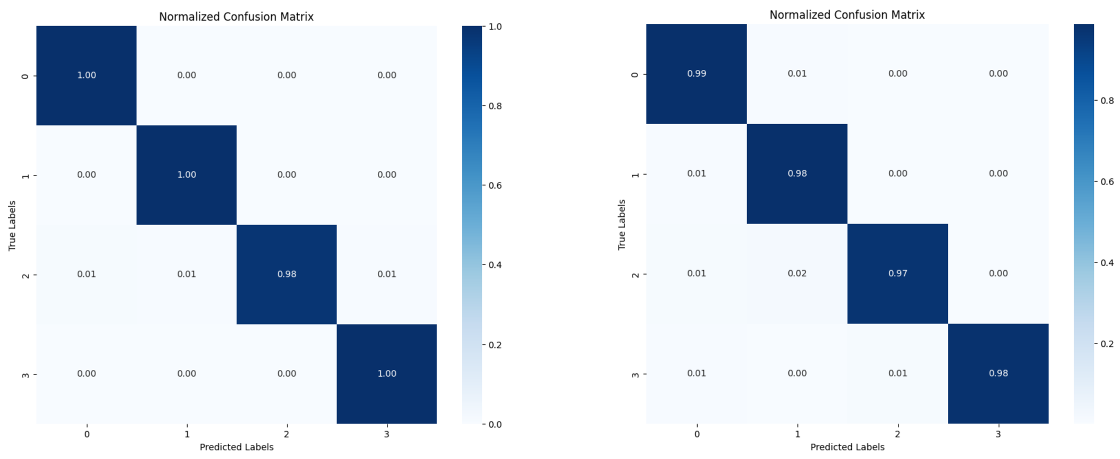
ViT Model:

* **Class 0 (Happiness):** The model performs exceptionally well in classifying happiness, with a high accuracy of 0.99. Only a small percentage (0.01) of the samples in this class are misclassified as other emotions.



(A) DGCNN (B) ViT

FIGURE 3.17: Loss Plot Over 3 Models Ploting The Average With Min And Max Noise



(A) DGCNN (B) ViT

FIGURE 3.18: Confusion Matrix for DGCNN and ViT

- **Class 1 (Sadness):** Similarly, the ViT model achieves a high accuracy of 0.98 in identifying sadness. It misclassifies only a negligible percentage (0.02) of the samples in this class.
- **Class 2 (Fear):** The ViT model demonstrates strong performance in detecting fear, with an accuracy of 0.97. However, it misclassifies a small portion (0.01) of the samples in this category as other emotions.
- **Class 3 (Neutral):** The ViT model shows excellent accuracy in recognizing neutral emotions, with a value of 0.98. Only a small percentage (0.01) of the samples in this class are misclassified.

Overall, the ViT model exhibits impressive classification results, with high accuracies across all emotion classes. It displays a strong ability to differentiate between different emotions, particularly in identifying happiness and neutral emotions.

DGCNN Model:

- **Class 0 (Happiness):** The DGCNN model achieves perfect accuracy (1.00) in classifying happiness, indicating no misclassifications in this category.
- **Class 1 (Sadness):** Similarly, the DGCNN model demonstrates flawless performance in identifying sadness, with a precision of 1.00.
- **Class 2 (Fear):** The DGCNN model performs well in detecting fear, with an accuracy of 0.98. However, it misclassifies a small portion (0.01) of the samples in this class as other emotions.
- **Class 3 (Neutral):** The DGCNN model achieves perfect accuracy (1.00) in recognizing neutral emotions, implying no misclassifications in this category.

The DGCNN model showcases exceptional classification results, with perfect accuracies in identifying happiness, sadness, and neutral emotions. It also performs admirably in detecting fear, with only a minor misclassification rate.

In summary, both the ViT and DGCNN models exhibit strong classification capabilities across all emotion classes. The ViT model achieves slightly lower accuracies compared to the DGCNN model but still demonstrates remarkable performance. Both models provide reliable predictions, making them valuable tools for emotion classification tasks.

3.5.2 DRL Case Studies

– **Constant Study Case: Finding the Optimal Value**

In this study case, we examine the effect of a constant parameter in our robot's reward function when it encounters static or dynamic obstacles. We generate emotions from a uniform distribution, with three possible states: positive, negative, and neutral. Typically, one of these states has a probability of 0.1, while the other two states each have a probability of 0.45.

We investigate the impact of the constant parameter in three different scenarios [Fig. 3.19 3.20 3.21]:

1. **Stepwise Addition**: In this case, the constant is added to the reward function in every call. 2. **Episode-wise Addition**: Here, the constant is added once per episode. 3. **Interval Addition**: The constant is added every two seconds.

These cases are tested with three different constant values: 0.5, 0.05, and 0.005. For each case, six lines are plotted, representing different outcomes such as "unknown," "success," "collision with a wall," "collision with a dynamic obstacle," and "tumble."

Overall, the plots for the three types of cases appear quite similar. However, some subtle differences can be observed. Notably, when the constant is set to 0.005, it yields the best performance for the reward function, considering that the negative emotion has a probability of 0.1, while positive and neutral emotions have a probability of 0.45. This value seems to provide the optimal parameter for our reward function compared to the other constant values.

Interestingly, regardless of whether the constant is added every step, every episode, or every two seconds, the differences in performance are relatively minor for this particular case.

– **DRL + LiDAR**

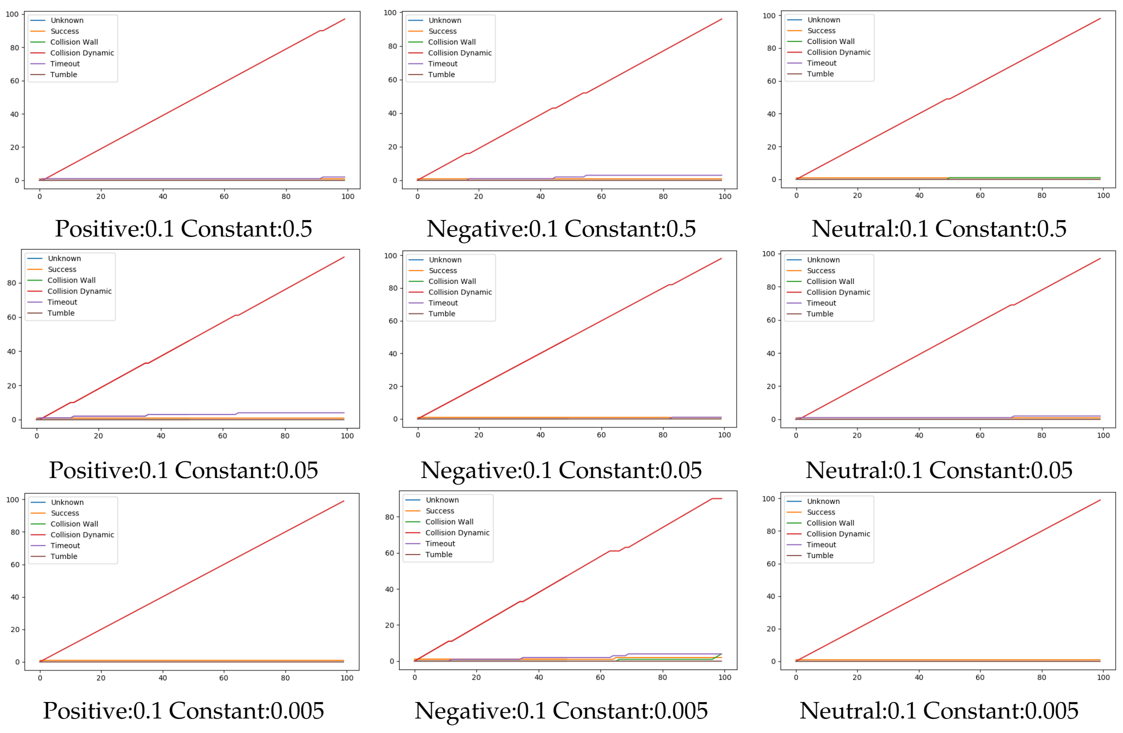


FIGURE 3.19: Case Study Adding Constant every Step

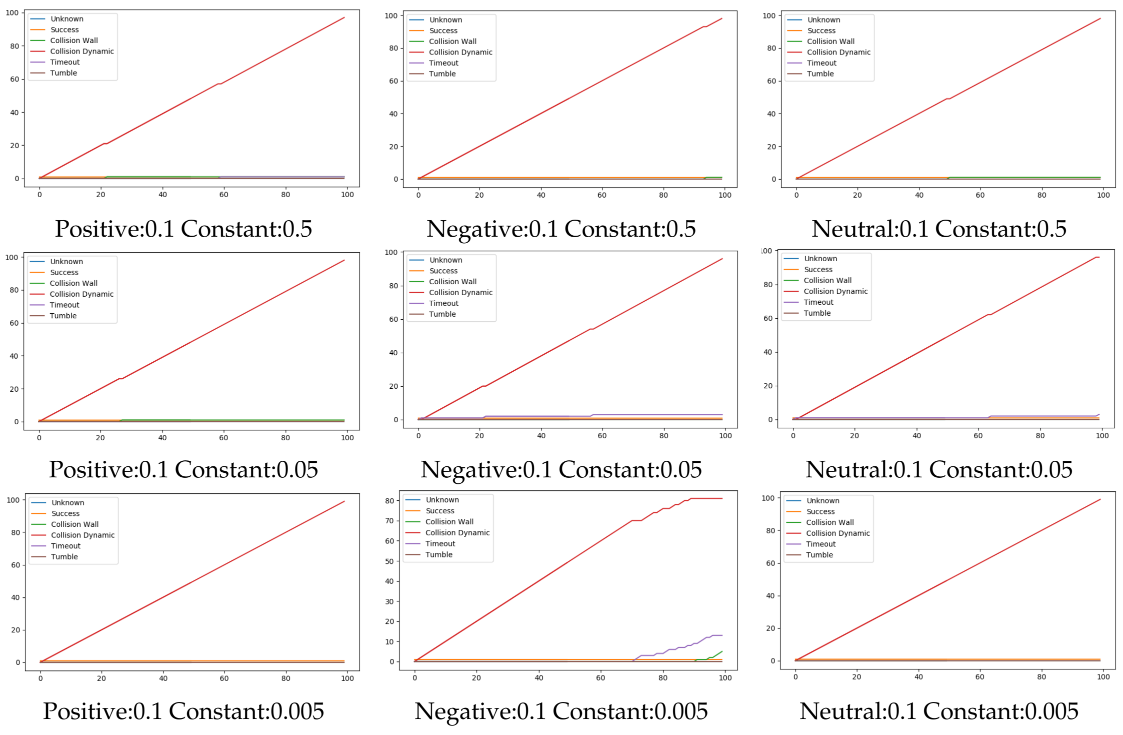
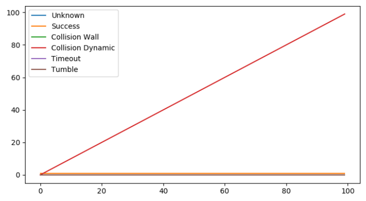
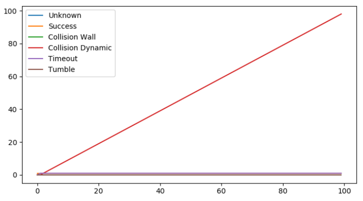
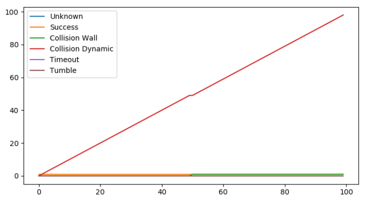


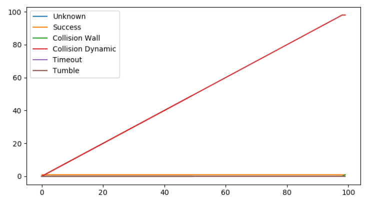
FIGURE 3.20: Case Study Adding Constant every 1 Episode

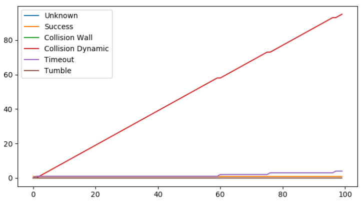


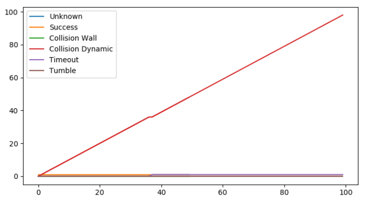




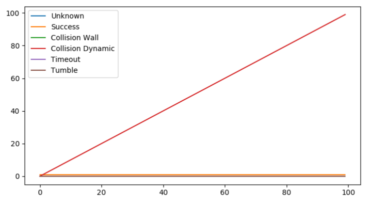
Positive:0.1 Constant:0.5 Negative:0.1 Constant:0.5 Neutral:0.1 Constant:0.5

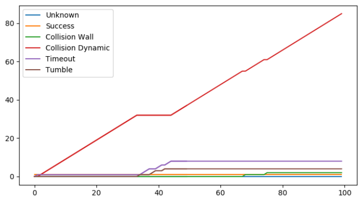


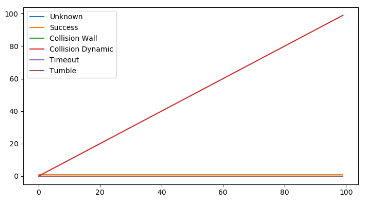




Positive:0.1 Constant:0.05 Negative:0.1 Constant:0.05 Neutral:0.1 Constant:0.05







Positive:0.1 Constant:0.005 Negative:0.1 Constant:0.005 Neutral:0.1 Constant:0.005

FIGURE 3.21: Case Study Adding Constant every 2 seconds

This is a case study for LiDAR results in three different deep reinforcement learning (DRL) algorithms: DDPG, DQN, and TD3. Each algorithm was trained for 100 episodes, and four different plots were generated to show the outcomes: average critic loss, average critic loss and average reward. Each model has different results [Fig 3.22 3.23 3.24].

Regarding the outcome plots for the three different algorithms, we can observe that DDPG is dominated by the collision dynamics outcome, while TD3 and DQN are dominated by collisions from walls (static obstacles). This means that our model learned to avoid static obstacles for DDPG, while the other two algorithms learned to avoid collisions with dynamic obstacles. Additionally, we can see that the success line for TD3 is the best among the three algorithms, achieving around 12 successes out of 100 episodes, while DQN slightly performs worse than TD3 with around 7 successes. On the other hand, DDPG achieves a very low number of successes, around 3.

In terms of average critic loss, we can see that TD3 shows a more convergent average loss, which corresponds to its better results in the outcome plots. DQN demonstrates a stable line approximated to 0 because it does not have

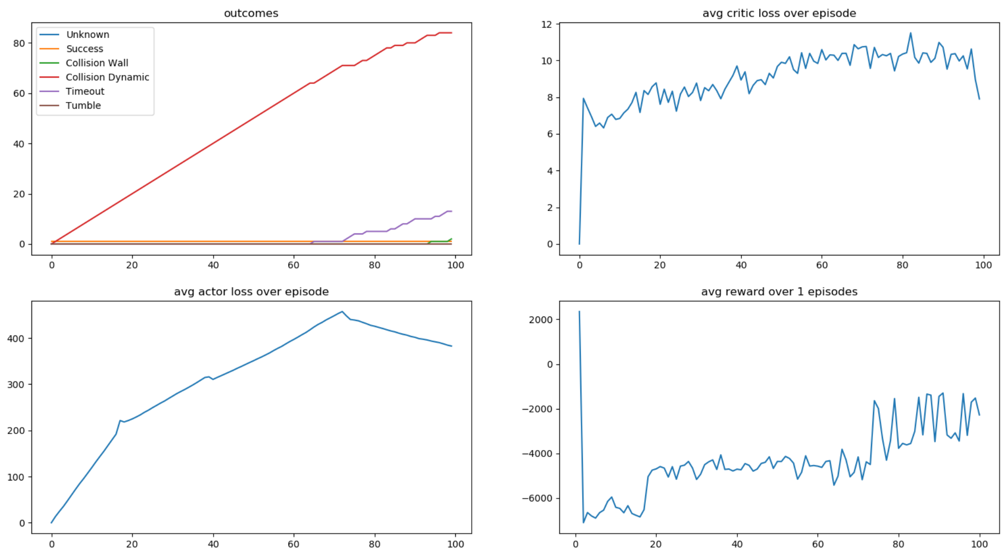


FIGURE 3.22: Results of Training DDPG Model using lidar

a critic network, indicating that no significant learning occurred and the successes achieved by the robot did not result from learning. DDPG shows an attempt to converge but did not fully achieve convergence, which explains the obtained outcome results.

For average actor loss, TD3 has the lowest value of around 120, followed by DDPG. DQN has the worst average actor loss of around 5000, primarily due to a large spike at the beginning of training the algorithm. This spike suggests that the actor network had poor initial weights.

Regarding the average reward over episodes, we can observe multiple spikes with higher reward values, approximately around 2000, for TD3. DDPG shows a slightly lower number of spikes, while DDPG exhibits very few spikes due to the low number of successes achieved.

– **DRL + Emotion**

We conducted another experiment by training the three algorithms, DDPG, DQN, and TD3, using only emotion feedback without lidar. The results are presented in Figure 3.25, 3.26, and 3.27. This approach shows promise as a novel technique. Each algorithm was trained for 100 episodes.

When analyzing the outcome line plots, we observed that all three models struggled with static obstacles. However, the success rate for all models was better in avoiding dynamic collisions. This suggests that our models were

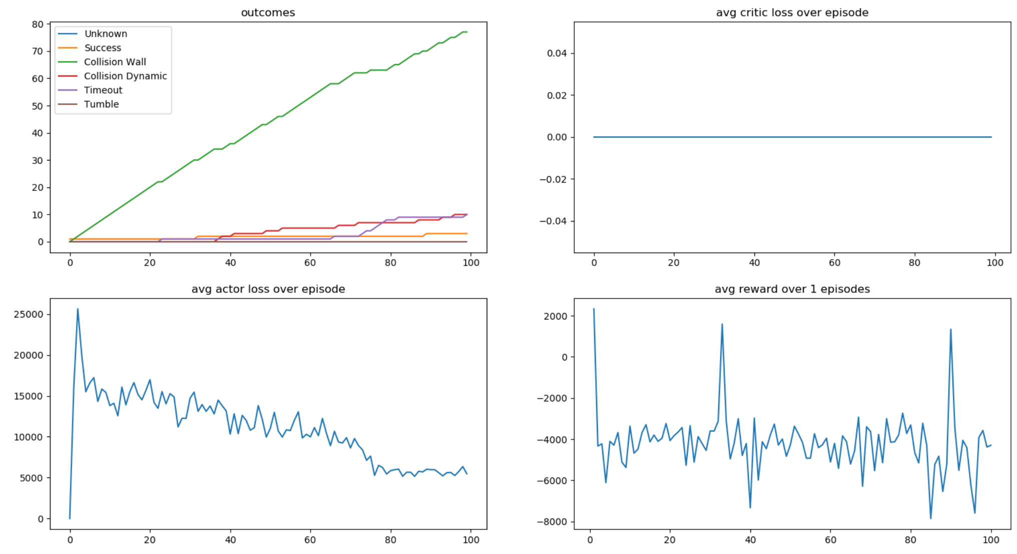


FIGURE 3.23: Results of Training DQN Model using lidar

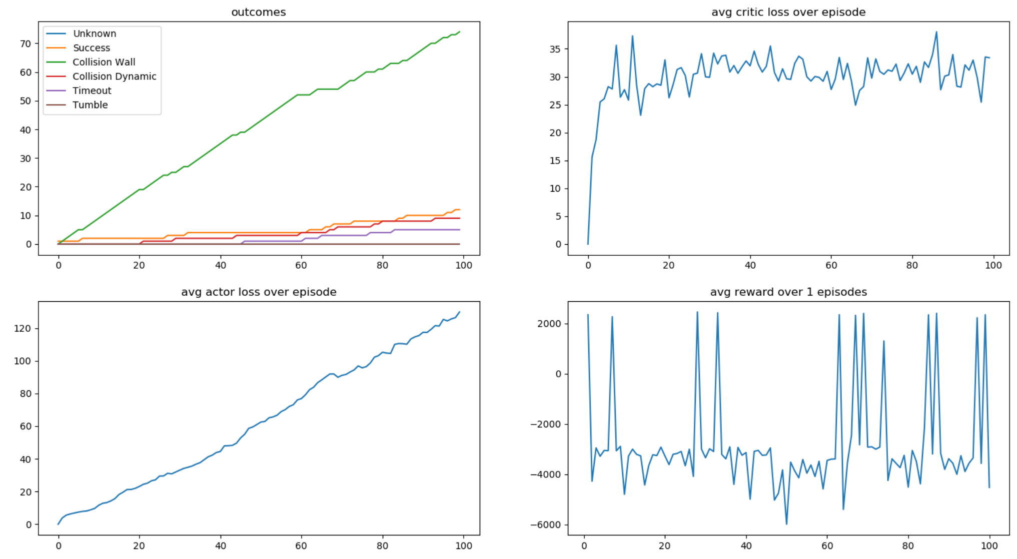


FIGURE 3.24: Results of Training TD3 Model using lidar

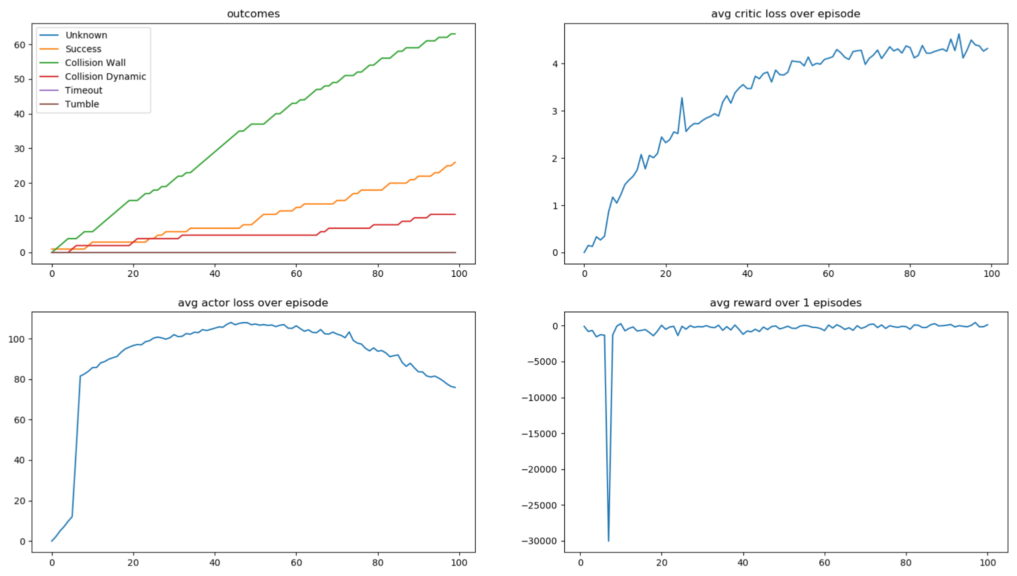


FIGURE 3.25: Results of Training DDPG Model using emotion

unable to effectively handle static walls without lidar information, but they still demonstrated a learning pattern. Specifically, DDPG achieved approximately 25 successful outcomes, TD3 achieved around 21, and DQN achieved about 10.

In terms of average critic loss, DDPG exhibited stable and converging loss with a low value of around 4. TD3, on the other hand, had unstable average critic loss that was slightly higher than DDPG. Interestingly, DQN had a critic loss of 0, indicating that it does not use a critic loss function.

When considering the average actor loss, DDPG showed the best convergence with a lower value of approximately 80. DQN attempted to converge at around 200, while TD3 performed better than DQN but worse than DDPG. These trends align with the outcome results of the models.

Examining the average reward plot, DDPG consistently achieved stable and high rewards, indicating that our model made fewer mistakes and successfully reached the goal. TD3 had more spikes in reward, reaching around 2000, but also experienced dips as a result of collisions. DQN had a worse plot compared to TD3, which can be attributed to its lower number of successful goal achievements.

– **DRL + LIDAR + Emotion**

In this case study, we utilized three models, namely DDPG, DQN, and TD3,

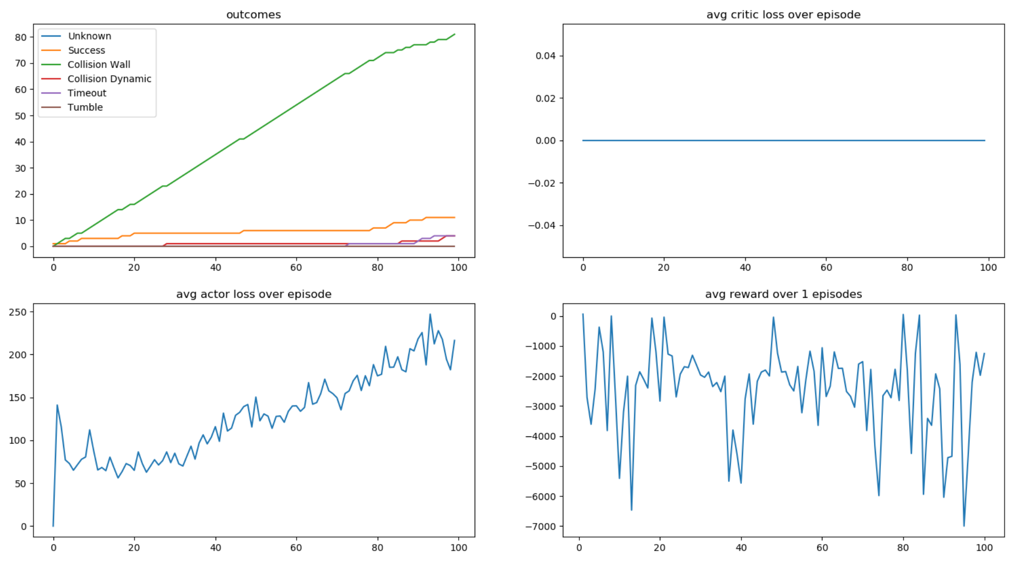


FIGURE 3.26: Results of Training DQN Model using emotion

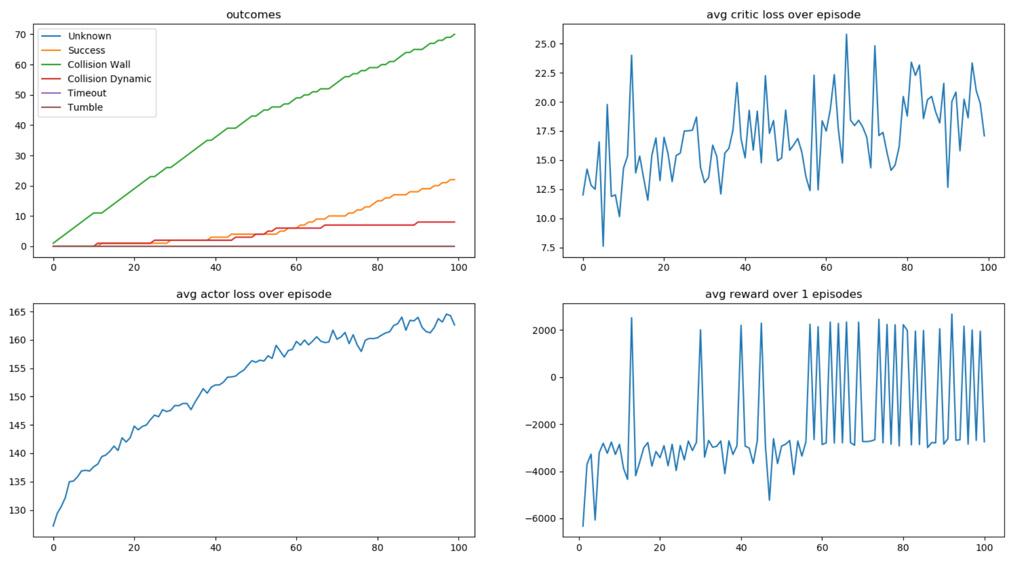


FIGURE 3.27: Results of Training TD3 Model using emotion

trained for 100 episodes using a hypered feedback system that combines lidar and emotion feedback [Figure 3.28, 3.29, 3.30].

Analyzing the outcome plots, we observe that DDPG achieved the highest success rate, with approximately 45 successful outcomes, surpassing the other two algorithms. DDPG primarily learned to avoid dynamic obstacles and reduced collisions with static obstacles. On the other hand, DQN demonstrated proficiency in reaching the goal position with around 32 successful outcomes, but struggled with avoiding static obstacles. TD3 obtained a similar number of successful outcomes as DQN and faced similar challenges with static obstacle avoidance. However, both DQN and TD3 learned to avoid dynamic obstacles earlier than DDPG.

Examining the average critic loss, we observe that DDPG mostly converged, although it exhibited significant noise. TD3 still experienced higher critic loss, while DQN had a fixed critic loss of 0.

Regarding average actor loss, DDPG demonstrated convergence around 50 episodes, followed by a steady decline, indicating effective learning from the feedback. DQN exhibited a spiky plot, suggesting significant mistakes in actor loss over time. TD3 showed a slightly higher average actor loss, which remained monotonically increasing.

Analyzing the average reward plots, we noticed that all three algorithms exhibited numerous spikes due to achieving the goal multiple times, indicating successful progress.

Finally, if we compare the three study cases of using DRL algorithms DDPG, DQN, and TD3 for lidar only, emotion only, and lidar plus emotion, some interesting observations emerge. First, when using lidar alone, the number of episodes (100) appears insufficient for the algorithms to effectively learn navigation towards the goal. Second, the novelty of incorporating emotion feedback without any sensor input yielded better results compared to lidar alone. However, it is important to note that merging both methods, lidar and emotion, together resulted in even better outcomes than either emotion or lidar alone. Specifically, DDPG achieved approximately 45 successful outcomes over the course of 100 episodes.

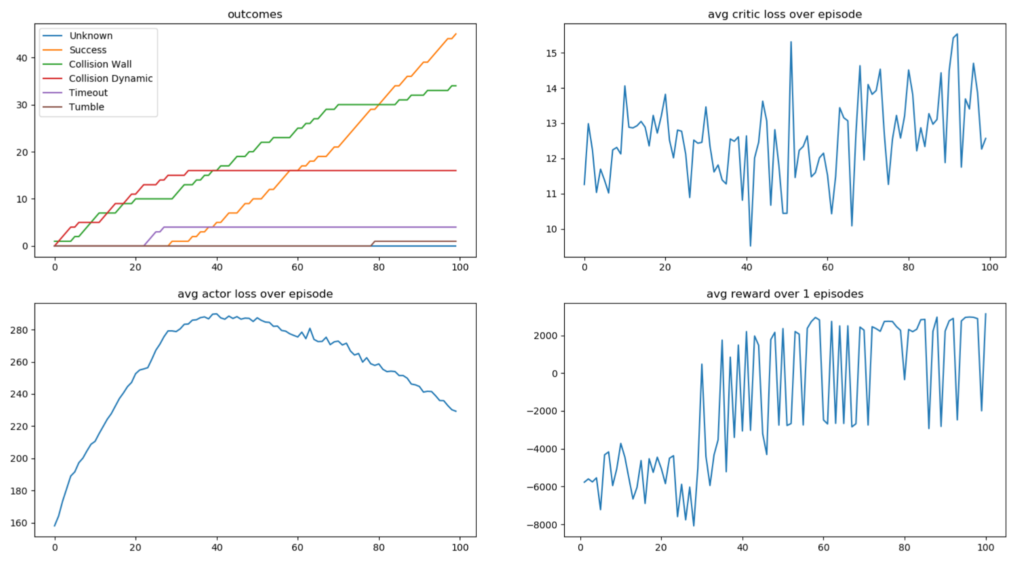


FIGURE 3.28: Results of Training DDPG Model using lidar + emotion

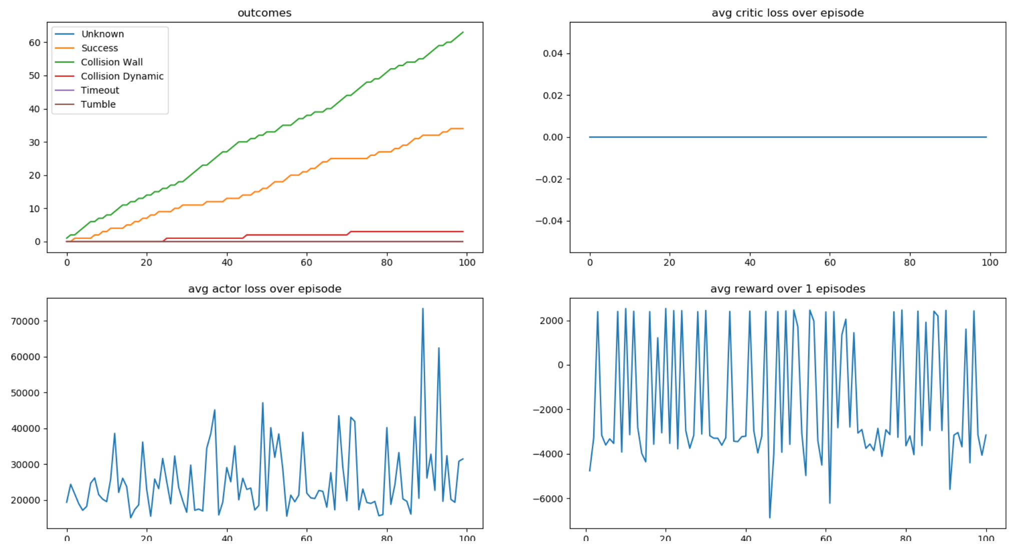


FIGURE 3.29: Results of Training DQN Model using lidar + emotion

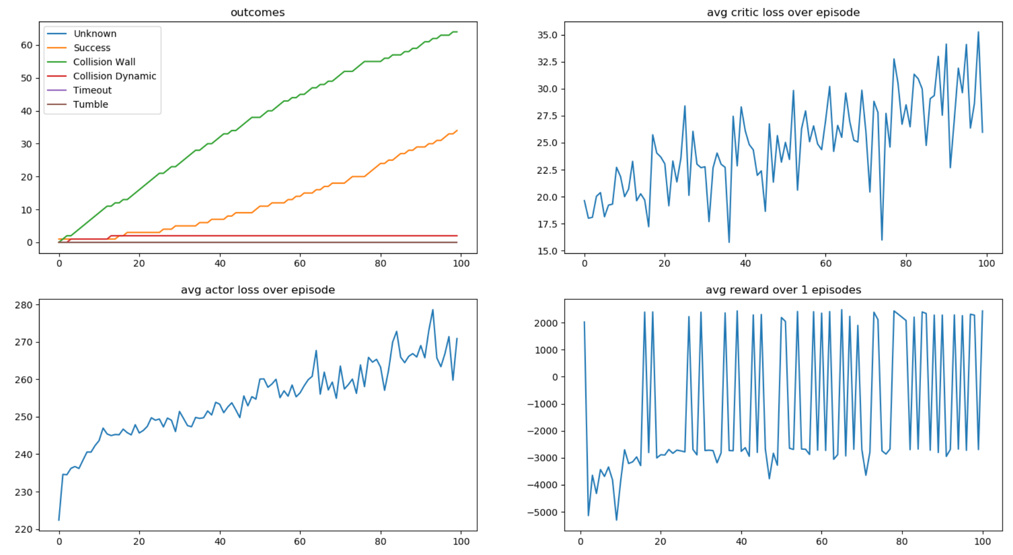


FIGURE 3.30: Results of Training TD3 Model using lidar + emotion

3.6 Conclusion

In this chapter, we conducted a comprehensive study on the utilization of deep reinforcement learning (DRL) with emotion feedback. Through various experiments, we determined optimal parameters for enhanced learning outcomes using DDPG, DQN, and TD3 algorithms with lidar and emotion feedback. Our investigations resulted in improved overall performance and a significant reduction in the number of episodes required for successful maze navigation. By incorporating natural sensor feedback, particularly human visual perception and brain signals, we gained valuable insights into the potential of utilizing such feedback in reinforcement learning tasks. Our findings demonstrate the effectiveness of combining deep reinforcement learning with emotion feedback, opening up new avenues for leveraging natural sensor feedback and presenting opportunities for future work in multi-robot scenarios and transfer learning using brain signals beyond emotions.

General Conclusion

4.1 Conclusion

In conclusion, the proposed system design, which combines brain signal analysis using EEG and deep reinforcement learning techniques, shows promising results in improving robot navigation in indoor applications. The integration of emotional feedback obtained from the EEG signal and environmental feedback from a 2D LIDAR sensor enables the robot to make informed decisions and navigate through maze-like environments effectively.

The results from the emotion classification part of the system demonstrate superior performance compared to state-of-the-art models. This indicates the potential of leveraging emotional feedback as a valuable input for the robot's decision-making process. The navigation part of the deep reinforcement learning system also exhibits significant enhancements by incorporating constant variables to prevent collisions with obstacles. Furthermore, a comparison of different feedback methods highlights the advantages of combining LIDAR and emotion feedback, surpassing the individual approaches.

The utilization of various deep reinforcement learning algorithms, including DDPG, TD3, and DQN, has further demonstrated the adaptability and effectiveness of the proposed system. Additionally, preprocessing techniques and classification models like ViT and DGCNN have proven to be successful in accurately classifying emotions.

4.2 Future work

Future research in this domain will focus on several important aspects. Firstly, exploring the impact of alternative reinforcement learning algorithms and investigating their suitability for the proposed system will provide valuable insights into system performance and optimization. Additionally, studying the effectiveness of dynamic policy networks in enabling robots to learn new tasks from human feedback holds promise for expanding the robot's capabilities.

Furthermore, the potential impact of brain signal feedback extends beyond robot navigation. Investigating the applicability of this novel method in more general tasks can open new avenues for research and development in the field of human-robot interaction.

Another area of future work involves studying the effectiveness of multi-robot swarm systems using the proposed method. The scalability and coordination capabilities of such systems can greatly benefit from the integration of neural feedback in reinforcement learning.

Moreover, the utilization of neural feedback offers the opportunity to reduce the reliance on numerous sensors, making the robot more cost-effective and efficient. Exploring the implications and benefits of this approach in terms of speed and adaptability will be valuable for further advancements.

Lastly, considering the increasing prevalence of chat GPT systems that heavily rely on reinforcement learning and human feedback, studying the intersection between these systems and the proposed method can lead to significant advancements and practical applications.

Overall, the combination of brain signal analysis, deep reinforcement learning, and human feedback in robot navigation opens up exciting possibilities for research and development in the field. The outcomes of this thesis contribute to the advancement of semi-mind controlled robots and pave the way for future innovations in human-robot interaction and intelligent systems.

Bibliography

[1] Paul F Christiano et al. "Deep reinforcement learning from human preferences". In: *Advances in neural information processing systems* 30 (2017).

[2] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[3] Ezdin Aslanci and Kutalmis Coskun. "USING HIERARCHIES IN REINFORCEMENT LEARNING FRAMEWORK WITH NON-STATIONARY ENVIRONMENTS". PhD thesis. June 2017. DOI: 10.13140/RG.2.2.18773.06888.

[4] Mohd Hafiz Mohd. "Revisiting discrepancies between stochastic agent-based and deterministic models". In: *Community Ecology* 23.3 (2022), pp. 453–468.

[5] David Silver et al. "Deterministic policy gradient algorithms". In: *International conference on machine learning*. Pmlr. 2014, pp. 387–395.

[6] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[7] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8 (1992), pp. 279–292.

[8] Volodymyr Mnih et al. "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. PMLR. 2016, pp. 1928–1937.

[9] Martin L Puterman. "Markov decision processes". In: *Handbooks in operations research and management science* 2 (1990), pp. 331–434.

[10] Elise Van Der Pol. "Deep reinforcement learning for coordination in traffic light control". In: *Master's thesis, University of Amsterdam* (2016).

[11] Marc G Bellemare, Will Dabney, and Rémi Munos. “A distributional perspective on reinforcement learning”. In: *International conference on machine learning*. PMLR. 2017, pp. 449–458.

[12] Nicholas Metropolis and Stanislaw Ulam. “The monte carlo method”. In: *Journal of the American statistical association* 44.247 (1949), pp. 335–341.

[13] Jianqing Fan et al. “A theoretical analysis of deep Q-learning”. In: *Learning for Dynamics and Control*. PMLR. 2020, pp. 486–489.

[14] W Bradley Knox and Peter Stone. “Interactively shaping agents via human reinforcement: The TAMER framework”. In: *Proceedings of the fifth international conference on Knowledge capture*. 2009, pp. 9–16.

[15] Gernot R Müller-Putz et al. “Brain-computer interfaces for control of neuroprostheses: from synchronous to asynchronous mode of operation/Brain-Computer Interfaces zur Steuerung von Neuroprothesen: von der synchronen zur asynchronen Funktionsweise”. In: (2006).

[16] David Steyrl, Reinmar J Kobler, Gernot R Müller-Putz, et al. “On similarities and differences of invasive and non-invasive electrical brain signals in brain-computer interfacing”. In: *Journal of biomedical science and engineering* 9.08 (2016), p. 393.

[17] Claudio Babiloni et al. “Human cortical electroencephalography (EEG) rhythms during the observation of simple aimless movements: a high-resolution EEG study”. In: *Neuroimage* 17.2 (2002), pp. 559–572.

[18] Gregory A Light et al. “Electroencephalography (EEG) and event-related potentials (ERPs) with human participants”. In: *Current protocols in neuroscience* 52.1 (2010), pp. 6–25.

[19] Alexander E Hramov, Vladimir A Maksimenko, and Alexander N Pisarchik. “Physical principles of brain–computer interfaces and their applications for rehabilitation, robotics and control of human brain states”. In: *Physics Reports* 918 (2021), pp. 1–133.

[20] Sarah N Abdulkader, Ayman Atia, and Mostafa-Sami M Mostafa. "Brain computer interfacing: Applications and challenges". In: *Egyptian Informatics Journal* 16.2 (2015), pp. 213–230.

[21] Biao Zhang, Jianjun Wang, and Thomas Fuhlbrigge. "A review of the commercial brain-computer interface technology from perspective of industrial robotics". In: *2010 IEEE international conference on automation and logistics*. IEEE. 2010, pp. 379–384.

[22] Stephanie M Casillo, Diego D Luy, and Ezequiel Goldschmidt. "A history of the lobes of the brain". In: *World Neurosurgery* 134 (2020), pp. 353–360.

[23] Harvey Levin and Marilyn F Kraus. "The frontal lobes and traumatic brain injury." In: *The Journal of neuropsychiatry and clinical neurosciences* (1994).

[24] Selma Buyukgoze. "NON-INVASIVE BCI METHOD: EEG - ELECTROENCEPHALOGRAPHY". In: *International Conference on Technics, Technologies and Education* (Jan. 2019), pp. 139–145. DOI: 10.15547/ictte.2019.02.095.

[25] Daniel Brandeis and D Lehmann. "Event-related potentials of the brain and cognitive processes: approaches and applications". In: *Neuropsychologia* 24.1 (1986), pp. 151–168.

[26] Yaki Stern, Amit Reches, and Amir B Geva. "Brain network activation analysis utilizing spatiotemporal features for event related potentials classification". In: *Frontiers in computational neuroscience* 10 (2016), p. 137.

[27] Yao-qin Qiu et al. "P300 aberration in first-episode schizophrenia patients: a meta-analysis". In: *PLoS One* 9.6 (2014), e97794.

[28] Kimitaka Kaga et al. "Effects of Button Pressing and Mental Counting on N100, N200, and P300 of Auditory-Event-Related Potential Recording." In: *Journal of International Advanced Otology* 10.1 (2014).

[29] Kimitaka Kaga et al. "Effects of Button Pressing and Mental Counting on N100, N200, and P300 of Auditory-Event-Related Potential Recording." In: *Journal of International Advanced Otology* 10.1 (2014).

[30] Setare Amiri, Reza Fazel-Rezai, and Vahid Asadpour. "A review of hybrid brain-computer interface systems". In: *Advances in Human-Computer Interaction* 2013 (2013), pp. 1–1.

[31] Davide Valeriani, Caterina Cinel, and Riccardo Poli. *Brain–computer interfaces for human augmentation*. 2019.

[32] Anders M Dale and Eric Halgren. "Spatiotemporal mapping of brain activity by integration of multiple imaging modalities". In: *Current opinion in neurobiology* 11.2 (2001), pp. 202–208.

[33] CJ Stam. "Use of magnetoencephalography (MEG) to study functional brain networks in neurodegenerative disorders". In: *Journal of the neurological sciences* 289.1-2 (2010), pp. 128–134.

[34] Hellmuth Obrig. "NIRS in clinical neurology—a 'promising'tool?" In: *Neuroimage* 85 (2014), pp. 535–546.

[35] Gerwin Schalk. "Can electrocorticography (ECoG) support robust and powerful brain-computer interfaces?" In: *Frontiers in neuroengineering* (2010), p. 9.

[36] Andreas Widmann, Erich Schröger, and Burkhard Maess. "Digital filter design for electrophysiological data–a practical approach". In: *Journal of neuroscience methods* 250 (2015), pp. 34–46.

[37] James V Stone. "Independent component analysis: an introduction". In: *Trends in cognitive sciences* 6.2 (2002), pp. 59–64.

[38] Li Hu et al. "Single-trial time–frequency analysis of electrocortical signals: Baseline correction and beyond". In: *Neuroimage* 84 (2014), pp. 876–887.

[39] Lennart Eriksson et al. "Methods for reliability and uncertainty assessment and for applicability evaluations of classification-and regression-based QSARs." In: *Environmental health perspectives* 111.10 (2003), pp. 1361–1375.

[40] Graham C Goodwin and Kwai Sang Sin. *Adaptive filtering prediction and control*. Courier Corporation, 2014.

[41] B Samanta and KR Al-Balushi. "Artificial neural network based fault diagnostics of rolling element bearings using time-domain features". In: *Mechanical systems and signal processing* 17.2 (2003), pp. 317–328.

[42] Md Sayed Tanveer and Md Kamrul Hasan. "Cuffless blood pressure estimation from electrocardiogram and photoplethysmogram using waveform based ANN-LSTM network". In: *Biomedical Signal Processing and Control* 51 (2019), pp. 382–392.

[43] Robert Wall Emerson. "ANOVA and t-tests". In: *Journal of Visual Impairment & Blindness* 111.2 (2017), pp. 193–196.

[44] Christian Schuldt, Ivan Laptev, and Barbara Caputo. "Recognizing human actions: a local SVM approach". In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* Vol. 3. IEEE. 2004, pp. 32–36.

[45] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.

[46] Gérard Biau and Erwan Scornet. "A random forest guided tour". In: *Test* 25 (2016), pp. 197–227.

[47] Jiuxiang Gu et al. "Recent advances in convolutional neural networks". In: *Pattern recognition* 77 (2018), pp. 354–377.

[48] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[49] Dr KM Kumar, Jhenkar L Nataraj, et al. "Analysis of EEG based emotion detection of DEAP and SEED-IV databases using SVM". In: (2019).

[50] Sathujoda Prabhakar, Amiya Ranjan Mohanty, and AS Sekhar. "Application of discrete wavelet transform for detection of ball bearing race faults". In: *Tribology International* 35.12 (2002), pp. 793–800.

[51] Muhammad Khateeb, Syed Muhammad Anwar, and Majdi Alnowami. "Multidomain feature fusion for emotion classification using DEAP dataset". In: *Ieee Access* 9 (2021), pp. 12134–12142.

[52] Wei Liu et al. "Multimodal emotion recognition using deep canonical correlation analysis". In: *arXiv preprint arXiv:1908.05349* (2019).

[53] Marti A. Hearst et al. "Support vector machines". In: *IEEE Intelligent Systems and their applications* 13.4 (1998), pp. 18–28.

[54] Jie-Lin Qiu, Xiao-Yu Li, and Kai Hu. "Correlated attention networks for multimodal emotion recognition". In: *2018 IEEE international conference on bioinformatics and biomedicine (BIBM)*. IEEE. 2018, pp. 2656–2660.

[55] Kaisheng Yao et al. "Depth-gated recurrent neural networks". In: *arXiv preprint arXiv:1508.03790* 9 (2015), p. 98.

[56] Yu-Ting Lan, Wei Liu, and Bao-Liang Lu. "Multimodal emotion recognition using deep generalized canonical correlation analysis with an attention mechanism". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–6.

[57] Andrea Garulli et al. "Mobile robot SLAM for line-based environment representation". In: *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE. 2005, pp. 2041–2046.

[58] Viet Nguyen et al. "Orthogonal SLAM: a step toward lightweight indoor autonomous navigation". In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2006, pp. 5007–5012.

[59] El Houssein Chouaib Harik and Audun Korsaeth. "Combining hector slam and artificial potential field for autonomous navigation inside a greenhouse". In: *Robotics* 7.2 (2018), p. 22.

[60] Gabriel Sepulveda, Juan Carlos Niebles, and Alvaro Soto. "A deep learning based behavioral approach to indoor autonomous navigation". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 4646–4653.

[61] Ye-Hoon Kim, Jun-Ik Jang, and Sojung Yun. "End-to-end deep learning for autonomous navigation of mobile robot". In: *2018 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE. 2018, pp. 1–6.

[62] JK Wang et al. "A LiDAR based end to end controller for robot navigation using deep neural network". In: *2017 IEEE International Conference on Unmanned Systems (ICUS)*. IEEE. 2017, pp. 614–619.

[63] Hao Quan, Yansheng Li, and Yi Zhang. "A novel mobile robot navigation method based on deep reinforcement learning". In: *International Journal of Advanced Robotic Systems* 17.3 (2020), p. 1729881420921672.

[64] Pengyu Yue et al. "Experimental research on deep reinforcement learning in autonomous navigation of mobile robot". In: *2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA)*. IEEE. 2019, pp. 1612–1616.

[65] Xidi Xue et al. "A deep reinforcement learning method for mobile robot collision avoidance based on double dqn". In: *2019 IEEE 28th International Symposium on Industrial Electronics (ISIE)*. IEEE. 2019, pp. 2131–2136.

[66] Xiaogang Ruan et al. "Mobile robot navigation based on deep reinforcement learning". In: *2019 Chinese control and decision conference (CCDC)*. IEEE. 2019, pp. 6174–6178.

[67] Hartmut Surmann et al. "Deep reinforcement learning for real autonomous mobile robot navigation in indoor environments". In: *arXiv preprint arXiv:2005.13857* (2020).

[68] Khaled Alaaeldin Abdelfattah Mustafa. "Towards continuous control for mobile robot navigation: A reinforcement learning and slam based approach". MA thesis. University of Twente, 2019.

[69] Hugh Durrant-Whyte and Tim Bailey. "Simultaneous localization and mapping: part I". In: *IEEE robotics & automation magazine* 13.2 (2006), pp. 99–110.

[70] Lei Tai, Giuseppe Paolo, and Ming Liu. "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 31–36.

[71] Yuke Zhu et al. "Target-driven visual navigation in indoor scenes using deep reinforcement learning". In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 3357–3364.

[72] Alexander Scott, Lynne E Parker, and Claude Touzet. "Quantitative and qualitative comparison of three laser-range mapping algorithms using two types of laser scanner data". In: *Smc 2000 conference proceedings. 2000 ieee*

international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0. Vol. 2. IEEE. 2000, pp. 1422–1427.

[73] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[74] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[75] Ravikiran Mane et al. "FBCNet: A multi-view convolutional neural network for brain-computer interface". In: *arXiv preprint arXiv:2104.01233* (2021).

[76] Bo Pan, Wei Zheng, et al. "Emotion recognition based on EEG using generative adversarial nets and convolutional neural network". In: *computational and Mathematical Methods in Medicine* 2021 (2021).

[77] Vernon J Lawhern et al. "EEGNet: a compact convolutional neural network for EEG-based brain–computer interfaces". In: *Journal of neural engineering* 15.5 (2018), p. 056013.

[78] Alexey Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[79] Tengfei Song et al. "EEG emotion recognition using dynamical graph convolutional neural networks". In: *IEEE Transactions on Affective Computing* 11.3 (2018), pp. 532–541.

[80] Peixiang Zhong, Di Wang, and Chunyan Miao. "EEG-based emotion recognition using regularized graph neural networks". In: *IEEE Transactions on Affective Computing* 13.3 (2020), pp. 1290–1301.

[81] Jie-Lin Qiu, Xiao-Yu Li, and Kai Hu. "Correlated attention networks for multimodal emotion recognition". In: *2018 IEEE international conference on bioinformatics and biomedicine (BIBM)*. IEEE. 2018, pp. 2656–2660.

[82] Rushuang Zhou et al. "A novel transfer learning framework with prototypical representation based pairwise learning for cross-subject cross-session EEG-based emotion recognition". In: *arXiv preprint arXiv:2202.06509* (2022).

[83] Wei Liu et al. "Multimodal emotion recognition using deep canonical correlation analysis". In: *arXiv preprint arXiv:1908.05349* (2019).