REINFORCEMENT LEARNING BASED AUTOMATED PATH PLANNING IN GARDEN ENVIRONMENT USING DEPTH – RAPIG-D

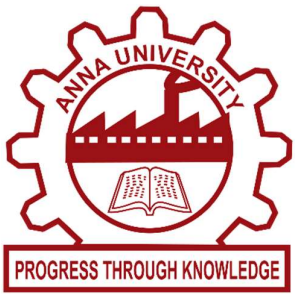
A Project Report

Submitted By:

SATHIYA MURTHI S (2018504604)
PRANAV BALAKRISHNAN (2018504581)
C ROSHAN ABRAHAM (2018504591)

In partial fulfillment for the award of the degree
of

BACHELOR OF ENGINEERING
in



ELECTRONICS AND COMMUNICATION
DEPARTMENT OF ELECTRONICS ENGINEERING
MADRAS INSTITUTE OF TECHNOLOGY

ANNA UNIVERSITY: CHENNAI 600044

JUNE 2022

ACKNOWLEDGEMENT

We consider it a privilege and our primary duty to express our gratitude and respect to all those who guided and inspired us in the successful completion of the project.

We owe solemn gratitude to **Dr. J. PRAKASH**, Dean, Madras Institute of Technology, for having given consent to carry out the project work at MIT Campus, Anna University.

We wish to express our sincere appreciation and gratitude to **Dr. M. GANESH MADHAN,** Professor and Head of the Department of Electronics Engineering, who has encouraged and motivated us in our endeavors.

We are extremely grateful to our project guide **Dr. V. SATHIESH KUMAR**, Assistant Professor, Department of Electronics Engineering, for his timely and thoughtful guidance and encouragement for the completion of the project.
We sincerely thank all our panel members **Dr. S. P. JOY VASANTHA RANI, Dr. A. VIJI, and MR. N. JAGADEESH KUMAR** for their valuable suggestions. We also thank all the teaching and non-teaching staff members of the Department of Electronics Engineering for their support in all aspects.

SATHIYA MURTHI S (2018504604)
PRANAV BALAKRISHNAN (2018504581)
C ROSHAN ABRAHAM (2018504591)

BONAFIDE CERTIFICATE

Certified that this Project Report titled **"Reinforcement Learning based Automated Path Planning in Garden environment using Depth"** is the bonafide work of **SATHIYA MURTHI S (2018504604), PRANAV BALAKRISHNAN (2018504581) & C ROSHAN ABRAHAM (2018504591)** who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Dr. M. GANESH MADHAN
Professor and Head
Department of Electronics Engineering
Madras Institute of Technology
Anna University
Chennai - 600044

Dr.V. SATHIESH KUMAR
Assistant Professor and Guide
Department of Electronics Engineering
Madras Institute of Technology
Anna University
Chennai - 600044

ABSTRACT

Reinforcement Learning-based path planning is a flexible solution that can account for a robot's capacity to independently map any unfamiliar area. In this article, a hardware implementation based on the SARSA algorithm for path planning and stereovision for depth estimation based obstacle detection is suggested and evaluated.

The robot is put through its paces in 3x3 areas with two obstacles. The objective is to map the surroundings by recognizing and mapping barriers as well as determining the best route to the target. The robot begins at one end of the environment and travels through it for a certain number of episodes, during which time it is noted that the robot can reliably recognize and map obstacles as well as discover the quickest way to the target in less than ten episodes. Currently, the destination is a fixed point that is regarded as the environment's opposite diagonal end.

The hardware implementation of Reinforcement learning-based path planning and the usage of depth estimation for obstacle detection are both validated in this paper.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

TERM	EXPLANATION
SARSA	State action state reward
UAV	Unmanned Aerial Vehicle
RL	Reinforcement Learning
3D	Three Dimensional
RoI	Region of Interest
SSH	Secure Shell
DMP	Digital Motion Processor
CV	Computer Vision
GPU	Graphics Processing Unit
MEMS	Micro Electro-mechanical system
AI	Artificial Intelligence

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Reinforcement Learning is a flexible and learning-based strategy that allows an agent to learn and select the best solution to a problem on its own, based on the environment's rewards and penalties. This enables the agent to deal with whatever state it may face while learning to attain the desired outcomes. SARSA, or State-Action-Reward-State Action, is a fundamental On Policy Reinforcement Learning Algorithm.

With more Reinforcement Learning innovations and implementations in Autonomous Robots, Self-driving vehicles, and Unmanned Aerial Vehicles (UAVs), there is a growing demand for adaptive mapping of unknown environments and the ability to determine the shortest path to any given destination in order to deal with unpredictable and dynamic environments.

Reinforcement Learning is an excellent way for meeting the requirements of such an environment by learning from repeated test iterations.

In this paper, this implementation has been used to plan routes and identify barriers in a garden environment in this research, and it can be used to automate the maintenance and caretaking of gardens and plants in the future.

This method can be applied to a variety of exploration and search jobs, including space exploration and mapping, as well as air crash and accident investigations.

1.1 REINFORCEMENT LEARNING

Reinforcement Learning is a machine learning approach that allows an agent to learn in an interactive environment through trial and error while receiving feedback from its own actions and experiences.

Though both supervised and reinforcement learning involve mapping between input and output, reinforcement learning uses incentives and punishments as signals for positive and negative behavior, unlike supervised learning, which provides the agent with a right set of behaviors for executing a task.

Reinforcement learning differs from unsupervised learning in terms of objectives. In unsupervised learning, the aim is to detect similarities and differences between data points; in reinforcement learning, the goal is to develop an appropriate action model that maximizes the agent's total cumulative reward.

The following are some essential phrases that explain the fundamental parts of an RL problem:

Environment — The physical world in which the agent functions

State — The agent's current circumstance

Reward — Environment feedback

Policy — Mapping an agent's state to its behaviors.

Value — The future reward that an agent will earn if they take a specific action in a specific condition.

1.1.1 ALGORITHMS IN REINFORCEMENT LEARNING

There are three primary approaches to implement a Reinforcement learning algorithm. They are

- Value-based
- Policy-based
- Model-based

Value-based:

In this approach, the algorithm finds the optimum value function and it will be the maximum value at any state. Therefore, a long-term reward will be expected by the agent under the policy.

Model-Based:

In this RL method, a virtual model is designed for each environment. The agent learns how to perform in a given specific environment.

Policy-based:

In a policy-based RL algorithm, a policy is determined such that the action performed in every state helps to gain maximum reward in the future. For any RL algorithm, there are two main policies for a learning agent.

They are

- On Policy
- Off policy

On policy:

In this, the learning agent learns the value function according to the current action derived from the policy which is in use. SARSA technique comes under On Policy and uses the action performed by the current policy to learn the Q-value.

Off policy:

In this, the learning agent learns the value function according to the action derived from another policy. Q-Learning technique comes under Off policy and uses the greedy approach to learn the Q-value.

1.2 STEREO CAMERA

A stereo camera is a type of camera with two or more lenses with a separate image sensor or film frame for each lens. This allows the camera to simulate human binocular vision and therefore gives it the ability to capture three-dimensional images, a process known as stereo photography. Stereo cameras may be used for making stereo-views and 3D pictures for movies, or for range imaging. The distance between the lenses in a typical stereo camera (the intra-axial distance) is about the distance between one's eyes (known as the intra-ocular distance) and is about 6.35 cm, though a longer baseline (greater inter-camera distance) produces a more extreme 3-dimensionality. Stereo cameras are mounted in cars to detect the lane's width and the proximity of an object on the road.

Figure 1.1: Xbox Kinect Stereo camera

1.3 DEPTH ESTIMATION

Depth Estimation is the task of measuring the distance of each pixel relative to the camera. Depth is extracted from either monocular (single) or stereo (multiple views of a scene) images. Traditional methods use multi-view geometry to find the relationship between the images. Newer methods can directly estimate depth by minimizing the regression loss, or by learning to generate a novel view from a sequence.

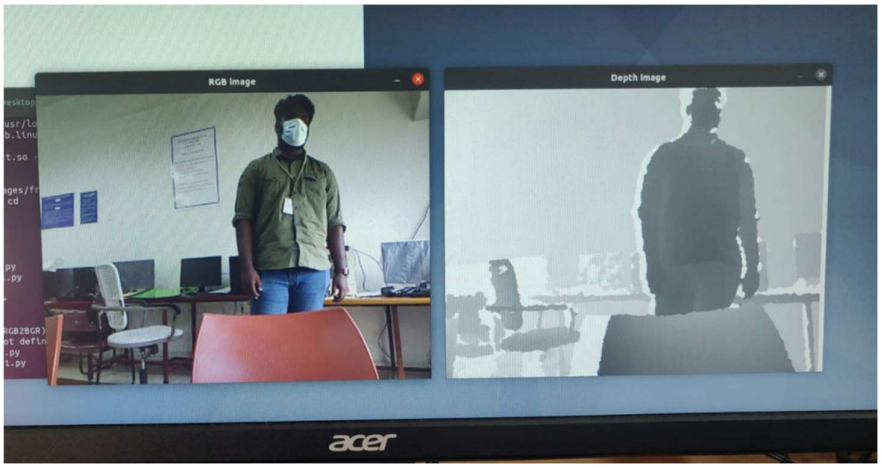


Figure 1.2: Depth Estimation

1.4 OPENCV

OpenCV is a huge open-source library for computer vision, machine learning, and image processing and it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as NumPy, Python is capable of processing the OpenCV array structure for analysis.

1.5 MPU 6050

MPU6050 is a Micro Electro-mechanical system (MEMS), it consists of a three-axis accelerometer and a three-axis gyroscope. It helps us to measure velocity, orientation, acceleration, displacement and other motion like features. MPU6050 consists of Digital Motion Processor (DMP), which has the property to solve complex calculations.

MPU6050 consists of a 16-bit analog to digital converter hardware. Due to this feature, it captures three-dimension motion at the same time.

1.9 NVIDIA JETSON NANO

The NVIDIA Jetson Nano (4GB) Developer Kit is a micro-controller that is generally used to prototype projects. Due to its GPU, it is commonly used for inference of neural networks and deep learning models on the edge. This makes it a suitable option for tasks such as Object detection, Image processing, and Image classification. It can be used to interface with sensors and cameras, run python scripts and implement Reinforcement learning algorithms.

1.10 PYTHON

Python is a programming language that supports the creation of a wide range of applications. Developers regard it as a great choice for Artificial Intelligence (AI), Machine Learning, and Deep Learning projects.

Python language comes with many libraries and frameworks that make coding easy. This also saves a significant amount of time.

Python projects can be integrated with other systems coded in different programming languages. This means that it is much easier to blend it with other AI projects written in other languages.

Also, since it is extensible and portable, Python can be used to perform cross languages tasks. The adaptability of Python makes it easy for data scientists and developers to train machine learning models. It can be used to implement the RL SARSA algorithm as well

CHAPTER 2
LITERATURE SURVEY

[7] Konor et al. used an upgraded Q-learning strategy to find the path between source and destination. The step distance (from one state to the next) as well as the ultimate destination are assumed. It's used to keep the entries in the Q-table up to date. Unlike classical Q-learning, where values are updated on a regular basis, the values are only input once. The Q-value calculated for the optimal action is saved at each state. Performance tends to improve in terms of traversal time and the number of states explored.

[8] The use of deep reinforcement learning for end-to-end mobile robot path planning is presented. To assess the state action values function, a deep Q-network is first developed and trained. As a result, this network determines the q-value associated with each action. Left, right, and forward are the actions employed. The RGB picture of the input is sent to the DQN once it is acquired from the environment. The robot's appropriate action is chosen via the action selection approach. As a result, the robot arrives at its destination without colliding with any impediments. Experiments in real time have shown that this strategy is successful. According to Sichkar et al., path planning is carried out using a Q-learning algorithm based on the Markov Decision Process.

[11] Using the classic Q-learning technique to discover an optimal path in complex scenarios is difficult. According to the authors, the robot determined/identified an ideal path from the source to the destination by avoiding collisions with obstructions in its propagation path. Q-learning and SARSA algorithms are used to find the shortest path between the source and destination. The approach was put to the test in a simulated setting with pre-set obstacles. In

the two algorithms utilised, different learning durations are provided. By avoiding collisions with things along the way, it also varies the amount of steps it takes to reach its goal.

[10] Traditional Breadth First Search (BFS) and Rapidly Exploring Random Trees (RRT) approaches fail to find the shortest path between the source and destination. As a consequence, the authors created and demonstrated a reinforcement learning-based path planning system. To begin, a randomly generated route graph is selected. It is not taken into consideration if the chosen path includes obstacles. The Q-learning method is used to find a collision-free path. According to the authors, the recommended technique delivered a smooth and shortest path when compared to RRT and BFS algorithms.

[12] There has been a proposal for a modified SARSA (state-action-reward-state-action) algorithm. The differences between Q-learning and SARSA algorithms were investigated, and the SARSA method was changed. After the robot arrives at its ultimate destination, the start and finish points of the updated algorithm are swapped, and the programme is rerun. The shortest path among all the pathways discovered is highlighted when all the iterations are completed. Although this technique has a high time complexity, it is the safest option. The Q-learning method was discovered to be the fastest and to provide the best path. When executional hazards are not taken into consideration, the Q-learning algorithm is shown to be the best. When the path length is unimportant, the SARSA method can be utilised. and having a short execution time When a safe optimal path is desired, the iterative SARSA method can be utilized , regardless of the temporal complexity.

[2] RL based planning algorithms provides a new resolution by integrating a high-level artificial intelligence. This work investigates the application of deep

reinforcement learning algorithms for USV and USV formation path planning with specific focus on a reliable obstacle avoidance in constrained maritime environments.

Path planning Reinforcement learning is still in its early stages, according to a thorough examination of the literature. The algorithm might be fine-tuned or refined further before being employed in real-time scenarios. This paper investigates the use of the Reinforcement Learning algorithm to path planning in an unknown environment. The SARSA algorithm is used to help the robot choose an appropriate action in an unfamiliar situation.

CHAPTER 3
MOTIVATION AND OBJECTIVES

3.1 MOTIVATION

- A learning based approach can be used to navigate and map an unknown environment and trace the best possible path between any two points.
- Stereo camera based depth mapping would be better suited to identify any type of obstacle and is a robust approach to obstacle detection.
- Current methods are limited by their inability to adapt to new environments and varying obstacles.

3.2 OBJECTIVES

- To perform hardware implementation of reinforcement learning based path planning to generate a map of an unknown environment by employing depth estimation based obstacle detection and analyze its performance.
- To estimate the best possible path to reach a given destination

CHAPTER 4
PROPOSED WORK

It is proposed to design a robot equipped with a stereo camera capable of generating a depth map that is integrated with SARSA algorithm - a Reinforcement Learning technique to find the optimal path from start to destination in a cell-based testing space while mapping the unknown environment by detecting and avoiding obstacles.

4.1 REINFORCEMENT LEARNING

Reinforcement learning is a learning method involving rewarding desired behaviors and/or punishing undesirable actions. Generally, a reinforcement learning agent is able to observe and interpret its environment, choose and take actions and learn through the reward offered. It shows adaptability and the ability to account for dynamic environments with limited to no parameters specified as opposed to other learning methods such as Supervised or Unsupervised learning. This is favorable when attempting to map an unknown environment, giving more flexibility.

4.1.1 SARSA ALGORITHM

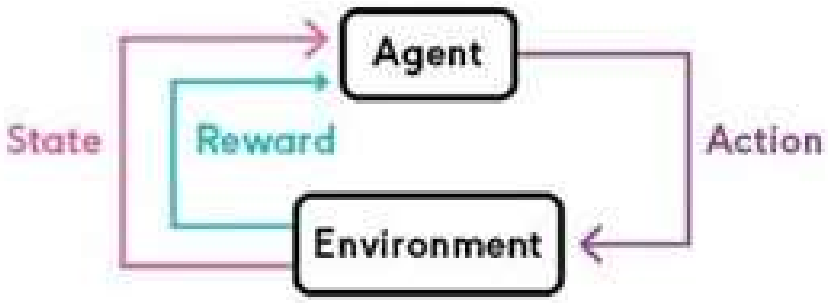


Figure 4.1: Flowchart of SARSA Algorithm

SARSA algorithm is a passive Reinforcement learning algorithm that can be applied to fully observable environments. It is an On policy algorithm. Q-value is to determine how favorable a specific action is in a specific state. To update the Q-values, an action is chosen at each state using a policy. The action is chosen based on this policy and in turn, leads to the reward for the next state. The name of the algorithm represents the sequence of its operation. The current state **S** is given to the agent by the environment, which chooses an action **A** based on the policy function and then this is fed to the environment. The environment determines the corresponding reward **R** for the action taken in the current state and the net state **S** which is given to the agent to take the next action A.

$$\boldsymbol{Q}\ (\boldsymbol{St}, \boldsymbol{At}) \leftarrow \boldsymbol{Q}\ (\boldsymbol{St}, \boldsymbol{At}) + \boldsymbol{\alpha}\ [\boldsymbol{Rt}\mathbf{+1} + \boldsymbol{\gamma Q}\ (\boldsymbol{St}\mathbf{+1}, \boldsymbol{At}\mathbf{+1}) - \boldsymbol{Q}\ (\boldsymbol{St}, \boldsymbol{At})]$$

Equation 4.1: SARSA Q-value update rule

Equation 4.1 tells how to calculate the Q-value for each state-action pair given by the SARSA algorithm.

Some parameters used in SARSA are

- Learning Rate
- Discount factor
- Initial conditions

Learning Rate:

It determines the extent to which the newly acquired information can override the old information. A factor value of 0 indicates the agent doesn't learn anything while a factor value of 1 indicates the agent should consider only the most recent information.

Discount factor:

It determines the importance of future rewards. A value of 0 indicates that the agent is opportunistic and a value of 1 indicates that the agent is striving or struggling for long high term reward. If the value exceeds 1, it indicates that the Q-values may diverge.

The policy function taken up for this implementation is as follows:
Ninety percent of the actions are chosen based on the most favorable choice using the determined Q-values and Ten percent of the actions are chosen at random.

4.2 SOFTWARE IMPLEMENTATION

There are many algorithms for path planning. In this work, SARSA (State – Action – Reward – State – Action) algorithm is used. The entire coding is done in Python Language. The validity of this algorithm is first estimated in a software implementation with a virtual agent and specified obstacles in a fixed environment.

4.3 HARDWARE IMPLEMENTATION

For hardware implementation, the NVIDIA Jetson Nano (4GB) Developer Kit (Figure 4.2) is used as the microcontroller to run the algorithm for path planning and interface with the XBOX 360 Kinect camera which is shown in Figure 1.1, and the sensors required for calibrated movement from one cell to the next in the testing environment. The Kinect camera provides the required depth map by utilizing a stereo camera setup along with an IR camera. An MPU6050 sensor (Figure 4.3) is used to accurately turn by calculating the angular displacement through integration of the angular velocity output of the sensor. A Li-Po battery (11.1V) is used to power four 12V 100 RPM DC motors which are

controlled by the L298N motor driver (Figure 4.4) and the Kinect camera. The Jetson Nano is externally powered using a 5V4A AC adapter.

4.3.1 COMPONENTS USED

Table 4.1: Components used in hardware implementation

COMPONENTS	SPECIFICATION	NUMBER USED
NVIDIA Jetson Nano	Developer Kit – 4GB	1
Robot Chassis	Dimensions: 250 x 200 x 46 mm	1
DC motors	12 V, 100 rpm	4
Wheels	Diameter – 130 mm Width – 60 mm	4
XBOX Kinect v1 Camera	Stereo camera	1
Motor Driver	L298N	1
Gyro sensor	MPU6050 – 6 axis motion sensor	1
Battery	11.1 V 4200mah Li-Po	1

Figure 4.2: NVIDIA Jetson Nano board

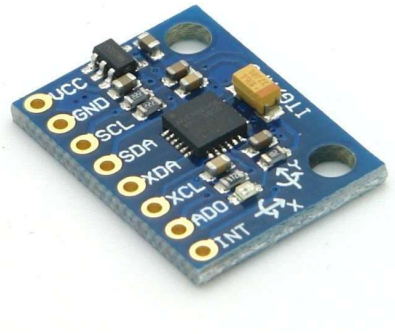


Figure 4.3: MPU6050 gyro sensor

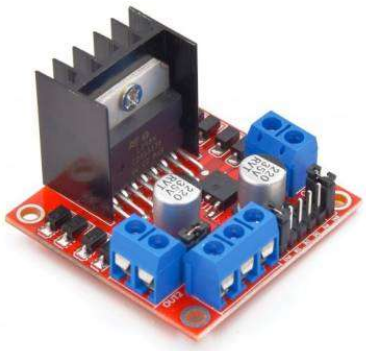


Figure 4.4: L298N Motor driver

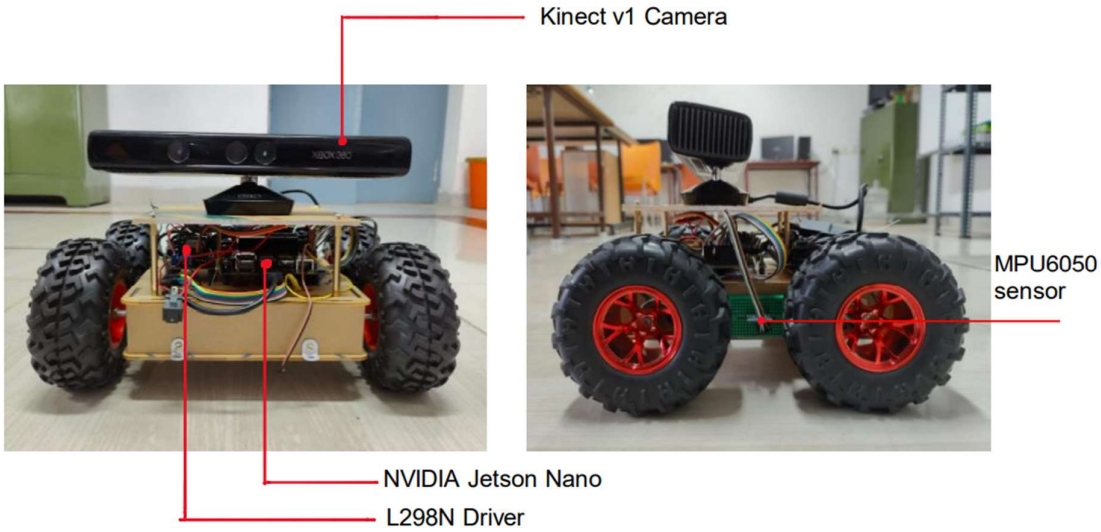


Figure 4.5: Robot designed using the mentioned components – Front and Side view

Figure 4.5 shows the robot that had been designed for estimating the validity of the hardware estimation. The components mentioned above have been put together and integrated to perform their respective functions in the task. The MPU6050 sensor is utilized to accurately turn 90 degrees as movement accuracy is a concern in a cell-based environment. The Kinect camera shall provide the depth map for obstacle detection and the NVIDIA Jetson Nano is the micro-controller used to integrate the modules and perform the Reinforcement learning algorithm. It can be remotely operated through SSH.

4.3.2 TESTING SPACE

A 3x3 testing space with cells of 2 feet in length and width is utilized to test this implementation. Figure 4.6 shows the testing space with the robot at the starting point (0,0). The diagonal bottom right point (2,2) is taken as the destination and boxes have been placed to represent obstacles at (0,1) and (2,1).

Figure 4.6: Testing space

4.3.3 FLOW CHART

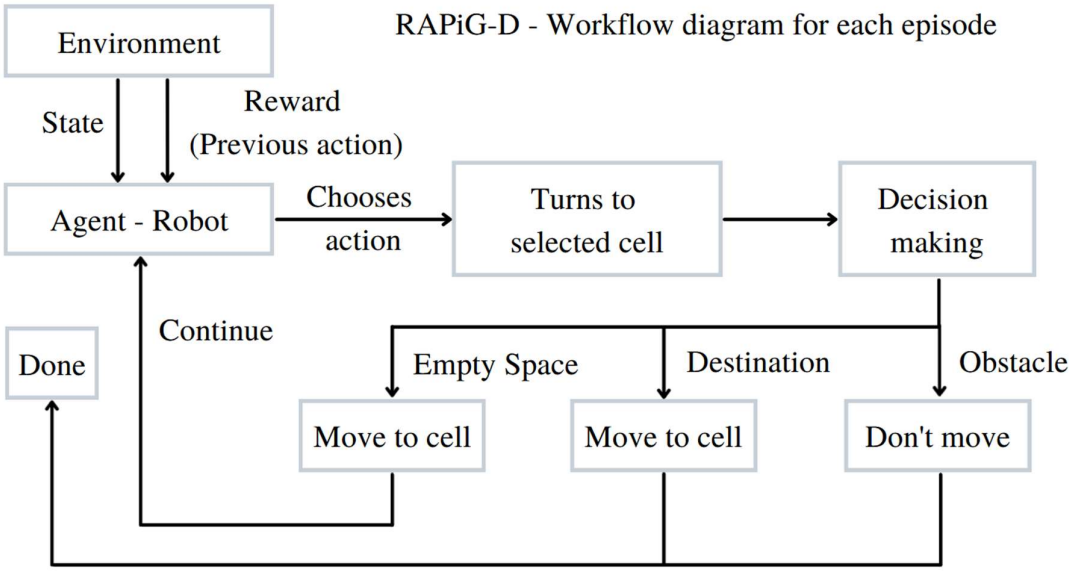


Figure 4.7: Flow diagram for each episode of the algorithm

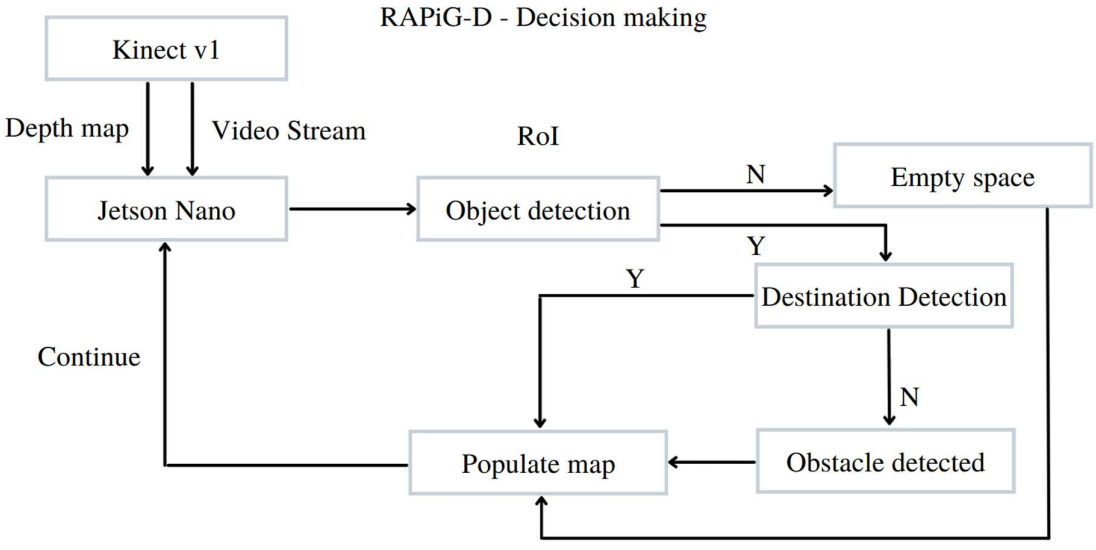


Figure 4.8: Flow diagram for decision making at each cell

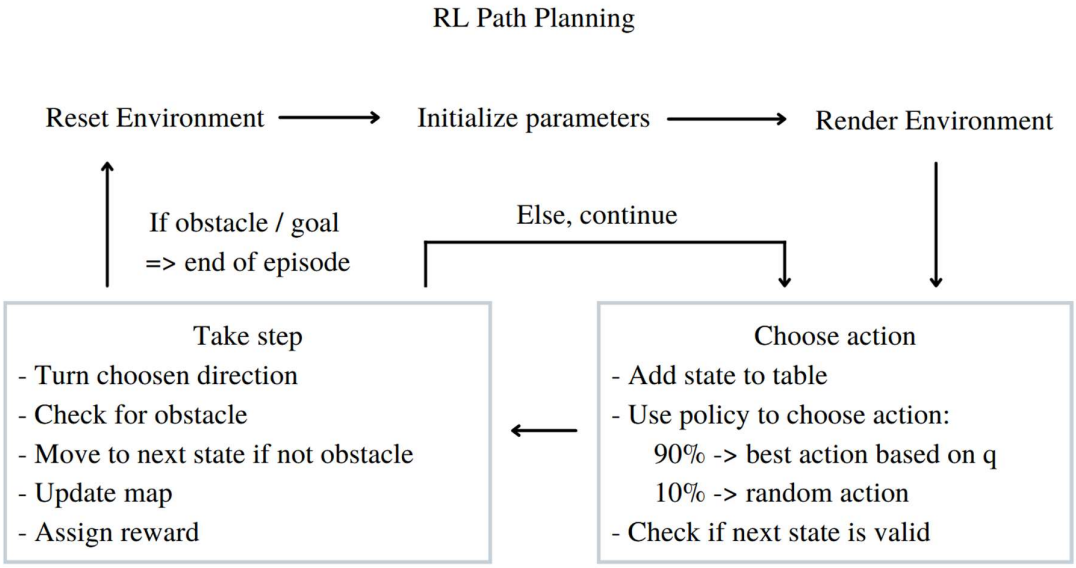


Figure 4.9: Path Planning Algorithm

4.3.4 ALGORITHM

Both the software simulation and the hardware implementation use a similar algorithm, with the exception that the hardware implementation uses depth estimation for obstacle detection rather than supplying obstacle coordinates in the simulation.

Figure 4.7 depicts the workflow of the algorithm implemented for each episode and Figure 4.8 shows the decision-making process at each cell. Figure 4.9 shows the path planning algorithm.

Step 1: Reset environment at start of each episode

Step 2: Choose action at each state based on policy function

Step 3: Check validity of next state

Step 4: Turn to required direction based on action chosen

Step 5: Check next state for presence of obstacle

Step 6: If no obstacle, move to next state and continue

Step 7: Check if new state is the destination given

Step 8: If destination, current route is checked for shortest path from start to destination

Step 9: Each episode ends when either an obstacle is detected or destination is reached and the obstacles are populated in the map

Step 10: A specified number of episodes are run and the agent learns the optimal path and location of obstacles

4.3.5 PATH PLANNING

At each state i.e., at each cell the agent – the robot chooses an action and then takes the step if it is valid based on the action by turning to the required direction – up, down, left or right. After that, the agent determines if the next state is an empty space, an obstacle, or the destination. This information is subsequently used to construct the environment map.

At the start of the first episode, a Q-table is created with zeros, and it is used to calculate how valuable it is to execute a specific action at a specific state in order to maximize the reward obtained. For each action at the current state, the Q-table is updated based on the reward given by the environment for the next state. If the next state is the destination, a reward of 1 is given, 0 if it is an empty space, and -1 if it is an obstacle.

In each episode, the agent explores the environment until it encounters an obstacle or reaches its objective. When the agent arrives at the destination, the path from the start point is logged to see if it is the shortest path it has travelled thus far. This is used to determine the best route to the desired location. The best path may then be utilized to get to the destination when required.

4.3.6 OBSTACLE DETECTION

This module is performed by using the depth map provided by the Kinect v1 camera. A threshold is applied on the depth map to detect objects in the immediate next cell i.e., around two feet away from the robot. A region of interest is also applied in the middle sixty percent width and length of the frame to increase the accuracy of obstacle placement. When an object is detected covering more than ten percent of this area, an obstacle is said to be detected.

4.3.7 CALIBRATED MOVEMENT

The task requires accurate four-directional movement to enable repetitive traversal in the testing space in each episode of the learning process. The forward and backward movements are controlled using the time taken to move from one cell to the next (1.32 seconds). For turns, the MPU6050 sensor is utilized. The angular velocity of the robot is extracted from the sensor which is then integrated over time (Figure 4.10) to identify the angular displacement. This is then used to control the turning angle.

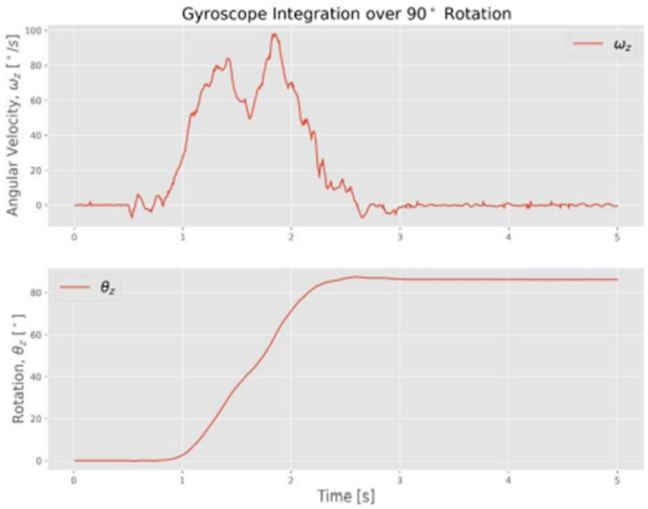


Figure 4.10: Angular velocity data integration

CHAPTER 5
RESULTS AND DISCUSSIONS

5.1 SOFTWARE IMPLEMENTATION

To check the validity of the path planning algorithm, a software simulation was performed on a larger 10 x 10 environment with 10 obstacles and the agent ran 500 episodes. Software implementation takes less time per episode. The start point is (0,0) and the destination is (9,9). Figure 5.1 shows the output of the SARSA algorithm – the map of the environment with the detected obstacles and the optimal path to reach the destination from the starting point. The obstacle coordinates are manually set and fed to the algorithm. Based on the policy, the RL agent goes around the environment, checking each coordinate to see if it is an obstacle, an empty space, or the destination. As a result, the shortest path is updated every time it reaches the goal, allowing the agent to choose the best path to the objective while avoiding obstacles.

Figure 5.2 shows a plot of the number of steps taken as each episode progresses. As the agent identifies obstacles and starts reaching the destination, the Q-table gets updated and the agent starts favoring the optimal path to reach the destination in the shortest path identified. Thus, as the episodes progress, the number of steps taken in each episode reduces. This is indicative of the learning process of the agent, allowing it to adapt to the unknown environment after exploring and mapping obstacles and identifying the shortest path to the given destination.

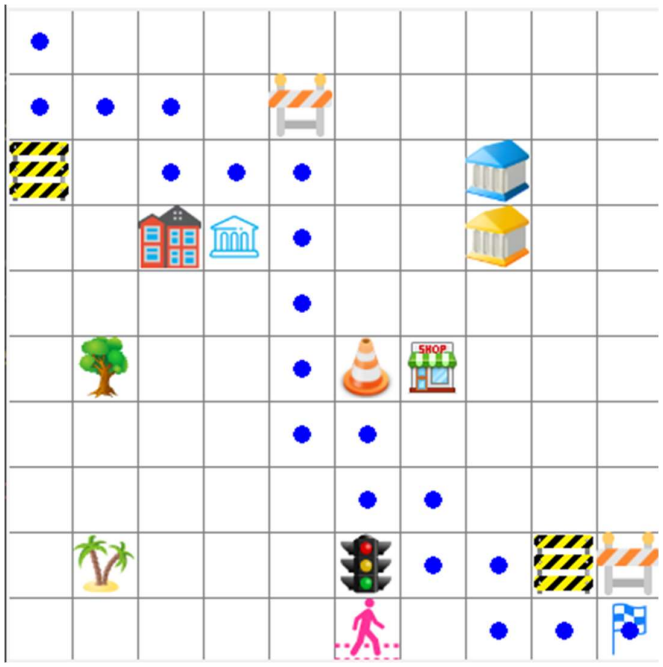


Figure 5.1: Map generated in software implementation

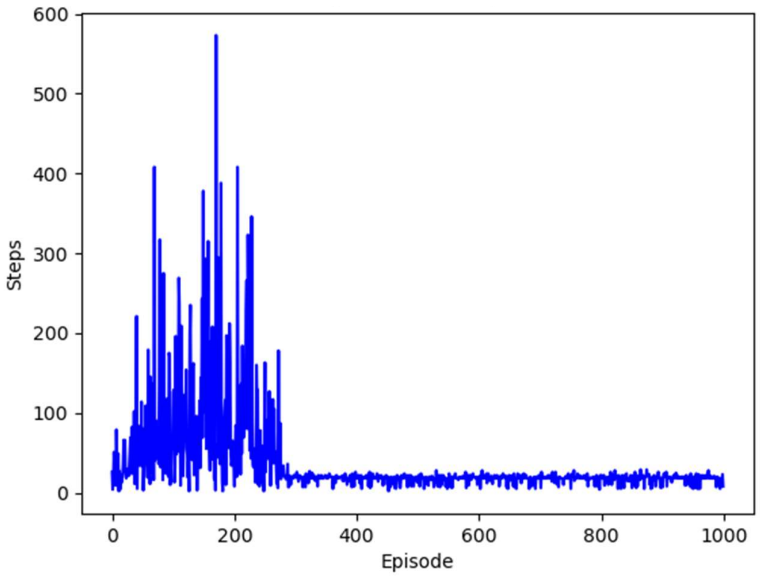


Figure 5.2: Plot of number of steps in each episode - software

5.2 HARDWARE IMPLEMENTATION

5.2.1 OBSTACLE DETECTION

Figure 5.3 shows the output of the Obstacle detection module. The depth map from the stereo camera is visualized. The red box is the region of interest taken for accurate object placement and the red outline of the box indicates the presence of an obstacle as the size of the object within the specified threshold of 50 cm is greater than 10 percent of the area of the RoI.

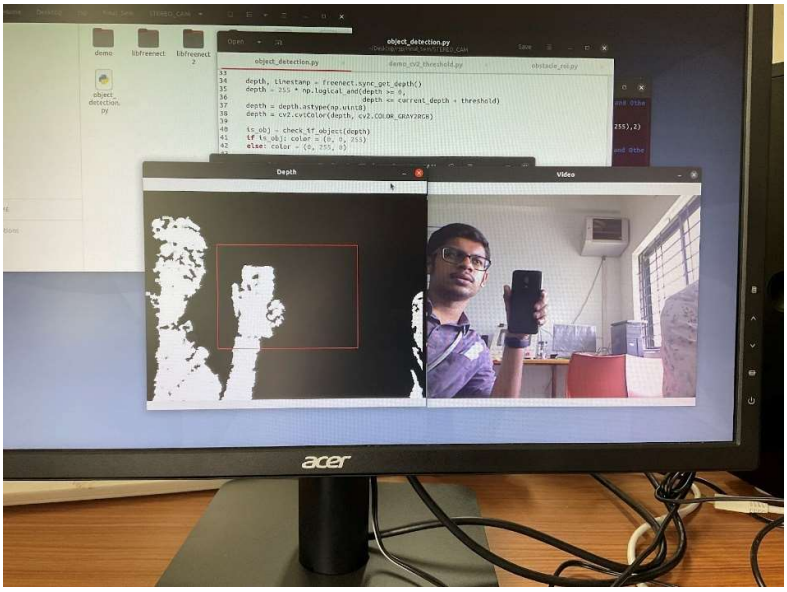


Figure 5.3: Obstacle detection output

5.2.2 PATH PLANNING

The robot agent was allowed to explore a 3 x 3 testing space (Figure 4.6) for 10 episodes. The starting point is at (0,0) and the destination is at (2,2). The agent successfully detected obstacles in adjacent cells and populated the detected obstacles as expected. The movement was also accurate, controlled by the MPU6050 gyro sensor. It was observed that after reaching the destination a

couple of times in the first 4 episodes, the agent began traversing the optimal path to reach the destination from the 5th episode. At the end of 10 episodes, the algorithm generates an exact map of the environment as shown in Figure 5.4 including the obstacles placed at (0,1), (2,1). The map also shows the optimal path to reach the destination as estimated by the algorithm. Figure 5.5 shows the plot of the number of steps taken by the agent in each episode of the hardware implementation.

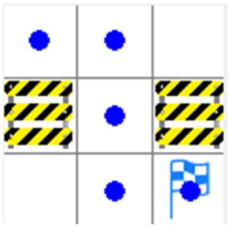


Figure 5.4: Map generated in hardware implementation

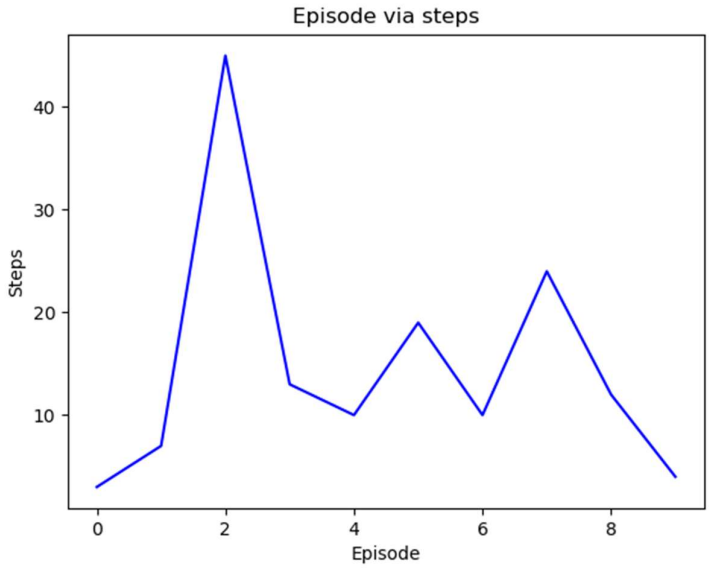


Figure 5.5: Plot of number of steps in each episode - hardware

Table 5.1: Final Q-Table – Optimal Path

Length of final q-table = 3

	0	1	2	3
[40.0,0.0]	0.0	0.000008	0.00000	0.00
[40.0,40.0]	0.0	0.000882	-0.01000	-0.01000
[40.0,80.0]	0.0	0.000000	0.04901	0.00

Table 5.2: Full Q-Table

Length of full q-table = 8

	0	1	2	3
[0.0,0.0]	0.00	-0.010000	3.615840e-08	0.00
Obstacle	0.00	0.000000	0.000000e+00	0.00
[40.0,0.0]	0.00	0.000008	0.000000e+00	0.00
[80.0,0.0]	0.00	-0.010000	0.000000e+00	0.00
[40.0,40.0]	0.00	0.000882	-1.000000e-02	-0.01
[40.0,80.0]	0.00	0.000000	4.900995e-02	0.00
[0.0,80.0]	-0.01	0.000000	2.673090e-04	0.00
Goal	0.00	0.000000	0.000000e+00	0.00

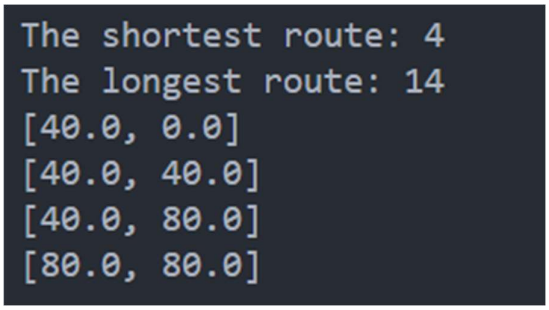


Figure 5.6: Shortest route

Figure 5.6 shows the shortest path determined by the algorithm. This is as shown in Figure 5.4 based on the pixel values. (Each cell is shown by 40 pixels in the map). Table 5.1 shows the final Q-table showing the Q-values of the states in the final optimal route to the destination. Table 5.2 shows the full Q-table with the Q-values for all the states explored by the agent.

It is observed that at any given state the Q-value corresponding to an action that would lead to an obstacle in the next state is negative and the action that would allow the agent to move closer to the destination or to the destination itself is positive. By this way, the agent chooses which action to take at each state and is able to identify the optimal path to reach the destination while avoiding obstacles.

CHAPTER 6
CONCLUSION AND FUTURE WORKS

Therefore, it is observed that the robot as an RL agent, successfully performs RL based path planning to accurately map the environment and its obstacles and determine the shortest path to reach a given destination. In the 3x3 environment, the robot agent performs accurate obstacle detection implementing depth estimation and by using the SARSA algorithm, identifies the optimal path to the destination within 10 episodes. As the number of obstacles and the size of the environment increases the time taken to identify the shortest path would also increase.

Thus, a simple RL algorithm such as SARSA is used to identify an optimal path and is able to allow the robot agent to map an unknown environment.

This implementation can be further extended to larger environments and even dynamic environments. Future implementations may involve increasing the degrees of freedom of the agent and the environment states and even going beyond cell-based environments. It can also be implemented in diversified applications ranging from garden or warehouse management, Air crash investigations, Search and Rescue operations, serving food at restaurants, and even Space exploration.

REFERENCES

1. S. Zheng and H. Liu, “Improved Multi-Agent Deep Deterministic Policy Gradient for Path Planning-Based Crowd Simulation” in IEEE Access, vol. 7, pp. 147755-147770, 2019.
2. Xinyuan Zhou, Peng Wu, Haifeng Zhang, Weihong Guo and Yuanchang Liu, “Learn to Navigate: Cooperative Path Planning for Unmanned Surface Vehicles Using Deep Reinforcement Learning”, IEEE Access, vol. 7, 2019
3. Chen Chen, Jiange Jiang, Ning Lv and Siyu Li, “An Intelligent Path Planning Scheme of Autonomous Vehicles Platoon Using Deep Reinforcement Learning on Network Edge”, IEEE Access, vol. 8, 2020.
4. Chao Wang, Jian Wang, SeYuan Shen, and Xudong Zhang,” Autonomous Navigation of UAVs in Large-Scale Complex Environments: A Deep Reinforcement Learning Approach”, IEEE Transactions On Vehicular Technology, vol. 68, pp. 2124-2136, March 2019.
5. D. Ebrahimi, S. Sharafeddine, P. -H. Ho, and C. Assi, “Autonomous UAV Trajectory for Localizing Ground Objects: A Reinforcement Learning Approach,” in IEEE Transactions on Mobile Computing, vol. 20, pp. 1312-1324, 1 April 2021.
6. B. Wang, Z. Liu, Q. Li, and A. Prorok, “Mobile Robot Path Planning in Dynamic Environments Through Globally Guided Reinforcement Learning,” in IEEE Robotics and Automation Letters, vol. 5, pp. 6932-6939, Oct. 2020.
7. A. Konar, I. Goswami Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar, “A Deterministic Improved Q-Learning for Path Planning of a Mobile Robot,” in IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 43, pp. 1141-1153, Sept. 2013
8. J. Xin, H. Zhao, D. Liu, and M. Li, “Application of deep reinforcement learning in mobile robot path planning,” In: Chinese Automation Congress

(CAC), 2017.

9. V. N. Sichkar, “Reinforcement Learning Algorithms in Global Path Planning for Mobile Robot,” In: International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM), 2019, pp. 1-5
10. P. Gao, Z. Liu, Z. Wu, and D. Wang, “A Global Path Planning Algorithm for Robots Using Reinforcement Learning,” In: IEEE International Conference on Robotics and Biomimetics (ROBIO), 2019.
11. Y. Long, and H. He, “Robot path planning based on deep reinforcement learning,” In: IEEE Conference on Telecommunications, Optics and Computer Science (TOCS), 2020
12. P. Mohan, L. Sharma and P. Narayan, “Optimal Path Finding using Iterative SARSA,” In: 5th International Conference on Intelligent Computing and Control Systems (ICICCS), 2021, pp. 811-817.
13. Valentyn N. Sichkar, “Reinforcement Learning Algorithms in Global Path Planning for Mobile Robot,” In: International Conference on Industrial Engineering, Applications and Manufacturing, 2019