
TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

SAGARMATHA ENGINEERING COLLEGE

A

PROJECT REPORT

ON

# AUTONOMOUS AGENT USING REINFORCEMENT LEARNING

BY

AANANDA PRASAD GIRI 43251

ABHASH MANANDHAR 43252

ISHAN POKHREL 43259

YUKESH NEPAL 43276

A PROJECT REPORT SUBMITTED TO THE DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR IN ELECTRONICS ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

SANEPA, LALITPUR, NEPAL

MARCH, 2024

# AUTONOMOUS AGENT USING REINFORCEMENT LEARNING

BY

Aananda Prasad Giri 43251

Abhash Manandhar 43252

Ishan Pokhrel 43259

Yukesh Nepal 43276

Er. Baikuntha K. Acharya

Lecturer

Deputy Head of Department

A project submitted to the Department of Electronics and Computer Engineering in partial fulfilment of the requirements for the degree of Bachelor in Electronics Engineering

Department of Electronics and Computer Engineering

Institute of Engineering, Sagarmatha Engineering College

Tribhuvan University Affiliate

Sanepa, Lalitpur, Nepal

## COPYRIGHT ©

## DECLARATION

We hereby declare that the report of the project work entitled **"Autonomous Agent Using Reinforcement Learning"** which is being submitted to the Sagarmatha Engineering College, IOE, Tribhuvan University, in the partial fulfillment of the requirements for the award of the Degree of Bachelor of Engineering in Electronics, Communication and Information Engineering, is a bonafide report of the work carried out by us. The material contained in this report has not been submitted to any University or Institution for the award of any degree.

We authorizee IOE, Sagarmatha Engineering College to lend this thesis to other institution or individuals for the purpose of scholarly research.

**Aananda Prasad Giri 43251**

**Abhash Manandhar 43252**

**Ishan Pokhrel 43259**

**Yukesh Nepal 43276**

## ACKNOWLEDGEMENT

First and important, we're grateful to Department of Electronics and Computer Engineering, Sagarmatha Engineering College for providing us the possibility and resources to paintings in this challenge as a part of our syllabus. We might additionally like to specific our sincere gratitude to our undertaking manager, Er. Baikuntha Kumar Acharya for his precious and insightful tips for the project. We are constantly thankful for his encouragement, help and assistance in every manner feasible with the undertaking.

We specific our unique way to our Head of Department, Electronics and Computer Engineering, Er. Bharat Bhatta for his immeasurable and optimistic recommendations in the course of the planning of this project. His willingness to give his time so generously has been very a great deal preferred.

Our task will not have been a hit with out the assist and guide of our assignment and lab coordinator Er. Bipin Thapa Magar who gave his precious time to help us in want. We could additionally want to thank him for his assist and assistance. We might additionally want to express unique thanks to our dad and mom and family contributors for his or her non-stop assist and motivation with out their steerage we wouldn't be here.

Last but now not the least, we would like to increase our thanks to all people from the BEI-076 batch, Sagarmatha Engineering College for his or her continuous assist, motivation, and beneficial hints for the completion of the project.

Aananda Prasad Giri 43251
Abhash Manandhar 43252
Ishan Pokhrel 43259
Yukesh Nepal 43276

## ABSTRACT

Our project addresses the important area of autonomous navigation, aiming to improve efficiency through sensor-based methods, moving away from reliance on traditional camera systems. In modern autonomous systems, camera-based approaches dominate, but they often face limitations in different environmental conditions. We seek to overcome these limitations by exploring alternative sensing modalities and leveraging reinforcement learning techniques. In this context, our project uses reinforcement learning and Q-learning methods in a global simulated grid environment.This simulation serves as a controlled and adaptive platform for our agents to learn optimal navigation strategies through simulated trial and error. Choosing a simulation environment allowed us to repeatedly refine and test our approach before implementing it in real-life scenarios. Equipped with three ultrasonic sensors, a DC motor, and a motor controller, our RL robot is designed to demonstrate adaptability and durability in dynamic real-world environments.Through rigorous testing in maze scenarios in a simulated environment, we validate the feasibility and effectiveness of our approach.Ultimately, our project aims to push the boundaries of autonomous navigation by demonstrating the practical effectiveness of traditional Q-learning methods in real-world applications.By focusing on sensor-based efficiency and adaptability, we are paving the way for advances in autonomous systems that can thrive in diverse environmental conditions.



**Keywords:** Q-learning, Autonomous navigation, Reinforcement Learning

## TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF ABBREVIATIONS

| | |
|---|---|
| $\alpha$ | Learning rate |
| $\epsilon$ | Exploration rate |
| $\gamma$ | Discount factor |
| A | Current Action |
| $A'$ | Next Action |
| AI | Artificial Intelligence |
| DQN | Deep Q Network |
| GPS | Global Positioning System |
| GPU | Graphics Processing Unit |
| IoT | Internet of Things |
| MDP | Markov Decision Process |
| QN | Q-Networks |
| Q(s,a) | Q-value function |
| r | Immediate reward |
| RD | Research and Development |
| RL | Reinforcement Learning |
| ROS | Robot Operating System |
| S | Current state |
| S' | Next State |

# CHAPTER 1

# INTRODUCTION

## 1.1 Background

In current years, the fields of robotics and synthetic intelligence have made extensive advances, leading to the emergence of self sufficient dealers capable of navigating and expertise their environment independently. These advances have attracted big interest throughout lots of industries, promising transformative affects in transportation, exploration and extra. A key mission in empowering those actors is the development of simultaneous localization and mapping (SLAM) techniques, which enable precise environmental mapping and identification of the placement of the actors within it. While traditional techniques frequently rely on visual SLAM techniques, our challenge adopts a brand new method via focusing on a sensor-wealthy agent ready best with ultrasonic sensors and magnetometer. This sensor configuration, in contrast to traditional strategies that depend upon GPS and photograph sensors, is designed to offer the environmental comments essential for independent navigation and target acquisition. By exploiting statistics from ultrasonic sensors and magnetometers, our gadget aims to enable the agent to navigate its environment successfully and achieve predefined desires. This departure from conventional sensor configurations represents a substantial innovation inside the subject of autonomous navigation. Through the mixing of ultrasonic sensors and magnetometers, our venture targets to display the feasibility and effectiveness of an automated navigation device that does not depend on GPS or photograph sensors. Going ahead, the consequences of this research are profound. Advances in self reliant agent navigation, pushed by means of sensor-rich configurations and new algorithms, promise many programs past traditional SLAM strategies. These packages include but aren't constrained to improving self sustaining vehicles and developing revolutionary answers for exploration and more. Our task is at the vanguard of this studies, bridging the gap among past achievements and destiny packages of sensor-based automated navigation systems.

## 1.2 Problem Definition

In the field of reinforcement learning (RL) and autonomous agents, there is a notable lack of comprehensive research focusing on the seamless integration of RL techniques into navigation systems. Current approaches often face challenges related to robustness and adaptability in dynamic environments, limiting the practical deployment of autonomous agents. This project aims to fill this gap by developing a novel framework that combines RL algorithms with sensor-equipped embodied agents.

The primary goal is to bridge the theoretical advancements in RL with practical applications in autonomous navigation. By integrating RL techniques with sensor-based navigation, the project seeks to enhance the adaptability, efficiency, and intelligence of autonomous agents, enabling them to autonomously navigate complex environments and reach predefined targets with precision. Through this endeavor, we aim to contribute to the advancement of intelligent navigation systems and facilitate the development of more effective autonomous agents across various domains.

## 1.3 Objective

The objective of this project is:

- To build an autonomous agent for reaching target using Reinforcement Learning.

## 1.4 Features

Our task boasts several key capabilities aimed at improving independent navigation abilities. These include the capacity to navigate the agent to the target autonomously, ensuring seamless and efficient traversal via the surroundings. Additionally, our machine is equipped with the functionality to comply with instructions and execute routine responsibilities autonomously, similarly augmenting its application and versatility. Furthermore, the device operates in actual-time, allowing

the agent to make timely and responsive navigation selections based totally on present day sensory inputs, ensuring adaptability and performance in dynamic environments. Combined, these features make contributions to the effectiveness and reliability of our self sustaining navigation machine, paving the manner for improved performance in numerous situations.

## 1.5 Feasibility

### 1.5.1 Technical Feasibility

The utility of deep reinforcement gaining knowledge of techniques has established to achieve success in improving the talents of independent agents. Leveraging open-source libraries and frameworks consisting of TensorFlow or PyTorch helps the implementation and development of deep reinforcement gaining knowledge of algorithms within our project. Moreover, the feasibility of our assignment hinges on accomplishing real-time overall performance, permitting the agent to successfully process sensory inputs, make navigation choices, and execute actions in a timely way. This real-time capability is essential for ensuring the effectiveness and responsiveness of our independent navigation system, thereby underpinning the general technical feasibility of the assignment.

### 1.5.2 Operational Feasibility

The operational feasibility of our mission encompasses several key components. Firstly, making sure get entry to to the specified computational assets, inclusive of both hardware and software program additives, is important for seamless development and implementation. Secondly, attaining actual-time performance is crucial, as the machine must effectively manner sensory inputs, make navigation choices, and execute moves in a well timed way to efficaciously navigate its surroundings. Additionally, the machine's robustness and flexibility are paramount, necessitating its capability to operate successfully in numerous environmental situations, including varying lighting or dynamic surroundings. Lastly, consumer-friendliness is a crucial attention, with the embodied agent designed to be intuitive to have interac-

tion with, capable of receiving commands or commands in a sincere way, hence improving ordinary usability and accessibility. Taken collectively, addressing these operational feasibility elements ensures the practical viability and effectiveness of our autonomous navigation machine.

## 1.6 System Requirements

The system requirements of our project are:

### 1.6.1 Software Requirements

The software requirements for our project are as follows:

(a) Programming Language: C++

(b) Deep Reinforcement Learning Framework : TensorFlow , PyTorch

### 1.6.2 Hardware Requirements

The hardware requirements for our project are as follows:

(a) Arduino UNO

(b) Motor Driver

(c) DC Motors

(d) Wheels

(e) Li-Po Battery

(f) Magnetometer

(g) Ultrasonic Sensor

### 1.6.3 Functional Requirements

The functional requirements for our project are as follows:

(a) Real-Time Response to Sensor Inputs: The robot must quickly process sensor data, adjusting its trajectory promptly to avoid collisions and navigate efficiently through the maze.

(b) Correct action execution: Execute correct actions based on RL decisions, such as moving, turning, or stopping.

### 1.6.4 Non-functional Requirements

(a) Real-time performance: The system should be capable of processing sensory inputs, making navigation decisions, and executing actions within a response time of less than 100 milliseconds to ensure smooth and timely operation.

(b) Accuracy and Stability of the Q-Learning System: The Q-Learning system should achieve a localization accuracy of at least 5 centimeters. It should maintain stable pose estimation with a deviation of no more than 2 degrees in orientation, even in dynamic environments.

(c) Robustness to environmental variations and lighting conditions: The system should be able to operate effectively in a variety of environmental conditions, including low light conditions and dynamic surroundings. It should maintain at least 90% accuracy in pose estimation and mapping under various lighting conditions and environments.

# CHAPTER 2

# LITERATURE REVIEW

The article by Yuxi Li provides a comprehensive literature review on deep reinforcement learning (RL). It explores the integration of deep learning techniques with RL algorithms for sequential decision-making. The review covers the foundational concepts of RL, the advantages and challenges of deep RL, various algorithms such as DQN and PPO, and their applications in robotics and game playing. It also addresses research gaps and future directions in deep RL. Overall, Li's review serves as a valuable resource for understanding the principles, algorithms, applications, and challenges of deep reinforcement learning, informing our project on visual SLAM and autonomous agent navigation [3].

"Reinforcement Learning Algorithms: An Overview and Classification" categorizes RL methods based on key dimensions such as model-based vs. model-free and value-based vs. policy-based approaches. It discusses traditional techniques like dynamic programming, as well as modern advances such as deep Q-networks and policy gradient methods. The article also highlights RL's practical applications across diverse domains, emphasizing its effectiveness in complex decision-making contexts. This structured framework provides valuable insights for researchers and practitioners, enhancing understanding of RL's theoretical foundations and practical implications in AI and autonomous systems [4].

Q-learning, introduced by Watkins and Dayan in 1992, is a fundamental reinforcement learning algorithm that iteratively refines the estimate for the optimal action-value function of a Markov Decision Process (MDP) by exploring various state-action pairs stochastically. It is a model-free method aimed at finding the optimal policy through iterative updates of the action-value function and has been shown to converge under specific mathematical properties. Q-learning's significance extends to its connections with other reinforcement learning algorithms, such as TD(0), highlighting its importance in stochastic approximation theory. Over the years, Q-learning has been extended and modified to enhance its performance and

applicability in various domains, including robotics and cognitive modeling, making it a cornerstone in the field of reinforcement learning [5].

Multi-agent reinforcement gaining knowledge of (MARL) has garnered large attention inside the realm of self-sufficient cars providing a comprehensive survey on deep reinforcement mastering's application in autonomous riding, highlighting both improvements and challenges. MARL's relevance in complicated RL problems like coordinating self-reliant automobiles, underscores the cooperative nature of multi-agent structures. In their evaluation, recognition on MARL challenges and programs, in particular emphasizing characteristics important for MARL, consisting of non-stationarity and scalability mainly into MARL algorithms for self-reliant motors, exploring latest advances and research guidelines in designing independent vehicles using MARL techniques. MARL offers a promising method for addressing coordination and decision-making demanding situations in self-reliant vehicle systems with the aid of facilitating collaborative studying amongst more than one seller in dynamic environments, aiming to enhance performance, safety, and adaptableness through shrewd selection-making and coordination strategies [6].

Multi-agent reinforcement studying for self-sufficient automobiles experimented with deep learning version educated with reinforcement gaining knowledge of to maximize the speed of the self-riding vehicle in the multi-lane freeway. The fashions they experimented with in three distinct site visitor situations showed an average pace from the range from eighty one.66km/h to 87.96km/h. Their nice version changed into a neural community configured to approximate the premiere Q-gaining knowledge of feature [7].

# CHAPTER 3

# RELATED THEORY

## 3.1 Hardware Constituent

### 3.1.1 Arduino Uno

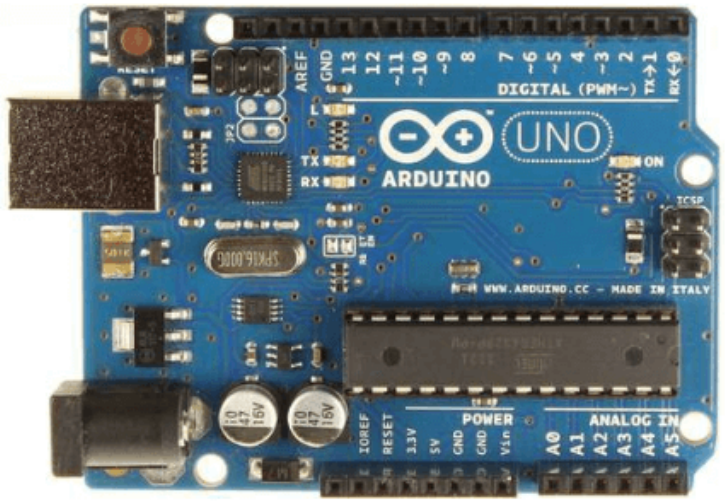


**Figure 3.1:** Arduino UNO

The Arduino Uno is a widely-used microcontroller board renowned for its simplicity, versatility, and affordability. Powered by the ATmega328P microcontroller chip, it operates at 16 MHz and boasts 32KB of Flash memory, 2KB of SRAM, and 1KB of EEPROM. Featuring 14 digital I/O pins, including 6 PWM outputs, and 6 analog input pins, it offers ample connectivity for interfacing with various sensors, actuators, and electronic components.

The board can be powered via USB connection or an external power supply within the recommended input voltage range of 7-12V. Its USB interface facilitates easy programming and serial communication with a computer, using the Arduino IDE based on simplified C/C++ programming language.

### 3.1.2 Ultrasonic Sensor


**Figure 3.2:** Ultra Sonic Sensor

An ultrasonic sensor is a type of sensor that emits ultrasonic sound waves and measures the time it takes for the waves to bounce back after hitting an object. These sensors are commonly used for distance measurement, object detection, and obstacle avoidance in various applications such as robotics, automation, and security systems. The sensor typically consists of a transmitter, which emits ultrasonic waves, and a receiver, which detects the reflected waves. By calculating the time difference between sending and receiving the waves, the sensor can determine the distance to the object.

$$\text{Distance} = Speed \cdot time \tag{3.1}$$

- Speed of sound in air = 344 m/s = 34,400 cm/s
- ultrasonic sensor measures time in µs.

$$\text{Distance measured} = time * 0.034/2cm \tag{3.2}$$

- The factor of 1/2 is used because the ultrasonic sensor measures the round trip (to the object and back).

### 3.1.3 Magnetometer

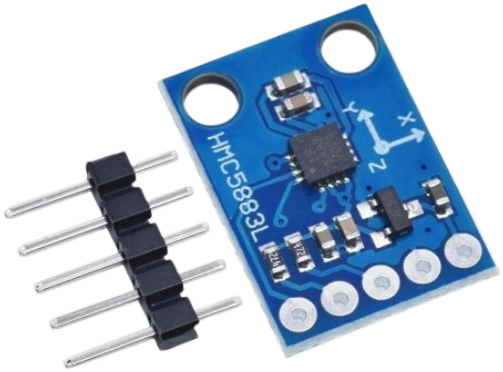


**Figure 3.3:** Magnetometer

A magnetometer sensor is a device used to measure the strength, direction, and variations of magnetic fields. It is a crucial component in various applications ranging from consumer electronics to scientific research. There are different types of magnetometers, including fluxgate, Hall effect, and SQUID magnetometers. Magnetometer sensors operate on the principle of electromagnetic induction or the Hall effect.Magnetometers are used in compasses for navigation and in electronic devices like smartphones for determining orientation. Magnetometers are used in geological surveys to map subsurface structures, locate mineral deposits, and study the Earth's magnetic field variations. Magnetometer sensors require calibration to account for environmental factors, sensor imperfections, and magnetic interference from nearby objects. Factors such as temperature variations, electromagnetic interference, and sensor orientation can affect the accuracy of magnetometer readings and must be considered during calibration and data interpretation.

## 3.2 Software

### 3.2.1 Reinforcement Learning

Reinforcement learning is a distinct learning paradigm within the field of machine learning, characterized by its focus on an agent's interaction with an environment to learn optimal decision-making strategies. Unlike supervised learning, where a model is trained on labeled datasets, and unsupervised learning, which explores patterns within unlabeled data, reinforcement learning involves an agent learning through trial and error. The primary goal is to generate an optimal policy that dictates the agent's actions in various states of the environment, maximizing cumulative rewards. Reinforcement learning stands out for its ability to learn without predefined rules or explicit knowledge about the environment. The learning process involves the agent receiving feedback in the form of rewards or penalties based on its actions, leading to the reinforcement of favorable behaviors and the avoidance of detrimental ones. This iterative learning approach distinguishes reinforcement learning from other machine learning paradigms, making it well-suited for scenarios where explicit guidance or extensive labeled datasets may be impractical or unavailable.

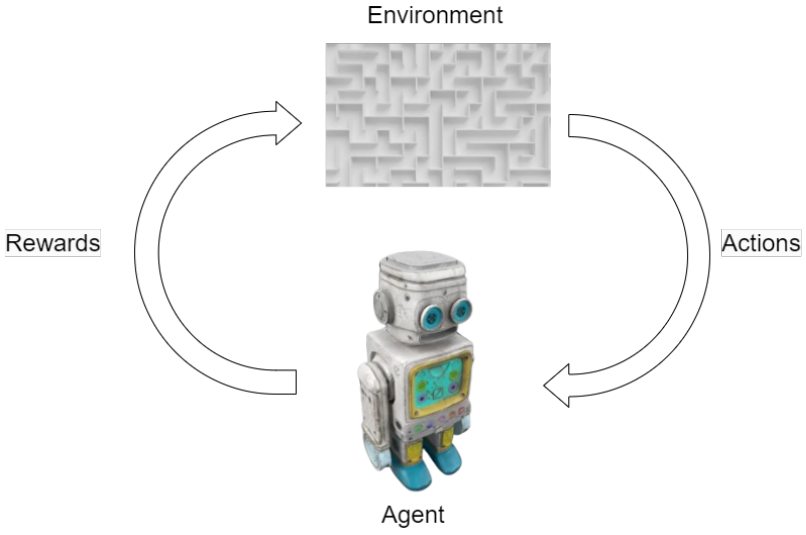


**Figure 3.4:** Reinforcement Learning

### 3.2.2 Q-Learning

Q-learning stands as a model-free reinforcement learning technique, dedicated to assessing the value of a specific action, denoted as A, within a given state, represented by S.

$$Q(s,a) = Q(s,a) + \alpha \cdot \Big(R + \gamma \cdot \max_{A} Q(s', A) - Q(s,a)\Big) \tag{3.3}$$

- Q(s, a) Represents the Q-value for a specific state s and action a, indicating the expected cumulative reward.
- $\alpha$ Denotes the learning rate, controlling the impact of new information on the existing Q-value.
- R: Signifies the immediate reward obtained after taking action a in state s.
- $\gamma$ Represents the discount factor, influencing the importance of future rewards.
- $\max_A Q(s', A)$ Denotes the maximum Q-value among all possible actions a' in the next state s', representing the expected cumulative future reward.

Q-learning determines the best policy for any finite Markov decision process by maximizing the anticipated value of the total reward across all steps, beginning from the present state. The function "Q" denotes the expected rewards for an action performed in a specific state, which is computed by the algorithm.

### 3.2.3 DQN

Deep Q Network (DQN) assumes a pivotal role in shaping the agent's decision-making processes and navigation strategies. DQN, a form of reinforcement learning, forms the backbone of our system's intelligence, leveraging a combination of sensory inputs from a 2D camera, ultrasonic sensor, GPS, and magnetometer. In essence, DQN operates as a neural network that learns to map state-action pairs to corresponding Q-values, representing the expected cumulative reward for taking a particular action in a given state. In our project, the DQN algorithm employs this fundamental principle to navigate and learn optimal strategies for the agent in its

dynamic environment. By continually updating its Q-values through interaction with the environment, DQN enables the agent to adapt its behavior, ensuring effective decision-making and autonomous navigation. This integration of DQN underscores its significance in providing the embodied agent with the cognitive capabilities essential for sophisticated and adaptive exploration of its surroundings.

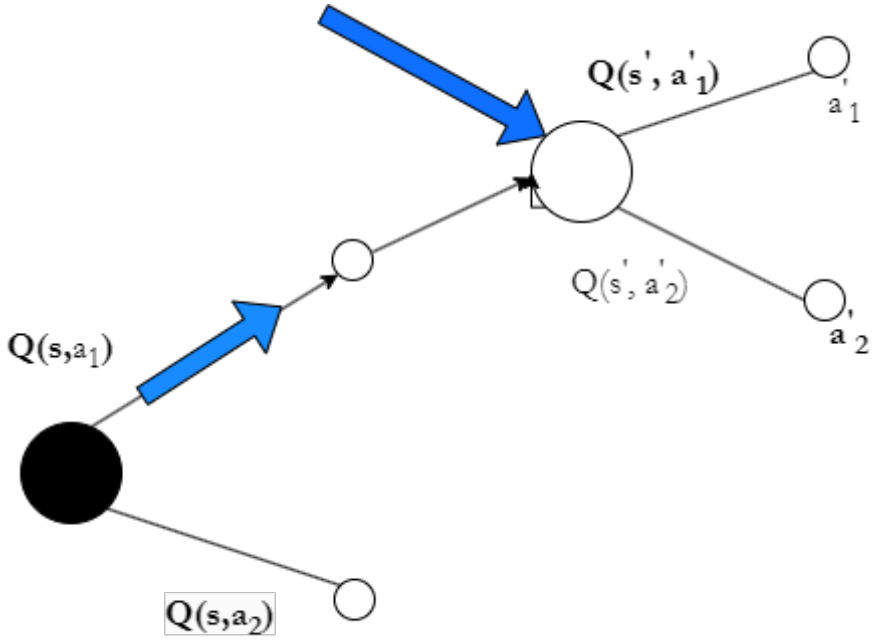


**Figure 3.5:** Deep-Q-Network

### 3.2.4 Manual Search Algorithms:

Manual Search Algorithms are employed when the distance between the agent and the target exceeds the precision of GPS coordinates. The following path-finding algorithms can be utilized:

- Flood Fill: A technique that systematically explores all connected cells to find a path from the agent to the target.
- Depth First Search: A graph traversal algorithm that explores as far as possible along each branch before backtracking.
- A* Search: A heuristic-based algorithm that intelligently explores the most promising paths by considering both the cost to reach a node and an estimated cost to the target.

- Breadth First Search: graph traversal algorithm that systematically explores all the vertices at the current depth level before moving on to vertices at the next level
- Wall Following Algorithm: A method where the agent follows walls to navigate through maze-like structures.
- Hamiltonian Search: Hamiltonian search, on the other hand, systematically explores the immediate vicinity by finding a cycle that passes through each nearby location exactly once.

By employing these search methods, autonomous agents can navigate with fine-grained control and effectively reach their targets within the limited range of GPS precision.

### 3.2.5 Logistic Regression

Logistic regression is a statistical method used to predict the probability of a binary outcome (e.g., yes/no, pass/fail, win/lose) based on one or more independent variables. It is a powerful tool for classification tasks and is widely used in various fields, such as machine learning, finance, healthcare, and marketing.

Key characteristics of logistic regression:

- Binary outcomes: Logistic regression is primarily used for predicting binary outcomes, although extensions exist for multi-class classification.
- Probabilities: The output of logistic regression is a probability between 0 and 1, representing the chance of the event occurring.
- Interpretability: Compared to other machine learning models, logistic regression is relatively interpretable. The coefficients of the model can be used to understand the relative importance of each variable and the direction of their relationship with the outcome.
- Limitations: Logistic regression assumes a linear relationship between the independent variables and the logit of the outcome. This may not be valid

for complex relationships, and other models may be more suitable in such cases.

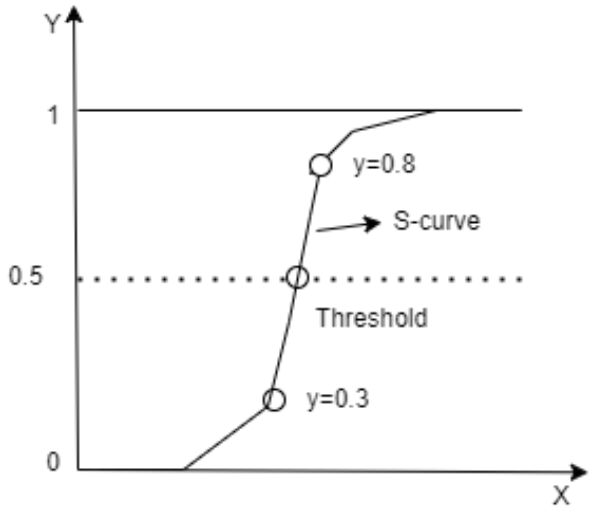


**Figure 3.6:** Logistic Regression

### 3.2.6 Decision Tree Classifier

A decision tree classifier is a versatile and widely used machine learning algorithm designed for both classification and regression tasks. Operating on the principle of recursively partitioning the input space based on feature conditions, decision trees construct a tree-like structure where each internal node represents a decision based on a specific feature. These decisions lead to subsequent branches, ultimately resulting in terminal nodes that represent the predicted outcome. Decision tree classifiers excel in capturing complex relationships within the data and are particularly effective in scenarios where interpretability and transparency of decision-making processes are crucial.

Key characteristics:

- Interpretability: Decision trees are highly interpretable, as their structure can be easily visualized and understood.
- Non-parametric: They don't make assumptions about the underlying data distribution.
- Handles mixed data: Can handle both numerical and categorical features.

- Prone to overfitting: Can overfit the training data if not properly pruned or regularized.
- Sensitive to feature scaling: May be biased towards features with more distinct values.

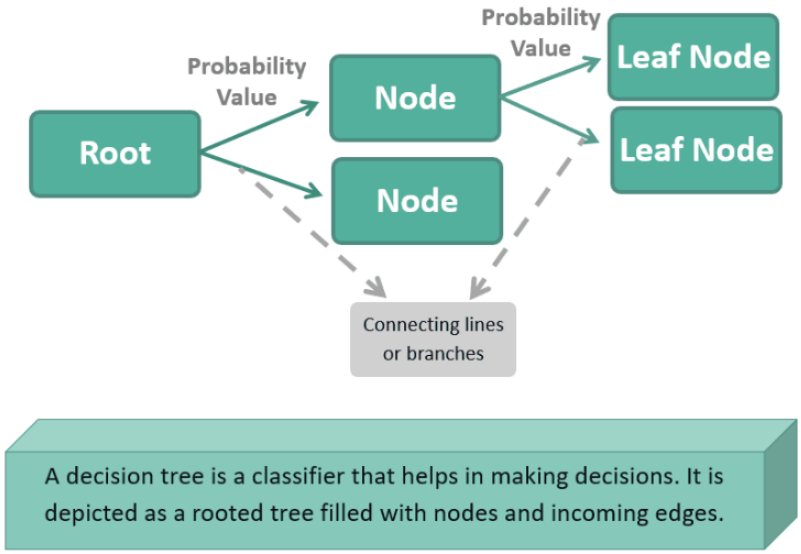


**Figure 3.7:** Decision Tree Classifier[1]

### 3.2.7 Markov Decision Process

A Markov Decision Process (MDP) is a mathematical model used for decision-making in situations where the outcome is uncertain. It is a discrete-time stochastic control process that provides a mathematical framework for modeling decision-making in situations where an agent takes actions in an environment, and the outcome depends not only on the agent's actions but also on random factors.

Here are the key components of a Markov Decision Process:

- States (S): A finite set of possible situations or conditions that the system can be in. The system is assumed to be in one of these states at any given time.
- Actions (A): A finite set of possible actions that the agent can take. The agent selects an action to influence the system.

- Transition Probabilities (P): The probabilities of moving from one state to another based on the chosen action. These probabilities capture the stochastic nature of the system.
- Rewards (R): A numerical value associated with each state-action pair, representing the immediate benefit or cost incurred when the agent takes a particular action in a specific state.

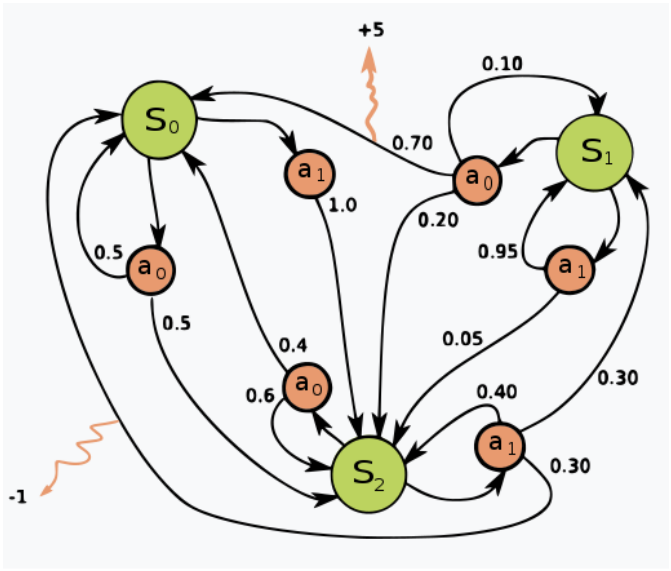


**Figure 3.8:** Markov Decision Process[2]

### 3.3 State-action Pair

A state-action pair reflects a specific circumstance within an environment, as well as the action that an agent does in response to it. States provide all important information about the environment's present configuration, including location, velocity, and any other relevant characteristics. Actions, on the other hand, refer to the decisions or options accessible to the agent while interacting with the environment, which may include motions, selects, or any other type of interaction. The combination of a state and an action indicates that the agent intends to transition from the present state to a new one by performing a specific action. In order for agents to make decisions that maximize long-term rewards or accomplish specific goals within the environment, it is essential for them to comprehend state-action pairings, which serve as the foundation for learning optimum policies or

value functions in reinforcement learning algorithms. Agents constantly learn to traverse and adapt to complicated environments by investigating various state-action pairings and their outcomes; this gradually improves the agents' capacity for making decisions.

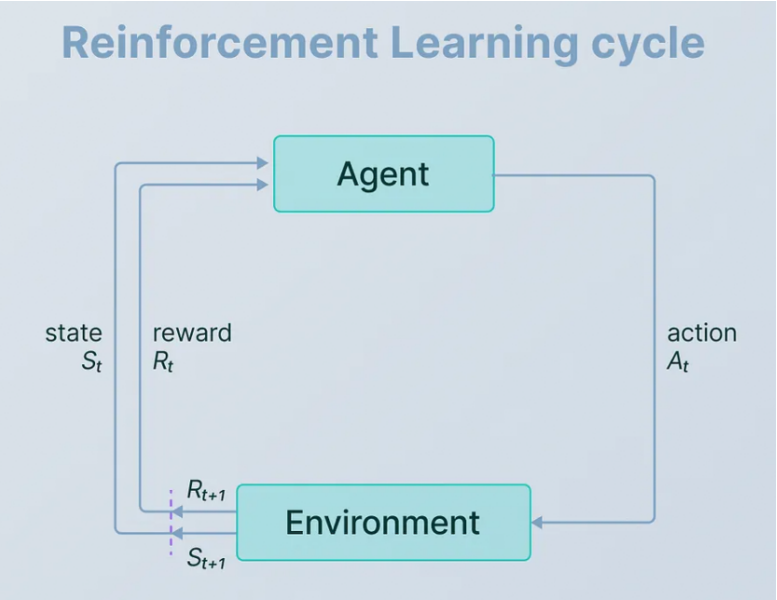


**Figure 3.9:** State-action pair

# CHAPTER 4

# METHODOLOGY

## 4.1 System Block Diagram

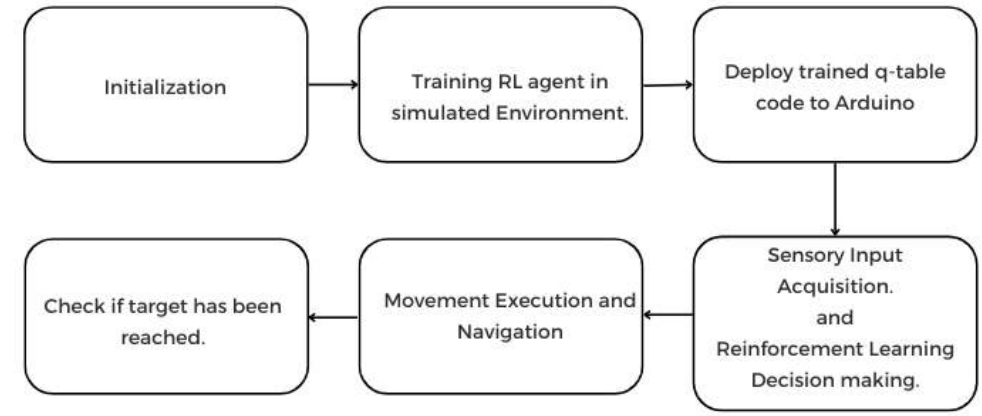


**Figure 4.1:** System Block Diagram

### 4.1.1 Initialization

At the onset of the project, the environment is initialized to provide a controlled setting for the training and testing of the RL agent. This involves defining the maze, defining the obstacle, setting up the parameters like "learning rate", "exploration rate", and "exploration decay factor", as well as defining the "initial state" and target of the agent within this environment. We defined actions as: 0-> "turn left", 1-> "turn right" and 2 -> "move forward" and actions as three binary numbers. Each binary number represents a path/obstacle towards the left, or right, and forward directions respectively. e.g. state 101 represents: "obstacle towards left", "path towards right" and "obstacle towards forward".

### 4.1.2 Training RL agent in simulated environment

The training is performed in a simulated environment using Q-learning. We have used three binary states indicating path or obstacle in three directions: left, right,

and forward. We also used three actions: "turn left", "turn right" and "move forward". Three binary states, combined with three actions each give us:

$$\text{No. of state action pairs} = 2^3 \cdot 3 = 24 \tag{4.1}$$

The state-action pairs are stored in q-table. Using a learning rate of 0.05, a discount factor of 0.9, an initial exploration rate of 0.1, and an exploration rate decay of 0.90, our agent learned to navigate the environment around just 5 epochs.

### 4.1.3 Deploy trained Q-table code to Arduino

Once the RL agent has completed its training and the Q-table has been obtained, the trained model is deployed onto the Arduino micro controller. The Q-table is then converted into 2D-matrix with 8 rows and three columns. Each row represents a unique state and each column represents an action. State is obtained from ultrasonic sensor and Q-table stores floating-point values indicating the predicted reward for selecting specific action at certain state.

### 4.1.4 Sensory Input Acquisition and Reinforcement Learning Decision Making

The Bot utilizes three ultrasonic sensors to get the distance towards the front, left, and right. Distance greater than 30 cm is path (represented by 0) and distance less than 30 cm is obstacles (represented by 1). So each sensor gives one of two states: path (0) or obstacles (1). Binary states from three ultrasonic sensors are then used to get the action with the highest value from the Q-table. The resultant action is then executed in actual hardware.

| State | Left | Right | Forward |
|---|---|---|---|
| 000' | -0.1456754 | 0.0289921 | -0.2658059 |
| 001' | 15.500594 | -0.15679 | -0.5751383 |
| 010' | -0.5 | -0.76106 | 19.256544 |
| 011' | 16.33089 | -2.5948408 | -2.4742242 |
| '100' | -0.8521801 | -0.7006457 | 12.950534 |
| 101' | -2.2842126 | -2.7767851 | -2.6859972 |
| 110' | -1.6150457 | -1.8485001 | 12.09658 |
| 111' | -2.7116605 | -3.1331679 | -3.1829111 |

**Figure 4.2:** Q-table

Above figure shows the trained Q-table we have obtained from training in simulation. For instance, If we wanted to find action at state 110 which represents obstacle on left, obstacle on right and path towards forward, first we would find row with state=101 in Q-table. In row with state 110, third column have the highest Q-value. So, it is the action chosen by Q-table, which represents action: "move forward".

### 4.1.5 Movement Execution and Navigation

Based on the action given by the Q-table, specific functions are invoked for executing actions. The pre-defined functions implement code to: "turn left", "turn right", "move forward" and "turn using magnetometer". "move forward" action rotates both wheels clockwise propelling the bot forward. For the "turn left" action, the left motor is rotated anti-clockwise and the right motor is rotated clockwise direction. For the "turn right" action, the right motor is rotated anti-clockwise, and the left motor is rotated in a clockwise direction. The "turn using magnetometer" function fetches the angle the agent is pointing towards. It calculates "deviation from the left" and "deviation from the right" between "target angle" and "heading angle". If "deviation from the left" is less than "deviation from the right", it is aligned to the target rotating towards the left. Otherwise, it is aligned to the target rotating towards the right.

### 4.1.6 Check If The Target Has Been Reached

Throughout the navigation process, the agent periodically checks whether it has reached its target destination. A target is said to be reached if it faces obstacles from three directions. If the target has been successfully reached, the navigation task is completed, and the agent halts its movement. Otherwise, it continues to navigate towards the target until it successfully reaches its objective.

## 4.2 System Flow Diagram

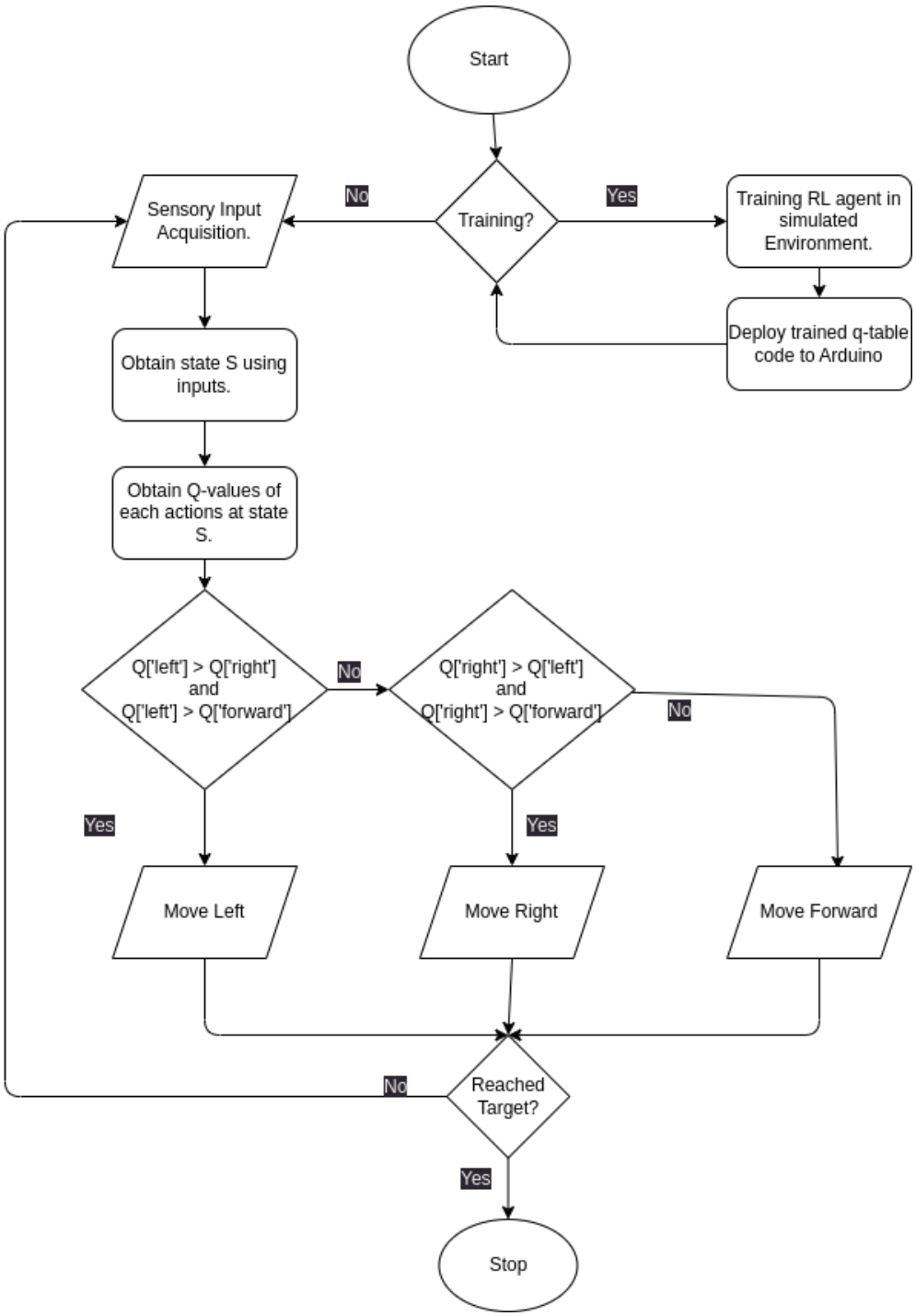


**Figure 4.3:** System Flow chart

### 4.2.1 Training Phase

(a) Training RL-Agent:

The RL-agent is trained in a simulated environment using Q-Network.

The training begins with 100% exploration rate and is decreased over a number of iterations by decaying the exploration rate. Q-value of state S for action A is updated using Bellman Equation.

(b) Deployment to Arduino:

The trained Q networks are flattened and deployed to Arduino for navigation in maze environment.

#### 4.2.2 Navigation

(a) Sensory Input Acquisition:

Three distances (left distance, right distance and front distance) are taken as input from three ultrasonic sensors.

(b) State Calculation:

States S is obtained distances given by ultrasonic sensors. There are three states and each state is either 0 or 1. Any obstacle within 30 cm gives state of '1' and state '0' if there is no obstacle within 30 cm which indicates the path.

(c) Action Calculation:

State S and actions A (left, right, forward) are used obtain value of each action from Q-table. The action with highest Q-value is performed using hardware.

(d) Performing Action:

Actions given by Q-Table are implemented controlling two motors. For "forward" action, both motors are rotated in clockwise direction. For "left" action, left-motor is rotated anti-clockwise and right-motor is rotated clockwise. For "right" action, right-motor is rotated anti-clockwise and left-motor is rotated in clockwise direction.

(e) Reached the target:

The RL agent checks if the target is reached. Target in our case is the end of maze where it gets obstacles in all three sides. If target is reached

the agent stops.

(f) Iteration and Continuous Operation: Steps "sensory input acquisition" to "Performing the Action" are repeated iteratively until the agent reaches the goal. It continually updates its state based on sensor inputs, makes decisions using Q-table, and navigates towards the destination. The RL-agent is stopped on reaching the destination, indicating completion of navigation task. This repetition ensures that the robot dynamically adapts to its environment and effectively achieves its navigation objectives.

### 4.3 Snake Game Environment

Creating a replica of the traditional Snake recreation using the Pygame library serves as an important element of our venture. By enforcing this familiar and extensively identified sport surroundings, we offer a tangible and engaging platform for experimenting with and trying out diverse aspects of autonomous navigation algorithms. The grid-primarily based format of the sport, where the Snake navigates to acquire Apples whilst heading off collisions, mirrors the demanding situations faced via independent dealers in actual-global navigation scenarios. This replication allows us to simulate and study navigation behaviors, collision avoidance techniques, and choice-making tactics within a controlled and interactive placing.

Our game implementation starts offevolved with the initialization of the game window the use of Pygame, observed through putting in the display. The game environment functions a Snake represented by using connected segments on a grid, with randomly positioned objectives. A game loop continuously updates the sport country, handling player inputs, Snake movement, collision detection, and map generation. Player inputs, captured through keyboard activities, enable control of the Snake's motion in 4 instructions. During every generation of the game loop, the Snake's head moves inside the current course, with frame segments following its direction. Collision detection occurs via checking for overlaps between the Snake's head coordinates and target coordinates or limitations. Upon reaching a goal, the Snake's length increases, and a new goal is placed at the grid. This implementation allows for interactive exploration and trying out of navigation algorithms inside a

familiar gaming environment.


**Figure 4.4:** Maze of the snake environment

## 4.4 Reinforcement Learning in Simulated Environment:

### 4.4.1 Snake Game with Reinforcement Learning:

Our project aims to develop an intelligent agent trained on the Snake Game using reinforcement learning techniques. The Snake Game map is be designed to be never-ending, meaning that as the agent travels, new map segments will be generated along the edges of the existing map within the observation window.

In the simulated Snake Game environment, we integrate Q-Learning algorithms with various path-finding algorithms, including Flood Fill, Hamiltonian Search, and A* Search, to train an agent (the robot car) to navigate efficiently. First, we define a reward function that incentivizes the agent to reach the charging point (target) within the Snake Game. This reward function assigns a high positive reward for reaching the target and a negative reward for collisions with obstacles or boundaries. Next, we train the agent using the acquired data and reinforcement

learning algorithms to optimize its navigation policy. During training, the agent learns to make decisions based on the current state and available actions, updating its Q-values accordingly to maximize cumulative rewards over time. By integrating Q-Learning with path-finding algorithms and optimizing the agent's navigation policy through reinforcement learning, we aim to develop an autonomous agent capable of efficiently navigating the Snake Game environment to reach its target.

### 4.4.2 Mapping Actions to Agent Controls:

Within the Snake Game environment, we map the actions (L: Move left, R: Move right, U: Move up, D: Move down, S: Stop) to the corresponding functions controlling the embodied agent. We establish a connection between these game actions and the actuators or motor controllers of the robot car, allowing the physical embodiment of the agent to mimic the movements of the virtual snake in the Snake Game by executing the mapped actions. This connection enables real-time translation of game actions into physical movements, ensuring synchronization between the virtual and physical environments and facilitating seamless interaction between the agent and the game world.

### 4.4.3 Navigation with Reinforcement Learning:

Within the Snake Game environment, we define a threshold distance to indicate when GPS signals may become less accurate, typically when the agent is within a certain proximity to obstacles or boundaries. Implementing a local search approach, we utilize various reinforcement learning algorithms to guide the agent (robot car) effectively from the threshold distance to the destination. By training the reinforcement learning agent to make optimal decisions using the defined actions within the Snake Game environment, it learns to navigate towards the charging point while adapting to changes in GPS accuracy and environmental conditions. This approach ensures robust and efficient navigation even in scenarios where GPS signals may be unreliable, enhancing the agent's overall performance and adaptability in dynamic environments.

#### 4.4.4 Iterative Evaluation and Fine-tuning:

To evaluate the performance of the robot car within the real-world Snake Game environment, we collect feedback on its navigation accuracy, stability, and real-time performance. This feedback is used to iterate on the system design, reinforcement learning algorithms, and navigation strategies, aiming to address any shortcomings and enhance overall performance. Through iterative refinement, we fine-tune system parameters and algorithms to improve navigation accuracy, stability, and real-time performance of the robot car within the Snake Game. This iterative process allows us to continuously optimize the system, ensuring that the robot car operates effectively and efficiently in diverse environments and scenarios.

### 4.5 Circuit Diagram

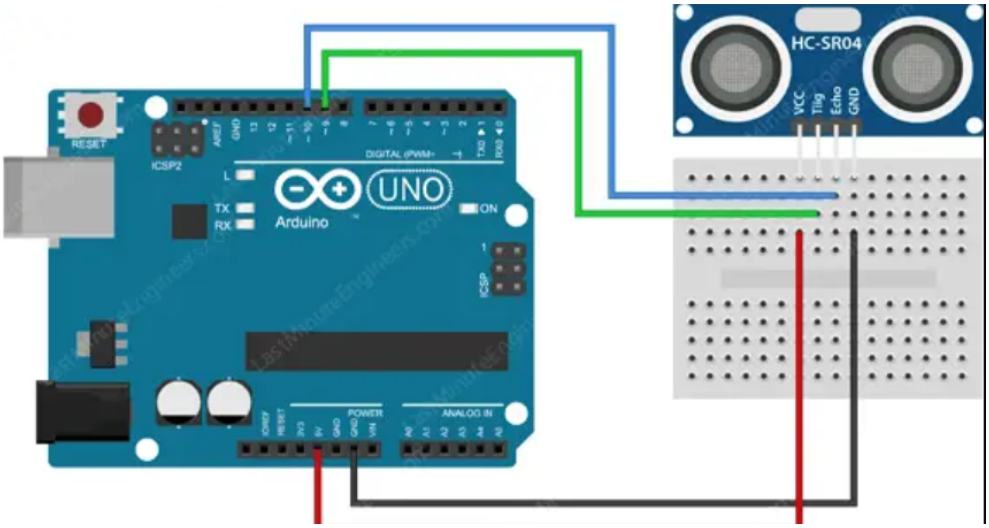


**Figure 4.5:** Arduino UNO with Ultrasonic Sensor

(a) Chassis and Motors

Car chassis with wheels and motors DC motors.

(b) Arduino UNO

Arduino UNO used to connect all the sensory inputs with calibration of the motor.

(c) Motor Driver

Motor driver module to control the DC motors.

(d) Ultrasonic sensors:

Three ultrasonic sensors are used each facing: Left, Right and Forward. Two facing left are used for left and right distance from two walls for path correction so as to make it move it in the center of path.

(e) Power Supply

Battery pack or power source for the motors (e.g., 12V battery or Li-Po battery)

(f) Mechanical Components

Mounting brackets, nuts, bolts, and hardware for assembling the chassis, motors, and sensors

(g) Cables and Connectors:

Jumper wires, male-female connectors, and soldering equipment

(h) Programming:

Libraries for motor control, sensor reading

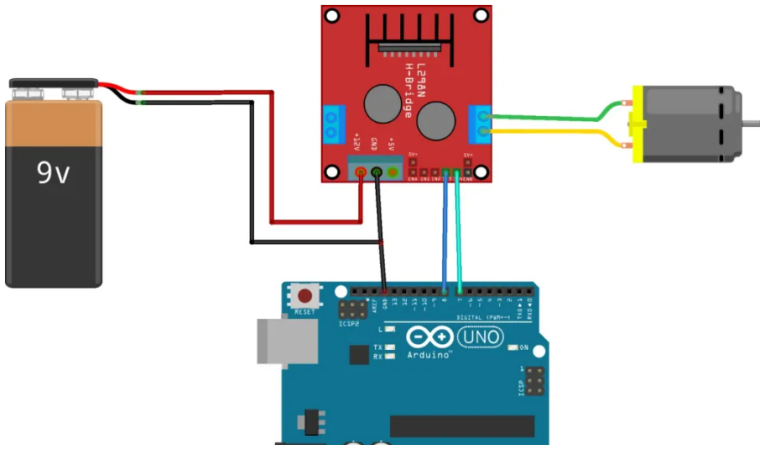


**Figure 4.6:** Arduino UNO with Motor Driver

## 4.6 Alternative Approaches

### 4.6.1 Leveraging Visual Data for Deep Q-Networks (DQN) Training

Utilizing camera output for training and implementation of Deep Q-Networks (DQN) involves leveraging unique features extracted from images as states.

However, this approach can be computationally intensive due to the complexity of processing visual data. Additionally, it often requires a large amount of real-world data to effectively train the DQN model. To address these challenges, we repeat the training process iteratively, refining the DQN architecture and hyperparameters to optimize performance while managing computational resources efficiently. We also employ techniques such as data augmentation and transfer learning to enhance the model's ability to generalize from limited real-world data. By continuously iterating on the training process and adapting the model to the specific requirements of the task, we aim to develop a robust and efficient DQN-based system capable of effectively utilizing camera output for navigation and decision-making tasks.

### 4.6.2 Sensor Fusion for Enhanced Navigation

Utilizing a GPS sensor and magnetometer for state representation includes integrating data from each sensor to facilitate navigation tasks. The source and purpose locations are described through GPS coordinates, presenting latitude and longitude values. The magnetometer yields the perspective , indicating the orientation relative to the magnetic north. Distance (r) between the current location and the destination is computed using the GPS coordinates. By combining distance and attitude (r, $\theta$) acquired in polar coordinates, vectorized coordinates (X, Y) derived in rectangular coordinates form. Below figure illustrates different states within our college premise. We repeat this process iteratively, refining the utilization of sensor information to optimize navigation overall performance in numerous environments and situations.

The learning process may be carried out in simulation the usage of Q-networks or Deep Q-Networks (DQN). However, there are numerous barriers related to depending completely on GPS sensors for navigation. Firstly, the accuracy and precision of GPS sensors range due to atmospheric conditions, boundaries together with buildings and timber, that could have an effect on the reliability of role estimation. Secondly, GPS receivers commonly have a constrained update rate, that means that function information won't be up to date

regularly enough for real-time navigation. Lastly, GPS cannot penetrate walls, leading to unreliable performance for indoor navigation duties. However, combining them with other sensors like camera, lidar and other models like path avoidance, point to point navigation in novel environment could be achieved.

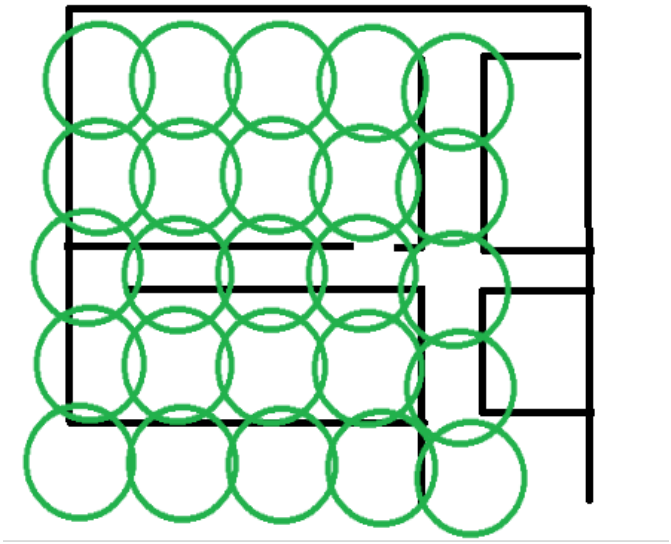

**Figure 4.7:** Precision circles

### 4.6.3 Our Implementation

In our implementation, we trained the RL agent using Q-Learning in a simulated maze environment, where it learned to make decisions based on given states. Subsequently, we flattened the learned weights and embedded them into Arduino code to enable decision-making in the physical environment. For hardware integration, we employed three ultrasonic sensors positioned to face Forward, Left, and Right, from which binary states (0 or 1) were obtained. A state of '1' indicated the presence of an obstacle within a 30 cm range, while '0' denoted an open path. Thus, the agent received three states at each position, utilized to determine actions based on the trained weights from the simulated environment. Additionally, for effective left/right turning and path calibration, we utilized a magnetometer to complement the ultrasonic sensors. This approach enables robust navigation and obstacle avoidance in real-world scenarios, ensuring the agent's adaptability and performance in

dynamic environments.

Further improvements include incorporating camera input and GPS sensor data alongside ultrasonic sensors and magnetometer for comprehensive state representation. Additionally, adopting continuous rotation instead of sharp left/right turns for movement in non-rectangular paths enhances navigation fluidity. Lastly, training the agent in a fixed environment where it autonomously navigates while considering all states and actions, using the gathered data for testing purposes, enhances its adaptability and robustness.

```
maze = np.array([
    [0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1],
    [1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1],
    [1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1],
    [1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1],
    [1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1],
    [1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1],
    [1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1],
    [1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0],
    [1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0],
    [1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0],
    [1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0],
    [1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1],
    [1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0]
])
source = (0,0)
destination = (14,11)
facing="right"
```

**Figure 4.8:** Maze Matrix

Figure 4.7 shows the maze matrix along with the source, destination, and facing we have used to train the agent in a simulated environment. In above matrix, 0 represents the path and 1 represents obstacle.

# CHAPTER 5

# RESULTS

## 5.1 Turning Using Magnetometer

Turnings in our maze were at the right angle, so we used a magnetometer to turn left or right. For instance: if the agent is moving towards 100° and it has to turn right, the new angle should be 100 + 90 = 190°. so the agent is rotated until it faces 190°.

## 5.2 RL-agent in Simulated Environment

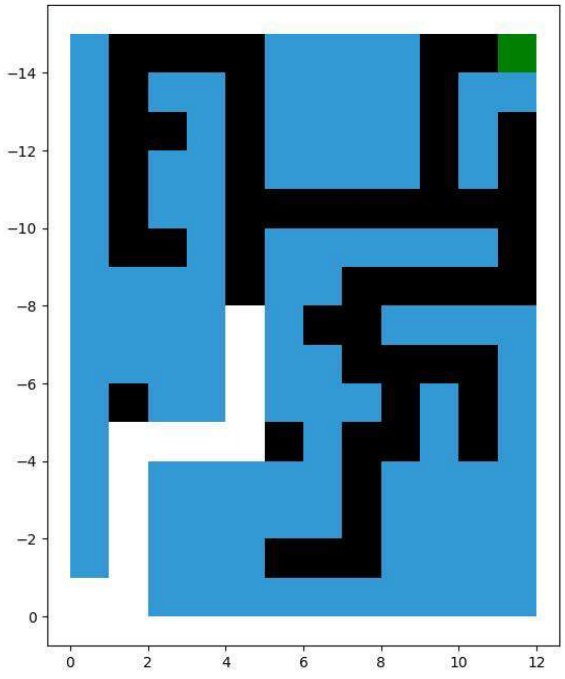


**Figure 5.1:** RL-agent in simulated environment

Given the map in matrix form, our agent can reach the goal. The black region indicates the path, the white region indicates the path it has travelled and the

green box represents the target. First, it initializes a Q-table Q(S, A) for storing values for each state-action pair. Initially it moves randomly as exploration rate is initialized to 1. Exploration rate gets discounted as number of episodes increases. Weights of Q-table are updated at the end of each episode using bellman equation. An episode ends when the agent reaches the goal. Bellman equation gives highest reward to final action and action before that gets discounted reward. Once it is trained, it can effectively reach to the goal.

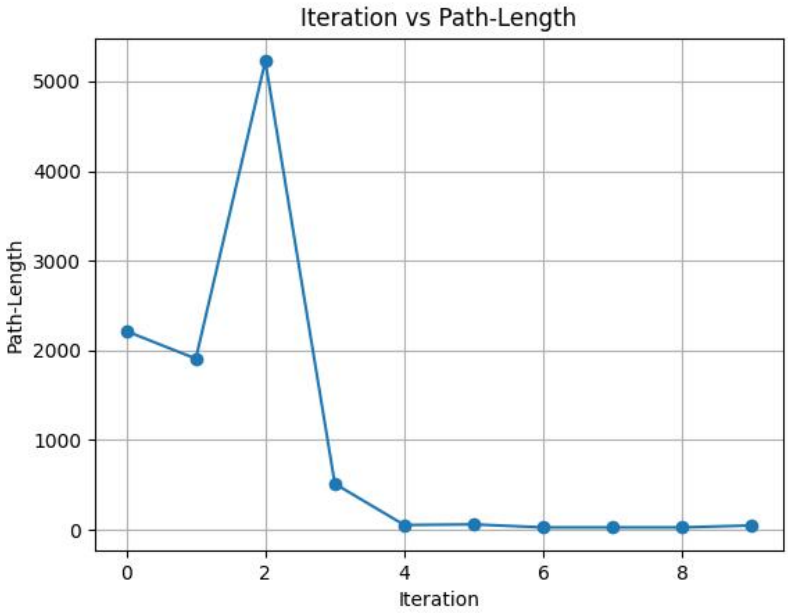


**Figure 5.2:** Episode vs path length

Figure 5.2 shows the plot between episode and path length. Initially the path length is high due to exploration and as number of epoch increases, path length is decreasing.

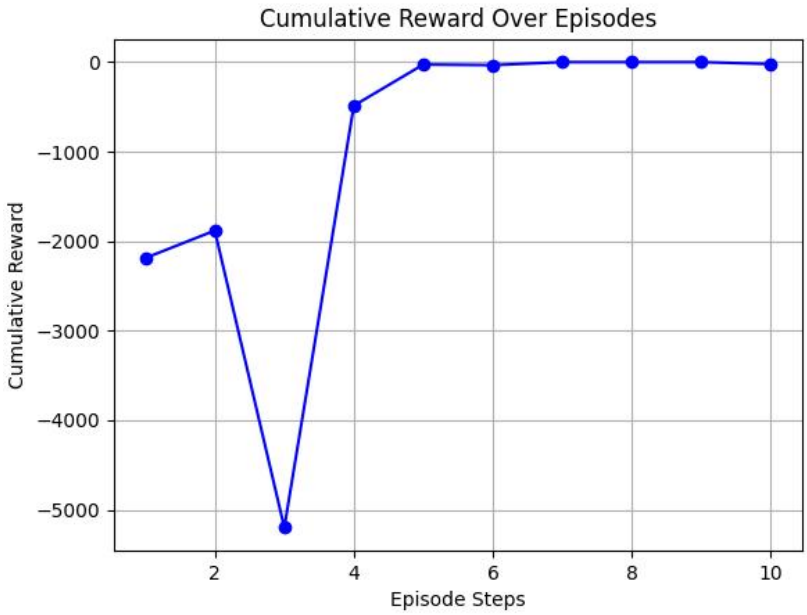


**Figure 5.3:** Cumulative Reward over Number of Episodes

Figure 5.5 shows a plot of Cumulative Reward over the number of Episodes. It shows increasing cumulative reward as the number of epochs increases. The rewards are constant after the fifth epoch which shows the completion of training. The training completed just over 5 epochs due to the small number of states (24 states) we have in our q-table.

## 5.3 RL-based Bot

We have implemented a RL-based Bot capable of navigating through a maze. It is able to reach the target when left at source using q-table continuously to get action at each state.

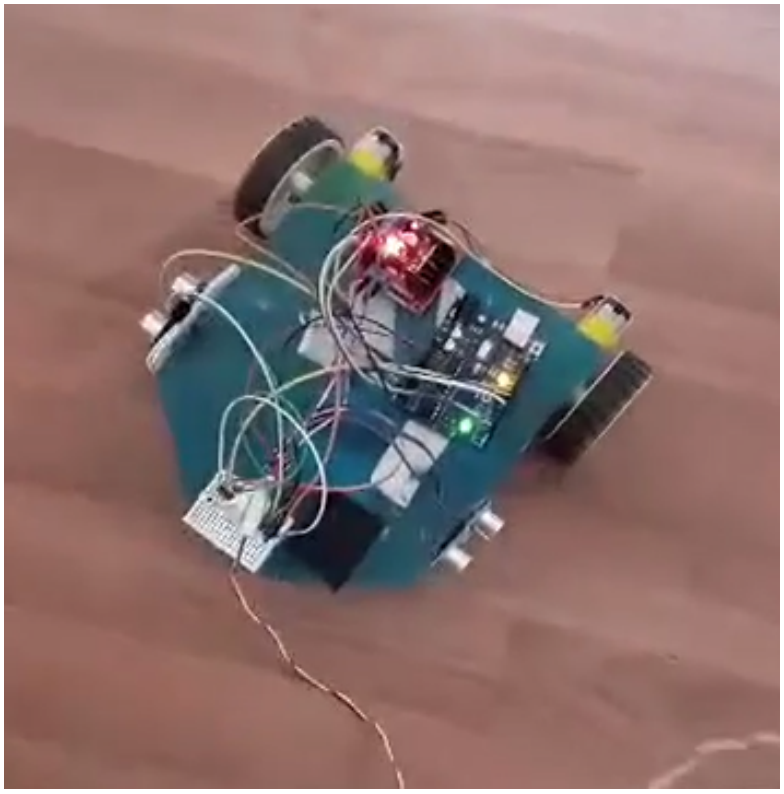

**Figure 5.4:** RL-agent

# CHAPTER 6

# EPILOGUE

## 6.1 Conclusion

The research has made adequate progress in constructing reinforcement learning (RL) agents that can navigate mazes in both simulated and real-world settings. Using Q-learning, these agents demonstrated outstanding learning and navigation ability in simulated environments, efficiently achieving their goals. After transitioning to physical mazes outfitted with ultrasonic sensors, the agents displayed extraordinary adaptability and effectiveness in negotiating real-world problems. This significant achievement demonstrates the great potential of RL approaches for addressing real navigation issues in robotics and autonomous systems. The successful deployment of RL agents in many environments demonstrates not only their robustness, but also their scalability and applicability to real-world circumstances.These findings shed light on the effectiveness of RL-based navigation systems and lay a solid platform for future developments in the area. As the field of robotics and autonomous systems evolves, the incorporation of RL approaches promises to play a critical role in offering intelligent and adaptable navigation skills, opening the door for transformational innovation and applications.

## 6.2 Limitations

The limitations for this project are:

a. Challenges in turning: Low update rates for the magnetometer resulted in instances where the robot rotated in place instead of making turns. Furthermore, removing the magnetometer leaves the agent without a means to ascertain whether it executed a perfect turn or deviated from the intended direction.

b. Hardware Limitations: Unavailability of Raspberry Pi prevented implementa-

tion of Deep Q-Networks (DQN) for advanced navigation strategies, limiting exploration of sophisticated reinforcement learning techniques.

c. Insufficient Representation of Environment: We used three ultrasonic sensors that gives binary input (path or not) to the agent, which is an insufficient representation for the maze let alone a real-world natural environment. It also have no way of knowing whether the bot has made a perfect turn or deviates from it.

## 6.3 Future Enhancement

The possible future enhancements for this project are:

a. Hardware Upgrade: Upgrading to Raspberry Pi for enhanced computational power would enable the implementation of advanced algorithms like Deep Q-Networks (DQN) and data from the camera or LIDAR for navigation would ensure a better representation of the environment than using ultrasonic sensors. With increased processing capabilities, the Raspberry Pi can efficiently handle complex calculations and training iterations required by DQN.

b. Deep Q-Networks (DQN): Incorporating DQN into the navigation system would optimize navigation decisions by leveraging the real-time learning capabilities of the Raspberry Pi. By continuously updating the navigation policy based on environmental feedback and reinforcement learning, the system can adapt to dynamic changes in the environment.

## REFERENCES


[1] R Priyadarshini, Mukil Alagirisamy, and N Rajendran. Medchain for securing data in decentralized healthcare system using dynamic smart contracts. pages 574–586, 2022.

[2] Wikipedia contributors. Markov decision process — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Markov_decision_process&oldid=1194955572`, 2024. [Online; accessed 25-February-2024].

[3] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[4] Fadi AlMahamid and Katarina Grolinger. 2021 ieee canadian conference on electrical and computer engineering (ccece). 2021.

[5] Christopher Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[6] Joris Dinneweth, Abderrahmane Boubezoul, René Mandiau, and Stéphane Espié. Multi-agent reinforcement learning for autonomous vehicles: A survey. *Autonomous Intelligent Systems*, 2(1):27, 2022.

[7] Song Yan Ho. Reinforcement learning for self-driving cars. 2018.