CUTTING EDGE REINFORCEMENT LEARNING MODEL IMPLEMENTATION ON AUTONOMOUS ROVER

Group Members

FAIZ UL HASAN GARDEZI	2019123
JAHANZAIB KHAN LUDIN	2019187
ZAIN UL FURQAN SHAHID	2019554
AWAIZ ADNAN	2019095

Faculty of Computer Sciences and Engineering

Ghulam Ishaq Khan Institute of Engineering Sciences and Technology

2023

Certificate of Approval

It is certified that the work presented in this report was performed by Jahanzaib khan Ludin, Zain ul furqan shahid, Faiz ul hasan garzedi and Awaiz adnan under the supervision of Dr Farhan Khan. The work is adequate and lies within the scope of the BS degree in Computer Science/Computer Engineering at Ghulam Ishaq Khan Institute of Engineering Sciences and Technology.

---------------------	---------------------
Dr Farhan khan	Engr Badre Munir
(Advisor)	(Co-Advisor)

Dr. Ahmar Rashid

(Dean)

ABSTRACT

This Final Year report describes the successful implementation of Reinforcement Learning (RL) techniques on an autonomous rover to enable it to navigate through an unknown environment while avoiding obstacles and finding the optimal path to a target location. The project utilized a Deep Q-Network (DQN) RL algorithm and LIDAR sensors to train the rover to make decisions based on rewards received from the environment. The rewards were based on the distance traveled and the number of obstacles avoided. The project achieved positive results, with the rover being able to navigate through the environment and reach the target location while avoiding obstacles. The evaluation metrics included success rate, distance traveled, and time taken to reach the target, and these indicated that the rover's performance improved with more training episodes and better sensor configurations. The project demonstrates the potential of RL techniques for developing autonomous rovers with obstacle avoidance and pathfinding capabilities, which can be used in real-world scenarios such as space exploration and search and rescue missions.

ACKNOWLEDGEMENTS

We want to express our deepest appreciation to all those who have provided us with support throughout our Final Year Project (FYP) on the implementation of reinforcement learning on autonomous rovers with obstacle avoidance and pathfinding capabilities.

Our first and foremost thanks go to our supervisor Dr. Farhan, whose unwavering guidance, advice, and support have been indispensable in the success of this project. His extensive knowledge and expertise in the field of robotics and reinforcement learning have been instrumental in shaping the project and pushing it to new heights.

We are also indebted to the faculty members who have imparted to us the necessary skills and knowledge to accomplish this project. Their guidance and expertise have been invaluable in helping us realize our full potential and accomplish our objectives.

We appreciate the opportunity to undertake this project and hope that the outcomes of this research will contribute to the advancement of robotics and autonomous systems.

TABLE OF CONTENTS

LIST OF FIGURES

1. CHAPTER

INTRODUCTION

Autonomous rovers are gaining popularity due to their ability to operate and explore hazardous and challenging environments without human intervention. However, in order for an autonomous rover to operate effectively in an unknown environment, it must be capable of making intelligent decisions and adapting its behavior to changing conditions. Reinforcement learning (RL) has emerged as a powerful technique for developing intelligent decision-making capabilities in autonomous systems.

The aim of this final year project is to develop and implement an RL-based control system for an autonomous rover that can navigate and operate in unknown environments. Reinforcement Learning (RL) is a type of machine learning in which an agent learns how to behave by interacting with an environment through a process of trial and error. In RL, the agent receives feedback in the form of rewards or punishments as it takes actions in the environment, and uses this feedback to learn the optimal behavior. The agent's goal is to maximize its long-term reward by selecting actions that lead to the greatest cumulative reward over time. Through this sequential decision-making process, the agent gradually improves its performance and learns to make better decisions in the future.

We are building a system whereby the sensor data is collected and a simulation of the environment is created in real time. Our Reinforcement learning model will then take this data as input tensors and perform iterations with our rover as a simulated agent which in turn gives us the optimal pathway. This pathway is then mapped onto the rover as output. This model will not need pre-trained data or training, hence can work in unknown and inhospitable situations. So, to demonstrate our model capability, we will be building a rover with custom designed electronics and hardware. Primary sensors are LIDAR and SONAR to work in sync to generate the simulation of nearby environment. The rover will have autonomous navigation, obstacle avoidance and geo-locating capabilities.

The project will be divided into several phases, including the design and construction of the autonomous rover, the development of the RL-based control system, and the evaluation of the system's performance in various environments. We will also conduct experiments to compare our RL-based approach with other state-of-the-art techniques for autonomous rover control.
The successful completion of this project will contribute to the advancement of autonomous systems and provide valuable insights into the use of RL for decision-making in unknown environments. This report presents the details of our approach, the experiments we conducted, and the results we obtained. We will conclude with a discussion of the implications of our findings for future research and development in this field.

2. CHAPTER

LITERATURE REVIEW

In reinforcement learning, an agent interacts with an environment to learn a policy that maximizes its reward. The environment represents the world in which the agent operates and provides feedback in the form of rewards or punishments based on its actions. The set of all possible situations in which the agent can find itself is known as states, which are used to determine the agent's behavior. Observations refer to the data that the agent receives from the environment, which may be a partial or noisy representation of the true state. The sequence of interactions between the agent and the environment, starting from an initial state and ending in a terminal state, is known as an episode and is used to learn a policy.

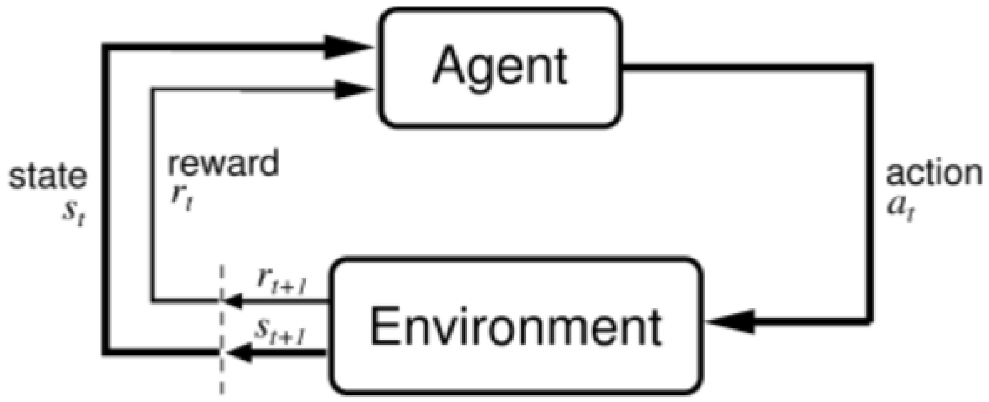


Figure 1: Reinforcement Learning Cycle

MARKOV DECISION PROCESS

MDPs provide a simple and effective way to model the problem of learning and decision-making through interaction. In this framework, there is an agent responsible for making decisions and learning from feedback, and an environment that provides the feedback and responds to the agent's actions. The agent and environment interact continuously, with the agent selecting actions and the environment presenting new situations based on those actions. The environment also provides rewards, which the agent aims to maximize over time by selecting the most appropriate actions. Overall, MDPs are a powerful tool for developing intelligent decision-making capabilities in autonomous systems.

More specifically, the agent and environment interact at each of a sequence of discrete time steps, t = 0, 1, 2, 3.... 2 At each time step t, the agent receives some representation of the environment's state, St 2 S, and on that basis selects an action, at 2 A(s).3 One time step later, in part because of its action, the agent receives a numerical reward, Rt+1 2 R ⇢ R, and finds itself in a new state, St+1. 4 The MDP and agent together thereby give rise to a sequence or trajectory that begins like this:

S0, A0, R1, S1, A1, R2, S2, A2, R3... (2.1)

In a finite MDP, the sets of states, actions, and rewards (S, A, and R) all have a finite number of elements. In this case, the random variables Rt and St have well defined discrete probability distributions dependent only on the preceding state and action. That is, for particular values of these random variables, s0 2 S and r 2 R, there is a probability of those values occurring at time t, given particular values of the preceding state and action:

p (s0, r|s, a). = Pr {St =s0, Rt =r | St1 =s, At1 =a}, (2.2)

Discount factor γ discounts the rewards of future states contributing to the cumulative reward. The discounted future rewards can be formulated with the formula below.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.2}$$

The value function can estimate the value of a state sequence start with state S, as shown follows:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t\right] \tag{2.4}$$

The optimal value function is the value functions that give the highest value for all states.

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{2.5}$$

Q-learning: Q-learning is a reinforcement learning mechanism that compare the expected utility of available actions given a state. Q-learning can train a model to find the optimal policy in any finite MDP. The Bellman equation suggests the Q-value function as shown in equation (2.6)

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a') \tag{2.6}$$

The Q-value function, expressed in equation 7, differs from the value function in equation 3 in that it estimates the maximum future reward for taking a particular action in a given state. The Q-value function takes into account the maximum discounted future reward that an agent can obtain by transitioning from state s(t) to the next state s(t+1)′. By using a learning rate α and the estimated utilities of the current state s(t) and the next state s(t+1)′, the Q-value function can be iteratively updated to converge towards the optimal Q-value function. This is achieved by computing the difference between the estimated utilities of the current and next state at time step t, as shown in equation 2.7.

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right) \qquad (2.7)$$

Then, Q-learning function can substitute the value function in equation 1.5, as shown below.

$$V^*(s) = \max_a Q^*(s, a) \qquad (2.8)$$

The optimal policy can be retrieved from the optimal value function with equation 2.9.

$$\pi^*(s) = \arg\max_a Q^*(s, a) \qquad (2.9)$$

DEEP Q-LEARNING

Neural network is utilized as the function approximator of the Q-learning algorithm to form the Deep Q-learning network (DQN). The learning goal of DQN is to find the best settings of the parameter θ of $Qs,a;\theta$ or the weights w of the function approximator. The objective of DQN is to minimize the Mean Square Error (MSE) of the Q-values. Besides that, experience replay technique uses first-in-first-out(FIFO) method to keep experience tuples in replay memory for every time step. The replay memory stores experience tuples for several episodes to ensure that the memory holds diversified experiences for different states. The experience tuples are sampled from replay memory randomly during Q-learning updates. The implementation of experience replay can remove correlations in the observation sequence and smoothing over changes in the data distribution by randomizes over the data. In practice, the most recent N experience records are sampled uniformly and randomly from replay memory. The random sampling gives equal probability to all experience tuples.

EXISTING SYSTEMS

Looking at the literature, there are a couple of works that tried to accomplish these capabilities using methods such as Ant Colony Optimization (ACO) [1], Genetic Algorithm (GA) [2], Particle Swarm Optimization (PSO) [3], Fuzzy Neural Network (FNN) [4], Learning from Demonstration (LfD) [5] and Reinforcement Learning (RL) [6]. However, in this paper we are interested in methods that are based on machine learning methods and specifically RL [7]. For example, Lee et al. [8] trained a network for their RL agent so their quadruped robot could avoid obstacles. In Kominami et al. [9], an RL agent is used in combination with virtual repulsive method by a multi-legged robot to tackle obstacles using its tactile sensors. Zhang et al. [10] used model predictive control (MPC) to train a RL agent to learn to fly a quad-copter; MPC is necessary for training but not necessary for the testing phase. In Zhang et al. [5], the Gaussian mixture model (GMM) and Gaussian mixture regression (GMR) are used for learning from demonstration LfD with the goal of avoiding obstacles. Sadeghi and Levine [11] used deep reinforcement learning (DRL) to make a drone explore and avoid obstacles using a monocular camera. A Deep Q-Network (DQN) [12]-based method was used in [6,13] to train their robot to explore autonomously and avoid obstacles while they used initialized weights for their network (weights are generated using a supervised learning method). Smolyanskiy et al. [14] used off-the-shelf hardware to design a deep neural network (DNN) called TrailNet for following a trail. In [15], we have developed an algorithm that learns from scratch and in an autonomous way using RL and Multi-Layer Perceptron (MLP) to explore autonomously. Some of the aforementioned works are using mono-chrome cameras and use a network in front of the mono-chrome camera to convert the RGB image to a depth image. Some other works use external devices such as Vicon system for the generation of reward and state, which emphasizes the fact that the system needs the external systems for training or even working. Furthermore, there are works that use DNNs, and works that use continuous action space, and works that use a real robot in order to implement their algorithms. Nonetheless, works that use a combination of all the aforementioned advantages are harder to find. Thus, our purpose is to design and implement an algorithm that works with DNNs, without

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

the help of external systems and by using a sensor fusion that provides noise-resistant data for our robot so our algorithm can focus on learning autonomously to explore and avoid obstacles using discrete and continuous actions.

3. CHAPTER

DESIGN AND SPECIFICATION

The autonomous rover reinforcement learning project will involve several components, including the physical design of the rover, the reinforcement learning algorithm, and the integration of the algorithm with the rover's control system. The system will be designed to operate in hazardous and unknown environments, enabling it to explore and carry out tasks autonomously. The project will require the system to be adaptable to different environments, both indoor and outdoor, and reliable, capable of handling unexpected errors and adapting to changing conditions. The project will also require a scalable and modular system, allowing for easy upgrades and modifications. The specification for the project will outline the hardware and software requirements, such as sensors, processing units, reinforcement learning algorithms, and control systems. The reinforcement learning algorithm will be developed using a combination of traditional and deep reinforcement learning techniques and integrated with the rover's control system to enable the system to operate autonomously.

The autonomous rover reinforcement learning project will be built with several components, including the physical design of the rover, the reinforcement learning algorithm, and the integration of the algorithm with the rover's control system. The goal of the project is to enable the rover to operate autonomously in hazardous and unknown environments, performing tasks and exploring without human intervention. To achieve this, the system will need to be able to navigate and operate in various environments, be robust, reliable, scalable, and modular, allowing for easy upgrades and modifications.

The reinforcement learning algorithm will learn the optimal policy for decision-making based on a reward function, using exploration and exploitation techniques to discover new and effective strategies for navigating and operating in unfamiliar settings. The control system for the rover will execute the actions chosen by the reinforcement learning algorithm, ensuring safe and efficient operation using feedback control and motion planning techniques.

In summary, the autonomous rover reinforcement learning project aims to create a robust and reliable system that can operate in different environments and learn effective decision-making strategies through trial and error. The project will contribute to the advancement of autonomous systems and provide valuable insights into the use of reinforcement learning for decision-making in unknown environments

Phase 1: Problem Definition

In the first phase, literature review is done in the area of Deep Reinforcement Learning and autonomous rover to identify challenges and research gaps. Considering the previous work and identified gaps, research questions and objectives are defined to formulate the research problem.

Phase 2: System Design

For the second phase, different simulation environments and DRL techniques are studied. The system architecture is designed with the details of each component which is later used to perform and evaluate the experiments.

Phase 3: Implementation

Simulation environment and previously selected algorithms are implemented.

Phase 4: Experiments

To evaluate the performance of rover obstacle avoidance strategy using different algorithms and state and action spaces, various experiments are designed and conducted, and the data is collected for analysis.

Phase 5: Evaluation & Analysis

In the last phase, the data from previous phase is analyzed based on mainly cumulative reward and success rate to evaluate the performance of autonomous rover in different scenarios. Lastly, findings of these experiments are communicated and shared using this thesis.

System Architecture

The architecture of the autonomous rover system that utilizes deep reinforcement learning is depicted in the figure, which was utilized to conduct experiments in this study. The system consists of three main components: simulation, OpenAI Gym environment, and the implementation of Stable Baselines-based deep reinforcement learning algorithms. All of the modules employed in the system, including the tools, libraries, and frameworks, are open source.

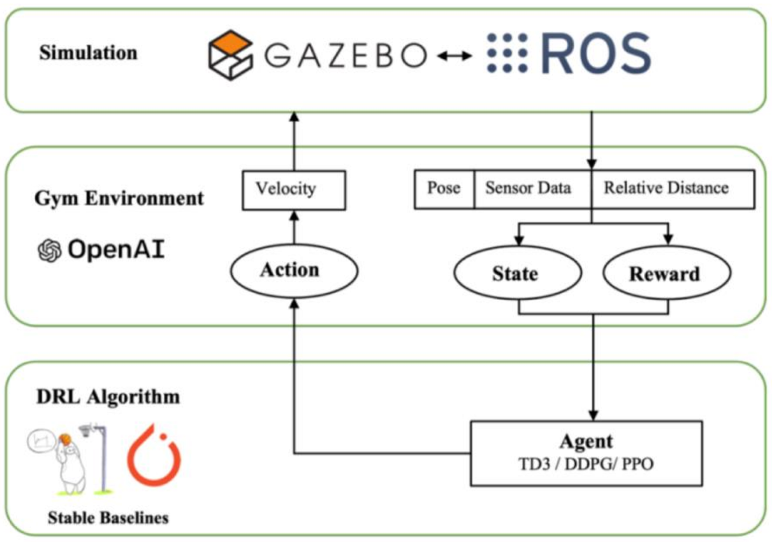


Figure 2:Project Overview

SIMULATION

Reinforcement learning involves the agent interacting with the environment repeatedly to learn the optimal policy through trial and error. However, it is not practical to use a physical robot for training, hence simulators are used to replicate the real-world environment where the robot will operate. The simulation module in the architecture consists of two components, namely Gazebo and ROS, which provide an environment for the agent to learn and interact with. This approach enables safe, cost-effective, and efficient training of the agent before deploying it in the real world.

Gazebo is a 3D robotics simulator that comes with a built-in physics engine, making it a powerful tool for simulating robot behavior. It offers a wide range of environments and models that can be used to design indoor and outdoor settings. It also provides support for a variety of plugins that can be used to generate sensor data, such as camera, LiDAR, and sonar. Turtlebot 3 was chosen as the robot for simulation, as it has extensive documentation available and simplifies the development and troubleshooting process. The simulation component of the overall system is essential for training the reinforcement learning algorithm, as it is impractical to train a real robot in a physical environment due to the cost and potential hazards involved. Additionally, the integration of Gazebo with other applications such as OpenAI Gym and Stable Baselines enables quick creation, training, and testing of different robotics control reinforcement models using Python.

Robot Operating System (ROS) is an open-source middleware platform with large set of tools, libraries, drivers, and algorithms to develop robotics applications both in simulation and real-world scenarios. ROS uses distributed publish / subscribe mechanism to communicate between different processes / programs, called nodes. A node publishes a message to

4. CHAPTER

PROPOSED SOLUTIONS

Simulation environment is started using ‘ROSlaunch’ command to open launch file which opens gazebo simulator, loads the defined simulation world and corresponding models, and spawns the Turtlebot in the gazebo simulation environment / world. To communicate with Gazebo and ROS in order to get the robot’s odometry data, obtain sensor data, publish velocity commands and to reset the simulation environment ‘rospy’ python library is used. The rospy library client API allows us to initialize and launch a ROS node, establish communication with ROS and Gazebo by publishing and subscribing to relevant topics using python code.

Once the simulation is up and running, pose and sensor data obtained using rospy is preprocessed and passed to DRL learning agent through OpenAI Gym ‘*step’* function which takes ‘*action’* as an input and returns state and reward values to the learning agent. Learning agent which is the DRL uses the state and reward values to optimize the model and predict action values for the given state. Predicted action (velocity in this case) is then published to the simulation using rospy API and the robot performs the given action. This process repeats for each timestep, which is whenever the step function is called.

The work flow of our algorithm is shown in Figure 3 and pseudocode in algorithm 1. The input to the algorithm is the preprocessed sensory inputs, which contains LIDAR sensor and range sensors signals. These input data shape the state of MMQN reinforcement learning module where they make a decision with attention to this state and its policy to find the best action. After taking an action and running it, the state of the algorithm will be changed to a new state. In MMQN, we are using an epsilon greedy policy; as a result, the decision of which action is the best is according to the value of each action in a particular state where it is calculated by MMQN ANN module. The MMQN ANN module acts as a function approximator for MMQN reinforcement learning module. Thus, the state of MMQN reinforcement learning module is the input for MMQN neural network module, and the output is the value of each action in that particular state. After selecting an action and moving to a new state, a reward will be calculated based on the sensor inputs, which can be interpreted as the response of environment to the action selected by the MMQN algorithm. Further, this reward will be used to update the value of the action just selected in our MMQN ANN module. In the following sections, we explain each process in detail.

```
Randomly initialize network $Q(s, a|\theta^Q)$
Initialize target network $Q'$ with weights $\theta^{Q'} \leftarrow \theta^Q$
procedure SENSOR FUSION($DepthImage, RangeSensors$) ▷ this procedure will generate our input for the short term memory
    if (each RangeSensors Value <0.5 m) then
        DepthImage values[at the related positions] = particular RangeSensors Value   ▷ if all RangeSensors value <0.5 then all the DepthImage values changes
    end if
    TransformedVector← TransformDepth(DepthImage)          ▷ Transform DepthImage into a vector of 80
    return TransformedVector
end procedure
procedure REWARD($RangeSensors, a$)                        ▷ this procedure will generate our reward
    $R_{t_{external}} = -0.3 \times (RangeSensor_{left}) + -0.5 \times (RangeSensor_{center}) + -0.3 \times (RangeSensor_{right})$
    $A_{forward} = 0, A_{turning} = 0$
    if (a = forward) then
        $A_{forward} = 1$
    else if (a = turning) then
        $A_{turning} = 1$
    end if
    $R_{t_{intrinsic}} = 0.5 \times A_{forward} + -0.25 \times A_{turning}$
    $R = R_{t_{external}} + R_{t_{intrinsic}}$
    return R
end procedure
procedure SHORTMEMORYSTATE($s$)        ▷ this procedure will generate our state which is a short term memory
    $s[1..7] \leftarrow s[0..6]$
    $s[0] \leftarrow SensorFusion$
    return s
end procedure
procedure MAIN($DepthImage, RangeSensors$)                                   ▷ main procedure
    while $t<infinity$ do                                       ▷ this procedure continues for ever
        $s_t \leftarrow ShortMemoryState(s_t)$
        Select action $a_t = a_{max}Q(s_t, a_t|\theta^Q)$
        Wait for robot to move to the new position
        $s_{t+1} \leftarrow ShortMemoryState(s_{t+1})$
        $r_t \leftarrow Reward$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in ReplayMemory
        Set $y_i = r_i + \gamma Q'(s_{i+1}, a|\theta^{Q'})$          ▷ With a batch of size 32 from ReplayMemory
        Update Q network by minimizing the loss: $L = \frac{1}{N}\Sigma_i(y_i Q(s_i, a_i|\theta^Q))^2$     ▷ With a batch of size 32 from ReplayMemory
        Update the target network, $Q' \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
        $r \leftarrow a \bmod b$
    end while
end procedure
```

Figure 3: MMQN algorithm overview

Q-learning

In the MMQN algorithm, a model-free off-policy method called Q-learning is utilized. This choice is made due to the absence of a model of the environment, and the intention to employ the experience replay method for training a multilayer neural network known as the Q-network. By opting for a model-free approach, the algorithm does not rely on a predefined understanding of the environment's dynamics.

Q-learning operates on the principle of temporal difference reinforcement learning. It employs an epsilon greedy policy to select actions. The objective of such policies is to maximize the cumulative discounted reward over time. This is achieved through an iterative process of selecting the action with the highest value in each state and updating its value based on the action value in the subsequent state, along with the reward obtained from the environment.

In this context, it is assumed that future rewards are discounted by a factor (gamma) per time step. As a result, the future discounted reward at a given time (t) can be calculated.

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \tag{4.1}$$

The optimal action value function is defined as the maximum expected return attainable by following any strategy after taking a sufficient number of actions 'a' in state 's'. This function represents the highest expected cumulative reward that can be achieved by selecting the most advantageous actions in each state, considering the dynamics of the environment and the rewards it provides. It captures the optimal decision-making policy in order to maximize the long-term rewards.

The optimal action value function is based on Bellman equation

$$Q^*(S_t, a_t) = Q(S_t, a_t) + \alpha.\Big(R(t+1) + \gamma.\max_a Q\Big(S_{(t+1)}, a\Big) - Q(S_t, a_t)\Big) \quad (4.2)$$

The formula mentioned above is commonly employed in various reinforcement learning algorithms to determine the optimal action value. However, these algorithms often encounter two primary challenges: the lack of generalization and the problem of the large number of states.

To address these challenges, as elaborated in the "Multilayer neural network" section, we utilize a function approximator to estimate the optimal Q-values for each action in a given state. This function approximator, denoted as Q*, allows us to generalize the learned values across similar states and reduces the computational burden associated with explicitly representing every possible state.

By employing a multilayer neural network as the function approximator, we can capture complex relationships and patterns in the environment. This enables us to approximate the optimal Q-values efficiently and handle a wide range of states, providing a more scalable and effective approach to reinforcement learning.

Sensor fusion

The autonomous rover is equipped with a lidar sensor capable of measuring distances ranging from 50 cm to 5 m. However, there may be a blind spot or unreliable measurements in close proximity to the robot. To address this issue, additional proximity sensors are utilized to accurately measure the depth in the robot's immediate surroundings.

The design architecture consists of three range sensors placed at different angles: -30 degrees, 0 degrees, and +30 degrees with respect to the robot's orientation. Each range sensor measures the distance between the rover and objects in its corresponding angle.

To achieve sensor fusion, the rover first obtains the depth sensor input from the lidar sensor. This depth image contains measurements from various distances but may have unreliable values within the range of less than 50 cm. To correct these unreliable depth values, the rover utilizes the readings from the range sensors. If any of the range sensors detect an object at a distance of less than 50 cm, the corresponding region of the depth image is updated with the value from the range sensor. This ensures that the unreliable depth values within the critical distance range are replaced with accurate readings from the proximity sensors.

By integrating the lidar sensor data with the proximity sensors' measurements, the autonomous rover can obtain a more reliable and comprehensive understanding of its immediate environment. This fused sensor data can then be utilized for reinforcement learning algorithms to make informed decisions and navigate effectively in various scenarios.

Top of Form

Bottom of Form

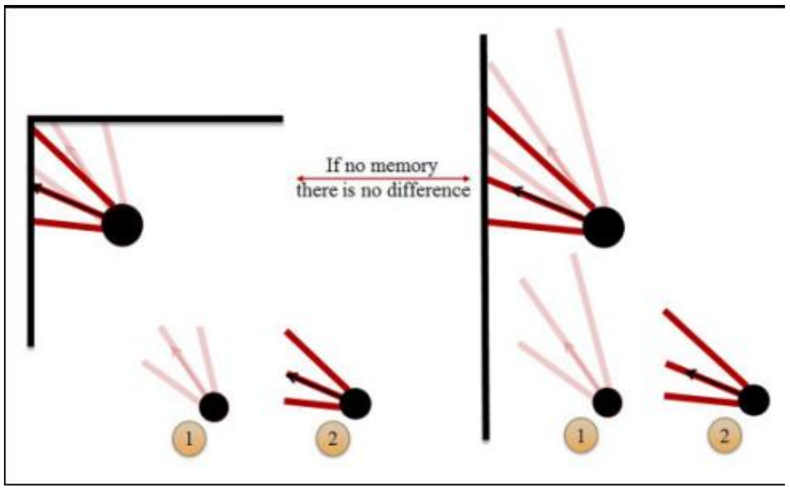


Figure 4 Different Scenarios

The above figure explains the different scenarios, Left scenario and right scenarios are different, however, only if the robot considers states 1 and 2 in each scenario. Otherwise, if the robot only considers state 2 of left and right scenarios, then both cases looks like a similar state.

In order to enhance our state, we mix several states and make a super state which actually is a short-term memory containing the recent n states that the robot has been in them as shown in the figure below

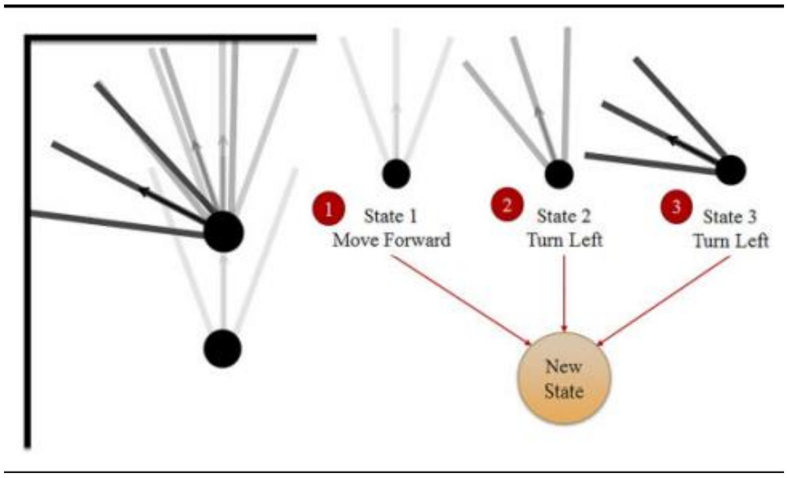


Figure 5 States

Memory-based method

Memory is a crucial aspect for intelligent agents or animals with learning capabilities. It allows them to retain information from past experiences, enabling them to make better decisions in the future. Similarly, our robot requires a memory mechanism to differentiate between similar states and make informed choices.

In this work, we employ a memory-based method to address this need. Specifically, we utilize a linear, constant short-term memory with varying sizes, such as 3, 5, 8, and 16. Among these sizes, the memory with a size of 3 is particularly emphasized.

By incorporating this memory mechanism, our robot can store and retrieve relevant information about previous states, actions, and rewards. This facilitates the ability to differentiate between similar states and aids in selecting optimal actions based on past experiences (figure below).

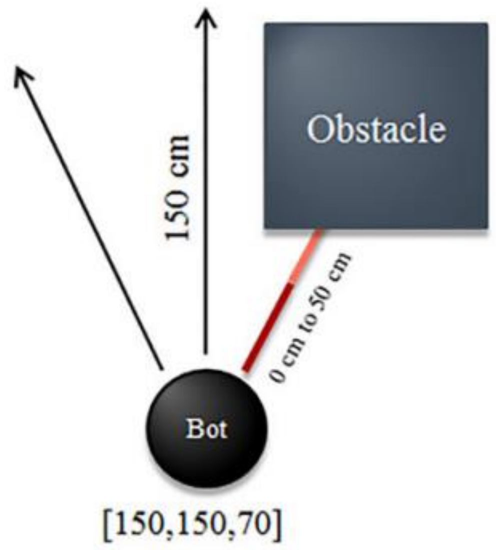


Figure 6 Reward Calculation

Employing a memory-based method allows the robot to learn from its interactions with the environment, leveraging the knowledge gained to improve decision-making and achieve better performance over time. By utilizing a memory mechanism, our robot can effectively utilize past experiences to inform future actions, leading to more intelligent and adaptive behavior.

Multilayer neural network

In the context of handling a large number of states, the traditional approach of storing action values in a table becomes impractical. This limitation becomes apparent when considering a state represented as an 80-dimensional vector, with each dimension ranging from 0 to 255. Consequently, the total number of states would be approximately 255^80, which is an enormous and unmanageable state space.

To address this challenge, we employ an Artificial Neural Network (ANN) as a function approximator instead of using a Q-table in our Q-learning algorithm. By utilizing an ANN, our algorithm gains the ability to generalize its learnings and make predictions in unseen states. The ANN we employ is a multilayer neural network with 400 inputs, representing our super state vector that includes a memory of size 5. The network has three outputs corresponding to the value of each possible action in a given state: left, right, and forward.

The multilayer neural network architecture consists of two hidden layers, the first with 512 neurons and the second with 256 neurons. Rectified Linear Unit (ReLU) activation functions are used in the first and second hidden layers, while the output layer employs a linear activation function. The network is fully connected, meaning that each neuron in one layer is connected to every neuron in the subsequent layer.

By utilizing this multilayer neural network, our algorithm can effectively approximate the Q-values for different states and actions. This approach allows for efficient handling of a large state space, facilitates generalization, and enables the agent to make informed decisions based on its learned knowledge.

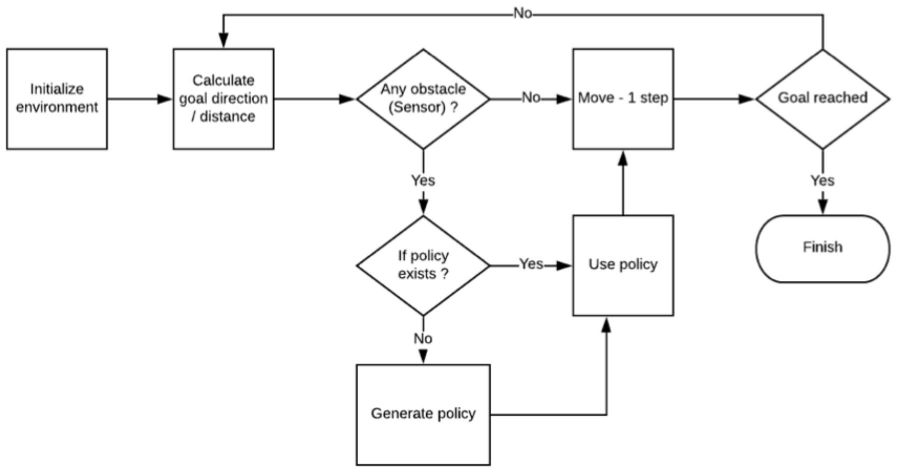


Figure 7: Decision Cycle

Evaluation

We prepared charts to analyze the performance of all algorithms executed in 200 independent episodes. Here are the results.

This chart represents results of dynamic policy algorithm.:

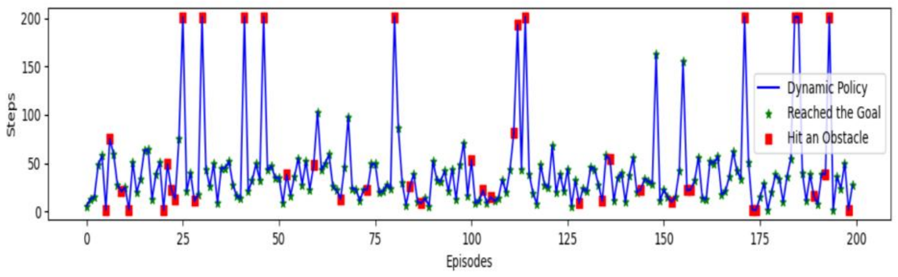


Figure 8: Dynamic Policy algorithm

This chart represents results of extended dynamic policy algorithm:

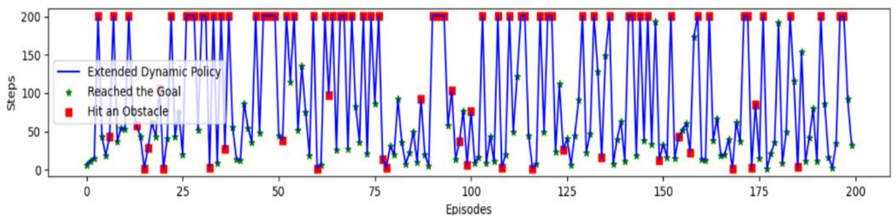
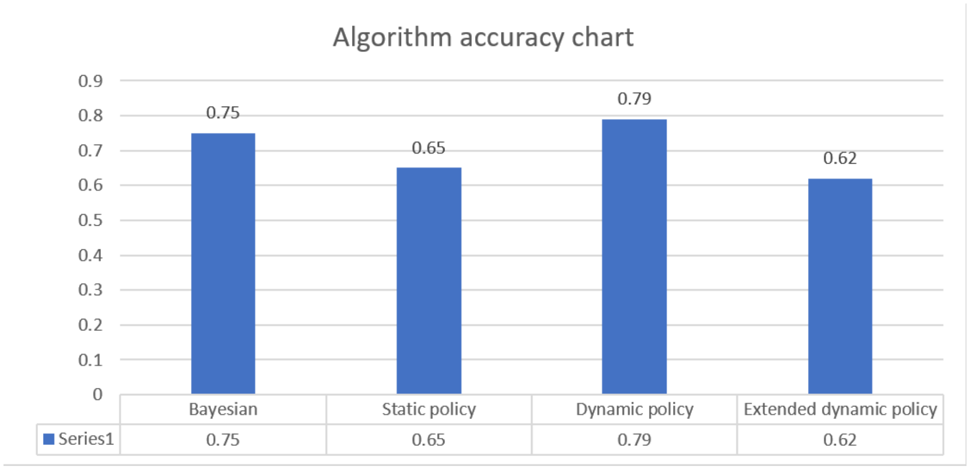


Figure 9: Extended Dynamic Policy Algorithm



	Bayesian	Static policy	Dynamic policy	Extended dynamic policy
■ Series1	0.75	0.65	0.79	0.62

Figure 10: Algorithm accuracy chart

Fig.1.6 The Dynamic Policy algorithm has the highest consistency among all the algorithm to navigate through obstacles. We tried to achieve the best performance using reinforcement learning algorithms.

SYSTEM REQUIREMENTS

- Lidar sensor/s: To help map the environment in a virtual simulation.
- Proximity sensor/s: The blind spots from the Lidar sensor will be detected by the proximity sensors.

- Arduino: This micro-controller is used to capture sensor data which will be converted to our required file format and will be sent over to raspberry pi.
- Raspberry pi (Pico): This micro-controller will further process the data and format it into a tensor. Which will be uploaded ESP.
- ESP32: ESP will send the data to the cloud

Functional Requirements

- **Sensor data Modeling**
  Model Data from the sensors and make a virtual environment in real time this is one of the key features as our model will be trained on this specific data set.

- **Reinforcement Learning**
  Our project utilizes reinforcement learning algorithm to train our rover to find the optimal state value pair or the optimal policy, So the algorithm is one of the key functional requirements.

- **Unmanned Rover**
  The suspension of our rover will be rocker bogy since we are accounting for unknown environments so we are considering the rugged terrain and any obstacles that may cause damage. The rover will also have hydraulic pumps on each wheel.

- **Cloud connection**

  One of the key requirements will be cloud computing since the microcontrollers are constraint and not well equipped to compute a large amount of data set which will be done through ESP.

- **GPS**

  Space-based satellite navigation system that provides time and location information anywhere.

- **Digital Maps**

  The process in which data collection is compiled and formatted in a virtual image.

- **Adaptive Cruise Control**

  Tracks distances to adjacent vehicles on the same lane. Detects objects in front of a vehicle at risk of emergency collision.

Overall Schematic Diagram

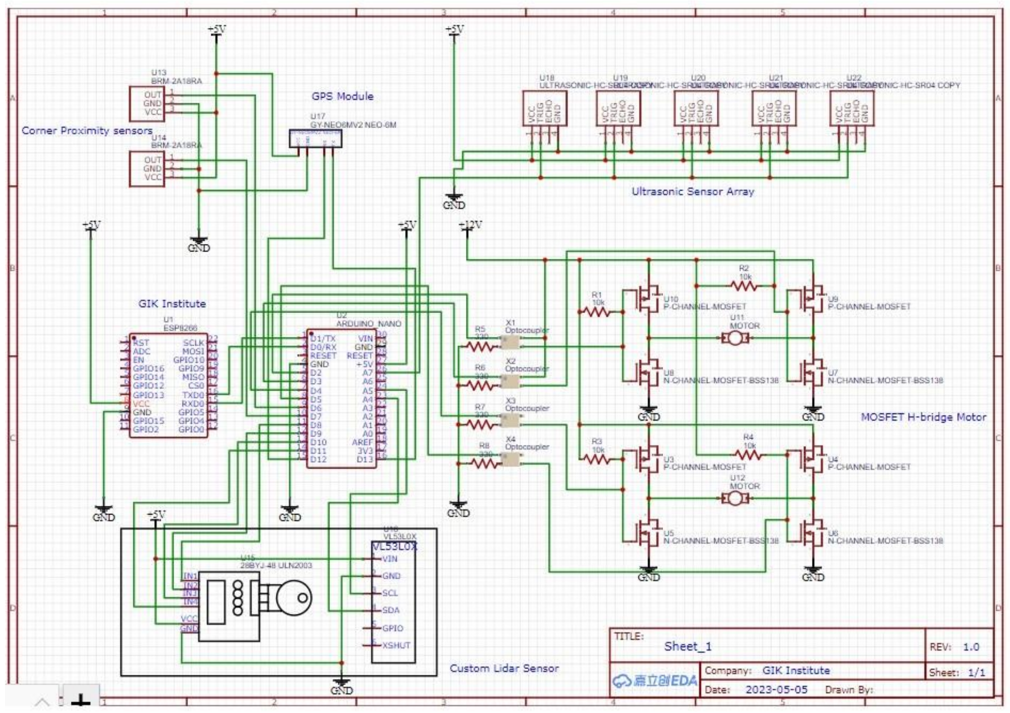


Figure 11: Overall schematic Diagram

Electronic Design

The electronic design is based on the Arduino Nano Microcontroller and ESP8266 Wi-Fi Module. The nano is interfaced with all the sensor arrays, LiDAR, proximity sensors, GPS module and motor controller. The ESP module communicates with the nano periodically through the serial BUS and uploads sensor data to cloud, while downloading any commands to be given to nano.

We have also designed a custom MOSFET based H-bridge motor driver that is electrically isolated via opto-couplers and can handle up to 20 Amps of current.

All components are soldered onto PCB and interconnections are done through jumper wires.

Sensor Stack

The rover employs the following sensors to map its surrounding environment.

- **Custom LiDAR Sensor**
  As we were unable to procure the required LiDAR sensor, we designed and built our own sensor from parts we had available. So our sensor consists of two main parts, the distance measuring chip and the rotating base for the chip. We used the vl5l0x laser distance sensor for distance measuring, it communicates via the I2C BUS and measured up to 2m in directional distance. For the rotating base we selected the bipolar stepper motor along with its ULN2003 based driver IC. The sensor works by taking 360 steps in the clockwise direction, then 360 steps in the counter clockwise direction to complete one scan. Each step is mapped to one degree of rotation and the sensor is sampled at each degree, hence after completing one scan we have 720 total reading from 360 to -360 degrees. The scanning rate is about 0.25 Hz, so each scan takes about 4 sec to complete.

- **Ultrasonic Sensor Array**
  The ultrasonic distance sensor array is built with 5 ultrasonic sensors (SR-HC04), placed in horizontal alignment. The working principle is based on the concept of phased array detection whereby each sensor in the array is pinged one after the other in a specific manner such that the wave interference of the signals from each sensor produces a directional

beam which scans across the width of the senor array and returns an average distance from the obstacle. Its angle of detection is 90 degrees from the normal in each direction.

- **Proximity Sensors**
  Two IR based proximity sensors are placed in the two most common blind spots of the rover, that is, the front corners of the vehicle. These avoid collision as they detect whether rover is close to any obstacle in the blind spot region.

- **GPS Module**
  To track the rover's location, speed and orientation we have used the NEO-6m GPS module and u-blox tracking software.

Hardware Design

The rover's physical dimensions are 35 cm by 70 cm and a total height of about 38cm. The rover has a rear wheel drive tri-wheeler design for better mobility and efficiency. The chassis is aluminum and wood based with the upper level made from hard acrylic. The wheels are soft padded for shock absorption and rubber grips have been installed for better traction.

5. CHAPTER

RESULTS AND DISCUSSION

DRL Algorithms Comparison

In order to evaluate the performance of different reinforcement learning algorithms for collision avoidance task, we trained several agents in the same environment for a minimum of 300,000 timesteps. We analyzed and compared their performance based on criteria such as cumulative reward, sample efficiency, and training time. Since different algorithms excel in different situations, this comprehensive comparison helped us identify the most suitable algorithm for our specific use case. Based on the results, we observed that algorithms based on actor-critic method such as TD3, DDPG, and SAC performed relatively better than policy based / policy gradient (PPO) and value-based (DQN) methods. This observation is supported by the cumulative reward vs timestep graph.

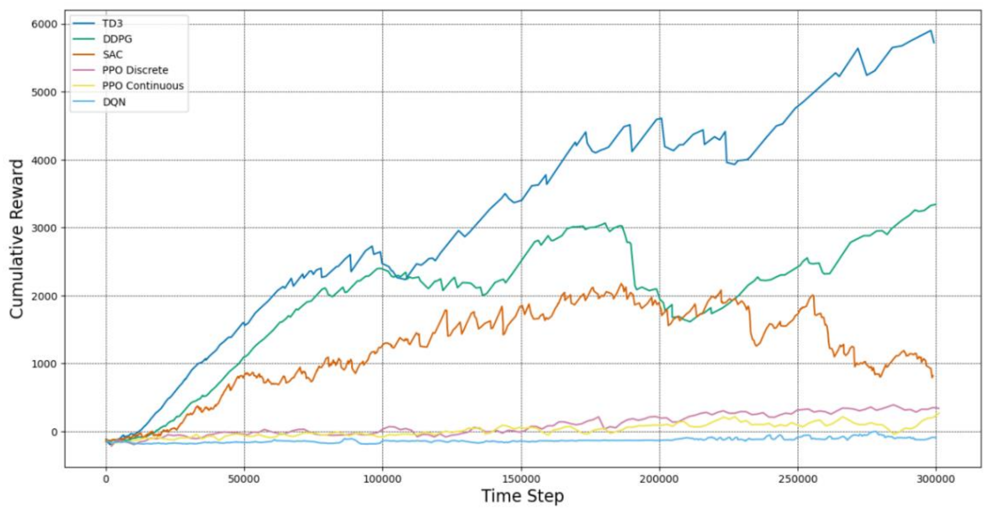


Figure 12: Cumulative Reward

The analysis of results shows that Actor-Critic method-based algorithms are more efficient in terms of samples as compared to Policy gradient method-based PPO algorithm and value-based DQN. The former algorithms are also off-policy, which means they can learn from data gathered by another policy, while PPO is an on-policy algorithm that learns from the current policy. Although PPO and DQN took less time to train up to 300,000 timesteps, their performance did not improve with further training.

SUSTAINABILITY ANALYSIS

People: A mobile robot which can navigate in unknown environments without the need of a map can be deployed in various scenarios such as: In hospitals for monitoring the patients and medicine delivery in isolated wards. In hazardous conditions they can be used to perform a particular task. Moreover, our results show that low-cost ranging sensors can learn robust navigation policy, hence affordable household robots can be developed which can be used for applications ranging from security monitoring to sweeping the floor.

Economic: Mobile robots are used in indoor environments for industrial automation such as in warehouses. Map-less navigation approach speeds up the deployment process of the mobile robots since time consuming process of building the map is not required and the robots can adopt to dynamic environments.

Environment: Our results shows that ranging sensor-based navigation approach is computationally less intensive as compared to vision-based approach. Moreover, the learnt policy can be deployed and used even if CPU is used for processing. Hence, an efficient approach will have less energy consumption and will help in conserving the battery life of the robot.

6. CHAPTER

Conclusions

The research presented and evaluated the application of Deep Reinforcement Learning (DRL) algorithms for autonomous navigation of mobile robots in unknown indoor environments. The proposed DRL algorithms demonstrated remarkable capabilities, allowing agents to learn how to interact with the environment without prior knowledge about it. These algorithms can be applied to various types of mobile robots, as they are model-free and do not require knowledge of the robot's kinematics and dynamics. Through simulations, the results showcased that the proposed navigation algorithms enable the mobile robot to autonomously navigate towards a target object location while avoiding collisions with both static and dynamic obstacles. Importantly, these algorithms eliminate the need for pre-developed environmental maps. Additionally, the use of an object detector model facilitates real-time recognition of the target object by the mobile robot.

A practical approach is proposed for collision avoidance and goal-oriented navigation tasks of mobile robots using DQN and DDQN agents. From the experiment, it has been seen that the DQN and DDQN agents trained in the Gazebo simulator can be deployed directly to the real mobile robot without tuning the parameters. The DDQN agent is more robust and better than the DQN agent in exploring the environment, avoiding collisions, and reaching the target object location from the simulation experiment in Gazebo. Thus, only the DDQN agent is used for real-world experiments in an unknown indoor environment. The real-world experiment is performed in two different environments to demonstrate that the deployed DDQN policy without a learning algorithm is capable of operating in the real world. In conclusion, the proposed method has great potential for autonomous mobile robot navigation compared with SLAM, however, the reward design is the challenging part of DRL for autonomous mobile robot navigation. In the real-world experiment, the final model configuration will have a single board computer (NVIDIA Jetson TX2), which has excellent processing speed with a hex-core CPU, a 256-core NVIDIA pascal GPU, and 8G LPDDR4 RAM, instead of a laptop as a controller unit.

In the future, we plan to make several improvements to our existing approach. One such improvement would be to propose a new reward function that could enable the evolution of different behaviors. We also aim to add new features to the environment, and deploy both the policy and learning algorithm on a real mobile robot, which would allow the agent to continue learning and achieve optimum performance in the real world. Additionally, we plan to make the environment of real-world experiments identical to that of the Gazebo simulation environment.

Another future improvement would be to design a Deep Deterministic Policy Gradient Agents (DDPG) algorithm to enable continuous action space, which would ensure smooth and continuous motion of the mobile robot. Furthermore, we plan to design the backend of a DRL algorithm from scratch, instead of using the DQN class of the stable baselines, in order to access all the callbacks such as loss, average max Q-value, and cumulative max Q-value. These improvements would help us to further enhance the performance of our approach in real-world scenarios. Some research work that will continue to integrate with this project are:

1. Multi-objective optimization: Designing a multi-objective optimization algorithm that can balance the trade-offs between different objectives such as collision avoidance, energy efficiency, and mission completion time.
2. Transfer learning: Developing a transfer learning framework that can transfer the knowledge learned in a simulated environment to the real-world environment.
3. Hybrid reinforcement learning: Combining reinforcement learning with other machine learning techniques such as supervised learning and unsupervised learning to improve the learning efficiency and generalization capability of the agent.
4. Multi-agent reinforcement learning: Studying the interactions between multiple autonomous rovers and designing a multi-agent reinforcement learning algorithm that can optimize the global reward while respecting the individual goals and constraints of each agent.
5. Safety constraints: Incorporating safety constraints in the reinforcement learning algorithm to ensure the safety of the rover and its environment.
6. Explainability: Developing an explainable reinforcement learning algorithm that can provide insights into the decision-making process of the agent and enhance the trust and interpretability of the system.

GLOSSARY

Reinforcement Learning (RL): A type of machine learning that focuses on teaching agents how to make decisions based on rewards and punishments.

Autonomous Rover: A vehicle that is capable of navigating and operating without human intervention.

Cutting-Edge: Refers to the most advanced or innovative technology currently available.

Model: A simplified representation of a system or process that can be used to predict or understand its behavior.

State: The current situation or configuration of the environment that the agent is in.

Action: A decision made by the agent to change the current state of the environment.

Reward: A signal given to the agent to reinforce desirable behavior and discourage undesirable behavior.

Exploration: The process of trying out new actions in order to learn more about the environment and discover better policies.

Exploitation: The process of using the knowledge gained from exploration to choose the best actions in the current situation.

Policy: The strategy or set of rules used by the agent to choose actions based on the current state.

Q-Learning: A popular RL algorithm that uses a table of Q-values to estimate the expected reward of taking an action in a given state.
Deep Q-Network (DQN): A neural network-based RL algorithm that uses a deep neural network to estimate Q-values.

Convolutional Neural Network (CNN): A type of neural network commonly used in computer vision tasks that is designed to identify patterns in image data.

Experience Replay: A technique used to train RL models by randomly sampling experiences from a replay buffer, which can improve sample efficiency and stabilize training.

Neural Network: A type of machine learning model inspired by the structure and function of the human brain, which is composed of interconnected nodes that can learn to recognize patterns in data.

REFERENCES

1. Du, R.; Zhang, X.; Chen, C.; Guan, X. Path Planning with Obstacle Avoidance in PEGs: Ant Colony Optimization Method. In Proceedings of the 2010 IEEE/ACM Int'L Conference on Green Computing and Communications &Int'L Conference on Cyber, Physical and Social Computing, GREENCOM-CPSCOM '10, Hangzhou, China, 18–20 December 2010; IEEE Computer Society: Hangzhou, China, 2010; pp. 768–773. [CrossRef]

2. Pauplin, O.; Louchet, J.; Lutton, E.; Parent, M. Applying Evolutionary Optimisation to Robot Obstacle Avoidance. arXiv 2004, arxiv:cs/0510076.

3. Lin, C.J.; Li, T.H.S.; Kuo, P.H.; Wang, Y.H. Integrated Particle Swarm Optimization Algorithm Based Obstacle Avoidance Control Design for Home Service Robot. Comput. Electr. Eng. 2016, 56, 748–762. [CrossRef]

4. Kim, C.J.; Chwa, D. Obstacle Avoidance Method for Wheeled Mobile Robots Using Interval Type-2 Fuzzy Neural Network. IEEE Trans. Fuzzy Syst. 2015, 23, 677–687. [CrossRef]

5. Zhang, H.; Han, X.; Fu, M.; Zhou, W. Robot Obstacle Avoidance Learning Based on Mixture Models. J. Robot. 2016, 2016, 7840580. [CrossRef]

6. Tai, L.; Li, S.; Liu, M. A deep-network solution towards model-less obstacle avoidance. In Proceedings of the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, Korea, 9–14 October 2016; pp. 2759–2764. [CrossRef]

7. Sutton, R.S.; Barto, A.G. Introduction to Reinforcement Learning, 1st ed.; MIT Press: Cambridge, MA, USA, 1998.

8. Lee, H.; Shen, Y.; Yu, C.H.; Singh, G.; Ng, A.Y. Quadruped robot obstacle negotiation via reinforcement learning. In Proceedings of the 2006 IEEE International Conference on Robotics and Automation, Orlando, FL, USA, 15–19 May 2006; pp. 3003–3010. [CrossRef]

9. Kominami, K.; Takubo, T.; Ohara, K.; Mae, Y.; Arai, T. Optimization of obstacle avoidance using reinforcement learning. In Proceedings of the 2012 IEEE/SICE International Symposium on System Integration (SII), Fukuoka, Japan, 16–18 December 2012; pp. 67–72.

10. [CrossRef] 10. Zhang, T.; Kahn, G.; Levine, S.; Abbeel, P. Learning Deep Control Policies for Autonomous Aerial Vehicles with MPC-Guided Policy Search. arXiv 2015, arxiv:1509.06791.

11. Sadeghi, F.; Levine, S. RL: Real Single-Image Flight without a Single Real Image. arXiv 2016, arxiv:1611.04201.

12. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.;

Ostrovski, G.; et al. Human-level control through deep reinforcement learning. Nature 2015, 518, 529–533. [CrossRef] [PubMed]

13. Lei, T.; Ming, L. A robot exploration strategy based on Q-learning network. In Proceedings of the IEEE International Conference on Real-time Computing and Robotics (RCAR), Angkor Wat, Cambodia, 6–10 June 2016; pp. 57–62. [CrossRef]

14. Smolyanskiy, N.; Kamenev, A.; Smith, J.; Birchfield, S. Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 4241–4247. [CrossRef]

15. Dooraki, A.R.; Lee, D.J. Memory-based reinforcement learning algorithm for autonomous exploration in unknown environment. Int. J. Adv. Robot. Syst. 2018, 15, 1729881418775849. [CrossRef]

APPENDIX

Appendix A:

This Appendix contains the Custom Lidar code that has been implemented in our Final Year project

```
#include <Wire.h>
#include <VL53L0X.h>

#define STEPPER_PIN_1 9
#define STEPPER_PIN_2 10
#define STEPPER_PIN_3 11
#define STEPPER_PIN_4 12

int step_number = 0;
VL53L0X sensor;

void setup() {
  pinMode(STEPPER_PIN_1, OUTPUT);
  pinMode(STEPPER_PIN_2, OUTPUT);
  pinMode(STEPPER_PIN_3, OUTPUT);
  pinMode(STEPPER_PIN_4, OUTPUT);

  Serial.begin(115200);
  Wire.begin();
  sensor.setTimeout(500);
  if (!sensor.init())
  {
    Serial.println("Failed to detect and initialize sensor!");
    while (1) {}
  }

  sensor.startContinuous();

}

void loop() {

  rot(false);
  delay(5);
  rot(true);
  delay(5);



}

void rot(bool d) {
  for(int i = 0; i < 2048; i++)
  {
    OneStep(d);
    delay(2);
  }
}


void OneStep(bool dir){
   if(dir){
switch(step_number){
  case 0:
  digitalWrite(STEPPER_PIN_1, HIGH);
  digitalWrite(STEPPER_PIN_2, LOW);
  digitalWrite(STEPPER_PIN_3, LOW);
  digitalWrite(STEPPER_PIN_4, LOW);
  break;
  case 1:
  digitalWrite(STEPPER_PIN_1, LOW);
  digitalWrite(STEPPER_PIN_2, HIGH);
  digitalWrite(STEPPER_PIN_3, LOW);
  digitalWrite(STEPPER_PIN_4, LOW);
  break;
  case 2:
  digitalWrite(STEPPER_PIN_1, LOW);
  digitalWrite(STEPPER_PIN_2, LOW);
  digitalWrite(STEPPER_PIN_3, HIGH);
  digitalWrite(STEPPER_PIN_4, LOW);
  break;
  case 3:
  digitalWrite(STEPPER_PIN_1, LOW);
  digitalWrite(STEPPER_PIN_2, LOW);
  digitalWrite(STEPPER_PIN_3, LOW);
  digitalWrite(STEPPER_PIN_4, HIGH);
  break;
}
  }else{
```

```
  switch(step_number){
 case 0:
 digitalWrite(STEPPER_PIN_1, LOW);
 digitalWrite(STEPPER_PIN_2, LOW);
 digitalWrite(STEPPER_PIN_3, LOW);
 digitalWrite(STEPPER_PIN_4, HIGH);
 break;
 case 1:
 digitalWrite(STEPPER_PIN_1, LOW);
 digitalWrite(STEPPER_PIN_2, LOW);
 digitalWrite(STEPPER_PIN_3, HIGH);
 digitalWrite(STEPPER_PIN_4, LOW);
 break;
 case 2:
 digitalWrite(STEPPER_PIN_1, LOW);
 digitalWrite(STEPPER_PIN_2, HIGH);
 digitalWrite(STEPPER_PIN_3, LOW);
 digitalWrite(STEPPER_PIN_4, LOW);
 break;
 case 3:
 digitalWrite(STEPPER_PIN_1, HIGH);
 digitalWrite(STEPPER_PIN_2, LOW);
 digitalWrite(STEPPER_PIN_3, LOW);
 digitalWrite(STEPPER_PIN_4, LOW);
}
 }
step_number++;
 if(step_number > 3){
  step_number = 0;
 }
}

void dist(int i) {
 Serial.print(i);
 Serial.print(",");
 Serial.print(sensor.readRangeContinuousMillimeters() / 10);
 if (sensor.timeoutOccurred()) { Serial.print(" TIMEOUT"); }

 Serial.print(".");
}
```

Appendix B:

Motor control code of the Autonomous Rover

```
#define M1 3
#define M2 4
#define M3 5
#define M4 6

#define MAX_SPEED 550
#define CORR_FACTOR 100

void setup() {

  pinMode(M1, OUTPUT);
  pinMode(M2, OUTPUT);
  pinMode(M3, OUTPUT);
  pinMode(M4, OUTPUT);

}

void loop() {

  forward(MAX_SPEED);
  delay(1000);
  backward(MAX_SPEED);
  delay(1000);
  rotLeft(MAX_SPEED);
  delay(1000);
  rotRight(MAX_SPEED);
  delay(1000);
  pvtLeft(MAX_SPEED);
  delay(1000);
  pvtRight(MAX_SPEED);
  delay(1000);
  halt();

}
void halt() {
  analogWrite(M1, LOW);
  analogWrite(M2, LOW);
  analogWrite(M3, LOW);
  analogWrite(M4, LOW);
}
void forward(int s) {
  analogWrite(M1, s);
  analogWrite(M2, LOW);
  analogWrite(M3, s);
  analogWrite(M4, LOW);
}

void backward(int s) {
  analogWrite(M1, LOW);
  analogWrite(M2, s);
  analogWrite(M3, LOW);
  analogWrite(M4, s);
}
```

```
void rotLeft(int s) {
  analogWrite(M1, s);
  analogWrite(M2, LOW);
  analogWrite(M3, LOW);
  analogWrite(M4, s);
}

void rotRight(int s) {
  analogWrite(M1, LOW);
  analogWrite(M2, s);
  analogWrite(M3, s);
  analogWrite(M4, LOW);
}

void pvtLeft(int s) {
  analogWrite(M1, s);
  analogWrite(M2, LOW);
  analogWrite(M3, LOW);
  analogWrite(M4, LOW);
}
void pvtRight(int s) {
  analogWrite(M1, LOW);
  analogWrite(M2, LOW);
  analogWrite(M3, s);
  analogWrite(M4, LOW);
}
```

Appendix C:

This appendix contains the ESP communication code.

```
#include <ESP8266WiFi.h>

const char* ssid = "Nix";
const char* password = "123456789";

; //
WiFiServer server(80);

void setup() {
  Serial.begin(115200);
  delay(10);
  pinMode(5, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(0, OUTPUT);
  pinMode(13, OUTPUT);
  digitalWrite(5, LOW);
  digitalWrite(4, LOW);
  digitalWrite(0, LOW);
  digitalWrite(13, LOW);

  // Connect to WiFi network
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  // Start the server
  server.begin();
  Serial.println("Server started");

  // Print the IP address
  Serial.print("Use this URL to connect: ");
  Serial.print("http://");
  Serial.print(WiFi.localIP());
  Serial.println("/");

}

void loop() {
  // Check if a client has connected
  WiFiClient client = server.available();
  if (!client) {
    return;
  }
```

```
// Wait until the client sends some data
Serial.println("new client");
while(!client.available()){
  delay(1);
}

// Read the first line of the request
String request = client.readStringUntil('\r');
Serial.println(request);
client.flush();

// Match the request



if (request.indexOf("/light1on") > 0)  {
  digitalWrite(5, HIGH);


}
if (request.indexOf("/light1off") >0)  {
  digitalWrite(5, LOW);


}

 if (request.indexOf("/light2on") > 0)  {
  digitalWrite(4, HIGH);


}
if (request.indexOf("/light2off") >0)  {
   digitalWrite(4, LOW);

  }
   if (request.indexOf("/light3on") >0)  {
   digitalWrite(0, HIGH);

  }
  if (request.indexOf("/light3off") > 0)  {
   digitalWrite(0, LOW);

  }
   if (request.indexOf("/light4on") > 0)  {
   digitalWrite(13, HIGH);

  }
  if (request.indexOf("/light4off") > 0)  {
   digitalWrite(13, LOW);

  }
// Set ledPin according to the request
//digitalWrite(ledPin, value);

  // Return the response
  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/html");
  client.println(""); //  do not forget this one
  client.println("<!DOCTYPE HTML>");
  client.println("<html>");
  client.println("<head>");
  client.println("<meta name='apple-mobile-web-app-capable' content='yes'
/>");
  client.println("<meta name='apple-mobile-web-app-status-bar-style'
content='black-translucent' />");
 client.println("</head>");
  client.println("<body bgcolor = \"#f7e6ec\">");
  client.println("<hr/><hr>");
  client.println("<h4><center> Esp8266 Electrical Device Control
</center></h4>");
  client.println("<hr/><hr>");
  client.println("<br><br>");
  client.println("<br><br>");
  client.println("<center>");
  client.println("Device 1");
  client.println("<a href=\"/light1on\"\"><button>Turn On </button></a>");
  client.println("<a href=\"/light1off\"\"><button>Turn Off
</button></a><br />");
  client.println("</center>");
  client.println("<br><br>");
   client.println("<center>");
   client.println("Device 2");
  client.println("<a href=\"/light2on\"\"><button>Turn On </button></a>");
  client.println("<a href=\"/light2off\"\"><button>Turn Off
</button></a><br />");
client.println("</center>");
  client.println("<br><br>");
    client.println("<center>");
   client.println("Device 3");
  client.println("<a href=\"/light3on\"\">&lt;button>Turn On </button></a>");
  client.println("<a              href=\"/light3off\"\"><button>Turn              Off
</button></a><br />");
client.println("</center>");
  client.println("<br><br>");
   client.println("<center>");
   client.println("Device 4");
  client.println("<a href=\"/light4on\"\"><button>Turn On </button></a>");
  client.println("<a              href=\"/light4off\"\"><button>Turn              Off
</button></a><br />");
client.println("</center>");
  client.println("<br><br>");
  client.println("<center>");
  client.println("<table border=\"5\">");
 client.println("<tr>");
  if (digitalRead(5))
         {
           client.print("<td>Light 1 is ON</td>");


         }
          else
          {
            client.print("<td>Light 1 is OFF</td>");


         }

         client.println("<br />");
```

```
if (digitalRead(4))
 {
  client.print("<td>Light 2 is ON</td>");

}
 else
 {


   client.print("<td>Light 2 is OFF</td>");


 }
 client.println("</tr>");



 client.println("<tr>");

 if (digitalRead(0))

 {
  client.print("<td>Light 3 is ON</td>");


 }


 else


 {
   client.print("<td>Light 3 is OFF</td>");
}

if (digitalRead(13))

{

 client.print("<td>Light 4 is ON</td>");

}

else

{

  client.print("<td>Light 4 is OFF</td>");

}

client.println("</tr>");
```

```
          client.println("</table>");

          client.println("</center>");
  client.println("</html>");
  delay(1);
  Serial.println("Client disonnected");
  Serial.println("");

}
```

Appendix D:

This appendix contains the 2D mapping code

```
import processing.serial.*;              // imports library for serial communication

Serial myPort;                     // defines Object for Serial
String ang="";
String distance="";
String data="";
int angle, dist;
static float prev_x = 0, prev_y = 0;
static int i = 0;

void setup() {
  size (1700, 900);
  myPort = new Serial(this,"COM2", 115200);      // starts the serial communication. Changfe this to suit your serial monitor on Arduino
  myPort.bufferUntil('.');      // reads the data from the serial port up to the character '.' before calling serialEvent
  background(0);
}

void draw() {
    drawText();
    drawRadar();
    drawObject();
    if(angle==165 || angle == 15){
```

```
    background(0,0,0);
    }

    fill(0,5);
    noStroke();
    noStroke();
    fill(0,255);
    rect(0,height*0.93,width,height);                    // so that the text having angle
and distance doesnt blur out
}

void serialEvent (Serial myPort) {                                           // starts
reading data from the Serial Port
                                                              // reads the data from the
Serial Port up to the character '.' and puts it into the String variable "data".
    data = myPort.readStringUntil('.');
    data = data.substring(0,data.length()-1);

    int index1 = data.indexOf(",");
    ang= data.substring(0, index1);
    distance= data.substring(index1+1, data.length());

    angle = int(ang);
    dist = int(distance);
    System.out.println(angle);
}

void drawRadar()
```

```
{
   pushMatrix();
   noFill();
   stroke(10,255,10);        //green
   strokeWeight(1);
   translate(width/2,height-height*0.06);
   line(-width/2,0,width/2,0);
   arc(0,0,(width*0.25),(width*0.25),PI,TWO_PI);
   arc(0,0,(width*0.15),(width*0.15),PI,TWO_PI);
   arc(0,0,(width*0.35),(width*0.35),PI,TWO_PI);
   arc(0,0,(width*0.45),(width*0.45),PI,TWO_PI);
   arc(0,0,(width*0.55),(width*0.55),PI,TWO_PI);
   arc(0,0,(width*0.65),(width*0.65),PI,TWO_PI);
   arc(0,0,(width*0.75),(width*0.75),PI,TWO_PI);
   arc(0,0,(width*0.85),(width*0.85),PI,TWO_PI);
   arc(0,0,(width*0.95),(width*0.95),PI,TWO_PI);
   line(0,0,(-width/2)*cos(radians(90)),(-width/2)*sin(radians(90)));
   popMatrix();
}
void drawObject() {
   strokeWeight(7);
   stroke(255,0,0);
   translate(width/2,height-height*0.06);
   float pixleDist = (dist/199.0)*(width);                    // covers the distance
from the sensor from cm to pixels
   float x=-pixleDist*cos(radians(angle));
   float y=-pixleDist*sin(radians(angle));
   if(dist<=199)                                    // limiting the range to 40 cms
```

```
    {
      point(-x,y);
    }
    //if(i == 1)
    //{
    //  prev_x = 0;
    //  prev_y = 0;
    //  i = 0;
    //}
    //else
    //{
    //  prev_x = x;
    //  prev_y = y;
    //}
    //i++;
    //int diff = int(sqrt((prev_x-x)*(prev_x-x) + (prev_y-y)*(prev_y-y)));
    //if((prev_x != 0) && (prev_y != 0) && (diff < 5))
    //{
    //  line(prev_x, prev_y, x, y);
    //}
}

void drawText()
{
    pushMatrix();
    fill(100,200,255);
    textSize(25);
    text("20cm",(width/2)+(width*0.04),height*0.93);
    text("40cm",(width/2)+(width*0.09),height*0.93);
    text("60cm",(width/2)+(width*0.14),height*0.93);
    text("80cm",(width/2)+(width*0.19),height*0.93);
    text("100cm",(width/2)+(width*0.24),height*0.93);
    text("120cm",(width/2)+(width*0.29),height*0.93);
    text("140cm",(width/2)+(width*0.34),height*0.93);
    text("160cm",(width/2)+(width*0.39),height*0.93);
    text("180cm",(width/2)+(width*0.44),height*0.93);
    translate(width/2,height-height*0.06);
    textSize(25);
    text(" 30°",(width/2)*cos(radians(30)),(-width/2)*sin(radians(30)));
    text(" 60°",(width/2)*cos(radians(60)),(-width/2)*sin(radians(60)));
    text("90°",(width/2)*cos(radians(91)),(-width/2)*sin(radians(90)));
    text("120°",(width/2)*cos(radians(123)),(-width/2)*sin(radians(118)));
    text("150°",(width/2)*cos(radians(160)),(-width/2)*sin(radians(150)));
    popMatrix();
}
```

Appendix E:

Github:

https://github.com/awaiz360/fyp

Jira:

https://team-16679952749642.atlassian.net/jira/software/projects/FYP/boards/1