Palestine Polytechnic University

College of Information Technology and Computer Engineering

Goalkeeper Robot using Reinforcement Learning and Computer Vision

Team Members:

Amro Amro

Ruba Irshaid

Majd Ewawi

Supervisor Name:

Dr. Hashem Tamimi

December 2021

Abstract

In this project, we aim to build an intelligent goalkeeper robot that is able to prevent the ball from passing the goal border. The robot will move in a linear direction. It will be trained by trial and error through reinforcement learning (using the Q-learning algorithm) and computer vision techniques. After training, the robot should be able to detect where the ball is and move toward the ball to prevent it from passing the goal border. The hardware components include an Arduino UNO microcontroller, linear actuator (slider with a stepper motor), motor drivers, HP laptop for processing, and a camera as an input device. Since Python is an efficient language for dealing with complex algorithms in machine learning (ML) and computer vision (CV), we are going to use it to develop the software part of the system.

List of Contents

List of Figures

List of Tables

Chapter One

Chapter 1: Introduction

1.1 Overview and Motivation

Nowadays, the appearance of Intelligent Systems (IS) in human life has become an inevitable truth, as well the development in this field is incredibly fast. One of the most important and powerful techniques of building intelligent systems is Machine Learning (ML). Machine Learning can make the machine learn and construct its decision-making model from experiences without human intervention.

One of those systems is the robotic football team and the agent we chose in our project, the goalkeeper, which has particularly challenging characteristics, different from the other teammates in the football team, when designing and coordinating the overall system.

The main purpose of a goalkeeper, human or robot, is to defend the goal from the kicks of the opponent team and it should perform a perfect coordination between all its behaviors, depending on the game state, such as: tracking and following the ball motion, intercepting the ball before reaching the goal, covering the goal and removing the ball from the goal neighborhood. Moreover, the goalkeeper should always keep track of the ball, reacting differently depending on the position. We will talk in more detail in the next chapter about our system and how it works.

1.2 Objective

In this project we aim to build a small-scale microcontroller-based Goalkeeper Robot capable of preventing the ball from passing the goal border through a track using Reinforcement Learning (RL) with the help of a single camera module.

1.3 Description of the System

In this project, the goalkeeper system consists of a robot, a microcontroller, and a camera module positioned at the top of the area where the ball can move. The robot is implemented as a linear actuator that can move linearly left and right only.

The camera will observe the ball, and the goalkeeper will move left or right depending on the previous position and the motion direction of the ball to prevent it from passing the goal border. Figure 1.1 shows the basic block diagram of our system.

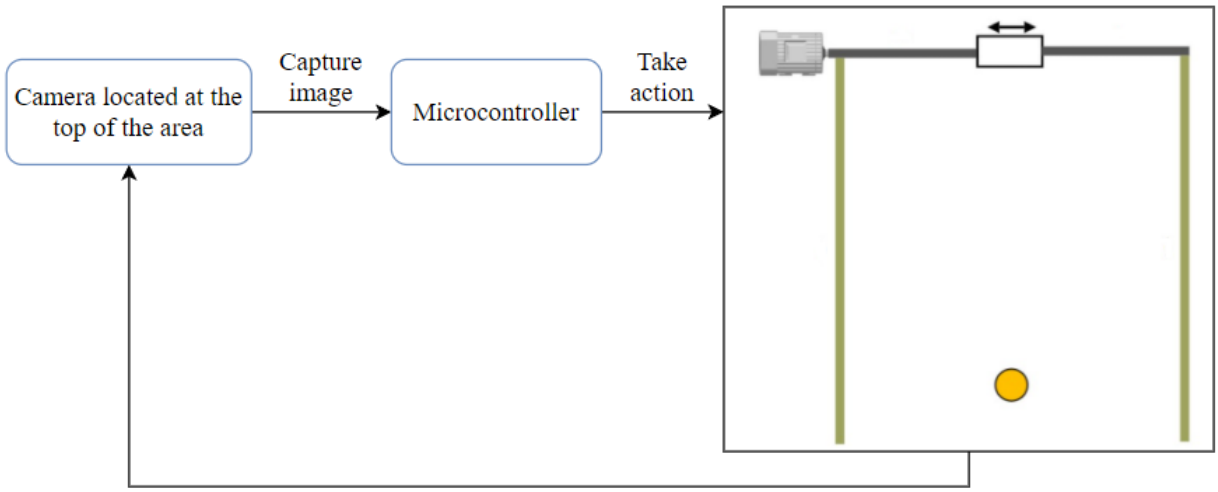

Figure 1.1. Basic system block diagram

1.4 Importance

A long-term goal of artificial intelligence and building intelligent systems is the development of algorithms capable of general efficiency in a variety of tasks and fields without the need for domain-specific tailoring. So, we will build a Goalkeeper robot capable of doing fast and intelligent reactions based on the variables of the environment. Hence, this project with a few adjustments can be a part of many systems that require similar behavior. In addition, it can be used as a game that provides a sense of fun and challenge for people.

1.5 Requirements

The system is expected to perform the following requirements:

- The user should be able to throw the ball within a specific area.
- The system should be able to detect a specific moving object which is a ball in our case in real-time using Computer Vision (CV).
- The robot should be able to move to prevent the ball from passing through the goal using Reinforcement learning (RL).

1.6 Constraints

The project is subject to the following constraints:

- The GoalKeeper robot will move in two directions left and right at constant speed.
- Only one ball should be in the area at a time.
- The ball has a distinctive (highly saturated) color that makes it easy to be detecte.

1.7 Overview of the Next Chapters

For the rest of the document, we will talk in more detail about the system from a theoretical background, design, implementation, and testing for software and hardware.

Chapter Two

Chapter 2: Background

This chapter introduces the different theoretical backgrounds required for this project, a brief description of the techniques, and the hardware components parts used in our system.

2.1 Theoretical background

In this section, we will provide some information about the general terms, such as machine learning, reinforcement learning, and computer vision:

2.2.1 Machine Learning

In 1959, computer scientist Arthur Samuel defined machine learning as a subfield of artificial intelligence that gives the computer the flexibility to learn by itself without being explicitly programmed.

Also, scientist Tom M. Mitchell provides a general definition of algorithms used in the field of machine learning: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E"[1].

Machine Learning algorithms can be categorized into three main fields, which are Supervised Learning (SL), Un-Supervised Learning (USL), and Reinforcement Learning. In Supervised Learning, the machine is provided with labeled data. Labeled data means that: when data with X: features appear in such a situation, what is the output of this situation Y: Label. In Un-Supervised Learning, the data is not labeled, and the task of the machine is to recognize the pattern between data then put them into clusters to discover more information about data.

In general, the availability of clean and enough data to train the system is not an easy process. So, in reinforcement learning (RL), there is no need for preprocessed data, and the task of the machine is to learn by itself through the concept of reward and punishment depending on his action. The goal is to maximize the total reward.

2.2.3 Computer Vision

Computer vision is a field of artificial intelligence (AI) that allows computers to extract information or get knowledge from digital images, videos, and other visual inputs, then make an action depending on such information. So, we can say that computer vision enables machines to see the world around them [2].

The image in digital form is represented by a 2D matrix of pixels where the dimensions correspond to the height and width of the image. Also, it can be represented using different color models such as RGB (red, green, blue), Gray, CMYK, or HSV. The values of pixels in the image vary depending on the color model used to represent it.

HSV is a color model used to represent digital images, in which each pixel is represented using three values: hue, saturation, and value. Hue value is used to indicate the primary color whose value ranges from 0 to 360, whereas saturation is a measure of the degree of intensity of color ranges from 0 to 100, and value is a measure of the amount of brightness of the color whose value ranges from 0 to 100.

In our project, each frame will be converted from the RGB color model into the HSV color model before being processed. The HSV is more robust to illumination changes of color than the RGB color model, making it a more attractive option in color thresholding applications compared to RGB [11].

2.2.4 Reinforcement Learning

Reinforcement learning is a machine learning (ML) method that takes place due to interactions between agents and the surrounding environment. In general, perceptions received by the agent are used not only for acting but also for improving the agent's ability to behave optimally in the future to achieve the goal. So, reinforcement learning is all about learning through trial and error [5].

How does reinforcement learning work?

In Reinforcement learning, the agent is rewarded for the desired behaviors and punished for the undesired ones. To be more specific, positive values are assigned to the desired actions to encourage the agent and negative values to undesired behaviors. With time, the agent learns to avoid the negative and seek the positive to achieve the goal [5].

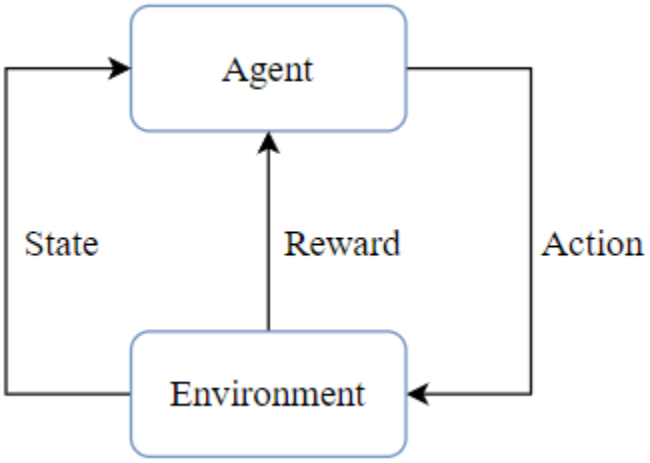

Figure 2.1. How does Reinforcement Learning work

2.2.5 Q-Learning:

Q-Learning is one of the most common reinforcement learning algorithms. In this algorithm, the agent tries an action at a particular state and evaluates its consequences in terms of the reward or punishment it receives and its estimate of the value for the current state to which it is taken. By trying all actions in all states repeatedly, it learns which are best overall, judged by long-term reward [6]. This algorithm is based on the Bellman Equation [14] as shown in equation 2.1. It's used to calculate the Q-value of each state-action in the environment which represents how good it is to take the action at state **s:**

$$Q(s\ ,a)\ new\ =\ Q(s,\ a)\ +\ \alpha\ [\ R(s,\ a)\ +\ Max[\ Q(\bar{s}\ ,\ a\ *)\ ]\ \ *\ \gamma\ -\ Q(s,\ a)\ ] \quad (2.1)$$

In the previous equation (2.1):

$Q(s\ ,a)$: The Q-value at state s and action a

$R(s,\ a)$: The reward at state s and action a

$Max\ [\ Q(\bar{s}\ ,\ a\ *)\ ]$: The maximum Q-value for the next state at all possible actions.

γ: The Discount value. It's a value between 0 and 1

α: The Learning rate

How does the Q-Learning method work?

In the Q-learning method, several steps are performed to generate the Q-Table which is the table where we calculate the maximum expected future rewards for action at each state (Q-value). Basically, this table will guide the agent to the best action at each state to achieve a goal. Figure 2.2. Shows the steps of the Q-learning algorithm [6].

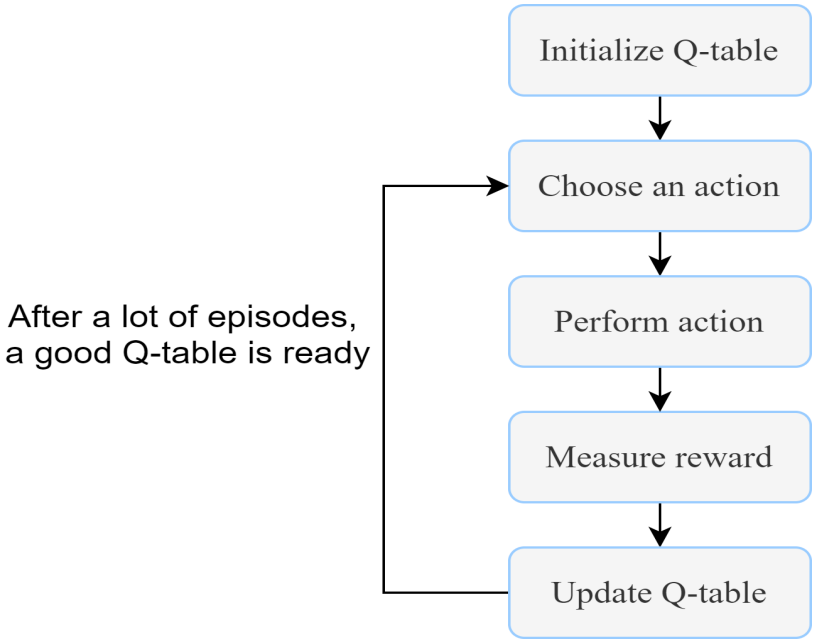

Figure 2.2. Q-learning Method steps

After many iterations, the Q-Table will contain the values that the agent will use to achieve the desired goal, the agent in any state will choose the action with maximum Q-value that leads it to the goal. The pseudo-code for the Q-Learning algorithm is as follows:

Initialize Q (s, a) arbitrarily

Repeat (for each episode):

Initialize s

Repeat (for each step of episode):

Choose a from S using policy derived from Q

Take action a, observe r, s'

Update :

$$Q(s\ ,a)\,new\ =\ Q(s,\ a)\ +\ \alpha[\ R(s,\ a)\ +\ Max[\ Q(\bar{s}\ ,\ a\ *)\ ]\ \ *\ \gamma\ -\ Q(s,\ a)\ ]$$

$$s \rightarrow\ s'$$

Until s **is terminal**

2.3 Literature Review

Foosball Table Goalkeeper Automation Using Reinforcement Learning [10]:

This project is done by a set of students at Darmstadt University of Applied Sciences in Germany. It aimed to automate physical foosball games (table soccer) systems. In the beginning, they create a simulated goalkeeper using the Unity platform and other pythons' libraries, then train it using the reinforcement learning algorithm in a try-and-error manner then they deploy it to the physical system.

They used discrete actions: move left, right, or do nothing, and the goalkeeper gets a positive reward when it held the ball and a negative one otherwise. Also, they used a Deep Q-learning algorithm to calculate the Q-values required to choose the best action.
Table 2.1. shows the results of system simulation using the Unity platform.

	Tested with 100% random shots	**Tested with 100% smart shots**
Trained with 80% smart shots	76.3 %	74.2%
Trained with 100% random shots	70.7%	62.9%

Table 2.1. Quantitative test results in simulation

Figure 2.3 shows how the goalkeeper moves in his environment.

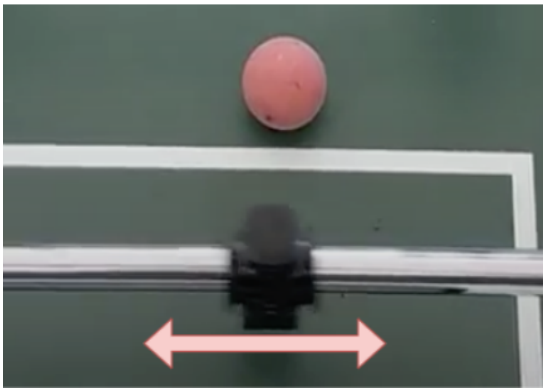

Figure 2.3. Goalkeeper movement

This project has the same goal as our project, which is to get a goalkeeper and train it until it can make the right movement decisions to prevent the ball from passing the goal border. What makes our project different is we will use a different algorithm which is the Q-learning algorithm. In their project, they used a Deep Q-learning algorithm. The difference between those two is the implementation of the Q-table. Critically, Deep Q-Learning replaces the regular Q-table with a neural network, which leads to more complexity. While using Q-learning we will have the same results with less software complexity. Also, we will reduce the hardware complexity - without losing the effectiveness of the system - by using simple hardware components: Arduino UNO, stepper motor, motor driver, and camera module. See Figure 2.4.

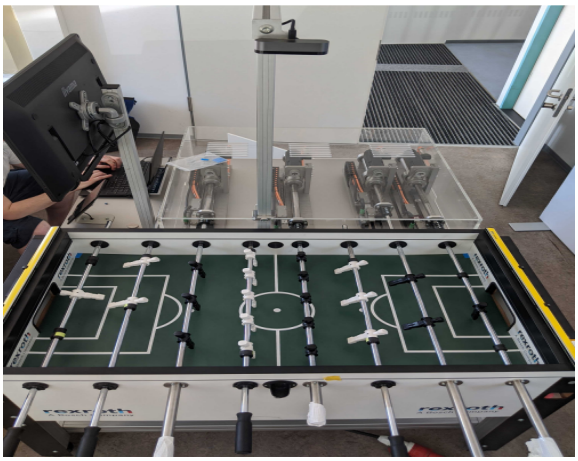

Figure 2.4. Goalkeeper hardware

The Arcade Learning Environment: Breakout-ram-v0 (Software System) [12]:

Also, Reinforcement learning (RL) is widely used in video games, for example, the Arcade Learning Environment (ALE) offers over 500 atari games using the Stella Atari emulator. In those games, the developers used SARSA(λ), a traditional technique for model-free reinforcement learning. Breakout-ram-v0 is one of those Atari games similar to our system idea with different algorithm implementation. The sliding bar can move left or right, and this game ends if the ball is not caught by the sliding bar or if the ball breaks all of the wall's pieces. See Figure 2.5.

Figure 2.5. Breakout-ram-v0

2.4 System Hardware Components

In this section, we will describe the hardware component that we will use in our project. We will discuss the alternatives.

2.4.1 Board Chips “Arduino UNO”

Arduino UNO is a microcontroller based on ATmega328p with an operating voltage of 5 volts. It has 14 digital I/O pins and 6 of them provide PWM output. It has a 32 KB flash memory and a clock speed equal to 16MHz. It has a USB port, we used it to power the Arduino by connecting it to the computer.[13]

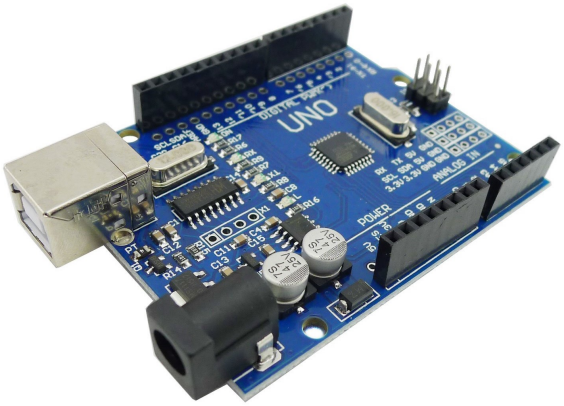

Figure 2.6. Arduino UNO

Our Choice:

Arduino UNO is our choice for the project since we need a real-time response for moving the keeper, and it's faster in dealing with hardware components "in the Goalkeeper project we deal with a stepper motor".

2.4.2 Logitech Camera c905

The c905 is a USB Portable Webcam, with 2-megapixel resolution for high-clarity video, it provides attractive video features such as light correction, contrast enhancements, and smooth high-quality video recording.

Figure 2.7. Logitech camera c905

The following are the important specifications of the camera module [9]:

- Max Resolution: 720p/30fps
- Diagonal field of view (dFoV): 55°
- Compatible with: Windows 7 or later, macOS 10.10 or later, and USB - A port

In this project, we will use the c905 camera to capture a video stream in the top of the Goalkeeper and feed the image as input to the laptop. It is also suitable because it offers a good field of view and suitable smooth videos.

The alternative in this system would be the use of the cheaper Arduino camera module instead of the Logitech camera c905. However, due to the wider field of view, better image quality, and the built-in video encoding hardware of the c905, it seemed like a better option.

2.4.3 V-Slot NEMA 23 Linear Actuator (Belt Driven) Bundle

The V-Slot Linear Actuator is a powerful component in any linear system such as our system. It consists of V-Slot Linear Rail, Mini V Gantry Plate (to save the ball from passing the goal border), Solid V wheel, GT2 Timing Pulley 30T, GT2 Belts, and Assembly Hardware.

The following are the important specifications of the V-Slot Linear Actuator [7]:

- provided with a NEMA 23 stepper motor.
- Max Speed: ~12000 mm/min
- Accuracy Positioning: 0.260mm
- Max Force: 2.5lb (11N)

Figure 2.8. V-Slot NEMA 23 Linear Actuator

The alternative in this system would be the use of a similar linear actuator with a DC motor rather than a stepper motor. However, stepper motors are more suited to applications that require accurate positioning and repeatability with a fast response to starting, stopping, and reversing as our system needs.

2.4.4 EasyDriver A3967

The motor driver acts as an interface between the motors and the control circuits. Motors require a high amount of current where the controller circuit works on low current signals. So, the purpose of motor drivers is to take a low-current control signal and then convert it into a higher-current signal that can drive a motor. The EasyDriver is a complete microstepping motor driver with a built-in translator. It is designed to operate bipolar stepper motors in full, half, quarter, and eighth-step modes.

Figure 2.9. EasyDriver

The following are the important specifications are of the EasyDriver [8]:

- Driving Voltage (6V - 30V)
- Logical Voltage (3V - 5.5V)
- Max drive current (750mA/phase)
- Automatic current-decay mode detection/selection
- Internal UVLO and thermal shutdown circuitry
- Crossover-current protection

The alternative in this system would be the use of the L298N Motor driver, both of which are compatible with our system. However, EasyDriver provides a higher voltage, it has a current protection circuit, and a thermal shutdown circuit which makes the EasyDriver better than the L298N motor driver.

2.5 Software Background

This section focuses on the software tools we plan to use in this system, such as programming languages, frameworks, and libraries.

2.5.1 Programming Languages

We are planning to use Python as our primary programming language. Python programs are simple and easy to read. Few lines in Python often correspond to tens of lines in other programming languages like C/C++, which means that writing programs in Python is time-saving practice. In addition, it deals with complex algorithms in machine learning (ML) and computer vision (CV) efficiently. Python also has a very large community, and a lot of resources help us in the coding and debugging process. And We use the Arduino IDE to program Arduino UNO in C++.

2.5.2 Libraries

The main libraries that we will use:

- **OpenCV:** Is an open-source C-based library to deal with digital data (images and videos) and includes a lot of computer vision (CV) algorithms [3].
- **NumPy:** Is a numerical computing tool and open-source python library that includes powerful functions for handling matrix arithmetic, Linear Algebra, Fourier transforms, statistical operations, basic statistical operations, etc [4].
- **PyGame:** Is a cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries designed to be used with the Python programming language [15].

Chapter Three

Chapter 3: System Design

This chapter introduces the design part of our project and the way its components are integrated together, showing the block diagram and schematic diagram for the design, in addition to some details about the algorithms we are going to use.

3.1. Conceptual Description of the System

This project aims to build a goalkeeper robot capable of tracking and detecting a moving ball and repelling it. In the system, we will use reinforcement learning to allow the goalkeeper to train itself through trial and error without having to rely on huge amounts of pre-existing data. A camera positioned at the top of the area where the ball will move will detect the ball and goalkeeper positions using computer vision. Following that, based on the prediction of the ball's final position, the goalkeeper can take one of three actions: move to the left, right, or do not change its position; a stepper motor will move the goalkeeper to perform these actions.

3.2. System Logic & Methodology

This section will briefly describe the software logic, system design and how we will build the overall system using the methods and components we discussed in the previous chapter. Figure 3.1 shows the software logic of the system starting from receiving images from the camera, detecting ball position, applying Q-learning algorithm and ending with Goalkeeper movement to the right position.

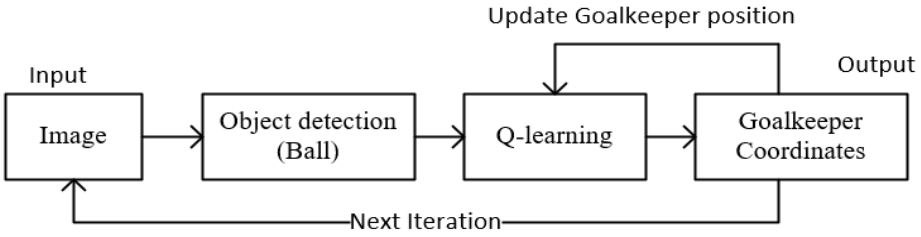

Figure 3.1. System Logic of the Goalkeeper Robot

3.2.1 Q-learning

The whole idea of Reinforcement learning is to define the best sequence of actions, in this case Goalkeeper movement, that allow it to achieve the goal in an efficient way by maximizing a long-term reward. And that sequence of actions is learned through the interaction with the environment and observation of rewards in every state. So after the training the Goalkeeper will be able to take the proper actions to prevent the ball from passing the defined goal based on the previous knowledge [5][6]. Figure 3.2. Shows the interaction between the Goalkeeper and the surrounding environment in the system.

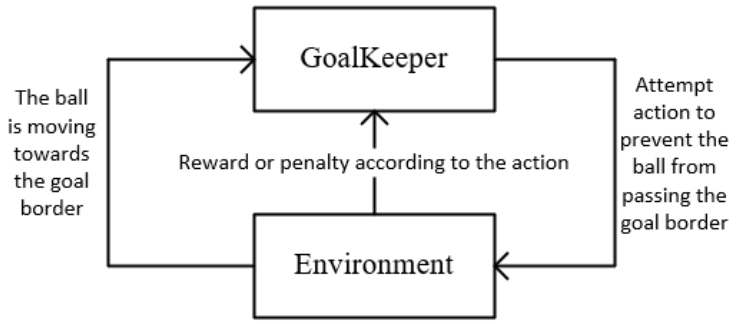

Figure 3.2. The interactions between Goalkeeper and the environment

To achieve the goal in Reinforcement learning, there must be a training stage that aims to generate the Q-Table that the Goalkeeper will use to take action in the real environment. The first step is to initialize the Q-Table with zeros for every state-action. Then the Goalkeeper starts to attempt actions at the different states in the system and observes the reward from the environment for the current state-action. After many episodes and using the good reward method the Goalkeeper will have larger Q-values for the state-action pairs which lead to a state where a higher reward action could be taken [6].

The training stage is done in a virtual environment that simulates the real system. Since we have a large number of states and actions and require an exponentially larger number of iterations to reach a good result, using the virtual environment we will cover thousands of iterations in a very short period of time.

We define the state space *S* in the following equation. In Our system, we have a set of metrics that would generate the final state. Those metrics are related to the ball position and the Goalkeeper position that can move horizontally.

$$S = \{ Ball\,(Y),\ Goalkeeper(Y) \} \quad (3.1)$$

In equation 3.1, ball (Y) and goalkeeper (Y) are explained in sections 3.2.2 and 3.2.3 respectively. X - coordinate for both goalkeeper and ball would be constant in the training stage. We can imagine that is making sense by thinking about the speed of the goalkeeper (see sections 1.6 and 2.4.3), so the speed is set to the max speed possible, which means the action of the goalkeeper when Y-Goalkeeper = 20 and Y-ball = 50, X-ball = 80 will not differ when the same Y-Goalkeeper, and Y-ball but X-ball = 90.

We also define the action space *A* in the following equation. We have exactly 3 actions that the Goalkeeper would take in a specific state.

$$A = \{ Left,\ Right,\ NoAction \} \quad (3.2)$$

Given the proposed observation space *S* and action and action space *A*, Table 3.1 displays the set of states and actions in this system. Each state-action pair in the table is unique and has a Q-value. Q-values are read and updated during the training process as described before.

State/ Action	Left	Right	No - Action
$S\,(\,Ball\,(Y),\ Goalkeeper(Y)\,)$	Q value	Q value	Q value
…	...	…	…
$S\,(\,Ball\,(Y),\ Goalkeeper(Y)\,)$	Q value	Q value	Q value

Table 3.1. The Q-Table Structure

A higher Q value indicates a more rewarding action for the given state. When the system is in the operational stage, and the Goalkeeper has already accumulated a good policy, it will select the action having the highest Q value at any state. We will discuss in more detail the number of states and the q-table size in the next chapter.

The last thing we need in this stage is a good reward function. The reward value returned by the environment to the agent should be representative of the correctness of any possible action at any given state. In other words, the reward value serves as an indication of reward and punishment. So, when the goalkeeper and ball have collided, the reward value will be increased by ten (10), and when the goalkeeper can not catch the ball, we will decrease the reward value by one (-1), and so on.

3.2.2. Computer Vision: Binary HSV Thresholding

The first step to detecting and isolating the ball is to specify its color, which also needs to be distinct from other colors in the environment (we chose orange).

In our project, we use a camera that takes 30 frames per second. We perform some processing on each frame to detect the ball and its position. First, we convert the image from RGB to be represented in the HSV color model. Then apply a threshold to the image to isolate the ball from other objects in the environment based on its color (the threshold range is determined by manually attempting different values for the threshold's minimum and maximum values and selecting the best values). As a result of this process, we get a binary image (mask image). Then a Blob detection (used to determine the connected components in the image) is performed on this image, with the largest blob representing the object to be detected.

Therefore, in order to determine the ball position, we calculate the CoG (center of gravity) for the connected component by using the following equations:

$$\mathrm{Xc} = \frac{\sum_{i=1}^{N} Xi}{N}, \mathrm{Yc} = \frac{\sum_{i=1}^{N} Yi}{N}$$

The point (Xc, Yc) represents the position of the ball in the coordinates. In our system, we are concerned just in Yc.

3.2.3. Goalkeeper Position and Movement

We first define some variables in stepper motor:

- The number of steps per revolution (step/rev): refers to how many steps that the stepper motor can do in one revolution that is equal to 360°/(1.8° per full step is a common step size rating) = 200 steps per revolution.
- The distance per revolution(dist/rev): indicates how much the travel distance in one revolution, this can be calculated by trial.

So, we can calculate the goalkeeper position by knowing the distance per revolution and dividing it by the distance we need to move. Example: let the dist/rev = 1.5 cm and we need to move the goalkeeper from Y = 0 to Y = 50. So, we need to find how many revolutions we need to move goalkeeper 0 to 50 to input this result into stepper motor: if one revolution leads to 1.5 cm, how many revolutions leads to 50 cm. Number of revolutions = 50 cm / 1.5 (cm/rev) ≈ 33 revolutions. The position of the V-slot (slider bar in sec 2.4.3) is fixed, so the X - coordinate of the goalkeeper will be constant.

3.3. System Block Diagram

As discussed in section 2.2.4 and 2.2.5, any RL system is made of two main components, the agent and the environment. Figure 3.3 shows the system block diagram from the perspective of an agent and an environment.

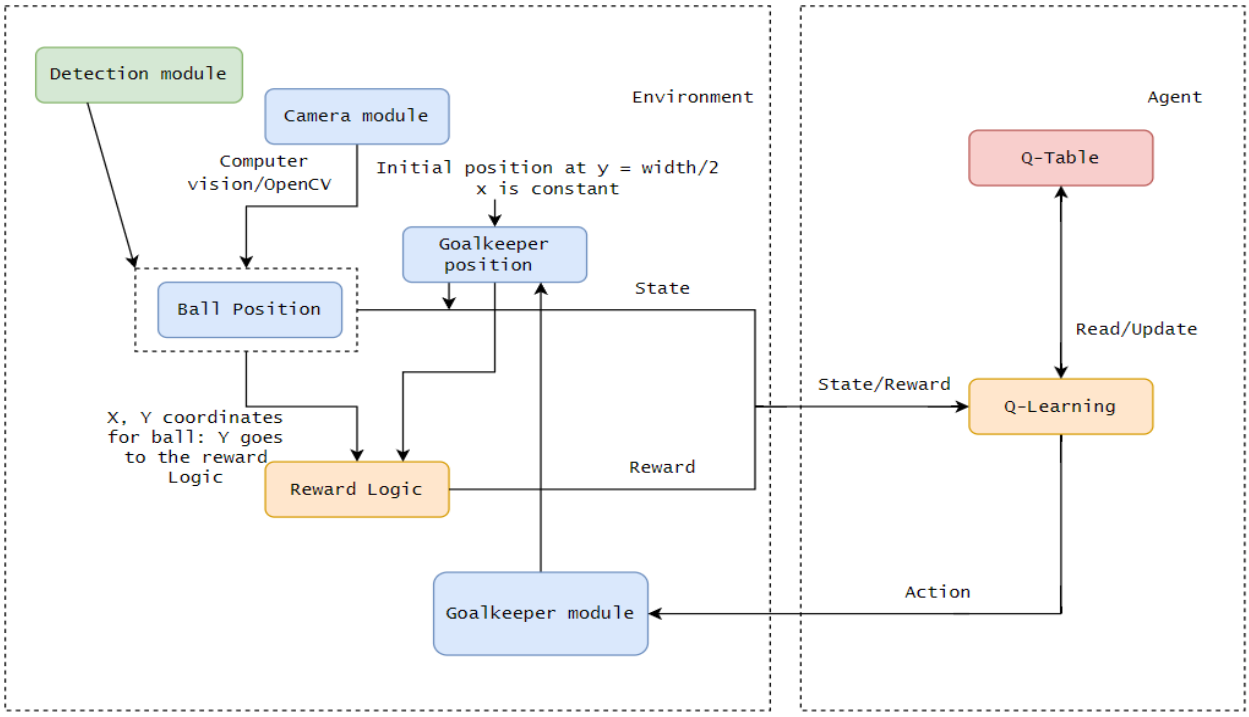

Figure 3.3. System Block Diagram

The diagram involves a loop between the two main components. The system starts by capturing the state of the environment using a camera module. This module is responsible for handling the logic related to managing the camera. The output of the camera module is an image, which is transferred to the laptop to be processed. Inside the laptop, the image input is processed by the detection module. The modules denoted by the blue color in the diagram represent a collection of logically related functions. The position of the ball produced by the detection module is then passed to the agent. We initialize the position of the goalkeeper at y = width/2, then we can find the position of it as we show in section 3.2.3. Finally, the agent issues an action in the form of control commands (go left, go right, do nothing) which are passed to the goalkeeper module. The goalkeeper module, in turn, translates the control commands into the corresponding physical motion signals that drive the system (V-Slot linear actuator).

3.4 System Schematic

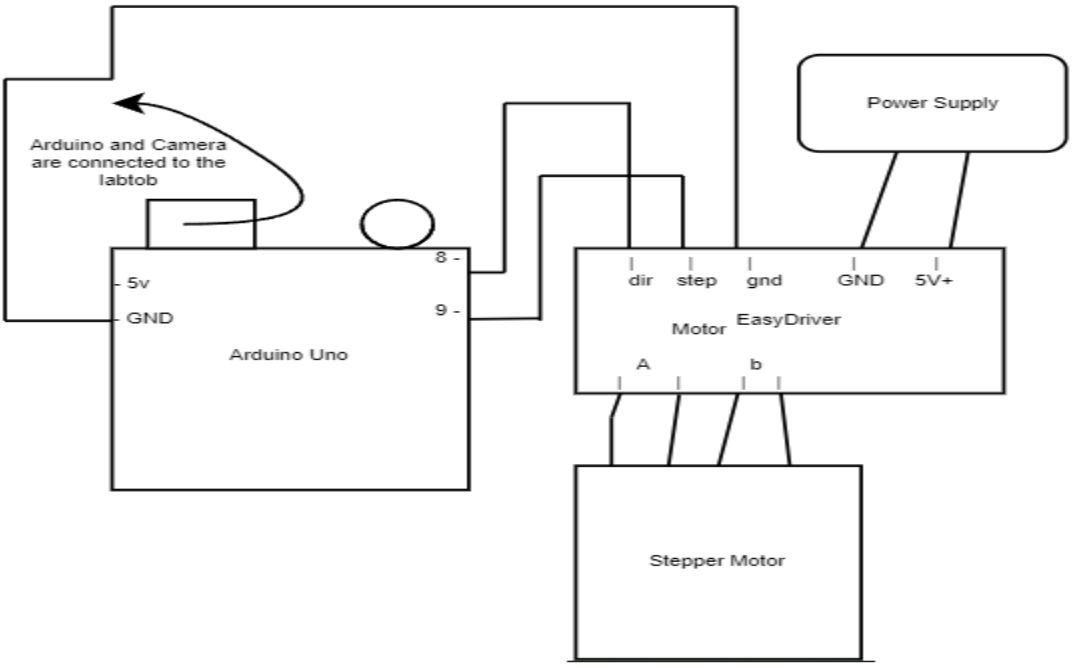

Figure 3.4. System schematic Diagram

Figure 3.4 shows the schematics of this project. It displays the various hardware components of the system and their interconnections. In this schematic, the stepper motor is connected to the EasyDriver. Which is controlled by the microcontroller to supply up to 30v. The motor driver is therefore directly powered by the power supply with voltage around 15v.

The Arduino Uno connects to the motor driver via digital Input/output bins. The Arduino is connected to a laptop with serial communication between them. Also, the Logitech camera C905 is connected to a laptop.

Moving the processing (detection and taking actions) to the laptop (high hardware resources), and sending actions to the Arduino via serial communication, will decrease the processing time and increase the response time of our system.

Chapter Four

Chapter 4: Implementation

This chapter explains the implementation part for the hardware components and the software algorithms. It dives into more details about the project overall, different hardware components, and software modules.

4.1 Hardware Implementation

Figure 3.4 provides a detailed schematic diagram of the connection of system components. Here, we attempt to describe the assembly process and arrangement of those components.

The process started with the assembly of the V-Slot NEMA 17 Linear actuator. Which is described in section 2.4.3, and can be found on the manufacturer's website.

After that, the linear actuator is put on the actual environment, see figure 4.1.

Figure 4.1. Assembled V-Slot with environment

Now, we can move to the second part of our system, which is the circuit we need to control the linear actuator (stepper motor). Figure 4.2 shows the hardware components we used to implement the main circuit for moving the stepper. The figure was generated using an online simulator called Fritzing which demonstrates the actual design of each component as well as the connections between them.

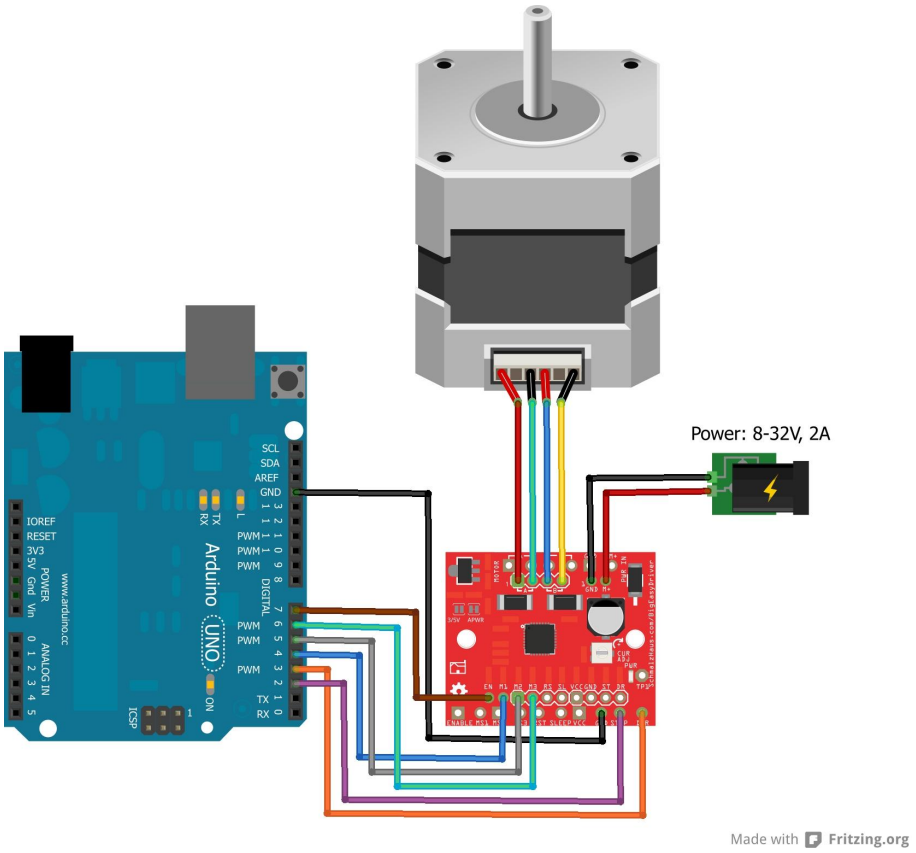

Figure 4.2. System circuit

In this system, the processing will be on the laptop. This means we need to build communication between Arduino and the laptop using a USB cable. The laptop sends the actions to the Arduino, and it will control the linear actuator see figure 4.3. Also, the camera is connected to a laptop to make the detection, and it is placed at the top of the environment in figure 4.1.

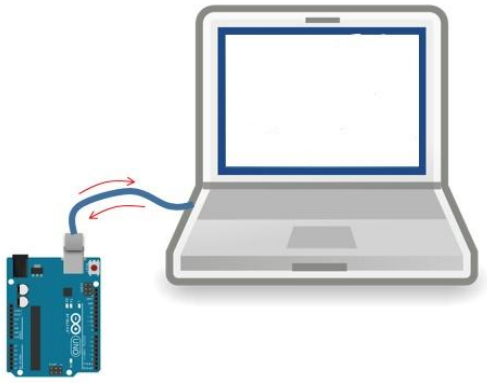

Figure 4.3. Serial communication between laptop and Arduino

As a result, the connection between the environment and the circuit in figure 4.2, implementing serial communication with a laptop and the circuit, and connecting the camera with the laptop, expresses our system's overall hardware implementation.

4.2 Software Implementation

This section describes the implementation details of the various software components of the system. It also explains the choice of different parameters and the features and functions of each software component in the system.

An essential step of this system is to install a few software libraries such as OpenCV, Numpy, and PyGame. We used the pip package manager to install all the packages we used on different platforms.

In the following, we will go in more detail about different modules in our software implementation that showed in figure 3.3.

4.2.1 Goalkeeper Module

The Goalkeeper module (linear actuator module or Arduino module) is responsible for controlling and managing the state of the physical goalkeeper. This module will be located in the Arduino and receive the actions (go left, go right, or do nothing) from the laptop via serial

communication. The following list summarizes the expected responsibilities of the Goalkeeper module.

- Initialize the stepper motor
- Provide an interface for easily controlling the goalkeeper direction and movement
- Provide an interface for easily controlling the speed of the goalkeeper

The module should expose the following functionalities:

- move():
  - Accepts a direction should the goalkeeper do from serial communication.
  - Calling one of the following functions based on the direction parameter.
- goRight():
  - Accepts a speed parameter (constant at the maximum speed).
  - Change the stepper motion to be clockwise.
- goLeft():
  - Accepts a speed parameter (constant at the maximum speed).
  - Change the stepper motion to be counterclockwise.

4.2.2 Camera Module

The camera module is responsible for controlling and managing the state of the camera connected to the laptop. This module is responsible for initializing and controlling a reliable video stream from the camera. The following list summarizes the expected responsibilities of the camera module.

- Initialize and manage a video stream from the camera
- Provide an interface for reading frames
- Close the stream upon termination

The module should expose the following functionalities:

- read_frame():
  - Returns the last frame in the buffer.

- print_status():
    - Outputs debugging information to the console including the resolution, FPS and buffer size.

- release():
    - Releases the camera stream.

4.2.3 Detection Module

This module with the camera module will provide the detection process of the ball in the system. The following list summarizes the expected responsibilities of the camera module.

- Initialize the camera module
- Provide a border to surround the environment so the ball can move in this border
- Return the (x, y) coordinates of the ball

The module should expose the following functionalities:

- draw_border():
    - Draw the border of in the frame

- get_the_coordinate_values():
    - Return the x and y for the ball in the frame

4.2.4 The Environment Simulation Module

This module is considered the most important module in our system. This module provides a virtual environment to train the goalkeeper without dealing with any hardware components. In virtual environments, we can make a lot of experiments, save time, and avoid hardware failures using just software. The following list summarizes the expected responsibilities of the environment module.

- Initialize the learning loop so the Q-Table will be generated in this module

- Provide a software environment to train the agent
- Provide an effective reward function that rewards good behavior and punishes inferior behavior

This module exposes the following functionalities:

- init():
    - Initializes the simulation's software components like: window, ball, paddle

- reset():
    - Reinitialize the state of the ball and paddle

- render():
    - Render the simulation environment with ball, paddle, and number of goals and hits, see figure 4.4.

- step():
    - Accepts action as a parameter
    - Executes the action and observes the change in features.
    - Return a calculated reward, the next state, and a flag indicating if we reached the terminal state or not
    - Terminal state is the state when the ball collides with paddle (goalkeeper)

- get_discrete_state():
    - Accepts state as parameter
    - Return the normalized state
    - This function is used to reduce the number of states possible

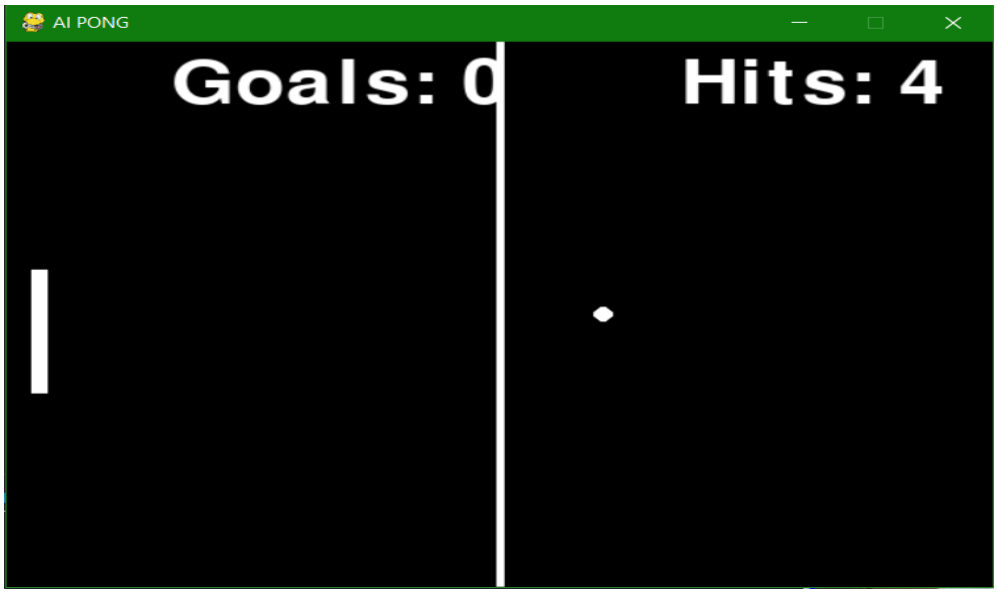

Figure 4.4. Simulation environment

4.2.5 The Learning Loop

The learning loop is a continuous interaction between the environment and the goalkeeper. in which the goalkeeper takes actions and the environment returns back the next state of the system and a reward value.The system will use these values from the environment to update the Q-table as we described in section 3.2.1.

The loop starts by initializing the Q-table with zero values. Since the goalkeeper has not learned anything yet. Then we will start the loop, the goalkeeper will start the loop taking action and observing the result.

An important part of this loop is called an episode. An episode usually ends after a specific number of iterations or when we reach some specific goal based on the system we are developing. We decided to end the episode when the goalkeeper reaches a terminal state. When the agent reaches the terminal state of the interaction loop and hence updates the table, the episode is ended, and a new episode is initiated with the modified table.

The advantage of using episodes is that it makes it easier to track the progress of the goalkeeper over different episodes, and we can easily compare the different learning stages. In each episode as long as the goalkeeper didn't reach the terminal state, the goalkeeper selects the best action for the current state based on the corresponding values in the Q-table. It then passes that action to the environment. The environment executes the action and returns the next state of the system, a reward value. The goalkeeper uses those returned values to update the table with the help of equation 2.1.

Finally, the next state is set as the current state of the system and another iteration of the loop is executed. Our choices for the learning rate, α, and the discount factor, γ are 0.00001 and 1 respectively. We chose those values by trial and error method.

4.2.6 Managing the Work

As for writing and managing the code, we made use of the popular version control tool called Git. Using Git, we were able to write code off-campus at home away from the system, push it to the cloud and easily pull it on any device. The code was hosted on a private GitHub repository.

Chapter Five

Chapter 5: Testing and Results

This chapter discusses the testing of the various system components and the overall operation of the system. It describes the results obtained from running the learning loop until obtaining a good policy.

5.1 Hardware Testing

5.1.1 Testing the Microcontroller (Arduino UNO)

Testing the microcontroller and ensuring its functionalities is vital to the success of the project. We started by running a simple program to ensure that the Arduino worked without any problems. We then connected it to the laptop and sent simple data from the laptop to Arduino, and checked if the data was transmitted correctly or not. We finally verified that all pins worked correctly, and the data received as expected.

5.1.2 Testing the Camera (Logitech C905)

Testing the camera is pretty easy, so we connected it with the laptop and confirmed that it works correctly.

5.1.3 Testing the System Circuit

After testing the microcontroller and camera, we can now do an integrated testing, by gathering all the hardware components in one circuit, see figure 4.1.

In this stage, we want to ensure that the goalkeeper can move right or left with a specific speed and a number of steps. It is verified that the testing passed correctly.

5.2 Software Testing

We tested the functionality of the software components described in section 4.2 starting from the lower-level components like goalkeeper module to the higher-level ones like the environment.

The methodology for testing the software components was very simple. We first ran each module separately (unit testing) and ensured that all of the functionality exposed by the software module is working correctly. The details of the software modules and their intended functionality are described, in detail, in section 4.2. As far as the testing is concerned, we ran the software module through a variety of cases and ensured that it returns the expected outputs.

The next step was integrating the software modules (integrating testing). We started by integrating the camera and detection modules together. After the Q-table is produced from the learning loop in the environment simulation, we write a python script that connects both camera and detection module with the goalkeeper module (Arduino module) by using the serial library. Similar to the previous unit tests, we ran the integrated system through a number of possible cases to ensure the proper functionality of the system.

5.3 Results

The following section introduces and discusses the results obtained from the operation of the project. It describes the outcomes of the project.

We conducted our experiments on a whiteboard with a yellow ball (40mm diameter). The goalkeeper was at the edge of the board, the camera that tracks the ball was at the top, while the Arduino was connected to the laptop. Based on the ball movement the goalkeeper was moving. In the beginning, it couldn't prevent the ball from passing the limits, and it was moving randomly, but the increase of the episodes led to improvement in the performance of the goalkeeper, and when it reached 4000 episodes, it was able to move with the ball movement and prevent it from passing the board limit.

Figure 5.1 shows the average reward accumulated by the agent over the course of 4000 episodes as a function of the episode number. The rate of punishable actions that the agent would take during the learning process decreased gradually as the agent moved through the first 1500 episodes until the agent reached a state where Q-table converged into an effectively rewarding policy. At that point, the agent was actively only choosing actions with a good Q-value and ignoring the ones with lower values.

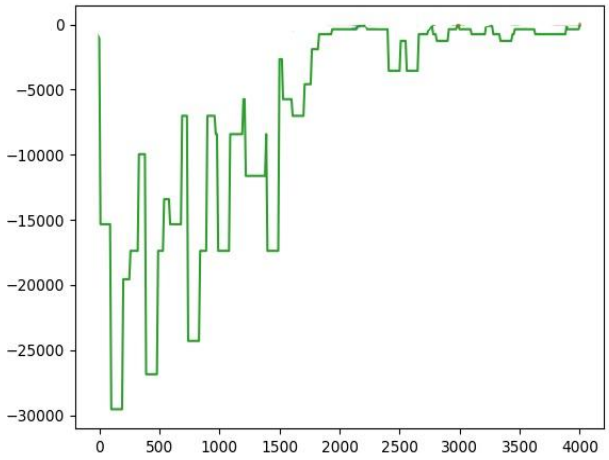

Figure 5.1. Accumulated reward as a function of episode

Recall table 3.1. After running 4000 episodes, the agent was able to achieve a policy that maximizes its view of the environment area, and therefore it was able to take suitable actions based on the high Q-Value in the Q-Table. However, we conducted our environment to be 600X440 pixels, but we are concerned about the Y-axis as we mentioned in section 3.2.1. That means the Q-Table should have 193600 (440 states possible for Goalkeeper-Y, and 440 states possible for Ball-Y states possible), and as we know, this is a big number, and it will affect the performance of our system.

A solution to this would be the use of the get_discrete_state() function that is mentioned in section 4.2.4. So, we can consider a group of pixels as one pixel, and this will not affect the accuracy of the system, because the size of a pixel is very small. So, we can neglect some of them. After trying many values of how many pixels can be grouped, we put every 15 pixels in one group, and this will reduce the Q-Table to be 30X30 states or 900 states, and this is a very small number compared to the previous value.

Finally, and as a result of the achieved Q-table, we are able to build an intelligent goalkeeper with minimal Q-Table size, and minimal hardware components possible.

5.4 Conclusion and Future Work

Developing an intelligent system using artificial intelligence techniques is not an easy task. However, Machine learning (ML) allowed for robust intelligent systems that could react to a wide range of situations.

In this project, we managed to build an intelligent microcontroller-based goalkeeper robot, with the help of a camera module. The camera was used to capture images of the environment. By processing the input and extracting the proper point of the ball, we were able to train our Reinforcement Learning agent to prevent the ball from passing through it.

We were able to achieve these results with a very minimal hardware and software configuration. This is done using an Arduino microcontroller, single camera module, one motor, and one driver.

One important lesson that we learned through the course of this project was the learning agent in the virtual environment, then using the results obtained from this learning, and applying them to the real world, is not an easy process, it requires a lot of adjustment, a lot of testing, and very precise is needed. However, the learning of agents in a virtual environment (VE) will save tons of time. Absolutely, this is a very important advantage of using the VE.

For future work, we can enhance the current system in different aspects:

- The current design of this system applies for one dimension, just moving right or left, so we can improve the movement by adding a new dimension. In other words, with the new design, the goalkeeper can move right, left, up, and down.

- The current implementation of the system is color distinctive when detecting the ball. We can improve this, so the system can see any ball with any color.

References

[1] "History and relationships to other fields. Machine Learning." Accessed 10 24, 2021. https://en.wikipedia.org/wiki/Machine_learning.

[2] "What is computer vision?" Accessed 10 26, 2021. https://www.ibm.com/topics/computer-vision.

[3] "OpenCV Documentation." Accessed 10 28, 2021. https://docs.opencv.org/.

[4] "NumPy Documentation." Accessed 10 28, 2021. https://numpy.org/doc/.

[5] "Reinforcement learning" Accessed 10, 29 2021. https://en.wikipedia.org/wiki/Reinforcement_learning.

[6] "Q-learning" Accessed 10, 29 2021. https://en.wikipedia.org/wiki/Q-learning.

[7] "V-Slot NEMA 17 Linear Actuator Bundle (Belt Driven)." Accessed 11 6, 2021. https://openbuildspartstore.com/v-slot-nema-17-linear-actuator-bundle-belt-driven/

[8] "A3967, Datasheet". Accessed 11 6, 2021 https://www.alldatasheet.com/datasheet-pdf/pdf/83571/ALLEGRO/A3967.html

[9] "C905 HD WEBCAM." Accessed 20 5, 2022. 25173.1.0.pdf (logitech.com)

[10] Tobias Rohrer, Ludwig Samuel, Adriatik Gashi, Gunter Grieser and Elke Hergenröther, Foosball Table Goalkeeper Automation Using Reinforcement Learning, 2021.

[11] Gonzalez, Rafael; Richard Woods. Basic Edge Detection, "Digital Image Processing", (3rd ed.), 2008.

[12] "Gym." Gym. Accessed 12, 24 2021. https://gym.openai.com/envs/Breakout-ram-v0/.

[13] "Arduino UNO ". Accessed 11, 22 2021. Arduino Uno Rev3 — Arduino Online Shop

[14] “Bellman equation”. Accessed 10, 29 2021.
https://en.wikipedia.org/wiki/Bellman_equation

[15] “Pygame”. Accessed 05, 23 2022.
https://en.wikipedia.org/wiki/Pygame