# NETWORK OFFLOADING POLICIES FOR CLOUD ROBOTICS: ENHANCED SITUATION AWARE ROBOT NAVIGATION USING DEEP REINFORCEMENT LEARNING

A Thesis

by

JAHUN OH

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

| | |
|---|---|
| Chair of Committee, | Srinivas Shakkottai |
| Committee Members, | Yoonsuck Choe |
| | Jinsil Seo |
| Head of Department, | Scott Schaefer |

August 2023

Major Subject: Computer Science and Engineering

Copyright 2023 Jahun Oh

# ABSTRACT

In the case of resource-limited robots, such as low-power drones or 4-wheel vehicles, there can be a lack of sufficient onboard computation capabilities or battery limitations for implementing precise navigation models. One plausible solution to this issue is the use of cloud robotics, which could assign tasks to the cloud. However, when communicating with the cloud through congested wireless networks, latency or data loss may happen. Furthermore, when computations are excessively dependent on the cloud, bottlenecks could occur, adversely affecting the performance of navigation tasks for which real-time communications are necessary.

To tackle this challenge, this research proposes a cloud-supported navigation system for resource-limited robots, which utilizes deep reinforcement learning to optimize strategies for offloading tasks to the cloud. This approach aims to control the quality of information obtained by the robot while minimizing the impact of potential communication issues. The offloading problem is formulated as a Markov Decision Process (MDP), and a reinforcement learning (RL) algorithm is applied to learn the optimized offloading policy. This method ensures efficient navigation decision-making even under constraints related to computation and energy resources.

The proposed system is assessed through the use of pre-built navigation models within the ROS navigation stack, with experiments carried out in both simulation and real-world settings. At the end of the research, this paper intends to provide an evaluation of the system's capabilities and limitations in various scenarios. The outcomes of these evaluations should indicate that the suggested system can substantially enhance navigation performance while concurrently reducing the expenses associated with cloud communication, which ultimately will provide for more efficient and reliable robotic navigation operations in different environments.

# DEDICATION

This thesis is dedicated to my newborn son. I wish that his generation will live in a world filled with peace and love. I hope that my research can enable the light of world peace, where he will see a future full of harmony and unity.

I also wish to dedicate this work to my first army battalion commander, Choi, who recently passed away. His guidance not only shaped me as a soldier, but also as a person. His brave spirit and wise words had a deep effect on me and gave me strength throughout this research and the career. Through this dedication, I hope to honor his memory and continue his inspiring influence.

CONTRIBUTORS AND FUNDING SOURCES

## Contributors

This research was facilitated by a thesis committee comprised of Professor Srinivas Shakkottai from the Department of Electrical and Computer Engineering, Yoonsuck Choe from the Department of Computer Science and Engineering, and Professor Jinsil Seo from the Department of Visualization.

The entire thesis was independently executed by the student.

## Funding Sources

The funding for this graduate study was provided by a fellowship from the Department of Defense of the Republic of Korea.

TABLE OF CONTENTS

Page

ABSTRACT ..... ii

DEDICATION ..... iii

CONTRIBUTORS AND FUNDING SOURCES ..... iv

TABLE OF CONTENTS ..... v

LIST OF FIGURES ..... vii

LIST OF TABLES ..... viii

1. INTRODUCTION ..... 1

1.1 Motivation ..... 1
1.2 Background and Related Work ..... 2
1.2.1 Deep Reinforcement Learning (DRL) and Robot Navigation Overview ..... 2
1.2.1.1 Reinforcement Learning ..... 2
1.2.2 Markov Decision Process ..... 3
1.2.2.1 Deep Q-Networks (DQN) ..... 4
1.2.3 Deep Reinforcement Learning and Robot Navigation Fundamentals ..... 5
1.2.4 Comparison with Other Machine Learning Methodology ..... 6
1.2.5 Navigation ..... 7
1.2.5.1 High Computational Demand of LiDAR-based SLAM ..... 8
1.2.6 Fog Robotics ..... 8
1.2.7 Shortcomings of Offloading to the Cloud ..... 9
1.3 Robot Navigation Terminology ..... 10
1.3.1 Sensing ..... 10
1.3.2 Localization ..... 10
1.3.2.1 Map Representation ..... 11
1.3.2.2 Simultaneous Localisation and Mapping (SLAM) ..... 11
1.3.3 Path-Planning ..... 11
1.3.3.1 Off-line Path Planning ..... 11
1.3.3.2 On-line Path Planning ..... 11
1.3.3.3 Dynamic Window Based Methods ..... 12
1.3.4 Related Work ..... 12

2. PROBLEM STATEMENT ..... 14

2.1 Challenges ..... 14
2.2 Research Questions ..... 15
2.3 Proposed Work ..... 16
2.3.1 Sensory Input and Computation Models ..... 17
2.3.2 Offloading Markov Decision Process Formulation ..... 17
2.3.3 Action Space ..... 20
2.4 Reinforcement Learning-based Navigation Algorithm ..... 21
2.4.1 Leveraging Dense and Sparse Rewards ..... 21
2.4.2 How to interpret Network Communication Strength ..... 22
2.4.3 Linking Network Communication Strength to Reward Function Design ..... 23
2.4.4 Ultimate Reward Function Incorporating Situational Awareness ..... 24

3. METHODOLOGY ..... 26

3.1 Project Setup ..... 26
3.1.1 Motivation for Utilizing Deep Reinforcement Learning ..... 26
3.2 Deep Q-Learning Algorithm Implementation ..... 28
3.2.1 Replay Buffer ..... 28
3.3 Navigation Environment ..... 28
3.3.1 Gazebo: The Simulation Environment ..... 28
3.4 Experiment Scenario: Mapless Navigation Framework ..... 30
3.4.1 Deep Q-Network ..... 31
3.4.2 Training and Interaction ..... 31
3.4.3 Challenges of Standard DQN and Adoption of Target Networks ..... 32
3.4.4 Mitigation of Overestimation Bias with Double DQN ..... 33
3.4.5 Proximal Policy Optimization and Limitations ..... 34
3.4.6 Limitations in Training and Mitigation Strategies ..... 35
3.4.7 Comparison with Baseline Policies ..... 36
3.5 Evaluation and Discussion ..... 37
3.6 Real-World Experimentation ..... 38
3.6.1 Implementation to Real World Settings ..... 38

4. SUMMARY AND CONCLUSIONS ..... 43

4.1 Summary ..... 43
4.2 Comparison of Reinforcement Learning Algorithms ..... 43
4.3 Designing an Effective Reward Function ..... 44
4.4 Translation of Simulated Policies to Real-World Applications ..... 44
4.5 Explainable AI and Future Work ..... 44

REFERENCES ..... 46

LIST OF FIGURES

FIGURE Page


1.1 Markov Decision Process ..... 3

2.1 State Space Definition ..... 19

2.2 Robot Mapping Offloading MDP ..... 22

3.1 Jackal Robot ..... 27

3.2 Gazebo Simulation World for Policy Training ..... 29

3.3 Reward Plot for DQN after 50 episodes ..... 32

3.4 Reward Plot for Double DQN after 50 episodes ..... 33

3.5 Double-DQN with Heuristic Policy in Buffer ..... 36

3.6 Comparison with Benchmark Policies over mixture of environment episodes ..... 37

3.7 Comparison with Benchmark Policies on Trajectories ..... 39

3.8 Real-world Experiment with Explainable AI ..... 40

3.9 Real-world Experiment Trajectories ..... 41

3.10 Real-world Experiment Jackal ..... 42

# LIST OF TABLES

TABLE Page

1.1 Comparison of Local and Cloud Processing ..... 9

3.1 JACKAL Specifications ..... 26

# 1. INTRODUCTION

## 1.1 Motivation

The role of autonomous mobile robots is critical in emergency response and military operations, especially in missions that involve navigation in unpredictable and potentially hazardous environments [1] [2]. Emerging civilian applications, such as package delivery or home cleaning via autonomous robots, also involve complex and dynamic environments. The advent of cloud robotics has substantially expanded the capabilities of robots, allowing for offloading of computationally intensive tasks to the networked computing resources. However, efficient use of this powerful tool presents its own set of challenges, especially in information-scarce contexts typical of operating in new and unknown environments.

Network congestion and the consequent latency or data loss are significant challenges that can critically undermine offloading operations. Chinchali et al. [3] highlighted this problem, particularly when streaming high-bitrate point cloud streams of LiDAR data using the Robot Operating System (ROS) in multi-robot environments.

Moreover, the excessive offloading of data can potentially lead to an unintended denial of service in the cloud, risking the disruption of continuous service, which is especially crucial for settings such as maintaining operational integrity in the military. This sparks the need to develop and implement effective offloading policies for cloud robotics.

This research inquires about the offloading mapping mechanisms for enhanced situation awareness for robot navigation by utilizing Deep Reinforcement Learning(DRL). The aim is to use DRL to optimize the offloading process, balancing the need for extensive environmental awareness with computational efficiency.

In emergency response and military scenarios where autonomous robots are deployed with little to no information about the environment, maintaining real-time awareness is of critical importance. This research recognizes the necessity of achieving this while preserving computational efficiency

for sustainable operations. By doing so, it aims to ensure the robust performance of autonomous operations in such scenarios, thereby contributing significantly to the development of future cloud robotics in different environments.

## 1.2 Background and Related Work

### 1.2.1 Deep Reinforcement Learning (DRL) and Robot Navigation Overview

In this section, we start by providing in-depth analysis and recent advancements in the field of Reinforcement Learning and Robot Navigation. The focus lies on several solutions to employing DRL while exploring their major advantages and shortcomings. DRL combines the ability of deep learning to handle high-dimensional and complex data with the interaction-based learning approach of reinforcement learning. Taking inspiration from the learning process in humans, where a child swiftly learns about the cause and effect in the physical world through 'trial and error', an intelligent agent employs reinforcement learning to learn from actions and sensory observations in its environment continually.

#### *1.2.1.1 Reinforcement Learning*

Reinforcement Learning (RL) is an area of machine learning where algorithms learn to make decisions based on feedback from the environment. The objective of an RL agent is to learn an optimal policy, which is a set of actions that maximizes the reward over time. On the iterative learning process, the agent takes actions, observes the outcomes, and updates its knowledge based on the received rewards or penalties. Unlike supervised learning, where the algorithm is provided with labeled examples, reinforcement learning algorithms learn through 'trial and error', exploring the environment and adjusting their behavior based on the feedback they receive.

Deep Reinforcement Learning (DRL), a subfield of RL, has demonstrated its ability to manage complex tasks, such as the game of Go, and process observations from various sources, like images, laser scans, and languages. As a completely data-driven method, DRL eliminates the need for handcrafted control rules, which often depend heavily on the human's limited experience and understanding of the task. The success of AlphaGo Zero against professional Go player, Lee Sae

Dol, has proven that a purely data-driven method can outrank a human's short-sighted strategy.

Furthermore, DRL is capable of learning from a high-dimensional domain, making it superior over heuristic approaches both in terms of information and computation. In comparison to other machine learning methods like supervised learning or imitation learning, RL and DRL, is more suited to problems with complex reward structures and a strong need for sequential interactions with the environment, such as in mapless navigation.

Despite these advantages, the immense search space makes DRL challenging to train, and its real-world application presents significant hurdles. This research aims to overcome these challenges, with a focus on developing practical DRL strategies for mapless navigation that are efficient to train, can be generalized, and can be implemented on real robots.

### 1.2.2 Markov Decision Process

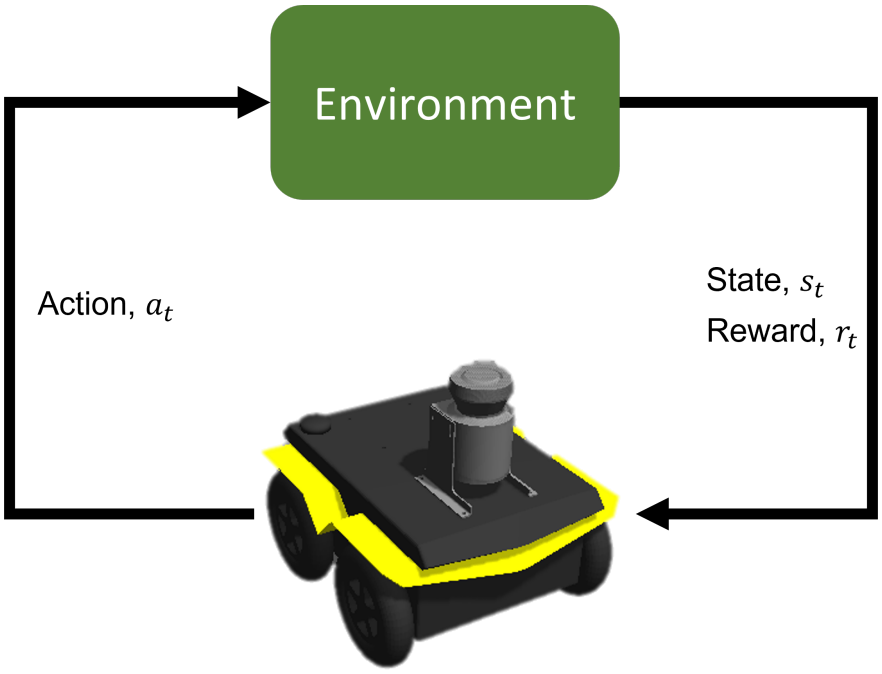


Figure 1.1: Markov Decision Process

Reinforcement learning is modeled as a Markov Decision Process (MDP), wherein actions are

mapped from states with the objective of maximizing a numerical reward signal. The fundamental principle of an MDP is the Markov property, which asserts that the future state is independent of the past, given the present state. Hence, a sequence of states $x_0, x_1, ..., x_T$ from time $0$ to $T$, is said to have the Markov property if and only if

$$P(x_{t+1}|x_t) = P(x_{t+1}|x_0, x_1, ..., x_t) \tag{1.1}$$

In this equation, the state $x_t$ lies in the state space $X$, and $P$ is the transition probability function, which describes the probability distribution of the next state given the current and historical states. Therefore, a Markov process is a memoryless random process that satisfies these dynamics.

#### *1.2.2.1 Deep Q-Networks (DQN)*

Deep Q-Networks (DQN) is a technique that merges deep learning with reinforcement learning to tackle complex decision-making scenarios. It employs a deep neural network as a decision for the Q-function, a central concept in Q-learning. Q-learning is a model-free RL algorithm that learns to make decisions with respect to the expected future rewards.

In Q-learning, the Q-function, denoted as $Q(s, a)$, represents the expected cumulative reward for taking an action $a$ in a particular state $s$, given that the agent follows policy $\pi$ afterward. The objective is to find the optimal policy that maximizes these future rewards.

Mathematically, the Q-function obeys the Bellman equation, which can be expressed as:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a') \tag{1.2}$$

Here, $s'$ denotes the next state after taking action $a$, $r$ is the reward, and $\gamma$ is the discount factor that weights the importance of future rewards.

DQN employs a deep neural network to approximate the Q-function, making it capable of dealing with high-dimensional state spaces that are typical in many real-world problems. The goal of the network is to minimize the difference between the predicted Q-values and the target

Q-values, referred to as the temporal difference error. The loss function for this can be written as:

$$L(\theta) = E_{(s,a,r,s')\sim U(D)}[(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2] \tag{1.3}$$

Here, $D$ denotes the replay buffer, $U(D)$ is a uniform random sample from the buffer, and $\theta$ and $\theta^-$ denote the parameters of the main and target networks, respectively.

DQN incorporates several strategies, such as experience replay and target networks, to stabilize the learning process and improve performance. Experience replay stores sets of agent's experiences (state, action, reward, and next state) in a memory buffer. During training, the agent samples random experiences from this buffer to update its Q-function. This practice helps to disrupt the correlation between consecutive experiences and mitigates the risk of over-fitting.

Target networks, on the other hand, are utilized to generate a steady target Q-function for updating the main Q-function, thus lessening the chances of divergence during training.

By integrating deep learning and reinforcement learning, DQN has been successfully applied to a variety of challenging problems, such as game playing, robotic control, and autonomous navigation, demonstrating its potential for furthering the field of robotics and artificial intelligence.

### 1.2.3 Deep Reinforcement Learning and Robot Navigation Fundamentals

In the context of reinforcement learning for robot navigation, the sequence from time $0$ to $T$, $x_0, a_1x_1, ..., a_T, x_T$, is referred to as a trajectory. At each point in time, the learning agent takes an action based on the current state of the environment. After executing the action, the robot receives a reward according to the reward function and then transitions to the next state. It's important to note that each action taken may influence all subsequent observations, actions, and rewards. This can lead to the feedback of the current action being delayed by multiple steps. The goal of this decision-making process is to maximize a discounted accumulative future reward, $R$, which infers the optimal action sequence of reaching a goal state from the current state. The policy $\pi$ is the strategy that the agent employs to decide an action based on the current observation, directly mapping from the state to action. The value $V$, defined as $V_\pi(x)$, is the expected sum of future

discounted rewards for a specific policy $\pi$ and state $x$, while the $Q$-value $Q_\pi(x, a)$ extends the value by taking an extra parameter, the action.

### 1.2.4 Comparison with Other Machine Learning Methodology

Reinforcement Learning (RL) distinguishes itself from other widespread learning paradigms like supervised and unsupervised learning. Supervised learning, which is the most extensively used learning approach, aims to discover a mapping function from input to output variables given labeled data. Training is performed to minimize the error between the agent's prediction and the given label for each input state, as depicted by the loss function:

$$L = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{1.4}$$

where $y_i$ is the true label, $\hat{y_i}$ is the predicted label, and $n$ is the number of samples. Given a sufficient number of training examples, the agent can utilize the learned knowledge to unseen situations. However, this approach does not suit interactive problems where there is no oracle with the optimal output.

In contrast, unsupervised learning aims to find the hidden structure of unlabeled data, typically using methods such as clustering or dimension reduction. Nevertheless, it doesn't align with the goal of reaching a certain outcome through a sequence of actions.

Therefore, it is assumed that the RL is the most suitable for learning from interactions with the environment. It is usable in complex robotic applications where the optimal trajectory isn't readily available due to the high-dimensional problem, thus requiring exploration of the solution through interaction with the environment.

A characteristic challenge of RL is the dilemma of exploitation and exploration. On one hand, the agent prefers to exploit actions that it estimates will yield high rewards. On the other hand, these highly rewarded actions can only be discovered by trying unexplored actions, represented by an epsilon-greedy policy or other exploration strategies.

Moreover, the design of the reward function holds critical importance. With sparse and delayed

rewards, the agent may only capture a positive learning signal after numerous trials. However, learning with a dense handcrafted reward function may prevent the agent from progressing toward the optimal policy. This can often be represented as an optimization problem, where the agent aims to maximize the discounted cumulative reward:

$$R = \sum_{t=0}^{T} \gamma^t r_t \tag{1.5}$$

where $R$ is the cumulative reward, $\gamma$ is the discount factor, $r_t$ is the reward at time $t$, and $T$ is the total number of steps.

### 1.2.5 Navigation

Reliable navigation of autonomous mobile robots, drones, and self-driving cars is an emerging area that has reached a level of maturity. While the concept of navigation, finding the optimal path from the robot's current location to the goal destination without hitting obstacles, is well understood, it is still challenging to make this work in dynamic scenarios. Standard robot navigation is divided into three parts: (i) mapping and localization module (simultaneous localization and mapping or SLAM), (ii) a global path planner, and (iii) a local planner (Planner), each relying on environment information (which can be a map or sensor readings) and the robot's position, respectively.

Although utilizing offline generated map simplifies planning and controlling the system, it also has its limitations. For instance, creating and updating a precise map is costly and can require information from human agents or need excessive computation. Besides, the map-based system's performance depends on the assumption that the environment is static, which is usually not the case in emergency response or military settings, which might not have an offline map at all.

To overcome the shortfall of only depending on navigation based on a map, a robot can create a direct connection between sensory inputs and robot actions. For global navigation, which usually cannot be managed by behavior-based navigation, a robot can make use of real-time environmental information, such as laser scan readings for mapping. However, depending on the specifications

of the sensor, it can be too computationally heavy for local computation, which could impair situational awareness in real-time, or can be communication-intensive, assuming offloading to a cloud-based server.

#### *1.2.5.1 High Computational Demand of LiDAR-based SLAM*

Table 1.1 shows how only relying on local computation can deteriorate system performance in a given theoretical scenario where,

- Local Hardware: Intel Core i7 machine with 16 GB RAM
- Lidar data generation rate: Velodyne HDL-64E S3 generates 2.2 million points per second (based on a 360-degree view, 10Hz rotation speed)
- Data point size: Assuming each data point contains x, y, z, and intensity - this would be roughly 16 bytes (4 bytes each for x, y, z as float values, and 4 bytes for intensity as a float value)
- For network transfer, we assume a fast internet connection with no latency

Even though the exact computational costs depend on various factors such as network connectivity, algorithm optimization, local hardware specifics, data rate, etc., we can get a sense of the system performance and limitations based on the theoretical scenario given above. Autonomous navigation heavily relies on real-time processing of sensory data, for example, millions of points generated (assuming the maximum range) by a Velodyne HDL-64E S3 LiDAR sensor every second. Each point contains x, y, z, and intensity values, contributing to a data volume of approximately 33.2 MB per second. Such high-frequency data impose a significant computational load on onboard systems. For instance, an Intel Core i7 machine with 16 GB RAM runs at about 85% CPU load and uses approximately 8 GB of memory to process the full range of this data locally.

### 1.2.6 Fog Robotics

Fog robotics is an emerging field that combines cloud computing and robotics to enhance the performance and capabilities of robots, thus offering a solution to the computational problem dis-

Table 1.1: Comparison of Local and Cloud Processing

| | Local Full Range | Local Reduced Range | Cloud Offloading |
|---|---|---|---|
| Data per second (MB) | 33.2 | 3.32 | 33.2 |
| Network transfer time (sec) | N/A | N/A | Variable* |
| Local CPU load (%) | 85 | 25 | 10 |
| Local Memory usage (GB) | 8 | 2 | Minimal |
| Local Hardware: Intel Core i7 machine with 16 GB RAM | | | |

cussed above. By offloading computationally intensive tasks to centralized servers in the cloud, robots with limited onboard computational resources can leverage the vast processing power and storage available in the cloud to perform complex tasks. This approach enables robots to access shared knowledge databases, learn from the experiences of other robots (federated learning), and benefit from computationally demanding tasks and algorithms that need to be updated continuously. Cloud robotics not only improves the overall efficiency of robots but also reduces their battery consumption and hardware requirements, making them more cost-effective and light weighted. By utilizing the extensive computational resources in the cloud, robots can achieve more precise outcomes and complete tasks that were locally unfeasible.

### 1.2.7 Shortcomings of Offloading to the Cloud

Offloading computation or storage to the cloud offers several benefits, such as enabling resource-constrained mobile robots to perform complex tasks more efficiently. However, these advantages come with certain risks that need to be considered. One of the primary costs associated with offloading is latency, which is the time delay experienced when waiting for results from the cloud. This latency can impact critically, especially in the area where real-time performance of robotic systems is necessary, especially where quick decision-making is crucial such as navigation, localization, or mapping tasks.

Another significant cost arises from the bandwidth limitations of wireless networks, such as cellular or Wi-Fi networks, which mobile robots typically rely on for communication. These networks can exhibit high levels of packet latency and often offer limited bandwidth for data trans-

mission. In many cases, the data being offloaded, such as 3d point cloud for precise navigation, can be quite large relative to the available bandwidth. As a result, data transmission may consume a significant portion of the network's limited resources, potentially causing congestion and further delays. Furthermore, there could be data loss due to a congested network.

Research related to cloud robotics often fails to address these offloading costs fairly. Some approaches involve offloading tasks without considering the network connection's state or varying conditions, which concludes that depending solely on cloud computation is the appropriate decision [4]. Other methods, such as Edge-Slam rely on networked resources for SLAM operation, which implies that these resources must be available at all times in real-time settings. Failure to provide adequate service can leave a robot unable to determine its current status, significantly impairing its ability to perform tasks [5].

In order to optimize offloading decisions and minimize costs, it is essential to develop new approaches that consider both the input features and the network conditions. It is thus important to account for these factors to ensure that cloud robotics systems can more effectively balance the benefits of offloading with the associated costs, ultimately leading to improved performance and efficiency.

## 1.3 Robot Navigation Terminology

Robot navigation can be divided into four tasks: Sensing, Localization, Mapping and Path Planning.

### 1.3.1 Sensing

Sensing focuses on extracting the geometrical relationship between the robot and the objects in the environment. Common technologies used for this include LiDAR, ultrasonic sensors, depth cameras, and vision-based sensors.

### 1.3.2 Localization

Localization provides the robot pose (i.e., its location and heading) in the map coordinate system. Localization can be achieved with or without prior knowledge about the environment.

Systems that require prior knowledge can use methods like fingerprinting techniques or signal propagation models [6]. In contrast, systems that don't require prior knowledge can use visual localization or laser range finders.

#### *1.3.2.1 Map Representation*

The environment can be represented by a map in different forms, like grid maps, landmarks, and point clouds.

#### *1.3.2.2 Simultaneous Localisation and Mapping (SLAM)*

SLAM is a method where the robot builds a consistent map of an unknown environment while simultaneously localizing itself on this map. SLAM is solved in different forms like probabilistic SLAM, EKF-SLAM, FastSLAM, and GraphSLAM. [7, 8]

### 1.3.3 Path-Planning

Path-planning algorithms generate a collision-free path to the destination with the least costs under certain criteria like time, energy, or the length of the path. It can be generalized into two categories: off-line and on-line path planning [9].

#### *1.3.3.1 Off-line Path Planning*

In off-line path planning, a full map of the environment is given, making it easier to search for the optimal path. However, building a full map is cost expensive and it assumes that the environment is static.

#### *1.3.3.2 On-line Path Planning*

On-line path planning has a wider application for robot navigation as it can tightly cooperate with SLAM algorithms. By integrating with Simultaneous Localization and Mapping (SLAM) algorithms, it allows the robot to simultaneously map unknown environments and determine its position within that map, enhancing its navigational capabilities and robustness against unforeseen obstacles or changing conditions.

#### *1.3.3.3 Dynamic Window Based Methods*

Dynamic window method discretely samples in the robot's control space and performs forward simulation for each sample velocity. Each trajectory is then evaluated and the highest-scoring trajectory is selected. This is usually used for local planners.

### 1.3.4 Related Work

In recent years, there has been a growing interest in developing efficient and robust algorithms for robotic systems, particularly in the context of reinforcement learning and cloud robotics. Tahir and Parasuraman (2022) [10] introduce the Analog Twin framework, which synchronizes two mobile robots to perform a goal navigation task. One robot acts as the master, and the other as the client, controlled by the master through a remote agent. The proposed framework aims to improve computing workload, network latency, and data throughput without affecting the control task performance. The paper validates the approach through extensive simulation experiments and real-world robot demonstrations, showing the framework's flexibility, versatility, and efficiency in realizing reactive planning applications with remote computational offloading capabilities compared to conventional offloading schemes.

Ali et al. (2023) [11] propose Edge-SLAM, a system that uses edge computing resources to offload parts of Visual-SLAM. The authors modify ORB-SLAM2 into a split architecture between the edge and the mobile device, keeping the tracking computation on the mobile device and moving the rest of the computation to the edge. This split architecture allows the functioning of the Visual-SLAM system long-term with limited resources without affecting the accuracy of operation. The computation and memory cost on the mobile device is kept constant, enabling the deployment of other end applications that use Visual-SLAM.

Shi et al. (2020) [12] provides a comprehensive survey of recent developments in efficient techniques for edge AI systems. The paper discusses algorithmic and system techniques for training and inference tasks at the network edge, presenting various communication-efficient techniques such as federated learning, quantization, sparsification, knowledge distillation, model and data

parallelism, compressed sensing, and edge caching. These techniques address key communication challenges in edge AI systems, including varying channel quality, traffic congestion, privacy concerns, and high energy consumption.

Chinchali et al. (2021) [3] address the problem of offloading sensing tasks to cloud resources for robotic systems, proposing a cost-based cloud offloading problem and deep reinforcement learning-based solutions to handle diverse network conditions and flexibly trade-off robot and cloud computation. The proposed offloading strategy demonstrates significant improvement in object detection compared to standard approaches, reducing the cost of cloud communication while maintaining high sensing accuracy.

The related studies offer essential insights into the creation of effective and reliable algorithms for robotic systems, emphasizing the potential of reinforcement learning, cloud robotics, edge computing, and communication-efficient methods in tackling the challenges associated with robot navigation in dynamic settings. However, many papers fail to consider the situation of network failure or intense use of network communication which could deteriorate the result proposed by the papers. Chinchali et al. (2021) [3] try to formulate the research to take into account the network's capability. However, it has approached for classification for object detection, which misses the core of the cloud robotics which is the navigation system. This research is primarily built upon Chinchali et al. (2021) [3]. While their work concentrated on the quality of service (computation for classification), we have opted to modify the problem to focus on the quality of information (mapping) that can be received concerning the cloud's ability to perform more heavy computation with larger scan data.

# 2. PROBLEM STATEMENT

## 2.1 Challenges

Deep Reinforcement Learning (DRL) has shown impressive results in environments such as gaming and simulation environments. However, replicating this success in real-world situations, which are often noisy and dynamic, is still challenging. The complexity of real-world environments results in high variance in data distribution and several factors contribute to this problem.

The first is the 'Curse of Dimensionality', which refers to the exponential increase in data and computation required for a high-dimensional robot system. This requires exploring a vast state-action space, making it necessary for the accumulation of numerous learning samples through interaction with the real-world environment. However, almost all of the robot reinforcement learning problems cannot rely solely on real-world gained datasets. This is because robots are expensive and can be easily damaged during real-world training. Also, obtaining training episodes from real robots is slow and often requires much human aid.

Training in a simulated environment and then adapting the learned policy to real-world scenarios appears to be a more practical approach. This method reduces the training period by allowing for accelerated simulation and easy resetting of the virtual world after a collision, avoiding real damage. However, simulation comes with its own set of drawbacks, including the issues of under-modeling reality. Because of the ' reality gap ', these problems make it difficult to directly transfer the learned policies from the virtual to the real world. Thus, one of the primary tasks is to narrow this gap between the virtual and real worlds, especially in conditions of sensory observations such as visual appearance, physics and robot kinetics.

In the world of robotics, especially in dynamic settings, another significant issue arises - the generalization ability to transfer learned experiences from familiar to unfamiliar scenarios. This ability is crucial because it is often challenging for a robot to determine the optimal action when it finds itself in an unknown state. The difficulty comes from the fact that the reward function subtly

conveys the desired behavior in reinforcement learning. As such, crafting an appropriate reward function is essential but can be quite a challenging task in real-world robotic problems.

A classic contradiction in designing reward functions lies between sparse and dense rewards. Dense rewards, grounded on prior expert knowledge, can significantly expedite the training process. However, this approach could lead the agent towards learning a sub-optimal policy for the given task. This occurs because the frequent, information-rich rewards might restrict the agent's exploration of the state-action space, limiting its understanding and decision-making abilities.

Contrarily, sparse reward functions push the agent to engage in extensive random exploration. It gains positive experiences only after observing a wide variety of negative rewards. In other words, rather than learning directly how to perform the task (positive reward), it initially learns how not to do the task (negative reward). This learning methodology inevitably slows down the convergence of the DRL policy. Despite the slower learning curve, this approach can potentially lead to more robust and generalized policies by forcing the agent to explore a wider variety of states and actions. The harmonic balance is required in reward function design to promote both effective learning and optimization.

## 2.2 Research Questions

To develop a system for offloading policies for enhanced situation-aware robot navigation systems using DRL, particularly in dynamic real-world environments, this thesis endeavors to tackle several critical questions. These questions provide a pathway to conceptualizing a system that effectively leverages the power of Deep Reinforcement Learning (DRL) to enhance the capabilities of autonomous robots, thereby improving their navigation and situational awareness.

**A. Can we design an effective reward function for the Reinforcement Learning algorithm?**

The first critical point of inquiry this thesis seeks to address is the formulation of a suitable reward function. The reward function is the core of the reinforcement learning algorithm, directing its learning trajectory by providing feedback on the quality of the agent's actions. Balancing the sparsity and density of the rewards is a challenge - with a reward function that's too sparse, the

agent must extensively explore and primarily learn from negative rewards, slowing the learning process. Conversely, a dense reward function could expedite learning but risk leading the agent towards sub-optimal policies. The challenge lies in creating a reward function that optimally encourages the agent toward the most effective decisions.

**B. Can we create the Most Suitable Deep Reinforcement Learning Algorithm for the Proposed System, and How Does It Compare to Other Policies?**

The second question to be explored is the selection of the most appropriate DRL algorithm that can effectively guide the training to reach the desired goal. Moreover, it is vital to evaluate the performance of DRL algorithms against other policies. For this investigation, the thesis proposes a comparative study of Deep Q Networks (DQN) against practical policy approaches. Such a comparative analysis will provide a comprehensive understanding of the strengths and weaknesses of DRL and other methods in various navigation scenarios.

**C. How Can We Ensure That a Policy Learned in a Simulated Environment Translates Effectively into Real-World Applications?**

The ultimate question to be addressed by this thesis involves determining the robustness and capability of the policies learned in a simulated environment to be reliable when applied to real-world settings. This objective aligns with the intention to diminish the 'reality gap' which is a significant issue when transitioning from simulations to real-world operations. It is vital to plan strategies that allow for effective transfer learning, such that the DRL-based system can successfully adapt and navigate in diverse and unpredictable real-world environments.

### 2.3 Proposed Work

In this section, we outline our proposed work for the development of an efficient offloading policy for a robot operating in a cloud robotics scenario. We will focus on the following four key aspects:

### 2.3.1 Sensory Input and Computation Models

We will investigate various sensory input formulation and computation models for both the robot and the cloud. Our goal is to design efficient models that accurately capture the environment's dynamics and uncertainties while minimizing computational and communication costs. This will involve the following steps:

Designing the sensory input space, denoted as $x_{t\,t=0}^{T}$, where $x_t$ represents the data (e.g., laser scan) arriving at time $t$. We will consider different types of sensors and input data relevant to the specific robotic application.

Developing computation models for both the robot and the cloud. We will denote the robot's model as $f_{\text{robot}}$ and the cloud's model as $f_{\text{cloud}}$. These models will map input queries $x_t$ to mapping computation of $y_t$ and their information gain scores $\inf$:

$$\hat{y}_{\text{robot}}, \inf_{\text{robot}} = f_{\text{robot}}(x_t) \tag{2.1}$$

$$\hat{y}_{\text{cloud}}, \inf_{\text{cloud}} = f_{\text{cloud}}(x_t) \tag{2.2}$$

### 2.3.2 Offloading Markov Decision Process Formulation

We will formulate the robot offloading problem as a Markov Decision Process (MDP) to model the sequential decision-making process under uncertainty. The MDP will be defined as

$$M_{\text{offload}} = (S_{\text{offload}}, A_{\text{offload}}, R_{\text{offload}}, P_{\text{offload}}) \tag{2.3}$$

,

State Space $S_{\text{offload}}$: The state space consists of the information necessary to make the decision between local processing and cloud offloading. This decision should be based on the current sensory input $x_t$ that includes:

- Front 60-degree scan range: $s_1 = \frac{\text{front\_60\_degree\_scan\_range}}{\text{max\_laser\_range}}$

- Current velocity: $s_2 = \frac{\text{current\_velocity}}{\text{max\_velocity}}$
- Temperature: $s_3$

The situational factors to be considered are:

- Distance to goal: $s_4 = \frac{\text{distance\_to\_goal}}{\text{maximum velocity}}$
- Angle to goal: $s_5 = \text{angle\_to\_goal}$
- Distance to network edge: $s_6 = \frac{\text{distance\_to\_network\_edge}}{\text{network communication range capacity}}$

We also consider the accumulated occupancy grid from the robot's sensors, denoted as $f_{\text{robot}}(x_{\tau_{\text{robot}}})$, and the "staleness" of the last cloud prediction, denoted as $\Delta t_{\text{cloud}} = t - \tau_{\text{cloud}}$.

Therefore, the state in the offloading MDP can be formally defined as:

$$s_{\text{offload}} = [s_1, s_2, s_3, s_4, s_5, s_6, f_{\text{robot}}(x_{\tau_{\text{robot}}}), \Delta t_{\text{cloud}}]$$

In this representation, $s_1$ to $s_6$ are the normalized sensory inputs and situational factors, $f_{\text{robot}}(x_{\tau_{\text{robot}}})$ represents the accumulated occupancy grid, $\Delta t_{\text{cloud}}$ is the time since the last query to the cloud.

The raw sensory input $x_t$ may be high-dimensional; including it directly in the problem state could result in a large state-space. Thus, we opt for using features of the inputs instead. These features are the max and min feature of the input and their choice depends on the impact of the robot's perception.

In this research, we have taken an approach to ensure that our work is transferable to different robotic systems with varying specifications without additional training. Specifically, we have normalized many of the key state factors that are essential to our learning models, such as scanning range, velocity, and distance. By doing so, these relative factors can provide a measure of state space that is not tied to a specific set of hardware or operating conditions.

This step is crucial for promoting the readiness of our model for transfer learning - a technique that leverages a pre-existing model's knowledge and applies it to a new, yet related situation. By

normalizing these crucial factors, we essentially create a flexible model that can adapt to different types of real-world robots that may have differing specifications.

This normalization not only enhances the model's flexibility but also contributes to the efficiency of the transfer learning process by simplifying the task of adapting the model to new conditions and reducing the time and resources required for additional training. Hence, this research ensures a robust and feasible solution that can readily serve to a wide range of robotic systems and applications.

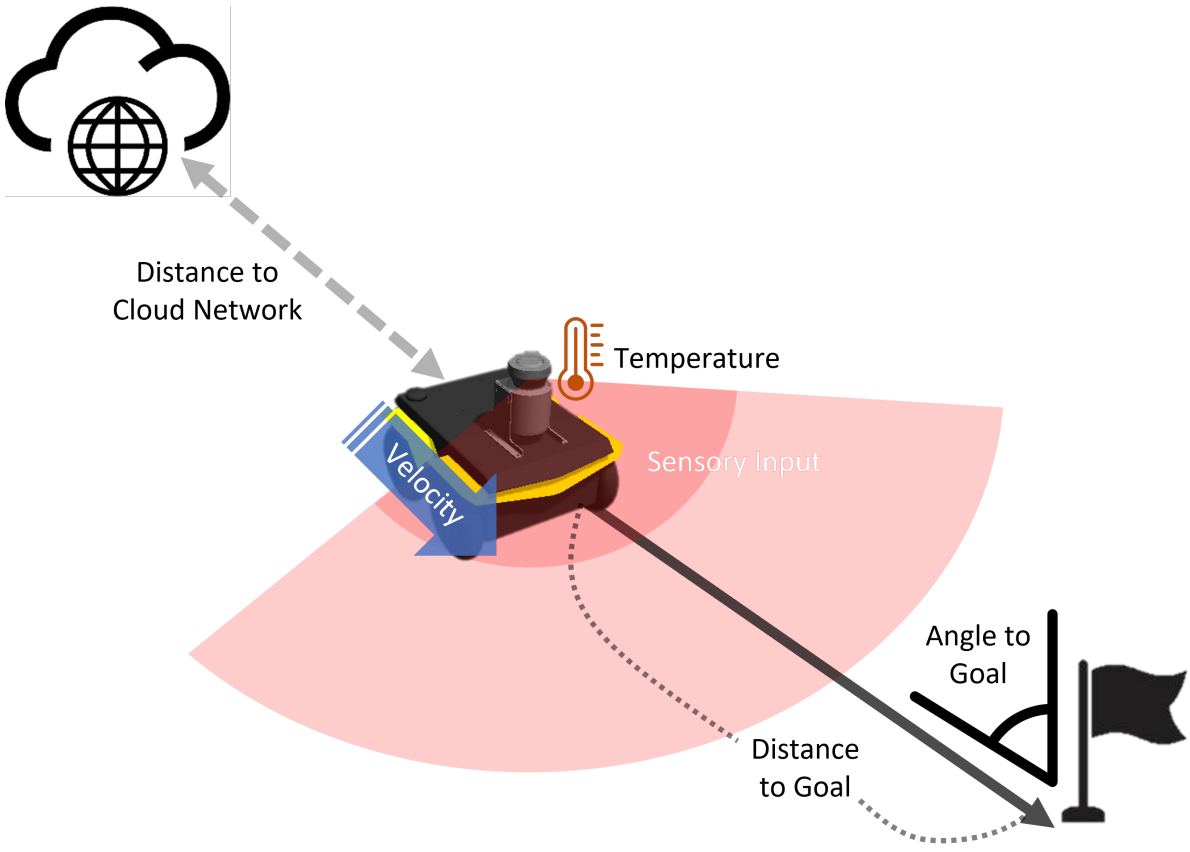


Figure 2.1: State Space Definition

### 2.3.3 Action Space

The action space for the Clearpath Jackal robot in the context of offloading computation for LiDAR scanning consists of two actions. Each action corresponds to making use of a particular computation resource to process the LiDAR data: either locally on the robot or remotely in the cloud. The action space consists of two discrete actions:

- Action 0: Use local computation for laser scan inputs (1.2m)
- Action 1: Use cloud computation for larger laser scan inputs (12m)

In Action 0, the robot utilizes its onboard computational resources, specifically the Intel Core i3 processor, to process the LiDAR data. This action would typically be more power-efficient as it avoids the energy consumption associated with data transmission for offloading. However, using local computation might restrict the ability to gain better situation awareness, which could be a limiting factor when the robot navigates to the goal, such as encountering a dead-end.

In contrast, Action 1 involves offloading the computation task to the cloud for further laser scan readings. This action allows the robot to access potentially more powerful computational resources, enabling it to process more extensive data from larger laser scan data. This could be especially beneficial in complex environments where a broader view is essential for effective navigation. Nonetheless, cloud computation introduces additional latency due to data transmission, and it also increases power consumption due to both the transmission process and the maintenance of a connection with the cloud while also executing local mapping for safety reasons. The reward function, therefore, needs to be designed in a way to balance these trade-offs, considering the power consumption, computational capability requirements and ability to gain a better understanding of the surrounding environment.

Action Space $A_{\text{offload}}$: The action space of the offloading decision-making problem is defined as the choice of which prediction $\hat{y}$ to use at each time step $t$. The offloading system can choose to use local predictions, a combination of local computation on the new input $x_t$ combined with past mapping computations that are an integration of past robot and cloud (if executed) predictions,

or querying the cloud model $f_{\text{cloud}}$ for better information gain. Specifically, we define one action $a_{\text{offload}}$ which can take 2 discrete values:

$$a_{\text{offload}} = \begin{cases} 0, & \text{use local computation } \hat{y}_t = f_{\text{robot}}(x_t) = f_{\text{robot}}(x_{\tau_{\text{robot}}}) + f_{\text{cloud}}(x_{\tau_{\text{cloud}}}) \\ 1, & \text{query cloud for computation } \hat{y}_t = f_{\text{cloud}}(x_t) \end{cases}$$

In this definition, $\tau_{\text{robot}} < t$ is the last time the robot model was computed, and $\tau_{\text{cloud}} < t$ is the last time the cloud model was queried.

Reward Function $R_{\text{offload}}$: A function that captures the trade-offs between network latency, energy, temperature(affecting computation capability), and situation(distance gain, collision), as well as the information gained by the action. We will discuss this in more detail in the following chapter.

Dynamics $P_{\text{offload}}$: The transition probabilities between states in the MDP, accounting for the impact of the robot's actions and the environment's dynamics.

## 2.4 Reinforcement Learning-based Navigation Algorithm

In this section, we propose a reinforcement learning-based navigation algorithm for a mobile robot. The algorithm aims to navigate the robot to a goal position while avoiding obstacles and efficiently requesting for cloud computation (the use of a larger scan). we implement a Deep Q-Network (DQN) that is trained using the robot's state and actions taken. The reward function is an essential component of the algorithm. Therefore, We will discuss the detail of the reward function in the following four subsections:

### 2.4.1 Leveraging Dense and Sparse Rewards

Crafting a reward function that allows for efficient learning is a challenge in RL-based approaches. Typically, a dense reward function may result in quicker convergence and better exploitation. However, it might sometimes lead the agent to find sub-optimal solutions. On the other hand, a sparse reward function can push the agent to discover more optimal solutions but might

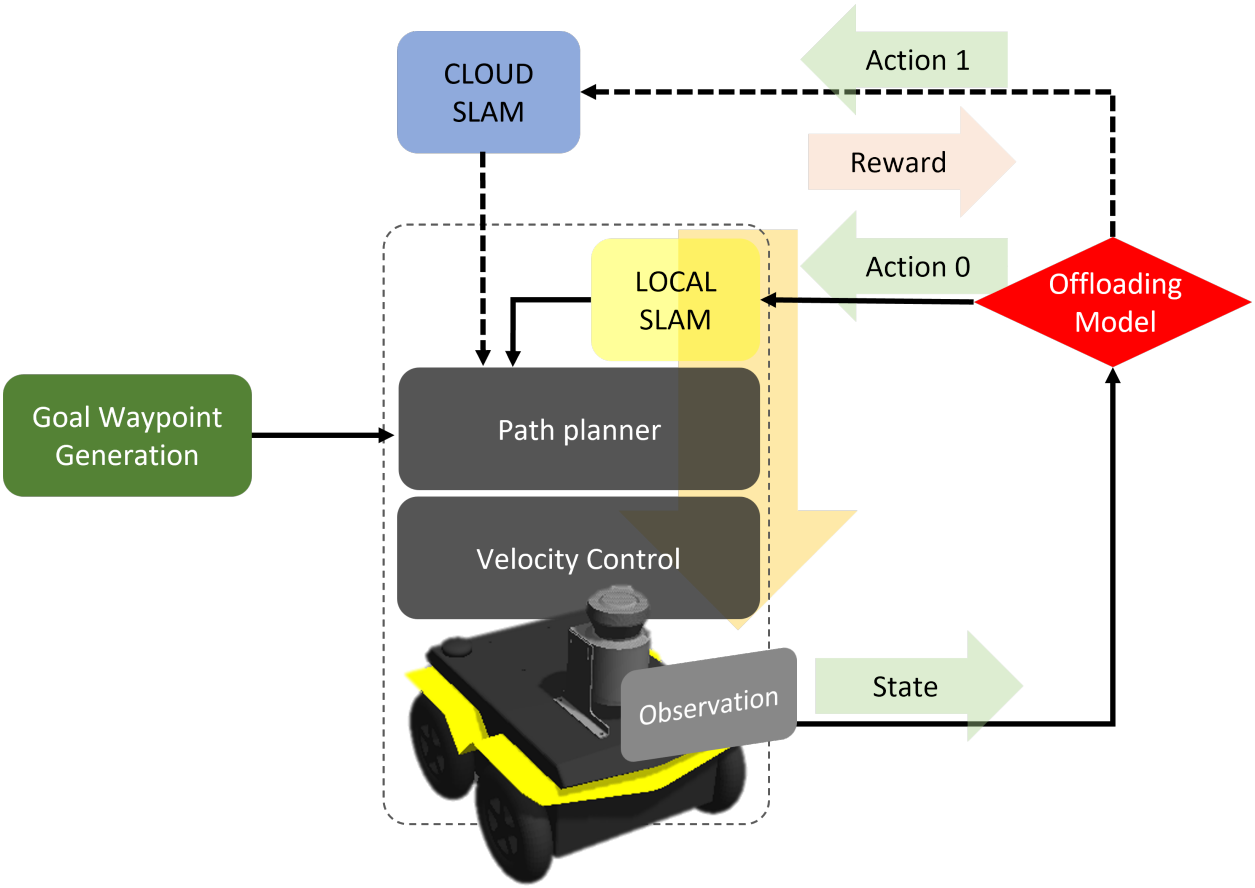


Figure 2.2: Robot Mapping Offloading MDP

also make the learning process more difficult due to a delayed reward signal.

In this scenario, we consider balancing between dense and sparse rewards by integrating both into the reward function. We can use dense rewards for the efficient navigation of the robot, such as approaching the goal and a sparse necessity of offloading. This allows for effective exploration and exploitation, encouraging the agent to find the optimal solution while ensuring a feasible learning curve.

### 2.4.2 How to interpret Network Communication Strength

The equation below is an approach to calculating the channel capacity between two nodes in a wireless network. The channel capacity is the maximum amount of data that can be transmitted

over a communication channel in a given amount of time. In Zhu et Al. [13], it is represented as

$$r_{i,j} = B_{i,j} \log_2 \left(1 + \frac{P\,(d_{i,j})^{-\alpha}}{N_j \sigma^2}\right) \tag{2.4}$$

where $r_{i,j}$ is the channel capacity between nodes i and j, $B_{i,j}$ is the bandwidth available for communication between nodes i and j, $P$ is the transmit power, $d_{i,j}$ is the distance between nodes i and j, $\alpha$ is the path loss exponent, $N_j$ is the number of interfering nodes in the vicinity of node j, and $\sigma^2$ is the noise power which is trivial in most of the scenario.

The signal power is calculated as $P\,(d_{i,j})^{-\alpha}$, which means that the signal power decreases as the distance between nodes i and j increases. The noise power is represented by $\sigma^2$, which is a constant value that represents the background noise in the communication channel.

### 2.4.3 Linking Network Communication Strength to Reward Function Design

The aforementioned equation can play an instrumental role in developing a suitable reward function in the context of offloading computations, especially considering network latency. The formula essentially represents the maximum rate of successful information transmission between two nodes, given particular network conditions. For offloading computations, this rate directly affects the time it takes to send a computational task from one node to another and subsequently receive the results. This transfer time is a significant component of network latency, and thus, higher channel capacity, denoted as $r_{i,j}$, can lead to reduced latency.

In this research, to simplify the computation of network strength, we only consider the distance between the nodes, assuming that other factors such as transmit power, bandwidth, path loss exponent, the number of interfering nodes, and noise power are constant. This simplification leads to a modified version of the channel capacity formula.

$$r_{i,j} = f(d_{i,j}) = \frac{1}{s_6} = \frac{\text{network communication range capacity}}{\text{distance\_to\_network\_edge}} \tag{2.5}$$

where $r_{i,j}$ is the simplified channel capacity between nodes i and j, and $d_{i,j}$ is the distance between nodes i and j. The function $f$ represents the inverse relationship between distance and

channel capacity.

### 2.4.4 Ultimate Reward Function Incorporating Situational Awareness

The reward function is designed to guide the agent towards the optimal path to the goal by giving appropriate information about the surrounding environment with respect to leveraging offloading efficiently, considering network capacity, environmental information, and temperature effects. We introduce an offloading benefit factor and an environment temperature factor in the reward function.

$$R_{\text{offload}} = - r_{\text{collision}} - r_{\text{distance\_gain}} - r_{\text{time taken(computation per timestep)}} - r_{\text{large cloud execution time}} + r_{\text{offload\_benefit x temperature\_impact}}$$

where each reward component is defined as:

- $r_{\text{collision}}$, for when collision is detected
- $r_{\text{distance}}$, the robot's approach to the goal, penalty if otherwise
- $r_{\text{time taken}}$, penalty for each timestep taken
- $r_{\text{large cloud execution time}}$, penalty if the time since the last cloud offloaded scan range is too stale that robot went over cloud scanned area
- $r_{\text{offload\_benefit x temperature\_impact}}$, measure of the increased situational awareness (i.e., knowledge of the occupancy grid) that offloading provides

The offloading benefit factor $r_{\text{offload\_benefit}}$ is a measure of the increased situational awareness (i.e., knowledge of the occupancy grid) that offloading provides. A more aware robot can make better navigation decisions, potentially reducing the path length to the goal. We can model this as:

$$r_{\text{offload\_benefit x temperature\_impact}} = k_1 \times \frac{(\text{Occupancy\_grid}_{\text{current}} - \text{Occupancy\_grid}_{\text{past}})}{|T - T_{\text{optimal}}|} \times r_{i,j} \quad (2.6)$$

where $k_1$ is a constant. $T$ is the current environmental temperature, $T_{\text{optimal}}$ is the optimal operating temperature for the robot. It is critical to note that when the temperature deviates significantly from the optimum, either high or low, the robot's computational abilities are detrimentally affected. This situation consequently leads to an increase in workload associated with offloading computations which is in addition to the local SLAM (Simultaneous Localization and Mapping).

Also, as previously noted, the accessibility of the network decreases in line with the increasing distance from the network node. As a result, we've chosen to adjust the information gain derived from offloading, taking into account both the factors of temperature and distance. In this context, a larger positive value would be indicative of the benefits from offloading computations being significantly greater. Essentially, the additional computations of offloading prove advantageous under these circumstances.

Therefore, to maximize the efficiency and the computational capability of the robot, maintaining optimal temperature and minimizing the distance to the network node is highly desirable. The interplay between these factors is pivotal for the robot's performance, and understanding this relationship could lead to more efficient design and utilization of robotic systems.

This reward function promotes efficient offloading and safe navigation, taking into account both the benefits of offloading for situational awareness and the impacts of environmental temperature on the operation.

# 3. METHODOLOGY

## 3.1 Project Setup

For the purpose of this research, we adopt an experimental setup in an indoor robotics setting. Specifically, we utilize the Jackal, a four-wheeled robot developed by Clearpath Robotics, in conjunction with the RPLidar A2 sensor. Our setup can be seen as a scaled-down version of a full-size outdoor vehicle with a Velodyne LiDAR that has ten times the range. This setup enables us to carry out our research effectively while remaining within feasible operational parameters. The specifications of the JACKAL robot are given in Table 3.1.

Table 3.1: JACKAL Specifications

| **Feature** | **Specification** |
|---|---|
| External Dimensions (L x W x H) | 508 x 430 x 250 mm (20 x 17 x 10 in) |
| Weight | 17 kg (37 lbs) |
| Max. Payload | 20 kg (44 lbs) |
| All-Terrain Payload | 10 kg (22 lbs) |
| Max. Speed | 2.0 m/s (6.6 ft/s) |
| Battery Chemistry | Lithium Ion |
| Capacity | 270 Watt hours |
| Run Time | 2 8 hours |
| Performance | Intel Core i5 4570T, Dual core, 2.9GHz, 4 GB RAM, WIFI |
| Operating Ambient Temperature | -20 to 45 °C (-4 to 113 °F) |

### 3.1.1 Motivation for Utilizing Deep Reinforcement Learning

As part of the approach to address the challenge of data offloading, deep reinforcement learning (RL) presents itself as a powerful tool for several reasons.

RL, in particular, is advantageous because it is a model-free policy search method, meaning it is not necessary to model the system's complex dynamics. Instead, RL focuses on learning optimal offloading policies from the features present in the state, without attempting to predict incoming

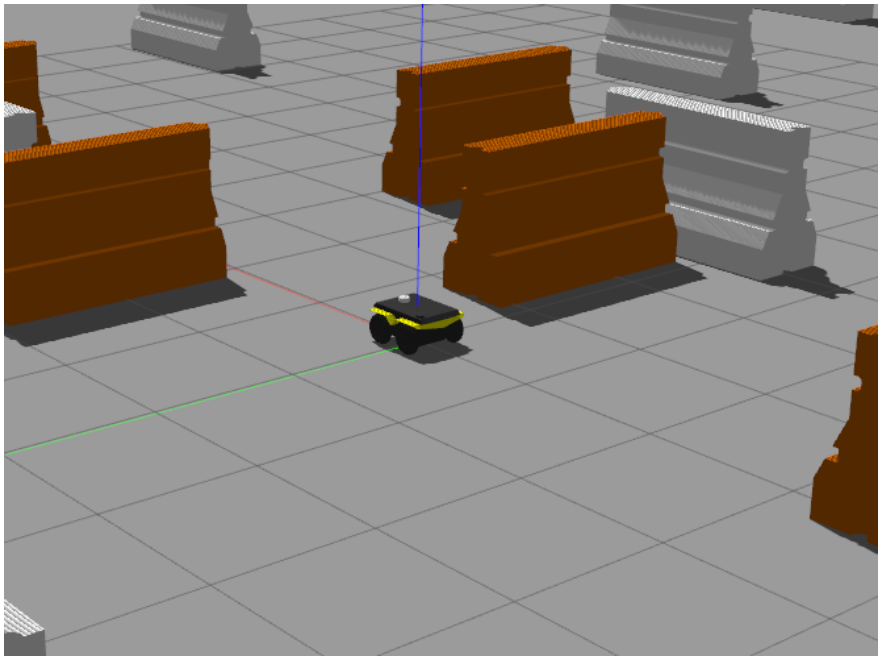

Figure 3.1: Jackal Robot

sensory data.

The RL framework further includes a recurrent policy. This policy aids in the superior estimation of latent variables that give context to incoming inputs. Given that the state vector consists of features from only the two most recent inputs, this recurrent policy accounts for the possible non-Markovian nature of the state, enabling more accurate modeling of the sensory data stream.

RL presents a host of other benefits when used to determine effective offloading policies. One such benefit is its ability to manage stochastic rewards simply and effectively. Our adopted reward function, which might exhibit stochastic behavior due to varying network conditions or the fluctuating cost of computation, is adequately handled within this framework.

Another significant advantage of RL is that it offers a cost-effective method for policy evaluation. Unlike other methods such as model predictive control[14] that require evaluating dynamics and optimizing action selection, an RL-based approach only necessitates the evaluation of a neural network.

The evaluation of the policy forms an integral part of the perception onboard the robot. There-

fore, the efficiency of this evaluation directly impacts latency. To this end, we represent the RL offloading policy with a deep neural network and employ the DQN algorithm for training. We will elaborate on the training procedure in the following section.

## 3.2 Deep Q-Learning Algorithm Implementation

In this research, we implemented a reinforcement learning mechanism using a Deep Q-Learning algorithm and a replay buffer, utilizing the Robot Operating System (ROS) in a navigation environment.

### 3.2.1 Replay Buffer

The Replay Buffer creates a storage repository for the experiences of the agent. Each experience is a tuple defined by five elements: the state of the environment, the action taken by the agent, the reward received, the subsequent state after taking the action, and a done status indicating whether the episode has ended. These experiences are stored in a deque with a predefined maximum size. The structure allows both the addition of new experiences and the ability to randomly sample a batch of experiences, which is critical for model training.

## 3.3 Navigation Environment

We construct a new navigation environment, referred to as NavigationEnvROS, using ROS to interact with robot, Jackal. This environment accepts a priority parameter that influences the agent's behavior based on the set preference. These preferences can be oriented towards awareness, balance, or cloud efficiency, affecting the navigation strategy of the agent.

### 3.3.1 Gazebo: The Simulation Environment

To train our RL model, we used Gazebo, a widely used simulation platform, to provide a virtual environment for our Jackal agent. Gazebo provides accurate simulation with real-world physics and execution of sensory data, making it able to test and train our agent's behavior prior to real-world deployment, which can be costly.

A challenge encountered during this research was that the predefined RL training frameworks

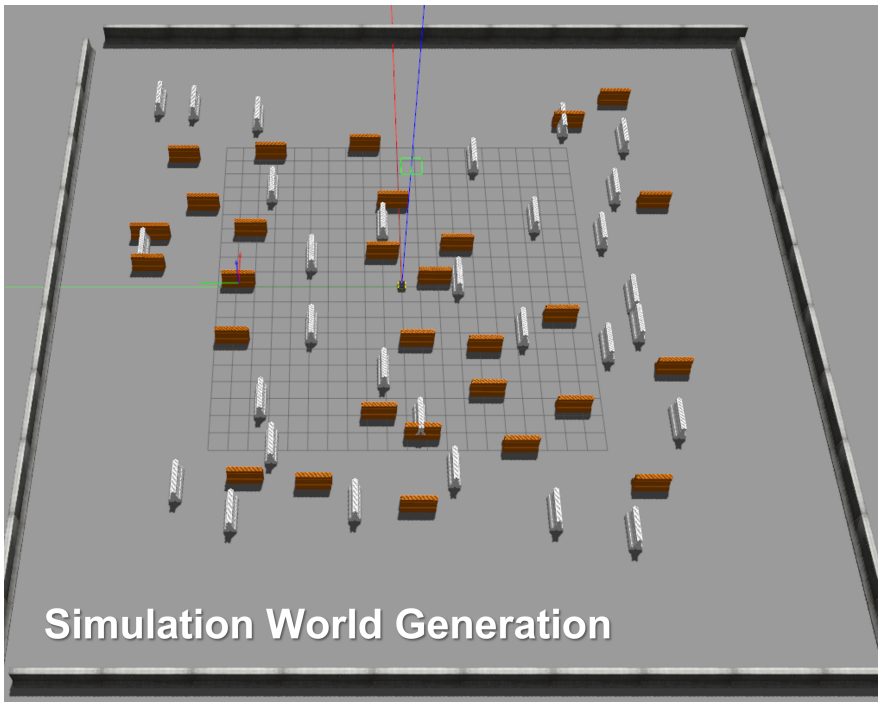


Figure 3.2: Gazebo Simulation World for Policy Training

available in Gazebo, such as 'openai$_r os'$, $mainly focus on training the robot controller. Moreover, they tend to ass$ $world settings$[15]$. In response to these issues, we developed an independent Gazebo environment capable of inc$ $trained policy compared to other policies, we adopted the default path-planning algorithm provided in ROS pac$ $planning process.$

Furthermore, Gazebo's flexibility allowed us to simulate various environmental conditions, including different obstacle configurations and network conditions(which were assumed as the distance to initial position). This facilitated a comprehensive evaluation of our agent's performance under diverse scenarios, providing safe and effective output to guide the iterative refinement of our reinforcement learning model.

This simulation environment enabled the training of our agent to ensure that the resulting policy was robust, adaptable, and well-suited for real-world deployment.

## 3.4 Experiment Scenario: Mapless Navigation Framework

The experimental scenario (see Figure 2.2 for a visual reference) has been designed as a mapless navigation task for a robot, which must reach a specific target waypoint by utilizing a sensory input stream, denoted as $x_t$. This stream consists of laser scan data, which the robot subsequently processes through Simultaneous Localization and Mapping (SLAM) using the gmapping algorithm.

In this context, the robot is faced with a decision problem. It has to balance between utilizing a local sensor reading that covers one-tenth of the maximum range, which could ensure system sustainability and efficiency in terms of computational resources and battery life, and offloading the full range laser data to a cloud server for more comprehensive SLAM processing.

$$s_{\text{offload}} = [s_1, s_2, s_3, s_4, s_5, s_6, f_{\text{robot}}(x_{\tau_{\text{robot}}}), \Delta t_{\text{cloud}}]$$

Where:

- $s_1$ represents the normalized front 60-degree scan range
- $s_2$ represents the normalized current velocity
- $s_3$ represents the current temperature
- $s_4$ represents the normalized distance to the goal
- $s_5$ represents the angle to the goal
- $s_6$ represents the normalized distance to the network edge
- $f_{\text{robot}}(x_{\tau_{\text{robot}}})$ represents the accumulated occupancy grid
- $\Delta t_{\text{cloud}}$ represents the time since the last query to the cloud

The reward function is designed to balance between accurate situational perception and minimizing both onboard computation and cloud query costs. The costs for both action 0 and action 1

were determined through extensive cost-benefit analysis, taking into consideration network conditions, computation overhead, and other variables.

### 3.4.1 Deep Q-Network

A Deep Q-Network (DQN) model is used to approximate the action-value function, also known as the Q-function. This function estimates the expected return for each possible action given the current state. The DQN model is a fully connected neural network with two hidden layers, featuring the Rectified Linear Unit (ReLU) activation function and an output layer that uses a linear activation function.

### 3.4.2 Training and Interaction

During each episode, the agent interacts with the environment based on an epsilon-greedy policy. The exploration rate, epsilon, initially starts from a pre-defined value and decays linearly to zero across the episodes. Based on this rate, the agent selects actions. If a random number is less than epsilon, the agent explores by selecting a random action. Otherwise, it exploits by selecting the action with the highest estimated Q-value.

After the agent takes action, the new state of the environment, the reward, and the done status are received from the environment. This new experience is then added to the replay buffer. If the buffer has sufficient data, a batch of experiences is sampled and used to update the DQN model. This involves predicting the Q-values for the next states and calculating the target Q-values for the actions taken. The model then undergoes a training step where it learns to match the current Q-values to these calculated target Q-values.

The cycle continues until the agent reaches a terminal state, indicated by the done status being true. At this point, the total reward of the episode is recorded, the trained model is saved, and the next episode begins.

This methodology of implementing a reinforcement learning agent in a navigation task uses deep Q-learning with a replay buffer and an epsilon-greedy exploration strategy, promoting efficient learning and adaptability in varied scenarios.

### 3.4.3 Challenges of Standard DQN and Adoption of Target Networks

The implementation of the standard Deep Q-Network (DQN) algorithm can face difficulties when dealing with large state spaces, as its learning efficiency tends to diverge due to inherent instability. This instability primarily emerges from the correlations present in the sequence of observations. To mitigate this, one common approach is the incorporation of a separate network, known as a target network, for evaluating actions. The target network's weights are not updated at each time step like the primary network but are instead periodically updated to match the primary network's weights. Mathematically, this can be represented as:

$$\theta_{\text{target}} \leftarrow \tau\theta_{\text{primary}} + (1-\tau)\theta_{\text{target}}$$

where $\tau$ is a hyper-parameter determining the rate at which the target network is updated. However, the utilization of a target network can lead to the overestimation of Q-values, which could result in the policy falling into a suboptimal state.

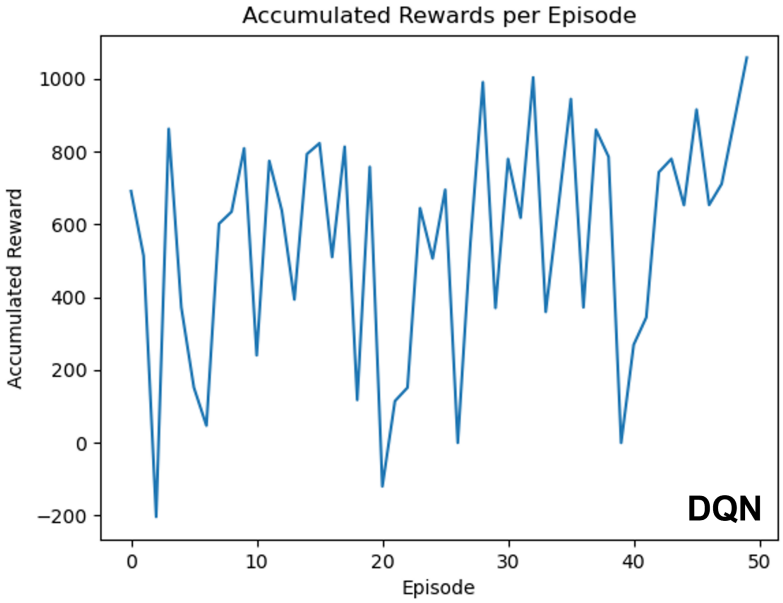


Figure 3.3: Reward Plot for DQN after 50 episodes

### 3.4.4 Mitigation of Overestimation Bias with Double DQN

To address the overestimation bias that can arise in a standard DQN, the Double DQN variant is employed. In contrast to the standard DQN, where the $\max$ operator is used to calculate the target Q-value and can lead to overly optimistic Q-value estimates, Double DQN decouples the selection and evaluation of actions by using two networks. Specifically, the online network is used to choose the action, and the target network is then used to estimate the Q-value of the chosen action.

In equation form, this update rule is represented as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[ r + \gamma Q'(s', \text{argmax}_{a'} Q(s',a')) - Q(s,a) \right]$$

where $Q$ is the online network, $Q'$ is the target network, $\alpha$ is the learning rate, $\gamma$ is the discount factor, $s$ and $a$ are the current state and action, and $s'$ and $a'$ are the next state and action. This decoupling method aids in reducing the overestimation of Q-values, contributing to a more accurate and efficient learning process, and leading to better policy performance.

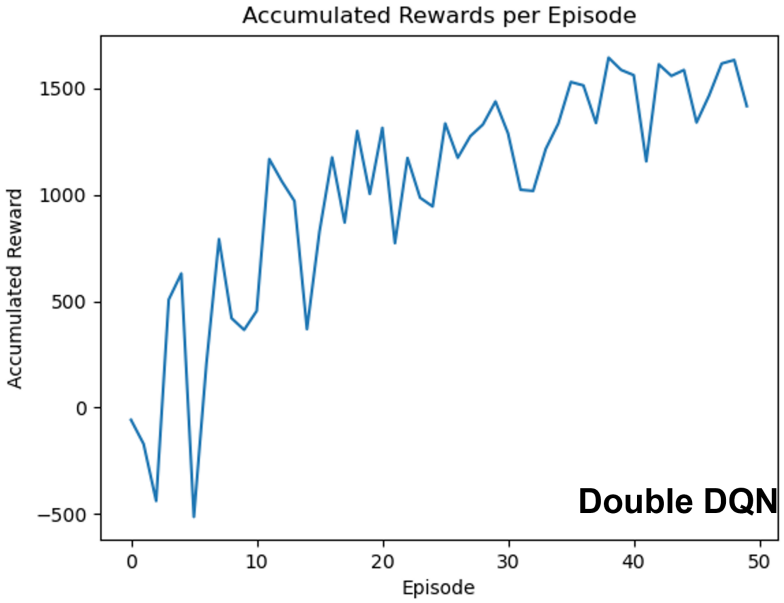


Figure 3.4: Reward Plot for Double DQN after 50 episodes

### 3.4.5 Proximal Policy Optimization and Limitations

In addition to the DQN and its variants, another popular approach for reinforcement learning is the Proximal Policy Optimization (PPO) algorithm. PPO falls under the category of policy gradient methods, which aim to optimize the policy directly rather than approximating a value function.

PPO aims to address the challenge of policy optimization by minimizing the divergence between the old and new policies during each update. This is achieved by using a surrogate objective function which adds a penalty term to the original policy gradient objective, encouraging the optimization to stay 'proximal' to the old policy.

The objective function for PPO is represented as:

$$L(\theta) = E\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

where $r_t(\theta)$ is the probability ratio, $\pi_\theta$ is the policy parameterized by $\theta$, $\hat{A}_t$ is an estimate of the advantage function at time $t$, and $\epsilon$ is a hyperparameter that controls the penalty for deviation from the old policy.

Our PPO implementation utilized two neural networks - an actor model and a critic model. Both models are deep neural networks consisting of two dense layers with 64 and 32 neurons respectively, each followed by ReLU activation function. The actor model ends with a softmax activation function to provide a probability distribution over actions, while the critic model ends with a single node to output the value function. We trained both models using the Adam optimizer, with a learning rate of 0.001.

Despite its merits, our application of PPO faced a challenge in the decision-making process. The model seemed to lean towards a sub-optimal policy that only performed cloud computation of the laser scan data. This was likely due to the agent's inability to adequately explore and exploit the benefits of selective cloud offloading (because cloud offloading can benefit for some time period), leading to a bias towards only cloud computation. The issue emphasizes the importance of exploration in reinforcement learning and the potential need for strategies that ensure a more

balanced exploration-exploitation trade-off in such complex decision-making scenarios.

### 3.4.6 Limitations in Training and Mitigation Strategies

The training process in our self-constructed Gazebo Framework presents certain limitations. Especially, there are limitations in terms of speeding up the training beyond certain thresholds. This poses a potential challenge, given that Deep Reinforcement Learning (DRL) requires multiple episodes to achieve convergence, potentially resulting in a slow progression toward the intended goal.

To counter these limitations, we have implemented a series of mitigation strategies. First, we feed the replay buffer using a heuristic policy for five episodes before initiating training. The heuristic policy, part of this methodology, incorporates 3 key strategies that facilitate operations and improve efficiency.

Firstly, navigation begins with offloading to cloud execution. This initial step serves to ensure that the process starts with robust information about its surroundings, leveraging the power of cloud computing for optimal results.

The subsequent strategy involves the execution of cloud operations only once a defined staleness threshold is surpassed. This threshold serves as a benchmark to ensure that only relevant and fresh data is processed in the cloud. Staleness in this context refers to the age or outdatedness of the data. By setting this threshold, we can minimize the processing of old or irrelevant data, thereby improving overall efficiency and reducing computational wastage.

Finally, an essential part of the policy is to discourage cloud offloading when the distance to the network node is significant. This is based on the understanding that network accessibility and performance often decrease with an increase in distance. By minimizing cloud offloading under these conditions, we can ensure more consistent network access, reduce latency, prevent data loss and maintain high performance, which in turn contributes to the effectiveness and reliability of the overall operations.

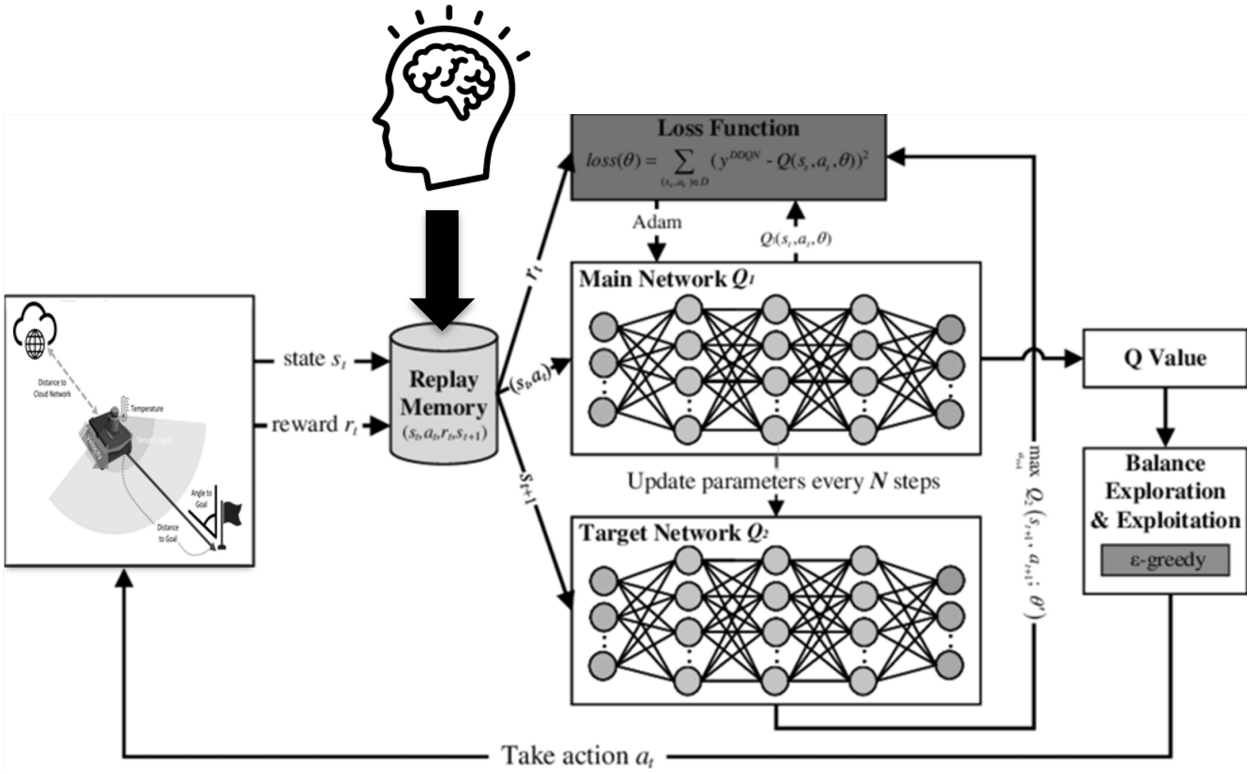


Figure 3.5: Double-DQN with Heuristic Policy in Buffer

### 3.4.7 Comparison with Baseline Policies

For a comprehensive understanding, we contrast the RL-based policy with the following established policies:

1) Robot-only Computation Policy $\pi_{\text{all-robot computation}}$: This policy opts for offload = 2 at every time step to interact with the robot model and optionally utilizes past robot predictions at offload = 0 as necessary.

2) All-cloud Offloading Policy $\pi_{\text{all-cloud offload}}$: This policy invariably selects offload = 1.

3) Oracle-based Mapping Policy $\pi_{\text{oracle scan}}$: This policy is based on the assumption that the robot is aware of the range to the nearest obstacle ahead and only scans a bit beyond this known range.

Policies 2 and 3 might appear overly simplistic. However, it is notable that these strategies are extensively used in standard robotics (where all computations are performed by the robot) and

standard cloud robotics (where all data is offloaded with holds to manage bandwidth requirements). Also, many of the DRL usage for robotics solely rely on one or the other.

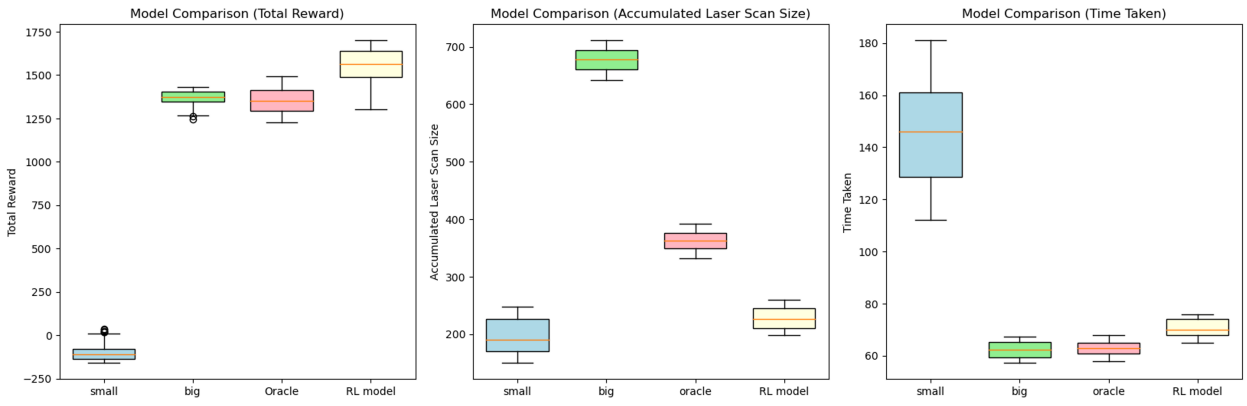


Figure 3.6: Comparison with Benchmark Policies over mixture of environment episodes

## 3.5 Evaluation and Discussion

In this section, we discuss the effectiveness of our reinforcement learning (RL) model in terms of trajectory performance, navigation quality, and computational efficiency.

From the depicted trajectories in Figure 3.7, it can be seen that our RL model can navigate adeptly through the environment by strategically offloading computations during critical time steps. Contrarily, a policy that relies solely on local computations and hence is constrained by the limited range of sensor readings can result in the robot being stuck due to encountering a dead-end. By dynamically adjusting the range of data considered, our RL model manages to avoid this issue, demonstrating its ability to adapt to environmental changes.

In terms of time efficiency, our RL model demonstrates commendable performance. Despite utilizing less computational resources, the RL model achieves a time-to-goal that is comparable to both the Cloud-only policy and the Oracle policy. The Oracle policy, in this context, signifies an ideal scenario where the robot knows the exact location of obstacles and can consequently compute the exact range to each obstacle.

The computational efficiency of the RL model is perhaps its most notable strength. As depicted in Figure 3.6, our RL model requires only a third of the computational resources that the Cloud-only policy uses. This assumes that cloud computation is possible even at far-off network edges, a condition that may not hold in real-world settings. Further, the RL model requires approximately half the computational resources of the Oracle policy. This offers significant improvements in terms of both energy usage and system sustainability, presenting a robust solution for autonomous navigation tasks.

## 3.6 Real-World Experimentation

### 3.6.1 Implementation to Real World Settings

A critical question that guided our research was: "How can we ensure that a policy learned in a simulated environment translates effectively into real-world applications?" To answer this, we tested our model on a real Jackal robot equipped with an Intel i5 processor and 4GB of RAM. As depicted in Figure 3.10, the robot performed cloud offloading whenever the state representations met the predefined conditions. This led to efficient and sustainable navigation in a real-world environment, validating our model's capability to generalize beyond simulated conditions.

To further augment the comprehensibility of our robot's decision-making process—a platform similar to the principles of Explainable AI—we integrated an Intel depth camera. This enabled the robot to visually convey its policy and decision-making process with respect to the robot's state to observers, enhancing the user's understanding of the trained model.

There are several promising directions for future research to extend the capabilities of our model. One such direction involves fusing depth camera sensor data with the robot's current state representations, which could potentially improve its navigation effectiveness. Visual data can also be incorporated to execute better scans when pre-defined action or target is identified.

Another potential future work, is to implement a reinforcement learning model that incorporates human feedback. Although our model has proven effective in controlled real-world experiments, emergency response or battlefield conditions are full of uncertainty. Thus, we aim to create

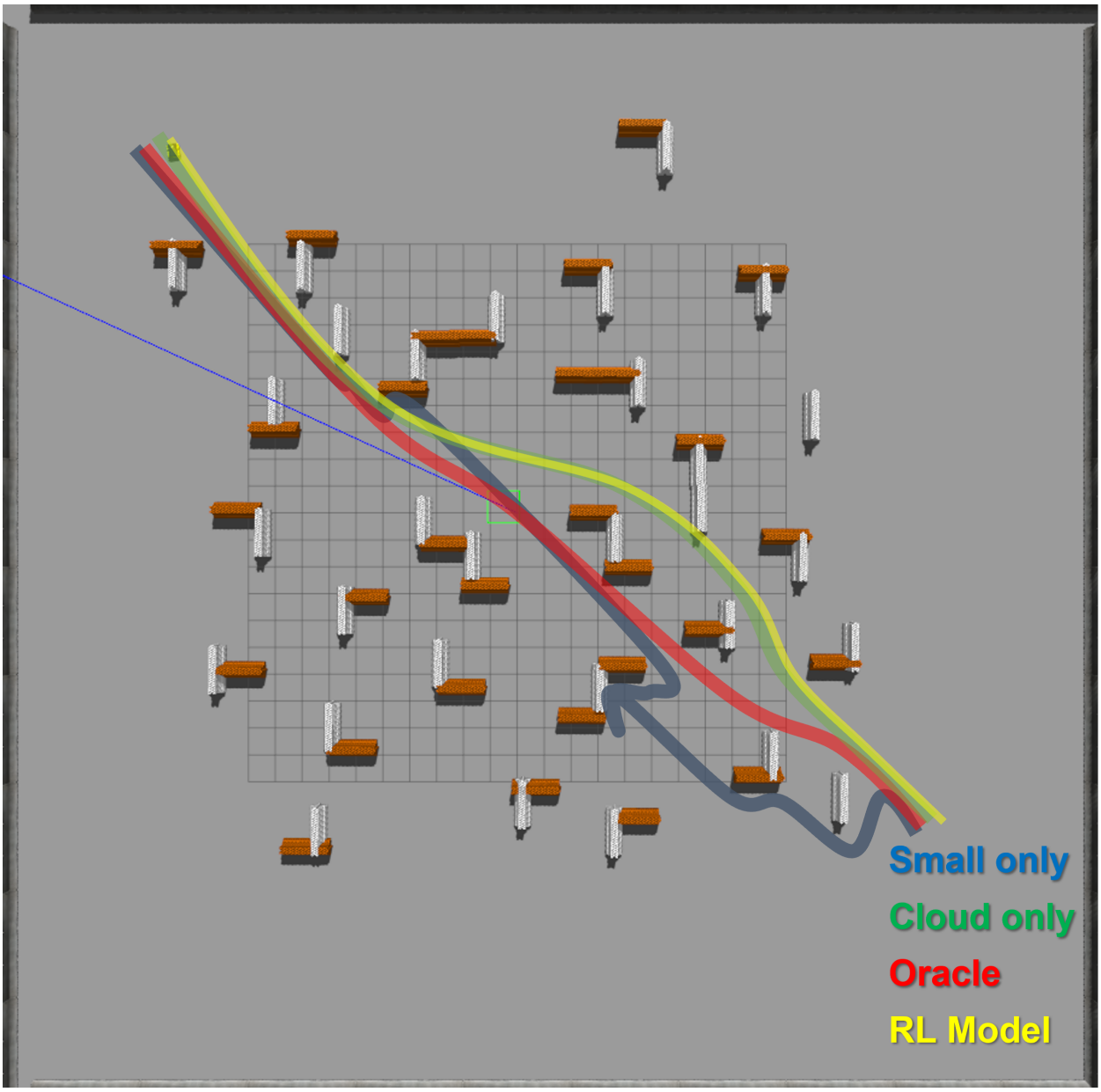


Figure 3.7: Comparison with Benchmark Policies on Trajectories

a learning model that not only operates according to learned policies but can also dynamically adjust its actions based on human feedback. This would allow a human operator to correct the robot's actions manually when necessary, and the model could update its policy based on this feedback.

To create a more immersive experience for the human operator, we envision transmitting the robot's visual imagery into a virtual reality (VR) headset. The operator could then provide feed-

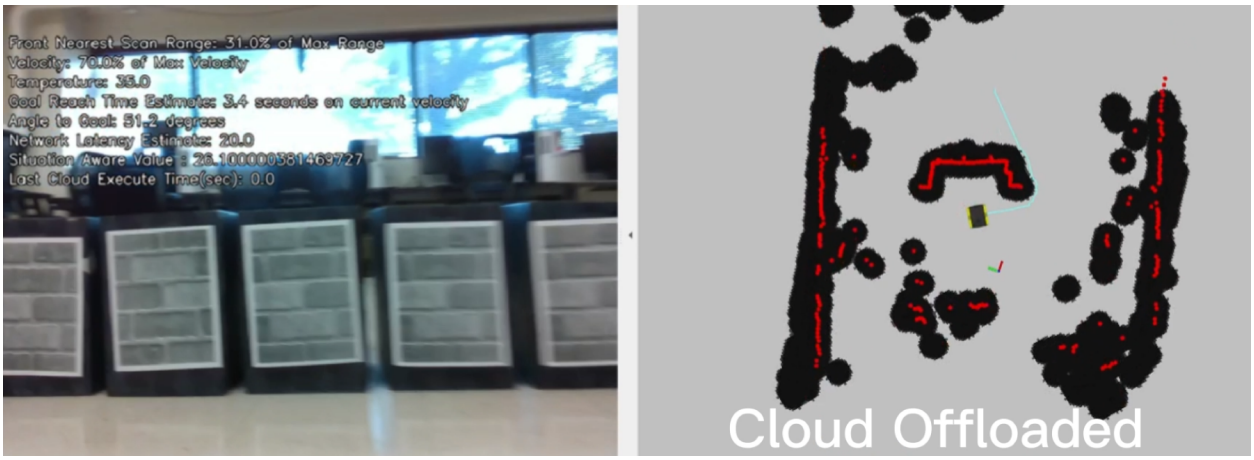


Figure 3.8: Real-world Experiment with Explainable AI

back by manipulating the robot's actions using VR controllers, thereby enhancing their control and understanding of the robot's actions.

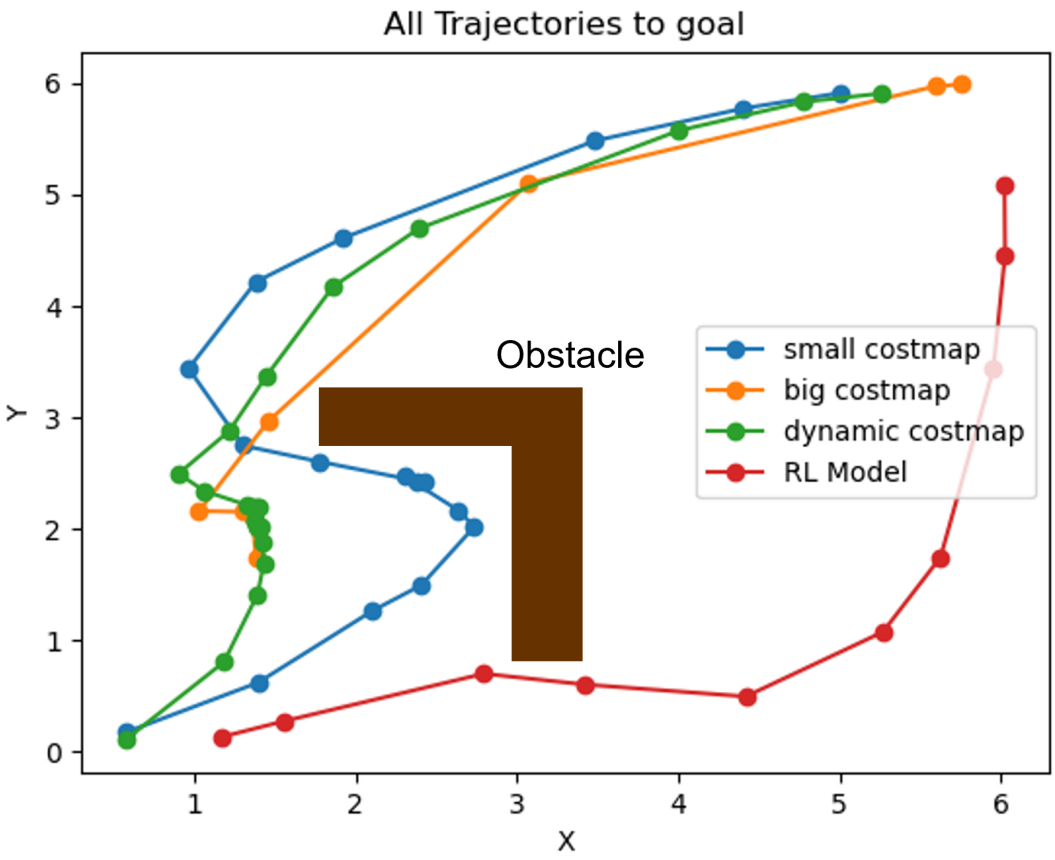


Figure 3.9: Real-world Experiment Trajectories


Figure 3.10: Real-world Experiment Jackal

# 4. SUMMARY AND CONCLUSIONS

## 4.1 Summary

This research has investigated the application of reinforcement learning, specifically Deep Q-learning with a replay buffer, in the context of an offloading decision regarding a mapless navigation problem. The objective was to develop a system capable of efficient navigation with limited computational and energy resources. Our approach provides a potential solution for utilizing fog robotics, where sustainability and efficiency are necessary.

Our approach utilized the Robot Operating System (ROS) as an interface for interaction with the navigation environment. This involved the creation of a customized environment where the training of the RL model was performed, while also running necessary packages. This approach fills an important gap since most reinforcement learning research has primarily focused on the manipulation of robot controllers, assuming perfect robot localization and not considering an environment where mapping, navigation stack or other prerequisite modules are necessary.

## 4.2 Comparison of Reinforcement Learning Algorithms

The Deep Q-Network (DQN) agent received sensory inputs and was tasked with deciding whether to perform local computations or offload data to a cloud server for computation. The decision-making process was guided by a reinforcement learning policy, which optimized for a balance between computational sustainability, system efficiency, and accurate situational perception.

We addressed the challenge of overestimation of Q-values, a common issue in traditional DQN algorithms, by implementing the Double DQN variant. This algorithm creates a separate neural network for action selection and action evaluation, which helped prevent overestimations and led to more optimal policies.

### 4.3 Designing an Effective Reward Function

A pivotal component in the successful implementation of our Reinforcement Learning (RL) system was the design of an effective reward function. The reward function was thoughtfully structured to capture the main objectives of our proposed system: accurate situational perception, minimizing onboard computation, and limiting cloud query costs. By embedding these objectives into the reward function, the RL agent was inspired to act in a manner that adheres to these goals, which as a result, aligns the learned policy with our intended outcomes. Therefore, our approach has demonstrated a practical methodology for forming a reward function to capture the distinctive needs of a specific RL application while minimizing the reality gap.

### 4.4 Translation of Simulated Policies to Real-World Applications

A primary concern of this research was ensuring the applicability of learned policies from simulated environments in real-world applications. We conducted extensive tests in both simulation and real-world scenarios, especially using a physical Jackal robot equipped with an Intel depth camera in actual domain settings. The real-world tests validated the simulation results, demonstrating effective navigation while successfully reducing the computation and cloud query costs. This validation provided a clear answer to our third research question, proving that RL policies learned in a simulated environment can indeed be translated effectively into real-world applications.

In comparison to other policies, such as cloud-only and Oracle, the RL model was able to reach goals with significantly less computation while maintaining similar performance in terms of time to reach the goal.

### 4.5 Explainable AI and Future Work

The implementation of an Explainable AI initiative, through the integration of an Intel depth camera, provided a deeper understanding of the robot's decision-making process. This could enhance users' understanding of our model, a crucial factor, especially in fields where the understanding of AI decisions is critical.

Looking ahead, there are several potential directions for future work. The integration of depth

camera sensor data with the robot's current state representations could enhance the robot's navigation capabilities. Furthermore, in line with the unpredictability of real-world scenarios, we plan to explore reinforcement learning models incorporating human feedback. This allows for a more dynamic and responsive system, which can update its policy based on manual corrections from a human operator.

Additionally, we aim to enhance the immersive experience for the human operator by using virtual reality technology. Transmitting the robot's visual imagery to a VR headset could enable the operator to provide more intuitive and real-time feedback, further improving the efficiency and effectiveness of our system.

In conclusion, this research has made a significant step towards efficient and sustainable mapless navigation through reinforcement learning. The positive results in both simulation and real-world experiments underline the potential of RL for developing sustainable robotic systems. As we continue to advance our model, we look forward to the ultimate goal: creating intelligent, efficient, and sustainable robotic systems for real-world applications.

# REFERENCES


[1] K. Goldberg and B. Kehoe, "Cloud robotics and automation: A survey of related work," Technical Report UCB/EECS-2013-5, EECS Department, University of California, Berkeley, 2013.

[2] J. Wan, S. Tang, H. Yan, D. Li, S. Wang, and A. V. Vasilakos, "Cloud robotics: Current status and open issues," *IEEE Access*, vol. 4, pp. 2797–2807, 2016.

[3] S. Chinchali, A. Sharma, J. Harrison, A. Elhafsi, D. Kang, E. Pergament, E. Cidon, S. Katti, and M. Pavone, "Network offloading policies for cloud robotics: a learning-based approach," 2019.

[4] M. Penmetcha and B.-C. Min, "A deep reinforcement learning-based dynamic computational offloading method for cloud robotics," *IEEE Access*, vol. 9, pp. 60265–60279, 2021.

[5] A. J. Ben Ali, Z. S. Hashemifar, and K. Dantu, "Edge-slam: Edge-assisted visual simultaneous localization and mapping," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, MobiSys '20, (New York, NY, USA), p. 325–337, Association for Computing Machinery, 2020.

[6] J. Laconte, A. Kasmi, R. Aufrère, M. Vaidis, and R. Chapuis, "A survey of localization methods for autonomous vehicles in highway scenarios," *Sensors*, vol. 22, no. 1, 2022.

[7] M. Montemerlo, *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, July 2003.

[8] J. A. Placed, J. Strader, H. Carrillo, N. Atanasov, V. Indelman, L. Carlone, and J. A. Castellanos, "A survey on active simultaneous localization and mapping: State of the art and new frontiers," 2023.

[9] K. Karur, N. Sharma, C. Dharmatti, and J. E. Siegel, "A survey of path planning algorithms for mobile robots," *Vehicles*, vol. 3, no. 3, pp. 448–468, 2021.

[10] N. Tahir and R. Parasuraman, "Analog twin framework for human and ai supervisory control and teleoperation of robots," 2022.

[11] A. J. B. Ali, M. Kouroshli, S. Semenova, Z. S. Hashemifar, S. Y. Ko, and K. Dantu, "Edge-slam: Edge-assisted visual simultaneous localization and mapping," vol. 22, no. 1, 2022.

[12] Y. Shi, K. Yang, T. Jiang, J. Zhang, and K. B. Letaief, "Communication-efficient edge ai: Algorithms and systems," 2020.

[13] X. Zhu, Y. Luo, A. Liu, M. Z. A. Bhuiyan, and S. Zhang, "Multiagent deep reinforcement learning for vehicular computation offloading in iot," *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9763–9773, 2021.

[14] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer science & business media, 2013.

[15] S. Yu and Z. Jiang, "Design of the navigation system through the fusion of imu and wheeled encoders," *Computer Communications*, vol. 160, pp. 730–737, 2020.

[16] A. Rahman, J. Jin, A. Cricenti, A. Rahman, and D. Yuan, "A cloud robotics framework of optimal task offloading for smart city applications," in *IEEE Global Communications Conference (GLOBECOM)*, 2016.

[17] G. Mohanarajah, V. Usenko, M. Singh, R. D'Andrea, and M. Waibel, "Cloud-based collaborative 3d mapping in real-time with low-cost robots," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 499–506, 2015.

[18] J. L. Jingyu Xiong, Hongzhi Guo, "Task offloading in uav-aided edge computing: Bit allocation and trajectory optimization."

[19] J. Judson, "Us army prepping robotic combat vehicles for big test with soldiers in 2022," 2021.

[20] D. Spatharakis, M. Avgeris, N. Athanasopoulos, D. Dechouniotis, and S. Papavassiliou, "Resource-aware estimation and control for edge robotics: A set-based approach," *IEEE Internet of Things Journal*, vol. 10, no. 3, pp. 2003–2020, 2023.

[21] M. Afrin, J. Jin, A. Rahman, A. Rahman, J. Wan, and E. Hossain, "Resource allocation and service provisioning in multi-agent cloud robotics: A comprehensive survey," *IEEE Communications Surveys  Tutorials*, vol. 23, no. 2, pp. 842–870, 2021.

[22] B. B and V. Rathinasabapathy, "Environmental temperature influence on wireless sensor networks," 05 2020.

[23] M. G. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *CoRR*, vol. abs/1908.00080, 2019.

[24] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4005–4018, 2019.