Palestine Polytechnic University

College of Information Technology And Computer Engineering

Graduation Project

Deep Reinforcement Learning - based Rotary Inverted Pendulum

Project Team

Qusai Hroub

Project Supervisor

Dr. Hashem Tamimi

Hebron - Palestine

2020 - 2021

Acknowledgement

I would like to express my special thanks to my supervisor Dr. Hashem Tamimi. Their valuable guidance and feedback helped me in completing this project. In addition, I would like to thank my family for their continuous support and my colleagues who have always been there with me. Last but not the least, I would like to thank all the people who helped, supported and encouraged me to successfully finish this project.

Abstract

Artificial Intelligence systems have become a very important term in many engineering fields. Intelligent robot systems are an example of Intelligence systems (IS). To develop an IS, we need Intelligent control methods specially when the dimension of state space gets very huge and it is impossible to cover all states in traditional programming techniques with feasible time.

This project aims to develop an intelligent algorithm based on deep reinforcement learning that can learn from the environment to make the rotary inverted pendulum stable in an upward position. The system is designed to be realized on a Raspberry Pi microcontroller. TensorFlow was used for implementing the deep reinforcement learning algorithm. The simulation result showed that the system was able to learn to balance the pendulum in an upward position that starts from a random state in a proper range.

Table of content

Acknowledgement **1**

Introduction **2**

1.1 Overview 2

1.2 Objectives 3

1.3 The RIP System Description 3

1.4 Problem Statement 4

1.5 Requirement 5

○ User requirement 5

○ System requirement 5

Background **7**

2.1. Machine Learning 7

2.1.1. Reinforcement-Learning Model 8

The Reinforcement-Learning Model Consists 8

Q-Learning 9

Epsilon-Greedy Action Selection 10

2.1.2. Deep Reinforcement-Learning 10

1. Activation Functions 1011

2. Optimizers 11

2.1.3. Major Difference Between RL and DRL 12

2.1.4. How DRL Works 12

2.2. Rotary inverted pendulum 13

2.2.1. Applications 13

2.2.2. Model Convention 13

2.2.3. Mathematical Model 14

2.2.4. Problems with the traditional solution 15

2.3. Literature Review 15

System Design **18**

3.1. Brief Description of the System 18

3.2. Design Options 18

3.2.1 Hardware Design 120

3.2.2 Software Design 2028

3.3. System Diagrams 29

3.3.1. General System Diagram 29
3.3.2. Block Diagram 30
3.3.3. System Schematics 30

Testing and Challenges 33
4.1. Hardware Validation 33
4.1.1 Unit Testing 33
4.1.2 Integration Testing 34
4.2. Software Validation 34
4.2.1 Unit Testing 335
4.3. Challenges 36

Implementation, Experiment, and Results 38
5.1. Software Implementation 38
5.1.1 TensorFlow 38
5.1.2 Pendulum Driver 38
5.1.3 Pendulum Environment 38
5.1.4 Raspberry Pi Software 39
5.1.5 Deep Reinforcement Learning 39
5.2. Hardware Implementation 39
5.2.1 Raspberry pi With Motor Driver 39
5.2.2 Raspberry pi With Motor Rotary Encoder 40
5.2.3 Raspberry pi With Pendulum Rotary Encoder 40
5.3. Experimental Implementation and Setup 40
5.3.1 Environment 40
5.3.2 Parameters 41
5.4. Simulation Results 42
5.4.1 Applying Sudden Change 44

Conclusion and Future Work 47
6.1. Conclusion 47
6.2. Future Work 48
1. Simulation 48
2. Implementation and Scope of the Project. 48

References 49

List of Abbreviations

AVL:	Adelson-Velskii and Landis
CCW:	Counter-Clockwise
CNN:	Convolutional Neural Network
CPR:	Counts Per Revolution
DC:	Direct Current
DRL:	Deep Reinforcement Learning
IS:	Intelligence System
ML:	Machine Learning
NN:	Neural Network
RE:	Rotary Encoder
RIP:	Rotary inverted pendulum
RL:	Reinforcement Learning
SL:	Supervised Learning
USL:	Unsupervised Learning

List of Figures

Fig. 1.1. a) RIP on rotating arm, b) inverted pendulum on a cart 3

Fig. 1.2. RIP System schematic. 4

Fig. 2.1. The Reinforcement-Learning Model 8

Fig. 2.2. The learning process with Q-Learning. 9

Fig. 2.3. DRL architecture 12

Fig. 2.4. Rotary inverted pendulum conventions 13

Fig. 3.1. Q-Table structure 23

Fig. 3.2 Multilayer Perceptron neural network of DQN 25

Fig. 3.3. Convolutional neural network of DQN 26

Fig. 3.4. Module components 27

Fig. 3.5. General System Diagram 29

Fig. 3.6. Block Diagram 30

Fig. 3.7. Block diagram of system schematics 31

Fig. 4.1. Expected behavior for pendulum equation 34

Fig. 4.2. Our module behavior for pendulum equation 35

Fig. 5.1. Pendulum Environment Structure 41

Fig. 5.2. Y-axis - Loss value VS. X-axis - Epoch (step) 42

Fig. 5.3. X-axis - Pendulum angular velocity vs Y-axis Step 43

Fig. 5.4. X-axis - Pendulum angle vs Y-axis Step 43

Fig. 5.5. The effect of a sudden change on the pendulum 45

Fig. 5.6. Render of the simulated pendulum 45

List of Tables

Table 3.1. A state information 21

Table 5.1. State information: First 15 steps for a trained module 44

Chapter One

Introduction

Chapter One

Introduction

1.1 Overview

The Intelligent System (IS) aims to make computers have the ability to simulate human intelligence. IS has many subset fields and applications. One of them is Machine Learning (ML) that makes the systems have the ability to learn by themselves and build their algorithm instructions from experiences without being explicitly programmed. The main focus of the ML on computer development is to make the computer programs have the ability to access data and use it to learn themselves.

ML problems are categorized into three main fields, which are Supervised Learning (SL), UnSupervised Learning (USL), and Reinforcement Learning (RL). The RL brings many advantages to control methods of the IS, one of these advantages is that when the system is very complex and the use of the RL model simplifies the problem. Another advantage that RL brings to IS is that it can balance the tradeoff between exploitation and exploration. This gives the system the ability to exploit past actions or explore unselected actions, which makes more control over the learning process.

In this project, we will use one version of RL which is Deep Reinforcement Learning to make an inverted pendulum balance vertically. Fig. 1.1.a. shows a rotary inverted pendulum (RIP) system which is one of two basic forms of inverted pendulum systems. In this form of the inverted pendulum system, the pendulum is installed on an arm that moves in a circular motion. This rotating arm moves in the horizontal plane to balance the free pendulum that can rotate in the vertical plane. In addition, Fig. 1.1.b shows the second basic form of the inverted pendulum where the pendulum is mounted on the vehicle called cart and it can rotate freely in the vertical plane on the cart where this cart moves in a straight line in the horizontal plane. In this project, we will deal with the first form of the inverted pendulum system which is the Rotary Inverted Pendulum.

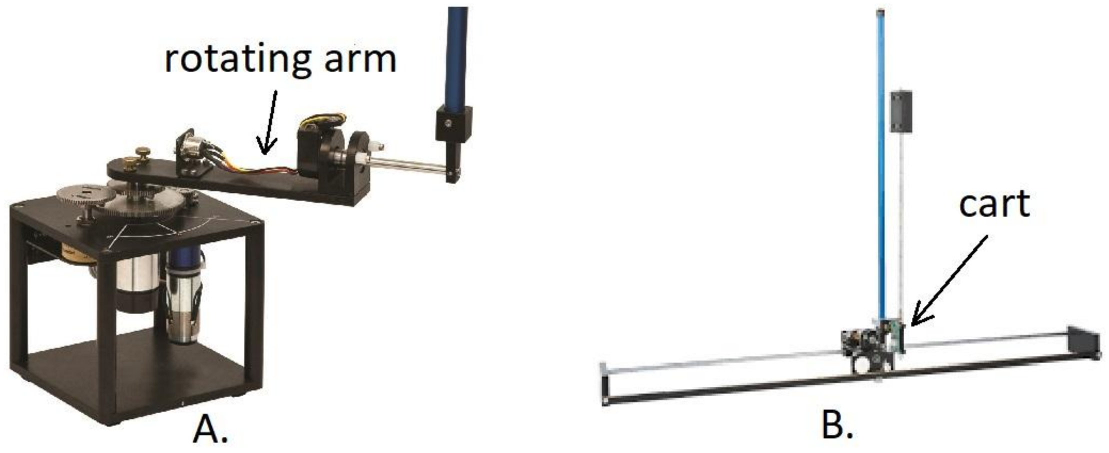

Fig. 1.1. a) RIP on rotating arm, b) inverted pendulum on a cart

1.2 Objectives

In this project, we have three objectives. Which are:

- Design and build a robot model that achieves specifications to balance the RIP.
- Design and implement an intelligent algorithm that can learn itself, using a machine learning paradigm which is RL
- Study the behavior of our solution and obtain experimental results.

1.3 The RIP System Description

The RIP system has a rotary servo motor system that controls an independent gear. The rotating arm of radius **R** is connected to the gear and is attached to the hinge of the pendulum that has length **L**. The arm moves in a circular motion in the horizontal plane and it must move in such a way that the pendulum is balanced in the upright position, where the pendulum can just rotate in a vertical plane to the plane of the arm as shown in Fig. 1.2. Let **α** be the angle of the pendulum to the upright position. and **θ** be the angle of the rotating arm. When the pendulum is moving clockwise the value of **α** is positive and is zero when the pendulum is in an upright position, where the value of **θ** is positive when the arm is moving in a clockwise direction. The symbol **m** represents the mass of the arm.

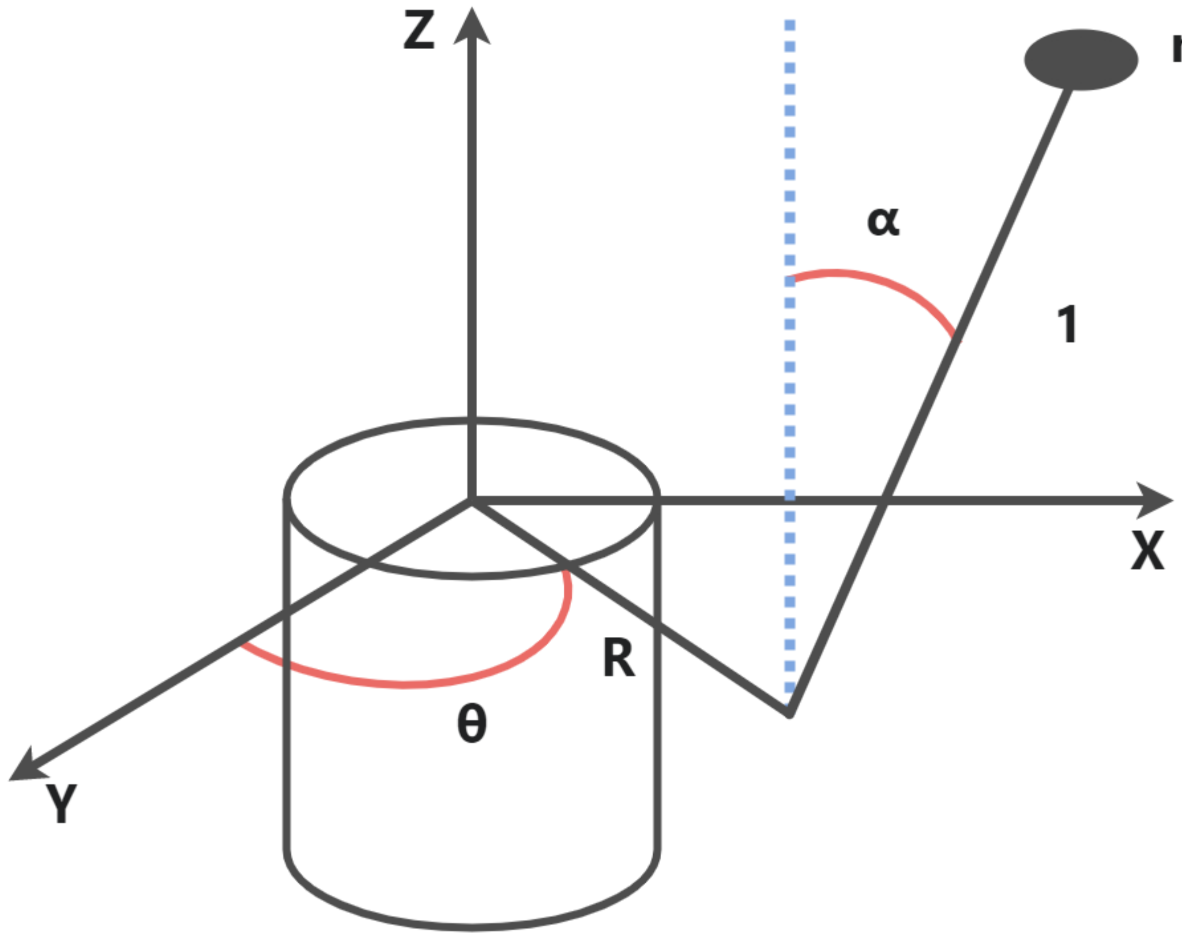

Fig. 1.2. RIP System schematic.

1.4 Problem Statement

RIP is an unstable and highly nonlinear system that has been used as a common application model in the field of nonlinear control engineering. This problem has many solutions using direct control methods but those solutions are not intelligent. In this project, we will perform some research for building a new solution using Reinforcement Learning (RL). Using RL we will build a system to learn by time how to control the pendulum without any help from humans and not based on training previous data.

1.5 Requirement

In this project, we have two types of requirements, which are:

- ### User requirement
  - The user would like to have an interface for controlling the system.
  - The user would like to have an interface to extract the model parameters to be used by identical systems.
- ### System requirement
  - The system needs to read data from sensors.
  - The system needs to be able to balance the pendulum in an upward direction.
  - The system needs to be able to address all sensors at the same time.
  - The system needs a suitable reinforcement learning algorithm.
  - Reliability: The system should be reliable, meaning able to learn how to balance the pendulum every time that the user launches it from a random state.
  - Speed: The system must be able to balance the pendulum in an acceptable response time.

Chapter Two

Background

Chapter Two

Background

In this chapter, we will present some of the information about machine learning, reinforcement learning, and rotary inverted pendulum.

2.1. Machine Learning

Machine Learning (ML) is one branch of IS that makes the systems have the ability to learn for themselves and build their driver or algorithm instructions from experiences without being explicitly programmed. ML gives the computer the ability to develop and change when getting new data. ML is split into three main fields, Supervised Learning (SL): is one of the most basic ML fields, in this field, the ML algorithm is trained on labeled data. By giving input data and output data (labeled data) to the model, through the training process, the model builds relationships and patterns between the input and the output, so the model can estimate the output for new input. For SL, the data must be labeled accurately to work well. Unsupervised Learning (USL): it is the same as SL, but this field has an advantage over the SL. The labeled data is not required. This means that the USL model can work with unlabeled data. The task of this field is to find the similarity, patterns, and differences between the input data then group them into clusters. The goal of USL is to understand the data and categorize it into similar groups (clusters)[1]. Reinforcement Learning (RL): is "an area of machine learning concerned with how software agents ought to take actions in an environment to maximize the notion of cumulative reward while the agent interacts with the environment"[2]. The difference between RL and SL is that it does not need labeled input/output pairs to be presented and also no need for sub-optimal actions to be explicitly corrected, but it focuses on finding a balance between exploration of uncharted territory and exploitation of current knowledge. In this project, we will use RL to solve our problem.

2.1.1. Reinforcement-Learning Model

Fig. 2.1. shows the RL model, an agent interacting with the environment via action and perception to involve the strategy of learning. In RL an episode is defined as a sequence of states, actions, and rewards, which ends with a terminal state. In each episode, the agent receives some information as input, **I**, from the environment, indicates the current state, **s**, of the environment, then generates action, **a**, as output. The action makes an effect on the environment and leads to a change in the agent's environmental state. This change of state translates to the agent through a scalar reinforcement signal, **r**. The agent behavior, **B**, must choose the actions that tend to positively affect the value of the reinforcement signal. The agent learns how to do this over time by using trial and error systematic techniques. The agent views the environment states using an input function. The agent's job is to find the mapping algorithm to map the input states to the output actions. There are many types of RL such as Q-Learning and Deep Q Neural Network (DQN)[3].

The Reinforcement-Learning Model Consists

The RL model has four components.

- A set of environment variables (states) **S**.
- A set of agent actions **A**.
- A set of scalar reinforcement signals **R**.
- An input function **I**.

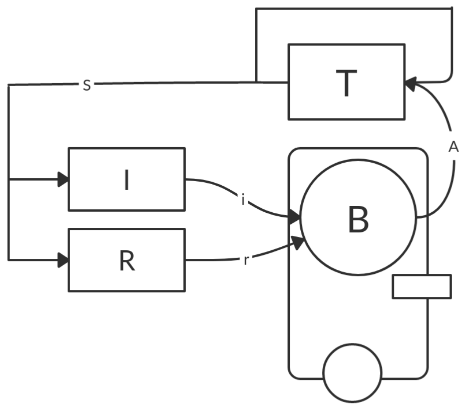

Fig. 2.1. The Reinforcement-Learning Model

Q-Learning

Q-Learning is a type of RL also called the value-iteration method, it learns the action-value function **Q(s, a)**: how good to take any action at a particular state. A scalar value is assigned over an action given the state **S**. In Q-Learning each state-action pair is assigned a Q-value in a table, which represents the sum of reinforcements that are calculated by a Q-value function.

$$Q_{st,at} = Q_{st,at} + \alpha * \left(r_t + \gamma * max\, Q(st + 1, a) - Q_{st,\, at}\right)$$

… (2.1)

Where α is the learning rate, $\mathbf{r_t}$ is the reward, γ is the discount factor, left hand $\mathbf{Q_{st,\, at}}$ is the new value, right hand $\mathbf{Q_{st,\, at}}$ is the current value, and st is the future value estimate. In the episodes, the state-action pairs are created. The episode ends and the new episode starts when the agent reaches the final state. The epochs can bound the training process where the epochs are the number of steps that the agent does in the environment. If the agent doesn't explore some state-action pairs, those pairs will not be recorded in the Q-table and the agent will have no idea how to use or handle them. Fig. 2.2. Shows the learning process with Q-Learning[7].

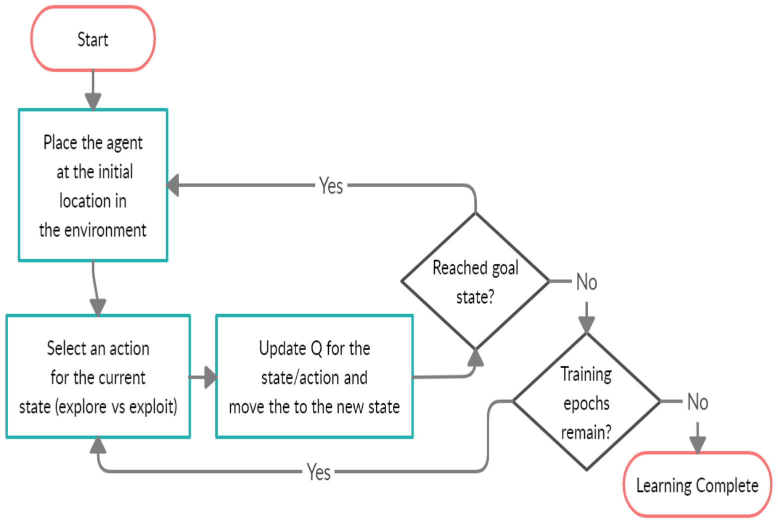
Start
Place the agent at the initial location in the environment
Yes
Reached goal state?
No
Select an action for the current state (explore vs exploit)
Update Q for the state/action and move the to the new state
Training epochs remain?
No
Yes
Learning Complete

Fig. 2.2. The learning process with Q-Learning.

Epsilon-Greedy Action Selection

In Q-Learning the action is selected based on the Q-Value but in epsilon-greedy action selection, the agent uses both exploitations to take advantage of prior knowledge and exploration to look for new options. One of the parameters that Epsilon-Greedy Action Selection has is Epsilon which introduces randomness into the algorithm, forcing it to try different actions. This helps not getting stuck in a local optimum.

2.1.2. Deep Reinforcement-Learning

Recently the RL combined with Deep Learning (DL), DL is a subset of ML where artificial Neural Networks (NN) have multi - deep - layers that enable learning from large amounts of data similar to how humans learn from experience. DL allows the machine to solve very complex problems even when the data set used is very various, unstructured, and inter-connected. RL can solve the problems using a variety of ML methods and techniques, like decision trees, SVMs, NN, and Convolutional Neural Network (CNN). Deep Reinforcement Learning (DRL) is an RL that uses Q-learning with Deep Neural Network (DNN) or Convolutional Neural Network (CNN) function approximation. The motivation behind this is that when the space of the state environments is very big, defining a Q-table would be a very complex process and time-consuming task. Instead of a Q-table, DNN or CNN for each action approximates Q-values based on the state. The advantage of CNN over DNN is that it allows modeling both time and space correlations in multivariate signals. One of the critical parts of the design of a neural network is the activation functions and the other one is the optimizers[18] as explained below.

1. Activation Functions

There are many different types of activation functions used in neural networks and the most used activation functions for the hidden layers are Rectified Linear Activation (ReLU), Logistic (Sigmoid), and Hyperbolic Tangent (Tanh), and for the output layer are Linear and Logistic (Sigmoid)[17].

○ **ReLU**

The most used activation function for hidden layers since it is simple to implement and effective at overcoming the limitations of other previously popular activation functions, such as Sigmoid and Tanh. It has a simple formula which is max(0.0, x). This means if the value is positive, return it as is and if not return 0. It is common to use it in MLP and CNN so we are going to use it in our experiments as activations function for our hidden layers[17].

○ **Logistic**

It takes any real value and returns a value in the range [0, 1]. The returns of larger positive values are close to 1 while the smaller negative values are close to 0. $1.0 / (1.0 + e^{-x})$is the formula of Logistic activation. Where e is Euler's number and x is the input for the activation[17].

○ **Tanh**

It is very close to the Logistic activation function and it has the same S-shape, the difference that the Tanh range starts from -1.$(e^{x} - e^{-x}) / (e^{x} + e^{-x})$ is the equation of the Tanh. In addition, the Tanh activation performs better than the logistic one[17].

○ **Linear**

The linear activation function does not change the input value so it is also called “identity” or “no activation”. (i.e. the input value used as-is for the output)[17].

2. Optimizers

The use of optimizers is to reduce the losses by changing the attributes of the neural network such as weights and learning rate. In this project, we are going to use Adam optimizer. Which is an adaptive learning rate optimization algorithm that’s been designed specifically for training deep neural networks. The reason behind selecting it as an optimizer in our experiments is that it is very fast, converges quickly, and corrects the vanishing learning rate and high contrast[16].

2.1.3. Major Difference Between RL and DRL

The main reason behind developing DRL was to manipulate environments that have uncountable actions and states. The Q-Table-based RL algorithm can be used for small and discrete environments (Environments with countable states and actions). The Q-Table-based RL can still be used for continuous environments by discretizing the states. But when a state in the environment is defined using multiple variables, the Q-Table becomes very large and complex. Due to that, the agent needs more time to explore every state and update the Q-Values, leading to a negative effect on performance, But in DRL, The DNN is used to approximate the Q-Table[19].

2.1.4. How DRL Works

Software agents can learn how to accomplish their goals with the aid of deep reinforcement learning, which integrates artificial neural networks with a reinforcement learning framework. In other words, it combines target optimization and function approximation, linking states and behaviors to the rewards they produce. A value function or a policy function in DRL can be roughly calculated using a neural network. We can train a neural network on samples from the state or action space to learn to predict how valuable those are relative to our target in reinforcement learning rather than using a lookup table to store, index, and update all possible states and their values, which is impossible with very large problems. Fig. 2.3. shows the architecture of the DRL.

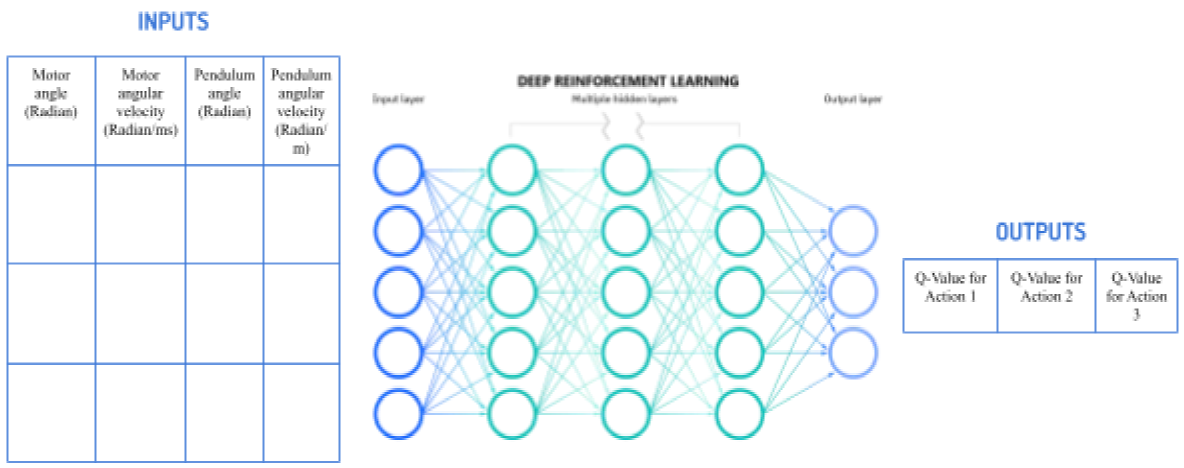

Fig. 2.3. DRL architecture

2.2. Rotary inverted pendulum

In this section, we will provide some of the information about the RIP model and the equations used in modeling it.

2.2.1. Applications

The RIP is a very famous test platform to verify the control theories based on its static instability. The RIP is also used in many real-life applications such as aerospace vehicle control, space rockets, and robotics. It is used as a benchmark tool, to measure the stabilization level of the system (i.e. is the system stable, is the vehicle stable on the road)[4]. It is used in the car's suspension systems. When the system gets unstable, the system modifies itself depending on values from the RIP system.

2.2.2. Model Convention

Fig. 2.3 shows the RIP system model. The rotary arm pivot is attached to the Rotary Servo system and is actuated. The arm has a total radius length of $\mathbf{L_r}$, a moment of inertia of $\mathbf{J_r}$, and its angle, $\boldsymbol{\theta}$, increases positively when it rotates counter-clockwise (CCW). When the control voltage is positive i.e. Vm > 0, the sarvo should turn in the CCW direction. At the end of the rotary arm, the pendulum link is connected. The pendulum has a total length of $\mathbf{L_p}$ and $\mathbf{L_{p2}}$ is the center of mass. $\mathbf{J_p}$ is the moment of inertia about its center of mass. The inverted pendulum angle, $\boldsymbol{\alpha}$. When the pendulum is in the CCW the value of $\boldsymbol{\alpha}$ is positive and is zero when the pendulum is perfectly in an upright position[5].

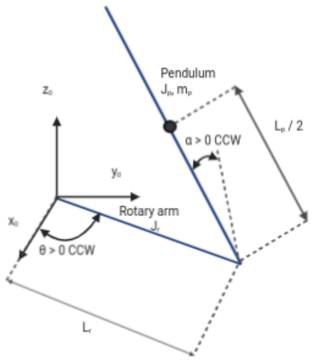

Fig 2.4. Rotary inverted pendulum conventions

2.2.3. Mathematical Model

Fig. 2.3. shows the angular of the pendulum α and the angular displacement of the routing arm θ which are the generalized coordinates for the system[6].

Nonlinear Equations of Motion

There are two equations to describe the motion for the RIP which are:

Equation 2.2 represents the force acting on the rotary arm:

$$\left(m_p L_r^2 + \frac{1}{4} m_p L_p^2 - \frac{1}{4} m_p L_p^2 \cos(\alpha)^2 + J_r\right)\ddot{\theta} - \left(\frac{1}{2} m_p L_p \; L_r \; \cos(\alpha)\right)\ddot{a}$$

$$+ \left(\frac{1}{2} m_p L_p^2 \sin(\alpha)\cos(\alpha)\right)\theta\dot{a} + \left(\frac{1}{2} m_p L_p \; L_r \; \sin(\alpha)\right)\dot{a}^2 = \tau - \beta_r \theta \ldots$$

(2.2)

and Equation 2.3 which represents the force acting on the pendulum:

$$-\left(\frac{1}{2} m_p L_p \; L_r \; \cos(\alpha)\right)\ddot{\theta} + \left(J_p + \frac{1}{4} m_p L_p^2\right)\ddot{a} - \frac{1}{4} m_p L_p^2 \sin(\alpha)\cos(\alpha)\theta^2$$

$$-\left(\frac{1}{2} m_p L_p \; g \sin(\alpha)\right) = -\beta_p \dot{a} \ldots \text{(2.3)}$$

In Fig 2.3, **θ(t)** is the rotary arm angle, and **α(t)** is the inverted pendulum angle, where **ȧ** and **ä** are 1st and 2nd derivative of **α** respectively (i.e. angular velocity and angular acceleration of the pendulum), **θ̇** and **θ̈** are 1st and 2nd derivative of **θ** respectively (i.e. angular velocity and angular acceleration of the rotary arm), β_r is the viscous friction torque (i.e. viscous damping), β_p is the viscous friction torque (i.e. viscous damping coefficient of the pendulum) and **T** is the torque applied at the base of the rotary arm. Equation 2.3. shows that the only force acting on the link is the damping force since the pendulum is not linked directly to the motor. (i.e. the torque just affects the base of the rotary arm).

The torque applied at the base of the rotary arm (i.e. at the load gear) is generated by the servo motor as described by equation 2.3

$$\tau = \frac{n_g k_g n_m k_t \left(V_m - K_g k_m \theta\right)}{R_m} \ldots \ (2.3)$$

Equations 2.2 and 2.3 match the typical form of an equation of motion for a single body. $\tau_1 = J\ddot{x} + b\dot{x} + g(x)$ where **x** is an angular position, **J** is the moment of inertia, **b** is the damping, **g(x)** is the gravitational function, and $\boldsymbol{\tau_1}$ is the applied torque (i.e. scalar value). For a generalized coordinate vector **q**, this can be generalized into the matrix form $\tau = D(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q)$ (2.4) where **D** is the inertial matrix, **C** is the damping matrix, **g(q)** is the gravitational vector, and $\boldsymbol{\tau}$ is the applied torque vector[5].

2.2.4. Problems with the traditional solution

The traditional solution needs to deal with all states in the state space, and for doing such the system must be explicitly programmed to deal with each state. When the state space gets very huge, writing code to handle all of that space can get harder, furthermore at some point it may be impossible. This kind of problem can be encountered using intelligence systems instead of explicit programming, where the IS doesn't need to know all the states in the state space, yet can handle them when appearing. When the system gets in a new state, it should be able to perform an appropriate action from it or learn how to handle it. Using RL will provide us with these specifications, and will allow us to solve the problem mentioned above. The main idea behind RL is to explore the environment and its states, then to learn and inform the agent how it should act, this makes RL-based systems able to deal with the new states by self-learning.

2.3. Literature Review

Nowadays, the researchers have already built different Intelligent modules to control the Inverted Pendulum (i.e Rotary Inverted Pendulum and On Cart version) in experimental environments. Most of those modules are built for the on Cart version using Fuzzy Logic, Reinforcement-Learning, and Deep Reinforcement-Learning. Here are some of the articles and papers that discuss those modules:

1. **Feedback Control of an Inverted Pendulum with the use of Artificial Intelligence[14]**

Is one of those articles that discuss building a solution based on artificial intelligence to control the on cart version of the pendulum. The pendulum is a reasonably simple system, but the physical models all have their peculiarities concerning friction, dead bands, and other nonlinearities. These characteristics are a common cause for obstacles to the control system, which are not always simple to overcome. This article claims that in such a case (i.e. controlling inverted pendulum) the best way to deal with this control problem may be to use an adaptive control method. So they built their solution based on a hybrid control system combining a neural network and genetic algorithm in an attempt to manage these challenges.

2. **Imitation Reinforcement Learning-Based Remote Rotary Inverted Pendulum Control in OpenFlow Network[15]**

This paper discusses building a solution based on artificial intelligence to control the RIP version of the pendulum remotely over the network. According to this study, to control the RIP stably in the control system, many classical and complex mathematical control models were used. Control using classical control engineering models is difficult without a deep understanding of the nonlinear control system. The initial state and the setting of the parameter values used in the control equations in these models are considerably important in achieving successful control. The finding of proper parameters is difficult since some aspects of the atmosphere in nature are seen as chaotic and simple changes in one part of the system cause complex effects everywhere. In this study, they control the RIP device using a deep reinforcement learning algorithm rather than classical control models.

Chapter Three

System Design

Chapter Three

System Design

In this chapter, we will present the detailed design of the system and the RL algorithm. We discuss the different hardware and algorithm alternatives.

3.1. Brief Description of the System

This project is not the first project in its field, there are many solutions to solve the RIP problem, but in this project, the main goal is to build a practical intelligent solution that can learn itself how to stabilize the inverted pendulum, this solution could work effectively, so the second goal is to study our solution result and compare it with other classical non-intelligent solution. To build this project we need some hardware components which are Raspberry Pi, two encoders of a specific type, a motor, a motor driver, a pendulum arm, and the pendulum. In addition, We need some connecting tools like pulleys and the system base. The reason to choose Raspberry Pi over Arduino is that the Raspberry Pi has a slightly more powerful processor. The reason behind choosing two encoders is to use one of them to monitor the motor rotations and the other to monitor the pendulum rotations.

3.2. Design Options

In this section, we will present the hardware and software components.

3.2.1 Hardware Design

In this project we will not design all hardware components from scratch, since some components are designed and well tested and redesign is not a good choice and waste of time (i.e. Redesign Raspberry Pi). The part that we will build from scratch is just the pendulum itself.

Raspberry Pi 4

A Raspberry Pi 4 is a general-purpose computer, provides a set of general-purpose input/output pins that allows you to control electronic components. The main operating system that installs on Raspberry Pi is Linux-based systems i.e. Debian. Raspberry Pi has a slightly powerful processor compared to Arduino that allows us to run multiple programs and control many electronic components simultaneously[8].

Encoder

The encoder in control is “a sensing device that provides feedback”, used to convert the motion to an electrical signal that is used as input for some types of control devices in a motion control system[9]. In our project we need a special type of encoder called Rotary Encoder (RE) which is also called shift encoder, where it is a sensing device that converts angular position or motion of shift or axle to electrical signals - may be an analog or digital signal - as output signals. There are two main types of RE, absolute and incremental. In the absolute type, the output indicates the current shift position. In the incremental type, the output provides information about the motion of shift, which we can translate or process into information such as speed, distance, and position[10]. In this project, we will use an incremental one since it provides more usable and important information i.e. rotation speed.

In this project, we need two encoders, one for senses motor rotation (base encoder) and one for senses pendulum rotation (pendulum encoder). The resolution of the pendulum encoder will be more than the one for the base encoder (500 CPR for the base and 1000 CPR for the pendulum encoder), Higher CPR for the Pendulum encoder is to obtain more accurate values for the pendulum angle and angular velocity since they are more critical than the motor ones. We can increase the resolution by using 2 external interrupt pins for each encoder by using this method the advantage could be 4 times CPR, in other words for a 500 CPR encoder, we could get a resolution of 2000 CPR[11]. The reason behind using RE instead of endless potentiometer is that the endless potentiometer is not available locally.

Brushed Motor

A brushed motor is a DC motor that produces a magnetic field in a wound rotor (the part that rotates) by passing an electrical current through a commutator and carbon brush assembly [12]. We are going to use the Brushed Motor to provide the needed torque to rotate the pendulum arm.

Motor Driver

A Motor Driver is a device that acts as an intermediary between the microcontroller, and motors. A motor driver is necessary because a microcontroller can usually only provide roughly 0.1 Amps of current whereas most actuators (DC motors, DC gear motors, servo motors, etc) require several Amps[13].

3.2.2 Software Design

The final code of RL and to control the encoders and motor driver will be written in C++, which is a general-purpose language. The reason for choosing this language is that it has very high performance and low-level features that allow us to develop our software solution to meet our specifications. While we are in the search stage to design and develop our solution we will use python programming language, since it is easy to use and has many well-tested tools that would be helpful in our project development. To build NN for our solution, We are going to use the TensorFlow library. TensorFlow is an open-source and free software library built for machine learning. We can use this library for many tasks, but it focuses on the training and inference of deep neural networks. So it will be very suitable for our design options, in addition to that, we have experience using it and this will save our time compared to learning new alternatives like Theano. In addition, we are going to use TF-Agents which makes designing, implementing, and testing new RL algorithms easier, by providing well-tested modular components that can be modified and extended. It enables fast code iteration, with good test integration and benchmarking.

Reinforcement Learning (RL) Q-Learning-Based

In this project RIP is our agent for RL. The RL model has four components. These are a set of environment variables (states) **S**, a set of agent actions **A**, a set of scalar reinforcement signals **R**, an input function **I**. In this project, we will define pendulum angle, pendulum angular velocity, motor angle, and motor angular velocity as the set of environment variables **S**, The three values will be determined later by trial and use as the agent action set **A**, and reinforcement signals will be determined for successful step and failed step later as the set of scalar reinforcement signals.

1. State (S)

Table 3.1 shows the time series of state information. This state is delivered from the agent environment. The unit of the state is defined by the set of environment variables. The state information contains four values.

Table 3.1 A state information: to be filled from the training

Motor angle (Radian)	Motor angular velocity (Radian/ms)	Pendulum angle (Radian)	Pendulum angular velocity (Radian/ms)

2. Actions (A) and Reward

The agent selects the action from Q-Table then the information of the selected action that is used to control the motor of the RIP in its environment. The value of three possible motor powers is defined as the agent action set (i.e. [+v, 0, -v]). The absolute value of those possibilities is the force that drives the motor. The negative sign represents the direction of motor motion from right to left while the positive sign represents the direction of motor motion from left to right. The Raspberry Pi converts the value of the selected action into motor voltage values. When an action occurs, the environment changes and the new state is read by encoders then redirects the new state data to the algorithm to take an appropriate reply to that action.

At each step, the reward information is determined by the new state of the pendulum in its environment. If its new state is considered as a failure state the reward value will be negative, and if it is considered as a success the reward value will be positive. The next pseudo-code shows the method of calculating the value of rewards. The state is considered as a success if the pendulum is in an upright range which is ∓7.5 (alpha_threshold_radians) (i.e. the pendulum approximately is in an upward position) at a step. Else, the step has failed. When the step fails the episode comes to end. The final score is set to the sum of the reward values of all steps during the episode when it ends.

```
reinforcementSignal
Begin
    reward = 0.0
    if |current pendulum angle| <= alpha_threshold_radians
        reward = 15.
    if |current pendulum angle| <= (PI / 2)
        reward =  ((PI / 2 -  |current pendulum angle| * 2 / PI) * 10
    else
        reward = -100.

    return reward
End
```

Reinforcement signal (i.e. reward signal)

3. Learning Process

In the learning process, the value or the new value of Q-Value at the new state and selected action is calculated as shown in the next pseudo-code.

```
inser_or_update_and_insert_q_value
begin
    old-q-value = get old value at current state and performed action

    if old-q-value is not exist then
        insert the reward value as it into the Q-Table
    else
    begin
        new-q-value = old-q-value + alpha * ((reward + gama * max-q-value) - old-q-value)
        insert new-q-value into Q-Table
    end
end
```

4. Q-Table

We designed our Q-Table based on the AVL tree. This solution contains 5 layers; the first layer has only one tree and the other 4 layers contain at least 1 tree. The key values for the node in the first 4 layers are from the state vector where these values for the node in the last layer are the indexes of actions in the action space and each node contains its Q-Value. Each node in the tree in any of the first 4 layers points to only one tree in the next layer, Fig 3.1. Shows how layers of the table are structured.

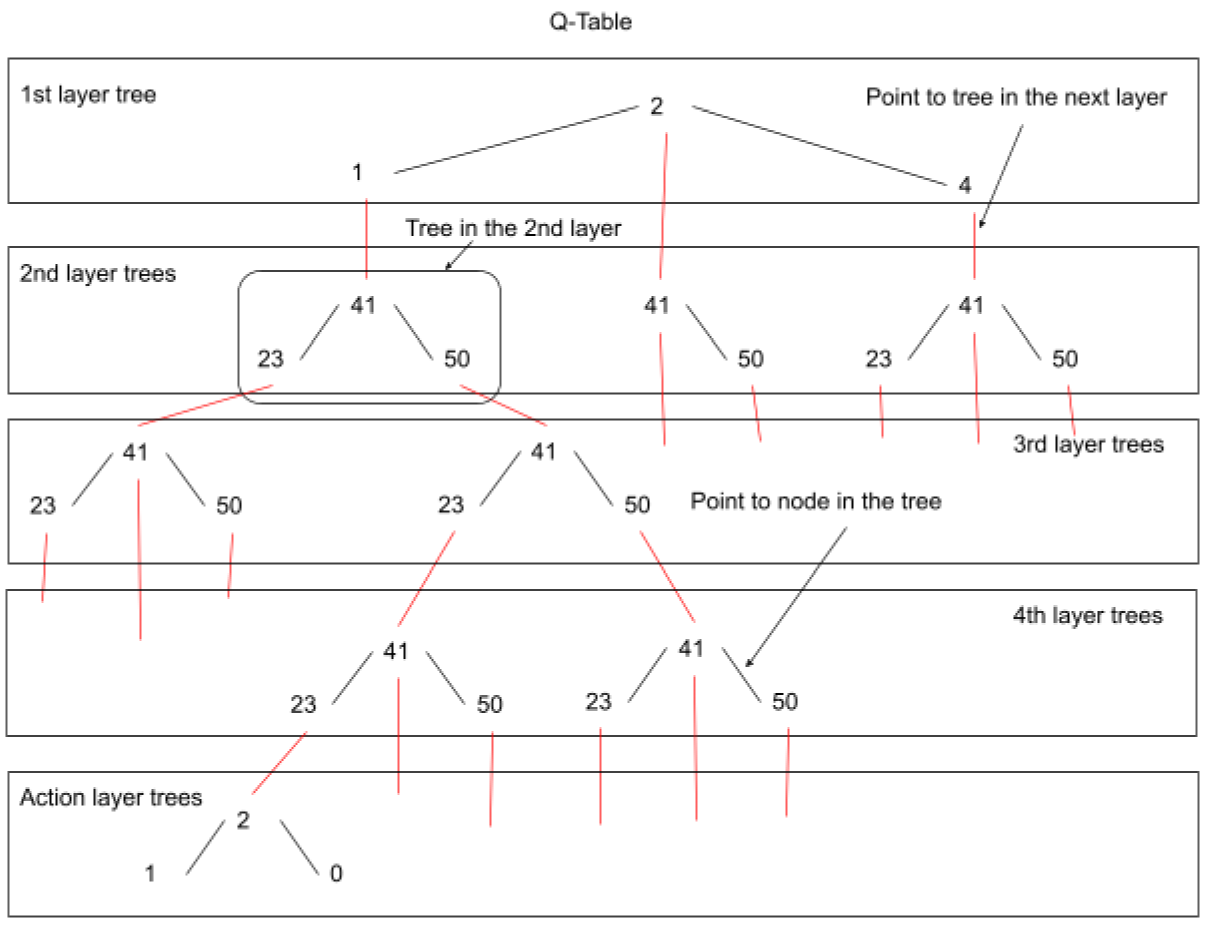

Fig. 3.1. Q-Table structure

5. Action Selection

The next pseudo-code shows how the action is selected based on the Q-Value and random policy.

```
choose_action
begin
        if epsilon is greater than random number then
                Return index of random action

        q-list = q-values on current state
        max-q-value = max value in q-list
        count  = the number of values in q-list are equal to max-q-value

        if count is equal to 1 then
                return the index of max-q-value in q-list

        indexes-of-max-q-value = get all indexes of max-q-value
        return random index from indexes-of-max-q-value
end
```

6. Training

The training process starts from a random state and is divided into episodes and epochs. Each epoch performs one step. In each step action is selected and performed, then the result of the step is used to calculate and update the q-value at the current state and the performed action. Each epoch ends when the maximum number of steps is achieved or when the result of the last performed step is a done-state.

7. Limitations

The state space is very large. As a result, the q-table is very large and complex. It needs a huge amount of memory and building it is very time-consuming. So we decided to move to DRL.

Deep Reinforcement-Learning (DRL)

We designed two models of neural networks, Those models are based on Multilayer Perceptron (MLP) and Convolutional Neural Network (CNN), and to build those models we are going to use different types of layers which are Conv2D, Dense, and Flatten. We are going to use the same design for State, Actions (A) and Reward, and Action Selection as in Reinforcement Learning (RL) Q-Learning-Based design.

1. Multilayer Perceptron Specifications

Fig. 3.2. describes the Multilayer Perceptron network architecture used to constitute the main and target networks, in the reinforcement learning process. The input state is a vector with 4 values. With one color channel, The first hidden layer is fully connected and contains 4 units. The second hidden layer also is fully connected and contains 100 units. The next hidden layer is fully connected and contains 200 units. The last hidden layer contains 50 units and it is also fully connected to the previous layer. The output layer contains just three units and the resulting values for the possible three actions are calculated from the output layer. The rectified linear unit (ReLU) is used as an activation function over all layers.

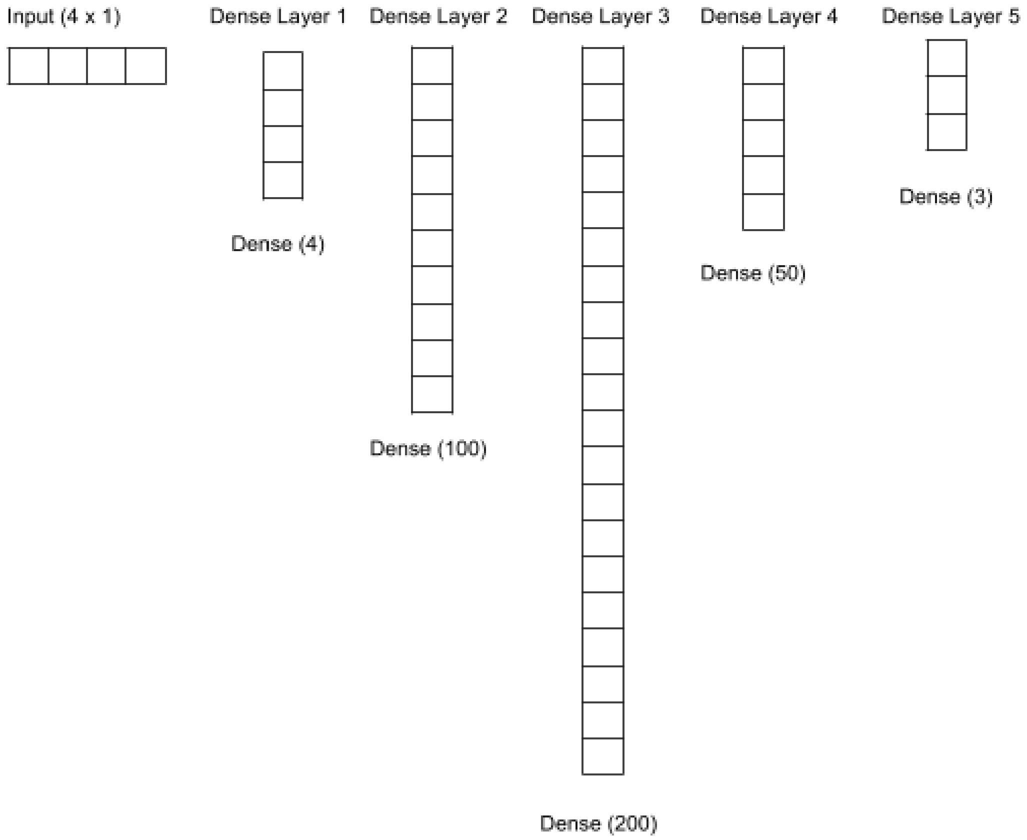

Fig. 3.2. Multilayer Perceptron neural network of DQN.

2. Convolutional Neural Network Specifications

Fig. 3.3. describes the convolutional neural network (CNN) architecture used to constitute the main and target networks in the reinforcement learning process. The input state can be expressed as a 10 × 4 pixel image with one color channel (i.e., gray image). The first hidden layer contains thirty-two 4 × 4 filters and the stride is 1. The second hidden layer contains 64 4×1 filters and the stride is 1. The next hidden layer is fully connected and contains 256 units. The last hidden layer contains 128 units and it is also fully connected to the previous layer. The output layer contains just three units and the resulting values for the possible three actions are calculated from the output layer. The rectified linear unit (ReLU) is used as an activation function over all layers. In this project, we are going to implement this option since it models both time and space correlations in multivariate signals.

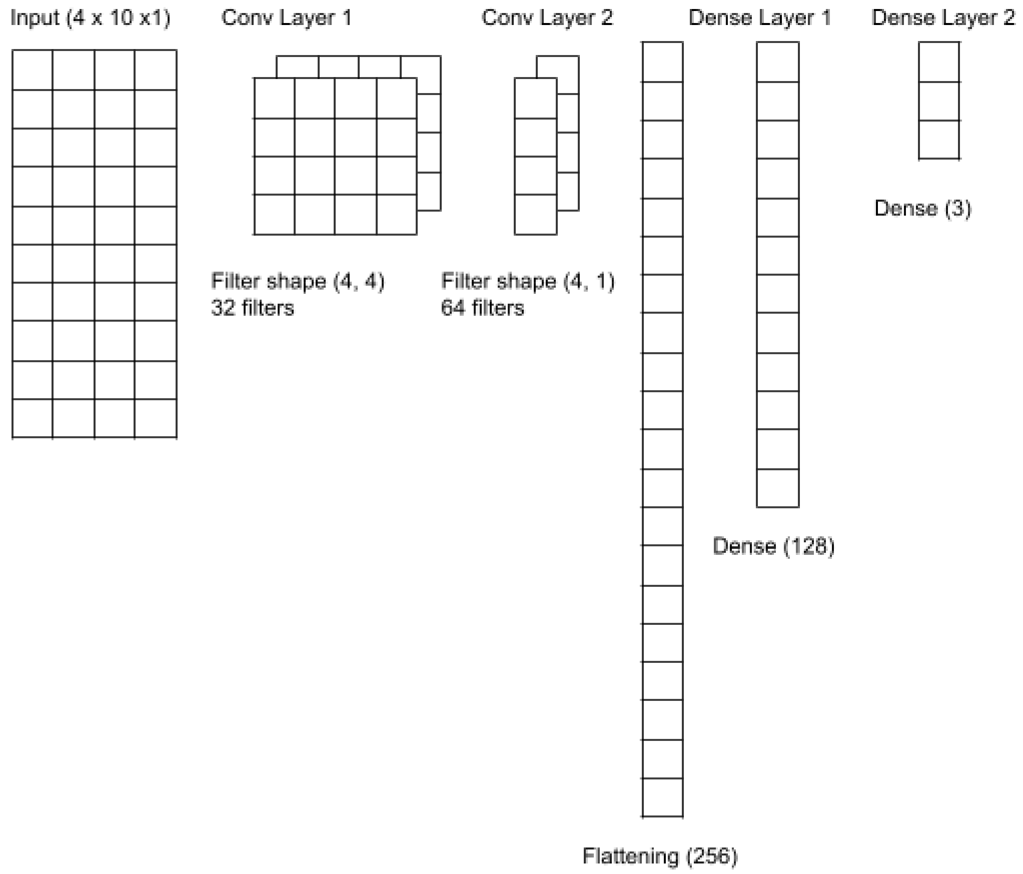

Fig. 3.3. Convolutional neural network of DQN.

3. Training and Module Components

At the beginning of the training process, we collect random data using a random policy to add an initial random data to the replay buffer that is used by the observers. Then we trigger the training function that triggers the collect driver to collect some data from the training environment based on the agent (i.e. q-agent in Fig. 3.4.) collect-policy. Then the train function triggers the next iterator to select a new group of experiences from the observers then send that group to the agent to use in the training process of the qnet-train neural network. After n epochs, the agent copies the weight from qnet-train to qnet-target, the reason behind that is to provide some stabilization of qnet-train weights that are used by collect policy to provide collect-drive with the next action to perform in train-env to collect new data.

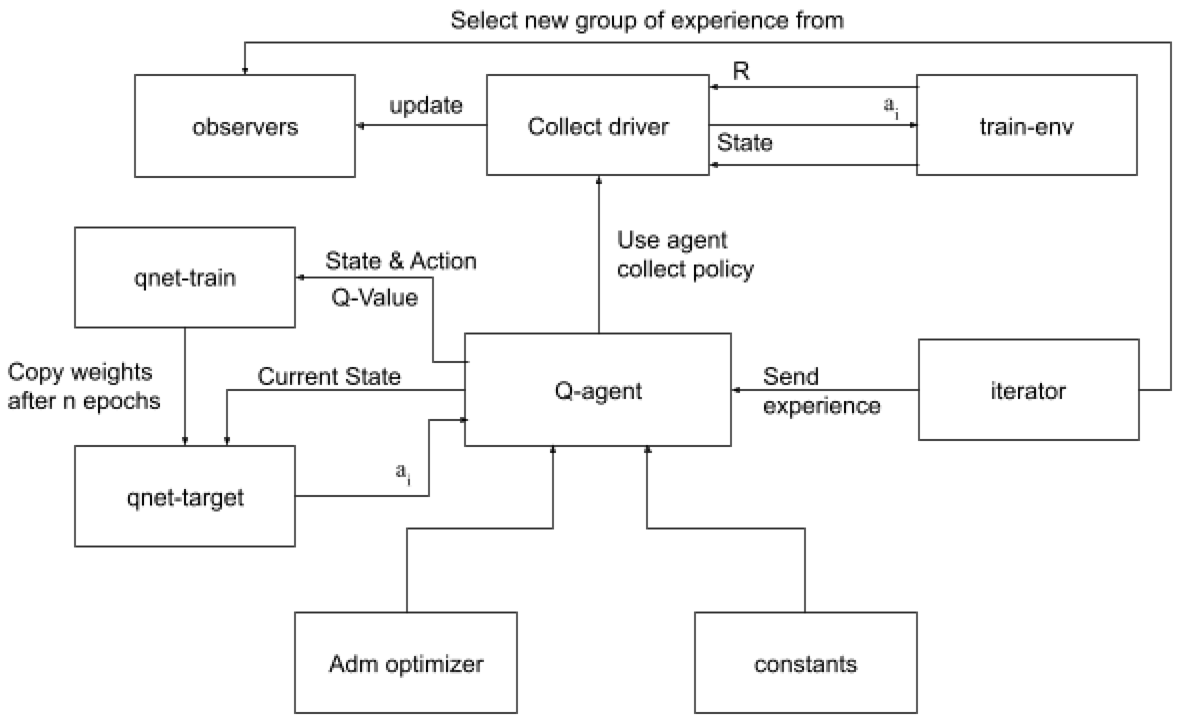

Fig. 3.4. Module components

4. Global Parameters

To control and customize our module, we have many global variables which are:

a. **Number of Epochs**

   Determine the number of steps that the agent does in the environment.

b. **Number of Episodes**

   Number of Epochs that are done as a group (i.e. number of epochs that are done before the Q-agent copying the weights from qnet-train to qnet-target).

c. **Batch Size**

   Is a global parameter that determines the length of the data set that is selected by the iterator.

d. **Initial Collect Steps**

   Is a global parameter that determines the number of steps in the random data collection which is equivalent to the number of states in the random data.

e. **Collect Steps Per Iteration**

   Is used to specify the number of steps performed by the Collect driver.

f. **Learning Rate**

   Is used by Adam optimizer as an initial learning rate, the value of it based on the result of our experiments is 1e-3.

g. **Epsilon**

   Is used by Epsilon-Greedy Action Selection to determine when to select random action or select the new action from the result of qnet-train NN.

5. Evaluation

We are going to use an evaluation environment(an independent environment from the training environment) to evaluate the performance of the agent policy that we are going to use in the deployment process.

3.3. System Diagrams

In this section, we will present a general system diagram of the system, block diagram, and schematic diagram.

3.3.1. General System Diagram

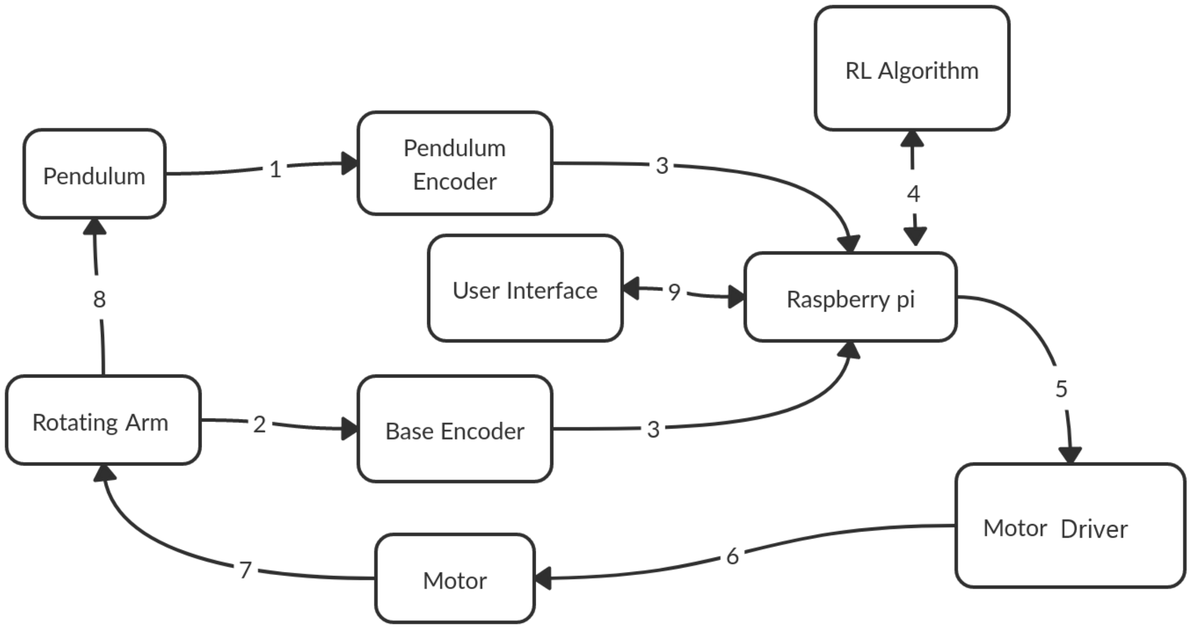

Fig. 3.5. General System Diagram

The general system diagram in Fig 3.5. shows how the hardware components are connected and work each one separately depending on its aim. In addition Fig. 3.5. shows how the whole system components (hardware and software) are connected to do the system work and get good results in balancing the pendulum. The numbers in Fig 3.5. refer to the following:

1. The Pendulum Encoder senses Pendulum rotation then sends the data to the Raspberry Pi.

2. The Base Encoder senses Rotation Arm rotation then sends the data to the Raspberry Pi.
3. Raspberry Pi reads data from The Pendulum and Base Encoders.
4. Raspberry Pi using the RL algorithm processes the data to determine the suitable reaction then sends the result to the Motor Driver.
5. The Motor Driver transmits information into the control signal to control the Motor.
6. The Motor receives the control signal.
7. The Motor rotates The Rotating Arm.
8. The Rotating Arm affects the Pendulum.
9. The User controls the system using its interface (Start, Stop, Extract model parameters.

3.3.2. Block Diagram

Block Diagram in Fig. 3.6. shows how the encoders will transmit the data (Motor rotation data and Pendulum rotation data such as their angle) to the Raspberry Pi and how the Raspberry Pi controls the motor using the motor driver based on the result of processing data in Raspberry Pi to rotate the rotating arm in suitable velocity.

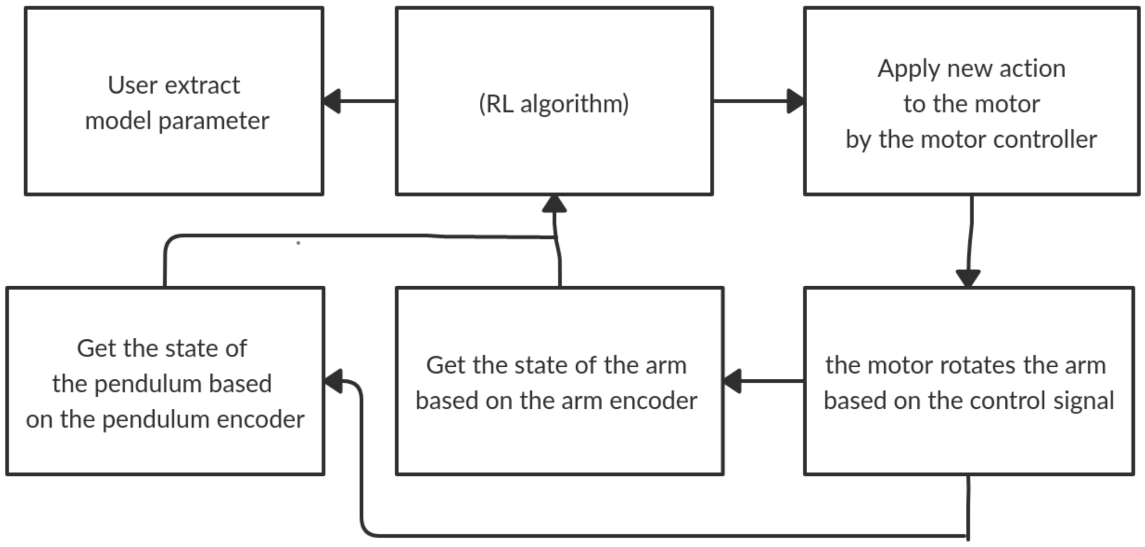
User extract model parameter
(RL algorithm)
Apply new action to the motor by the motor controller
Get the state of the pendulum based on the pendulum encoder
Get the state of the arm based on the arm encoder
the motor rotates the arm based on the control signal

Fig. 3.6. Block Diagram

3.3.3. System Schematics

Fig. 3.7. Shows the schematic diagram, which shows the internal connection of the system components, which are Raspberry Pi 4, two encoders, motor driver, and motor. The motor has two connectors, one of them is connected to output 1 of channel one of the L298N motor driver and the other connected to output 2 of channel one. To control motor torque, the GPIO 18 is connected to the EN line for the motor driver, and to control the direction of motor rotation GPIO 11 and GPIO 8 are connected to IN1 and IN2 for channel one. In addition, the motor driver is connected to the voltage source to provide a suitable current to run the motor. Two encoders are connected directly to the Raspberry Pi, Where the SIG A of the motor encoder is connected to GPIO 17, and SIG B is connected to GPIO 27 and The SIG A of the pendulum encoder is connected to GPIO 6 and SIG B is connected to GPIO 5.

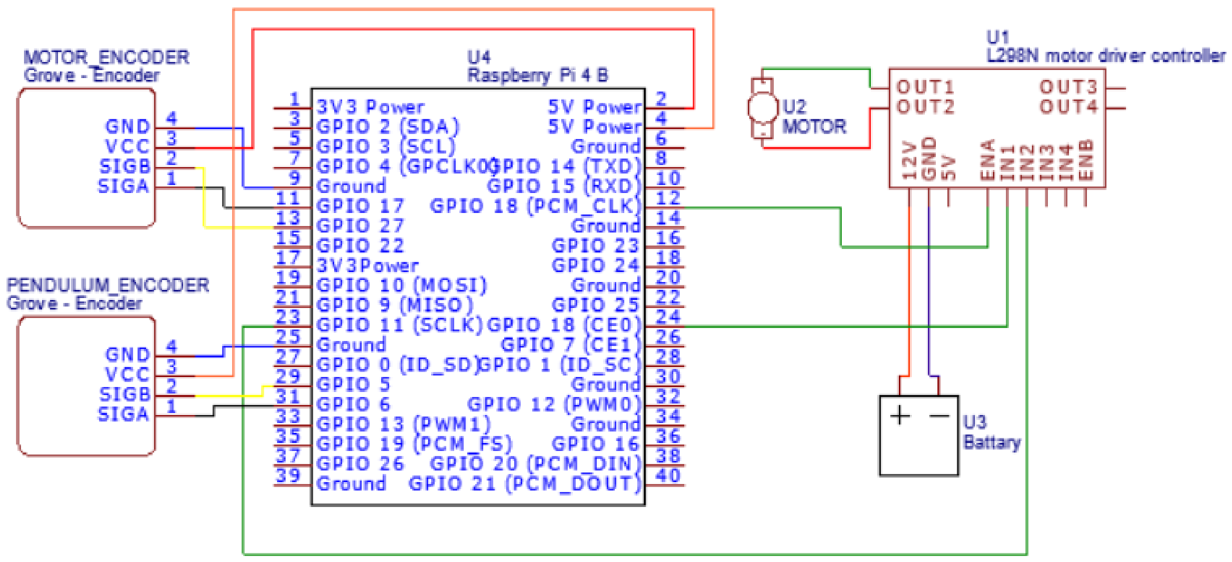

Fig. 3.7. Block diagram of system scheme

Chapter Four

Testing and Challenges

Chapter Four

Testing and Challenges

In this chapter, We will describe and discuss the steps, and results of the testing of the system and the challenges we faced.

4.1. Hardware Validation

In this section, We will present the steps of hardware validation.

4.1.1 Unit Testing

We started by testing all the parts separately, to ensure that all of the functions work perfectly and without errors.

1. Testing the working of Raspberry Pi 4.

   We install an operating system in a micro SD card then we install this card into its place in Raspberry Pi, then we connect the Raspberry Pi to a voltage source. After it completes the booting process we test the working of USB ports by connecting a working USB stick and GPIO pins by trying to light a LED - Passed.

2. Testing the working of the motor.

   We connected the motor to the voltage source and tested if it is rotating or not - Passed.

3. Testing the working of the motor encoder.

   We connected the encoder to Raspberry Pi and tried to read the angle - Passed.

4. Testing the working of the pendulum encoder.

   We connected the encoder to Raspberry Pi and tried to read the angle - Failed.

   So we replaced it and tried to use a different type, but it failed again. In our thought, the reason behind that is that encoders are very sensitive and our hardware was not that stable. In the end, due to time limitations and local availability, we decided not to complete the hardware part.

4.1.2 Integration Testing

Then We test all components (Raspberry Pi, motor, motor encoder, analog to digital converter, potentiometer) connected.

4.2. Software Validation

In this section, We will present the steps of software validation.

4.2.1 Unit Testing

We started by test all parts as following (all unit testing are passed successfully):

1. **Pendulum Model**

   a. Testing the working of mathematical equations, Fig 4.1. shows the expected behavior, where Fig 4.2 shows our module behavior. Where Y-axis represents the values of motor angle and pendulum angle and motor angular velocity and pendulum angular velocity, where the X-axis represents the time.

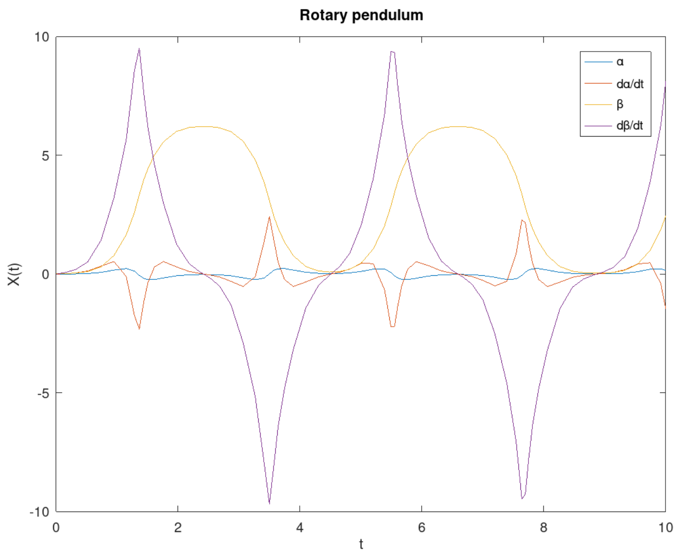

Fig 4.1Expected behavior for pendulum equation (α - motor angle, dα/dt - motor angular velocity, β - pendulum angle, dβ/dt - pendulum angular velocity)

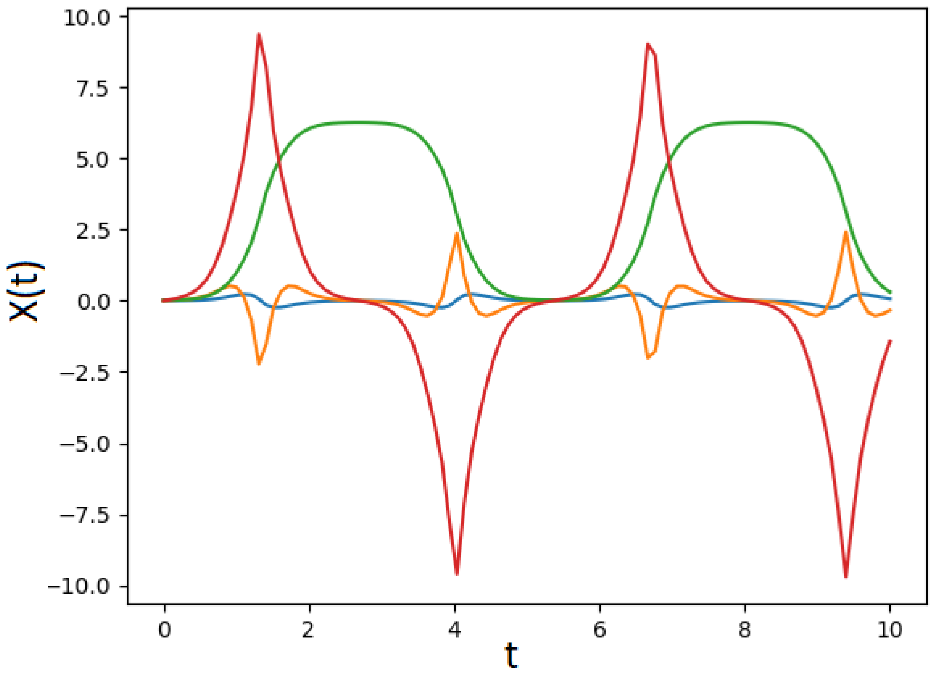

Fig 4.2 Our module behavior for pendulum equation (blue - motor angle, orange - motor angular velocity, green - pendulum angle, red - pendulum angular velocity)

b. Testing the working of step-function.

We called the step-function and tested if it performed the selected action or not by printing the value of the performed action.

c. Testing the working of util functions.

We tested the working of util functions by calling the functions for specific values of their parameters and comparing the results by the expected one.

2. Deep Reinforcement Learning

a. Testing the working of building the training and evaluation environment from our custom environment.

We called the load function from suite_gym and sent the name of our custom environment, waiting to see if the suite_gym would load it without errors or not. The result was that the suite_gym was able to load our environment without errors.

4.3. Challenges

The challenges were we faced to get some hardware components of the RIP, train our algorithm needs highly capable hardware which was unavailable, so we used cloud solution from Google, We face some challenges with controlling Raspberry Pi GPIO using C++, it was hard to find a good library that supports Raspberry Pi4 and works as expected with C++ with acceptable performance, Numerous failures of some hardware components , like encoders. Modeling the RIP was a great challenge since it needed a good amount of knowledge that was out of my knowledge.

Chapter Five

Implementation, Experiment, and Results

Chapter Five

Implementation, Experiment, and Results

In this chapter, We will describe the implementation of the software and the hardware parts of the system, the experimental setup, and results.

5.1. Software Implementation

In this section, We will present the detailed design and implementation of the software part.

5.1.1 TensorFlow

We start with installing TensorFlow using the pip package manager. We installed TensorFlow 2.4. since it was compatible with other packages that were previously installed like NumPy. Then we installed TF-Agents and selected dqn_agent and other vital packages like dynamic collect driver which we will use to collect stats from the environment.

5.1.2 Pendulum Driver

We build a software drive using C++ to provide the basic functionality to control the motor (perform an action) and monitor the motor angle and angular velocity, and the pendulum angle and angular velocity. In addition, we built an Application Programming Interface (API) to use this drive from python3 to use it in the pendulum environment.

5.1.3 Pendulum Environment

It is a sim-carbon copy of the simulation pendulum environment that will be described in Experimental Implementation and Setup; the only difference is that instead of using a mathematical module to simulate the pendulum behavior, it uses the Pendulum driver to communicate with the physical environment and perform an action when a step function is called. In addition, it shows a simple render that shows the pendulum state in the physical environment.

5.1.4 Raspberry Pi Software

We started with installing the Raspbian OS image from their website, then installed that image into an SD card and mounted it to our Raspberry Pi, after that, we booted into the system then installed the development tools we need such as IDE, C++ compiler, and the interpreter of python.

5.1.5 Deep Reinforcement Learning

We used TensorFlow to build the NN described in Chapter three. To do that, we implement util functions. The first util function was to construct a Dense layer using the Dense class from TensorFlow Library. This function takes one parameter, which is the number of neurons (units) in the layer. The second function was to build a convolutional layer. This function takes a tuple of 3 values, which are filters which is the number of filters in the layer, input_shape which is a tuple of 3 numbers that define the dimensions of the input to that layer, and kernel_size which is a tuple of 2 numbers that define the kernel dimensions. The final function that we built to use in building the NN is to build the whole NN using the two previous functions, this function takes two parameters, which are num_actions which is the number of actions and it is equivalent to the number of neural network outputs, the second parameter is a tuple that describes CNN layers. In addition to that, we build Training and Module Components as described in chapter three.

5.2. Hardware Implementation

In this section, We will present the detailed design and implementation of the hardware part.

5.2.1 Raspberry pi With Motor Driver

We connected the Raspberry Pi 4 with the motor driver to control the motor as it described in Fig 3.7. That shows the GPIO 18 is connected to the EN line for the motor driver, and to control the direction of motor rotation GPIO 11 and GPIO 8 are connected to IN1 and IN2 for channel one. In addition, we connected the motor driver to the voltage source to provide a suitable current to run the motor.

5.2.2 Raspberry pi With Motor Rotary Encoder

We connected the Raspberry Pi 4 with the rotary encoder to read the motor angle and angular velocity as described in Fig 3.7.

5.2.3 Raspberry pi With Pendulum Rotary Encoder

We started with connecting the Raspberry Ri 4 with the rotary encoder to read the pendulum angle and angular velocity as described in Fig 3.7., but we were unable to read any signal so we were unable to complete the hardware implementation as described in Chapter four.

5.3. Experimental Implementation and Setup

In this section, We will describe the experimental environment, type of devices, and experimental parameters.

5.3.1 Environment

In the Environment subsection, We will describe the two environments that we used to do experiments.

- **Physical Environment**

  For our physical environment, we used RE of type (E6B2-CWZ6C) to monitor the pendulum angle and angular velocity. To provide the torque we used a long shaft motor of type (MG513IPH) and its internal encoder to monitor its angle and angular velocity. The effective length of the pendulum rod is 135mm and its weight is 140g and the effective length of the rotating arm is 156mm and its weight is 140g, the total height when the pendulum is in the target state (i.e. the rod is in upward positions) is 297mm.

- **Simulation**

  For early experiments, we built a custom gym environment using Python3 programming language and gym library to simulate the RIP environment. We followed the structure of Gym environments. Fig. 5.1. shows the structure of our virtual environment.

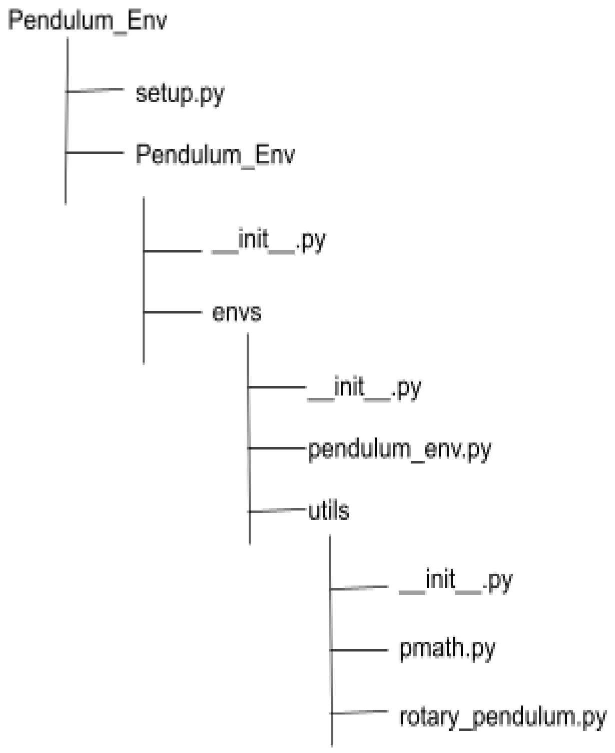

Fig. 5.1. Pendulum Environment Structure

The file pendulum_env.py contains the class that presents the RIP and its environment while the file rotary_pendulum.py contains the mathematical equation that is used to resolve the pendulum state in its environment and the file pmath.py contains the math util functions. To solve the pendulum mathematical model we used Ordinary Differential Equations (ODE) with dopri5 as integrator which is an explicit Runge-Kutta method of order (4)5 due to Dormand & Prince (with step size-control and dense output).

5.3.2 Parameters

In our experiments, the value of Initial Collect Steps and Batch Size was 512. Initial Collect Steps and Batch Size are equal since we need to have a suitable amount of data to run the first episode. The Number of Epochs was 40000 and The Number of Episodes was 200 to give the agent a chance to explore its environment before restarting from a new random state. In addition, The value of "Collect Steps Per Iteration" is kept as one in all of our experiments since increasing its value will rapidly increase the time taken by the training process and the value of Learning Rate is kept as 1e-3, which is the default value for Adam optimizer. The Epsilon value was 0.2, and we chose this value by trial and error. In addition, the discount factor for future rewards was 0.85. Finally, the set of actions was {-0.6, 0, 0.6} .

5.4. Simulation Results

In this section, we will show the results for the last solution (i.e. solution based on CNN) in the simulated environment.

Fig. 5.2. shows losses value at each epoch. Where the max loss value at the beginning of the training is almost 1300, after about 4000 epoch the losses delta get smaller and smaller, the loss values in the range [0, 4000] decrease rapidly, while in the range [15000, 4000] the change is smaller than in the first 4000 epochs. The change in the loss curve is not smooth because the agent explores new states.

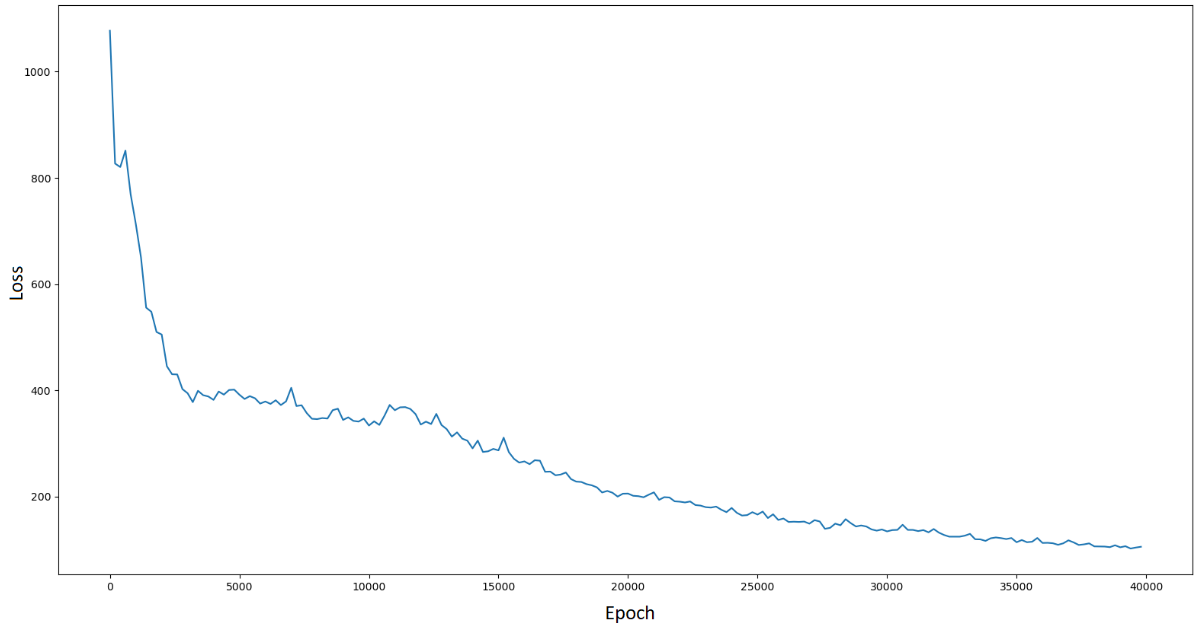

Fig. 5.2. Y-axis - Loss value VS. X-axis - Epoch (step)

Fig. 5.3. shows the change in the pendulum angular velocity and Fig. 5.4. shows the change in the pendulum angle in the first 400 steps after the training is done (i.e. in the first 2400 milliseconds), on the other hand, Table 5.1. shows the values for the first 15 steps (i.e. first 90 milliseconds).

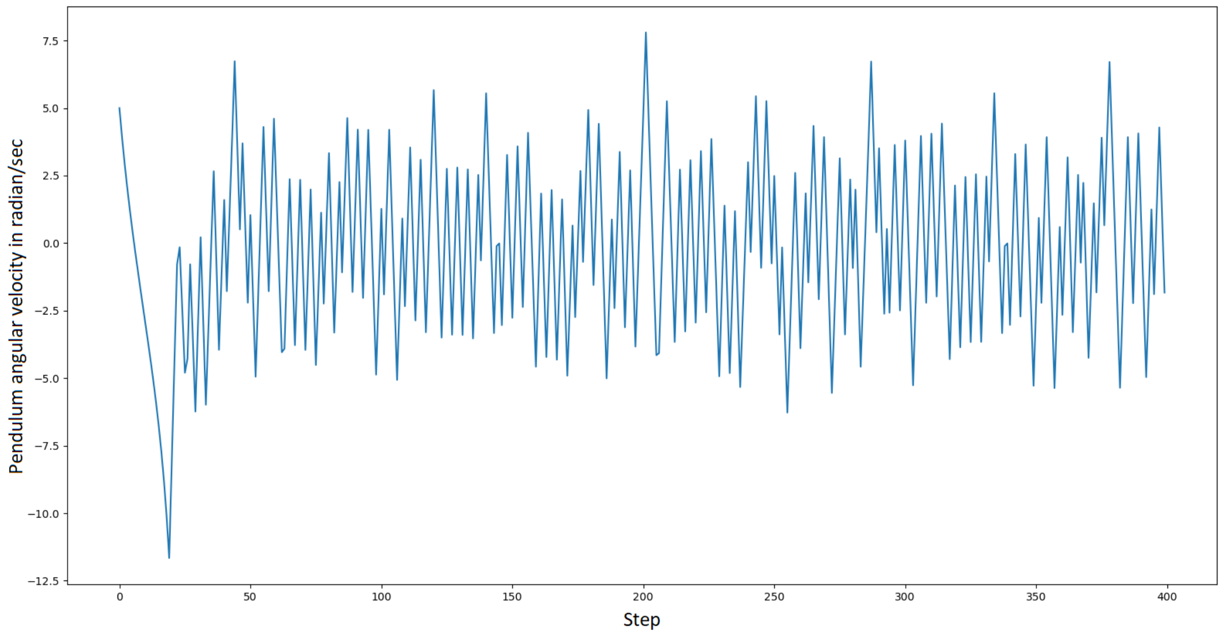

Fig. 5.3. Y-axis - Pendulum angular velocity vs X-axis Step

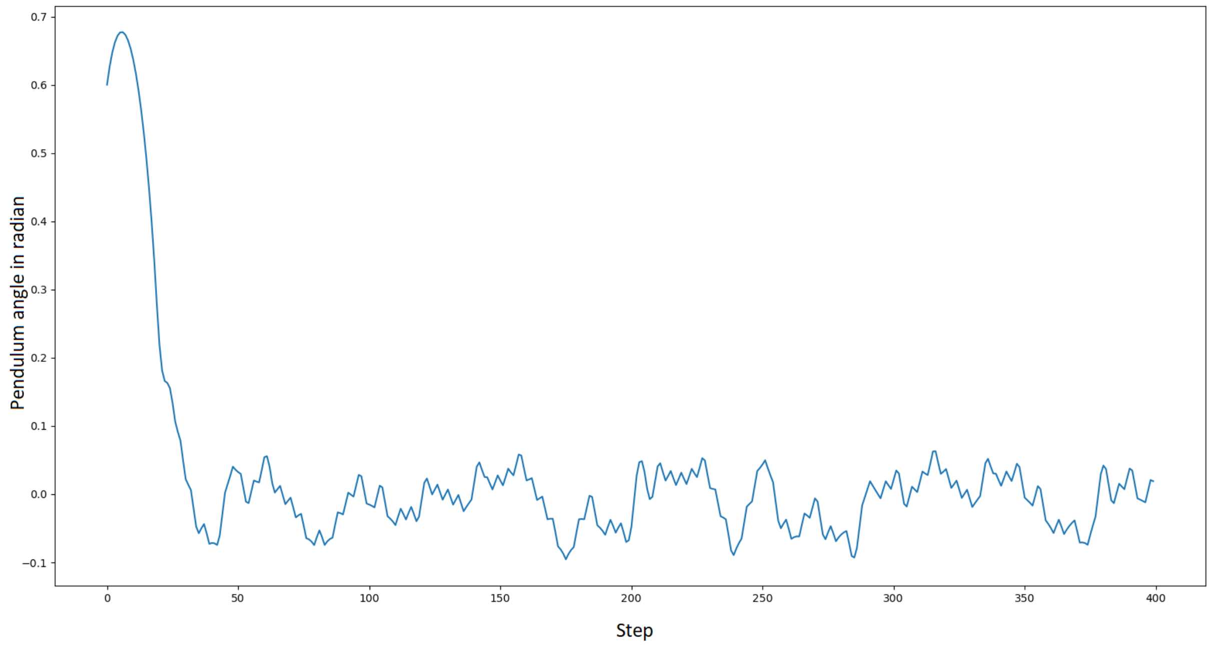

Fig. 5.4. Y-axis - Pendulum angle vs X-axis Step

Table 5.1. State information: First 15 steps for a trained module

Motor angle (Radian)	Motor angular velocity (Radian/ms)	Pendulum angle (Radian)	Pendulum angular velocity (Radian/ms)
0.00	0.00	0.60	5.00
-0.00	-1.03	0.63	3.91
-0.01	-2.02	0.65	2.93
-0.03	-2.97	0.66	2.04
-0.05	-3.90	0.67	1.21
-0.07	-4.82	0.68	0.44
-0.11	-5.73	0.68	-0.30
-0.14	-6.64	0.67	-1.01
-0.19	-7.55	0.67	-1.69
-0.23	-8.47	0.65	-2.36
-0.29	-9.40	0.64	-3.04
-0.35	-10.36	0.61	-3.72
-0.41	-11.34	0.59	-4.41
-0.48	-12.35	0.56	-5.14
-0.56	-13.40	0.53	-5.92

5.4.1 Applying Sudden Change

In our experiments, in a simulated environment, we defined a sudden change as an unspecified action that the motor took before the q-agent took its decision and performed it in its environment. Our module can start from any random state in the range [-0.6, 0.6] radians for pendulum angle and [-10, 10] radians/sec for pendulum angular velocity and get the pendulum in an upward position. In case the sudden change affects the state of the pendulum. Having that range will give it a chance to recover and return to a stable state. Fig. 5.5. shows how the pendulum recovered from a sudden change effect at step 200.

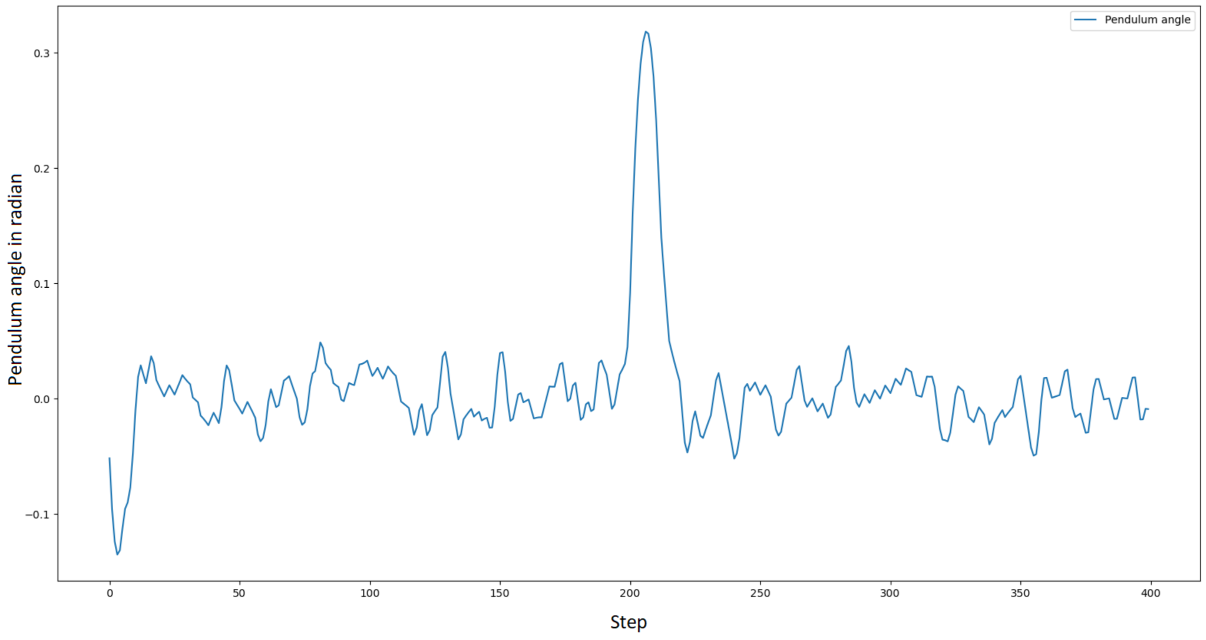

Fig. 5.5. The effect of a sudden change on the pendulum

Fig. 5.6 shows the render of the simulated pendulum beside the plot for the pendulum and motor angles (rod state).

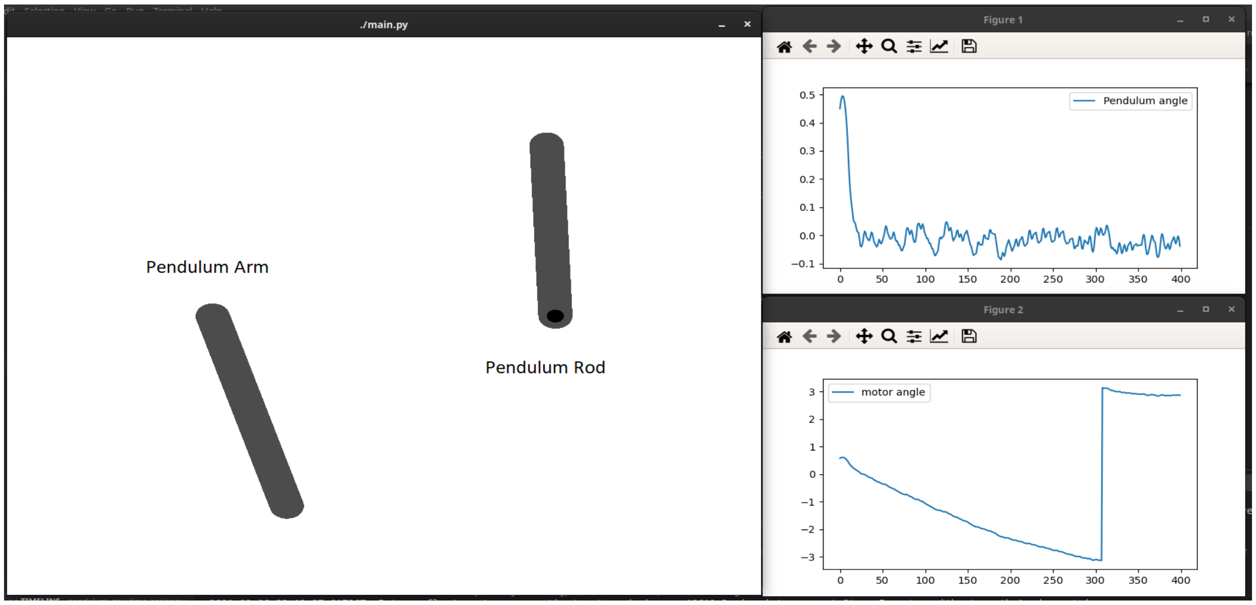

Fig. 5.6. Render of the simulated pendulum

Chapter Six

Conclusion and Future Work

Chapter Six

Conclusion and Future Work

6.1. Conclusion

Systems have become a vital part of our daily life, and intelligent control systems/subsystems are required to improve bigger ones. One of those major systems is cars, and to improve the stability of cars, we need a capable tool to benchmark the reliability of our solution. Rotary's inverted pendulum is a tool that we can use as a benchmark tool. We can give it some intelligence by using reinforcement learning, which gives it the ability to learn itself (i.e. it makes it an intelligence tool).

This system contains a free pendulum that can move in the vertical plane and a rotary arm that holds the pendulum. The arm can move in the horizontal plane, and this arm also is connected to the motor that we use to provide the system with the torque that we need to stabilize the pendulum in an upward position. This same motor is controlled by a controller that is connected to a Raspberry Pi microcontroller. In addition, we used two encoders: one for monitoring the pendulum and the other for monitoring the rotary arm. Those encoders provide information about the motion of shift, which we can translate or process into information. From that information, we can gather the speed that can be used to inform the algorithm where it takes the right action and how it is close to its goal (i.e. stabilize the pendulum in an upward position).

The software portion of this system is represented by machine learning, mainly reinforcement learning. We use the DQL algorithm to make our solution have the ability to learn itself and build the correct way to stabilize the pendulum after training time. Raspberry Pi reads data from encoders and uses these readings to take appropriate action, then it executes this action by sending a signal to the motor driver to change the motor angle at an appropriate speed and direction. The main challenge that we faced was in the hardware implementation which is the encoder failing, and the second one was in modeling the pendulum.

Our solution was able to learn how to balance the RIP and get it at an upward position, start from, and handle sudden changes in the range [-0.6, 0.6] for pendulum angle and [-10, 10] pendulum angular velocity.

6.2. Future Work

1. Simulation

- Integrating ODEs on the GPUs or dropping it out and using another way to find the current state of the pendulum in its environment for simulation purposes.
- Replace the 2d simulation with a 3d simulation viewer, to show a more realistic agent and how it acts in its environment.

2. Implementation and Scope of the Project.

- Use a more accurate software for monitoring the state of the pendulum in its environment.
- Develop a robust algorithm that may be used as a solution to other control problems.

References

[1] Machine learning - Wikipedia, the free encyclopedia, [https://en.wikipedia.org/wiki/Machine_learning], [Accessed November 2020].

[2] Reinforcement learning - Wikipedia, the free encyclopedia, [https://en.wikipedia.org/wiki/Reinforcement_learning], [Accessed November 2020].

[3] Michael L. Littman, and Andrew W. Moore, Reinforcement Learning: A Survey, Journal of Artificial Intelligence Research 4, pp. 237-285, 1996.

[4] Jie Wen, Yuanhao Shi, and Xiaonong Lu, Stabilizing a Rotary Inverted Pendulum Based on Logarithmic Lyapunov Function, Journal of Control Science and Engineering, pp. [https://doi.org/10.1155/2017/4091302], 2017.

[5] Quanser Rotary Pendulum, Workbook, [https://nps.edu/documents/105873337/0/Rotary+Pendulum+Workbook+_Instructor_.pdf/e17aa0a2-5f98-4957-b4a7-e80f0f52a4a3?t=1436282377000], [Accessed November 2020].

[6] Navin John Mathew, K. Koteswara Rao, and N. Sivakumaran, Swing Up and Stabilization Control of a Rotary Inverted Pendulum, 2013.

[7] Train a software agent to behave rationally with reinforcement learning, IBM, [https://developer.ibm.com/articles/cc-reinforcement-learning-train-software-agent/], [Accessed November 2020].

[8] Raspberry Pi - Wikipedia, [https://en.wikipedia.org/wiki/Raspberry_Pi], [Accessed November 2020].

[9] What is an Encoder, [https://www.encoder.com/articles/what-is-an-encoder], [Accessed November 2020].

[10] Rotary Encoder, [https://en.wikipedia.org/wiki/Rotary_encoder], [Accessed November 2020].

[11] Rotary Inverted Pendulum,
[https://www.seas.upenn.edu/~jiyuehe/rotary-inverted-pendulum/Encoder.html] , [Accessed November 2020].

[12] DC Motors and Stepper Motors used as Actuators,
[https://www.electronics-tutorials.ws/io/io_7.html], [Accessed November 2020].

[13] Basics: What is a Motor Controller?,
[https://www.robotshop.com/community/tutorials/show/basics-what-is-a-motor-controller], [Accessed November 2020].

[14] Feedback Control of an Inverted Pendulum with the use of Artificial Intelligence,
[https://www.researchgate.net/publication/220064514_Feedback_Control_of_an_Inverted_Pendulum_by_Use_of_Artificial_Intelligence], [Accessed June 2021]

[15] U-Bong Kim, Hyun-Kyo Lim, Chan-Myung Kim, Min-Suk Kim, Yonk-Geun Hong, and Yung-Hee Han,, Imitation Reinforcement Learning-Based Remote Rotary Inverted Pendulum Control in OpenFlow Network,
[https://ieeexplore.ieee.org/ielx7/6287639/8600701/08668411.pdf?tp=&arnumber=8668411&isnumber=8600701&ref=]

[16] Various Optimization Algorithms For Training Neural Network,
[https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6], [Accessed August 2021].

[17] How to Choose an Activation Function for Deep Learning
[https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/], [Accessed August 2021].

[18] What is Deep Learning and How does it work,
[https://towardsdatascience.com/what-is-deep-learning-and-how-does-it-work-2ce44bb692ac], [Accessed August 2021].

[19] Reinforcement Learning: Difference between Q and Deep Q learning,
[https://www.globaltechcouncil.org/reinforcement-learning/reinforcement-learning-difference-between-q-and-deep-q-learning/], [Accessed September 2021].