UNIVERSIDADE D
COIMBRA

Daniel da Silva Palaio

DeepRL-based Motion Planning For Indoor Robot Navigation

Dissertation supervised by Professor Doctor Urbano José Carreira Nunes and submitted to the Electrical and Computer Engineering Department of the Faculty of Science and Technology of the University of Coimbra, in partial fulfillment of the requirements for the Master Degree in Electrical and Computer Engineering, specialization in Automation.

October of 2021

UNIVERSIDADE D
COIMBRA

DeepRL-Based Motion Planning for Indoor Robot Navigation

Daniel da Silva Palaio

Coimbra, October of 2021

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

DeepRL-Based Motion Planning for Indoor Robot Navigation

Dissertation supervised by Professor Doctor Urbano José Carreira Nunes and submitted to the Electrical and Computer Engineering Department of the Faculty of Science and Technology of the University of Coimbra, in partial fulfillment of the requirements for the Master Degree in Electrical and Computer Engineering, specialization in Automation.

Supervisor:
Prof. Dr. Urbano José Carreira Nunes

Co-Supervisor:
Master Luís Carlos Artur da Silva Garrote

Jury:
Prof. Dr. Hélder de Jesus Araújo
Prof. Dr. Rui Paulo Pinto da Rocha
Prof. Dr. Urbano José Carreira Nunes

Coimbra, October of 2021

Acknowledgments

This dissertation marks the end of a six-year academic journey, only possible to conclude with some people's fundamental support, to whom I have to thank.

First and foremost, I am genuinely grateful to Prof. Dr. Urbano Nunes for entrusting me with this challenging dissertation subject and providing all the necessary resources to achieve the established objectives. A word to Master Luis Garrote for the shared knowledge and constant assistance during this work. For their availability, guidance, and helpful advice, the utmost appreciation.

A very heartfelt thanks to my family, especially to my mother, father, brother, and grandparents, for the unconditional support, motivation, and faith in every decision that ultimately led me to where and what I am today.

To the colleagues that I was fortunate to meet and naturally became my closest friends, Daniel Craveiro, Francisco Alves, Gonçalo Lopes, Guilherme Carvalho, Hugo Figueiras, and João Duarte, a big thank you. Together we made an incredible team, shared great experiences, and created unforgettable memories. To the years to come!

A particular acknowledgment to Bosch Thermotechnology, Engineering Controls team, for in times of uncertainty have believed in my ability to reconcile the work I set myself and the completion of this dissertation. To the people of this great institution who have always expressed their support, concern, and curiosity for this project, among whom I have to single out my workmates Luís Simões, Maria João Loureiro, and Rafael Gaspar, my sincere gratitude.

Lastly, a special thanks to Milene Lopes, my main source of inspiration, for the unparalleled and endless support, standing by my side through the ups and downs along the way.

This work has been supported by MATIS-CENTRO-01-0145- FEDER-000014, Portugal, and by ISR-UC FCT through grant UIDB/00048/2020.

Abstract

Robots, driven by substantial technological advances, are no longer confined to executing industrial-related duties. Among the several subclasses of the widespread robotics field, mobile robotics - answerable for developing non-stationary platforms capable of navigating indoor and outdoor environments - has been one of the extents responsible for the diffusion of robot applications across various domains.

Within robot navigation, motion planning is the primitive that establishes a route from an initial to a target point. In uncharted environments, however, the path definition can be a significantly more challenging task. Due to the inability to delineate a course based on map knowledge, a local navigation resolution is settled instead, with short-term paths being outlined according to the observed surrounding environment.

This dissertation presents an original local motion planning strategy for unexplored indoor environments based on Deep Reinforcement Learning (DeepRL), a contemporary Machine Learning field. Typically, DeepRL navigation applications use raw data directly as input to their framework's Artificial Neural Networks (ANNs), which may format them towards the intrinsic properties of the robot's onboard cameras/lasers. Contrarily, in an effort to create a sensor-agnostic navigation approach, the proposed method pre-processes the collected sensory data into normalized environment representations named costmaps. To comply with the implemented variations and system requirements, the employed ANNs and complementary models were designed from scratch.

The introduced path planning algorithm is partitioned into two distinct stages: training and testing. In the training phase, an intelligent mobile platform learns, via trial-and-error, which actions must be adopted to attain its target without colliding with obstacles. Optimally, training generates at least one fine-tuned model, further tested in an online stage, that empowers the robot to effectively perform a collision-free motion.

To validate the presented local motion planning approach, a virtual robot - the Turtlebot - was applied in multiple simulation environments, with and without obstacles. Using the developed framework to sustain the Turtlebot's decision-making, promising results were yielded over several trials in both types of domains.

Keywords: Motion Planning, Autonomous Indoor Robot Navigation, Deep Reinforcement Learning, Artificial Neural Networks, Costmaps

Resumo

Robôs, impulsionados por consideráveis avanços tecnológicos, têm vindo a ser progressivamente integrados nas mais diversas áreas, contrariando a conceção de serem apenas ferramentas de suporte industrial. De entre as várias subdivisões da robótica, a robótica móvel - encarregue da criação de plataformas com a habilidade de navegar em ambientes interiores e exteriores - é uma das que mais tem contribuído para a expansão e disseminação de aplicações robotizadas.

No que se refere a navegação robótica autónoma, o planeamento de caminho é a premissa responsável pela definição de um trajeto entre um ponto inicial/atual até um ponto final. No entanto, em ambientes desconhecidos ou não mapeados, a complexidade de definir um caminho aumenta significativamente, sendo necessário recorrer a informação sensorial para estabelecer um princípio de navegação local.

Nesta dissertação é apresentada uma estratégia de planeamento de caminhos, projetada para ambientes interiores inexplorados, baseada em *Deep Reinforcement Learning* (DeepRL). Contrariando as aplicações de navegação DeepRL que utilizam imagens do ambiente circundante diretamente como entrada das suas Redes Neuronais Artificiais, o método sugerido pré-processa os dados recolhidos em representações normalizadas do ambiente, denominadas *costmaps*. Numa tentativa de criar uma abordagem independente do tipo de sensor usado, tanto a rede neuronal como os restantes modelos computacionais empregados foram concebidos de raiz para atender às variações implementadas e aos requisitos do sistema.

O algoritmo de planeamento de caminho apresentado pode ser decomposto em dois estágios distintos: treino e teste. Na fase de treino, um robô sob a influência do método de navegação local desenvolvido aprende, através de tentativa e erro, que ações deve eleger para atingir o seu ponto objetivo sem colidir com obstáculos. Durante o processo de treino é gerado, otimamente, um modelo regulado da rede neuronal responsável por suster, numa fase de testes posterior, a tomada de decisões de vertente navegacional do robô.

Com o intuito de validar o planeador de caminho proposto, uma plataforma virtual apelidada de *Turtlebot* foi aplicada em vários ambientes de simulação, com e sem obstáculos. Adotando o algoritmo de planeamento de caminho concebido, o *Turtlebot* evidenciou comportamentos bastante promissores nos mais diversos meios virtuais, legitimando deste modo o sistema de navegação DeepRL desenvolvido.

Palavras-chave: Planeamento de Caminho, Navegação Robótica Autónoma Interior, Deep Reinforcement Learning, Redes Neuronais Artificiais, Costmaps

"Nothing will work unless you do."
Maya Angelou

Contents

List of Acronyms

ANN Artificial Neural Network

APF Artificial Potential Field

CNN Convolutional Neural Network

D3QN Dueling Double Deep Q-Network

DDQN Double Deep Q-Network

DL Deep Learning

DP Dynamic Programming

DQN Deep Q-Network

DeepRL Deep Reinforcement Learning

DWA Dynamic Window Approach

MC Monte Carlo

ML Machine Learning

RL Reinforcement Learning

RNN Recurrent Neural Network

ROS Robot Operating System

RRT Rapidly-Exploring Random Trees

TD Temporal Difference

TEB Time Elastic Bands

List of Figures

List of Tables

1

Introduction

This chapter introduces the concepts explored to implement the proposed work. The motivation for the project development is also addressed, along with the *a priori* outlined objectives and key contributions.

1.1 Context and Motivation

Robotics is a wide-ranging scientific field accountable for studying, designing, producing, and applying autonomous/human-assisted programmable machines named robots. Due to significant software advances and gradual cost reduction of hardware, robotic systems are gradually expanding their scope, enhancing numerous sectors' productivity, efficiency, and work environment safety [5]. Despite the progress made, indoor mobile robots, a subset of robots able to navigate structured environments, remain absent from daily human routines. For robots to be introduced in human-populated domains, they must showcase the capability to model the different and unexpected human conducts and operate accordingly [6], a demanding feature progressively matured by the robotics community [5].

One of the main challenges in developing indoor mobile robots is designing a flawless and robot-employment-appropriate navigation method. Commonly, navigation techniques are partitioned into three stages [7]: localization, map building, and motion planning. The latter, also known as path planning, establishes an optimal/near-optimal collision-free path between two locations. The optimal path definition varies according to the requirements of each application: it can be the path that minimizes turning, braking, or the distance between the source and goal destination. Several strategies have been formulated over the past years to solve the motion planning computational problem, with Machine Learning-based approaches being some of the methodologies that have exhibited the most promising results [8].

Machine Learning (ML) [9] is a subset of Artificial Intelligence that produces self-improving algorithms through experience and is organized into Supervised Learning [10], Unsupervised Learning [11], and Reinforcement Learning (RL) [12]. In supervised algorithms, learning is generating pattern-finding models through human prearranged clusters of data named datasets. On the other hand, unsupervised learning methods create predictive models after hidden features of unlabelled input data [11]. Reinforcement Learning, in turn, is a general framework for studying sequential decision-making. In RL, an intelligent agent, its primary intervenient, learns via trial-and-error which actions must adopt to attain a predetermined objective. The goal

of Reinforcement Learning is that the agent, over time and guided by a reward system that evaluates the quality of its actions, gathers sufficient experience to select the commands that maximize the expected sum of rewards.

Reinforcement Learning algorithms, nonetheless, face some limitations regarding sample, memory, and computational complexity [13]. These problems, however, can be surpassed using Deep Learning (DL) [14]. Deep Learning aims to construct computational systems capable of identifying compact features in high-level abstractions. Their most common architectures are Artificial Neural Networks (ANNs), multi-layered configurations inspired in biological neural networks. Applications resultant from the merge of RL and DL are under an innovative and contemporary Machine Learning division defined as Deep Reinforcement Learning (DeepRL) [13], a field that has been enabling the institution of methods capable of facing previously intractable problems [13].

Concerning indoor navigation, traditional approaches rely on an obstacle map to establish a plan of action [7]. Notwithstanding, in unfamiliar or dynamic domains, motion planning strategies must be flexible so that a mobile platform adapts its behavior to unforeseen conditions. With Deep Reinforcement Learning sustaining the navigation's motion planning mission, the system's agent in the form of a robot assumes an improved ability to assess environmental stimuli and consequently appoint ideal actions, invaluable qualities in unknown environment navigation [15, 16].

1.2 Proposed Framework

The main purpose of this dissertation was to formulate and develop a motion planning algorithm for indoor robot navigation. Based on Deep Q-Network (DQN) frameworks [17], the projected methodology is structured to use only retrieved sensory data to control the mobile platform, not requiring upfront information of the environment (planning decoupled from mapping).

The presented DeepRL-based local navigation pipeline, shown in Fig. 1.1, is divided into two stages: training and testing. In training, the agent is given a certain amount of episodes e, sets of discrete time steps $t \epsilon T$, to learn which actions contribute to enclosing the distance to a target point. In each iteration t, the agent observes its surroundings and converts the gathered data into an environment state s_t. The employed Artificial Neural Network then maps s_t into a speed command a_t, triggering an environment variation. A new sensory observation is performed, and a state s_{t+1} subsequently computed. In this stage, after each action and resultant state transition $s_t \rightarrow s_{t+1}$, a reward r_t - a scalar value indicative of the quality of the action based on the originated transition - is generated. The framework's ANN is then fed with transition tuples (s_t, a_t, s_{t+1}, r_t) of past experiences to tune its parameters, an operation that determines, over time, if the action a_t is viable to be executed on similar s_t states. This process continues until a terminal condition is attained.

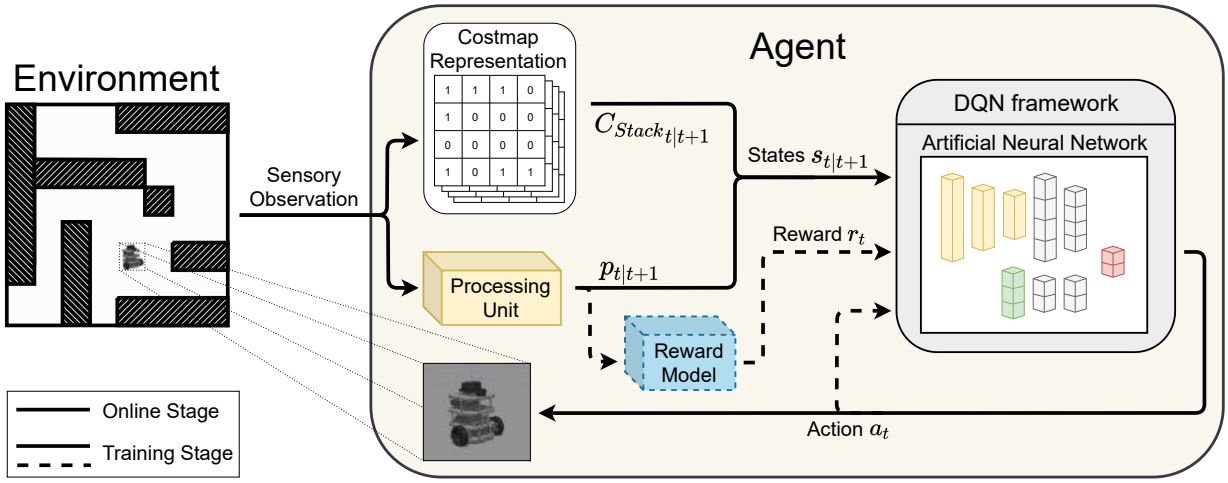


Figure 1.1: Proposed DeepRL-based local navigation framework.

Each defined state s comprises a stack of 4 sequential costmaps C_{Stack} and additional robot-pose-relative data p: robot-target Euclidean distance, robot-nearest obstacle Euclidean distance, and robot-target orientation disparity. The sensor findings are converted into costmaps - grid-like representations of the robot's surroundings - in order to reduce the impact of the sensor's intrinsic properties on the framework's network inputs.

Training aims to generate at least one ANN model that maximizes the total episode rewards. For an accurately designed DeepRL system, such models also promote the best possible agent's behavior. In the present conjuncture, the optimal behavior corresponds to a compelling collision-free motion towards the target.

In the online phase, the fine-tuned models obtained from training are utilized to conduct the agent's decision-making. Ideally, in this stage, the mobile platform successfully navigates throughout its environment until the goal location arrives.

1.3 Objectives and Key Contributions

To develop the proposed DeepRL-based motion planning for indoor robot navigation, several objectives, listed in chronological order of execution, were established and subsequently fulfilled:

1. Implementation of the original DQN approach and dueling variations [18];
2. Validation of the implemented DQN methods in benchmark environments;
3. Design of the RL inherited state, action, and reward models;
4. Design of suitable Artificial Neural Network architectures;
5. Validation of the DeepRL-based local navigation method in obstacle-free environments;
6. Validation of the DeepRL-based local navigation method in environments with obstacles.

The main implementations and contributions of the project are addressed in the following chapters:

Developed Work (Chapter 4)

Presents the strategy behind the development of each and every component of the proposed DeepRL local navigation system.

Software and Hardware Materials (Chapter 5)

Describes the software and hardware materials exploited to fulfill the defined objectives.

Results and Discussion (Chapter 6)

Documents the validation of the developed DeepRL-based local motion planning approach for indoor robot navigation in virtual environments, with and without obstacles.

DQN Implementation (Appendix A)

Presents the pseudo-codes of the original Deep Q-Learning and dueling variations implemented, validated, and employed in the suggested DeepRL navigation system.

Artificial Neural Network Architecture Deduction (Appendix B)

Outlines the structure building process of the primary Artificial Neural Network utilized in the DeepRL motion planning pipeline.

2

Background Material

This chapter covers the fundamentals of robot motion planning, Reinforcement Learning, and Deep Learning. Deep Reinforcement Learning is likewise addressed, with a particular emphasis on Deep Q-Network-based frameworks.

2.1 Motion Planning

Motion planning, in robotics, is the process of breaking down a movement task, such as following a path or moving from an initial to a goal point, into discrete actions [5, 19]. Since an effective methodology can save a considerable amount of time and reduce a mobile robot's wear and capital investment [20], motion planning presents itself as a significant navigation primitive.

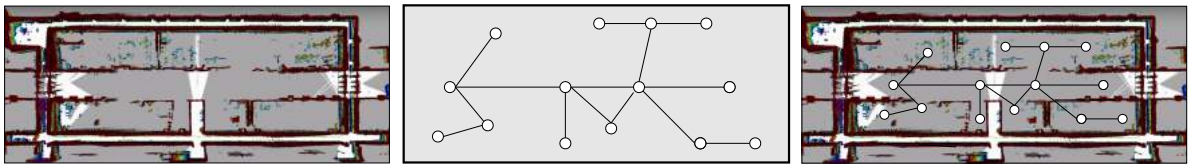
2.1.1 Motion Planners

Motion planning can be divided into global and local planning, according to whether all environment information is accessible or not [20]. Global planners generate optimal/near-optimal paths under the completely known environment. Further updates are performed on the established global map model. Local planners, on the other hand, do not operate with the entire environment knowledge. Alternatively, short-term paths are designed based on local representations built from sensory retrieved data.

2.1.2 Environment Representations

Each class of planners needs either the *a priori* knowledge of the environment or to gather data throughout the robot's motion. In both instances, the known data is converted into a feature map [20]. This resultant representation can be categorized under two main paradigms [21, 22]:

- Metric (Fig. 2.1a) - decomposition of the environment into a grid-based arrangement;
- Topological (Fig. 2.1b) - decomposition of the environment into a simplified graphical model of nodes and arcs. Nodes define different obstacle-free places and landmarks. Arcs, in turn, are the connections between nodes, portraying a path between locations.



(a) Metric [1] (b) Topological (c) Hybrid

Figure 2.1: Metric, topological, and hybrid environment representations.

While grid-based methods produce accurate metric renderings, their complexity often inhibits efficient planning in large-scale indoor environments. Topological maps, on the other hand, are precise representations often challenging to learn and maintain [21, 22]. Nevertheless, it is possible to combine both decompositions at an increased complexity cost, surpassing the downsides of each and originating a more robust environment model. An hybrid exemplar is presented in [22], where local grid-based representations are used on top of a global topological map.

2.2 Reinforcement Learning

Reinforcement Learning (RL) [12] is a Machine Learning (ML) sub-field entrusted to solve optimization problems. Structure-wise, RL frameworks (Fig. 2.2) are composed of an agent and its environment. A bi-directional communication is established between both, enabling the agent to learn by trial-and-error using the feedback, in the form of scalar rewards, from the outcomes of its executed actions. The elements of conventional Reinforcement Learning methodologies are presented in Table 2.1. Amongst them, it is possible to underline the states, actions, and rewards:

- States - environment representations;
- Actions - commands executed by the agent;
- Rewards - scalar values, evaluations of the adopted actions considering the triggered state transitions.

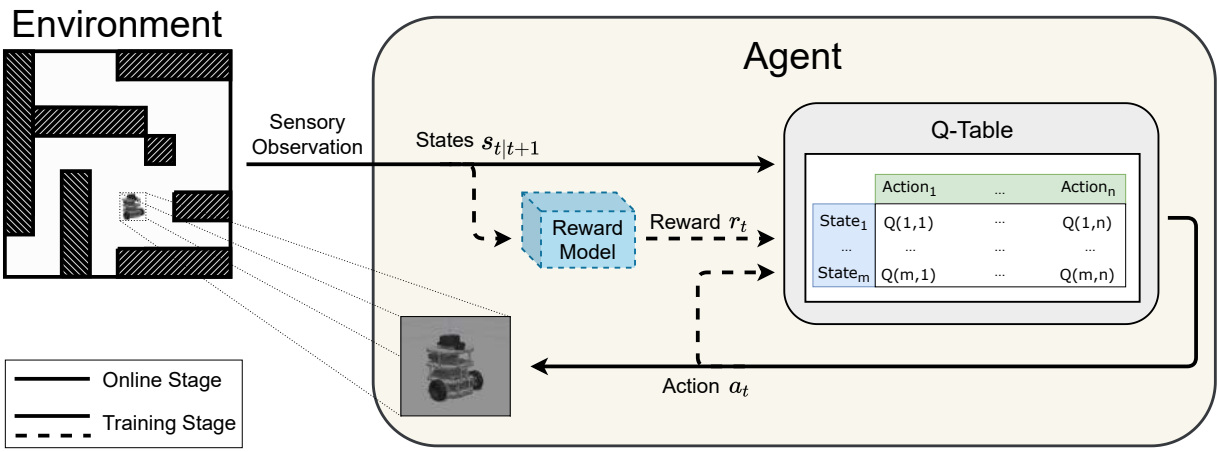


Figure 2.2: Reinforcement Learning framework.

Reinforcement Learning algorithms are iterative methods divided into episodes, sets of discrete time steps. In each step t, the agent performs an environment observation, converts its findings into a state s_t, and selects an action a_t based on it. The agent then perceives the following state s_{t+1} and receives a reward r_t from the environment. This process continues until the agent reaches a terminal state or a step limit T, after which the environment resets, and a new episode starts. The return of each episode e is the total accumulated step rewards, discounted by a factor γ:

$$R_e = \sum_{i=0}^{T} \gamma^i \cdot r_{t+i} \tag{2.1}$$

The goal of learning is that the intelligent agent, using its experience, improves its decision-making ability to maximize the accumulated rewards R.

Table 2.1: Reinforcement Learning parameters.

Parameter	Designation	Description
a_t	Action	Command executed by the agent, selected from a set of valid actions $\mathbb{A}$
s_t	State	Instantiation of the state space $\mathbb{S}$, a set of environment representations
r_t	Reward	Immediate scalar reward returned according to a state transition $s_t \rightarrow s_{t+1}$ provoked by an executed action a_t
$P(s_{t+1}\|s_t, a_t)$	Transition Model	Representation of how the environment changes in response to the agent's actions
$\pi(s_t)$	Policy	Mapping function that specifies which action the agent must adopt in each state
γ	Discount Factor	Sets the balance between immediate and future rewards. The higher the value, higher the influence of long-term rewards
$V(s_t)$	Value Function	Expected total return that an agent can receive from the state s_t onward
$Q(s_t, a_t)$	Action-Value Function	Expected total return for selecting the action a_t in the state s_t
α	Learning Rate	Sets the impact of past experiences in the learning process

2.2.1 Policy Evaluation

As previously noted, Reinforcement Learning aims to maximize the total expected rewards. Therefore, the best state-action policy π, a value function that maps states into actions, must be inferred. According to the best-policy deduction and further practice, RL methods can be categorized as follows:

- Model-free - the agent operates by refining a value function directly from experience, relying exclusively on a trial-and-error logic;
- Model-based - the agent adjusts, according to the environment observations, a dynamic transition model $P(s_{t+1}|s_t, a_t)$ that returns which action to take based on each transition probability.

To assess and improve a policy, Reinforcement Learning resorts to various algorithms referred to as Policy Evaluation techniques [12]. Dynamic Programming (DP), Monte Carlo (MC), and Temporal Difference (TD) are their most noteworthy methodologies.

Dynamic Programming

Model-based methods predominantly use Dynamic Programming (DP) [12] to improve the followed policy/value function. The transition and reward models are known entities, enabling an accurate computation of the expected reward sum over future states. The value function $V(s_t)$ is updated according to the following equation:

$$V(s_t) \leftarrow r_t + \gamma \cdot \sum_{s_{t+1} \in S} P(s_{t+1}|s, a_t) \cdot V(s_{t+1}) \tag{2.2}$$

The last term of (2.2) is the discounted total future rewards. It results from the sum, for every possible state s_{t+1}, of the products of the transition model P with the estimated returns $V(s_{t+1})$. The concept of using the expectation of successor states to update $V(s_t)$ is designated *bootstrapping*.

Monte Carlo

Monte Carlo (MC) [12] methods, unlike DP, do not bootstrap. Instead, the value function updates are computed tuple-by-tuple (s_t, a_t, s_{t+1}, r_t) from a sampled environment:

$$V(s_t) \leftarrow V(s_t) + \alpha \cdot [r_t - V(s_t)] \tag{2.3}$$

Temporal Difference

Temporal Difference (TD) [12] approaches can be defined as Dynamic Programming and Monte Carlo hybrids. TD updates the value function by combining the environment's sampling to approximate an expectation over-states (next-state distribution) from MC, and the *bootstrapping* notion to estimate the discounted sum of future rewards from DP:

$$V(s_t) \leftarrow V(s_t) + \alpha \cdot [r_t + \gamma \cdot V(s_{t+1}) - V(s_t)] \tag{2.4}$$

2.2.2 Exploration vs. Exploitation

One fundamental dilemma in Reinforcement Learning is the exploration versus exploitation trade-off [13]. Exploration refers to the probing process of experimenting random actions to explore the environment. As it may take several time steps to evaluate which action is optimal at each state, exploration is recommended to determine the long-term actions that lead to high rewards. Exploitation, in turn, is the operation of selecting the actions with the highest estimated values, $Q^*(s, a)$ (see Table 2.1), to encourage the best possible outcome.

In model-free Reinforcement Learning algorithms, the transition between exploration and exploitation is established by ϵ-greedy strategies. In these techniques, actions are chosen randomly with a probability of ϵ. Reciprocally, actions with the highest estimated value, $Q^*(s, a)$, are chosen with complementary probability. Herewith, by decaying ϵ over time, the agent progresses from exploration towards exploitation [13].

2.2.3 Q-Learning

Q-Learning [12, 13] is a RL model-free Temporal Difference algorithm used to optimize, by trial-and-error, the intelligent agent's decision-making aptitude.

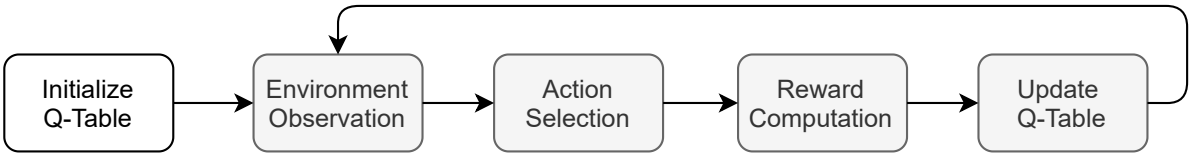


Figure 2.3: Q-Learning block diagram.

As illustrated in Fig. 2.3, Q-Learning starts by initializing a *Q-Table* (Fig. 2.4), each row corresponding to a state, $s \in \mathbb{S}$, and each column to an action, $a \in \mathbb{A}$.

	$Action_1$	...	$Action_n$
$State_1$	Q(1,1)	...	Q(1,n)
...	...	...	...
...	...	...	...
$State_m$	Q(m,1)	...	Q(m,n)

Figure 2.4: Q-Table.

In an initial phase, the agent interacts with the environment exploring new states and actions using an ϵ-greedy exploration method. In this period, the *Q-Table* is filled with each state-action pair *Qvalue*. *Qvalues*, $Q(s, a)$, as previously described in Table 2.1, represent the agent's expected total rewards from taking the action a in the state s. Each *Qvalue* results from the computation of the Bellman equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot [r(s_t, a_t) + \gamma \cdot max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.5}$$

- $r(s_t, a_t)$: Immediate reward for transitioning from the state s_t to the state s_{t+1};
- $max_a Q(s_{t+1}, a)$: Rewards that may be returned several time steps deeper into the sequence, in the form of the optimal *Qvalue* of the successor state s_{t+1};
- $Q(s_t, a_t)$: TD prediction;
- $r(s_t, a_t) + \gamma \cdot max_a Q(s_{t+1}, a)$: TD target;
- $r(s_t, a_t) + \gamma \cdot max_a Q(s_{t+1}, a) - Q(s_t, a_t)$: TD error ($\delta_t$).

The objective of exploration can be interpreted as subjecting the agent to various environment states to fill the *Q-Table*. The more diversified environment representations the agent gets exposed to, the more comprehensive the table grows. Once the agent progresses from exploration to exploitation, it consults the *Q-Table* and chooses the actions that maximize the total accumulated rewards, $Q^*(s, a)$:

$$a_t \leftarrow argmax_{a \in Q(s,:)}(Q(s_t, a)) \equiv Q^*(s_t, a) \tag{2.6}$$

2.3 Deep Learning

Deep Learning (DL) [23, 14] is a subset of ML that uses Artificial Neural Networks (ANNs) to extract essential features from data structures [24]. ANNs are computational models inspired by biological neural networks, and similarly, their core elements are called neurons. An artificial neuron can be characterized as a function, as it takes input data, processes it, and returns an output signal. In DL, neurons are linked and organized into layers, which in turn can be categorized according to their placement within ANN architectures:

- Input layer - the very beginning of an ANN workflow. Acquires data from the outer DL system and forwards it for further processing;
- Output layer - termination of an ANN workflow. Returns the data treated by prior layers back to the DL system;
- Hidden layers - input and output in-between layers designed to identify and process data features. ANNs restricted to one hidden layer are termed Shallow Networks. Architectures with two or more hidden layers are named Deep Networks.

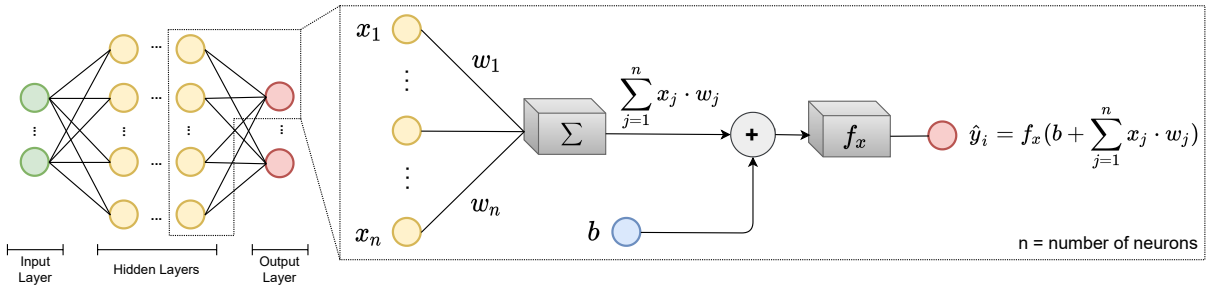
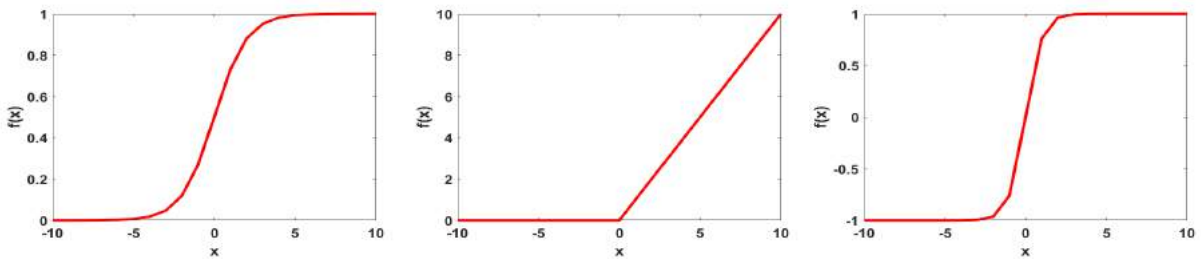


Figure 2.5: Deep Artificial Neural Network architecture.

The output signal of each neuron, $\hat{y}_i$, results from the computation, performed by a non-linear activation function f_x, of the sum of a bias b with the product between the signals of active neurons in previous layers x_j and the weights associated with the channels that establish the neuron connections w_j (see Fig. 2.5):

$$\hat{y}_i = f_x(b + \sum_{j=1}^{n} x_j \cdot w_j) \tag{2.7}$$

The terminology behind activation functions precedes their operating principle, as they determine if a neuron signal is transmitted. As activations in one layer affect activations in succeeding ones, activation functions (Fig. 2.6) are crucial elements of Deep Learning algorithms.



(a) Sigmoid (b) ReLU (c) tanh

Figure 2.6: Sigmoid, ReLU, and Hyperbolic Tangent activation functions.

2.3.1 Convolutional Neural Networks

Convolutional Neural Networks [14, 24] are specialized networks for processing grid-like data [25]. Stage-wise, CNNs can be disassembled into feature extraction and classification (Fig. 2.7). Feature extraction is composed of several convolution layers with non-linear activation functions, followed by pooling layers. Each convolution layer acts as a filter, extracting features from the input data or previous layered neuron signals. The pooling layers are used to compress the convolution layers' outputs. The return of the last feature extraction layer serves as input to the fully connected layers of the classification phase, the stage responsible for processing the high-level features and ultimately generating the CNNs outputs.

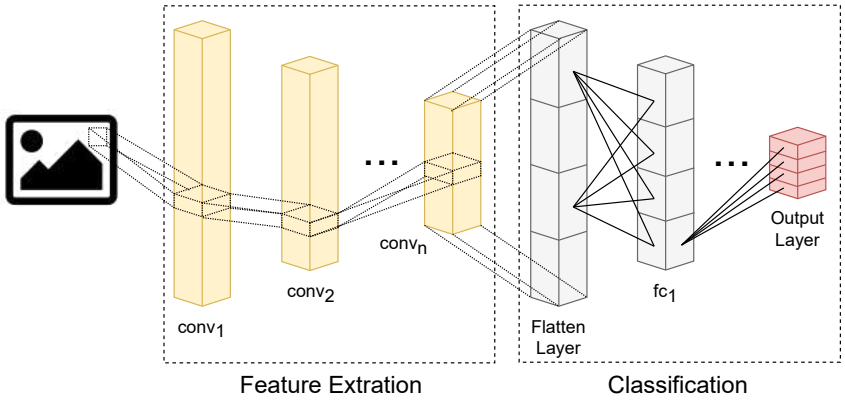


Figure 2.7: Convolutional Neural Network architecture.

2.3.2 Recurrent Neural Networks

Networks in which neuron signals are broadcasted uniquely to next-layered neurons are defined as Feedforward ANNs. Contrarily, networks that allow connections and signal propagation between neurons in the same or previous layers are determined as Recurrent Neural Networks (RNNs) [14, 24]. RNNs (Fig. 2.8) are a class of ANNs where connections between nodes form a directed graph along a temporal sequence. Derived from Feedforward networks, RNNs have a particular architecture that enable information to persist over time due to its internal state (memory). These configurations are well suited for sequence modeling tasks, being, for example, the backbone of many Neural Machine Translation applications [26], like Google Translator [27, 28].

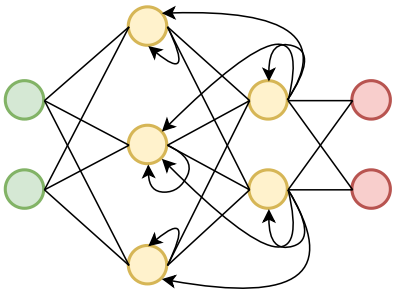


Figure 2.8: Recurrent Neural Network architecture.

2.3.3 Artificial Neural Network Training

Artificial Neural Networks are the core constituents of Deep Learning frameworks. However, when designed and firstly initialized, their weights $\mathbf{w}$ and biases $\mathbf{b}$ are randomly generated, raising the need to be fine-tuned before being applied towards their intents.

In supervised learning, the procedure of tuning the ANN parameters $\theta \equiv (\mathbf{w}; \mathbf{b})$, also termed training, is executed with labelled datasets. These sets of structured input-output data give the system the means to compare the network's output signals $\hat{y}_i$ with expected values from the dataset y_i. In each training iteration, the difference between y_i and $\hat{y}_i$ serves as object to a loss function $L(\theta)$ that, in turn, yields a loss value, an indication of the global disparity between the network and dataset outputs. The higher the loss value, the worse the network performance is. Thereby, in supervised learning, training is inferring the optimal set of weights and biases that minimize the cost function $L(\theta)$:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.8}$$

The predominant methodologies utilized to reduce the loss value are the gradient descent approaches [23, 24]: the representation of how the weights and biases need to change to decrease the loss is set by the negative gradient of $L(\theta)$. Under these techniques, the most applied algorithm to train multi-layer networks is Backpropagation [29]. Backpropagation efficiently computes, by the chain rule, the gradient of $L(\theta)$ given the loss, weights, and biases. This operation is done one layer at a time, iterating backward from the output to the input layer, avoiding redundant calculations of intermediate terms.

While supervised learning algorithms tend to be more accurate than unsupervised ones, they require upfront human intervention to label the data appropriately. In contrast, unsupervised learning methods are presented with unlabeled data only. Thus, without prior knowledge nor datasets to lead the training process, the goal of unsupervised DL architectures is to uncover, extract, analyze, and model hidden data patterns to adjust the network parameters θ.

Whether developing supervised or unsupervised systems, the utmost objective of training ANNs is that the resultant tuned model performs well when tested towards both training and similar data. However, during experimental trials, some behavioral issues can surface from the intricate and possibly inaccurate learning process. The most common problems encountered while training Deep Artificial Neural Networks are overfitting and underfitting [30, 25]:

- Overfitting - the network achieves good performances under training data only;
- Underfitting - the network training proves insufficient [30]. The network exhibits poor performances for both training and similar data.

Several approaches have been proved to solve overfitting, such as regularization techniques [25], network hyperparameter regulation [31], and data augmentation [30]. To deal with underfitting, the best methods are to increase the model complexity and train longer.

2.4 Deep Reinforcement Learning

Reinforcement Learning is an efficient way to learn control models without referencing the ground truth [32]. However, RL algorithms tend to grow computationally costly over extensive and dynamic environments due to their tabular frameworks. Such problem can be overcome by introducing the hierarchical data processing of Deep Learning into RL architectures. Methods that embrace this type of structure (Fig. 2.9) are within the scope of Deep Reinforcement Learning (DeepRL), an innovative field that has been producing applications across the entire spectrum of science [33].

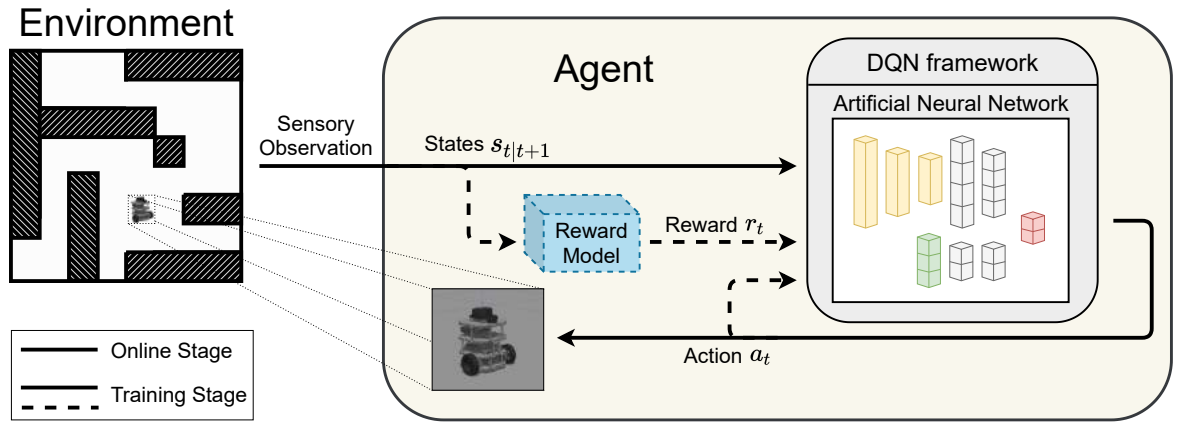


Figure 2.9: Deep Reinforcement Learning framework.

2.4.1 Value-based and Policy-based Learning

One of the problems involving Reinforcement Learning is that the amount of memory required to store data rapidly expands as the number of states increases. Deep Reinforcement Learning tries to tackle this well-defined problem by using Artificial Neural Networks as function approximators [33], generalizing from seen to unseen states when the state space is large or continuous.

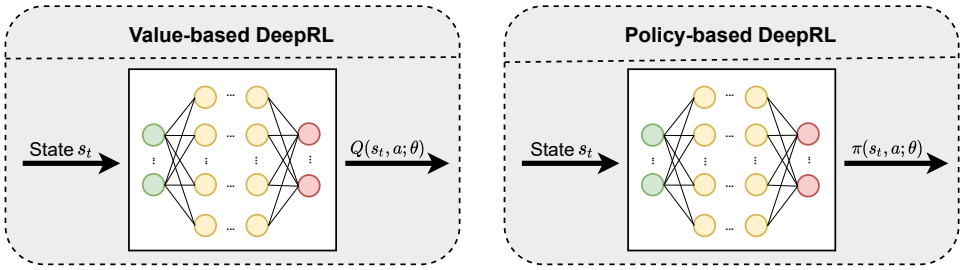


Figure 2.10: Value-based and Policy-based Deep Reinforcement Learning.

DeepRL strategies follow one of two methodologies: Value-based or Policy-based (see Fig. 2.10). In Value-based learning, ANNs act as action-value functions, $Q(s, a; \theta)$. Upon fine-tuned, they estimate the state-action *Qvalues* and infer a deterministic signal of which action to take given the current state [34]. Contrarily, Policy-based approaches employ ANNs to parameterize the policy $\pi(s, a; \theta)$ and optimize the action space using Policy Gradient techniques [35, 36].

2.4.2 Deep Q-Learning

Deep Q-Learning [17], also denoted as Deep Q-Network (DQN) due to its architecture, is a Value-based learning method designed upon the Q-Learning algorithm described in Section 2.2.3. Both approaches share the same principle of computing the state-action *Qvalues*, diverging however in the *modus operandi*: while Q-Learning resorts to a *Q-Table* to guide the training process and consequent agent's decision-making, in DQN such tasks are conducted by Artificial Neural Networks.

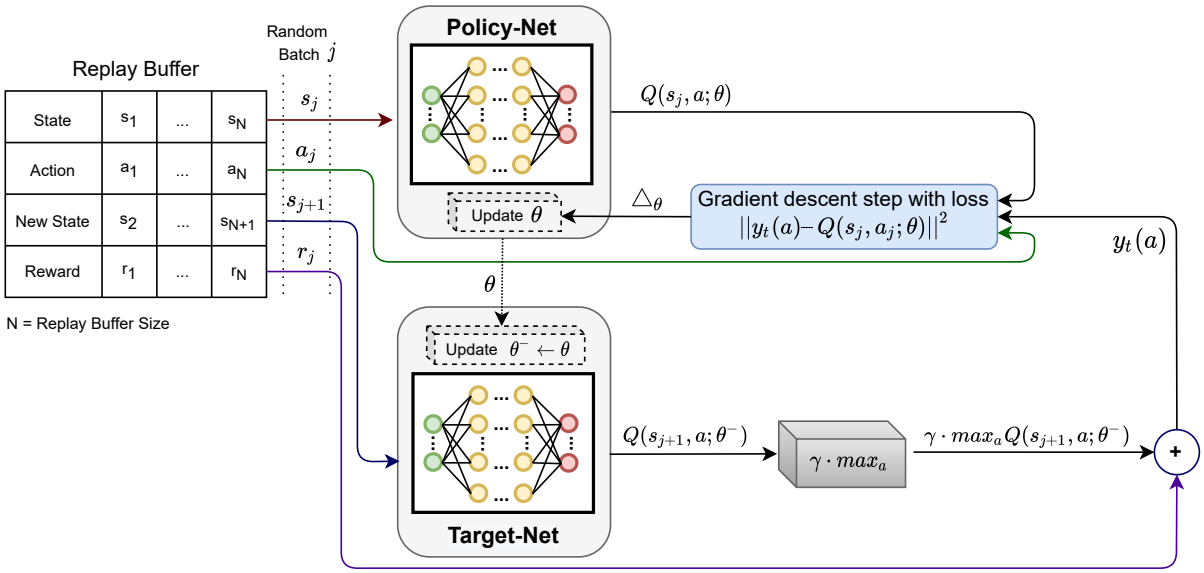


Figure 2.11: Deep Q-Learning training stage.

As shown in Figure 2.11, Deep Q-Learning has three main components: a policy network, a target network, and a replay buffer. The policy network (θ) is responsible for estimating the $Qvalues$ for the current state transition, whereas the target network (θ^-) computes the optimal $Qvalue$ of the successor state. In DQN, each $Qvalue$ is calculated using the Bellman optimal equation (2.5, $\alpha = 1$):

$$Q(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \cdot max_a Q(s_{t+1}, a) \tag{2.9}$$

The replay buffer [37], in turn, is a tabular arrangement that stores the transition tuples (s_t, a_t, s_{t+1}, r_t) witnessed in each training step t.

Analogously to Q-Learning, in Deep Q-Learning the agent starts by interacting with the environment, exploring new states using an ϵ-greedy exploration method and filling the replay buffer with transition tuples. At the end of each step, a random batch of tuples (s_j, a_j, s_{j+1}, r_j) is sampled and subsequently fed to the policy and target networks. A loss value is further calculated based on the networks' output $Qvalues$:

$$L(\theta) = ||y_t(a) - Q(s_j, a_j; \theta)||^2 \equiv ||r_j + \gamma \cdot max_a Q(s_{j+1}, a; \theta^-) - Q(s_j, a_j; \theta)||^2 \tag{2.10}$$

The error between the targets $y_t(a)$ and predictions $Q(s_j, a_j; \theta)$ of (2.10) is then backpropagated to tune the policy network parameters θ:

$$\triangle_\theta = \alpha \cdot L(\theta) \cdot \nabla_\theta Q(s_j, a_j; \theta) \tag{2.11}$$

With the employment of a replay buffer, the policy network updates do not depend exclusively on the last environment observation. Instead, random state-transition tuples (s_j, a_j, s_{t+1}, r_j) are utilized to adjust θ, ensuring that the agent keeps training over its new behavior, consolidating it rather than over circumstances that might no longer be relevant.

Unlike the policy network parameters θ - tuned at every training step - the target's weights and biases θ^- are updated periodically by inheriting the policy network parameter values, $\theta^- \leftarrow \theta$. This operation prevents the target function from rapidly changing, improving the robustness of the training process.

Optimally, after several steps, episodes, and network updates, the learning process generates a refined network model (θ^*). In the online stage, this model sustains by itself the agent's decision-making. Therefore, the DQN framework no longer needs the replay buffer nor the target network (see Fig. 2.12), engineering a significant computational cost reduction. In this testing phase, the system operating mode is reasonably straightforward. The employed network (θ^*), in each step t, processes the input state representation s_t and computes the $Qvalues$ $Q(s_t, a; \theta^*)$ accordingly. The action a_t chosen to be executed is the one associated with the highest expected total return, $Q^*(s_t, a; \theta^*)$:

$$a_t \leftarrow argmax_{a \in Q(s,:;\theta^*)}(Q(s_t, a; \theta^*)) \equiv Q^*(s_t, a; \theta^*) \tag{2.12}$$

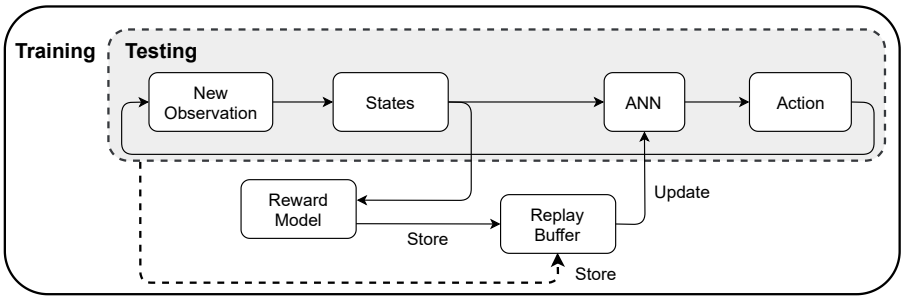


Figure 2.12: Deep Q-Learning block diagram.

2.4.3 Dueling Deep Q-Network Architectures

Due to the groundbreaking results of DQN-based algorithms, several adjustments to the original method naturally arose to further improve its functioning [38, 18]. One of the modifications that proved to increase the reliability and learning rate of Deep Q-Learning was the replacement of the single-stream network architectures with dueling structures [18]. Dueling networks (Fig. 2.13b), within Value-based algorithms, decouple the estimation of *Qvalues* into two streams to compute the state value $V(s_t)$ and the advantage affection $A(s_t, a_t)$. The advantage portrays the benefit of selecting a specific action compared to the others available. Its value is calculated by subtracting the predicted state value $V(s_t)$ from the optimal state-action $Q^*(s_t, a_t)$:

$$A(s_t, a_t) = Q^*(s_t, a_t) - V(s_t) = r_t + \gamma \cdot V^*(s_{t+1}) - V(s_t) \tag{2.13}$$

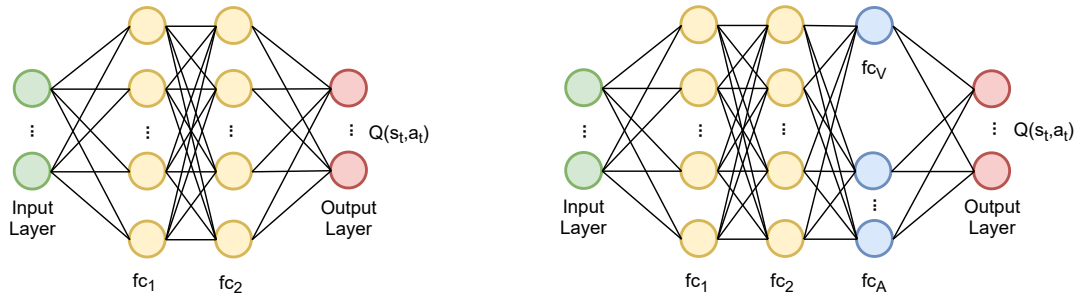
The *Qvalue* $Q(s_t, a_t)$ estimation in dueling models results from the advantage of taking the action a_t in the state s_t, $A(s_t, a_t)$, subtracted by the mean of the advantages of all actions, plus the state value $V(s_t)$:

$$Q(s_t, a_t) = A(s_t, a_t) - \frac{1}{k}\sum_{a=1}^{k} A(s_t, a) + V(s_t) \qquad \text{k = action space size} \tag{2.14}$$

The advantage mean plays a regularization role, as it is applied to the advantage stream output values to reduce the existent noise.

The superiority of dueling network architectures is revealed during training. As shown in Fig. 2.13 (b), the value stream layer fc_V is composed of a single neuron, whereas the number of neurons of the advantage stream fc_A is equal to the action space size k. In DQN, only the output neuron corresponding to the highest *Qvalue* contributes to the network backpropagation updates. Dueling DQN, in turn, has the advantage stream reproducing this single-stream DQN functioning and the additional state value neuron, adjusted in each iteration, that provides the intakes of every experience to the policy network tuning.

In conclusion, the difference between DQN and its dueling variation resides in the *Qvalue* computation. If complemented with identical Deep Reinforcement Learning modules, such as the action, state, and reward models, both implementations can be represented with the same Algorithm 2 of Appendix A.



(a) Single-stream DQN

(b) Dueling DQN

Figure 2.13: Single and dual-stream network architectures.

2.4.3.1 Double Deep Q-Network and Dueling Double Deep Q-Network

A study by van Hasselt *et al.* [38] uncovered that DQN, despite its innovative functioning and unprecedented outcomes, tends to suffer from substantial state-action value overestimations that ultimately lead to impoverished policies. To prevent this possible issue, researchers proposed a variant named Double DQN. Double Deep Q-Learning aims to reduce overestimations by decomposing the Deep Q-Learning target into action selection and evaluation:

$$y_t(a) = \begin{cases} r_t + \gamma \cdot max_a Q(s_{t+1}, a, \theta^-), \text{ if DQN} \\ r_t + \gamma \cdot Q(s_{t+1}, argmax_a Q(s_{t+1}, a; \theta); \theta^-), \text{ if Double DQN} \end{cases} \tag{2.15}$$

The selection of the action in the $argmax$ operation is executed by the policy network (θ). The target network (θ^-) is then employed to fairly evaluate the value of the policy.

By combining the technique proposed by van Hasselt *et al.* with dueling architectures, the Dueling Double Deep Q-Network (D3QN) algorithm can be deduced. D3QN [18] preserves the dueling DQN configuration to extract its attested benefits (enhanced training strategy) and utilizes the action selection and evaluation methodology of Double DQN to resolve the DQN overestimation problem. With this formulation, D3QN can easily be constructed upon DQN with minimal computational overhead, as shown in Algorithm 3.

3

State of the Art

In this chapter, some of the most renowned robot motion planning algorithms are covered while giving particular emphasis to navigation approaches based on RL and DeepRL. A brief report of the DeepRL field evolution is also presented.

3.1 Motion Planning

The motion planning problem in robotics has motivated multiple implementations, each characterized according to the used planner, environmental modeling, and path search algorithm.

3.1.1 Global Path Planning

The predominant global path search methods in indoor navigation are classified as heuristic-search algorithms [20]. Under this category are methodologies such as Dijkstra [39, 40], Floyd-Warshall [40], A* [20, 41, 42], and Rapidly-Exploring Random Trees (RRT) [43].

In Dijkstra, paths are created through the selection of neighbor nodes in topological environment representations. Each formulated path has an associated cost resultant from the weighted arcs that establish the route's node connections. The path selected is the one with the minimum cost. While Dijkstra calculates the shortest path from one node to all nodes, Floyd-Warshall determines the shortest path between all pairs of nodes. A*, in turn, follows an heuristic evaluation function to determine the shortest path between the initial and final nodes [41]:

$$f(i) = g(i) + h(i) \tag{3.1}$$

- i - robot current localization;
- $g(i)$ - past-cost function from the starting node to i;
- $h(i)$ - Euclidean distance from i to the target node.

The RRT is a sampling-based method notably implemented in large low-dimensional state spaces. In each iteration, a random node is selected within a maximum range and is connected to the nearest one from a congregation termed tree. As node choices are purely random, several tree branches (path representations) are generated. Conceptually, at least one route progressively converges towards the goal area.

3.1.2 Local Path Planning

Alongside global path planning, robust motion planning methodologies can be implemented using local path search algorithms over dynamic metric feature maps.

Induced by the Artificial Potential Field (APF) [20, 44] strategy, a platform is subject to an artificial force field. Attractive and repulsive forces are applied to the robot by the goal destination and surrounding objects, respectively, originating a resultant force that defines the movement course. Dynamic Window Approach (DWA) [45], in turn, reduces the search space to a dynamic window centered around the current velocities of a mobile platform. Guided by an objective function, DWA aims to approximate a robot to its desired destination by selecting, iteratively, the optimal solution within the window composed of reachable/admissible speeds. The Time Elastic Bands (TEB) [39, 46] approach has a similar working principle: with the knowledge of the vehicle's geometric, kinetic, and dynamic constraints, this methodology generates a sequence of speed commands that allow a robot to navigate through an established set of intermediate waypoints.

3.1.2.1 Reinforcement Learning in Indoor Navigation

RL-based navigation frameworks fit under local path planning. These motion planning approaches are instrumental in unknown or dynamic environments, where a continuous learning process is required to comply with the changes verified over time.

One of the most challenging tasks to optimize Reinforcement Learning methods is the definition of their state, action, and reward models. A list of RL navigation implementations and respective models is shown in Table 3.1.

Table 3.1: RL-based motion planning strategies and respective models, adapted from [1].

Article	States	Actions	Rewards
Zuo *et al.* [47]	Robot-obstacle distances	Linear-angular speed turns	State-action based
G. Yen and T. Hickey [48]	3x3 costmap environment representations	Unknown	Penalties for each step and collision; Reward for reaching the goal
Garrote *et al.* [1]	Environment and path representation data (binary strings)	Linear-angular speed pairs	Defined according to: – distance to target; – distance to the nearest obstacle; – deviation from the predefined path

3.2 Deep Reinforcement Learning

DeepMind [49], in 2013, revolutionized the Artificial Intelligence field by using Deep Q-Learning to develop methods capable of beating human experts at Atari Video Games [17, 50]. Driven by the unmatched results observed at the time, the AlphaGo [51], AlphaZero [52], and AlphaStar [53] applications were subsequently invented. In 2015, AlphaGo achieved remarkable performances in the game of Go by combining Deep CNNs to represent the Go knowledge and RL to train from self-play. In 2017, a similar approach of AlphaGo, named AlphaZero, was released that mastered besides Go, the games of Chess and Shogi. In 2019, AlphaStar, using DeepRL, dominated the complex game of StarCraft II, beating some of the best teams in the world in real-time gameplay.

Following the success of Deep Reinforcement Learning in the gaming field, and inspired by the results of its frameworks when directly trained from raw images, a new era of research in autonomous navigation using visual information started.

3.2.1 Deep Reinforcement Learning in Indoor Navigation

Deep Reinforcement Learning-based navigation techniques, similarly to RL, fit under the category of local path planning. As mentioned in Section 3.1.2, RL algorithms feature the demanding task of determining the action, state, and reward models that reveal the best training performances and post-training agent behavior. On top of such endeavor, in DeepRL systems developers also need to structurize the framework's ANNs appropriately. The fine-tuning of these DeepRL modules can be an intricate and lengthy process. However, once they complement each other properly, excellent applications with an improved sense of adaptation and generalization towards unknown states can be attained, as shown in this sub-section.

An example of a DQN-based strategy towards cognitive exploration in unknown indoor environments is presented by Lei Tai and Ming Liu in [32]. With a CNN processing the raw depth images obtained from a Kinect RGB-D camera, the operated mobile platform successfully learned to execute a collision-free exploration in unfamiliar virtual scenarios. The same researchers proposed in [54] a mapless motion planner approach using a single SICK TiM570 laser mounted on a robot. With a state model composed of ten sparse laser findings and the relative target position (concerning the mobile robot coordinate frame), the mobile platform managed to attain the desired targets without colliding with any obstacles. The DeepRL method utilized to sustain the training routine was an asynchronous Deep Deterministic Policy Gradient (DDPG) based algorithm [55].

In [56], Liang *et al.* present an application that enables a mobile robot to perform real-time collision avoidance in dense crowds. Using a Policy-based DeepRL procedure named Proximal Policy Optimization (PPO) [57], the agent implicitly learns from different kinds of interactions with dynamic and static obstacles.

Xie *et al.* [58] and Ruan *et al.* [7] propose comparable works using the state-of-the-art D3QN architecture to establish an end-to-end mobile robot navigation with dynamic obstacle avoidance. While Ruan *et al.* use a platform equipped with a Kinect RGB-D camera in both simulated and real domains to perform the respective validations, Xie *et al.* utilize this type of sensor only in virtual environments. With a monocular RGB camera mounted on a real mobile robot instead, the depth images in training had to be corrupted with random noise to ensure that the resultant fine-tuned models were transferable to reality.

Chen *et al.* [15], in turn, suggest a method that enables a fully autonomous robotic navigation using a DeepRL Value-based approach, V-Learning. The platform, moving at human walking speed in a crowded environment, manages to detect and track up to three other agents while executing a socially aware motion planning. By adjusting the method's reward model, the robot proves capable of adopting the right or left-hand social norms.

To face the aforementioned agent's detection and tracking number limitation, Michael Everett, Yu Fan Chen, and Jonathan P. How [16] propose a solution with a RNN architecture to observe an arbitrary number of agents. The presented DeepRL-based motion planning among decision-making agents employs a Policy-based GPU/CPU Asynchronous Advantage Actor-Critic (GA3C) [59, 36] learning approach, a strategy that uses a system of queues and a dynamic scheduling technique for training Deep Artificial Neural Networks.

As evidenced, Deep Reinforcement Learning accomplishes considerable results with different paradigms, frameworks, and ANN architectures. The simulators, Machine Learning libraries, and platforms used in the covered applications are listed, per article, in Table 3.2.

Table 3.2: DeepRL-based motion planning strategies and respective software and hardware materials.

Article	**Simulator**	**ML Library**	**Platform**
Lei Tai and Ming Liu [32]	Gazebo [60]	Caffe [61], on a NVIDIA GeForce GTX 690	Turtlebot with a Kinect RGB-D camera
Lei Tai and Ming Liu [54]	V-REP [62]	Unknown	Turtlebot a single SICK TiM570 laser
Liang *et al.* [56]	Gazebo [60]	Tensorflow [63], on a Nvidia GeForce RTX 2080Ti GPU	Turtlebot and ClearPath Jackal robots, with a 2D Lidar and RGB-D camera
Xie *et al.* [58]	Gazebo [60]	Unknown	Turtlebot with a Kinect RGB
Ruan *et al.* [7]	Gazebo [60]	Unknown	Turtlebot with a Kinect RGB-D camera
Chen *et al.* [15]	Unknown	Tensorflow [63], on an i7-5820K CPU	Platform equipped with a 2D Lidar, three Intel Realsense cameras, and four webcams
Michael Everett, Yu Fan Chen, and Jonathan P. How [16]	Unknown	Tensorflow [63], on a NVIDIA GTX 1060	Platform equipped with a 2D Lidar and three Intel Realsense R200 cameras

4

Developed Work

This chapter presents in-depth the strategy behind the development of the DQN-based training framework, employed ANNs, and state, action, and reward models of the proposed DeepRL local navigation system.

4.1 Proposed DeepRL-based Pipeline

The developed DeepRL-based local motion planning strategy, sketched in Fig. 1.1 (*Proposed Framework* - Section 1.2), incorporates a new feature responsible for mapping the retrieved sensory data into costmaps, grid-like environment representations. To comply with such novelty, the employed ANN and the state, action, and reward models had to be planned and subsequently created from scratch. The methodology behind their inference is discussed in the following sections.

The suggested approach unfolds into two different stages: training and testing. Training is the procedure of tuning the framework's ANN according to past experiences. The Deep Q-Learning and its dueling variations addressed, respectively, in Sections 2.4.2 and 2.4.3, were the methods exploited to accomplish such important task in the proposed pipeline. An overview of the executed training stage is illustrated in Fig. 4.1 and delineated step-by-step in Algorithm 1.

In the training stage, whenever the robot reaches its target, a network model - a copy of the employed DQN architecture and its current weights and biases - is saved. For a correctly designed DeepRL navigation framework, it is expected that at least one fine-tuned model gets saved during the learning process.

After the training completion, the system progresses to an online stage in which a mobile platform, controlled by a saved network exemplar, is introduced into a virtual environment to attest its decision-making policy. Optimally, training generates one or more regulated models that empower the agent to perform an effective motion planning towards its target, avoiding obstacles.

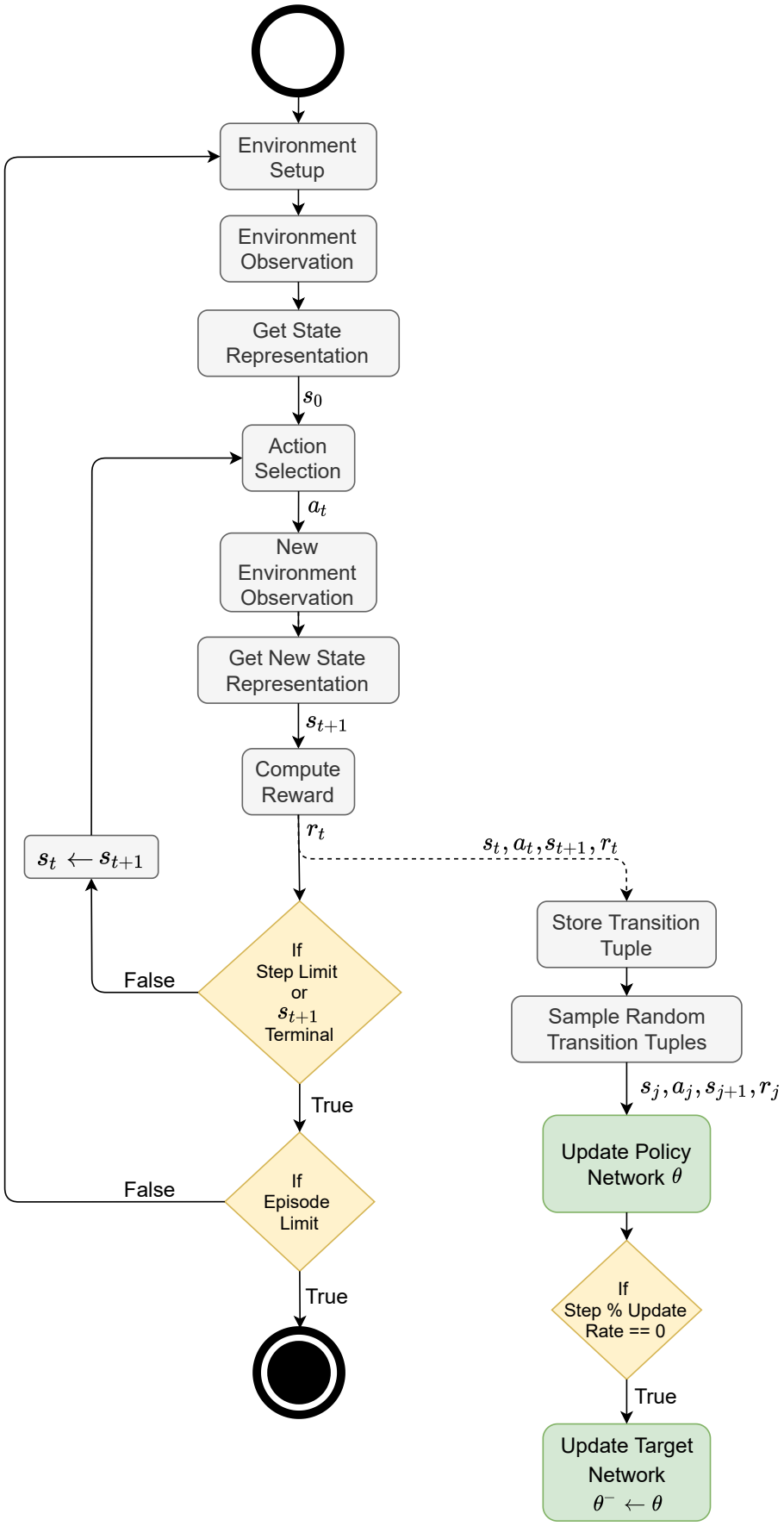


Figure 4.1: Training stage overview.

Algorithm 1: DeepRL-based Motion Planning for Indoor Robot Navigation

Define the initial and target points
Define the number of training episodes e and respective steps t
Define the network's hyperparameters and the state, action, and reward models
Initialize the system's ROS node and establish its subscriptions and publishers
Initialize the policy DQN (θ) and the target DQN (θ^-)
Initialize the replay buffer D to capacity N and define the batch size N_b
Define the target network update frequency N^-
for *episode* $e \in \{1, , M\}$ **do**
    Environment setup, s_0
    for *step* $t \in \{1, , T\}$ **do**
        Take action a_t from s_t using an ϵ-greedy exploration method
        Get the robot's odometry data (/odom Odometry callback)
        Get the sensor readings (/scan Scan callback)
            Construct the local costmap and compute C_{Stack}
            Compute the Euclidean distance to the goal d_T and nearest obstacle d_O
            Compute the orientation disparity between the robot and the target, ϕ
        Define s_{t+1}
        if s_{t+1} *is terminal* **then**
            if $d_T \leq d_{min}$ **then**
                Reward the agent, $r_t = R_D$
                Save DQN model
            end
            else
                Penalize the agent, $r_t = P_O$
            end
            break
        end
        else
            Compute the reward value r_t following the exploited reward model
        end
        Store transition tuple (s_t, a_t, s_{t+1}, r_t) in D
        Sample a random minibatch of N_b tuples (s_j,a_j, s_{j+1}, r_j) from D
        Compute $y_t(a)$ (2.15)
        Compute the policy network $Qvalues$, Q(s_j, a; θ)
        Perform a gradient descent step with loss $||y_t(a) - \mathrm{Q}(s_j, a_j; \theta)||^2$, updating θ
        if t % N^- *is equal to 0* **then**
            Replace the target DQN parameters $\theta^- \leftarrow \theta$
        end
        $s_t \leftarrow s_{t+1}$
    end
end

4.2 Artificial Neural Networks

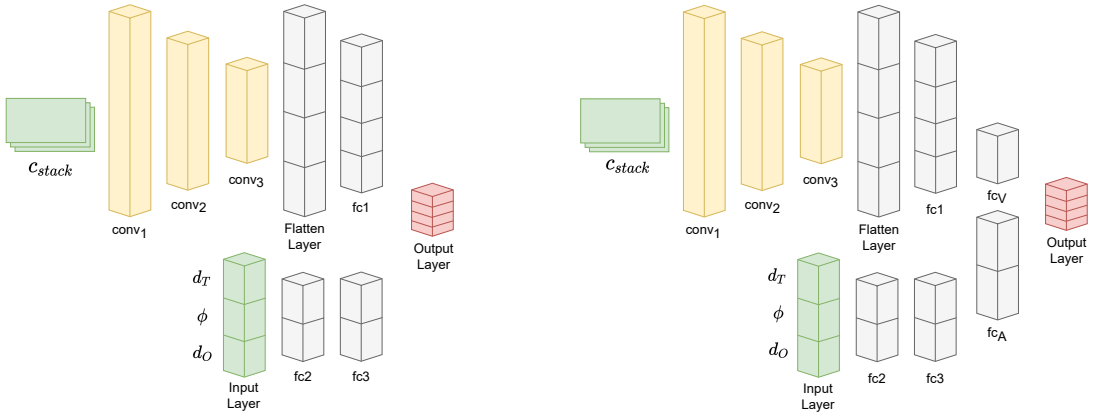
Alongside common DeepRL applications, the core unit of the developed local motion planning algorithm is the employed Artificial Neural Network. In this project, the architecture used in the standard Deep Q-Learning implementations and validations was the DQN presented in Fig. 4.2 (a). Alternatively, for the Dueling and Dueling Double Deep-Q Learning approaches, the dueling network depicted in Fig. 4.2 (b) was utilized instead.

The Deep Q-Network shown in Fig. 4.2 (a), an obstacle-oriented architecture based on the empirical study covered in Appendix B, can be interpreted as a merge of two different Artificial Neural Networks: a Convolutional Neural Network and a Deep Feedforward Network. The CNN segment is inspired by the exemplar proposed by Mnih *et al.* [17]. Structure-wise, it is composed of an input layer arranged to receive the costmap grid-like structure (C_{Stack}), followed by three convolutional, one flatten, and one fully connected layers. Parallel to the CNN, a two hidden-layered Deep Feedforward Network is used to handle the system's target-relative data $p = \{d_T, d_O, \phi\}$:

- d_T: Robot-target Euclidean distance;
- d_O: Robot-nearest obstacle Euclidean distance;
- ϕ: Robot-target orientation disparity.

Its k-neuron output layer, k being the action space size, results from the concatenation of the preceding CNN and Feedforward's fully connected layers.

The dueling DQN illustrated in Fig. 4.2 (b), compared to the standard and previously described DQN of Fig. 4.2 (a), features one additional advantage fc_A (k-neuron) and value fc_V (1-neuron) dual-stream layer. This extra layer is fed with the signals from the concatenation of the preceding CNN and Feedforward's fully connected layers, and computes the outputs of the dueling architecture according to the (2.14) expression.



(a) Deep Q-Network (b) Dueling Deep Q-Network.

Figure 4.2: Proposed ANN architectures.

To test the developed system with large action spaces, more elaborate ANN models - based upon the structures shown in Fig. 4.2 - were required to be designed in order to comply with the system complexity increase.

Table 4.1: Layer configurations of the proposed ANNs (Fig. 4.2).

Layer	Parameters
Input	$\{C_{Stack}, p\} \equiv \{C_{Stack}, d_T, d_O, \phi\}$
conv1	Filters = 32, Kernel size = 8, Strides = 4, Activation = relu
conv2	Filters = 64, Kernel size = 4, Strides = 2, Activation = relu
conv3	Filters = 64, Kernel size = 3, Strides = 1, Activation = relu
fc1	Neurons = 256, Activation = relu
fc2	Neurons = 64, Activation = relu
fc3	Neurons = 64, Activation = relu
fcV	Neurons = 1, Activation = None
fcA	Neurons = Number of actions, Activation = relu
Output	Neurons = Number of actions

4.3 State, Action and Reward Models

Aside from the Artificial Neural Network design, the definition of appropriate state, action, and reward models are equally important to optimize the functioning of Deep Reinforcement Learning applications.

4.3.1 State Models

As shown in the proposed DeepRL-based local motion planning pipeline pictured in Fig. 1.1, Deep Reinforcement Learning is formulated upon the interaction between an intelligent agent and its environment. This bi-directional relation provides the agent with prominent information regarding its surroundings, which is then converted into states, specific data structures supported by the framework's ANN.

In conventional DeepRL motion planning methods, the sensor-captured images/2D laser findings of the environment are used directly as ANN inputs. Consequently, network updates can be affected by intrinsic and extrinsic features such as image resolution and blurring, sensor distance range and angular resolution, sensor positioning, environment lighting and brightness, *etc* [64, 65]. To minimize these potential prejudicial generalization factors, a solution based on costmaps is presented. In this work, a costmap (see Fig. 4.3) is arranged as a robot-centric 1-meter length 40x40 grid, being each one of its cells initialized to 0 in every discrete step, and set to 1 if overlapped by any scan reading.

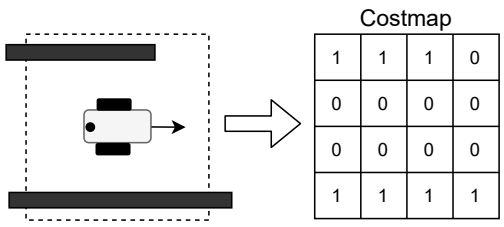


Figure 4.3: Costmap representation.

A costmap, as it only portrays the mobile platform's vicinity, fails to represent the robot's short-term paths and the continuous environment variations. For the agent to learn from this training-relevant data, the established state model $\mathbb{S}$ (4.1), formulated upon the robot's pose/localization information and onboard 2D laser observations (data acquisition process described in Section 5.2.1), was composed in part of a stack of 4 sequential costmaps C_{Stack} (Fig. 4.4) to represent the environment local mapping.

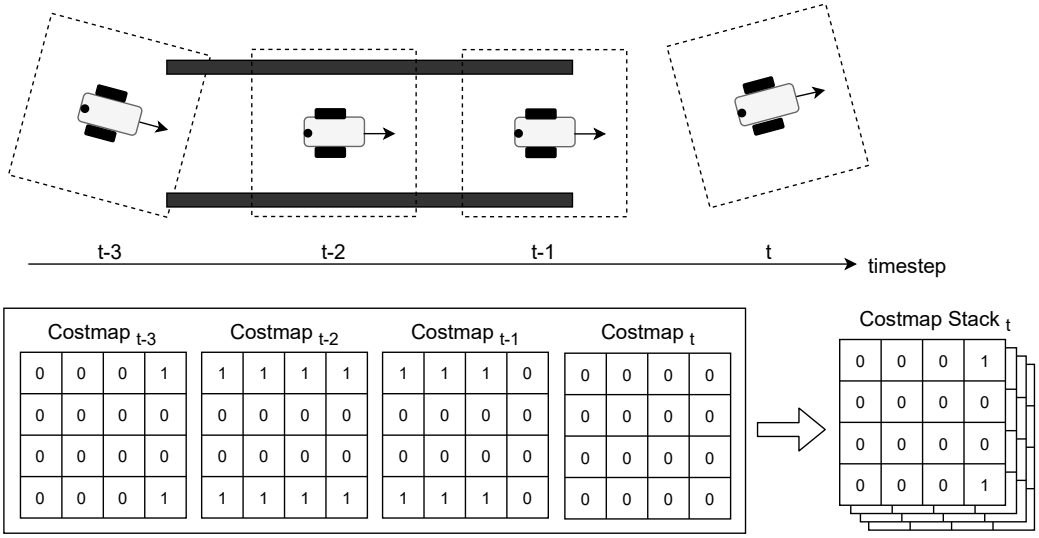


Figure 4.4: C_{Stack} data structure.

Complementing the agent's surrounding awareness through C_{Stack}, data concerning the robot-target and robot-nearest obstacle correlations $p = \{d_T, d_O, \phi\}$ (spatial offsets, see Fig. 4.5) was defined as the final integrating resource of the state model $\mathbb{S}$:

$$\mathbb{S} = \{C_{Stack}, p\} \equiv \{C_{Stack}, d_T, d_O, \phi\} \quad \begin{cases} d_T\text{: Robot-target Euclidean distance (m)} \\ d_O\text{: Robot-nearest obstacle Euclidean distance (m)} \\ \phi\text{: Robot-target orientation disparity (rad)} \end{cases} \tag{4.1}$$

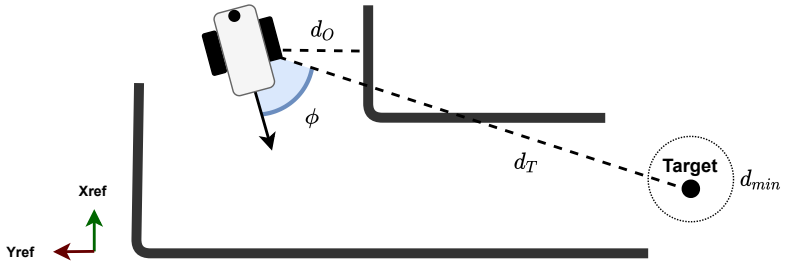


Figure 4.5: Robot-target-relative data representation.

A reduced state model $\mathbb{S} = \{C_{Stack}, d_T, \phi\}$ directed towards obstacle-free environments was also applied to validate the developed DeepRL-based framework on a starting experimental phase.

4.3.2 Action Sets

In Deep Reinforcement Learning, actions are the agent's response to an environment observation. Transposing such definition to the present navigation scope, actions are the agent's response, in the form of linear and angular speeds pairs (v, w), to sensory-based environment state representations. The action sets defined to control the robot's movement are presented in Table 4.2.

Table 4.2: Action sets.

Action Set	{**Linear vel.** (m/s); **Angular vel.** (rad/s)}	**Description**
Set 1 (3 Actions)	{0.1; 0.0}	Forward movement
	{0.0; 0.4}	Left turn
	{0.0; -0.4}	Right turn
Set 2 (3 Actions)	{0.1; 0.0}	Forward movement
	{0.025; 0.4}	Left turn w/ linear component
	{0.025; -0.4}	Right turn w/ linear component
Set 3 (3 Actions)	{0.15; 0.0}	Forward movement
	{0.0; 0.5}	Left turn
	{0.0; -0.5}	Right turn
Set 4 (6 Actions)	[{0.1; 0.0}, {0.15; 0.0}]	Forward movements
	[{0.025; 0.5}, {0.0375; 0.7}]	Left turns w/ linear component
	[{0.025; -0.5}, {0.0375; -0.7}]	Right turns w/ linear component
Set 5 (12 Actions)	[{0.16; 0.0}, {0.185; 0.0}, {0.21; 0.0}, {0.235; 0.0}]	Forward movements
	[{0.053; 0.5}, {0.062; 0.625}, {0.07; 0.75}, {0.078; 0.875}]	Left turns w/ linear component
	[{0.053; -0.5}, {0.062; -0.625}, {0.07; -0.75}, {0.078; -0.875}]	Right turns w/ linear component

4.3.3 Reward Models

In DeepRL frameworks, actions are evaluated qualitatively based upon the triggered environment variation: given the resultant environment state transition $s_t \rightarrow s_{t+1}$, a scalar reward value r_t is estimated according to the engaged linear reward model. r_t is the sole feedback that the agent receives regarding its decision-making, and is acknowledged as an indication whether the action a_t is advised to be executed on similar s_t states.

To fulfill the goal of driving a mobile robot towards its target with an obstacle-avoidance policy, three distinct reward models were outlined under the following premises:

- Reward the agent for every action that contributes to reducing the Euclidean distance to the target, d_T. Penalize it otherwise;
- Reward the agent for every action that contributes to decreasing the robot's orientation and target angular difference, ϕ. Penalize it otherwise;
- Reward the agent if the target is attained, $d_T < d_{Tmin}$;
- If navigating on environments with obstacles, reward the agent for every action that contributes to increasing the Euclidean distance to the nearest obstacle, d_O. Penalize it otherwise;

- If navigating on environments with obstacles, penalize the agent in case of collision, $d_O < d_{Omin}$.

In a preliminary experimental stage, a straightforward linear formula was utilized to help designing an ANN architecture capable of processing the state model inputs and accurately converting its features into *Qvalues*:

$$r_t = k_D \frac{d_{min}}{d_{T_{t+1}}} + k_\phi \frac{\delta}{\delta + |\phi_{t+1}|} \qquad \begin{cases} k_D = \text{Target distance const.} \\ k_\phi = \text{Orientation const.} \\ \delta = \text{Angular const.} \end{cases} \tag{4.2}$$

This procedure (described in Appendix B) was performed in obstacle-free environments with the reduced state model $\mathbb{S} = \{C_{Stack}, d_T, \phi\}$.

The reward model (4.2) generates positive rewards based on the system's prompt target-related resources $d_{T_{t+1}}$ and $|\phi_{t+1}|$. Contrarily, the reward model used to validate the developed navigation module in obstacle-free environments only transmits to the agent negative rewards:

$$r_t = -k_D \cdot d_{T_{t+1}} - k_\phi \cdot |\phi_{t+1}| \tag{4.3}$$

The main idea behind the inference of both reward models is that the agent determines which actions contribute to the highest episode reward sum. However, when following the model (4.3), due to the rewards being negative regardless of the provoked state transition, the agent is further encouraged not to take redundant actions and reach its target in the minimum number of episode steps.

A more complex reward model was employed to validate the algorithm in environments with static obstacles. This model can be interpreted as a confluence of three separated units, each one producing a reward value based on the target-related and obstacle-related attributes' transitions of the complete state model $\mathbb{S} = \{C_{Stack}, d_T, d_O, \phi\}$:

$$r_D = k_D \ \frac{d_{T_t} - d_{T_{t+1}}}{d_{T_{t+1}}} \qquad r_\phi = k_\phi \ \frac{|\phi_t - \phi_{t+1}|}{|\phi_{t+1}|} \qquad r_O = k_O \ \frac{d_{O_{t+1}} - d_{O_t}}{d_{O_{t+1}}} \tag{4.4}$$

Each scalar value r_D, r_ϕ, and r_O can either be positive, representing an improvement of the respective state resource, or negative otherwise. The step reward for non-terminal states is obtained from the summation of r_D, r_ϕ, and r_O. For terminal states, the agent is granted with a substantial reward R_D if the target point is attained or a severe penalty P_O in case of collision:

$$r_t = \begin{cases} R_D, \text{ if } d_T < d_{Tmin} \\ P_O, \text{ if } d_O < d_{Omin} \\ r_D + r_\phi + r_O, \text{ otherwise} \end{cases} \tag{4.5}$$

The reward sets utilized to shape the robot's decision-making faculty towards favorable actions are displayed in Table 4.3.

Table 4.3: Reward sets.

Reward Set	**Reward Model**	**{Parameter: Value}**
Set 1	Eq. 4.2	$\{k_D$: 70, d_{min}: 0.1, k_ϕ: 900, δ: 0.1$\}$
Set 2.1	Eq. 4.3	$\{k_D$: 5.0, k_ϕ: 0.05$\}$
Set 2.2	Eq. 4.3	$\{k_D$: 5.0, k_ϕ: 0.1$\}$
Set 2.3	Eq. 4.3	$\{k_D$: 5.0, k_ϕ: 0.8$\}$
Set 3.1	Eq. 4.5	$\{R_D$: 0.0, P_O: 0.0, k_D: 6.0, k_ϕ: 0.003, k_O: 0.0$\}$
Set 3.2	Eq. 4.5	$\{R_D$: 4.0, P_O: -4.0, k_D: 9.0, k_ϕ: 0.001, k_O: 0.008$\}$
Set 3.3	Eq. 4.5	$\{R_D$: 2.0, P_O: -2.0, k_D: 6.0, k_ϕ: 0.001, k_O: 0.0075$\}$
Set 3.4	Eq. 4.5	$\{R_D$: 2.0, P_O: -2.0, k_D: 8.0, k_ϕ: 0.001, k_O: 0.005$\}$

5

Software Tools and Hardware Materials

This chapter presents and briefly describes the software and hardware materials explored to fulfill the established objectives.

5.1 Operating System

Ubuntu [66] is a free and open-source operating system that enables unconditional development privileges and has official long-term support, constant maintenance, and a built-in firewall that helps reduce every possible security risk. Due to its compatibility with most hardware and software releases, and by offering the necessary instruments to easily combine various modules such as robot frameworks, simulators, and Integrated Development Environments (IDEs), Ubuntu has naturally become the conventional operating system to implement and simulate robotic applications.

For meeting all the requirements of this project, Ubuntu 20.04 was the stable version utilized to set up the proposed DeepRL-based motion planning approach. Its installation procedure was reported and can be found at https://github.com/DanielPalaio/Ubuntu_Setup.

5.2 Robot Operating System

The Robot Operating System (ROS) [67] is an open-source multi-language robot software framework that provides a vast collection of tools, libraries, and protocols for building, writing, and running code across multiple machines. The main resources of ROS are entitled as nodes, messages, topics, and services. Nodes are software units responsible for the computation of tasks, and they interact with each other by sending and receiving clusters of data named messages. The communication between nodes follows one of two protocols:

- Topics - publisher-subscriber protocol. A node publishes a message to a specific topic, being received by nodes subscribed to that same resource;
- Services - synchronous procedure calls defined by a pair of messages: a request and the respective reply.

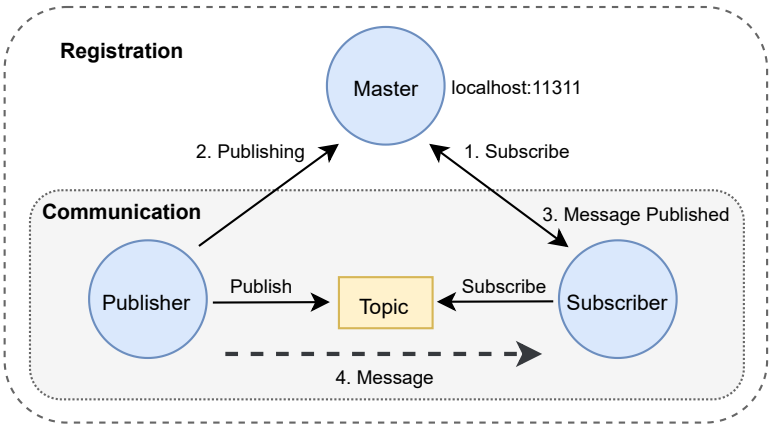


Figure 5.1: ROS publisher-subscriber communication protocol.

In this project, the protocol exploited was the publisher-subscriber. This routine, illustrated in Fig. 5.1, has a Master server that keeps track of the publishers and subscribers of each topic, topic addresses, services, and published messages. The registration and communication operations respect, in sequence, the subsequent steps:

1. Subscribers report to the Master that they wish to subscribe to a topic with a determined address;
2. Publisher informs the Master that it is publishing on that same topic;
3. Subscribers are notified by the Master;
4. Subscribers establish a connection with the publisher to receive the message.

ROS, running on Ubuntu 20.04, was the key framework connecting the various developed DeepRL navigation modules with the used robotics simulator. Its installation setup and an introductory navigation guideline to rapidly start exploring ROS to control a mobile robot are documented at https://github.com/DanielPalaio/ROS_Navigation_Guidelines.

5.2.1 ROS Resources

Figure 5.2: ROS topics subscribed and published by the proposed DeepRL-based system's node.

Table 5.1: ROS topics description.

Topic	Message Type	Description
/odom	nav_msgs/Odometry	Robot position and orientation (world coordinates)
/scan	sensor_msgs/LaserScan	Cluster of scanned points (world coordinates)
/cmd_vel	geometry_msgs/Twist	Linear and angular robot velocities
/costmap	nav_msgs/OccupancyGrid	Grid occupation data (int. array)
/map_metadata	nav_msgs/MapMetaData	Grid width and height
/tf2	tf2_msgs/TFMessage	Translation and rotation between frames

The ROS topics published and subscribed by the proposed DeepRL-based system's node are presented in Figure 5.2 and described in Table 5.1.

From the /scan topic, the agent collects the distance sensor findings to construct the environment metric representations C_{Stack}. Such readings, combined with the virtual robot's odometric data fetched from /odom, provide the necessary inputs to compute the target-related and obstacle-related state assets $p = \{d_T, d_O, \phi\}$. Throughout the DeepRL algorithm implementation, the data acquired from /scan was also converted into *nav_msgs/OccupancyGrid* ROS messages and published to a created /costmap topic to visualize and troubleshoot the making of the costmaps.

As described in Section 4.3.2, actions were defined as linear and angular speed pairs (v, w). To transmit such commands to the robot, they were arranged to match the ROS message type format *geometry_msgs/Twist* and further published to */cmd_vel*, the topic responsible for controlling the mobile platform's actuators.

5.3 Gazebo

Robotics simulation software constitutes a simple and economical alternative to validate complex systems, platforms, and prototypes [68]. With simulators, applications can be tested in controlled low-risk environments, avoiding incidents with real robots. Additionally, it is possible to evaluate the simulated experimental outcomes and adjust the necessary parameters to pursue better results.

Contemporary robotics simulators offer various physics engines, a vast library of robots, sensors, and actuators, advanced programmatic and graphical interfaces, and multiple plugins enabling robot locomotion and sensor readings' simulations [69]. After analyzing several options [70, 62, 71], Gazebo [72] was the robotics simulator selected to perform as a baseline to the developed navigation framework experiments. Gazebo is the most diffused software in the mobile, humanoid, and service robot research areas [69], and as portrayed in Table 3.2, it was used in numerous state-of-the-art DeepRL-based motion planning implementations.

5.3.1 Virtual Environments

The results presented in this dissertation were obtained from applying a mobile platform, controlled by the developed DeepRL-based motion planning strategy, into small-scale Gazebo virtual environments, with and without obstacles.

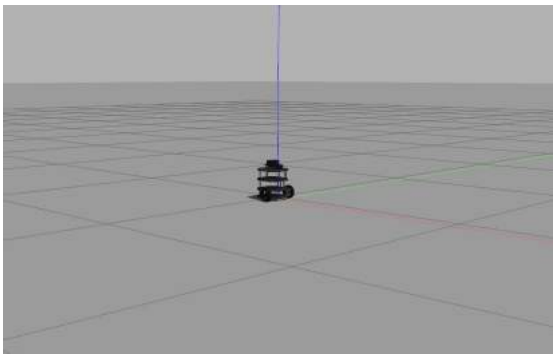
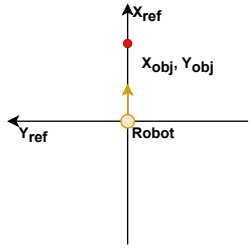


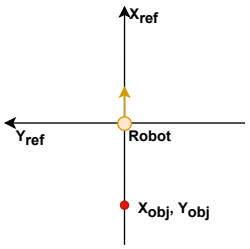
Figure 5.3: Gazebo empty environment.

Each environment contributed to the project by producing reciprocal results that commissioned a more embracing validation of the proposed DeepRL-based algorithm. The first and simpler virtual environment utilized to test the navigation system was the empty Gazebo domain displayed in Fig. 5.3, with the target configurations of Fig. 5.4.



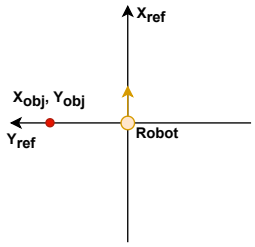
Initial Position: (0.0, 0.0)
Target: (0.8, 0.0)

(a) Move forward



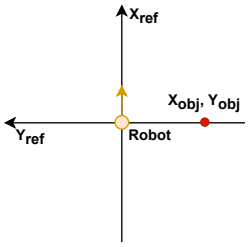
Initial Position: (0.0, 0.0)
Target: (-0.8, 0.0)

(b) Move backward



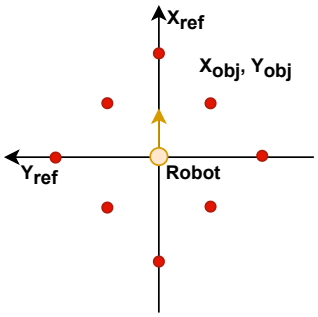
Initial Position: (0.0, 0.0)
Target: (0.0, 0.8)

(c) Turn left



Initial Position: (0.0, 0.0)
Target: (0.0, -0.8)

(d) Turn right

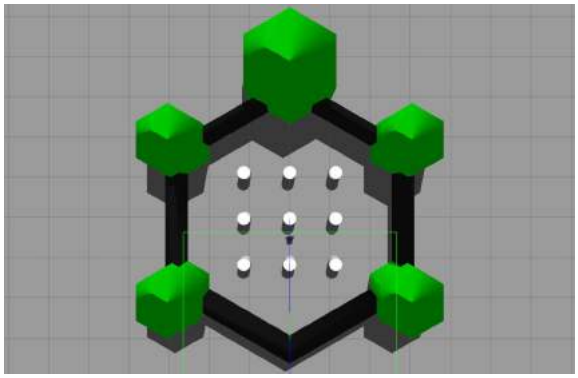


Initial Position (X,Y)	Target (X_{obj},Y_{obj})
(0.0, 0.0)	(0.8, 0.0)
	(0.5, 5.0)
	(0.0, 0.8)
	(-0.5, 0.5)
	(-0.8, 0.0)
	(-0.5, -0.5)
	(0.0, -0.8)
	(0.5, -0.5)

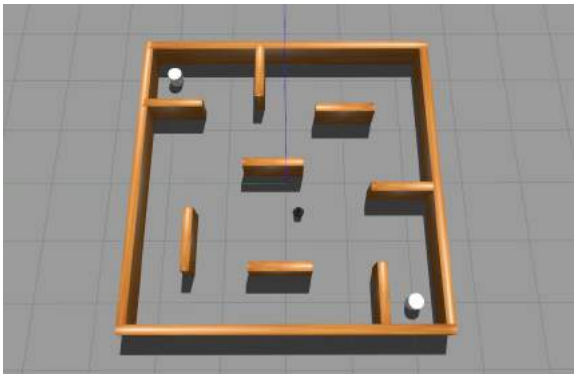
(e) Generalization scenario

Figure 5.4: Obstacle-free target configurations.

To try out the DeepRL-based system in environments with static obstacles, specific Gazebo domains - the Gazebo's *World* and *Stage 4* environments shown in Fig. 5.5 - were selected due to their obstacle arrangements. Within the same environments, the mobile robot was exposed to distinct scenarios in order to verify its capability to surpass the imposed challenges.



Gazebo's *World* environment



Gazebo's *Stage 4* environment

Figure 5.5: Gazebo's *World* and *Stage 4* environments.

5.4 RViz

Robots, in order to observe their surroundings, generally resort to sensors such as cameras or laser scanners. In computational science and computer graphics, a persistent requirement is representing the gathered sensory data to help the user understand any input information hidden insights [73]. RViz [73], an abbreviation of Robot Visualization, is a 3D visualization tool for the Robot Operating System framework that provides, among others, a view of robot models, sensor data, and map representations.

In this project, RViz proved to be a noteworthy support instrument, being used to visualize and troubleshoot the modulation of the collected virtual environment data into local costmaps (see Fig. 5.6).

Figure 5.6: Gazebo environments and respective Rviz costmap representations.

5.5 Turtlebot

To develop the proposed DeepRL-based algorithm, robot frameworks, environment visualization tools, and navigation simulators were, as described in this chapter thus far, indispensable materials to fulfill the outlined objectives.

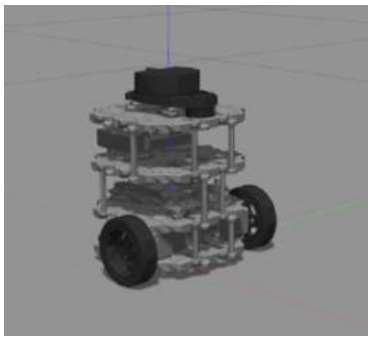


Figure 5.7: Turtlebot burger.

Nonetheless, the central element of the system that acquires and transmits all sorts of data to the software control modules, making the complete navigation pipeline work in consonance, is the mobile robot. Following the models used in the state-of-the-art covered applications, the platform utilized in this project was the standard Turtlebot burger depicted in Fig. 5.7.

Turtlebot is available in ROS turtlebot3 [74, 2], a package that enables users to promptly develop robotic applications without the need to construct robots or virtual environments. A pointer to the installation tutorial of the turtlebot3 package is accessible at https://github.com/DanielPalaio/ROS_Navigation_Guidelines.

The Turtlebot's main hardware specifications are presented in Table 5.2, and its onboard laser properties listed in Table 5.3.

Table 5.2: TurleBot hardware specifications [2].

Maximum translational velocity	0.22 m/s
Maximum rotational velocity	2.84 rad/s (162.72 deg/s)
Maximum payload	15kg
Size (L x W x H)	138mm x 178mm x 192mm
Laser Distance Sensor	360 Laser Distance Sensor LDS-01 [3]

Table 5.3: 360 Laser Distance Sensor LDS-01 [3].

Distance Range	120 - 3,500mm
Distance Accuracy (120mm - 499mm)	±15mm
Distance Accuracy (500mm - 3,500mm)	±5.0%
Distance Precision (120mm - 499mm)	±10mm
Distance Precision (500mm - 3,500mm)	±3.5%
Scan Rate	300±10 rpm
Angular Range	360°
Angular Resolution	1°

5.6 Python and Pycharm

The programming language elected to develop the software modules of this project was Python [75], version 3.8. Python is the preferred language for creating Machine Learning applications, having various tools and an extensive maintained package library to assist developers. The packages utilized in this work to support the implementation of the designed pipeline are presented in Table 5.4.

The IDE used to endorse the scripting was the Ubuntu-compatible PyCharm Community Edition 2020.3.3 [76]. PyCharm provides a wide range of features such as a built-in debugger, test runner, and terminal, all tightly integrated to create a productive programming environment.

Table 5.4: Python packages.

Package	Version	Description
Tensorflow-GPU [77]	2.4.1	Machine Learning library
Numpy [78]	1.19.5	Package for computation of arrays, matrices, and linear algebra
Pandas [79]	1.2.3	Data analysis and manipulation tool
Rospy [80]	1.3.0	Client library for ROS that enables Python to interface with topics, services, and parameters
Mathplotlib [81]	3.1.2	Library for creating static, animated, and interactive visualizations
Virtualenv	20.0.17	Tool that keeps project dependencies isolated from other projects and environments

5.6.1 TensorFlow

TensorFlow [63] is an open-source library for Machine Learning that offers the necessary tools to create, parameterize, train, and optimize Artificial Neural Networks. Among the available ML libraries for Python, Tensorflow was the one selected to use in this project due to its development support, information and tutorial accessibility, hardware compatibility, and prior utilization in some of the most distinguished DeepRL applications, such as the Deep Mind works addressed in Section 3.2.

Within Tensorflow, Keras [82], a built-in high-level and object-oriented API that provides user-friendly solutions to create and train network architectures, was also used to assemble the framework's ANNs.

5.7 NVIDIA GeForce GTX 1060

Training multilayer Artificial Neural Networks is a lengthy process that can rapidly escalate in conformity with the complexity of the networks and their input states, action space size, environment dimension, and task difficulty. However, this setback can be appeased by assigning the computation of the network updates to a GPU.

For ML libraries to run on a GPU, the graphics card must access the latest drivers and support CUDA [83] and its respective libraries. CUDA, short for Compute Unified Device Architecture, is an NVIDIA parallel computing framework that enables developers to utilize the full potential of the GPU's graphics processors, also termed CUDA cores. The NVIDIA CUDA Deep Neural Network library (cuDNN) [84], in sequence, is a GPU-accelerated library of primitives for Deep ANNs. cuDNN provides highly-tuned implementations for standard network routines, allowing calculations to be completed on the GPU.

Setting up the GPU can be an elaborated task, compensated nonetheless with the improved overall performance of the Deep model. The GPU utilized in this project was an NVIDIA GeForce GTX 1060 [4]. Its principal specifications are presented in Table 5.5.

Table 5.5: NVIDIA GeForce GTX 1060 specifications [4].

Card Length	210mm x 128mm
CUDA Cores	1280
Video Memory	6GB GDDR5
Memory Bus	192-bit
Engine Clock	Base:1556 MHz, Boost:1771 MHz
Memory Clock	8 GHz
Power Consumption	120W
Supported OS	Windows, Linux

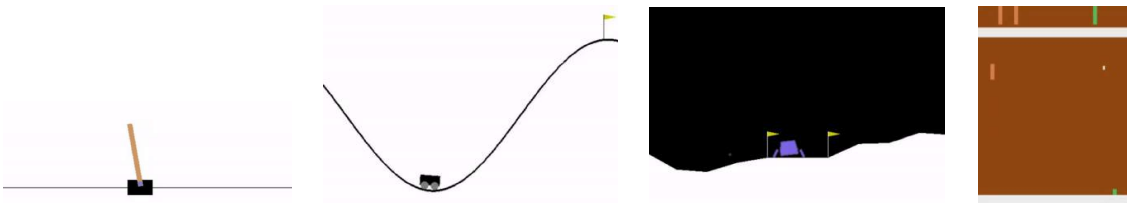
For both Windows and Linux operative systems, the step-by-step guide to install the latest GPU drivers, set up CUDA and cuDNN, and prove the workability of Tensorflow-GPU, is documented at https://github.com/DanielPalaio/CUDA_cuDNN_Setup.

5.8 OpenAI Gym

OpenAI Gym [85] is an arrangement of benchmark problems formulated in line with the episodic design of RL to validate Machine Learning methodologies. It provides a variety of configured agents and domains in a convenient software package, permitting developers to focus exclusively on algorithm development and testing. More than one hundred different benchmark environments are available in the toolkit package under the following categories [85]:

- Algorithmic, classic control, and toy text - small-scale tasks from RL literature;
- Atari - classic Atari games, with screen images or RAM as input;
- Box2D - continuous control tasks using the 2D physics engine Box2D [86];
- 2D and 3D robots - robot control using the MuJoCo physics engine [87].

In this project, the OpenAI Gym classic control *CartPole-v0* and *MountainCar-v0*, the Box2D *LunarLander-v2*, and the Atari *Pong-v4* benchmark environments, pictured in Fig. 5.8, were utilized to validate the implemented Deep Q-Learning algorithm and its dueling variations before their introduction in the more complex DeepRL-based navigation architecture.



(a) *CartPole-v0* (b) *MountainCar-v0* (c) *LunarLander-v2* (d) *Pong-v4*

Figure 5.8: OpenAI Gym benchmark environments.

An in-depth description of the environments' state, action, and reward models, episode termination conditions, solved requirements, employed ANN structure and hyperparameters, and validation evidences are presented in Section 6.1. These materials can be further complemented with the developed source code and video results accessible at:

- https://github.com/DanielPalaio/CartPole-v0_DeepRL
- https://github.com/DanielPalaio/MountainCar-v0_DeepRL
- https://github.com/DanielPalaio/LunarLander-v2_DeepRL
- https://github.com/DanielPalaio/Pong-v4_DeepRL

6

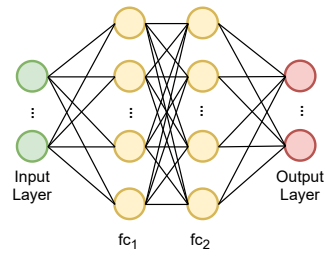
Results and Discussion

This chapter presents the results from testing the developed DeepRL-based motion planning strategy in virtual domains, with and without obstacles. The validation process of the implemented DQN frameworks in benchmark environments is also demonstrated and discussed.

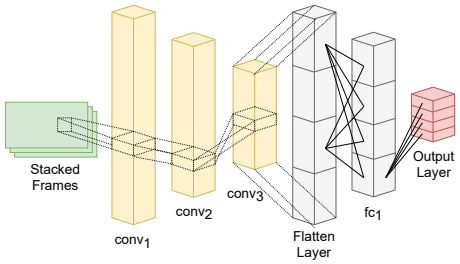
6.1 DQN Frameworks Validation

The first task performed within the project scope was the implementation and validation of the original Deep Q-Learning and respective dual variations (Dueling Deep Q-Learning and Dueling Double Deep Q-Learning) in the OpenAI Gym *CartPole-v0*, *MountainCar-v0*, *LunarLander-v2*, and *Pong-v4* benchmark environments. This stage was fundamental to facilitate later troubleshoots by reducing the extent of possible errors in the more complex pipeline.

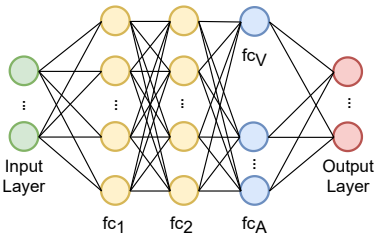
In these experiments, the mainstream ANN architectures shown in Fig. 6.1 were adapted to comply with the different environment's state, action, and reward models.



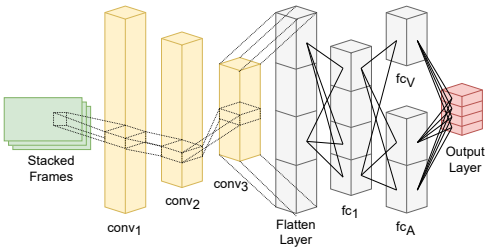
(a) Feedfoward Neural Network



(b) Convolutional Neural Network



(c) Dueling Feedfoward Neural Network



(d) Dueling Convolutional Neural Network

Figure 6.1: ANN configurations employed in the implemented DQN frameworks' validation process.

On an important note, the simulations presented in this *DQN Frameworks Validation* section had the sole purpose of validating the implemented DQN methodologies towards different system configurations. Further analysis to assess which framework demonstrates a superior training aptitude was performed in Section 6.2.1.

6.1.1 *CartPole-v0* OpenAI Gym Environment

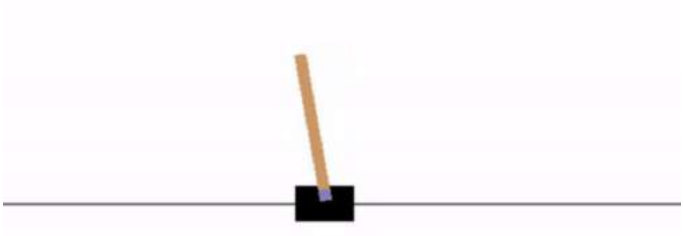


Figure 6.2: *CartPole-v0* OpenAI Gym environment.

CartPole-v0 (Fig. 6.2) is a virtual environment where the agent - a cart with a pole - learns which actions to take in order to maintain the pole in a vertical position during the entire episode length. According to the cart position, cart velocity, pole angle, and pole velocity at its top, the cart moves to the left or right, being rewarded for every step that the pole angle and the cart position do not exceed defined limits. The episode termination conditions and solved requirement are presented in Table 6.1.

Table 6.1: *CartPole-v0* episode termination and solved requirement.

Episode Termination	12° < Pole angle < -12° 2.4 < Cart position < -2.4 Episode length > 200 steps
Solved Requirement	Average reward of 195.0 over 100 consecutive trials

As shown in Fig. 6.3, every DQN architecture, prompted by the networks parameterized per Table 6.2 and the hyperparameters defined in Table 6.3, was able to surpass the solved requirement.

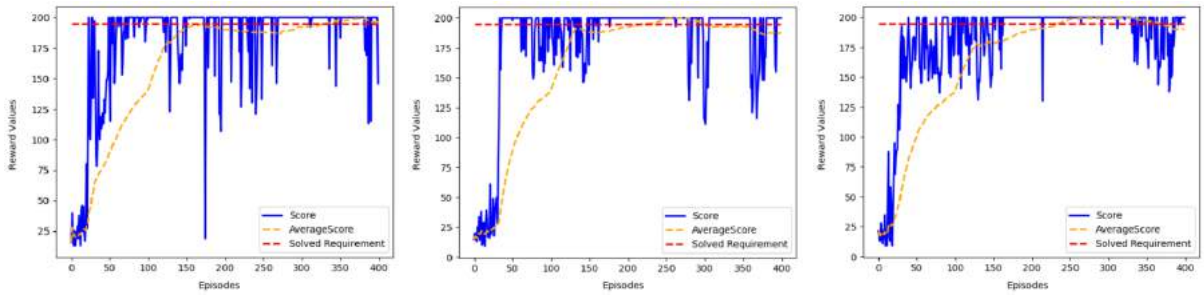
Table 6.2: Layer configuration (number of neurons and activation functions) of the ANNs used in the DQN-based frameworks' validation towards the OpenAI Gym *CartPole-v0* environment.

Framework	ANN	Input	fc1	fc2	fcV	fcA	Output
DQN	Fig. 6.1 (a)	4	256 'relu'	256 'relu'	-	-	2
Dueling DQN	Fig. 6.1 (c)	4	128 'relu'	128 'relu'	1	2	2
D3QN	Fig. 6.1 (c)	4	128 'relu'	128 'relu'	1	2	2

Table 6.3: *CartPole-v0* model hyperparameters.

Parameter	Value	Parameter	Value
Buffer Size	100000	Epsilon	1.0
Batch Size	64	Epsilon Decay (per step)	0.001
Discount Factor γ	0.99	Epsilon Final Value	0.01
TargetNet Update Rate (DQN)	100	Learning Rate lr (DQN)	0.001
TargetNet Update Rate (Dueling DQN)	120	Learning Rate lr (Dueling DQN)	0.00075
TargetNet Update Rate (D3QN)	120	Learning Rate lr (D3QN)	0.00075

The learning rate lr, first mentioned at this point, is a hyperparameter that controls how much to change the model in response to the estimated error each time its weights are updated [88].



(a) DQN (b) Dueling DQN (c) D3QN

Figure 6.3: *CartPole-v0* DQN-based framework's training scores (episode total rewards).

6.1.2 *MountainCar-v0* OpenAI Gym Environment

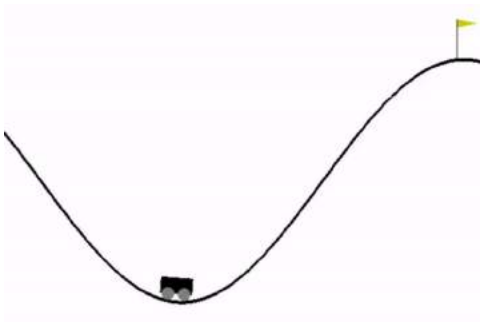


Figure 6.4: *MountainCar-v0* OpenAI Gym environment.

MountainCar-v0 (Fig. 6.4) is a benchmark environment where the agent - a four-wheel vehicle - has as objective climbing a mountain to reach its summit. However, the car does not have enough momentum to attain the mountain's top simply by moving in one direction from its resting position. To succeed, the agent must learn the right combination of actions (move left, right, or no action) having the car's position and velocity as the state representation. This learning process is guided by a reward model that penalizes the agent for each step taken, encouraging it to solve the problem in the least possible iterations. The episode termination conditions and solved requirement are presented in Table 6.4.

Table 6.4: *MountainCar-v0* episode termination and solved requirement.

Episode Termination	Car position == 0.5° Episode length > 200 steps
Solved Requirement	Average reward of -110.0 over 100 consecutive trials

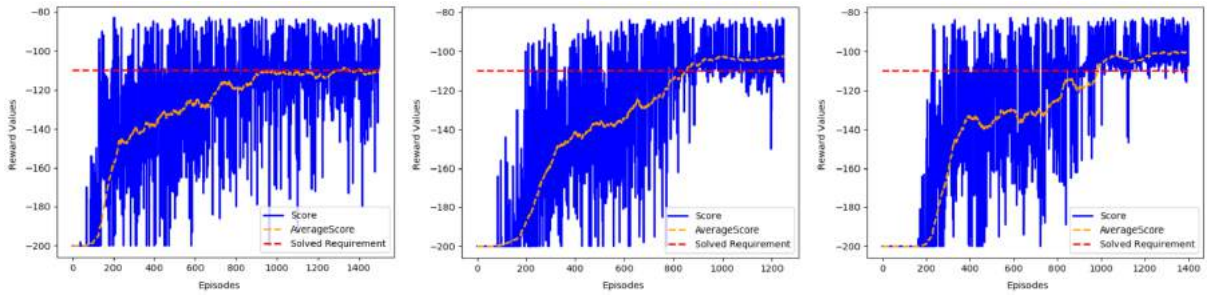
As demonstrated in Fig. 6.5, the implemented DQN, Dueling DQN, and D3QN algorithms, using the network structures and hyperparameters of Tables 6.5 and 6.6, respectively, managed to satisfy the environment's completion condition.

Table 6.5: Layer configuration (number of neurons and activation functions) of the ANNs used in the DQN-based frameworks' validation towards the OpenAI Gym *MountainCar-v0* environment.

Framework	ANN	Input	fc1	fc2	fcV	fcA	Output
DQN	Fig. 6.1 (a)	2	256 'relu'	256 'relu'	-	-	3
Dueling DQN	Fig. 6.1 (c)	2	128 'relu'	128 'relu'	1	3	3
D3QN	Fig. 6.1 (c)	2	128 'relu'	128 'relu'	1	3	3

Table 6.6: *MountainCar-v0* model hyperparameters.

Parameter	Value	Parameter	Value
Buffer Size	100000	Epsilon	1.0
Batch Size	64	Epsilon Decay (per step)	0.001
Discount Factor γ	0.99	Epsilon Final Value	0.01
TargetNet Update Rate (DQN)	100	Learning Rate lr (DQN)	0.001
TargetNet Update Rate (Dueling DQN)	120	Learning Rate lr (Dueling DQN)	0.00075
TargetNet Update Rate (D3QN)	150	Learning Rate lr (D3QN)	0.001



(a) DQN (b) Dueling DQN (c) D3QN

Figure 6.5: *MountainCar-v0* DQN-based framework's training scores (episode total rewards).

6.1.3 *LunarLander-v2* OpenAI Gym Environment

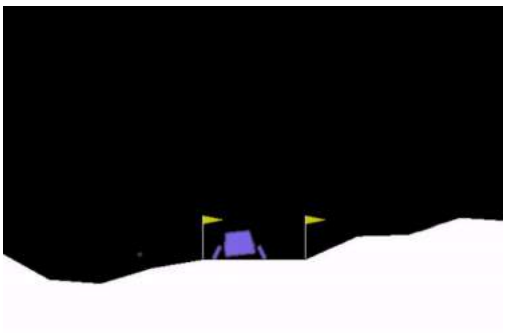


Figure 6.6: *LunarLander-v2* OpenAI Gym environment.

LunarLander-v2 (Fig. 6.6) is a substantially more complex environment compared to *CartPole-v0* and *MountainCar-v0*. In *LunarLander-v2*, the agent - a bipedal spaceship - must execute a controlled landing in a marked area by actuating its right, left, and main engines based on the state model's lander position, angle, speed, and touchdown status.

To harmlessly land in the delimited pad, a learning proceeding is required to adjust the agent's behavior. This training process is led by an intricate model that rewards the lander according to the state transitions, episode termination condition (Table 6.7), and action selection, as follows:

- Moving towards the landing pad gives a scalar reward between 100 and 140;
- Moving away from the landing pad gives a scalar reward between -140 and -100;
- If the lander crashes, a scalar reward of -100 is given;
- If the lander comes to rest, a scalar reward of 100 is given;
- Each leg with ground contact corresponds to a scalar reward of 10;
- Firing the main engine corresponds to a scalar reward of -0.3 per frame;
- Firing the side engines corresponds to a scalar reward of -0.3 per frame.

Table 6.7: *LunarLander-v2* episode termination and solved requirement.

Episode Termination	Lander crashes	Lander comes to rest	Episode length > 400 steps
Solved Requirement	Average reward of 200.0 over 100 consecutive trials		

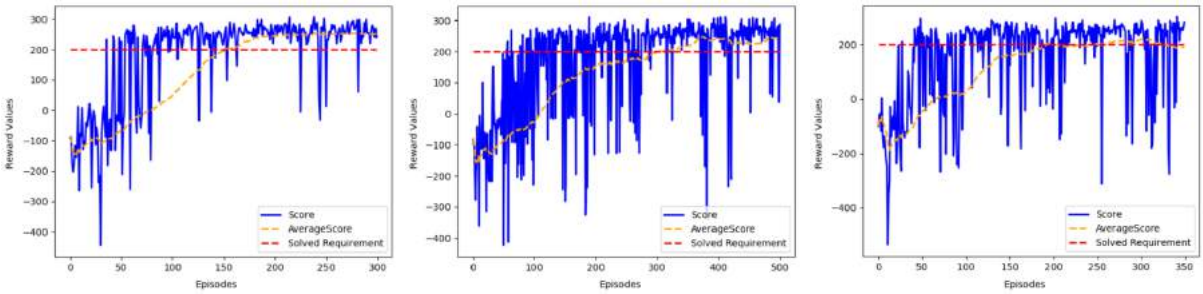
Empowered by the network architectures and the hyperparameters defined in Tables 6.8 and 6.9, respectively, the intelligent agent, as shown in Fig 6.7, was able to wrap up *LunarLander-v2*, producing greater scores than the environment's solved requirement.

Table 6.8: Layer configuration (number of neurons and activation functions) of the ANNs used in the DQN-based frameworks' validation towards the OpenAI Gym *LunarLander-v2* environment.

Framework	ANN	Input	fc1	fc2	fcV	fcA	Output
DQN	Fig. 6.1 (a)	8	256 'relu'	256 'relu'	-	-	4
Dueling DQN	Fig. 6.1 (c)	8	128 'relu'	128 'relu'	1	4	4
D3QN	Fig. 6.1 (c)	8	128 'relu'	128 'relu'	1	4	4

Table 6.9: *LunarLander-v2* model hyperparameters.

Parameter	Value	Parameter	Value
Buffer Size	100000	Epsilon	1.0
Batch Size	64	Epsilon Decay (per step)	0.001
Discount Factor γ	0.99	Epsilon Final Value	0.01
TargetNet Update Rate	120	Learning Rate	0.00075



(a) DQN (b) Dueling DQN (c) D3QN

Figure 6.7: *LunarLander-v2* DQN-based framework's training scores (episode total rewards).

6.1.4 *Pong-v4* OpenAI Gym Environment

Figure 6.8: *Pong-v4* OpenAI Gym environment.

The OpenAI Gym *Pong-v4* (Fig. 6.8), based on the renowned Atari Pong game, was the last benchmark environment in which the implemented DQN frameworks underwent a validation process. In *Pong-v4*, the agent - the green racket displayed in Fig. 6.8 - is intended to learn via trial-and-error which actions to adopt in order to beat the opposing orange racket.

The state model of this particular OpenAI Gym problem is composed of a stack of 4 (80, 80) cropped grey-scaled images, representations of the four latest game iterations. According to these observations, and having at its disposal six distinct actions (move down, up, or remain in the same position - 2 actions per event), the agent has to perfect its action selection to outclass the opposing racket. In *Pong-v4*, rewards are originated in agreement with each rally's winner.

Using a CNN structured per Table 6.12 and the hyperparameter combination of Table 6.11, the DQN frameworks managed perfect performances in the game of Pong (21-0 scores), thus concluding successfully a vast validation process in the *CartPole-v0*, *MountainCar-v0*, *LunarLander-v2*, and *Pong-v4* OpenAI Gym benchmark environments.

Table 6.10: *Pong-v4* episode termination and solved requirement.

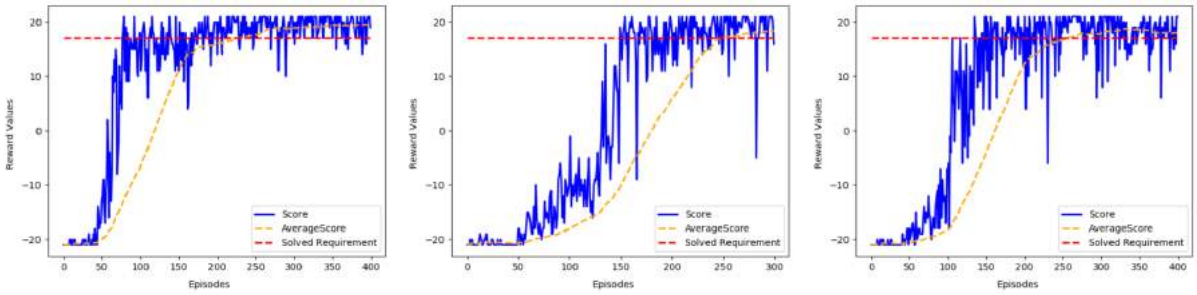
Episode Termination	Total score of 21 is reached	Episode length > 400000
Solved Requirement	Average score of 17 over 100 consecutive trials	

Table 6.11: *Pong-v4* model hyperparameters.

Parameter	Value	Parameter	Value
Buffer Size	100000	Epsilon	1.0
Batch Size	64	Epsilon Decay (per step)	0.001
Discount Factor γ	0.99	Epsilon Final Value	0.01
TargetNet Update Rate	1000	Learning Rate lr	0.0001

Table 6.12: Layer configuration (number of neurons, filters, kernel size, strides, and activation functions) of the ANNs used in the DQN-based frameworks' validation towards the OpenAI Gym *Pong-v4* environment.

Framework	ANN	Input	conv1	conv2	conv3	fc1	fcV	fcA	Output
DQN	Fig. 6.1 (b)	(4,80,80)	filters=32 kernel size=8 strides=4 'relu'	filters=64 kernel size=4 strides=2 'relu'	filters=64 kernel size=3 strides=1 'relu'	512	-	-	6
Dueling DQN	Fig. 6.1 (d)	(4,80,80)	filters=32 kernel size=8 strides=4 'relu'	filters=64 kernel size=4 strides=2 'relu'	filters=64 kernel size=3 strides=1 'relu'	512	1	6	6
D3QN	Fig. 6.1 (d)	(4,80,80)	filters=32 kernel size=8 strides=4 'relu'	filters=64 kernel size=4 strides=2 'relu'	filters=64 kernel size=3 strides=1 'relu'	512	1	6	6



(a) DQN (b) Dueling DQN (c) D3QN

Figure 6.9: *Pong-v4* DQN-based framework's training performances (episode total rewards).

6.2 Motion Planning in Obstacle-free Environments

As pointed out throughout this dissertation, the proposed local motion planning approach differs from the conventional DeepRL-based navigation methodologies. Being an original strategy with no working guarantees from the outset, a thorough experimentation phase was conducted in a low-scale obstacle-free environment (Fig. 6.10) to validate the developed system.

This section also delivers evidence of which DQN framework demonstrated a superior training competence to further lead this routine in the more complex environments with obstacles.

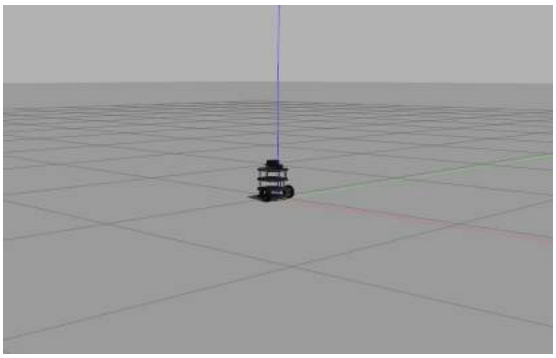
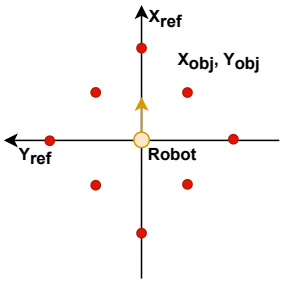


Figure 6.10: Gazebo obstacle-free environment.

6.2.1 DQN, Dueling DQN, and D3QN Comparison

The primary executed task on Gazebo's plain-type environment was the deduction of a Deep Q-Network capable of handling the reduced state model $\mathbb{S} = \{C_{Stack}, d_T, \phi\}$ and accurately computing the output *Qvalues*. This probing process is reported in Appendix B, and the resultant obstacle-oriented architectures are addressed in Section 4.2.

After the network inference, the complete DeepRL-based motion planning navigation system was tested in the scenario illustrated in Fig. 6.11 to validate, evaluate, and compare the training performances of the DQN, Dueling DQN, and D3QN learning methodologies. This operation was executed equitably, being each framework embodied with the same networks (pictured in Fig. 6.12), hyperparameters, and simulation variables (Table 6.13).

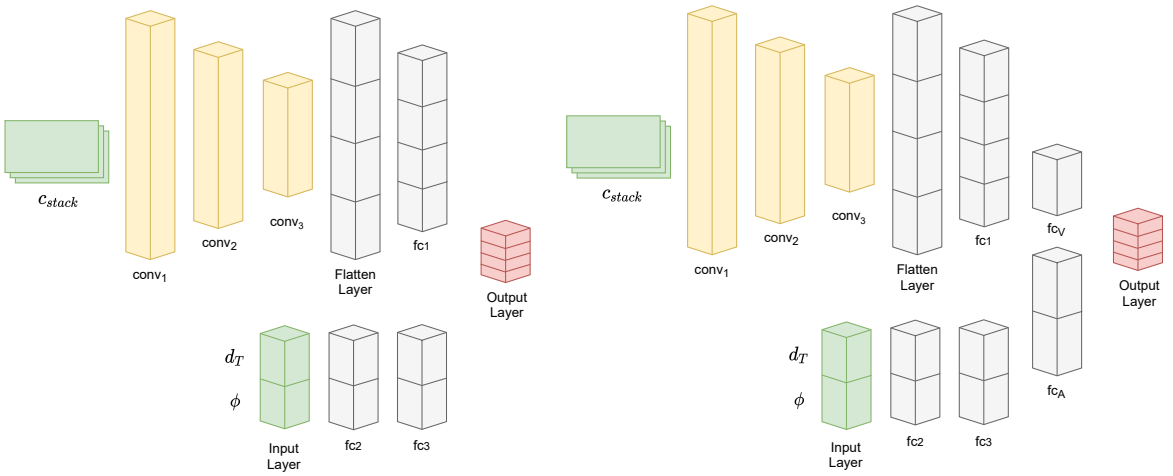


Initial Position (X,Y)	Targets (X_{obj},Y_{obj})
(0.0, 0.0)	(0.8, 0.0)
	(0.5, 5.0)
	(0.0, 0.8)
	(-0.5, 0.5)
	(-0.8, 0.0)
	(-0.5, -0.5)
	(0.0, -0.8)
	(0.5, -0.5)

Figure 6.11: Fig. 5.4 (e) obstacle-free training and testing scenario.

The Fig. 6.11 scenario has a total of eight different targets. Each one was selected with a periodicity of eight episodes. As a result, the replay buffer was constantly filled with heterogeneous data, stimulating the agent in the training phase to attain every target rather than focusing only on a singular goal.

The networks used on this obstacle-free validation stage, presented in Fig. 6.12, are adaptations from the architectures considered in Section 4.2. Apart from the state inputs (input layer parameterization - reduced state model), the remaining structure stayed authentic to the Table 4.1 layer configuration.



(a) Deep Q-Network (b) Dueling Deep Q-Network.

Figure 6.12: Network architectures employed in the proposed DeepRL-based motion planning framework's validation process.

Table 6.13: Simulation variables and framework hyperparameters utilized to validate the Deep Q-Learning, Dueling Deep Q-Learning, and Dueling Double Deep Q-Learning algorithms in obstacle-free environments.

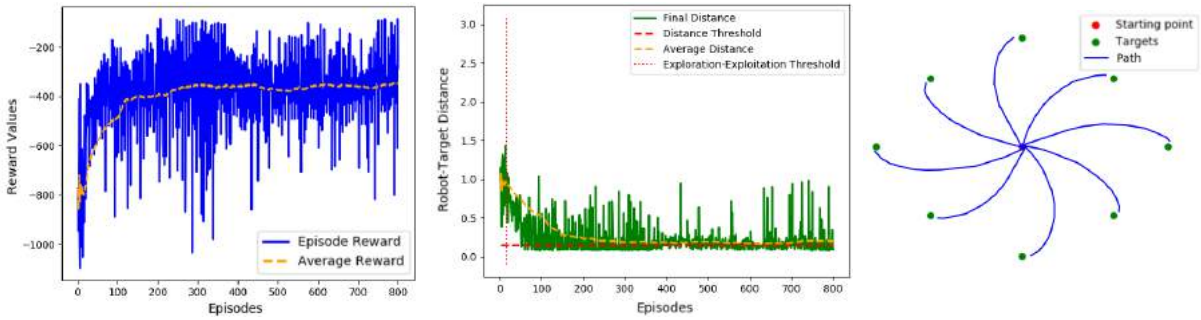
Parameter	Value	Parameter	Value
Number of Episodes	800	Number of Steps	80
Buffer Size	64000	Learning Rate lr	0.001
Batch Size	256	Discount Factor γ	0.99
Epsilon	1.0	Epsilon Decay (per step)	0.001
Epsilon Final Value	0.01	TargetNet Update Rate	40

Besides the employed state model $\mathbb{S} = \{C_{Stack}, d_T, \phi\}$, different Reward Sets from Table 4.3 were utilized to analyze each framework's learning aptitude given different d_T and ϕ associated weights. The action model used was the Set 1 of Table 4.2, an elementary arrangement composed of three commands: move forward, turn left, and turn right (turns with angular speed only). In compliance with these engaged state, action, and reward models, a post-training robot motion, to be considered optimal, must reveal, in order, the following behavioral pattern:

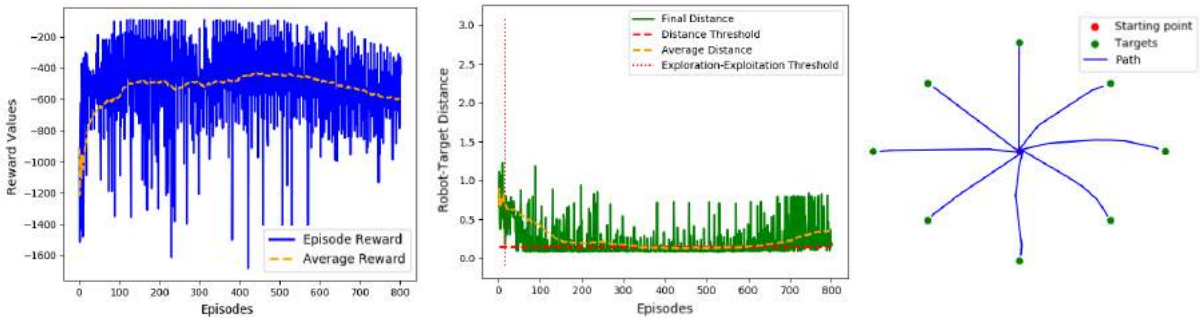
1. Adjust the orientation (minimize ϕ) to the target with a pure rotation over its center point;
2. Advance with a linear motion (forward action) towards the target (minimize d_T).

The following Figures 6.13, 6.14, and 6.15 present relevant training and post-training data that corroborate the developed DeepRL-based local motion planning strategy operability on empty environments, using every implemented DQN framework.

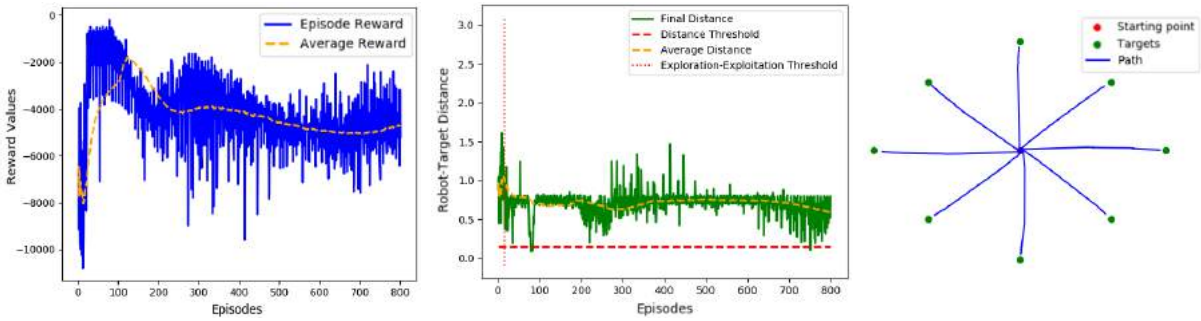
Deep Q-Learning



(a) Artificial Neural Network: Fig. 6.12 (a) w/ Table 4.1 Parameterization $\mathbb{S} = \{C_{Stack}, d_T, \phi\}$; Action Set 1; Reward Set 2.1 {k_D: 5.0, k_ϕ: 0.05}



(b) Artificial Neural Network: Fig. 6.12 (a) w/ Table 4.1 Parameterization $\mathbb{S} = \{C_{Stack}, d_T, \phi\}$; Action Set 1; Reward Set 2.2 {k_D: 5.0, k_ϕ: 0.1}



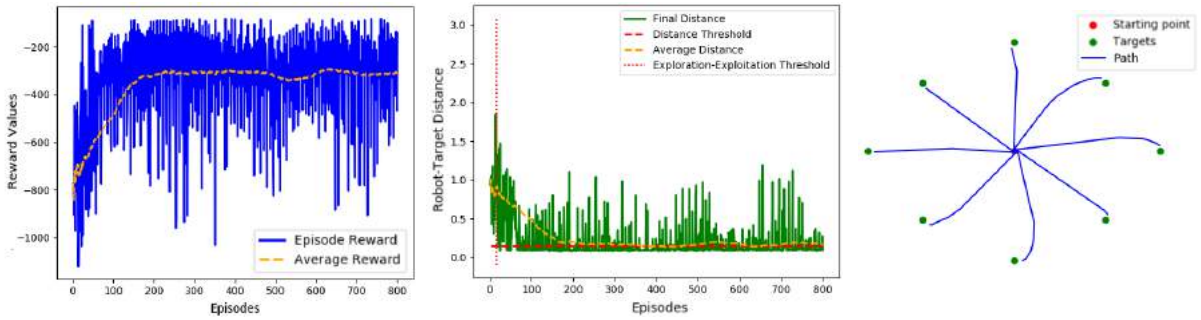
(c) Artificial Neural Network: Fig. 6.12 (a) w/ Table 4.1 Parameterization $\mathbb{S} = \{C_{Stack}, d_T, \phi\}$; Action Set 1; Reward Set 2.3 {k_D: 5.0, k_ϕ: 0.8}

Figure 6.13: Training scores and final episode distances, and robot post-training paths resultant from employing the developed DeepRL-based local motion planning strategy, with the Deep Q-Learning algorithm, over the obstacle-free scenario 6.11. Action and Reward Sets from Tables 4.2 and 4.3, respectively.

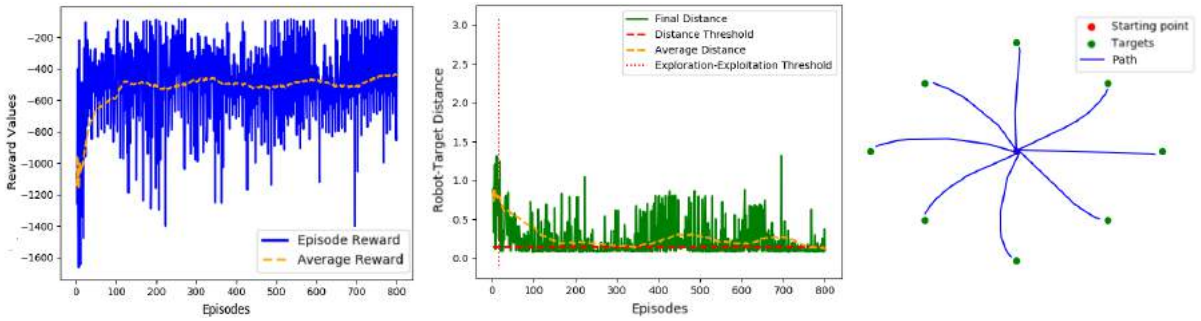
Table 6.14: Deep Q-Learning (Fig. 6.13) training details.

Training/Testing	Training Duration	Saved Models	Test Model	Test Model's Training Target
Fig. 6.13 (a)	$\geq$ 4 h	$\geq$ 500	116/800	(0.5, 0.5)
Fig. 6.13 (b)	$\geq$ 4 h	$\geq$ 500	100/800	(0.5, 0.5)
Fig. 6.13 (c)	$\geq$ 4 h	$\leq$ 5	81/800	(0.0, -0.8)

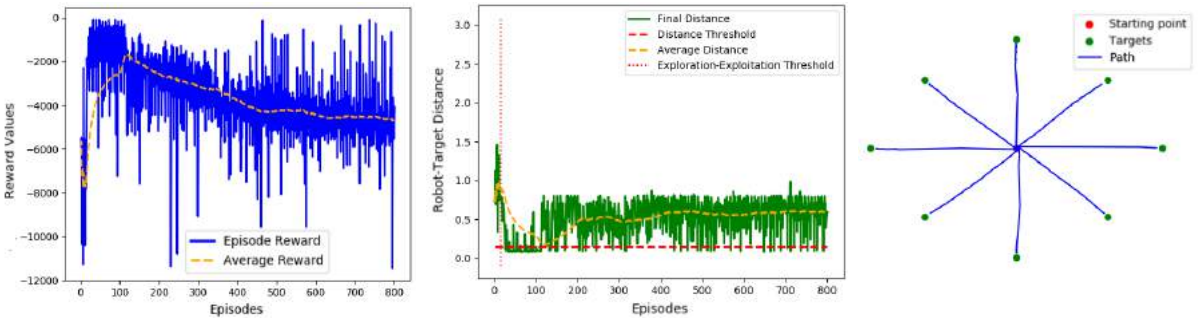
Dueling Deep Q-Learning



(a) Artificial Neural Network: Fig. 6.12 (b) w/ Table 4.1 Parameterization $\mathbb{S} = \{C_{Stack}, d_T, \phi\}$; Action Set 1; Reward Set 2.1 {k_D: 5.0, k_ϕ: 0.05}



(b) Artificial Neural Network: Fig. 6.12 (b) w/ Table 4.1 Parameterization $\mathbb{S} = \{C_{Stack}, d_T, \phi\}$; Action Set 1; Reward Set 2.2 {k_D: 5.0, k_ϕ: 0.1}



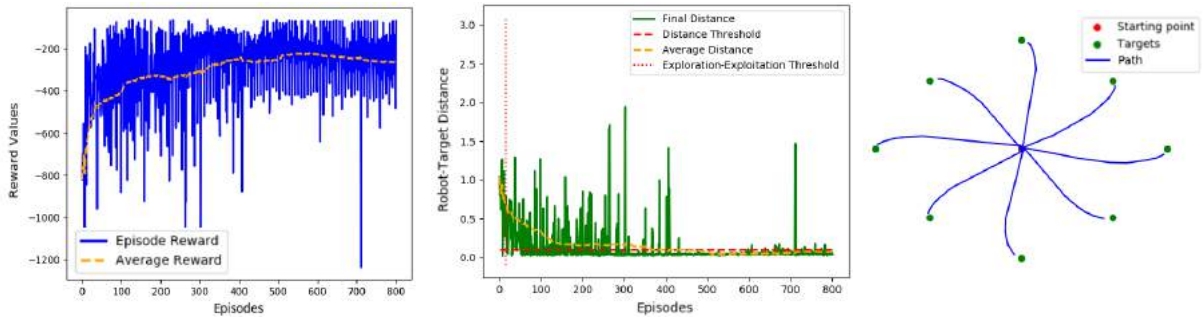
(c) Artificial Neural Network: Fig. 6.12 (b) w/ Table 4.1 Parameterization $\mathbb{S} = \{C_{Stack}, d_T, \phi\}$; Action Set 1; Reward Set 2.3 {k_D: 5.0, k_ϕ: 0.8}

Figure 6.14: Training scores and final episode distances, and robot post-training paths resultant from employing the developed DeepRL-based local motion planning strategy, with the Dueling Deep Q-Learning algorithm, over the obstacle-free scenario 6.11. Action and Reward Sets from Tables 4.2 and 4.3, respectively.

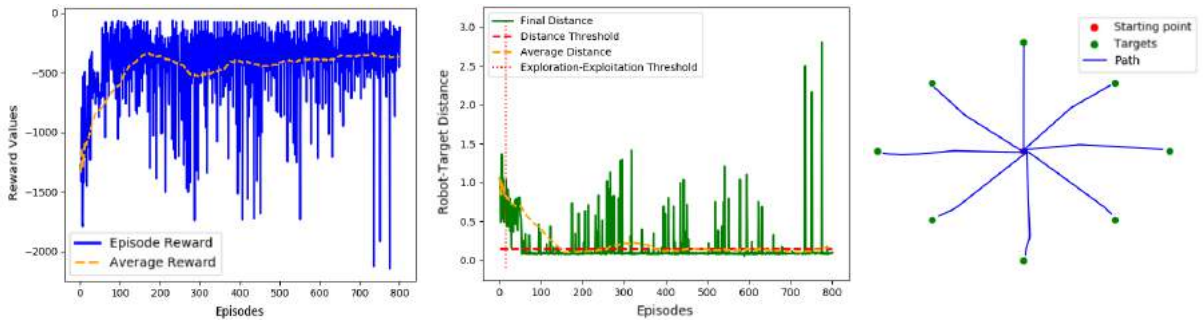
Table 6.15: Dueling Deep Q-Learning (Fig. 6.14) training details.

Training/Testing	Training Duration	Saved Models	Test Model	Test Model's Training Target
Fig. 6.14 (a)	$\geq$ 4 h	$\geq$ 500	128/800	(0.8, 0.0)
Fig. 6.14 (b)	$\geq$ 4 h	$\geq$ 350	75/800	(-0.5, -0.5)
Fig. 6.14 (c)	$\geq$ 4 h	$\leq$ 100	39/800	(-0.8, 0.0)

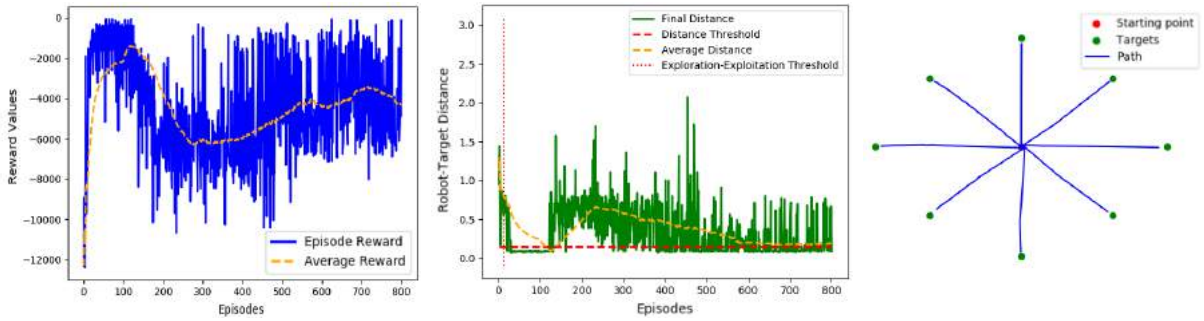
Dueling Double Deep Q-Learning



(a) Artificial Neural Network: Fig. 6.12 (b) w/ Table 4.1 Parameterization $\mathbb{S} = \{C_{Stack}, d_T, \phi\}$; Action Set 1; Reward Set 2.1 {k_D: 5.0, k_ϕ: 0.05}



(b) Artificial Neural Network: Fig. 6.12 (b) w/ Table 4.1 Parameterization $\mathbb{S} = \{C_{Stack}, d_T, \phi\}$; Action Set 1; Reward Set 2.2 {k_D: 5.0, k_ϕ: 0.1}



(c) Artificial Neural Network: Fig. 6.12 (b) w/ Table 4.1 Parameterization $\mathbb{S} = \{C_{Stack}, d_T, \phi\}$; Action Set 1; Reward Set 2.3 {k_D: 5.0, k_ϕ: 0.8}

Figure 6.15: Training scores and final episode distances, and robot post-training paths resultant from employing the developed DeepRL-based local motion planning strategy, with the Dueling Double Deep Q-Learning algorithm, over the obstacle-free scenario 6.11. Action and Reward Sets from Tables 4.2 and 4.3, respectively.

Table 6.16: Dueling Double Deep Q-Learning (Fig. 6.15) training details.

Training/Testing	Training Duration	Saved Models	Test Model	Test Model's Training Target
Fig. 6.15 (a)	$\geq$ 4 h	$\geq$ 500	404/800	(0.5, -0.5)
Fig. 6.15 (b)	$\geq$ 4 h	$\geq$ 500	78/800	(-0.5, 0.5)
Fig. 6.15 (c)	$\geq$ 4 h	$\geq$ 300	34/800	(0.0, -0.8)

As shown in Figures 6.13, 6.14, and 6.15, each implemented DQN learning algorithm generated no less than one fine-tuned model that conferred the robot the ability to attain, in a post-training extent, every Fig. 6.11 scenario targets.

As expected, based on the concepts covered in Section 2.4, the original Deep Q-Learning method was the learning technique that delivered the least favorable training performances (Fig. 6.13). On the other hand, accrediting once more the theoretical analysis covered in *Background Material*, the state-of-the-art D3QN proved to be the superior DQN-based training methodology. This last statement is mainly supported by the results of Figures 6.13 (c), 6.14 (c), and 6.15 (c), evidence obtained from trials carried out with the Reward Set 2.3. Only the D3QN framework, fitted with this sharp orientation-regulated reward set combination, managed to converge towards high episode rewards and maintain such behavior long-term (Fig. 6.15).

From this point onward, advocated by the presented outcomes and theoretical standards, every application of the proposed DeepRL-based navigation approach had as learning strategy the D3QN algorithm.

6.2.2 Non-Pure-Rotational Turnings

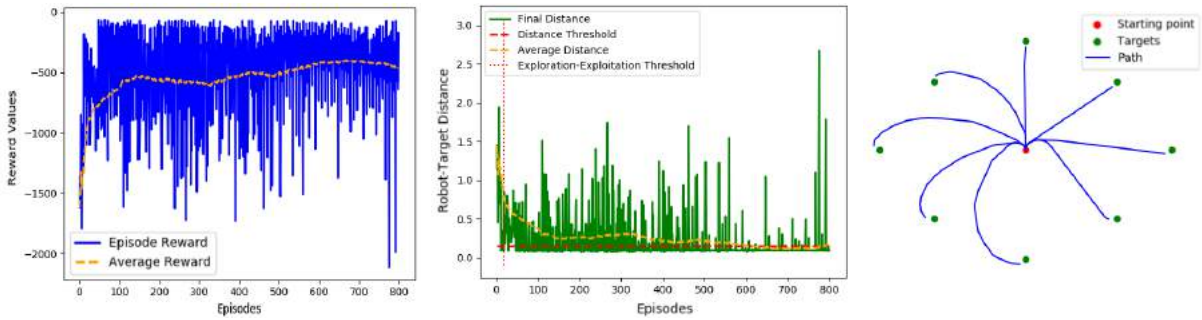
The previous *DQN, Dueling DQN, and D3QN Comparison* (Section 6.2.1) experiments were setted up having an action space composed of three commands: move forward, turn left, and turn right. Those turns were arranged with angular speed only, giving the virtual robot the feature of changing its orientation by rotating on itself.

To witness different robot behaviors from the ones observed, for example, in Fig 6.15, the proposed DeepRL-based motion planning approach was tested with the Action Set 2 of Table 4.2, a model composed in part of non-pure-rotational turnings (linear-angular speed pairs). The obstacle-free training and testing scenarios (Fig. 6.11), simulation variables, and framework's hyperparameters (Table 6.13) utilized in the prior section trials remained unchanged.

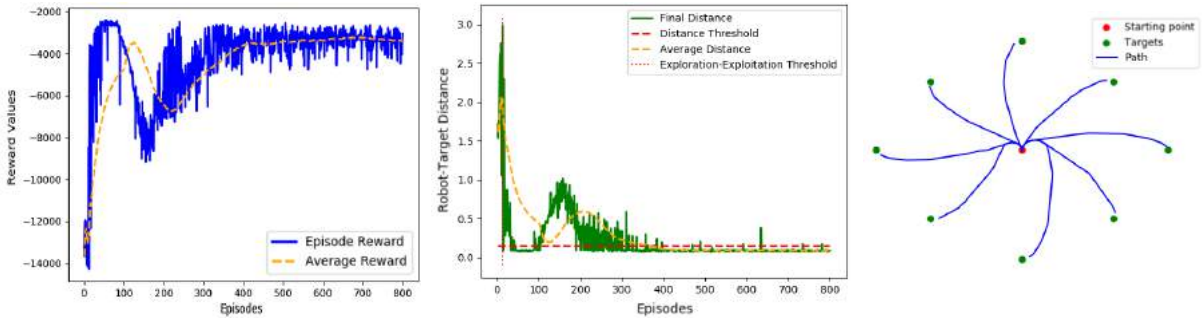
The results of this experiment are presented in Fig. 6.16. It is noticeable that the robot to attain some of the targets had to perform a wide turn, a reasonable behavior according to the actions available, but that in these types of empty domains is not entirely fitting. Despite the flawless learning process and the acceptable robot's post-training movement, subsequent employments of the developed DeepRL-based navigation approach on obstacle-free environments were conducted with the Action Set 1 instead, due to the robot's straightforward motion revealed with this model in Section 6.2.1.

Table 6.17: Fig. 6.16 training details.

Training/Testing	Training Duration	Saved Models	Test Model	Test Model's Training Target
Fig. 6.16 (a)	$\geq$ 4 h	$\geq$ 500	87/800	(-0.8, 0.0)
Fig. 6.16 (b)	$\geq$ 4 h	$\geq$ 500	773/800	(-0.8, 0.0)



(a) Artificial Neural Network: Fig. 6.12 (b) w/ Table 4.1 Parameterization $\mathbb{S} = \{C_{Stack}, d_T, \phi\}$; Action Set 2; Reward Set 2.2 $\{k_D$: 5.0, k_ϕ: 0.1$\}$



(b) Artificial Neural Network: Fig. 6.12 (b) w/ Table 4.1 Parameterization $\mathbb{S} = \{C_{Stack}, d_T, \phi\}$; Action Set 2; Reward Set 2.3 $\{k_D$: 5.0, k_ϕ: 0.8$\}$

Figure 6.16: Training scores and final episode distances, and robot post-training paths resultant from employing the developed DeepRL-based local motion planning strategy with the Action Set 2 (Table 4.2), over the obstacle-free scenario 6.11. Reward Sets from Table 4.3.

6.2.3 Generalization

As described in Sections 2.4.2 and 2.4.3, in DQN frameworks the agent's behavior is adjusted through random transition tuples of past experiences in each training step. Consequently, its decision-making proficiency is progressively shaped based upon the entire training scope. As shown in previous experiments and respective results, using multiple targets in training gave the intelligent agent the capability to reach every defined objective under the control of a fine-tuned network model.

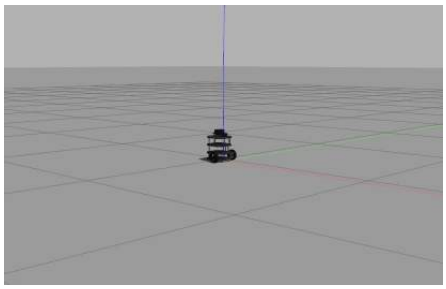
To verify the authentic generalization aptitude of the developed DeepRL-based agent, trials with different training and testing scenarios were executed. In the training phase, the mobile platform was stimulated to attain a single target. On the online stage, guided by a saved network model, the robot was subjected to a more complex scenario to reach multiple objective points.

The following subsections present the results of three individual generalization tests: training towards left, right, and backside located targets, and testing over the Fig. 6.11 scenario composed of eight goals, seven of them never acknowledged by the agent in the training routine. All experiments shared the same system configurations disclosed in Table 6.18.

Table 6.18: Simulation variables, DQN framework, Artificial Neural Network architecture and parametrization, state, action and reward models, and system hyperparameters utilized in the generalization trials.

Parameter	Value	Parameter	Value
DQN framework	D3QN	ANN	Fig 6.12 (b)
State Model	$\{C_{Stack}, d_T, \phi\}$	ANN Parametrization	Table 4.1
Action Set (Table 4.2)	Set 1	Reward Set (Table 4.3)	Set 2.2
Number of Episodes	200	Number of Steps	80
Buffer Size	16000	Learning Rate lr	0.001
Batch Size	256	Discount Factor γ	0.99
Epsilon	1.0	Epsilon Decay (per step)	0.001
Epsilon Final Value	0.01	TargetNet Update Rate	40

6.2.3.1 Training Towards a Target Located to the Left of the Agent



Gazebo obstacle-free environment

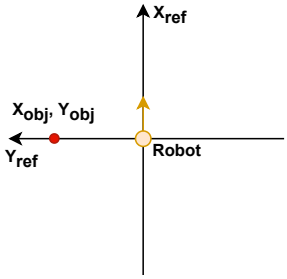


Fig. 5.4 (c)
Training scenario

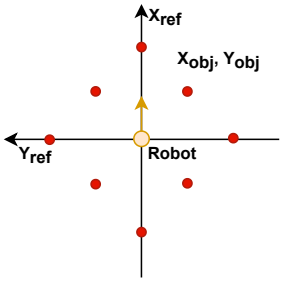


Fig. 5.4 (e)
Testing scenario

Figure 6.17: Fig. 5.4 (c) and Fig. 5.4 (e) obstacle-free training and testing scenarios.

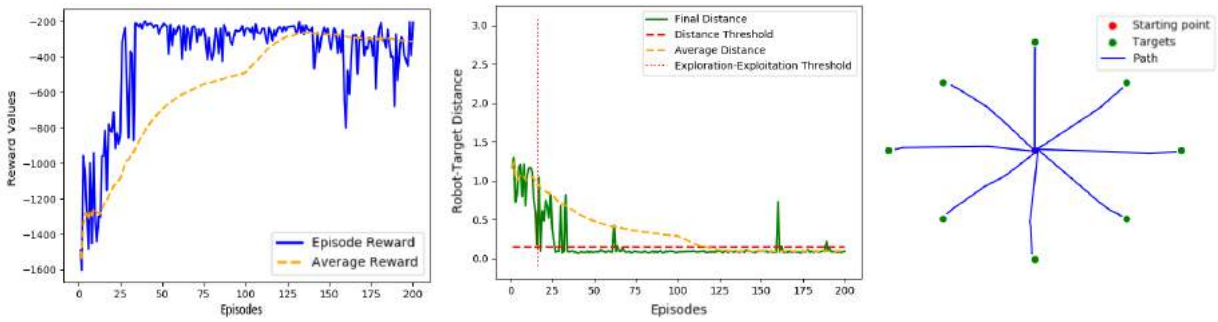
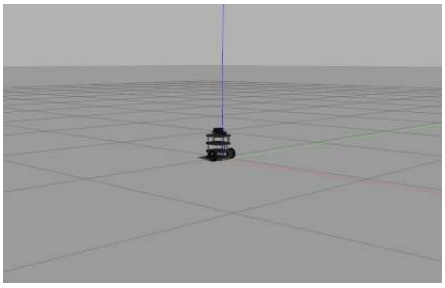


Figure 6.18: Training scores and final episode distances, and robot post-training paths resultant from training the agent in the Fig. 5.4 (c) scenario. Online phase executed in the Fig. 5.4 (e) arrangement.

6.2.3.2 Training Towards a Target Located to the Right of the Agent



Gazebo obstacle-free environment

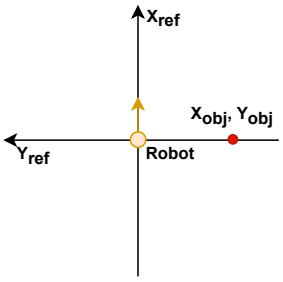


Fig. 5.4 (d)
Training scenario

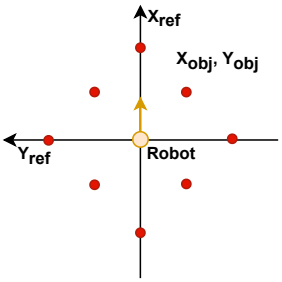
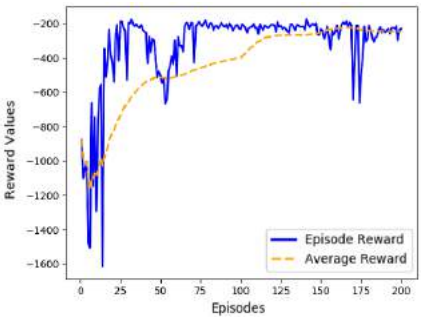
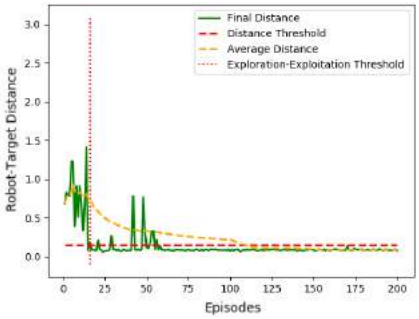


Fig. 5.4 (e)
Testing scenario

Figure 6.19: Fig. 5.4 (d) and Fig. 5.4 (e) obstacle-free training and testing scenarios.





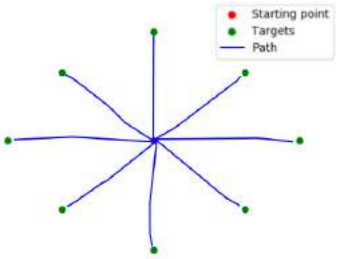
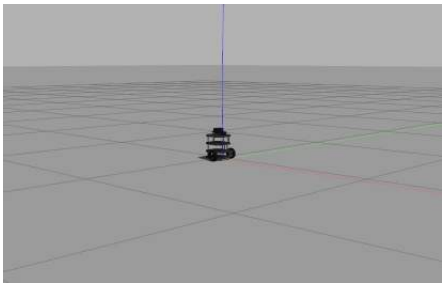


Figure 6.20: Training scores and final episode distances, and robot post-training paths resultant from training the agent in the Fig. 5.4 (d) scenario. Online phase executed in the Fig. 5.4 (e) arrangement.

6.2.3.3 Training Towards a Target Located Behind the Agent



Gazebo obstacle-free environment

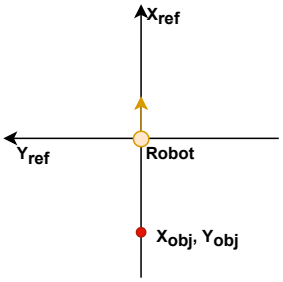


Fig. 5.4 (b)
Training scenario

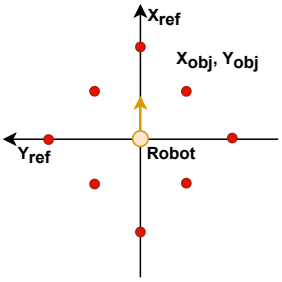


Fig. 5.4 (e)
Testing scenario

Figure 6.21: Fig. 5.4 (b) and Fig. 5.4 (e) obstacle-free training and testing scenarios.

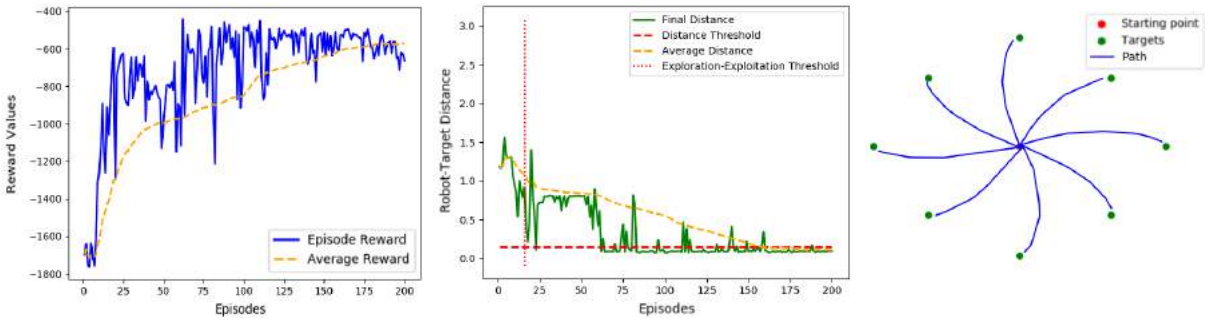


Figure 6.22: Training scores and final episode distances, and robot post-training paths resultant from training the agent in the Fig. 5.4 (b) scenario. Online phase executed in the Fig. 5.4 (e) arrangement.

Table 6.19: Generalization experiments (Figures 6.18, 6.20, 6.22) training details.

Training/Testing	Training Duration	Saved Models	Test Model	Test Model's Training Target
Fig. 6.18	$\geq$ 1 h	$\geq$ 150	37/200	(0.0, 0.8)
Fig. 6.20	$\geq$ 1 h	$\geq$ 150	26/200	(0.0, -0.8)
Fig. 6.22	$\geq$ 1 h	$\geq$ 125	26/200	(-0.8, 0.0)

As evidenced in Figures 6.18, 6.20, and 6.22, every single-target training produced at least one fine-tuned model capable of commanding the robot towards every Fig. 5.4 (e) scenario goal, an undisputed evidence of the agent's exceptional adaptation ability. As anticipated from *Artificial Neural Network Training* (Section 2.3.3), when tuning ANNs with this constricted setups, the longer the training process, the more the network gets overfitted towards the unique target. This issue is noted in Table 6.19, as the models that revealed a praiseworthy generalization aptitude were saved early in the training stage.

6.2.4 Obstacle-Oriented Model

The final implementation of the proposed DeepRL-based local motion planning approach in obstacle-free environments was carried out to prove its functioning when employing a network architecture designed towards domains with obstacles (Fig. 6.23, configured per Table 4.1). In this particular case, the agent, utilizing for the first time the complete state model $\mathbb{S} = \{C_{Stack}, d_T, d_O, \phi\}$ and the Reward Set 3.1 based on the obstacle-direct formula (4.5), had to learn by itself to neglect the obstacle-avoidance-related state inputs, as such data has a null/detrimental effect on the learning process.

This new framework arrangement was trained and tested in the familiar Fig. 6.11 obstacle-free scenario. The additional system variables are presented in Table 6.20, and the validation results of the obstacle-oriented navigation approach in empty environments are denoted in Fig. 6.24.

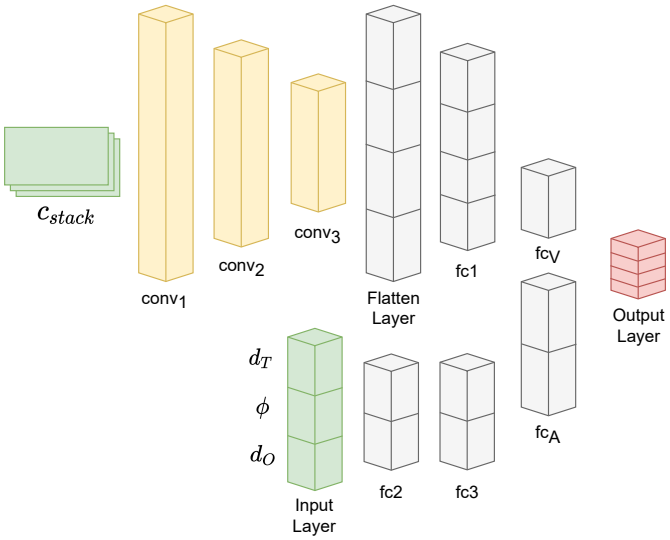


Figure 6.23: Proposed Dueling Deep-Q Network, designed towards environments with obstacles.

Table 6.20: Simulation variables, DQN framework, Artificial Neural Network architecture, action and reward models, and system hyperparameters utilized to validate the obstacle-oriented DeepRL-based navigation approach in empty environments.

Parameter	Value	Parameter	Value
DQN framework	D3QN	ANN	Fig 6.23
Action Set (Table 4.2)	Set 1	Reward Set (Table 4.3)	Set 3.1
Number of Episodes	500	Number of Steps	80
Buffer Size	40000	Learning Rate lr	0.001
Batch Size	256	Discount Factor γ	0.99
Epsilon	1.0	Epsilon Decay (per step)	0.001
Epsilon Final Value	0.01	TargetNet Update Rate	20

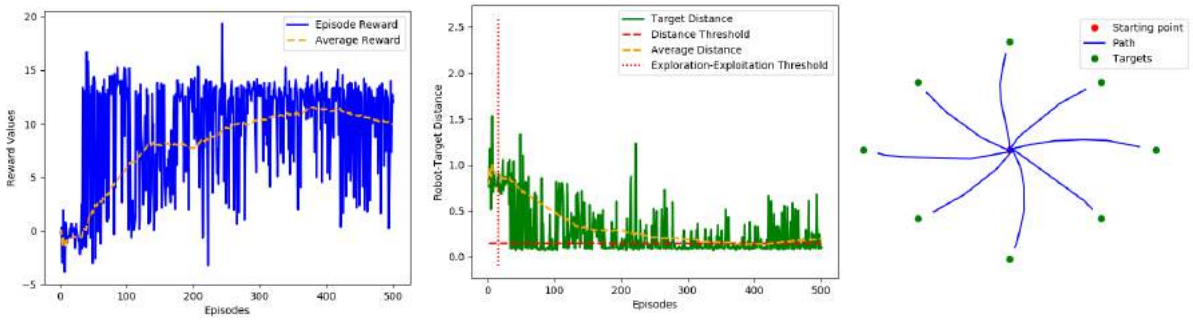


Figure 6.24: Training scores and final episode distances, and robot post-training paths resultant from validating the developed obstacle-oriented DeepRL-based local motion planning strategy over the obstacle-free Fig. 6.11 scenario.

Table 6.21: Figure 6.24 training details.

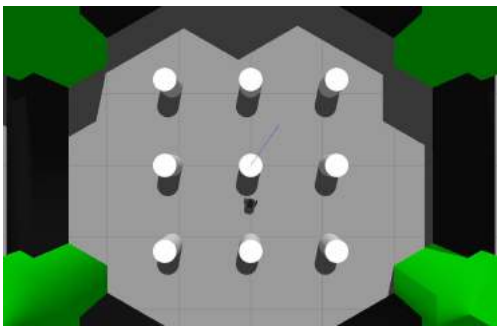
Training/Testing	Training Duration	Saved Models	Test Model	Test Model's Training Target
Fig. 6.24	$\geq$ 2 h	$\geq$ 300	85/500	(0.5, 0.5)

From the results exhibited in Fig. 6.24, it is possible to confirm that the developed DeepRL-based local navigation system, resorting to an ANN explicitly designed to handle data from domains with obstacles, was able to perform a robust training procedure in an obstacle-free environment regardless. This tryout concluded what was an exhaustive testing phase on virtual open environments. In retrospect, a considerable amount of tasks were successfully executed in this experimentation stage:

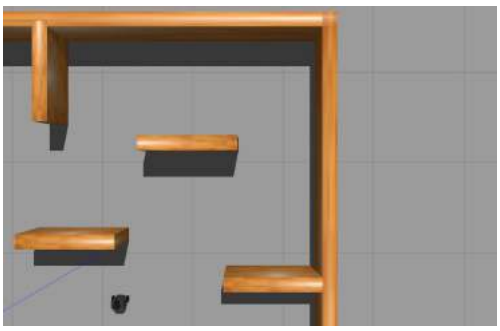
- Validation of each implemented DQN framework;
- Assessment of the various action and rewards sets' impact on the agent's training performances and post-training motion;
- Evaluation of the DeepRL agent's generalization aptitude;
- Validation of the designed obstacle-oriented system in obstacle-free environments.

6.3 Motion Planning in Environments with Obstacles

Finalized the validation of the developed DeepRL-based local motion planning strategy in obstacle-free domains, the logical subsequent step was to test the system towards environments with obstacles. As mentioned in Section 5.3.1, two established domains were chosen to sustain such implementations: the *World* and *Stage 4* Gazebo environments displayed in Fig. 6.25.



Gazebo's *World* environment



Gazebo's *Stage 4* environment

Figure 6.25: Gazebo's *World* and *Stage 4* environments.

6.3.1 Gazebo's *World* Environment

The primary application of the DeepRL-based navigation methodology in occupied domains was based on the experiment addressed in Section 6.2.4. Using the same learning method (D3QN), network architecture, and Reward Set, the agent was set up to be introduced in Gazebo's *World* stage to undertake two scenarios (Fig. 6.26). In each arrangement, two targets were defined (selected apart episode-wise) to have obstacles purposely blocking the ideal paths between them and the resting point. To attain such endpoints, the robot is bound to learn, through trial-and-error, to execute an efficient and collision-free path planning.

The first indications of the developed motion planning strategy's capability to effectively train and posteriorly endorse the agent's decision-making policy in environments with obstacles are presented in Fig. 6.27. Further simulation variables and framework hyperparameters utilized in these Gazebo's *World* trials are reported in Table 6.22.

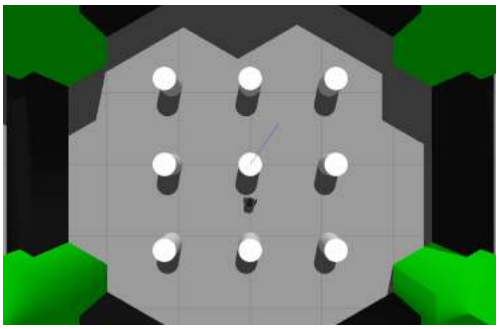
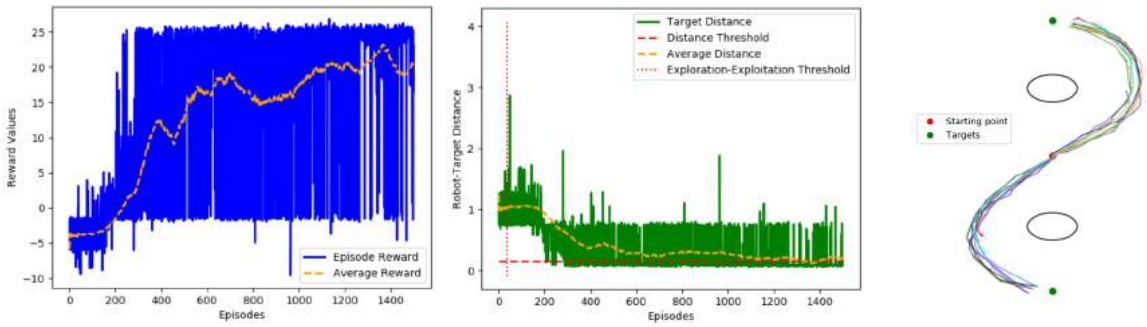


Fig. 6.27 (a) scenario	
Initial Position	Targets
(-0.5, 0.0)	(-1.5, 0.0) (0.5, 0.0)
Fig. 6.27 (b) scenario	
Initial Position	Targets
(0.0, -0.5)	(0.0, -0.5) (0.0, 1.5)

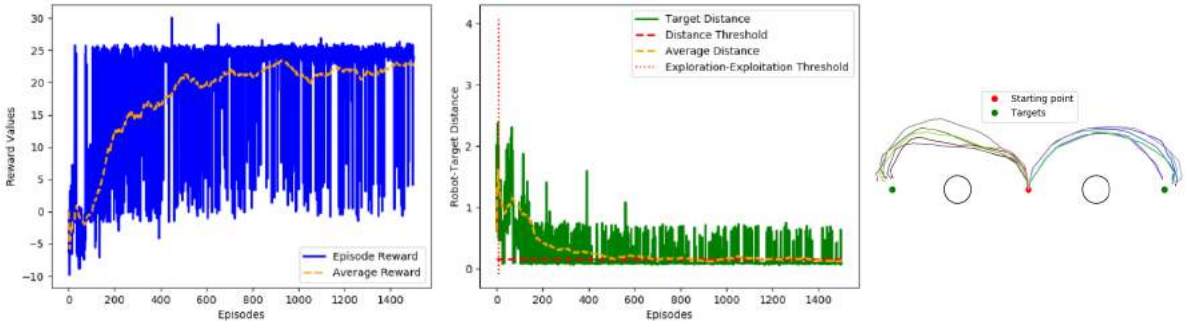
Figure 6.26: Gazebo's *World* environment and innate scenarios defined to trial the developed DeepRL-based local motion planning approach.

Table 6.22: Simulation variables, DQN framework, Artificial Neural Network architecture and parametrization, state, action and reward models, and system hyperparameters utilized in Gazebo's *World* experiment.

Parameter	**Value**	**Parameter**	**Value**
DQN framework	D3QN	ANN	Fig 6.23
State Model	$\{C_{Stack}, d_T, d_O, \phi\}$	ANN Parametrization	Table 4.1
Action Set (Table 4.2)	Set 3	Reward Set (Table 4.3)	Set 3.2
Number of episodes	500	Number of steps	150
Buffer size	75000	Learning rate lr	0.001
Batch size	256	Discount factor γ	0.99
Epsilon	1.0	Epsilon decay per step	0.001
Epsilon final value	0.01	Target net update rate	50



(a) Initial position: (-0.5, 0.0); Targets: {(-1.5, 0.0), (0.5, 0.0)}



(b) Initial position: (0.0, -0.5); Targets: {(0.0, -0.5), (0.0, 1.5)}

Figure 6.27: Training scores and final episode distances, and robot post-training paths resultant from employing the DeepRL-based local motion planning strategy over the Gazebo's *World* environment.

Table 6.23: Fig. 6.27 training details.

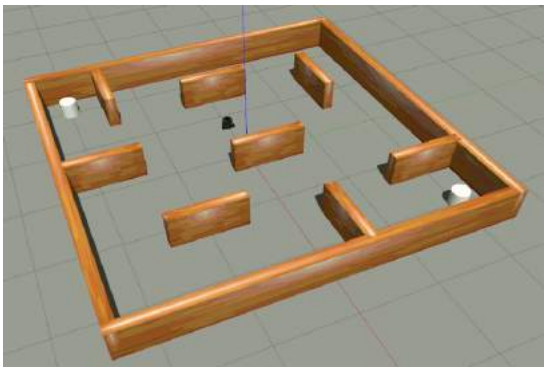
Training/Testing	Training Duration	Saved Models	Test Model	Test Model's Training Target
Fig. 6.27 (a)	$\geq$ 14 h	$\geq$ 800	1360/1500	(0.5, 0.0)
Fig. 6.27 (b)	$\geq$ 14 h	$\geq$ 1000	906/1500	(0.0, 1.5)

The Fig. 6.27 results show a very sharp learning curve with the agent converging towards high episodic rewards in both exploited scenarios. Regarding the robot's post-training behavior, it is also proved that at least one saved model was generated from each training proceeding that conferred to the agent the aptitude to continuously and autonomously adapt its motion to prevent any collision with obstacles and ultimately reach the intended targets.

6.3.2 Gazebo's *Stage 4* Environment

As demonstrated in Section 6.3.1, the Turtlebot, fitted with the suggested motion planning approach, managed to learn how to navigate in the obstacle-populated *World* stage expeditiously. Fulfilled the basic primitives of robot navigation in the *World* arrangement, the invented DeepRL-based system was then set to be tested in Gazebo's *Stage 4* environment. Three different *Stage 4* instances (Fig. 6.28) were defined with targets deliberately positioned to be reached, from the resting pose, only by a platform employing an obstacle-avoidance approach.

In addition to the method's validation in these scenarios, a survey on the agent's training and post-training behavior with larger action spaces is also presented in this section. The necessary modifications to the network's structure to comply with the resultant system's complexity augmentation are also reported.



Scenario 1 (Subsection 6.3.2.1)	
Initial Position	Target
(-0.5, -0.2)	(1.0, 0.0)
Scenario 2 (Subsection 6.3.2.3)	
Initial Position	Target
(-0.5, -0.2)	(1.7, 0.0)
Scenario 3 (Subsection 6.3.2.4)	
Initial Position	Target
(-0.5, -0.2)	(1.7, -0.5)

Figure 6.28: Gazebo's *Stage 4* environment and scenarios' initial and target points.

6.3.2.1 Scenario 1

As the name implies, Scenario 1 (Fig. 6.29) was the first arrangement that the Turtlebot was subjected to in Gazebo's *Stage 4* environment. Following the notable results in the *World* domain, the preliminary experiment in this *Stage 4* scene was sustained with the same network architecture and framework hyperparameters. The remaining simulation variables and Reward Set were adjusted according to the Scenario 1 properties (Table 6.24).

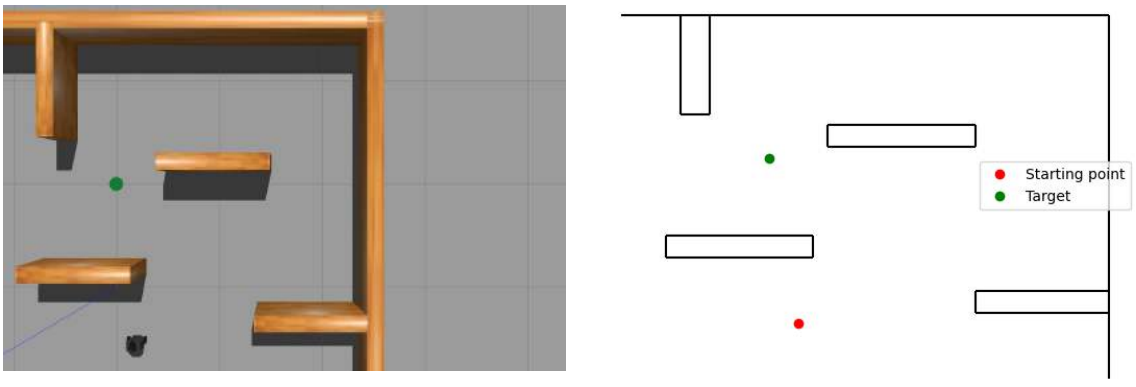


Figure 6.29: Gazebo's *Stage 4* Scenario 1.

Table 6.24: Simulation variables, DQN framework, Artificial Neural Network architecture and parametrization, state, action and reward models, and system hyperparameters utilized in the short-distance path scenario of Gazebo's *Stage 4* environment (Fig. 6.29).

Parameter	Value	Parameter	Value
DQN framework	D3QN	ANN	Fig. 6.23
State Model	$\{C_{Stack}, d_T, d_O, \phi\}$	ANN Parametrization	Table 4.1
Action Set (Table 4.2)	Set 3	Reward Set (Table 4.3)	Set 3.3
Number of Episodes	1000	Number of Steps	180
Buffer Size	180000	Learning Rate lr	0.001
Batch Size	256	Discount Factor γ	0.99
Epsilon	1.0	Epsilon Decay (per step)	0.001
Epsilon Final Value	0.01	TargetNet Update Rate	10

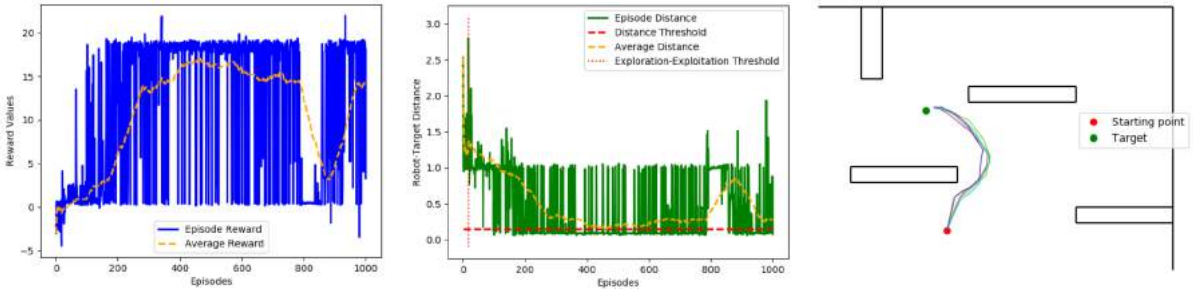


Figure 6.30: Training scores and final episode distances, and robot post-training paths resultant from employing the DeepRL-based local motion planning strategy over the Gazebo's *Stage 4* Scenario 1.

Table 6.25: Fig. 6.30 training details.

Training/Testing	Training Duration	Saved Models	Test Model
Fig. 6.30	$\geq$ 8 h	$\geq$ 550	446/1000

The results from this leading tryout, shown in Fig. 6.30, were once again very satisfactory. As the training scores indicate, the agent, under the control of the developed path planning approach, managed to learn which actions maximize the accumulated rewards. However, between episodes 700 and 900, the agent had consecutive attempts in which collided with the nearest wall from its resting point, causing a severe decline of the average episodic reward. This setback can be attributed, among other causes, to the randomness of the sampled transition tuples which can momentarily worsen the training performances. Nonetheless, the agent autonomously corrected its demeanor rapidly.

Despite the collision-free navigation evidenced in the online stage, the traced trajectories suffered from an excessive deviation from the first encountered obstacle, causing an overturn that nearly induced a collision with a second wall. With still margin to improve the robot's motion, tests with larger action spaces, reported in the following subsections, were performed in an effort to eradicate this issue.

6.3.2.1.1 Action Space with 6 Actions

Gazebo's *Stage 4* Scenario 1 - the simpler rest-target outline amongst the considered Fig. 6.28 scenarios - was further utilized to test the navigation method with extended action spaces. The results exhibited heretofore, in both obstacle-free and obstacle-populated environments (Sections 6.2 and 6.3, respectively), were obtained from the implementation of the DeepRL-based local motion planning strategy with Action Sets composed of three actions (move forward, turn left, and turn right). After validating the developed navigation pipeline with reduced action sets - a statement corroborated by the results demonstrated up to this point - a variety of tests was subsequently defined to verify the system's response when the agent is provided with a larger set of available commands to choose from.

Under this extent, the action space progressed from three to six commands (with non-pure-rotational turns), giving rise to the Action Set 4 (Table 4.2). By preserving the majority of the prior Fig. 6.30 Scenario 1 stint system configuration, a trial was executed with the newly established six-command action space in order to assess the obstacle-oriented network's capability to conduct the learning process with this expanded set. The correspondent results are presented in Fig. 6.31.

Table 6.26: Simulation parameters, adjusted from Table 6.24, for an action space of 6 commands.

Parameter	Value	Parameter	Value
DQN framework	D3QN	State Model	$\{C_{Stack}, d_T, d_O, \phi\}$
Action Set (Table 4.2)	Set 4	Reward Set (Table 4.3)	Set 3.3
Number of Episodes	1200	Number of Steps	80
Buffer Size	96000	Learning Rate lr	0.001
Batch Size	256	Discount Factor γ	0.99
Epsilon	1.0	Epsilon Decay (per step)	0.001
Epsilon Final Value	0.01	TargetNet Update Rate	10

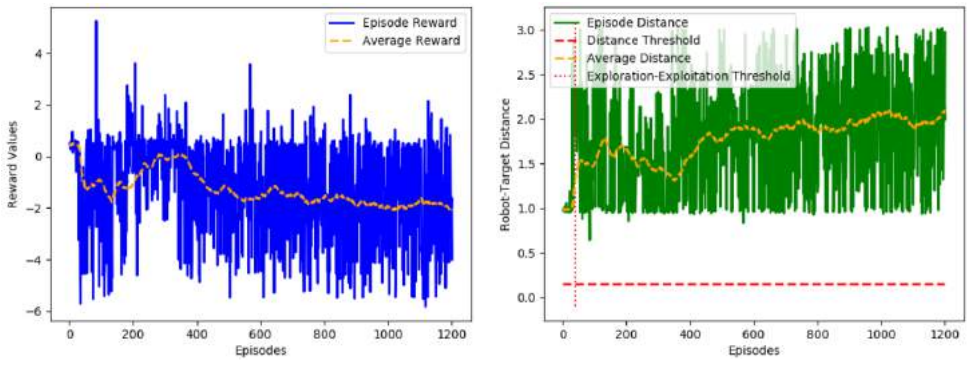


Figure 6.31: Training scores and final episode distances resultant from employing the DeepRL-based local motion planning strategy, with the Fig. 6.23 ANN and Action Set 4 (Table 4.2), over Gazebo's *Stage 4* Scenario 1.

The Fig. 6.31 results prove, unequivocally, that an adjustment to the network's structure is required to comply with the complexity augmentation resultant from the action space enlargement. In an attempt to fulfill this necessity, an additional hidden layer, fc4, was added into the designed obstacle-oriented DQN (see Fig. 6.32). Using once more the system parameters defined in Table 6.26, the results from trying out a different number of fc4 neurons are given in Fig. 6.33.

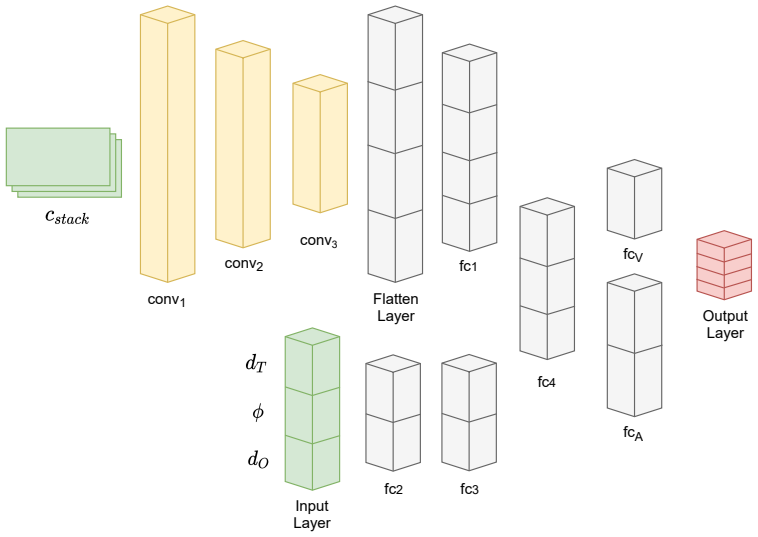
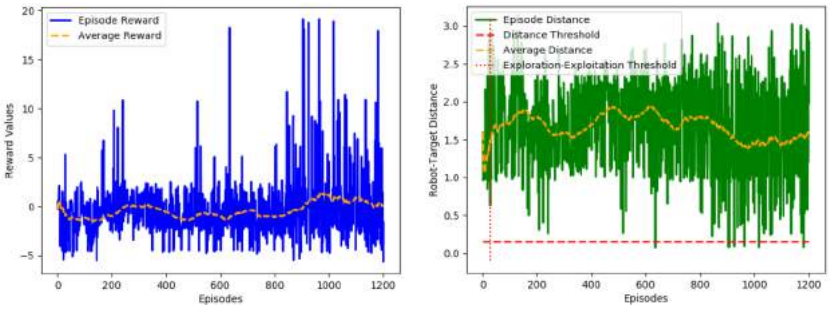
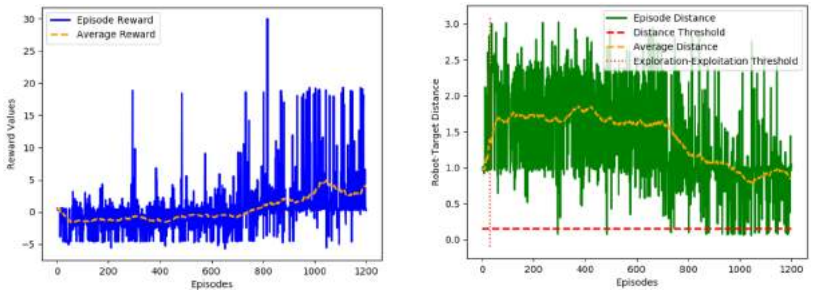


Figure 6.32: Fig. 6.23 obstacle-oriented Dueling Deep Q-Network with one additional hidden layer, fc4.

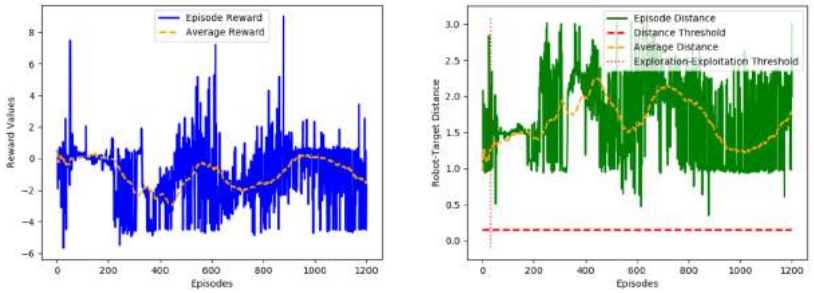
From the training scores and final episode distances presented in Fig. 6.33, it is possible to conclude that one additional hidden layer was still not enough to address the system's complexity augmentation caused by the utilization of a larger action set. Consequently, continuing the study in question, two hidden layers were introduced instead of one (see Fig. 6.34). As demonstrated by the generated outcomes of Fig. 6.35, this last-mentioned architecture was eventually able to engender fine training performances. However, the robot's behavior exhibited in the post-training phase was nearly identical to the reflected when trained with only three possible actions (Fig. 6.30), an outcome that in hindsight did not compensate for the network's increased complexity and training difficulties/challenges that come from such extension.



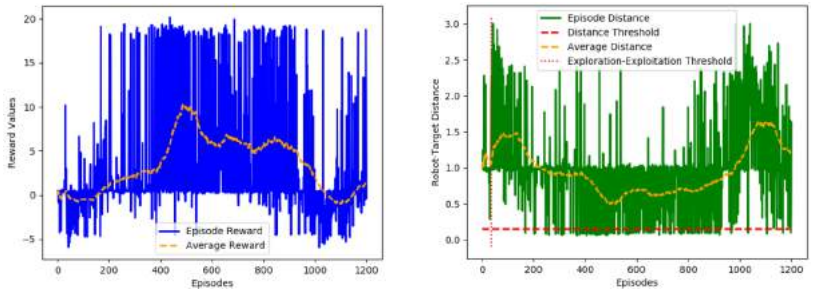
(a) fc4 neurons = 64



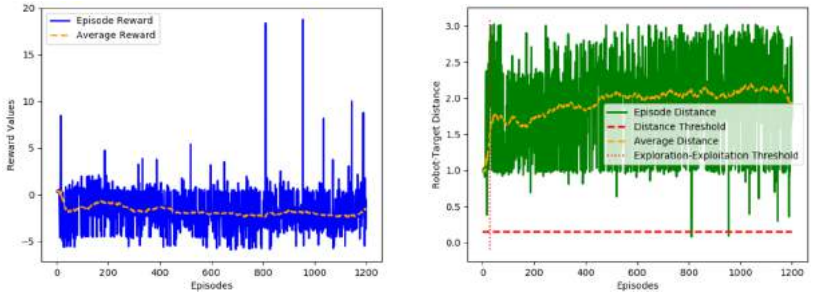
(b) fc4 neurons = 128



(c) fc4 neurons = 256



(d) fc4 neurons = 512



(e) fc4 neurons = 1024

Figure 6.33: Training scores and final episode distances resultant from employing the DeepRL-based local motion planning strategy, with the Fig. 6.32 ANN (different number of fc4 neurons) and Action Set 4 (Table 4.2), over Gazebo's *Stage 4* Scenario 1.

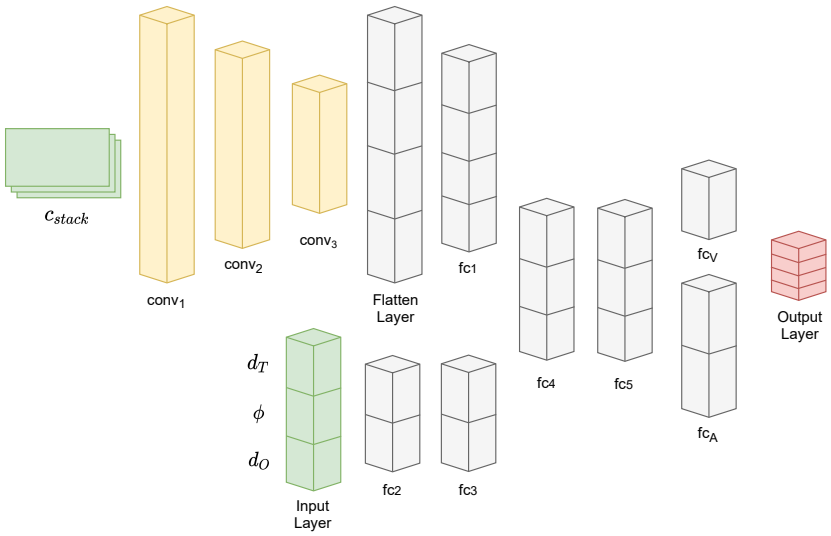
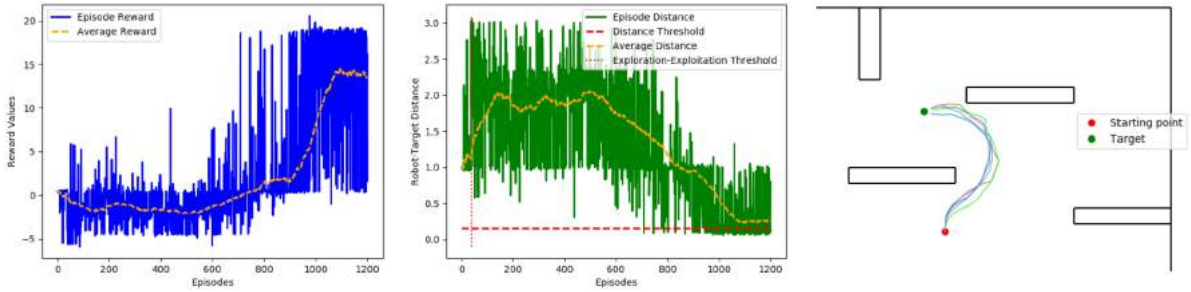


Figure 6.34: Fig. 6.23 obstacle-oriented Dueling Deep Q-Network with two additional hidden layers, fc4 and fc5.



Layer neurons: {fc4; fc5} = {128; 64}

Figure 6.35: Training scores and final episode distances, and robot's post-training paths resultant from employing the DeepRL-based local motion planning strategy, with the Fig. 6.34 ANN (fc4 = 128; fc5 = 64) and Action Set 4 (Table 4.2), over Gazebo's *Stage 4* Scenario 1.

Table 6.27: Fig. 6.35 training details.

Training/Testing	Training Duration	Saved Models	Test Model
Fig. 6.35	$\geq$ 8 h	$\geq$ 250	1145/1200

6.3.2.1.2 Action Space with 12 Actions

In line with the overall system's complexity augmentation attributable to the ANN rework (hidden layers addition) and the action space increment, an agent's post-training motion improvement was reasonably expected, a predicament that in practice was not verified. In a last attempt to mitigate the observed over-turning and promote a smoother and more effective movement on the online stage, the action space was further extended to twelve linear-angular speed pairs (Action Set 5 of Table 4.2). A new layer configuration inference was performed to secure an ANN capable of supporting such output composition accordingly.

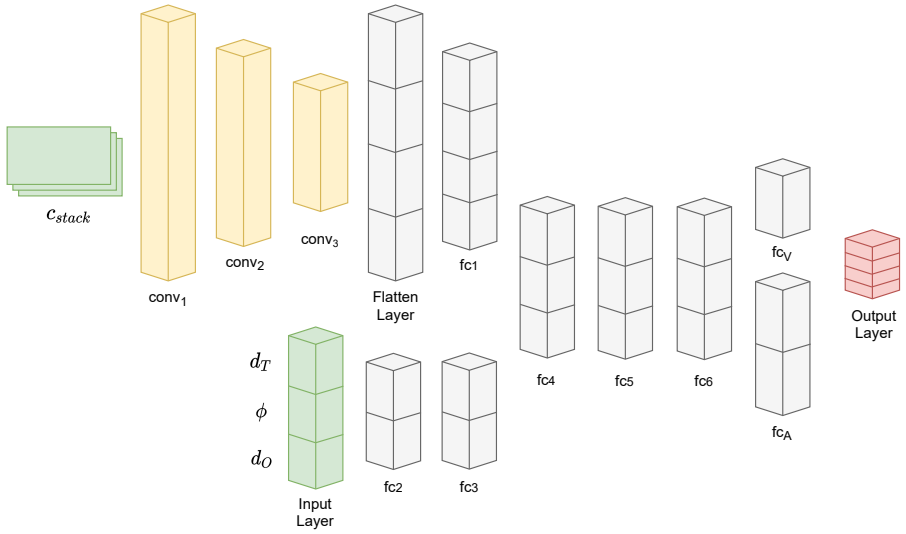


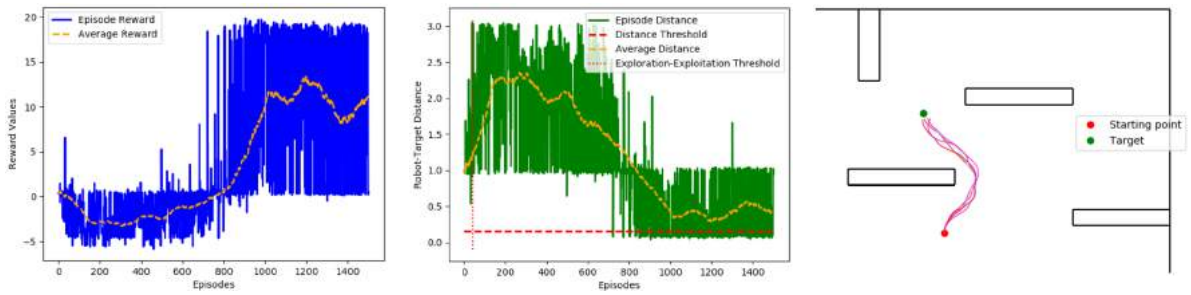
Figure 6.36: Fig. 6.23 obstacle-oriented Dueling Deep Q-Network with three additional hidden layers, fc4, f5, and fc6.

The aforementioned Artificial Neural Network remodeling procedure resulted in the architecture sketched in Fig. 6.36, a structure based on the Fig. 6.23 obstacle-oriented DQN, with three additional hidden layers.

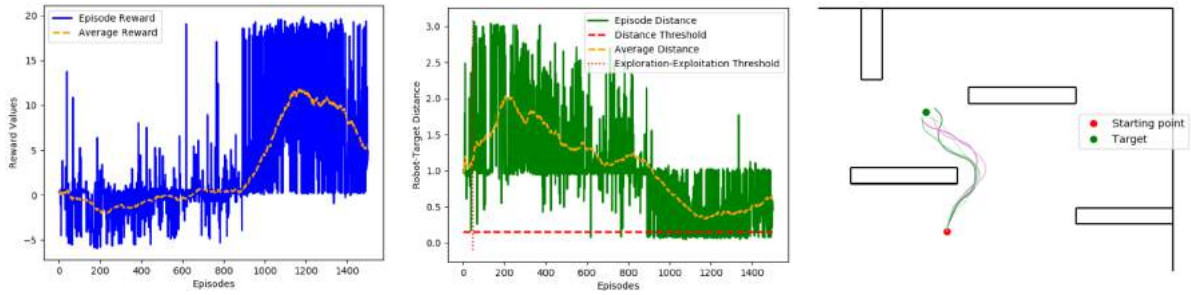
Among the various trials executed with the Table 6.28 system's configuration, three implementations stood out, both in training and testing, using the following network arrangements: {fc4; fc5; fc6} = {256; 256; 256}, {fc4; fc5; fc6} = {512; 512; 256}, and {fc4; fc5; fc6} = {512; 512; 512}. The outcomes of these experiments are presented in Fig. 6.37.

Table 6.28: Simulation parameters, adjusted from Table 6.24, for an action space of 12 commands.

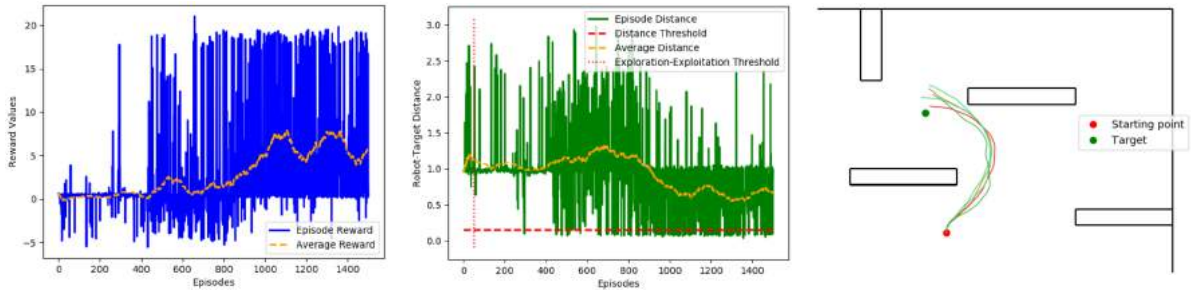
Parameter	Value	Parameter	Value
DQN framework	D3QN	ANN	Fig 6.36
Action Set (Table 4.2)	Set 5	Reward Set (Table 4.3)	Set 3.3
Number of Episodes	1500	Number of Steps	70
Buffer Size	105000	Learning Rate lr	0.001
Batch Size	256	Discount Factor γ	0.99
Epsilon	1.0	Epsilon Decay (per step)	0.001
Epsilon Final Value	0.01	TargetNet Update Rate	10



(a) Layer neurons: {fc4; fc5; fc6} = {256; 256; 256}



(b) Layer neurons: {fc4; fc5; fc6} = {512; 512; 256}



(c) Layer neurons: {fc4; fc5; fc6} = {512; 512; 512}

Figure 6.37: Training scores and final episode distances, and robot's post-training paths resultant from employing the DeepRL-based local motion planning strategy, with the Fig. 6.36 ANN (different layer configurations) and Action Set 5 (Table 4.2), over Gazebo's *Stage 4* Scenario 1.

Table 6.29: Fig. 6.37 training details.

Training/Testing	Training Duration	Saved Models	Test Model
Fig. 6.37 (a)	$\geq$ 8 h	$\geq$ 280	1198/1500
Fig. 6.37 (b)	$\geq$ 8 h	$\geq$ 170	1174/1500
Fig. 6.37 (c)	$\geq$ 8 h	$\geq$ 150	1358/1500

The Fig. 6.37 results show that one particular layer configuration, {fc4; fc5; fc6} = {256; 256; 256}, managed to produce training and online-stage performances that satisfy the fundamental objective of this stage: improve the post-training robot's motion, eliminating the previously denoted Figures 6.30 and 6.35 shortcomings.

6.3.2.2 Network's Trainable Parameters

Throughout this dissertation, the ANN complexity and its effect on the training process were always a subject that kept being mentioned, especially when utilizing larger actions spaces that required architectures with more layers and neurons. To better understand the implication of each network in the respective training, Table 6.30 presents the trainable parameters of every Artificial Neural Network utilized up to this point.

The more layers/neurons a network possesses, the greater its capacity to learn how to solve more complicated tasks. Antagonistically, when employed towards simple challenges, elaborate ANNs often perform poorly. Such affirmations can be corroborated with the Table 6.30 data: for a simple navigation scenario and having only three output neurons (number of actions), a network with 100324 trainable parameters was competent enough to produce great results (Fig. 6.30). However, by increasing the action space to six commands, the navigation task becomes much harder to sustain, and the same network (101287 neurons - variation resultant from the output layer's direct association with the action space) proves incapable of learning. Such assignment with six actions was eventually fulfilled with a network with 148839 trainable parameters partially spread over two additional hidden layers (Fig. 6.35). Arrangements with only one additional hidden layer, although having 120039 up to 434919 neurons, also revealed inept to conduct an effective training (Fig. 6.33).

In conclusion, the number of neurons and trainable parameters are meaningless if the network layers do not shape a well-balanced architecture. When working with a properly planned ANN, however, the more trainable parameters, the better the network's aptitude for learning elaborate tasks, but the more difficult and time-consuming training becomes, a trade-off that must be taken into account when designing DeepRL frameworks.

Table 6.30: Network's trainable parameters.

ANN Architecture	Results.	Trainable Parameters
Fig. 6.23	Fig. 6.30	100.324
Fig. 6.23	Fig. 6.31	101.287
Fig. 6.32		
{fc4} = {64}	Fig. 6.33 (a)	120.039
{fc4} = {128}	Fig. 6.33 (b)	141.031
{fc4} = {256}	Fig. 6.33 (c)	183.015
{fc4} = {512}	Fig. 6.33 (d)	266.983
{fc4} = {1024}	Fig. 6.33 (e)	434.919
Fig. 6.34		
{fc4; fc5} = {128; 64}	Fig. 6.35	148.839
Fig. 6.36		
{fc4; fc5; fc6} = {256; 256; 256}	Fig. 6.37 (a)	316.141
{fc4; fc5; fc6} = {512; 512; 256}	Fig. 6.37 (b)	660.717
{fc4; fc5; fc6} = {512; 512; 512}	Fig. 6.37 (c)	795.373

6.3.2.3 Scenario 2

Scenario 1 was the arrangement that endorsed the initial validation of the developed DeepRL-based motion planning approach in the Gazebo's *Stage 4* domain. Following this comprehensive testing term, an intermediate trial was performed in Scenario 2 (Fig. 6.38), a similar environment setup with a slightly more distant target from the resting point. This scenario - a preparatory scene to the more complex and final experimental stage, Gazebo's *Stage 4* Scenario 3 (Fig. 6.40) - served mainly to adjust the reward weights towards a faraway target.

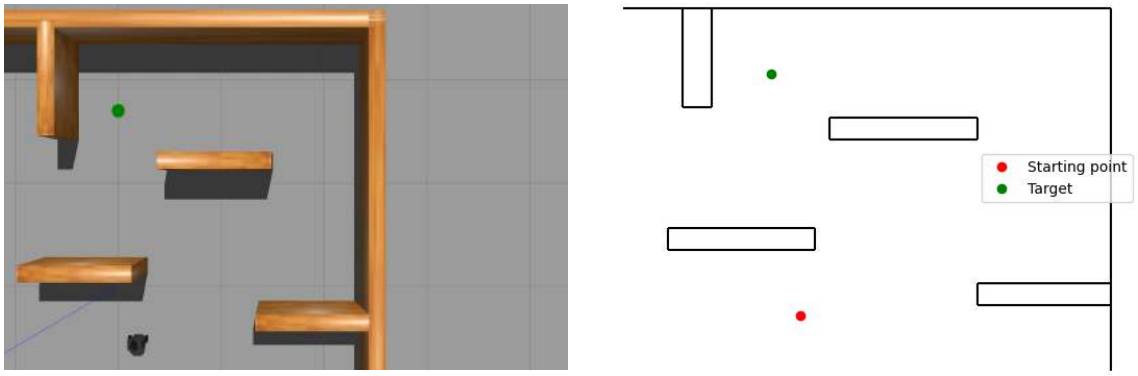


Figure 6.38: Gazebo's *Stage 4* Scenario 2.

For this implementation, the more straightforward network structure (Fig. 6.23) with an action set composed of three actions (Set 3 of Table 4.2) was sufficient to adjust the reward weights towards a bigger scenario. The results of this intervening trial are presented in Fig. 6.39.

Table 6.31: Simulation variables, DQN framework, Artificial Neural Network architecture and parametrization, state, action and reward models, and system hyperparameters utilized in Gazebo's *Stage 4* Scenario 2 (Fig. 6.38).

Parameter	Value	Parameter	Value
DQN framework	D3QN	ANN	Fig 6.23
State Model	$\{C_{Stack}, d_T, d_O, \phi\}$	ANN Parametrization	Table 4.1
Action Set (Table 4.2)	Set 3	Reward Set (Table 4.3)	Set 3.4
Number of Episodes	1000	Number of Steps	180
Buffer Size	180000	Learning Rate lr	0.001
Batch Size	256	Discount Factor γ	0.99
Epsilon	1.0	Epsilon Decay (per step)	0.001
Epsilon Final Value	0.01	TargetNet Update Rate	10

Table 6.32: Fig. 6.39 training details.

Training/Testing	Training Duration	Saved Models	Test Model
Fig. 6.39	$\geq$ 10 h	$\geq$ 350	955/1000

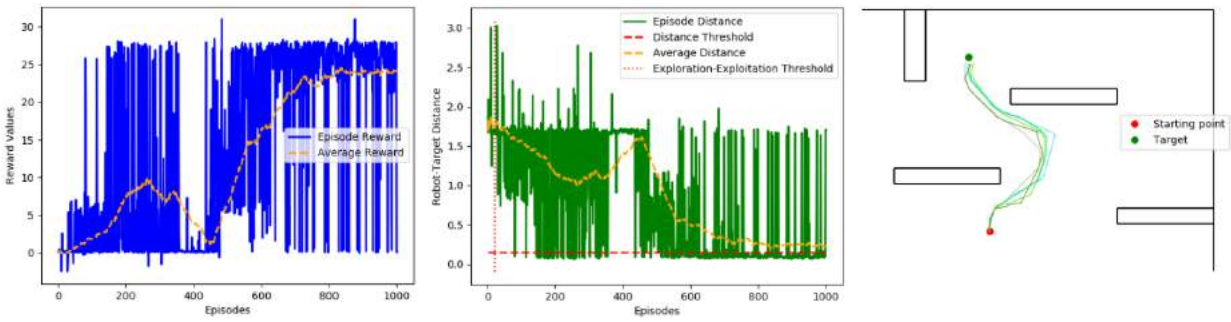


Figure 6.39: Training scores and final episode distances, and robot post-training paths resultant from employing the DeepRL-based local motion planning strategy over the Gazebo's *Stage 4* Scenario 2.

The agent's post-training behavior revealed, as expected, the same previously noted problems caused by low-scale action spaces: obstacle over-deviation and discrete-type motion. Regardless of the far-from-ideal observed paths, the virtual robot managed to attain its target employing the newly deduced Reward Set 3.4 (Table 4.3), a resolution planned for environments with outlying targets.

6.3.2.4 Scenario 3

Of all environments and scenarios utilized in prior validations and experiments, Gazebo's *Stage 4* Scenario 3 (Fig. 6.40) is the arrangement that has the greatest distance between initial and final points. With the target being further located from the robot's resting point, more steps per episode are necessary to permit the agent to explore its surroundings and ultimately attain its objective point. The more the steps, the more obstacle-avoidance and orientation-amendment situations the agent is subject to, and consequently more varied state representations it has to learn from, making the training more difficult and time-consuming. Due to these challenging training particularities, Scenario 3 was elected to uphold the dissertation's final testing phase.

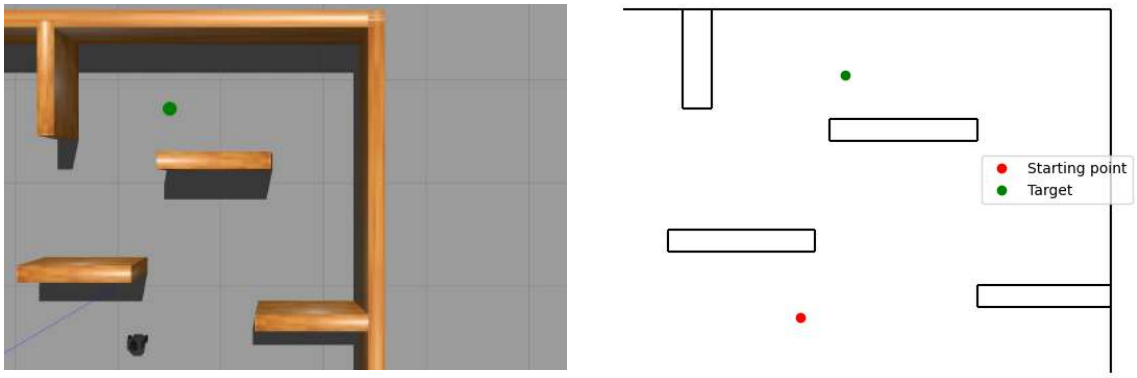


Figure 6.40: Gazebo's *Stage 4* Scenario 3.

An initial trial was carried out with the simpler network designed towards obstacles (Fig. 6.23) and the reduced Action Set 3 of Table 4.2. This procedure, backed by the simulation configuration defined in Table 6.33 (with the Reward Set inferred in Scenario 2), was executed to evaluate if the agent, guided by the developed DeepRL-based motion planning method, was qualified to overcome the Scenario 3 navigation challenge.

The results obtained from this motion planning mission, presented in Fig. 6.41, show that the agent, in a post-training phase, was able to conduct a curve and counter-curve motion, constantly adjusting its orientation according to the nearest obstacle and target point. This online-stage behavior, along with the decent training performance, demonstrates once again the robustness of the proposed navigation system in environments with obstacles.

Table 6.33: Simulation variables, DQN framework, Artificial Neural Network architecture and parametrization, state, action and reward models, and system hyperparameters utilized in Gazebo's *Stage 4* Scenario 3 (Fig. 6.40).

Parameter	Value	Parameter	Value
DQN framework	D3QN	ANN	Fig 6.23
State Model	$\{C_{Stack}, d_T, d_O, \phi\}$	ANN Parametrization	Table 4.1
Action Set (Table 4.2)	Set 3	Reward Set (Table 4.3)	Set 3.4
Number of Episodes	1000	Number of Steps	200
Buffer Size	200000	Learning Rate lr	0.001
Batch Size	256	Discount Factor γ	0.99
Epsilon	1.0	Epsilon Decay (per step)	0.001
Epsilon Final Value	0.01	TargetNet Update Rate	10

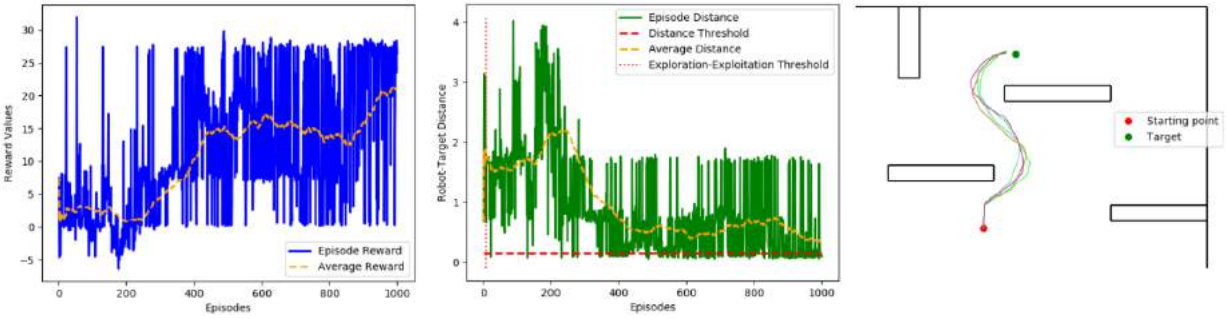


Figure 6.41: Training scores and final episode distances, and robot post-training paths resultant from employing the DeepRL-based local motion planning strategy over the Gazebo's *Stage 4* Scenario 3.

Table 6.34: Fig. 6.41 training details.

Training/Testing	Training Duration	Saved Models	Test Model
Fig. 6.41	$\geq$ 13 h	$\geq$ 200	995/1000

6.3.2.4.1 Action Space with 12 Actions

Confirmed the feasibility of Gazebo's *Stage 4* Scenario 3, the ultimate experiment to attest the adeptness and potentiality of the proposed DeepRL-based motion planning for indoor robot navigation was then outlined: train the agent (embodied with the Fig. 6.36 network architecture) in Scenario 3 with an action space of twelve actions (Set 5 of Table 4.2), and perform a series of tests on every previously exploited Gazebo virtual environment.

This last trial was designed to aggregate the validations and studies held in a single experiment. The objective was to verify, from a sole training, the agent's ability to learn how to navigate in obstacle-populated environments with a wide range of available actions, and later, in an online stage, validate its generalization aptitude by subjecting it to unknown scenarios.

The final learning procedure was executed having the Table 6.35 system and simulation parameters as configuration. The resulting training scores and final episode distances are shown in Fig. 6.42, the respective test in Gazebo's *Stage 4* Scenario 3 (training arrangement) is displayed in Fig. 6.43, and lastly, the motion executed by the agent over unfamiliar environments, *i.e.*, virtual domains unobserved in training, are presented in Fig. 6.44.

Table 6.35: Simulation parameters, adjusted from Table 6.33, for an action space of 12 commands.

Parameter	Value	Parameter	Value
DQN framework	D3QN	ANN	Fig 6.36
State Model	$\{C_{Stack}, d_T, d_O, \phi\}$	ANN Parametrization	Table 4.1 w/ {fc4; fc5; fc6} = {256; 256; 256}
Action Set (Table 4.2)	Set 5	Reward Set (Table 4.3)	Set 3.4
Number of Episodes	3000	Number of Steps	180
Buffer Size	250000	Learning Rate lr	0.001
Batch Size	256	Discount Factor γ	0.99
Epsilon	1.0	Epsilon Decay (per step)	0.001
Epsilon Final Value	0.01	TargetNet Update Rate	10

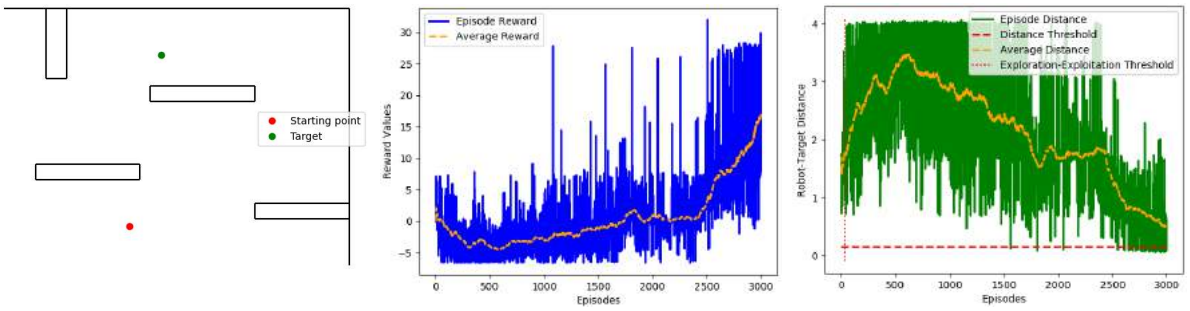


Figure 6.42: Scenario 3 arrangement, and training scores and final episode distances resultant from employing the DeepRL-based local motion planning strategy, with the Fig. 6.36 ANN ({fc4; fc5; fc6} = {256; 256; 256}) and Action Set 5 (Table 4.2), over Gazebo's *Stage 4* Scenario 3.

Table 6.36: Fig. 6.42 training details.

Training/Testing	Training Duration	Saved Models	Test Model
Fig. 6.42	$\geq$ 20 h	$\geq$ 100	2936/3000

The fact that the agent attained all the targets it was instructed for is utterly impressive. In Gazebo's *Stage 4* scenarios, due to their resemblance/parity to the training arrangement (Gazebo's *Stage 4* Scenario 3), the Turtlebot managed to conduct a remarkable collision-free navigation in the online phase, never jeopardizing its primary objective of reaching the target point (Figures 6.43, 6.44b, 6.44c, and 6.44d). The same occurrence was verified in the Gazebo *World* trajectories (Figures 6.44e and 6.44f). However, in Gazebo's empty environment, a domain completely different from the one used in training, the robot behavior diverged somewhat from the acceptable standards (Fig. 6.44a). Even so, the Turtlebot managed to reach the threshold of all targets.

This experiment put an end to the navigation system testing. Noticeably, substantiated by the results presented in this dissertation, it is possible to conclude that the developed DeepRL-based local motion planning approach for indoor robot navigation, designed from scratch, not only operates effectively in environments with and without obstacles, but also demonstrates an immense potential to be prospectively applied in real domains. Nonetheless, it is essential to continue improving the suggested strategy to materialize this supposition.

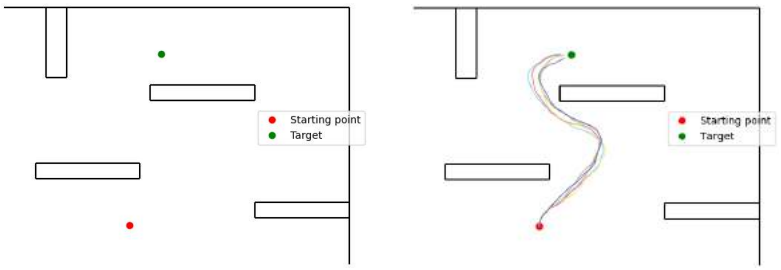
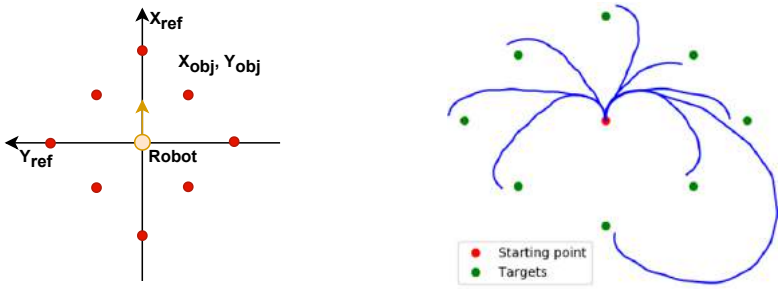
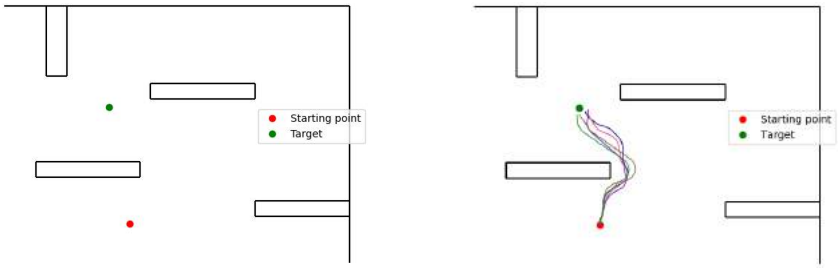


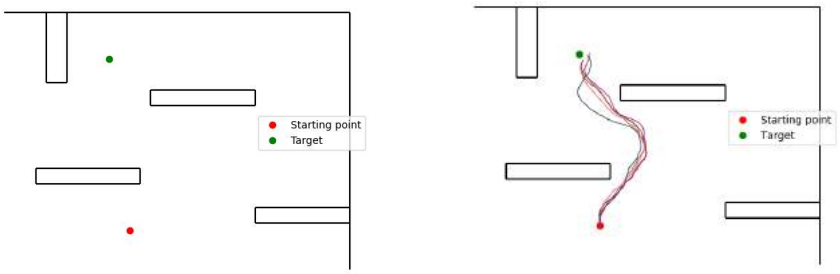
Figure 6.43: Scenario 3 arrangement and online-stage testing (post Fig. 6.42 training).



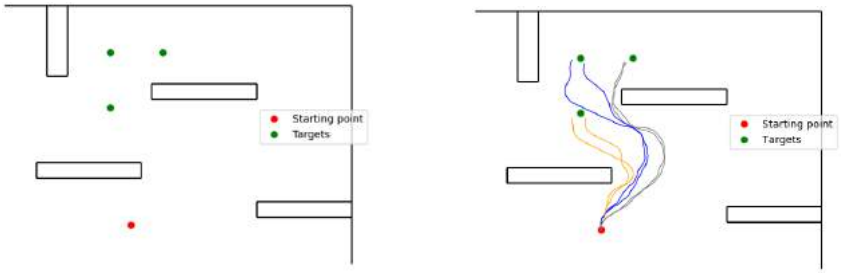
(a) Gazebo's obstacle-free Fig. 5.4 (e) scenario.



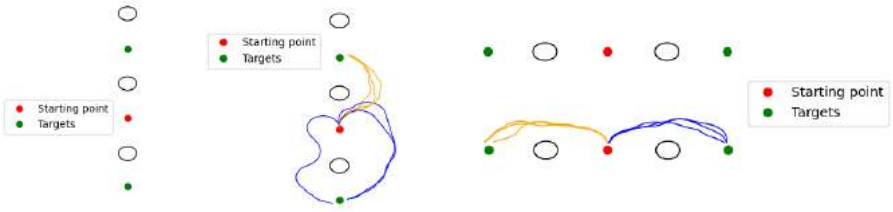
(b) Gazebo's *Stage 4* Scenario 1.



(c) Gazebo's *Stage 4* Scenario 2.



(d) Gazebo's *Stage 4* Scenario 1, 2, and 3.



(e) Gazebo's *World* Scenario (a). (f) Gazebo's *World* Scenario (b).

Figure 6.44: Fig. 6.42 post-training experiments in Gazebo's obstacle-free, *World*, and *Stage 4* environments.

7

Conclusion

In this dissertation, a Deep Reinforcement Learning-based motion planning approach, revolved around the Deep Q-Learning algorithm, is proposed. The urge to conceive such system came from the noted lack of existing strategies that take into consideration the straightforward transferability to different sensor-equipped mobile platforms. In an attempt to address this deficit in the DeepRL-based indoor robot navigation scope, the suggested technique was developed with each of its components (DQN framework and state, action, and reward models) specifically designed from scratch to establish a differentiated sensor-agnostic local motion planning method. A survey on robot motion planning and Machine Learning's Reinforcement Learning, Deep Learning, and the contemporary Deep Reinforcement Learning fields was attentively consummated to support, conceptually, this innovative framework intent.

The starting point of the development process was the implementation of the Deep Q-Learning, Dueling Deep Q-Learning, and the state-of-the-art Dueling Double Deep Q-Learning algorithms based on their respective pseudo-codes. These Deep Q-Network-based learning methodologies were later validated in referenced OpenAI Gym benchmark environments with the assistance of prevailing ANN structures and hyperparameters.

After validating the implemented Deep Q-Network-based methodologies, the proposed system's state, action, and reward models - essential elements of RL and DeepRL architectures - were devised. The defined state model was composed of continuous features that portray the robot's orientation and distance to the target, and distance to the nearest obstacle. A costmap stack, the distinguishing factor of the suggested navigation framework, was further outlined as a state integral part, representing the agent's surrounding environment and the short-term paths resultant from its decision-making. Different reward models were also designed, some of them oriented towards obstacle-free environments, and others more intricate to compute appropriate rewards in the more complex obstacle-populated domains. Moreover, several sets of actions were profiled with various linear-angular speed pair combinations.

Upon implementing and validating the DQN-based learning algorithms and establishing the state, action, and reward models, the only missing element to complete the proposed local motion planning pipeline was a Deep Q-Network capable of properly converting the state representations into $Qvalues$. Such architecture was inferred on an open environment, via trial-and-error, using the networks utilized in the OpenAI Gym validation stage as a basis.

With every module created, the proposed motion planning algorithm was finally set to be applied to a virtual robot - the Turtlebot - to ultimately conduct its action-selection policy. Being a framework plotted entirely from scratch, the testing/validation phase was launched without any working guarantees (apart from the response shown when deducting the base ANN structures).

The preliminary experiments of the proposed local motion planning approach were carried out in a Gazebo obstacle-free environment. The system was validated in these types of domains using each implemented DQN framework. The Dueling Double Deep Q-Network, in line with the acknowledged theoretical concepts retained from the *Background Material* and *State of the Art* sections, was the learning methodology that revealed the best overall training executions. D3QN was therefore chosen to integrate the arranged DeepRL-based local motion planning system onward.

In addition to the validation step on obstacle-free domains, individual studies were conducted to assess the agent's generalization aptitude. An obstacle-oriented ANN architecture was also introduced into the suggested navigation pipeline to corroborate the system's functioning in open domains, even when designed towards obstacle-populated environments. From the outcomes of these trials - strong indicators of the great potential of the presented framework - some conclusions were inferred:

- The utilized ANN must have the ability to map its state inputs into output commands;
- The employed state model must meticulously represent the robot's environment surroundings and provide the agent with essential data regarding the robot-target and robot-obstacles relation;
- The reward model must reward or penalize the agent in a fair and balanced way according to the outcomes provoked by its actions;
- The used action set must be composed of commands that the agent can execute;
- The employed reward model and action set directly influence the quality of training and the resultant post-training robot motion;
- The larger the state representation (number of network inputs), the more difficult training becomes;
- The number of episodes and steps must be defined to provide the agent enough exploration room, and must be sufficient to permit the agent to reach its goal;
- The size of the utilized replay buffer should be enough to preserve the transition data of every step, allowing the agent to learn from a broad domain of past experiences. However, the larger the replay buffer, the greater the computational training cost;

- The training batch - the set of past experiences used in each step to tune the network - should be as large as possible. The larger the batch size, the more robust and all-inclusive the training turns, but the longer the elapsed step time becomes, making the agent less responsive. The employed GPU also influences this trade-off: the more powerful the GPU, the larger the batch that can be processed between steps without compromising the integrity of training. For the NVIDIA GeForce GTX 1060 (GPU utilized in this project), the dimension of the batch was settled at 256;
- Regarding the target ANN update rate, the shorter the update step period, the more volatile the learning process becomes.

Once the proposed navigation system was validated in obstacle-free environments, a natural subsequent testing stage was instituted in domains with static obstacles. The DeepRL-based framework was also tested with enlarged sets of speed commands, allowing the agent to select actions from a more comprehensive extent. For these latter tests, it was necessary to rethink and adjust the framework's network to comply with the system's evolution.

The results obtained from this alignment were again outstanding, as the agent exhibited the capability to safely navigate in obstacle-populated scenarios and reach the proposed targets, even in environments different from those experienced in training. This set of trials concluded what was a thorough experimental stage. Further arguments were deduced during its execution:

- The larger the action space (number of network outputs), the more difficult training becomes;
- Following an overall system's complexity growth, resultant for instance from the state model or action space augmentation, the employed network should be revised and, in case of need, updated by increasing the number of layers' neurons or adding hidden layers to its architecture;
- A more complex network inherently has more trainable parameters that must be adjusted throughout training, making it more challenging and time-consuming.

The Turtlebot, exercising the developed DeepRL-based local motion planning for indoor robot navigation, proved capable of learning via trial-and-error, with no datasets nor *a priori* knowledge of the environment, to execute a collision-free motion and attain its defined targets. The fact that this project was endorsed without any guarantees of the proposed navigation method's functioning makes the substantiated results and validations doubly remarkable and satisfactory.

7.1 Future Work

The results presented in this dissertation demonstrate the full potential of the developed navigation framework. Nonetheless, it is recognized that there is still a wide range of improvements that can be made.

Dynamic Obstacles

Introduce the trained agent in an environment with dynamic obstacles.

Final Orientation

Restructure both the neural networks and the reward models to enable the specification of the robot's final orientation (at the target location).

Superior Hardware

Test the developed navigation framework having a superior GPU sustaining training.

Sensor-Agnostic Framework

The presented navigation framework was designed to be independent of the type of robot and sensor. One of the points to be worked on in the future would be to verify this premise.

Real-World Validation

Validate the developed DeepRL-based motion planning algorithm in real mobile platforms and environments.

Socially Aware Motion Planning

Adjust the proposed navigation method to enable a human-sensible motion planning. In order to achieve a socially aware conduct, the developed DeepRL-based path planning approach would need to be complemented with a DL-based perception module. With the merge of both perception and navigation units, an end-to-end DeepRL-based socially aware motion planning strategy could be attained (sketch in Fig 7.1). When applied to a mobile robot, it would detect and track labeled real-world elements, such as people and other robots, and use that data to tune the platform's behavior.

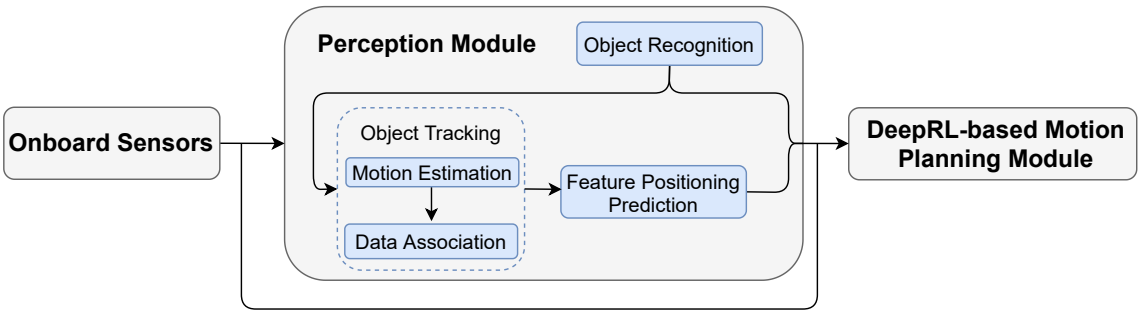


Figure 7.1: End-to-end DeepRL-based socially aware motion planning block diagram.

Bibliography

[1] Luís Garrote, Diogo Temporão, Samuel Temporão, Ricardo Pereira, Tiago Barros, and Urbano J. Nunes. Improving Local Motion Planning with a Reinforcement Learning Approach. In *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 206–213, April 2020.

[2] Turtlebot3 features. https://emanual.robotis.com/.

[3] 360 laser distance sensor lds-01. https://www.robot-advance.com/EN/art-360-laser-distance-sensor-lds-01-2352.htm.

[4] Nvidia geforce gtx 1060. https://www.zotac.com/us/product/graphics_card/zotac-geforce-gtx-1060-amp-editionspec.

[5] A Roadmap for US Robotics: From Internet to Robotics, 2020.

[6] Jiyu Cheng, Hu Cheng, Max Q.-H. Meng, and Hong Zhang. Autonomous Navigation by Mobile Robots in Human Environments: A Survey. In *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1981–1986, December 2018.

[7] Xiaogang Ruan, Dingqi Ren, Xiaoqing Zhu, and Jing Huang. Mobile Robot Navigation based on Deep Reinforcement Learning. In *2019 Chinese Control And Decision Conference (CCDC)*, pages 6174–6178, June 2019. ISSN: 1948-9447.

[8] Szilárd Aradi. Survey of deep reinforcement learning for motion planning of autonomous vehicles, 2020. arXiv: 2001.11231.

[9] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[10] Vladimir Nasteski. An overview of the supervised machine learning methods. *HORIZONS.B*, 4:51–62, December 2017.

[11] Ibm - unsupervised learning. www.ibm.com/cloud/learn/unsupervised-learning.

[12] R. S. Sutton and A. G. Barto. Reinforcement Learning: An Introduction. *MIT Press*, 2018.

[13] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine*, 34(6):26–38, November 2017. Conference Name: IEEE Signal Processing Magazine.

[14] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani. State-of-the-Art Deep Learning: Evolving Machine Intelligence Toward Tomorrow's Intelligent Network Traffic Control Systems. *IEEE Communications Surveys Tutorials*, 19(4):2432–2455, 2017. Conference Name: IEEE Communications Surveys Tutorials.

[15] Yu Fan Chen, Michael Everett, Miao Liu, and Jonathan P. How. Socially aware motion planning with deep reinforcement learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1343–1350, September 2017. ISSN: 2153-0866.

[16] Michael Everett, Yu Fan Chen, and Jonathan P. How. Motion Planning Among Dynamic, Decision-Making Agents with Deep Reinforcement Learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3052–3059, October 2018. ISSN: 2153-0866.

[17] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[18] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling Network Architectures for Deep Reinforcement Learning. *arXiv:1511.06581 [cs]*, April 2016. arXiv: 1511.06581.

[19] Robotis - motion planning. https://robotics.umich.edu/research/focus-areas/motion-planning/.

[20] Han-ye Zhang, Wei-ming Lin, and Ai-xia Chen. Path Planning for the Mobile Robot: A Review. *Symmetry*, 10(10):450, October 2018. Number: 10 Publisher: Multidisciplinary Digital Publishing Institute.

[21] Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, February 1998.

[22] Nicola Tomatis, I. Nourbakhsh, and Roland Siegwart. Combining Topological and Metric: A Natural Integration for Simultaneous Localization and Map Building. January 2001.

[23] Juergen Schmidhuber. Deep Learning in Neural Networks: An Overview. *Neural Networks*, 61:85–117, January 2015. arXiv: 1404.7828.

[24] Md. Zahangir Alom, Tarek Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Nasrin, Mahmudul Hasan, Brian Essen, Abdul Awwal, and Vijayan Asari. A State-of-the-Art Survey on Deep Learning Theory and Architectures. *Electronics*, 8:292, March 2019.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[26] Zhixing Tan, Shuo Wang, Zonghan Yang, Gang Chen, Xuancheng Huang, Maosong Sun, and Yang Liu. Neural machine translation: A review of methods, resources, and tools, 2020.

[27] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.

[28] Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351, 2017.

[29] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. Conference Name: Proceedings of the IEEE.

[30] H. Zhang, L. Zhang, and Y. Jiang. Overfitting and underfitting analysis for deep learning based end-to-end communication systems. In *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–6, 2019.

[31] Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay, 2018.

[32] Lei Tai and Ming Liu. Towards Cognitive Exploration through Deep Reinforcement Learning for Mobile Robots. *arXiv:1610.01733 [cs]*, October 2016. arXiv: 1610.01733.

[33] Yuxi Li. Deep reinforcement learning: An overview, 2018.

[34] Muhammad Mudassir Ejaz, Tong Boon Tang, and Cheng-Kai Lu. Autonomous Visual Navigation using Deep Reinforcement Learning: An Overview. In *2019 IEEE Student Conference on Research and Development (SCOReD)*, pages 294–299, October 2019. ISSN: 2643-2447.

[35] Lingxiao Wang, Qi Cai, Zhuoran Yang, and Zhaoran Wang. Neural policy gradient methods: Global optimality and rates of convergence, 2019.

[36] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. *arXiv:1602.01783 [cs]*, June 2016. arXiv: 1602.01783.

[37] Shangtong Zhang and Richard S. Sutton. A deeper look at experience replay, 2018.

[38] Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. *arXiv:1509.06461 [cs]*, December 2015. arXiv: 1509.06461.

[39] Pablo Marín, Ahmed Hussein, David Martín Gómez, and Arturo de la Escalera. Global and Local Path Planning Study in a ROS-Based Research Platform for Autonomous Vehicles. *Journal of Advanced Transportation*, 2018:1–10, February 2018.

[40] Risald, Antonio E. Mirino, and Suyoto. Best routes selection using Dijkstra and Floyd-Warshall algorithm. In *2017 11th International Conference on Information Communication Technology and System (ICTS)*, pages 155–158, October 2017. ISSN: 2338-185X.

[41] Xiang Liu and Daoxiong Gong. A comparative study of A-star algorithms for search and rescue in perfect maze. April 2011.

[42] Anthony Stentz. The Focussed D* Algorithm for Real-Time Replanning. In *In Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1652–1659, 1995.

[43] Luis Garrote, Cristiano Premebida, Marco Silva, and Urbano Nunes. An RRT-based navigation approach for mobile robots and automated vehicles. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, pages 326–331, July 2014. ISSN: 2378-363X.

[44] Qidan Zhu, Yongjie Yan, and Zhuoyi Xing. Robot Path Planning Based on Artificial Potential Field Approach with Simulated Annealing. In *Sixth International Conference on Intelligent Systems Design and Applications*, volume 2, pages 622–627, October 2006. ISSN: 2164-7151.

[45] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The Dynamic Window Approach to Collision Avoidance. *Robotics & Automation Magazine, IEEE*, 4:23–33, April 1997.

[46] Martin Keller, Frank Hoffmann, Carsten Hass, Torsten Bertram, and Alois Seewald. Planning of Optimal Collision Avoidance Trajectories with Timed Elastic Bands. *IFAC Proceedings Volumes*, 47(3):9822–9827, January 2014.

[47] B. Zuo, J. Chen, L. Wang, and Y. Wang. A reinforcement learning based robotic navigation system. In *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3452–3457, October 2014. ISSN: 1062-922X.

[48] G. Yen and T. Hickey. Reinforcement learning algorithms for robotic navigation in dynamic environments. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*, volume 2, pages 1444–1449 vol.2, 2002.

[49] Deepmind. https://deepmind.com/.

[50] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. page 9.

[51] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Has-

sabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. Number: 7587 Publisher: Nature Publishing Group.

[52] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *arXiv:1712.01815 [cs]*, December 2017. arXiv: 1712.01815.

[53] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, November 2019. Number: 7782 Publisher: Nature Publishing Group.

[54] Lei Tai, Giuseppe Paolo, and Ming Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation, 2017.

[55] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[56] Jing Liang, Utsav Patel, Adarsh Jagan Sathyamoorthy, and Dinesh Manocha. Realtime Collision Avoidance for Mobile Robots in Dense Crowds using Implicit Multi-sensor Fusion and Deep Reinforcement Learning. *arXiv:2004.03089 [cs]*, April 2020. arXiv: 2004.03089.

[57] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[58] Linhai Xie, Sen Wang, Andrew Markham, and Niki Trigoni. Towards Monocular Vision based Obstacle Avoidance through Deep Reinforcement Learning. *arXiv:1706.09829 [cs]*, June 2017. arXiv: 1706.09829.

[59] Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. Reinforcement Learning through Asynchronous Advantage Actor-Critic on a GPU. *arXiv:1611.06256 [cs]*, March 2017. arXiv: 1611.06256.

[60] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004.

[61] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding, 2014.

[62] E. Rohmer, S. P. N. Singh, and M. Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326, 2013.

[63] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[64] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. *arXiv:1710.06537 [cs]*, March 2018. arXiv: 1710.06537.

[65] B. Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep Reinforcement Learning for Autonomous Driving: A Survey. *arXiv:2002.00444 [cs]*, February 2020. arXiv: 2002.00444.

[66] Mark G Sobell. *A practical guide to Ubuntu Linux*. Pearson Education, 2015.

[67] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Ng. *ROS: an open-source Robot Operating System*, volume 3. January 2009. Journal Abbreviation: ICRA Workshop on Open Source Software Publication Title: ICRA Workshop on Open Source Software.

[68] M. Santos Pessoa de Melo, J. Gomes da Silva Neto, P. Jorge Lima da Silva, J. M. X. Natario Teixeira, and V. Teichrieb. Analysis and comparison of robotics 3d simulators. In *2019 21st Symposium on Virtual and Augmented Reality (SVR)*, pages 242–251, 2019.

[69] Serena Ivaldi, Vincent Padois, and Francesco Nori. Tools for dynamics simulation of robots: a survey based on user feedback, 2014.

[70] Webots. http://www.cyberbotics.com. Open-source Mobile Robot Simulation Software.

[71] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[72] Sven Koenig, Maxim Likhachev, and David Furcy. Lifelong Planning A∗. *Artificial Intelligence*, 155(1):93–146, May 2004.

[73] HyeongRyeol Kam, Sung-Ho Lee, Taejung Park, and Chang-Hun Kim. Rviz: a toolkit for real domain data visualization, 10 2015.

[74] Turtlebot. https://www.turtlebot.com/.

[75] Python. https://www.python.org/.

[76] Pycharm. https://www.jetbrains.com/pycharm/.

[77] Tensorflow. https://www.tensorflow.org/.

[78] Numpy. https://numpy.org/.

[79] Pandas. https://pandas.pydata.org/.

[80] Rospy. http://wiki.ros.org/rospy.

[81] Matplotlib. https://matplotlib.org/.

[82] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[83] Cuda. https://developer.nvidia.com/cuda-zone.

[84] cudnn. https://developer.nvidia.com/cudnn.

[85] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[86] Box2d. https://box2d.org/.

[87] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[88] Ann learning rate. https://machinelearningmastery.com/.

A

DQN Pseudo-codes

Deep Q-Learning and Dueling Deep Q-Learning

Algorithm 2: Deep Q-Learning and Dueling Deep Q-Learning pseudo-code [17]

Define the initial and target points
Define the number of training episodes e and respective steps t
Define the network's hyperparameters and the state, action, and reward models
Initialize the policy DQN (θ) and the target DQN (θ^-)
Initialize the replay buffer D to capacity N and define the batch size N_b
for *episode* $e \in \{1, , M\}$ **do**
    Environment setup, s_0
    for *step* $t \in \{1, , T\}$ **do**
        Select action a_t from s_t using an ϵ-greedy exploration method
        Take action a_t
        Observe s_{t+1}
        Compute reward value r_t
        Store transition tuple (s_t, a_t, s_{t+1}, r_t) in D
        Sample a random minibatch of N_b tuples (s_j,a_j, s_{j+1}, r_j) from D
        $y_t(a) = r_j + \gamma \cdot max_a \mathrm{Q}(s_{j+1}, a; \theta^-)$
        Perform a gradient descent step with loss $||y_t(a) - \mathrm{Q}(s_j, a_j; \theta)||^2$, updating θ
        if t % N^- *is equal to 0* **then**
            Replace the target DQN parameters $\theta^- \leftarrow \theta$
        end
        $s_t \leftarrow s_{t+1}$
    end
end

Dueling Double Deep Q-Learning

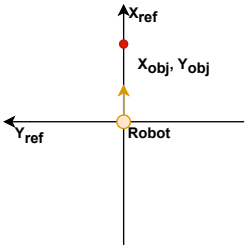
Algorithm 3: Dueling Double Deep Q-Learning pseudo-code, adapted from [7]

Define the initial and target points
Define the number of training episodes e and respective steps t
Define the network's hyperparameters and the state, action, and reward models
Initialize the policy DQN (θ) and the target DQN (θ^-)
Initialize the replay buffer D to capacity N and define the batch size N_b
for *episode* $e \in \{1, , M\}$ **do**
    Environment setup, s_0
    for *step* $t \in \{1, , T\}$ **do**
        Select action a_t from s_t using an ϵ-greedy exploration method
        Take action a_t
        Observe s_{t+1}
        Compute reward value r_t
        Store transition tuple (s_t, a_t, s_{t+1}, r_t) in D
        Sample a random minibatch of N_b tuples (s_j,a_j, s_{j+1}, r_j) from D
        $a^{max}(s_{j+1}; \theta) = argmax_a \mathrm{Q}(s_{j+1}, a; \theta)$
        $y_t(a) = r_j + \gamma \cdot \mathrm{Q}(s_{j+1}, a^{max}(s_{j+1}; \theta); \theta^-)$
        Perform a gradient descent step with loss $||y_t(a) - \mathrm{Q}(s_j, a_j; \theta)||^2$, updating θ
        if *t % N^- is equal to 0* **then**
            Replace the target DQN parameters $\theta^- \leftarrow \theta$
        end
        $s_t \leftarrow s_{t+1}$
    end
end

B

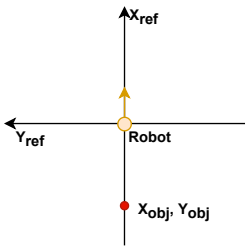
Artificial Neural Network Inference

Training and Testing Scenarios



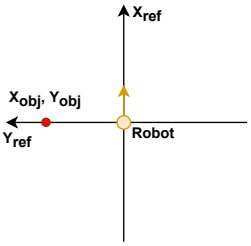
Initial Position: (0.0, 0.0)
Target: (0.8, 0.0)

(a) Move forward



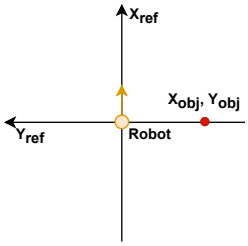
Initial Position: (0.0, 0.0)
Target: (-0.8, 0.0)

(b) Move backward



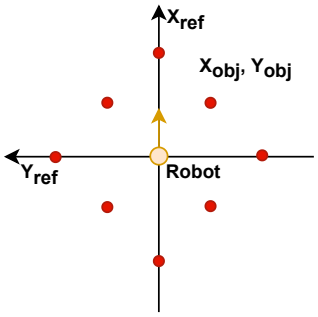
Initial Position: (0.0, 0.0)
Target: (0.0, 0.8)

(c) Turn left



Initial Position: (0.0, 0.0)
Target: (0.0, -0.8)

(d) Turn right



Initial Position (X,Y)	**Target** (X_{obj},Y_{obj})
(0.0, 0.0)	(0.8, 0.0)
	(0.5, 5.0)
	(0.0, 0.8)
	(-0.5, 0.5)
	(-0.8, 0.0)
	(-0.5, -0.5)
	(0.0, -0.8)
	(0.5, -0.5)

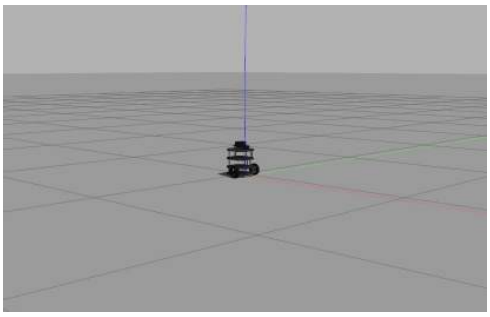
(e) Generalization scenario

Figure B.1: Obstacle-free target configurations.

Table B.1: Simulation parameters.

Parameter	Value	Parameter	Value
DQN framework	DQN	State Model	$\{C_{Stack}, d_T, \phi\}$
Action Set (Table 4.2)	Set 2	Reward Set (Table 4.3)	Set 1
Number of Episodes	500	Number of Steps	80
Buffer Size	50000	Learning Rate lr	0.001
Batch Size	256	Discount Factor γ	0.99
Epsilon	1.0	Epsilon Decay (per step)	0.001
Epsilon Final Value	0.01	TargetNet Update Rate	40

Shallow Artificial Neural Network



Gazebo obstacle-free environment

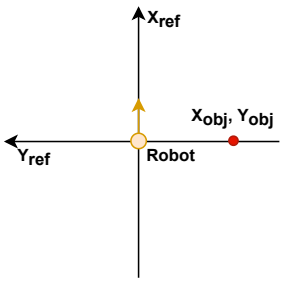
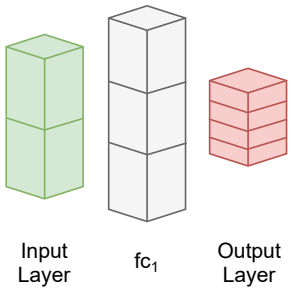


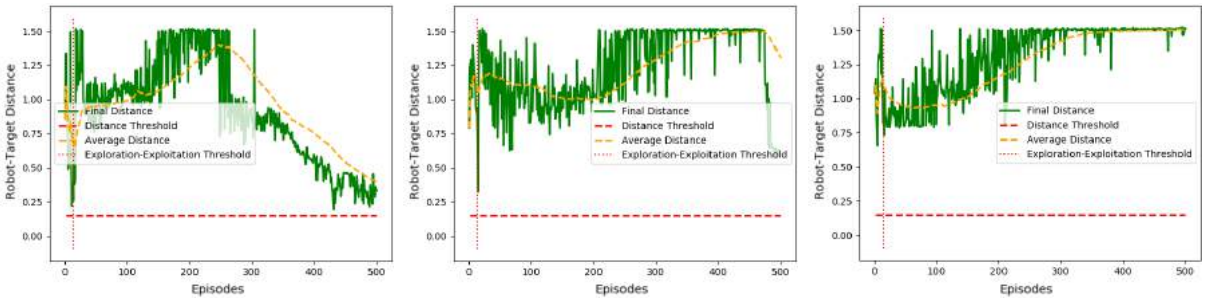
Fig. B.1 (d)
Training and testing scenario

Figure B.2: Fig. B.1 (d) obstacle-free training scenario.

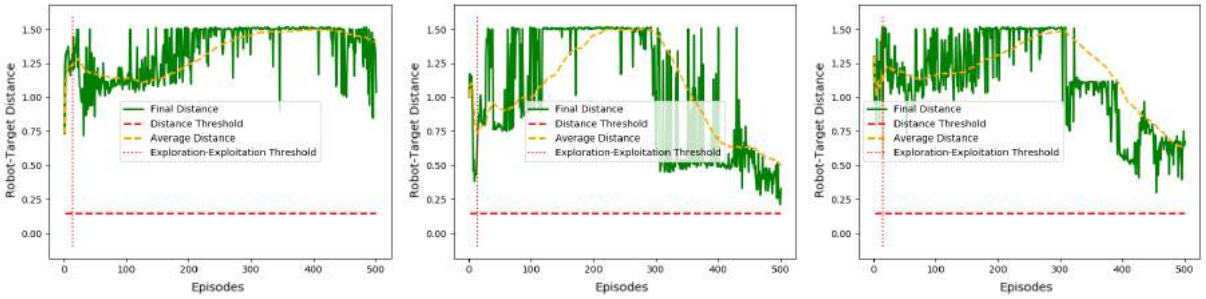


Layer	Parameters
Input	$\{d_T, \phi\}$
fc1	Neurons = Different arrangements (Fig. B.4) Activation = relu
Output	Neurons = Number of Actions

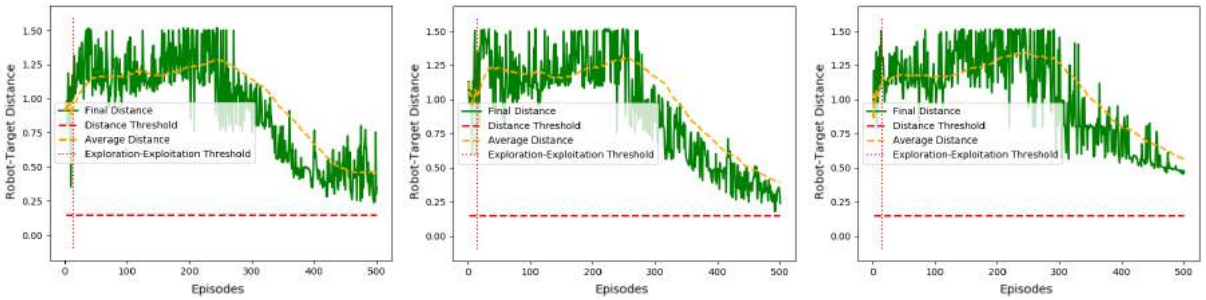
Figure B.3: Shallow ANN and respective layer configuration.



(a) fc1 neurons = 64



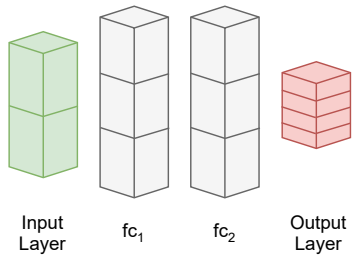
(b) fc1 neurons = 128



(c) fc1 neurons = 256

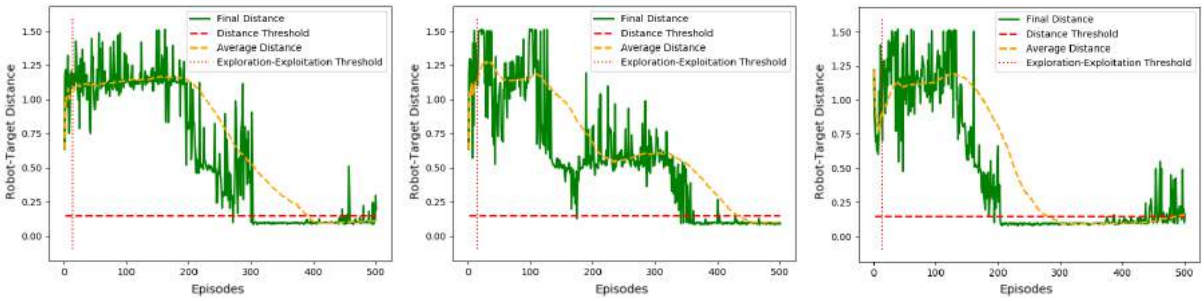
Figure B.4: Training final episode distances from employing the Deep Q-Learning algorithm with the Fig. B.3 Shallow ANN (different number of fc1 neurons) over the Fig. B.1 (d) training scenario. Experiments carried out with the Table B.1 simulation variables.

Deep Artificial Neural Network

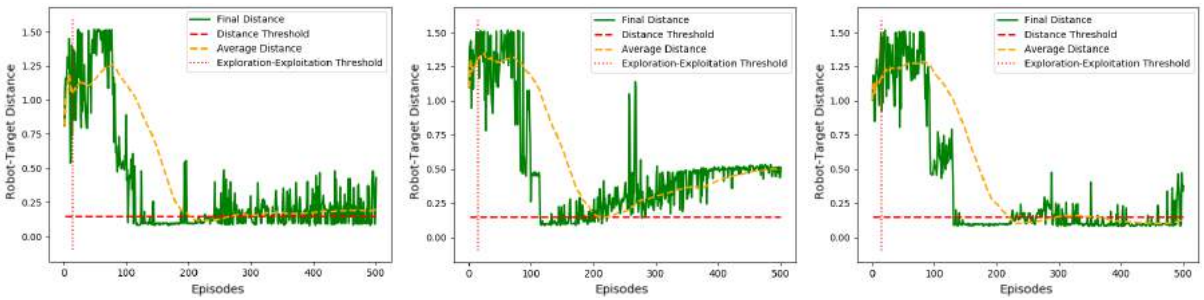


Layer	Parameters
Input	$\{d_T, \phi\}$
fc1 & fc2	Neurons = Different arrangements (Fig. B.6) Activation = relu
Output	Neurons = Number of Actions

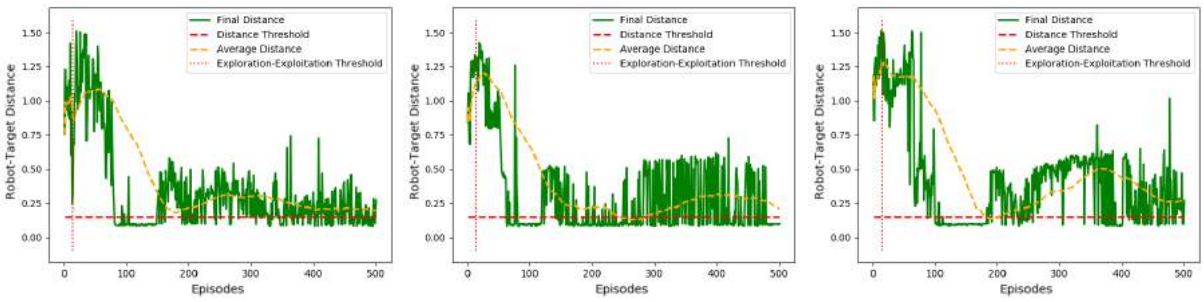
Figure B.5: Deep ANN and respective layer configuration.



(a) Layer neurons: {fc1; fc2} = {64; 64}.



(b) Layer neurons: {fc1; fc2} = {128; 128}.



(c) Layer neurons: {fc1; fc2} = {256; 256}.

Figure B.6: Training final episode distances from employing the Deep Q-Learning algorithm with the Fig. B.5 Deep ANN (different number of fc1 and fc2 neurons) over the Fig. B.1 (d) training scenario. Experiments carried out with the Table B.1 simulation variables.

CNN and Deep ANN Merge-Resultant Deep Q-Network

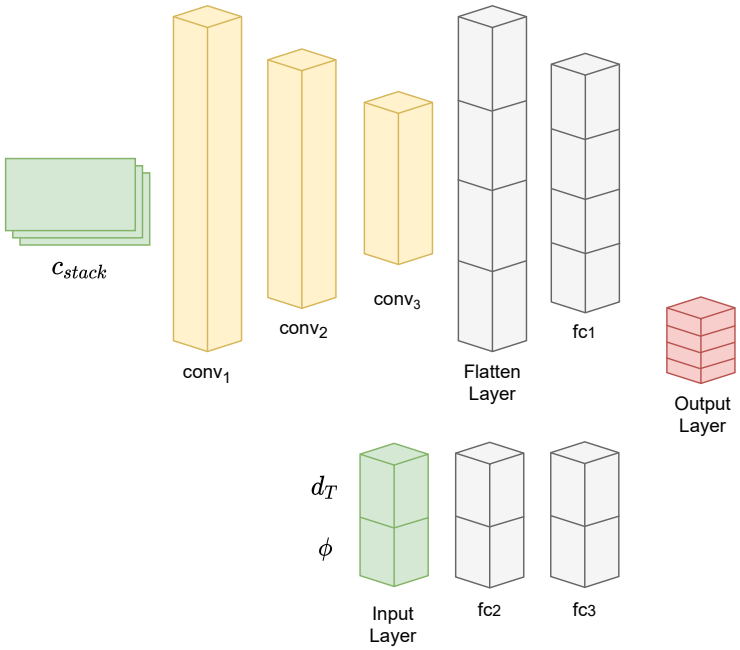
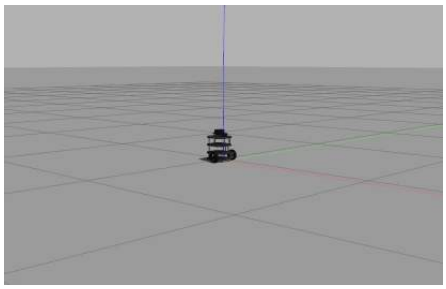


Figure B.7: Proposed Deep Q-Network.

Table B.2: Proposed DQN base layer configurations.

Layer	Parameters
Input	$\{C_{Stack}, d_T, \phi\}$
conv1	Filters = 32, Kernel size = 8, Strides = 4, Activation = relu
conv2	Filters = 64, Kernel size = 4, Strides = 2, Activation = relu
conv3	Filters = 64, Kernel size = 3, Strides = 1, Activation = relu
fc1	Neurons = 256, Activation = relu
fc2	Neurons = 64, Activation = relu
fc3	Neurons = 64, Activation = relu
fcV	Neurons = 1, Activation = None
fcA	Neurons = Number of actions, Activation = relu
Output	Neurons = Number of actions



Gazebo obstacle-free environment

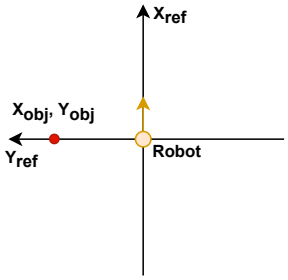


Fig. B.1 (c)
Training scenario

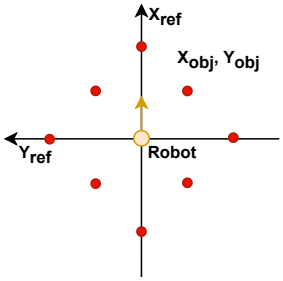


Fig. B.1 (e)
Testing scenario

Figure B.8: Fig. B.1 (c) and Fig. B.1 (e) obstacle-free training and testing scenarios.

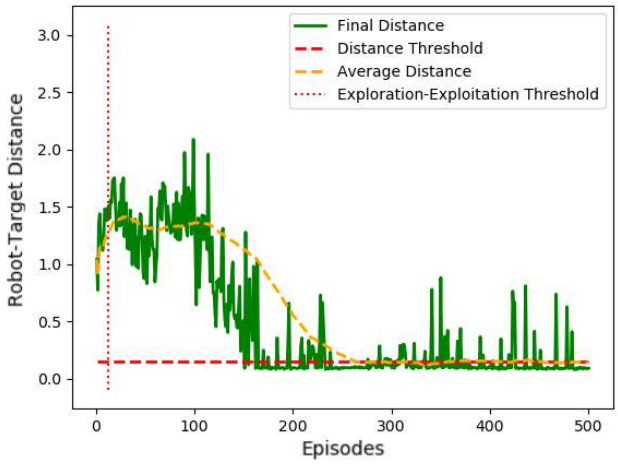


Figure B.9: Training final episode distances from employing the Deep Q-Learning algorithm with the Fig. B.7 proposed ANN over the Fig. B.1 (c) training scenario. Experiments carried out with the Table B.1 simulation variables.

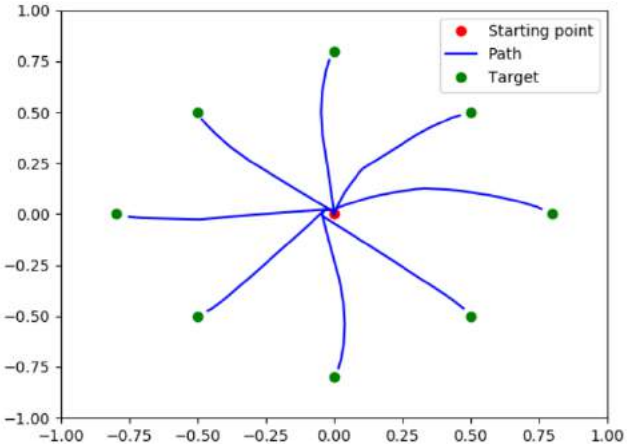
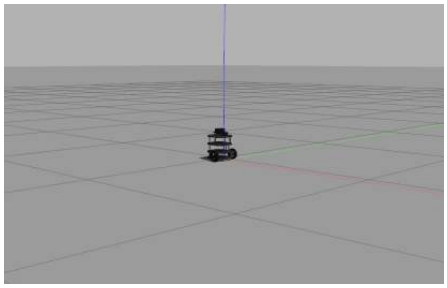
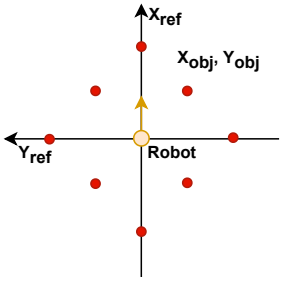


Figure B.10: Online stage robot paths over the Fig. B.1 (e) testing scenario. Agent controlled by a fine-tuned model saved from Fig. B.9 training.





Initial Position (X,Y)	Targets (X_{obj},Y_{obj})
(0.0, 0.0)	(0.8, 0.0)
	(0.5, 5.0)
	(0.0, 0.8)
	(-0.5, 0.5)
	(-0.8, 0.0)
	(-0.5, -0.5)
	(0.0, -0.8)
	(0.5, -0.5)

Fig. B.1 (e)

Figure B.11: Fig. B.1 (e) obstacle-free training and testing scenario.

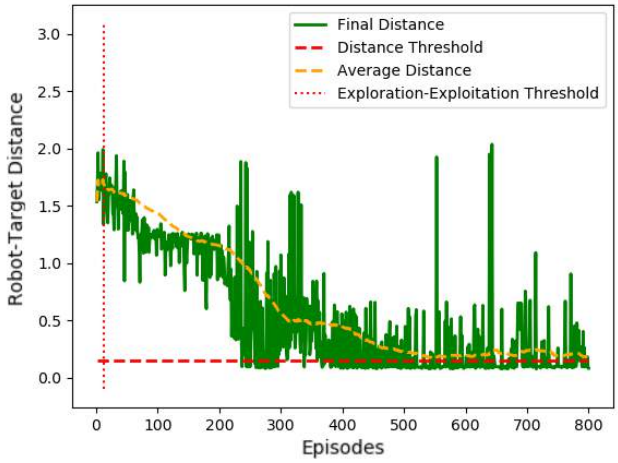


Figure B.12: Training final episode distances from employing the Deep Q-Learning algorithm with the Fig. B.7 proposed ANN over the Fig. B.1 (e) training scenario. Experiments carried out with the Table B.1 simulation variables.

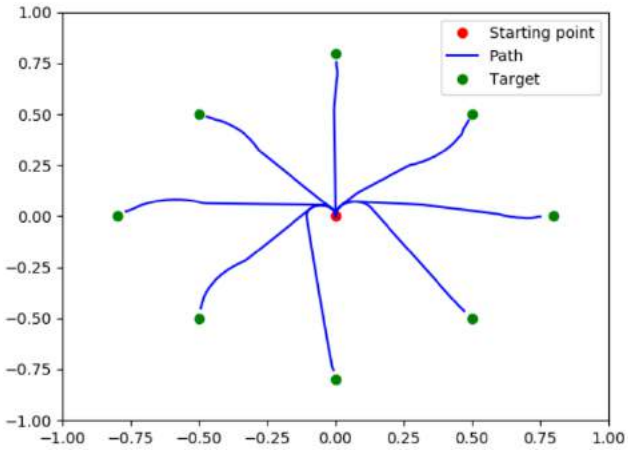


Figure B.13: Online stage robot paths over the Fig. B.1 (e) testing scenario. Agent controlled by a fine-tuned model saved from Fig. B.12 training.