

**Final Year Project Report**  
**Bachelor of Engineering**

**A Solution for Last Mile Transport: QUIC Edge  
Computing Networks for ROS-Based  
Autonomous Vehicle Formation**


Student:

Supervisor:

2023-24

# Coursework Declaration and Feedback Form

*The Student should complete and sign this part*

Student Name:	Student GUID:
Course Code: UESTC4006P	Course Name: INDIVIDUAL PROJECT 4
Name of Supervisor:	
Title of Project: A Solution for Last Mile Transport: QUIC Edge Computing Networks for ROS-Based Autonomous Vehicle Formation	
<b>Declaration of Originality and Submission Information</b>	
<i>I affirm that this submission is all my own work in accordance with the University of Glasgow Regulations and the School of Engineering requirements</i> Signed (Student):	 UESTC4006P
Date of Submission: 4/25/2024	
<i>Feedback from Lecturer to Student – to be completed by Lecturer or Demonstrator</i>	
Grade Awarded: Feedback (as appropriate to the coursework which was assessed):	
Lecturer/Demonstrator:	Date returned to the Teaching Office:

## Abstract

In the rapidly evolving logistics industry, the challenge of last-mile delivery remains a critical bottleneck, particularly in urban environments where navigation and efficient route planning are complex due to high traffic and numerous obstacles. This report introduces a specialized approach designed to enhance map construction for autonomous vehicle formations, which is crucial for optimizing last-mile delivery strategies effectively. We propose a novel edge-computing system that integrates the QUIC protocol within ROS-topic communications to accelerate and stabilize network interactions, improved leader-follower formation augmented by APF, and advanced path-planning algorithms implemented using Turtlebot3 Burgers. This integrated approach is tailored to address the challenges of coordinating multiple vehicles SLAM in obstacle-rich urban environments. Our validation, conducted through simulations in Gazebo and Rviz environments, demonstrates significant improvements in network efficiency, data handling capabilities, and navigational accuracy. The results show an enhancement in the SLAM performance of autonomous vehicles, promising a transformative impact on the efficiency and reliability of urban last-mile delivery.

**Keywords:** Last-mile delivery, ROS (Robot Operating System), Multi-vehicle formation, QUIC (Quick UDP Internet Connections), Path-planning algorithms, Leader-follower model, SLAM (Simultaneous Localization and Mapping), Edge computing, APF (Artificial Potential Field)

## **Acknowledgements**

I express my sincere gratitude to the attendees of my wedding ceremony, who were present to witness the union between myself and my spouse, including BLOSPRIT\_KUKAS, PRINTBALL and JOMOR, individuals with whom I share a deep and meaningful friendship. Special recognition would also given to my supervisor, and my academic advisor, Peng Cai, who never ceased reminding me to study and prepare for taking make-up examinations. I am immensely grateful that I couldn't even to say a word to my parents for urging me to abandon my super-soft bed at home and proceed to school's dormitory in order to complete this paper, thereby averting any potential delay in my graduation. Additionally, I would like to acknowledge my owner, a Labrador named Lucky, who took possession of my comfortable bed and received all the love from my family. The unprecedented piano music and blessings of my spouse have made a significant difference to my successful completion of my course. It is with deep gratitude that I compose this final essay of my undergraduate years.

# Contents

Abstract.....	3
Acknowledgements.....	4
1 Introduction.....	9
1.1 Solutions' evolution of last-mile delivery problem .....	9
1.2 Core technological challenges in autonomous delivery.....	10
1.3 Main innovations and contributions.....	11
1.4 Organization of remaining sections .....	12
1.5 Summary of the introduction .....	12
2 Related works.....	13
2.1.1 Comparative analysis of working principles over QUIC and TCP.....	13
2.2 Real-time obstacle avoidance for autonomous vehicles .....	16
2.2.1 Grid-based global path-planning algorithms .....	16
2.2.2 Local obstacle avoidance approaches .....	18
2.3 Multi-vehicle collaborative formation system .....	20
2.3.1 Graph theory and consensus protocol in collaborative formation .....	20
2.3.1 Formation control algorithms and corresponding math model.....	22
2.4 Summary of the related works .....	23
3 QUIC-based Multi-Vehicle Formation SLAM Framework .....	24
3.1 Overview and analysis of key challenges in autonomous systems.....	24
3.2 Enhanced vehicles cluster communication system with edge networks.....	25
3.2.1 Vehicle clusters' communication system with edge networks .....	25
3.2.2 Operational mechanics in autonomous vehicle formations .....	26
3.3 Hybrid communication system with ROS and QUIC.....	27
3.3.1 Configuration of virtual machines and basic file hierarchy of ROS.....	27
3.3.2 Topic communication structure of ROS .....	28
3.3.3 NAT mode communication between multiple virtual machines .....	29

3.3.4	Implementation of hybrid communication system .....	30
3.4	Integration of leader-follower technique with APF algorithm .....	32
3.4.1	Robot's specifications and its simulation on Gazebo and Rviz.....	33
3.4.2	Implementation of integrated leader-follower technique.....	34
3.5	ROS SLAM system with Lazy Theta* and TEB algorithm .....	37
3.5.1	Gmapping algorithm for SLAM and sensor coordinate's conversion .....	37
3.5.2	Global and Local Cost Maps with Lazy Theta* and TEB for Navigation.....	39
4	Experimental results and evaluation .....	41
4.1	Main configurations and parameters for experiment setup.....	41
4.2	Experimental results and evaluation on SLAM framework .....	44
5	Conclusions and future works.....	47
5.1	Conclusions.....	47
5.2	Suggestions for future works .....	47
	References.....	49
A	Appendices.....	52
B	University's Plagiarism Statement.....	65

## List of Figures

Figure 1. Comparison of TCP/IP and QUIC protocols .....	14
Figure 2. TCP three handshakes .....	15
Figure 3. 1-RTT and 0-RTT for TLS 1.3 .....	15
Figure 4. TLS 1.0 four handshakes .....	16
Figure 5. Eight-neighbour method .....	18
Figure 6. Traditional A* compared with Theta* .....	18
Figure 7. Communication graph topology and matrix representation .....	21
Figure 8. Framework of autonomous vehicle clusters collaborative networking .....	25
Figure 9. Basic file hierarchy of ROS .....	28
Figure 10. Simple ROS topic communication for two nodes .....	29
Figure 11. NAT-Network topology diagram .....	30
Figure 12. QUIC-based host and client communication on ROS .....	31
Figure 13. Main components of TurtleBot3 Burger .....	33
Figure 14. Graphic description of integrated leader-follower technique .....	35
Figure 15. Framework of ROS move_base package .....	39
Figure 16. Default recovery behaviours of ROS move_base package .....	40
Figure 17. Autonomous vehicles employing APF for formation and initiating Gmapping ....	44
Figure 18. Generating 2D Maps via Navigation Commands in Rviz .....	44
Figure 19. Comparison of overall route planning for A* and Lazy Theta* .....	45

## List of Tables

TABLE I: TURTLEBOT3 BURGER HARDWARE SPECIFICATION .....	34
TABLE II: ARTIFICIAL POTENTIAL FIELD CONSTANTS .....	41
TABLE III: KEY PARAMETERS FOR GMAPPING ALGORITHM ON ROS .....	42
TABLE IV: CRUCIAL PARAMETERS FOR GLOBAL_COSTMAP .....	43
TABLE V: CRUCIAL PARAMETERS FOR LOCAL_COSTMAP .....	43
TABLE VI: ESSENTIAL PARAMETERS FOR TEB_LOCAL_PLANNER .....	43
TABLE VI: NETWORK LATENCY BETWEEN QUIC AND ROSTCP .....	44



# 1 Introduction

This section analyses the logistics sector, emphasizing challenges in last-mile delivery. It highlights the role of autonomous vehicle technologies in enhancing delivery efficiency and delves into technical issues like integrating QUIC (Quick UDP Internet Connections) within ROS (Robot Operating System), developing advanced path-planning algorithms, and coordinating autonomous fleets. The report proposes strategies to optimize delivery through improved map construction and advanced navigation techniques in vehicle formations.

## 1.1 Solutions' evolution of last-mile delivery problem

Last-mile delivery, the final step of the delivery process from a distribution centre to the end user, presents enduring challenges within the logistics industry. The after-COVID-19 increase in e-commerce highlights a global escalation in the demand for last-mile parcel delivery services. From a conceptual standpoint, last-mile delivery refers to the preparation of parcels at the distribution centre and subsequent transportation to the customer's address [1]. It is noteworthy that the delivery cost accounts for a minimum of 28% of the overall transportation expenses within the supply chain, with almost half of these costs attributed to last-mile delivery. As a result, approaches to reduce costs and enhance efficiency have attracted considerable attention in both the academic and industrial domains [2].

Rapid advancements in technology, especially in autonomous vehicles, offer promising solutions for addressing persistent logistical challenges in last-mile delivery. By strategically integrating autonomous vehicles into delivery systems for exploratory mapping purposes, a pioneering approach emerges to eliminate the need for manual exploration and mapping [3]. This commitment highlights the substantial benefit of reduced labour expenses, thereby emphasising the reduction in manpower costs as a primary advantage. The employment of autonomous fleets to generate up-to-date and accurate digital maps directly tackles a critical challenge in last-mile delivery, shedding light on the economic implications of reducing manual labour in delivery systems through autonomous technologies.

The development of sensor technologies, such as LIDAR (Light Detection and Ranging), radar, and cameras, which have seen notable breakthroughs, plays a crucial role in enhancing the perception, situational awareness, and navigational capabilities of autonomous vehicles. These advancements are crucial for vehicles to accurately understand and navigate intricate urban settings, guaranteeing the safety and effectiveness of self-driving delivery operations.

Levinson et al. conducted a conclusive study that provides an analysis of LIDAR's impact, highlighting its crucial role in enhancing autonomous vehicle navigation [4].

Continuous progress in networking and network topology plays an essential role in facilitating the sustained and uninterrupted connectivity of autonomous vehicles, thereby supporting the intricate data flows and timely decision-making processes inherent in autonomous vehicle operations. Mesh networks, known for their exceptional robustness and adaptability, represent a notable innovation in network topology. In such networks, autonomous systems possess the capability to dynamically readjust communication paths. Willke et al. [5] introduce a notable idea that explores dynamic routing protocols and demonstrates how mesh networks maintain trustworthy communication and integrity in inter-vehicle communication. This lets a mobile node, like a robot or vehicle, talk directly with peers using wireless radio, lowering the risks of signal loss and interruption.

Additionally, the incorporation of 5G technology into vehicular networks signifies significant progress in the speed at which data is transmitted and the reduction of latency. Added to that, the introduction of the QUIC protocol [6] enhances internet communication by placing the priority on users' security and privacy, as well as providing greater data transmission efficiency over networks like mesh networks. Zhao et al. conducted a study that examines the potential synergies among 5G technology, QUIC protocol, and massive machine-type communication, emphasising how this new approach can enhance vehicular communication and optimise network efficiency by enhancing the network communication overhead of the base station by about 10% and reducing the computing burden of the base station by an average 40% in CPU usage [6].

## **1.2 Core technological challenges in autonomous delivery**

Within the domain of autonomous vehicle delivery, notably targeting last-mile deliveries, the advent of technological improvements has shown great potential. However, the implementation of completely self-governing delivery systems encounters considerable technical obstacles.

The seamless integration of the QUIC protocol into ROS-enabled autonomous vehicular formations stands as a prominent challenge in the realm of vehicular communications. The incorporation of QUIC, with its rapid transmission capabilities and swift connection establishment, into ROS architectures disrupts the established communication framework of ROS, which is typically based on a publisher-subscriber model designed for facilitating inter-

process communication within the autonomous vehicle ecosystem [7]. Introducing QUIC necessitates the revision of ROS communication paradigms, as QUIC's stream multiplexing and transport layer features may not align directly with the current ROS message-passing infrastructure. This necessitates an innovative technique to harmonize their disparities while upholding the integrity and the modular design principles that ROS follows.

Implementation of autonomous vehicles delivery in urban area is made more challenging by the unpredictable pedestrian and the changing nature of city traffic. To address these issues, it is necessary to develop advanced path-planning algorithms that allow vehicles to navigate intricate urban environments [8]. These algorithms need to not only understand the complex spatial dynamics of urban areas but also show exceptional adaptability to their constantly changing nature.

Furthermore, the coordination of several vehicles in formations becomes another crucial factor to consider, particularly for tasks that include mapping large areas and conducting search operations. Efficient synchronization among vehicles, involving complex interactions between leaders and followers, is crucial for preserving the structural stability of these formations. This coordination necessitates the ability to make immediate modifications in order to accommodate new information or environments, hence guaranteeing the safety and effectiveness of the formation [9]. In this kind of situation, the significance of strong coordination within the formation is of paramount significance, and it is essential for vehicles avoiding collisions and optimizing operating efficiency and coverage area.

### **1.3 Main innovations and contributions**

We introduce a novel framework for autonomous multi-vehicle formation SLAM (Simultaneous Localization and Mapping) leveraging the capabilities of the QUIC protocol in this report. This framework is designed to address the challenges associated with map construction reliability and efficiency, which are critical during delivery processes and particularly for solving the last-mile delivery problem. This report makes four main contributions:

1. Propose a novel QUIC-based multi-vehicle formation SLAM framework that significantly improves the efficiency of map construction in delivery environments.
2. Devise a hybrid communication system that combines edge network and ROS-based topic communication with the QUIC protocol.

3. Present a synergistic approach using the leader-follower method coupled with the APF on ROS-based simulation, equipping the vehicles with the ability to navigate complex environments while maintaining formation integrity.
4. Construct a revised SLAM method for single ROS-based vehicle and fine-tune related parameters to ensure robust localization and mapping accuracy.

We conduct a comprehensive performance assessment in the SLAM of a hybrid communication system that combines edge network and ROS-based topic communication with the QUIC protocol, an improved leader-follower framework and a revised SLAM system. Our analysis demonstrates the advantages of this integration in terms of real-time data exchange and efficiency in map construction for multi-vehicle formation, which are essential for effective autonomous robots' collaboration. The introduced framework and methodologies not only contribute to the field of autonomous delivery systems but also have the potential to be extended to other domains requiring intelligent robots' collaboration tasks.

## **1.4 Organization of remaining sections**

The remaining sections of this report are organized as follows: Section II presents the related works on autonomous vehicle coordination system and path-planning techniques. Section III introduces the QUIC-based framework of the multi-vehicle SLAM system. In Section IV, we present our experimental results. We summarize our work in Section V.

## **1.5 Summary of the introduction**

In summary, we provide a focused analysis of the logistics sector particularly emphasizing the challenges in last-mile delivery and discuss how autonomous vehicle technologies are pivotal in addressing these challenges by enhancing delivery efficiency. We delve into core technical obstacles that hinder the full realization of autonomous delivery systems, including the integration of advanced communication protocols within ROS-enabled vehicular frameworks, the crucial need for effective coordination among vehicles in autonomous fleets, and the sophisticated path-planning algorithms for navigating dynamic urban environments. Our report investigates the integration of the QUIC protocol and edge network with ROS to improve SLAM efficiency in autonomous vehicle formations. These strategies encompass the development of a multi-vehicle formation SLAM framework, a hybrid communication system, an advanced formation technique, and revised SLAM system, potentially transforming industry standards and practices.

## 2 Related works

In this section, we delve into the challenges and advancements in three pivotal domains: IoT (Internet of Things) network communication protocols, autonomous vehicle navigation, and multi-vehicle collaboration systems. we discuss the implementation and optimization of QUIC, highlighting its role in enhancing the efficiency and reliability of IoT networks. We then explore path-planning algorithms such as Lazy Theta\* and TEB (Timed Elastic Band), which are crucial for real-time obstacle avoidance in autonomous vehicles. Finally, we examine the application of graph theory and consensus methods in multi-vehicle collaborative formation systems, emphasizing its importance in formation controlling and resilient multi-vehicle formations.

### 2.1 Communication protocol deployment of the Internet of Things

The IoT facilitates a wide range of applications by linking a multitude of sensors and actuators to the internet. An essential component of this integration is the deployment of transport layer protocols, which guarantee efficient data exchange across end-to-end applications. Two commonly used transport layer protocols in IoT networks are TCP (Transport Layer Protocol) and UDP (User Datagram Protocol). TCP is renowned for its reliable end-to-end connection and congestion control, preventing buffer overflow, but at the cost of higher overhead and resource use [10]. Conversely, UDP is distinguished by its minimal overhead but lacks inherent mechanisms for reliability and congestion control [11].

#### 2.1.1 Comparative analysis of working principles over QUIC and TCP

In the industrial internet sector, TCP is favoured for the majority of applications due to its enhanced reliability. However, it can result in unexpectedly long latencies, potentially reaching up to 10 times longer than the required latency in scenarios described in [12].

As shown in Fig. 1, the classic TCP/IP stack uses the standard HTTPS stack, which consists of HTTP, SPDY, TLS (Transport Layer Security), and TCP layers. Emerging protocols like QUIC, which ingeniously combine cryptographic protocols with transport process, notably diminish round-trip times (RTTs). This solves the latency problems that TCP/IP has and changes how protocols work in IoT networks.

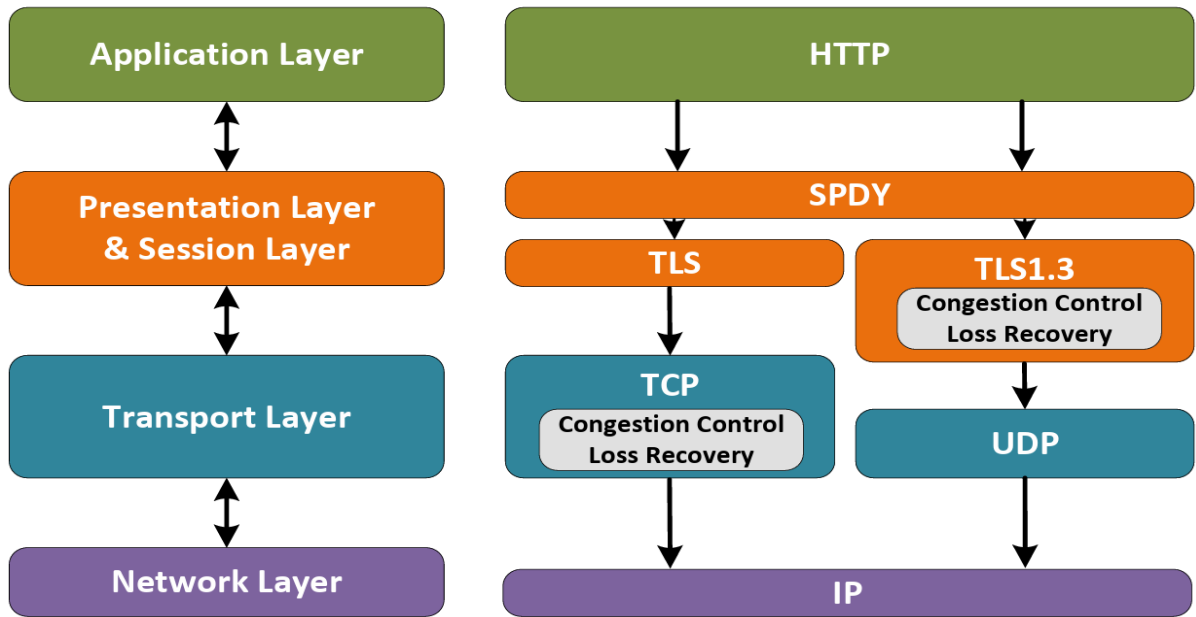


Figure 1. Comparison illustration of TCP/IP and QUIC protocols

Fig.2 illustrates that in the standard HTTPS model, TCP necessitates three handshakes to establish a connection, costing two RTTs. Upon building this TCP connection, to secure end-to-end data exchange over the transport layer, TLS 1.0 is required, involving additional four handshakes shown in Fig.4. In contrast, QUIC, which integrates HTTP, SPDY, TLS 1.3, and UDP, significantly diminishes those handshake latency. It establishes a TLS 1.3 secured connection in just one RTT, or even zero RTT as shown in Fig.3 if prior communication between endpoints has been established. QUIC allocates a reliable stream for cryptographic handshakes, thereby enhancing handshake efficiency significantly.

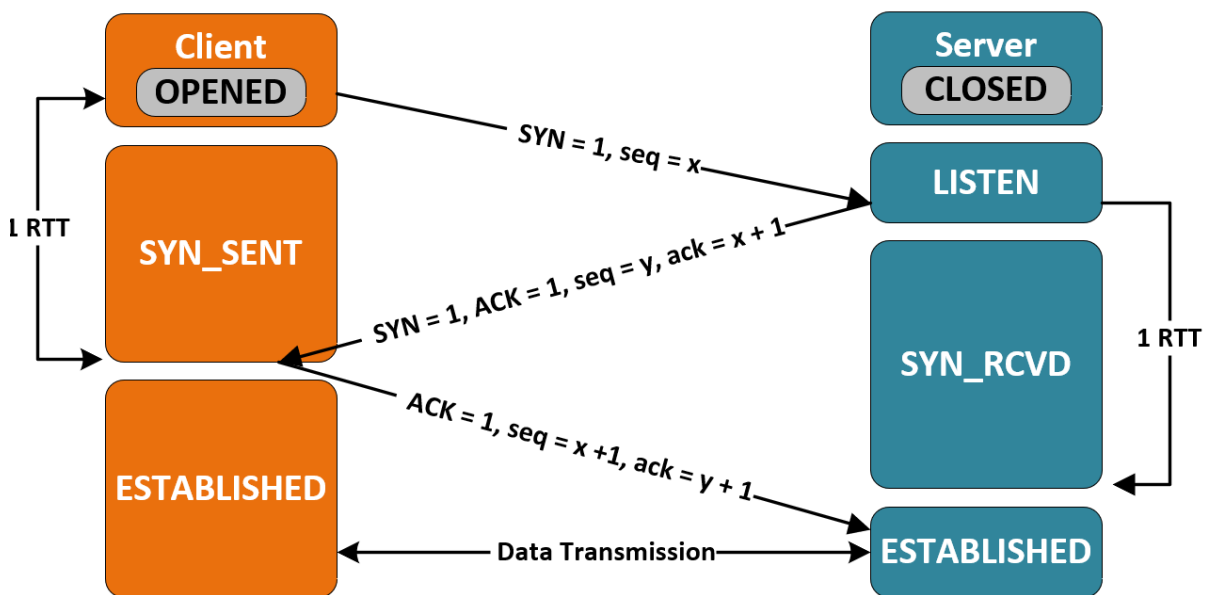


Figure 2. TCP three handshakes

QUIC efficiently arranges data within connections into separate streams, effectively avoiding delays caused by head-of-line blocking. When there is packet loss, only the streams that are impacted are paused while waiting for the lost packets to be resent. In addition, QUIC reduces latency by implementing effective loss detection techniques, which include early retransmits and tail loss probes [13].

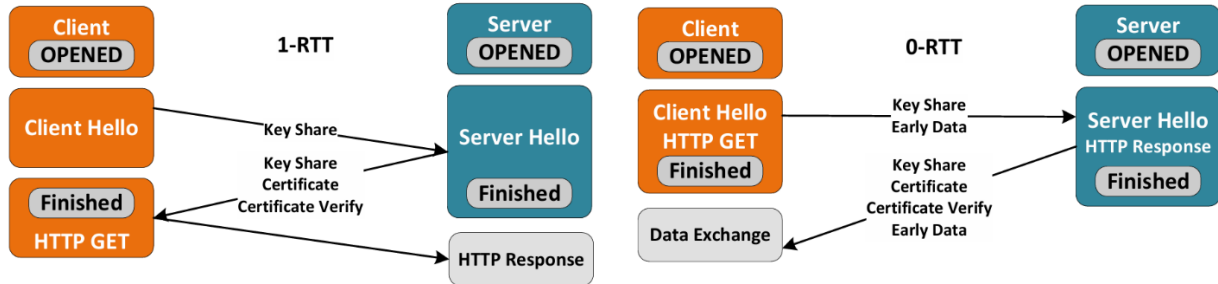


Figure 3. 1-RTT and 0-RTT for TLS 1.3

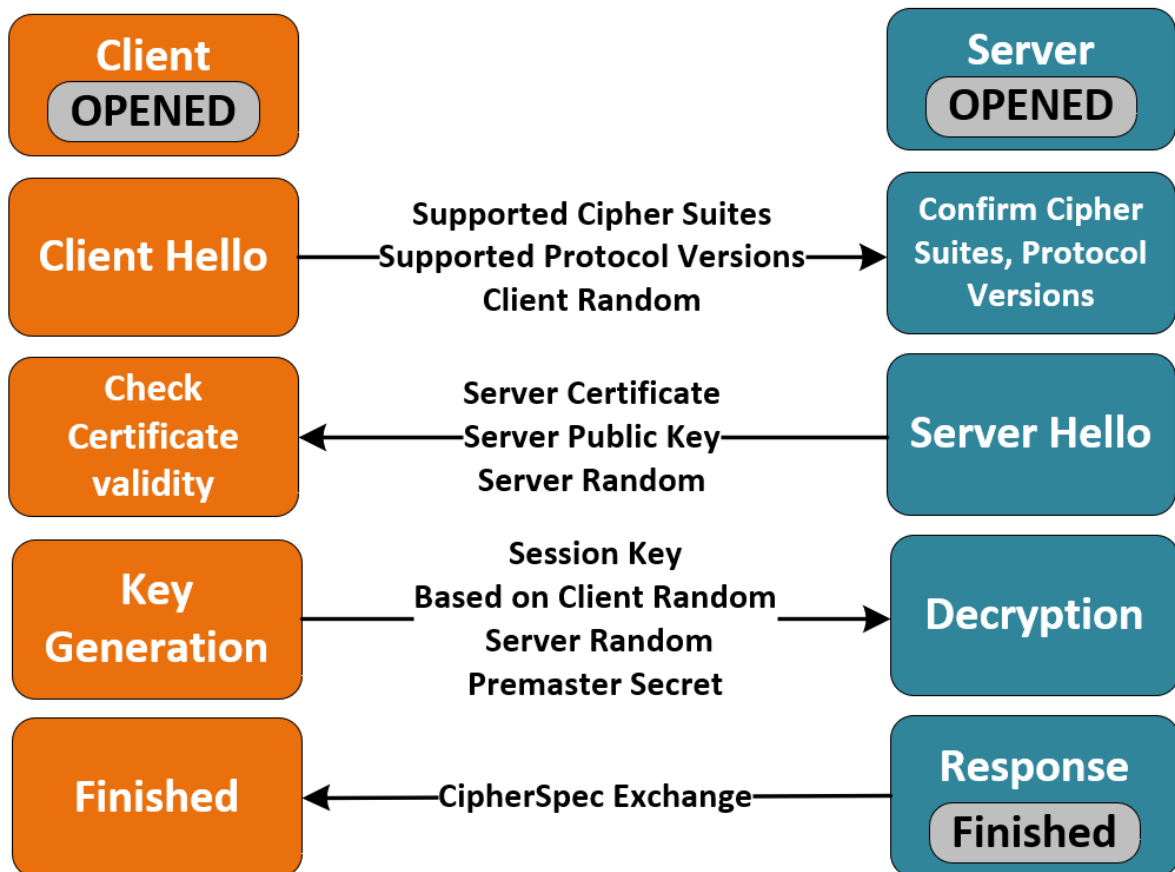


Figure 4. TLS 1.0 four handshakes

Deploying QUIC on IoT devices can be resource intensive. Eggert's study confirms that QUIC can operate on certain IoT devices, exemplified by its implementation on two 32-bit microprocessor boards using Quant and picots [15]. This setup required 63 kB of flash, 16 kB of heap memory, 4 kB of stack memory, and 0.9 J of energy per transaction. They underscore

the necessity for optimization to accommodate more resource-constrained 16-bit processors, thus broadening QUIC's applicability in the IoT domain.

Furthermore, the integration of QUIC with MQTT on Raspberry Pi 3 Model Bs, as demonstrated by Kumar and Dezfouli, required the development of inter-process communication APIs and modifications to data structures. This effort highlights the complexities involved in adapting QUIC for IoT environments. Specifically, it illustrates the need for additional software layers to effectively bridge the gap between application requirements and transport capabilities [16].

The burgeoning need for protocols that reconcile low latency with reliability in the IoT landscape is evident [17]. While traditional protocols like MQTT are prevalent in autonomous vehicle networks, employing TCP/IP for reliable exchanges, the implementation of QUIC, especially in resource constrained IoT devices, presents challenges. It necessitates the development of custom protocol stacks and meticulous resource management to ensure effective deployment.

## **2.2 Real-time obstacle avoidance for autonomous vehicles**

In autonomous vehicle navigation, the unpredictable nature of real-world driving environments necessitates algorithms capable of swiftly responding to unexpected obstacles. A critical element of real-time obstacle avoidance is the ability to process sensor data and react almost instantaneously. Real-time obstacle avoidance strategies must integrate both global and local planning techniques. Global path-planning methods often depend on static environmental representations, but adapting these to incorporate dynamic, local obstacle avoidance requires a more adaptive approach [18]. This integration enables autonomous vehicles to effectively navigate and respond to real-world traffic scenarios.

### **2.2.1 Grid-based global path-planning algorithms**

A foundational aspect of path-planning in autonomous navigation is the conversion of continuous environment data into a discrete grid-based representation of the 2D map, where the robot's current location and the target are defined. Typically, the algorithm would start evaluating the adjacent grid squares - the eight neighbours around the robot's present location depicted in Fig.5 - to determine the next step in the journey when navigating through this grid. One notable pioneer in this regard is the Dijkstra algorithm, which systematically explores all immediate neighbouring squares from the robot's current position, marking each until the destination is reached. Although Dijkstra's algorithm guarantees a path to the target



if one exists, it is criticized for its high computational demands and inability to identify the quickest route.

An alternative, Greedy Best-First Search (GBFS) employs a heuristic cost function:

$$h(n) = D(n, target) \quad (1)$$

where  $h(n)$  represents the estimated cost from the current node  $n$  to the target node, prioritizing nodes based on their proximity to the target. This heuristic cost is commonly calculated using the Euclidean distance metric, defined for two points,  $A(x_1, y_1)$  and  $B(x_2, y_2)$ , as follows:

$$D(A, B) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (2)$$

Unlike Dijkstra, which methodically explores all neighbouring nodes, this metric guides GBFS to initiate the search from the node that is closest to the target, optimizing the search path by minimizing the estimated distance remaining to reach the goal.

The  $A^*$  algorithm enhances grid-based pathfinding by integrating the thorough exploration of Dijkstra's algorithm with the efficiency of Greedy Best-First Search (GBFS). It determines the total cost for each node by summing the accumulated cost  $g(n)$  to reach that node from the start, and the estimated distance  $h(n)$  from the node to the target:

$$f(n) = h(n) + g(n) \quad (3)$$

$A^*$  prioritizes nodes likely to lead to the target efficiently, balancing both search space navigation and the overall estimated cost. While  $A^*$  is more efficient and thorough than its predecessors, it can result in paths with excessive turns and redundant nodes. To mitigate these issues, strategies such as expanding search neighbourhoods have been proposed, enabling more direct paths and reducing suboptimal nodes, as discussed by Li et al. [19].

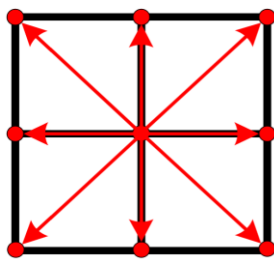


Figure 5. Eight-neighbour method

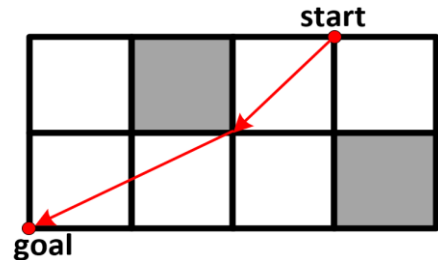
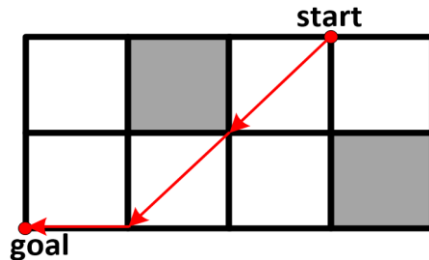


Figure 6. Traditional  $A^*$  compared with Theta\*

Instead of expanding the search area to remove suboptimal nodes as described in [19], Choi et al. [20] propose the use of Theta\* and Lazy Theta\* algorithms as evolutionary extensions of the traditional  $A^*$  algorithm. These algorithms also offer a solution for deleting poor nodes when browsing grid maps. Theta\* assesses the feasibility of linking a given node

to its neighbouring nodes indirectly, by going through the node's parent. This approach relies on the presence of an unobstructed line of sight (LOS) between the parent node and the neighbouring nodes. As shown in Fig.6 if there are no obstacles in the path, Theta\* would skip the current node and immediately connect the parent node to the neighbouring node.

However, Theta\* struggles with the requirement of performing several LOS checks. Lazy Theta\* builds on this by delaying LOS checks until absolutely necessary, thereby optimizing computational efficiency and accelerating the search process.

### 2.2.2 Local obstacle avoidance approaches

A critical component of real-time obstacle avoidance lies in the ability to process and react to sensor data in a fraction of a second. Unlike global path-planning methods, which often rely on static grid-based representations of the environment, local obstacle avoidance requires a more fluid, adaptive approach. Algorithms such as Dynamic Window Approach (DWA), Velocity Obstacles (VO), and Rapidly-exploring Random Trees (RRT) are designed to operate in the vehicle's immediate vicinity, continuously recalculating the vehicle's trajectory to ensure safety and progress towards the goal.

Considering DWA, to determine the best trajectory that avoids collisions while maintaining progress, an analysis of vehicle kinematic models is essential. Take non-omnidirectional vehicles for instance, which are limited to forward movement and rotation  $(v_r, \omega_r)$ . The trajectory based on the speed and the angular rotation of the vehicle:

$$\theta_r(t + \Delta t) = \theta_r(t) + \omega_r(t)\Delta t \quad (4)$$

$$x_r(t + \Delta t) = x_r(t) + v_r(t) \times \cos(\theta_r(t))\Delta t \quad (5)$$

$$y_r(t + \Delta t) = y_r(t) + v_r(t) \times \sin(\theta_r(t))\Delta t \quad (6)$$

According to the above equation, the motion trajectory for the subsequent time step can be deduced from the vehicle's current linear velocity  $v_r$  and angular velocity  $\omega_r$ . To ensure effective navigation, it is essential for DWA to regulate vehicles' speed at next time  $(v_n, \omega_n)$  points within their maximum and minimum limit:

$$V_s = \{ (v_n, \omega_n) \mid v_n \leq |v_{max}| \cap \omega_n \leq |\omega_{max}| \} \quad (7)$$

Due to motor torque limitations, the robot's acceleration is capped, leading to the formation of dynamic windows that define feasible speed ranges for subsequent movements:

$$V_d = \{(v_n, \omega_n) \mid v_n \in [v_r - a_{max}^- \Delta t, v_r + a_{max}^+ \Delta t] \cap \omega_n \in [\omega_r - \tau_{max}^- \Delta t, \omega_r + \tau_{max}^+ \Delta t]\} \quad (8)$$

To prevent collisions, the speed limit is also dynamically updated in real-time based on the vehicle's distance to the obstacle, denoted as  $dist(v_r, \omega_r)$ , ensuring safe navigation:

$$V_a = \{(v_n, \omega_n) \mid v_n \leq \sqrt{2dist(v_r, \omega_r) \times a_{max}\Delta t} \cap \omega_n \leq \sqrt{2dist(v_r, \omega_r) \times \tau_{max}\Delta t}\} \quad (9)$$

As long as the upcoming speed  $V_n$  falls within this range, feasible trajectories can be obtained:

$$V_n = V_s \cap V_d \cap V_a \quad (10)$$

DWA selects the optimal trajectory using the evaluation function  $G$ , which considers three main factors: *heading*, which calculates the angular difference between the vehicle's current orientation and the steering angle at the next moment, the smaller angle difference requires less torque, causing a higher evaluation value; *dist* stands for the distance from the vehicle to the obstacle at the next moment, the lower distance, the higher penalty factor; and *vel* is the vehicle's current velocity. Weights  $\alpha$ ,  $\beta$  and  $\gamma$  adjust the importance of these factors based on scenario specifics. If a feasible trajectory to avoid obstacles isn't found, a recovery behaviour is initiated, causing the vehicle to rotate in place to reassess its options:

$$G(v_n, \omega_n) = \sigma(\alpha \times heading(v_r, \omega_r) + \beta \times dist(v_r, \omega_r) + \gamma \times vel(v_r, \omega_r)) \quad (11)$$

DWA simulates the vehicle's velocity and motion trajectory for only the immediate next step within a rolling window, which lacks foresight and provides basic obstacle avoidance. In contrast, the TEB algorithm solves local trajectory through graph optimization techniques, allowing for dynamic adjustments based on real-time sensor data and environmental changes [21]. In detail, TEB algorithm constructs the local path trajectory of the global path as a graph. In this graph, the vehicle's status at time  $i$  is represented by:

$$\mathbf{X}_i^T = [x_i, y_i, \theta_i, \delta T_i] \quad (12)$$

$(x_i, y_i, \theta_i)$  is the vehicle's position and orientation, and  $\mathbf{X}_i$  represents each node on the graph. These nodes are constrained by edges such as the vehicle's dynamic, discussed in DWA, time interval, and obstacle avoidance. In TEB, the nodes connected by edges represent an elastic band that adapts to the environment by stretching, contracting, and bending. Obstacle avoidance requirements are integrated into this model through penalty functions that act like forces exerted on the band:

$$F_{total}(\mathbf{X}) = \sum_{i=1}^N (F_{repel}(\mathbf{X}_i) + F_{attract}(\mathbf{X}_i) + F_{smooth}(\mathbf{X}_i)) \quad (13)$$

$$F_{repel}(\mathbf{X}_i) = -k_{obst} \times \nabla dist(\mathbf{X}_i) \quad (14)$$

$$F_{attract}(\mathbf{X}_i) = k_{global} \times (\mathbf{X}_{goal} - \mathbf{X}_i) \quad (15)$$

$$F_{smooth}(\mathbf{X}_i) = k_{smooth} \times (\theta_i - \theta_{i-1}) \quad (16)$$

Repulsive force keeps the vehicle at a safe distance from obstacles, with its magnitude being proportional to the gradient of the vehicle's distance from the obstacle  $\nabla dist(\mathbf{X}_i)$  and the coefficient of repulsive force  $-k_{obst}$ . Smoothing force prevents abrupt directional changes, with the smoothing coefficient  $k_{smooth}$  adding to the angle difference  $\theta_i - \theta_{i-1}$ . Attractive force encourages the robot to follow the predetermined path, defined by the coefficient  $k_{global}$  and the direction towards the target position  $\mathbf{X}_{goal}$  on the global path. By integrating these forces, the TEB algorithm effectively balances obstacle avoidance and path tracking, facilitating efficient and safe path planning in complex environments.

## 2.3 Multi-vehicle collaborative formation system

Multi-vehicle collaborative formation system is an advanced application of single-agent system, where multiple vehicles interact to execute complex tasks collectively. These systems are pivotal in sectors like transportation, where they enhance operational efficiency, improve fault tolerance, and provide economic benefits by enabling parallel task execution and resource sharing [22].

### 2.3.1 Graph theory and consensus protocol in collaborative formation

Graph theory plays a crucial role in structuring the interactions within multi-vehicle collaborative formations, particularly through the implementation of leader-follower dynamics. A communication topology graph can be represented as  $G = (V, E)$ , where  $V = \{v_1, \dots, v_n\}$  denotes the set of vertices corresponding to  $n$  ( $n \geq 2$ ) vehicles, and  $E = \{(v_i, v_j) \mid i \neq j\}$  represents the collection of directed edges in the graph. Assuming the graph is a directed graph, if  $(v_i, v_j) \in E$ , this implies vertex  $i$  and  $j$  are adjacent and information exchange can be done from  $i$  to  $j$ . The neighbour set of  $v_i \in E$  is denoted as  $N = \{v_i \in V \mid (v_i, v_j) \in E\}$  and a graph is connected if  $\forall i, N_i \neq \emptyset$ . The sum of the neighbours on vertex  $v_i$ , counting both incoming and outgoing edges, is referred to as the degree of vertex  $v_i$ , denoted as  $\deg(v_i)$ . In the context of algebraic graph theory, the properties of the graph can be analysed. Let  $\deg^+(v_i)$  denote the out-degree of vertex  $v_i$ , which represents the number of edges for which the vehicle is the data sending end, and  $\deg^-(v_i)$  denote the in-degree of vertex  $v_i$ , representing the number of edges for which the vehicle is the data receiving end. We can construct the adjacency matrix  $A$  of graph  $G$ , as illustrated in Fig.7, where  $A(G)$  is a  $n \times n$  matrix and  $[A(G)]_{ij} = 1$  for  $(v_i, v_j) \in E$ ,  $[A(G)]_{ij} = 0$  for

$\nexists \{(v_i, v_j) \in E\}$ . Meanwhile, the diagonal degree matrix  $D(G)$  can be obtained by  $D(G) = \text{diag}(d_1, \dots, d_n)$ , where  $d_i = \sum_{j=1}^N a_{ij}$ , and the Laplacian graph is depicted as:

$$L(G) = D(G) - A(G) \quad (17)$$

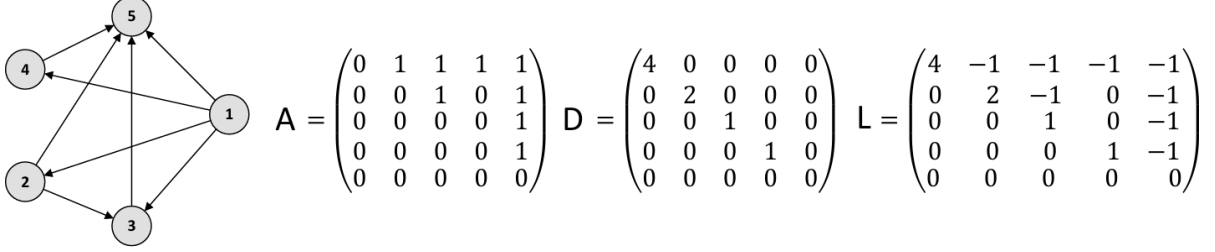


Figure 7. Communication graph topology and matrix representation

From Fig. 7, node 1 possesses the highest out-degree value, which suggests that it could be selected as the leader of the formation, whereas node 5 has no out-degree value, indicating its minimal impact on the system. In an ideal scenario, the first-order continuous-time model of the formation system can be denoted by  $\dot{x}_i = u_i$ , where  $u_i$  represents the user input vector for the  $i$ -th vehicle, which could be force, torques or velocity commands applied to the vehicle's actuators to influence its state, and  $x_i$  denotes the state vector. In networked systems or robot swarms, the control input  $u_i$  for vehicle  $i$  is determined using a consensus algorithm:

$$u_i = \sum_{j \in N_i} a_{ij} (x_j - x_i) \quad (18)$$

where  $a_{ij}$  indicates the presence (1) or absence (0) of a connection between agents  $i$  and  $j$ , and  $x_j - x_i$  denotes the relative state difference.

Equation 18 represents the basic control input update formula. In order to account for time variation, the discrete time step  $k$  must be included. This leads a revised equation:

$$u_i(k) = \sum_{j \in N_i} a_{ij} (x_j(k) - x_i(k)) \quad (19)$$

to represent the control input for every agent at every discrete time point. Within the leader-follower formation model, each vehicle must maintain a specific relative position to achieve the desired formation shape. To represent the ideal or target relative state between agents  $i$  and  $j$  at time step  $k$ , an additional term  $r_{ij}(k)$  is introduced, which guarantees the dynamic modifications align with the formation goals, capturing the temporal progression of the agents' placements. Furthermore, to ensure system stability and prevent instability due to excessive control inputs, a scaling factor  $\epsilon$  is applied to adjust the magnitude of the control input. The complete time-dependent control input formula for follower vehicles can be denoted as:

$$u_i(k) = \epsilon \sum_{j \in N_i} a_{ij} (x_j(k) - x_i(k) - r_{ij}(k)) \quad (19)$$

The position of the leader vehicle is also constrained by the follower vehicles, with the leader's speed influenced by the adjacent nodes as well as the distance to the target point:

$$u_N(k) = m + kD(k) + \sum_{i \in N} a_{Ni} r_{Ni}(k) \quad (20)$$

where  $r_{Ni}$  represents the relative position between the leader and other robots,  $m$  and  $k$  are constants,  $D(k)$  denotes the distance from the leader to the target point at time  $k$ .

### 2.3.1 Formation control algorithms and corresponding math model

Formation control algorithms are pivotal for the stability and robustness of multi-vehicle systems in complex scenarios. Key strategies include the leader-follower, virtual structure, behaviours-based, and consistency theory-based methods [23]. An innovative application of these algorithms is demonstrated by Haksar et al., who apply reinforcement learning for aerial vehicles in wildfire management [24].

In complex natural environments, ensuring safety through effective collision avoidance is critical for formation control. In a multi-vehicle system, The APF (Artificial Potential Field) method [25] is used by enabling each vehicle to navigate toward the target by balancing attractive and repulsive forces. The attractive potential field function is expressed as:

$$U_a(\mathbf{X}) = \frac{1}{2} k_a \|\mathbf{X}_i(t) - \mathbf{X}_g(t)\|^2 \quad (21)$$

where  $U_a(\mathbf{X})$  denotes the attractive potential,  $\mathbf{X}_i(t)$  the current position of vehicle  $i$ ,  $k_a$  a positive attraction coefficient, and  $\mathbf{X}_g(t)$  the target location. The attractive force is derived as:

$$F_a(\mathbf{X}) = -\nabla U_a(\mathbf{X}) = -k_a (\mathbf{X}_i(t) - \mathbf{X}_g(t)) \quad (22)$$

In contrast, the repulsive potential when an obstacle is within a critical radius  $\rho_o$  is given by:

$$U_r(\mathbf{X}) = \frac{1}{2} k_r \left( \frac{1}{\|\mathbf{X}_i(t) - \mathbf{X}_o(t)\|} - \frac{1}{\rho_o} \right)^2 \quad (23)$$

for  $\|\mathbf{X}_i(t) - \mathbf{X}_o(t)\| \leq \rho_o$  and 0 otherwise, where  $k_r$  is the repulsive force coefficient.

The corresponding repulsive force is:

$$F_r(\mathbf{X}) = -\nabla U_r = \begin{cases} k_r \left( \frac{1}{\|\mathbf{X}_i(t) - \mathbf{X}_o(t)\|} - \frac{1}{\rho_o} \right) \frac{1}{\|\mathbf{X}_i(t) - \mathbf{X}_o(t)\|^2} (\mathbf{X}_i(t) - \mathbf{X}_o(t)), & \|\mathbf{X}_i(t) - \mathbf{X}_o(t)\| \leq \rho_o \\ 0, & \|\mathbf{X}_i(t) - \mathbf{X}_o(t)\| \geq \rho_o \end{cases} \quad (24)$$

Vehicles navigate by moving along the negative gradient of the combined potential fields, although this approach may lead to local minima traps.

## 2.4 Summary of the related works

This section provided a detailed exploration of the current technological challenges and advancements in three critical areas: communication protocols in IoT networks, autonomous vehicle navigation, and multi-vehicle collaborative systems, setting the groundwork for furthering our contributions in these fields.

Initially, we analysed the deployment of transport protocols within IoT networks, highlighting the trade-offs between TCP and UDP regarding reliability and overhead. While TCP is favoured for its reliability, it suffers from high latency. In contrast, QUIC reduces latency significantly by integrating features like TLS 1.3, and UDP, which streamline the handshake process to potentially zero RTTs. QUIC also minimizes delays from packet loss and head-of-line blocking. However, its deployment in IoT devices faces challenges due to resource demands, necessitating custom protocol stacks and optimized resource management.

We then transitioned to the challenges of autonomous vehicle navigation, particularly the need for effective real-time obstacle avoidance. Techniques like Dijkstra and GBFS are expanded upon with advanced methods like the A\* algorithm, which optimizes pathfinding by considering both the cost to reach a node and the estimated distance to the target. Furthermore, local avoidance strategies like the DWA and TEB algorithm are detailed, showcasing their ability to dynamically adjust trajectories based on immediate environmental data and sensor inputs, ensuring both safety and progression towards the goal.

Additionally, we explored multi-vehicle collaborative formation systems, focusing on the essential role of interaction and coordination among multiple autonomous agents. The application of graph theory and consensus protocols structures these interactions, particularly through leader-follower dynamics. These models support robust and efficient formations, vital in high-coordination scenarios. Formation control algorithms like leader-follower and artificial potential field methods balance attractive and repulsive forces, ensuring stability and effective navigation in complex environments.

### 3 QUIC-based Multi-Vehicle Formation SLAM Framework

To improve the efficiency of map construction in delivery environments, we introduce a novel QUIC-based multi-vehicle formation SLAM framework. This framework employs edge networking and a hybrid communication system that integrates ROS-based topic communication with the QUIC protocol, which ensures rapid and reliable data exchange. Additionally, we have developed a synergistic approach that merges the leader-follower technique with the APF using the TurtleBot3 Burger model. Moreover, we incorporate Lazy Theta\* and TEB for real-time obstacle avoidance and path-planning. These enhancements facilitate efficient and safe navigation in complex environments.

#### 3.1 Overview and analysis of key challenges in autonomous systems

The review of related works has underscored three interconnected challenges within the realms of IoT network communication protocols, multi-vehicle collaborative systems and autonomous vehicle navigation. Effective multi-vehicle collaboration relies on robust communication protocols, which enable synchronization and support real-time coordination and data sharing across vehicles. This synchronization directly enhances the performance capabilities of navigation systems, which rely heavily on the precision and timeliness of data to efficiently evaluate environmental inputs, detect obstacles, and optimize route planning. Moreover, advancements in navigation technology not only boost the overall system performance but also significantly augment the safety measures in multi-vehicle formations, necessitating the continuous development of high-quality communication protocols to support these advanced systems. Tackling these interconnected challenges is therefore crucial for developing reliable and competent autonomous vehicle systems that can adeptly handle the complexities of urban environments, ensuring both operational integrity and public safety.

To address those issues faced by autonomous vehicle systems in urban environments, our approach incorporates several improved technologies within a cohesive framework. In Section 3.2, we enhance local vehicle cluster communication through edge networks, utilizing edge computing to improve speed and reliability. Section 3.3 details the implementation of a hybrid communication system that merges ROS with the QUIC protocol to optimize data transmission and enhance security across edge networks. Section 3.4 discusses integrating the leader-follower technique with the APF algorithm to enhance navigational accuracy and formation stability, with simulations conducted in Gazebo and Rviz. Section 3.5 explores the use of ROS-based SLAM systems combined with integrated



path planning algorithms like Lazy Theta\* and TEB to improve obstacle avoidance and route optimization.

### 3.2 Enhanced vehicles cluster communication system with edge networks

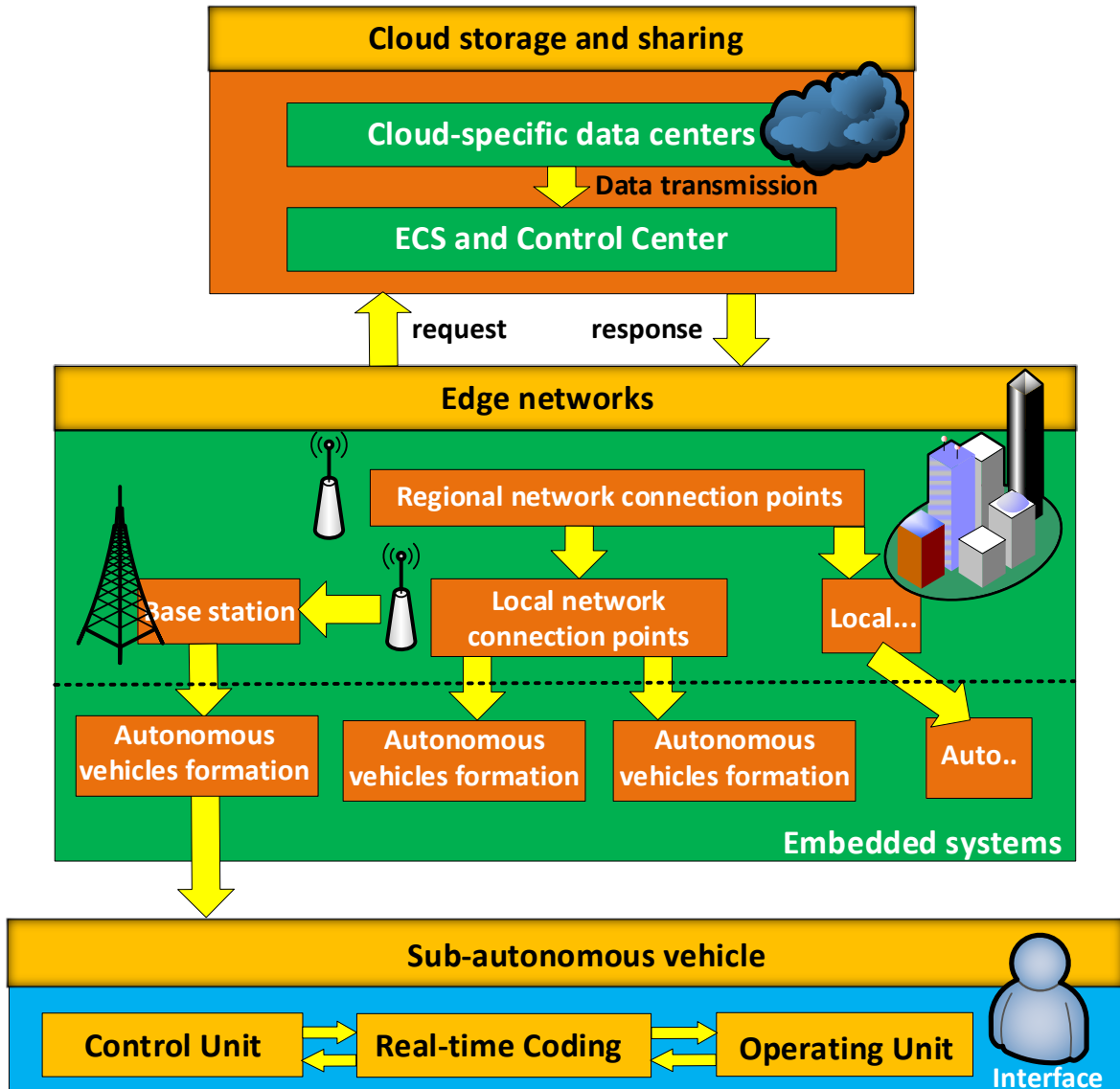


Figure 8. Framework of autonomous vehicle clusters collaborative networking

#### 3.2.1 Vehicle clusters' communication system with edge networks

In the realm of autonomous vehicle formations, the strategic integration of edge network systems plays a pivotal role in enhancing communication and operational efficiency. This integration employs a cloud storage and sharing framework centred around cloud-specific data centres, which are designed to interact efficiently with an ECS (Edge Control System) through robust data transmission protocols. As shown in Fig. 8, in our proposed framework for autonomous vehicle clusters, the ECS handles requests to the centralized cloud data

centres, which process these requests and provide responses. This setup ensures seamless, interconnected network operations across multiple edge systems, facilitating both centralized and localized management, and optimizing responsiveness and data processing at the edge, which is crucial for maintaining continuous and efficient vehicle operations.

Upon receiving responses from the central cloud, our edge networks distribute tasks to regional network connection points, including local network connections and base stations. These points are integral in managing autonomous vehicle formations, facilitating direct control or through local network connections. The adoption of the leader-follower method within these formations optimizes data flow management. In this model, follower vehicles transmit real-time sensor data to the leader vehicle, which processes this information to make immediate decisions and issue commands. This data processing occurs at edge gateways within the network, significantly reducing the volume of data transmitted to the cloud and thereby conserving bandwidth.

### **3.2.2 Operational mechanics in autonomous vehicle formations**

Within the framework of autonomous vehicle formations, each vehicle, referred to as a sub-autonomous vehicle, is equipped with a control unit, a real-time coding unit, and an operating unit. These components are intricately interconnected, forming a cohesive decision-making system. The control unit serves as the central command centre, interpreting decisions and commands from the leader vehicle and interfacing directly with the real-time coding unit, which is responsible for the immediate encoding and decoding of sensor data, ensuring that the data is processed promptly and accurately for real-time responsiveness. The operating unit then executes these commands, adjusting the vehicle's mechanisms in response to dynamic environmental conditions. This tripartite interaction ensures that each sub-autonomous vehicle operates both autonomously and as part of a larger formation, effectively responding to both centralized directives and local data inputs.

The leader-follower framework not only facilitates efficient data communication within vehicle clusters but also supports dynamic team formation and real-time operational adjustments. By leveraging edge computing, the system minimizes latency and maximizes decision-making efficacy, which is crucial in scenarios requiring rapid responsiveness. The strategic placement of local network connection points and base stations ensures robust network coverage and operational efficiency, enabling seamless coordination and management across the network of autonomous vehicle formations. This proposed integration of edge computing and network management strategies enhances the performance and

scalability of autonomous vehicle systems, illustrating an enhanced approach to vehicular technology and network architecture.

### **3.3 Hybrid communication system with ROS and QUIC**

In this section, we devise a hybrid communication system that integrates ROS-based topic communication with the QUIC protocol. This setup aims to leverage the low-latency and multiplexing capabilities of QUIC alongside the robust messaging framework provided by ROS. We explore the configuration of virtual machines for this system, detail the structure of ROS-based topic communication, and discuss the process of enabling hybrid communication between multiple virtual machines using NAT (Network Address Translation).

#### **3.3.1 Configuration of virtual machines and basic file hierarchy of ROS**

To simulate autonomous vehicle, in this report, we apply the ROS, an open-source robotics middleware suite. This framework provides a rich collection of functionalities, encapsulating common robotic functionalities into libraries, significantly reducing the amount of time developers need to spend on writing from the beginning. In our experimental configuration, we employ VMware Workstation 17 Pro to deploy the Linux simulation on a personal computer running Windows 10 (64-bit). The virtual computer is configured with 4GB of RAM, 4 CPU cores, and 40GB of HDD storage, running Ubuntu 18.04 LTS as its operating system. Upon this virtual environment, we opt for ROS Melodic release version for its compatibility with Ubuntu 18.04. Since ROS's full support for several programming languages, including C++, Python, and Octave, Python was selected as the primary programming language for our project due to its widespread use in robotics for scripting and quick development. Our edge cloud, located in Chengdu, designated for hosting a QUIC server, is equipped with a dual-core processor serving as the computational core, supplemented by 8 GB of RAM and a 100 GB SSD for storage. The control units within this network are configured to manage snapshot data packages with a capacity of up to 1 TB per month and support a bandwidth of 8 Mbps, operating within a Python 3.8 environment.

Our experimental setup focuses on demonstrating a hybrid communication system using the ROS turtlesim package. To manage our development, we established ``catkin_ws`` as our development workspace. This workspace is pivotal as it hosts all project-related files and is structured into three main folders: ``build``, ``devel``, and ``src``. The ``build`` folder stores temporary files and caches to expedite compile times, the ``devel`` folder contains compiled binaries and scripts for testing, and the ``src`` folder holds the source code for ROS packages.

For our case, the `hybrid_communication` package was developed within the `src` directory, serving as the central repository for our project files. The package includes two crucial files: `package.xml` and `CMakeLists.txt`. The former section provides an overview of the package's metadata, which includes the package's name, version, and dependencies for both building and running. The latter section describes the compilation directives, such as package names, libraries to be built and commands related to the build process, both of which guaranteeing that dependencies are properly managed and that packages are constructed in a consistent and predictable way.

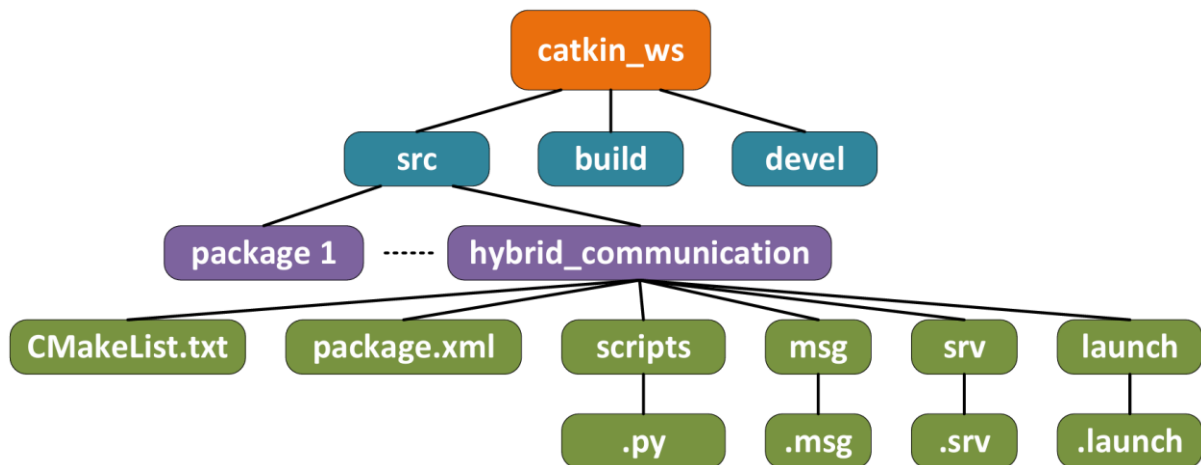


Figure 9. Basic file hierarchy of ROS

### 3.3.2 Topic communication structure of ROS

In ROS framework, the node is the fundamental unit of communication, which operates as an isolated process that performs specific tasks. Nodes, such as `turtle` and `teleop_turtle`, are executable files that interact with other nodes using ROS's messaging system. Each node must have a unique name to prevent conflicts. For straightforward keyboard control of turtle, we employ ROS's basic topic communication, which is optimal for real-time, periodic, and low-complexity data transfers. Nodes communicate via topics by registering with the ROS Master, which is the system's node manager. During registration, nodes provide essential details such as node names, topic names, message types, URI addresses, and ports. The ROS Master facilitates data transfer between publishing nodes and subscribing nodes. The publisher, such as `teleop_turtle`, sends messages through a topic, which the subscriber, `turtle`, receives and acts upon, thus establishing a direct communication link.

This structured communication architecture ensures rapid data exchange and coordination among nodes, promoting efficient interactions within the ROS environment. In our two-node communication model, the `teleop_turtle` node publishes `geometry_msgs/Twist` messages

containing motion data via the `/cmd_vel` topic, The `turtle` node subscribes to this topic, receiving and interpreting the messages to control the turtle's motion.

Furthermore, nodes in ROS framework can operate across multiple hosts, with the ROS Master centralizing their management by providing name and registration services, which facilitates node identification and interaction without the need for explicit IP addresses. For example, Node1 can transmit messages to Node2 regardless of their specific network addresses. This capability is particularly advantageous when nodes are distributed across different virtual machines, enhancing the communication system's efficiency and scalability.

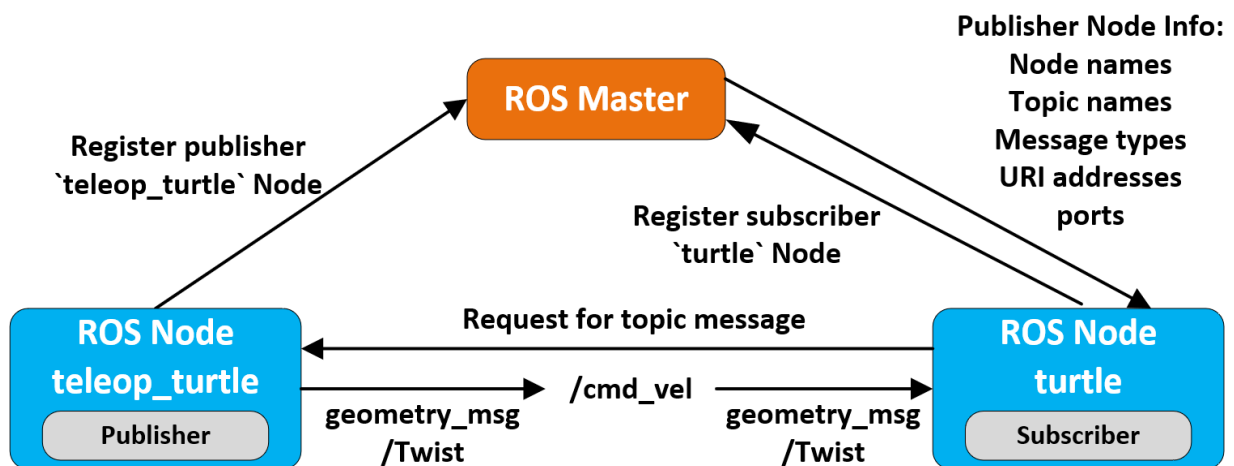


Figure 10. Simple ROS topic communication for two nodes

### 3.3.3 NAT mode communication between multiple virtual machines

To implement our hybrid communication model, it is necessary to have a systematic setup that includes two separate virtual machines (VMs). One of the VMs is set up to replicate the network environment as the server machine, while the second VM imitates the network configuration of the client. VMware establishes a comprehensive network topology, depicted in Fig.10, applying the local Wi-Fi to represent the host, with the default host gateway as 192.168.1.1, and offers the IP address 192.168.1.101 to the PC.

In this setup, VMware automatically generates a virtual network adapter linked to a default virtual switch, simplifying network management by eliminating the need for manual static IP configuration. The built-in DHCP (Dynamic Host Configuration Protocol) server assigns private IP addresses to the VMs linked to the virtual switch. The VMs are assigned a gateway address that corresponds to the address of virtual adapter. As a result, the VM 1 is assigned with the IP address 192.168.226.129, while the secondary and the third machine are assigned the IP address 192.168.226.130 and 192.168.226.131. This setup simulates the presence of different IP addresses and one virtual adapter with the local Wi-Fi.

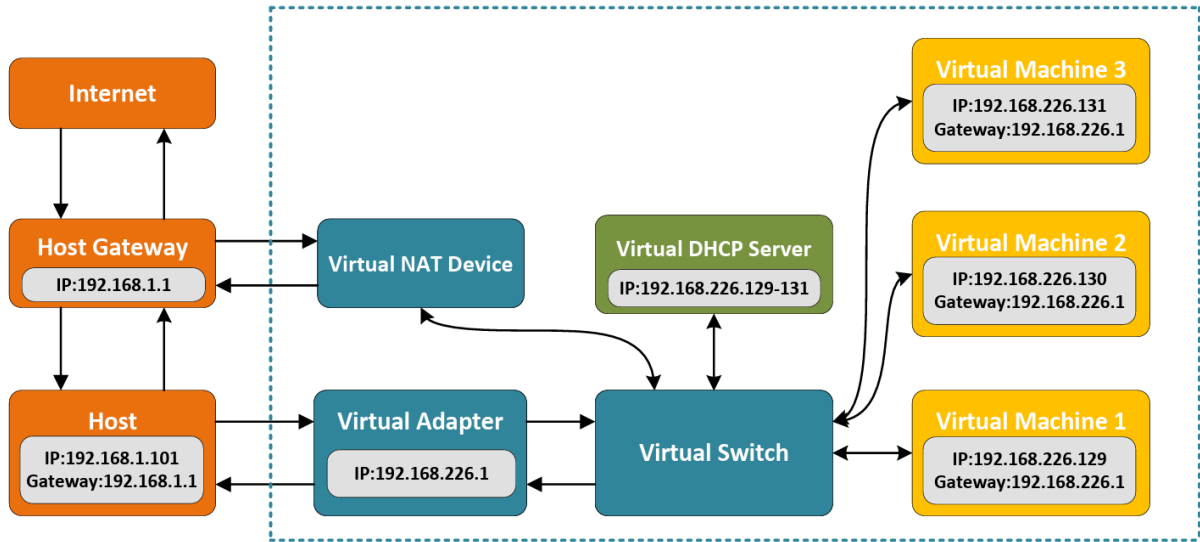


Figure 11. NAT-Network topology diagram

NAT mode improves network security and simplifies configuration by allowing external connectivity through the host machine's IP address. This configuration simplifies network management and strengthens system integrity by dividing each virtual machine's networking environment into separate compartments. By combining NAT technology with virtual network components, a strong and secure networking infrastructure is created, which is perfect for carrying out hybrid communications within the ROS framework. The robust and secure network structure is crucial for developing and testing autonomous robotic systems, illustrating the importance of efficient networking in autonomous robotics research.

### 3.3.4 Implementation of hybrid communication system

In our hybrid communication system, we detail the process of establishing a QUIC connection between two virtual machines (VMs) using ROS topic communication. We utilized `aioquic`, an open-source Python library, to implement the QUIC protocol. Notably, `aioquic` is compatible with Python 3.8, whereas ROS Melodic primarily supports Python 2.7. To resolve this compatibility issue without altering the native ROS Python environment, we employed Miniconda, which is a tiny installer for the Conda package management system. This approach allowed us to create a dedicated Python 3.8 virtual environment named `quic` on our Linux system, ensuring that our QUIC implementation could coexist with the existing ROS setup without interference.

Our implementation commences with the configuration of the host machine to expose its IP address, thereby ensuring robust network connectivity. This setup includes initializing a ROS Master on port 11311 to manage interactions involving native ROS topics. Additionally,

we configure port 4433 to support our QUIC server connection. Within the ROS network, it is typically mandatory for each ROS node to register with the ROS Master to enable synchronized communication and command exchange. However, as depicted in Fig.12, the `quic\_client` node functions as a QUIC client and distinctively omits the registration with the ROS Master. This node establishes a QUIC connection using TLS 1.3 with the `quic\_server` on the host machine, focusing on capturing keyboard inputs, translating these into Twist messages understandable by ROS, and directly transmitting messages to the `turtle` node.

Contrary to directly receiving ROS messages from the `quic\_client` node, the server circumvents the conventional necessity for message transmission via server but instead utilizes node registration, which imitates the topic communication handled by the ROS Master. This configuration permits the client node to directly manipulate the turtle's movements on the display board through keyboard inputs, which epitomizes the concept of data passthrough, facilitating direct data transmission between endpoints without intermediary processing, thereby enhancing efficiency and responsiveness in interactions.

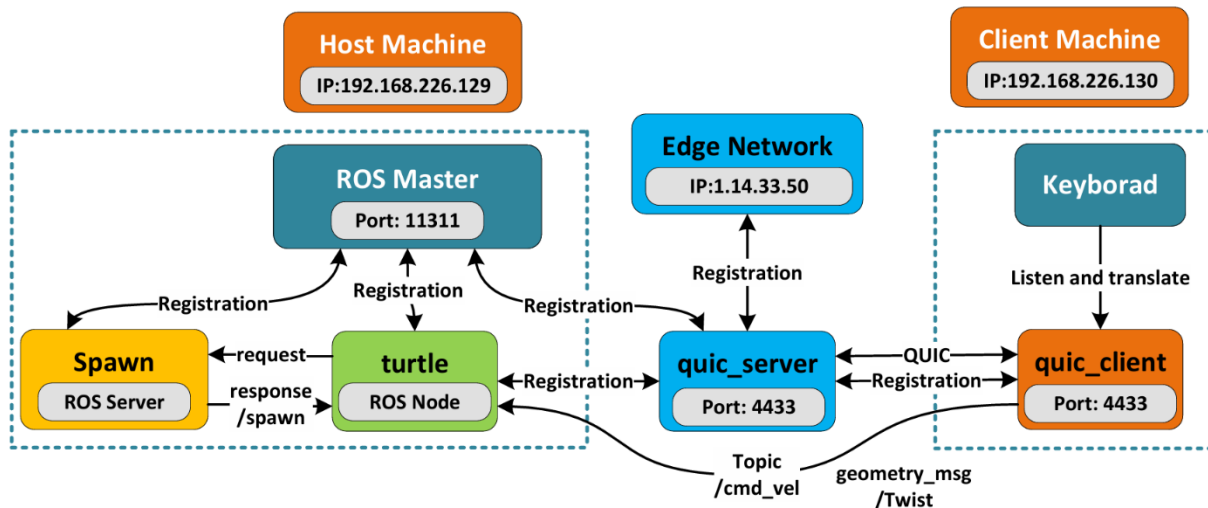


Figure 12. QUIC-based host and client communication on ROS

In our hybrid communication system, the implementation is executed using three principal algorithms. **Algorithm 1** initiates the setup of a QUIC server that listens on port 4433, employing TLS 1.3 to ensure secure communications. **Algorithm 2** delineates the operations of the QUIC client, which establishes a connection to the server using the host's IP address and the specified port, capturing keyboard inputs and transmitting them directly as ROS messages to the turtle node. **Algorithm 3** further elucidates this concept by demonstrating how the `turtle` node, upon being spawned, emulates QuicNode behaviour using the cast() function, and registers over the QUIC channel. This direct linkage minimizes

communication overhead and optimizes the pathway for rapid, secure, and immediate command execution.

**Algorithm 1** Implementation of QUIC Server

```
ROS_MASTER.start()
init_ros_node("quic_server")
s = QuicServer(listening = "0.0.0.0", port = 4433, server_certificate, key_for_certificate)
s.listen()
```

**Algorithm 2** Implementation of QUIC Client

```
c = QuicClient(IP = host_IP, port = 4433, client_key = temporary_key_for_session)
c.connect()
quic_client = QuicNode.registration(c)
while True do
    key = os.get_key()
    msg = geometry_msg.Twist(switch(key))
    quic_client.publish("/turtle/cmd_vel", msg, queue_size = 10)
end while
```

**Algorithm 3** Implementation of Hybrid Client

```
init_ros_node("turtle")
wait_for_service("/spawn")
spawn_turtle_service = RosServiceProxy("/spawn", Spawn)
response = spawn_turtle_service(x_position, y_position, theta, "turtle")
if response.name then
    hybrid = QuicNode.registration(IP = host_IP, port = 4433, cast(QuicNode, "turtle"))
    sub = hybrid.subscribe("quic_client", geometry_msg.Twist, callback=movement)
    rospy.spin()
end if
```

### 3.4 Integration of leader-follower technique with APF algorithm

In our study, the TurtleBot3 Burger model is employed as the primary robotic unit for vehicle formation due to its compact, modular design, which boasts extensive versatility and



robust support from an open-source community. This robot is fully compatible with ROS, making it suitable for a broad spectrum of robotics applications. Simulations are conducted in a Gazebo environment with three official TurtleBot3 Burger models loaded. Visualization of the robot's sensor data and operational state in real time is accomplished using Rviz. Follower behaviours are programmed in Python within the ROS Melodic framework, enabling dynamic responses to the leader robot's movements and the detection of surrounding obstacles via top-mounted LiDAR. In a tri-vehicle setup, while maintaining a triangular formation, the APF computes the potential fields induced by nearby obstacles, dictating the trajectory adjustments for each follower robot to avoid collisions effectively.

### 3.4.1 Robot's specifications and its simulation on Gazebo and Rviz

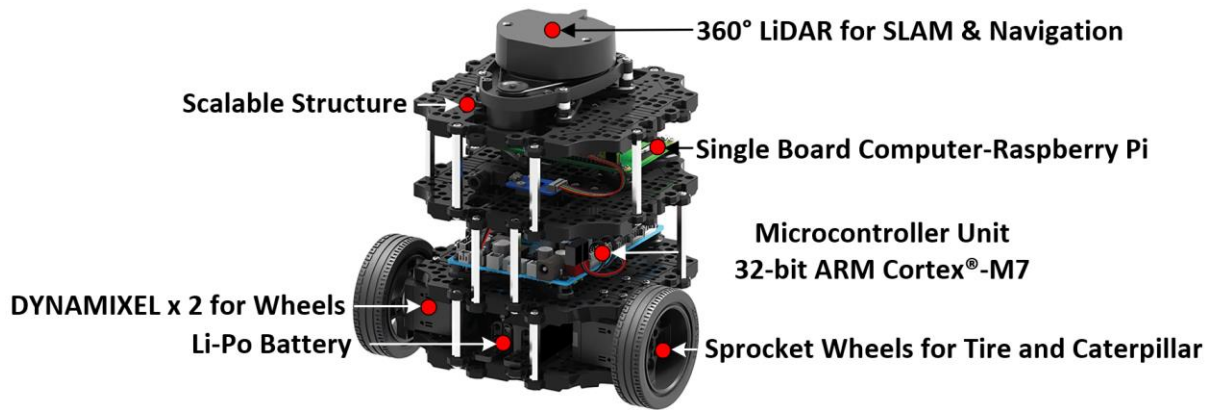


Figure 13. Main components of TurtleBot3 Burger

Fig. 13 details the main components of the TurtleBot3 Burger. Central to its operation is the Raspberry Pi, a Single Board Computer that runs ROS for high-level algorithm execution and system integration. This is coupled with a 32-bit ARM Cortex®-M7 Microcontroller Unit, which handles real-time operations and controls the DYNAMIXEL actuators to ensure precise wheel movement. Sprocket wheels linked to caterpillar tracks provide additional mobility and enable for stable movement. The robot's environmental interaction is managed by a 360° LiDAR sensor, which is crucial for SLAM and navigation. The presence of a scalable structure allows for customization, while a Li-Po battery supplies power for sustained operations, demonstrating the interdependency of these components in maintaining robust robotic functionality. Table I presents essential hardware specifications necessary for the robot's simulation, including the robot's maximum linear velocity of 0.22 m/s and maximum angular velocity of 2.84 rad/s. Additionally, it details the robot's size for collision simulation, actuators, the LDS for maximum environmental detection, and the IMU for orientation and motion tracking.

TABLE I: TURTLEBOT3 BURGER HARDWARE SPECIFICATION

Maximum Translational Velocity	0.22 m/s
Maximum Rotational Velocity	2.84 rad/s (162.72 deg/s)
Size (Length x Width x Height)	138mm x 178mm x 192mm
Actuator	XL430-W250
Laser Distance Sensor	360 Laser Distance Sensor LDS-01
Inertial Measurement Unit	Gyroscope 3 Axis & Accelerometer 3 Axis

In our study, TurtleBot3 Burger robot models are deployed in the Gazebo simulation environment, a widely recognized and versatile 3D robotics simulator used across the robotics community. Gazebo is employed for this project as it enables precise configuration of environmental variables and physical properties, facilitating the simulation of real-world physics. In addition to Gazebo, Rviz (ROS Visualization) plays an essential role as a visualization tool. It facilitates the real-time visualization of sensor data and provides critical visual feedback on the robot's trajectory, which is crucial for assessing the operational effectiveness of the integrated leader-follower technique. This capability allows for the continuous monitoring of the TurtleBot3 Burger's navigation performance and its adaptive path adjustments in response to dynamically detected obstacles, ensuring precise and effective navigation in complex environments.

### 3.4.2 Implementation of integrated leader-follower technique

After placing three TurtleBot3 Burger models in our Gazebo environment, they are configured to initiate an equilateral triangle formation. In this configuration, the leading vehicle is positioned at the vertex of the triangle, while the two follower vehicles are oriented towards the direction of the leading vehicle. The leader vehicle is controlled using the keyboard technique outlined in Section 3.3.2, along with the VM simulation settings detailed in Section 3.3.1. As depicted in Fig. 14A, the assumption is made that the leader vehicle's coordinates are known. The process of converting radar coordinate systems and determining the leader vehicle's position in the world coordinate system would be detailed in Section 3.5.

In this configuration, the `leader_tf` coordinate node of the leader vehicle is registered to the `quic_server` in the edge networking, enabling QUIC publication to two follower vehicles nodes, `followerA_formation` and `followerB_formation`. The target coordinates of the leader vehicle are constantly translated to (0.0, 0.0) within the local coordinate system of the multi-vehicle formation. Follower Vehicle A remains permanent at coordinates (-0.87, 0.5) in

order to maintain the equilateral triangle with a side length of 1m, while Follower Vehicle B endeavours to stay in place at coordinates  $(-0.87, -0.5)$ . The implementation of this triangular formation minimises the amount of overlap in SLAM, enabling the radar systems of each vehicle to efficiently detect obstacles and generate real-time maps.

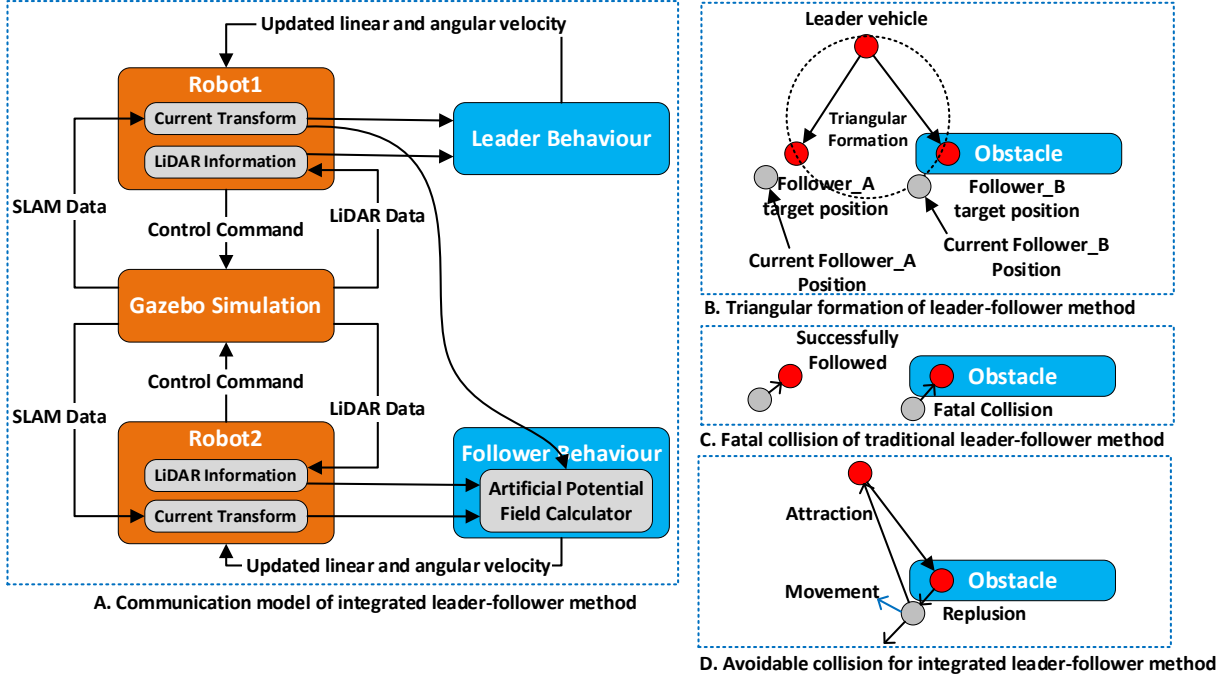


Figure 14. Graphic description of integrated leader-follower technique

Although remaining a multi-vehicle formation is relatively simple in wide roads, it becomes imperative to maintain triangular formation when entering narrow passageways in order to avoid collisions among the follower vehicles. Shown in Fig. 14B, 14C and 14D to prevent any conflicts or algorithm failures caused by both follower cars targeting a same location, we deploy the graph theoretical principles discussed in Section 2.3.1, ensuring both vehicles using the same following algorithm to operate without issues. In this case, the 'leader\_tf' node's information is transmitted to the vehicle that has a higher out-degree, which helps in efficiently managing the allocation of resources. Since our simulation including three vehicles, we simplify this process by guiding Vehicle A to the target position and then transmitting the states and coordinates to Vehicle B, which effectively avoids any conflicts or competition for resources.

The implementation codes are displayed in **Algorithm 4**. The following vehicle acquires coordinate data from the leader vehicle and operates a comparative analysis with its own positioning data. The system next computes the difference between these coordinates and assesses its influence on the follower's present operational condition. The designated states consist of: GET\_TB3\_DIRECTION, a default state of inactivity where no actions are

initiated; TB3\_DRIVE\_FORWARD, where the vehicle moves forward based on the results of the artificial potential field; and the directional states, TB3\_RIGHT\_TURN and TB3\_LEFT\_TURN, which adjust the vehicle's current direction. The vehicle's behaviour is dynamically controlled using a while loop, which depends on the current condition to ensure responsive and adaptable navigation.

**Input:** follower\_transform, leader\_transform

**Output:** Autonomous navigation commands to follower vehicle

---

**Algorithm 4** Turtlebot3 Autonomous Formation

---

```

Initialize hybrid node "follower_formation"

while not rospy.is_shutdown() do

    Calculate difference between follower_transform and leader_transform

    Calculate potential field forces based on difference and obstacles' distance

    update_follower_state(tb3_state)

    if tb3_state == TB3_DRIVE_FORWARD then

        Update linear and angular velocity based on potential field to move forward

    else if tb3_state == TB3_RIGHT_TURN then

        Update angular velocity to rotate right until target rotation is achieved

    else if tb3_state == TB3_LEFT_TURN then

        Update angular velocity to rotate left until target rotation is achieved

    end if

    Reset state to GET_TB3_DIRECTION if necessary

    Sleep according to the rate

end while

```

---

**Algorithm 5** adjusts the tb3\_state dynamically based on global variables that determine safe distances from obstacles in front, left, and right of the robot's chassis. The value of these parameters is detailed in Section VI. The function receives the current radar data from the following vehicle to compare with those safety thresholds values. Based on this comparison, tb3\_state changes to ensure optimal navigation and obstacle avoidance.

---

**Algorithm 5** Update Follower Vehicle's State

---

```

function update_follower_state(tb3_state)

```

```

if distance to centre obstacle is greater than forward safe distance then
    if distance to left obstacle is less than side safe distance then
        tb3_state = TB3_RIGHT_TURN
    else if distance to right obstacle is less than side safe distance then
        tb3_state = TB3_LEFT_TURN
    else
        tb3_state = TB3_DRIVE_FORWARD
    end if
else
    tb3_state = TB3_RIGHT_TURN
end if
end function

```

---

### 3.5 ROS SLAM system with Lazy Theta\* and TEB algorithm

In the realm of autonomous robotics in ROS environments, the ``move_base`` package plays a crucial role in navigation. It includes necessary features such as global and local path planning, motion control, environment sensing and map construction. This package relies on essential technologies such as Gmapping for SLAM, as well as costmap construction and path planning techniques including Lazy Theta\* and TEB. These technologies work together to ensure that the robot can navigate and adapt to its environment successfully, which is important for safe and effective navigation to specific destinations.

#### 3.5.1 Gmapping algorithm for SLAM and sensor coordinate's conversion

Within the framework of the ROS, our project employs the Gmapping algorithm to carry out SLAM by integrating LIDAR data with a particle filter algorithm. This amalgamation is crucial for enhancing both localization and mapping capabilities, which are fundamental for autonomous navigation. Contrasting with algorithms like Hector and Cartographer, which are tailored for uneven terrain scenarios, Gmapping algorithm offers a versatile and robust framework suitable for our simulation environments in flat warehouse.

The Gmapping algorithm first subscribes to the ``/scan`` topic, which publish the LIDAR's data. The LIDAR sensor rely on an internal motor to perform a complete 360-degree rotational sweep, enabling itself to acquire precise distance readings of nearby

obstructions. This thorough scan produces a dataset of point clouds, which serves as the essential input for the Gmapping. In order to precisely incorporate the spatial data obtained from LIDAR with the robot's real-time position in our simulation environment, the coordinate transformation is necessary. This adjustment connects the LIDAR data with the coordinate frame of the robot's base, which subscribes to two additional ROS topics: ``/tf`` and ``/odom``.

Initially, odometry data is gathered through wheel encoders and transmitted via the ``/odom`` topic, providing essential information about the robot's displacement and orientation relative to its starting position. Simultaneously, environmental data is collected by a LIDAR sensor, which is initially aligned with sensor's own local coordinate system. The ``/tf`` topic provides real-time updates on the conversions between different coordinate frames. It aligns all spatial data, such as odometry and LIDAR scans, to the robot's present physical and navigational location, known as the ``base_frame``, in the global map. In order to achieve precise 2D mapping with the Gmapping algorithm, it is essential to transform ``base_link`` to the ``base_footprint`` frame. This transformation is crucial because the ``base_footprint`` frame projects the ``base_link`` onto the ground plane, focusing solely on the x and y axes. As a result, any vertical differences are eliminated.

During the initialization process, the Gmapping algorithm processes input from the ``/tf``, ``/odom`` and ``/scan`` topics to figure out the initial position of the robot. It then generates a varied collection of particles, with each particle representing a hypothetical state of the robot. As the robot continues to move, it gathers information from LIDAR and odometry again, which improves the accuracy of each particle's position through repeated adjustments. The update mechanism consists of two phases: a prediction phase and a correction phase. In the prediction phase, the future state of each particle is estimated based on the robot's dynamics and inherent motion uncertainties. In the correction phase, the validity of each particle's state is assessed by comparing its predicted sensor outputs with the actual LIDAR measurements. Particle weights are modified according to this comparison, giving preference to those with predictions that closely match with the observed data.

By employing iterative processing, the particles that consistently produce accurate predictions are given more importance, so improving the estimation of the robot's position and orientation. At periodic intervals, Gmapping algorithm generates the precise position and orientation of the robot, as well as a detailed map that is essential for navigation and strategic decision-making in autonomous robotic systems.

### 3.5.2 Global and Local Cost Maps with Lazy Theta\* and TEB for Navigation

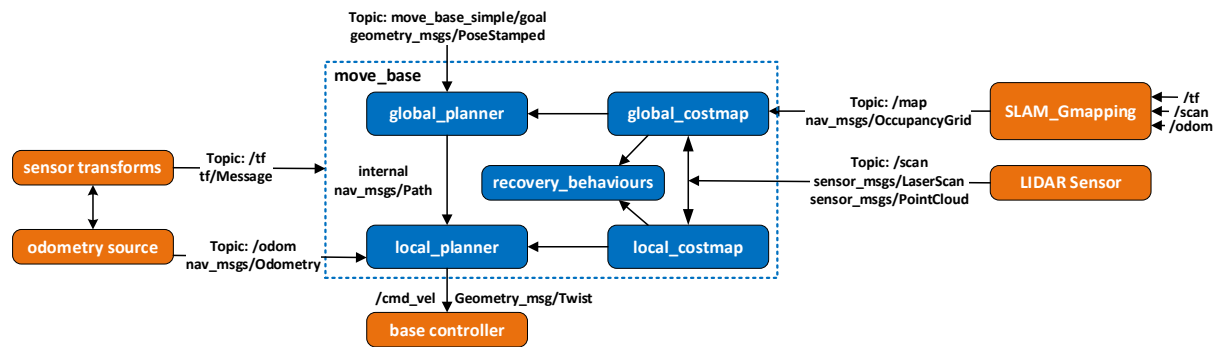


Figure 15. Framework of ROS move\_base package

This section presents the incorporation of our navigation system using the ROS move\_base package. The interconnections between the components are illustrated in Figure 15. The system begins by activating the Gmapping module, which creates an initial map integrating data obtained from the ``/scan``, ``/tf``, and ``/odom`` topics. The initial map is visualised in Rviz as a 2D image, which includes details such as size and scale, and uses grayscale values to indicate the likelihood of obstacles. However, the immobile characteristic of SLAM makes it a poor choice for navigation, which ignores the dynamic movements.

Instead of the SLAM map, the navigation system constructed by the move\_base package makes use of a costmap, which is built by overlaying dynamic information onto a static map. The global\_costmap covers the entire map region and serves the purpose in global path-planning. Conversely, the local\_costmap specifically concentrates on the immediate surroundings of the robot, continuously adapting to reflect any alterations in the environment. Furthermore, the costmap construction integrates multiple layers, which starts with a fixed SLAM map, real-time obstacle information and an inflating layer that takes into account the robot's size to avoid collisions. Additional layers, such as virtual walls, might also be included as required. The parameters of each layer, such as the radius of inflation and the scaling of cost, are modified inside the ROS framework to ensure that the robot keeps a safe distance from obstacles while minimizing deviations from its optimal path.

Once the costmap is activated, the move\_base package starts waiting for a navigation goal, which refers to the desired destination, position, and orientation of the robot within the map's frame of reference. This goal is typically sent in Rviz over a ROS topic like ``/move_base_simple/goal`` or through an action client. Upon setting a navigation goal, the goal and global\_costmap are sent to the global\_planner, which uses Lazy Theta\* to devise a feasible route to the target using the data encapsulated in the costmap. Although

global\_planner, can generate a route that covers the entire map, it does not have the ability to adapt to new or changing obstacles, highlighting the need for a responsive local\_planner.

Hence, the global\_planner transmits complete path data to the local planner via an internal ROS topic containing nav\_msgs/Path messages, which relies on sensor transformations. These are broadcasted on the `/tf` topic, including tf/message. Additionally, the local\_planner relies on obtaining odometry data from the source that publishes the `/odom` topic, which consists of nav\_msgs/Odometry, coupled with data from the local\_costmap. The local\_planner uses this combination of data to adjust the robot's trajectory in real-time using TEB algorithm, which guarantees that navigation remains safe and efficient, even in the presence of dynamic barriers or changes in the surroundings. After that, the local\_planner combines the processed data to create geometry\_msgs/Twist messages, which indicate the robot's immediate angular and linear velocities. Subsequently, these directives are transmitted to the `/cmd_vel` topic, which is received by the base controller to carry out movements.

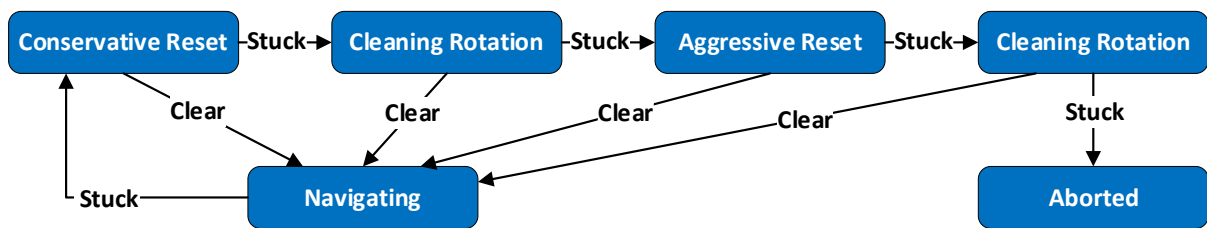


Figure 16. Default recovery behaviours of ROS move\_base package

Within the ROS framework, recovery behaviours are activated when the navigation stack encounters instances where the robot is either immobilised or unable to continue along the intended path due to unexpected impediments. The first taken action is the Conservative Reset, in which the impediments like pedestrians in the local\_costmap are temporarily removed to determine if a more straightforward path becomes viable. If this phase is unsuccessful in resolving the issue, the system move to a Clearing Rotation, where the robot will do a complete 360-degree rotation in its current position. This motion enables the robot's sensors to conduct a completely new scan of the surroundings, potentially detecting accessible routes that were previously undiscovered. If these procedures fail to solve the problem, an Aggressive Reset is used, which potentially remove all of the local\_costmap's impediments that were previously discovered. If all recovery efforts are unsuccessful, the state is designated as Aborted, signifying the need for manual intervention or a substantial alteration in strategy to proceed.



## 4 Experimental results and evaluation

This section provides detailed information on the experimental setups that were used in our study, focusing on the setup of the TurtleBot3 Burger in a simulated "warehouse-world" environment with Gazebo and Rviz. We also discuss important parameters for the APF, Gmapping, costmap, and TEB algorithms. The analysis we conducted on the QUIC-based multi-vehicle formation SLAM architecture provided valuable insights into its operational capabilities. The results reveal notable progress in the real-time processing of data and the effectiveness of autonomous vehicle clusters in navigation. However, our assessments revealed aspects that need additional enhancement, involving decreasing the computing resource of each vehicle in the formation, addressing the issue where the Gmapping algorithm does not guarantee loop closure, significantly affecting the accuracy and reliability of the generated maps, and the urgent requirement for algorithmic improvement like APF and Lazy Theta\* to lower CPU consumption and better adapt to intricate urban settings.

### 4.1 Main configurations and parameters for experiment setup

To validate our collaborative system described in Section III, we configured the virtual machines and edge cloud as specified in Section 3.3.1. The autonomous vehicle used in our experiments is the TurtleBot3 Burger, whose hardware specifications are detailed in Section 3.4.1. The experimental simulation environment employed is "warehouse-world," an open-source model provided by AWS-robotics, specifically designed for robotic mapping tasks within the Gazebo Simulation framework. For global path planning, we implemented the Lazy Theta\* algorithm using the official ROS open-source motion-planning library. This implementation adheres to the default parameters specified in the library, without any special modifications.

TABLE II details the critical constants of the APF used in integrated leader follower method. In order to avoid the following vehicle being affected by unnecessary repulsive forces generated by the force field when there are no nearby barriers, which could cause the vehicle to enter a state of local oscillation and have difficulty maintaining a smooth trajectory, a 'Repulsive Range' constant is established. This range indicates that the follower vehicle would experience repulsive forces exclusively when obstacles come within this predetermined distance.

TABLE II. ARTIFICIAL POTENTIAL FIELD CONSTANTS

Attractive Constant	1.0
Repulsive Constant	0.8
Repulsive Range (m)	0.7

Table III outlines the basic parameters used in the Gmapping technique implemented in the ROS framework. The 'Map Update Interval' parameter determines the frequency of updates in the Rviz visualisation, directly affecting the allocation of processing resources. 'Particles' parameter is crucial as it determines the number of particles used in each iteration of the particle filter algorithm. Insufficient particles may lead to inaccurate localization, while an excessive count can result in excessive waste of computing resources. The term 'Delta' denotes the level of precision in the grid map. The parameters 'Linear Update', 'Angular Update', and 'Temporal Update' are thresholds that trigger scan-matching updates necessary for enhancing the robot's pose estimation by considering its movement and the elapsed time. The term 'Iteration' refers to the number of times the scan-matching process which is essential for aligning fresh sensor scans with the current map, hence improving the accuracy of the map and the reliability of the navigation system. This is achieved by continuously updating the estimated position of the robot in response to real-world data.

TABLE III: KEY PARAMETERS FOR GMAPPING ALGORITHM ON ROS

Map Update Interval (s)	5.0	Angular Update (rad)	0.5
Particles	30	Temporal Update (s)	3
Delta (m)	0.05	Iterations	5
Linear Update (m)	1.0		

Tables IV and V specify key parameters for the `global_costmap` and `local_costmap` in robotic navigation systems. Table IV defines the 'Inflation Radius,' which expands the cost region beyond areas of collision to facilitate obstacles avoidance during path planning. The 'Update Frequency' for global maps is set at 1.5 Hz to optimise the balance between CPU burden and map responsiveness. The 'Publish Frequency' is set at 1 Hz, which is generally slower than the update rate to guarantee that the Rviz visualisation appropriately reflects the most current map modifications. Table V specifically focuses on the `local_costmap` parameters. The term 'Obstacle Range' refers to the distance at which barriers have an impact on local path planning. It ensures that only obstacles that are nearby are taken into account. The 'Raytrace Range' feature improves map accuracy by eliminating obstructions from the map after they are beyond the specified range. The 'Inflation Radius' of the local map is

generally greater than that of the global map in order to avoid collisions when performing rapid manoeuvres. The 'Update Frequency' and 'Publish Frequency' of the local\_costmap are greater than those of the global\_costmap. This is because the local\_costmap is designed to prioritise real-time updates and quick response to dynamic barriers, which is essential for ensuring safe and effective navigation in surroundings that are constantly changing.

TABLE IV: CRUCIAL PARAMETERS FOR GLOBAL\_COSTMAP

Inflation Radius (m)	0.3
Update Frequency (Hz)	1.5
Publish Frequency (Hz)	1.0

TABLE V: CRUCIAL PARAMETERS FOR LOCAL\_COSTMAP

Obstacle Range (m)	2.5	Update Frequency (Hz)	5.0
Raytrace Range (m)	3.0	Publish Frequency (Hz)	3.0
Inflation Radius (m)	0.5		

Table VI provides details about the essential parameter modifications for the TEB path-planning method, with a particular emphasis on constraints related to velocity and acceleration. The parameters 'max\_vel\_x' and 'acc\_lim\_x' define the maximum values for velocity and acceleration. The 'acc\_lim\_x' parameter also includes deceleration. Motors that have slow response times necessitate a precise restriction on acceleration in order to maintain stability. Furthermore, the parameters 'max\_vel\_theta' and 'acc\_lim\_theta' control the rate of change of angular velocity and angular acceleration, which are particularly important in keeping the vehicle from tipping over or experiencing wheel slide during sudden and fast manoeuvres. In addition, by increasing the 'penalty\_epsilon' value, a buffer similar to an inflation layer is introduced, which penalises speeds that are close to the limit. Increasing the value of 'weight\_optimaltime' enhances the speed of acceleration on straight sections and improves the accuracy of navigation along the edges of the path during turns. Modifying the 'weight\_kinematics\_forward\_drive' parameter limits the ability to design backward motion. The presence of consistent reverse tracks indicates a shortage of feasible ahead routes.

TABLE VI: ESSENTIAL PARAMETERS FOR TEB\_LOCAL\_PLANNER

max_vel_x (m/s)	0.5	penalty_epsilon	0.5
acc_lim_x ( $\text{m/s}^2$ )	2	weight_optimaltime	5
max_vel_theta (rad/s)	1.5	weight_kinematics_forward_drive	100

acc_lim_theta (rad/s <sup>2</sup> )	3.2		
-------------------------------------	-----	--	--

## 4.2 Experimental results and evaluation on SLAM framework

TABLE VII: NETWORK LATENCY BETWEEN QUIC AND ROSTCP

QUIC/tf_message (ms)	45.3	ROSTCP/tf_message (ms)	58.6
----------------------	------	------------------------	------

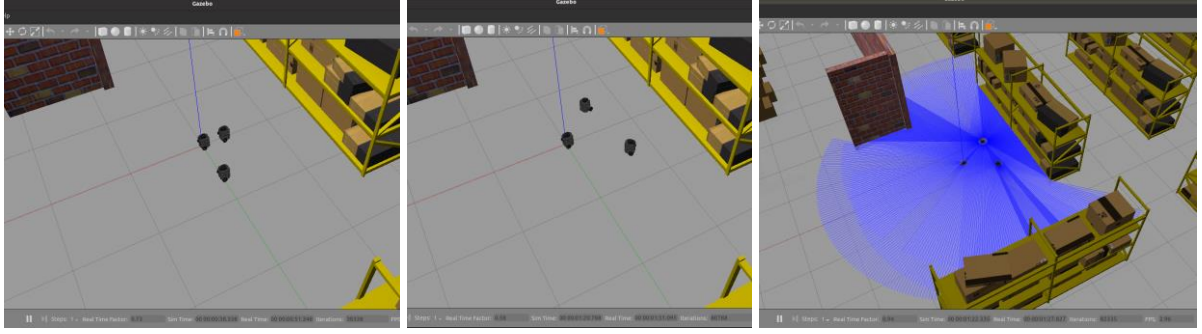


Figure 17. Autonomous vehicles employing APF for formation and initiating Gmapping

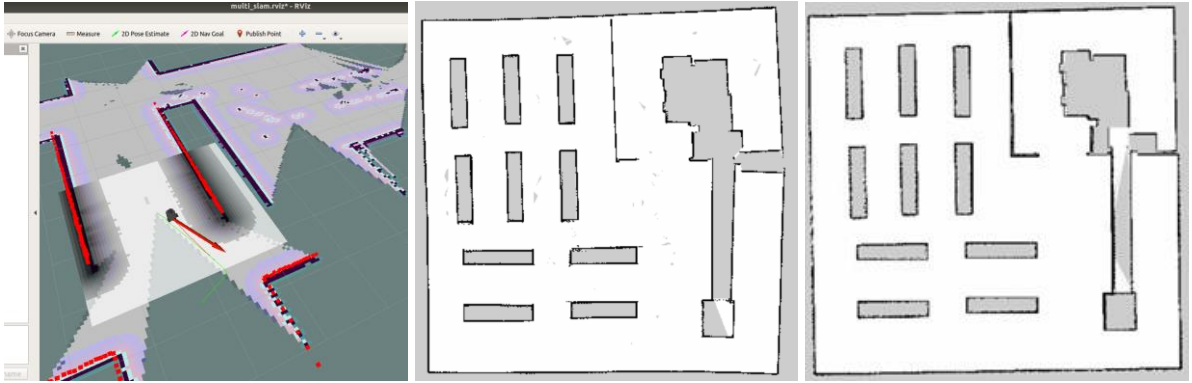


Figure 18. Generating 2D Maps via Navigation Commands in Rviz

Both QUIC and ROSTCP protocol handles network packages efficiently and conduct integrated leader-follower SLAM framework, the time delay is detailed in Table VII, where QUIC improves 29.36% overall network transmission in ROS. Fig. 17 showcases the proficient route planning abilities of our self-driving vehicle through the use of the APF algorithm. Although the near proximity of the two vehicles, the APF algorithm effectively avoids collisions by creating an optimal triangular arrangement, which is suited for SLAM navigation as explained in Section 3.4.2. The adoption of the Lazy Theta\* method in the navigation leads to a far smoother trajectory, as shown in Figure 18, in contrast to the sporadic vertical routes commonly observed in the standard A\* approach. In addition, the mapping results, as depicted in Figure 18, demonstrate that the Gmapping algorithm yields a comparable outcome to the official mapping results in a warehouse setting, effectively facilitating navigation.

However, our Gmapping method is set to update the map every five seconds at most and thirty particles at a time, to minimize the computational resources, results in specific locations on the final map are displayed as grey dots. These indicate places that were not included in the mapping process, with constraint which is generally ascribed to the configurations of particle filter technique. For particle filter algorithm, to obtain a highly comprehensive map similar to the official version depicted in Fig. 18, the navigation robot had to repeatedly travel along paths that were influenced by noise. This greatly prolonged the time required to finish the mapping process. In addition, the final mapping, although comprehensive, shows divergence from the required square geometry since the Gmapping method lacks loop closure, which is lack for ensuring perceive closure.

Furthermore, as explained in Section 3.4.2 regarding narrow path situations, the robot showed reluctance in exploring possible routes since the inflate radius for obstructions is set too high, resulting in obvious empty spaces in the mapped area. The differences discovered and the failure of the mapping process to converge indicate significant departures from the desired square layout, which emphasise the need to further optimise the particle filter method or explore alternate algorithms to improve mapping accuracy and efficiency.

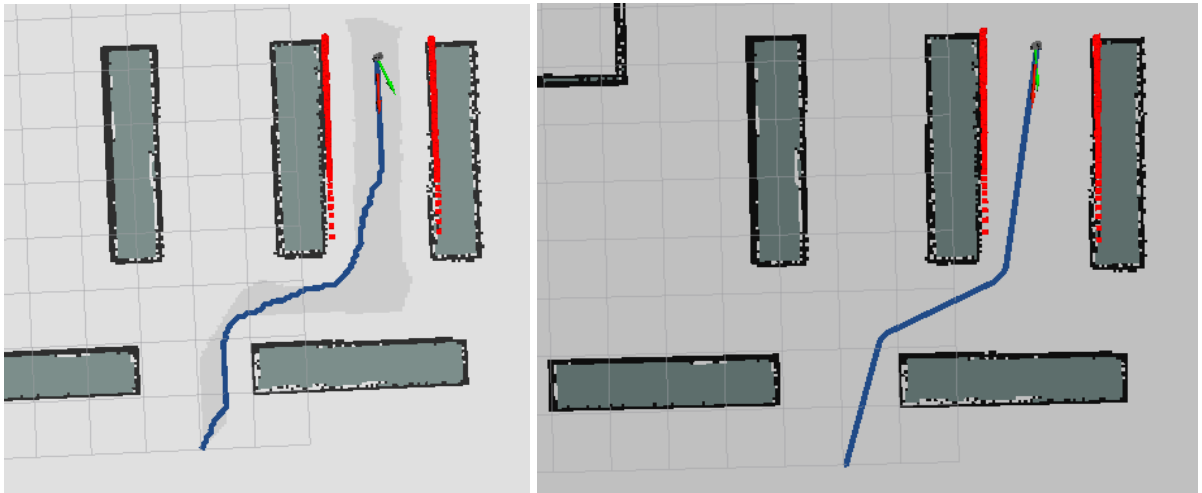


Figure 19. Comparison of overall route planning for A\* and Lazy Theta\*

In the meantime, we deploy one autonomous vehicle to navigate on a pre-existing map, allowing us to solely compare pathfinding algorithms. As shown in Fig. 19, the A\* algorithm took 32.96 seconds to finish the pathfinding tests, whereas the Lazy Theta\* algorithm accomplished the same task in 28.11 seconds. The Lazy Theta\* algorithm's planned path exhibits smoother characteristics, resulting in less rotation and minimised tyre slippage in real-world scenarios. This is achieved by deleting unnecessary nodes, hence improving the efficiency of the pathfinding process. However, we also employ command-line

tools to monitor the CPU overhead of each pathfinding approach. The Lazy Theta\* method exhibited a higher CPU utilisation of 15% in the Rviz window, in contrast to the A\* algorithm which had a lower utilisation of only 12%. The reason for this increase is linked to the requirement of Lazy Theta\* to traverse the full tree, which results in a longer initialization time. While it improves the efficiency of pathfinding, it requires substantial computational resources from the Raspberry Pi of the autonomous vehicle.

Both global and local path planning consume CPU resources. Although effort has been done to minimize the CPU resources consuming, the leader vehicle holds comprehensive global and local path planning skills, instead the following vehicle employs the algorithm outlined in Section 3.4.2 to navigate around intricate obstacles. Even so, the CPU utilisation reached its highest point at 64%. This encompasses the additional computational load caused by Gmapping particle filtering and QUIC synchronisation. In this procedure, each particle represents a complete map, resulting in the consumption of significant resources. In addition, the communication system, which was redesigned using Python, created several nodes to further require CPU resources to receive each individual piece of transmitted data.

As a result, it is imperative to enhance the efficiency of computational resources in order to guarantee the seamless functioning of IoT devices in practical situations, preventing system failures caused by overwhelming computational requirements.

## **5 Conclusions and future works**

### **5.1 Conclusions**

This project introduces several contributions to the field of robotic navigation and mapping, particularly in delivery environments. We propose a novel QUIC-based multi-vehicle formation SLAM framework that markedly enhances the efficiency of map construction. Additionally, we have devised a hybrid communication system that merges edge network capabilities and ROS-based topic communication with the QUIC protocol, offering a robust solution for high-speed data transmission. Our research also presents a synergistic approach utilizing the leader-follower method coupled with the APF on ROS-based simulations. This method equips vehicles with the ability to navigate complex environments while maintaining formation integrity. Furthermore, we have constructed a revised SLAM method for a single ROS-based vehicle and fine-tuned related parameters to ensure robust localization and mapping accuracy.

Despite these advances, our project has identified critical areas for improvement. The system evaluation highlighted the need for better management of data transmission from peripheral devices to central cloud servers and the challenges of implementing the QUIC protocol on devices like the Raspberry Pi, which may not inherently support it, requiring additional configurations. Moreover, the Gmapping algorithm does not guarantee loop closure, which is pivotal for accurate map construction. The extended response times and computational demands of the Lazy Theta\* algorithm in dynamic urban environments, brings issues coupled with the incorporation of APF in the leader-follower technique, inducing local oscillations in the follower vehicle's trajectory due to the interplay of attractive and repulsive forces. These issues underscore the need for further refinement of the algorithms and system enhancements to effectively handle the complexities of varying urban settings.

### **5.2 Suggestions for future works**

Reflecting on the shortcomings and challenges encountered during this project, it is evident that future research should aim for development include refining data transmission management to ensure efficient and secure data flows from peripheral devices to central cloud servers, exploring advanced network protocols suitable for distributed systems, and resolving compatibility issues with hardware such as Raspberry Pi to support the QUIC

protocol. This may involve creating or adapting middleware that facilitates QUIC integration in non-supportive environments.

Moreover, optimizing map construction algorithm like Gmapping and Lazy Theta\* algorithm to decrease reaction times and ensure precise map in dynamic urban settings is essential. Efforts should also extend to stabilizing vehicle trajectories when employing APF in the leader-follower technique, potentially through developing advanced control strategies that mitigate oscillations caused by the interplay of attractive and repulsive forces. These technical enhancements should be complemented by innovations in system energy efficiency and the integration of predictive analytics to manage resources effectively.

Implementing these comprehensive improvements will not only address the specific technical limitations identified but also enhance the system's practical applicability, making it a more robust solution for urban logistics and autonomous vehicle management. Such advancements are crucial for realizing the full potential of autonomous vehicular technology in enhancing urban transportation systems, ultimately leading to smarter, more reliable, and sustainable urban mobility solutions. These efforts will collectively improve the framework's effectiveness and reliability across various urban settings, ensuring consistent performance even under fluctuating network conditions and diverse operational demands.



## References

- [1] Sharma Abhinav; Jain Arpit; Ram Mangey, "3 AI for COVID-19: The Journey So Far," in *Use of AI, Robotics, and Modern Tools to Fight Covid-19*, River Publishers, 2021, pp. 29-43.
- [2] R. Yao, "Multiple Optimization Methods of Automatic Application in Last-mile Delivery System," *2023 International Conference on Power, Electrical Engineering, Electronics and Control (PEEEEC)*, Athens, Greece, 2023, pp. 291-294.
- [3] K. -M. Yang, J. -B. Han and K. -H. Seo, "A Multi-robot Control System based on ROS for Exploring Disaster Environment," *2019 7th International Conference on Control, Mechatronics and Automation (ICCMA)*, Delft, Netherlands, 2019, pp. 168-173.
- [4] J. Levinson et al., "Towards fully autonomous driving: Systems and algorithms," *2011 IEEE Intelligent Vehicles Symposium (IV)*, Baden-Baden, Germany, 2011, pp. 163-168.
- [5] O T. L. Willke, P. Tientrakool and N. F. Maxemchuk, "A survey of inter-vehicle communication protocols and their applications," in *IEEE Communications Surveys & Tutorials*, vol. 11, no. 2, pp. 3-20, Second Quarter 2009.
- [6] H. Zhao et al., "QUIC-Enabled Data Aggregation for Short Packet Communication in mMTC," *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, New York, NY, USA, 2022, pp. 1-2.
- [7] F. Chiti, R. Fantacci, D. Giuli, F. Paganelli, and G. Rigazzi, "Communications Protocol Design for 5G Vehicular Networks," in *5G Mobile Communications*, Cham: Springer, 2017, pp. 625-649.
- [8] J. Han, M. Cui, Y. Lv, K. Liu, Q. Dai and H. Guo, "Moving Horizon path planning for intelligent vehicle oriented to dynamic obstacle avoidance," *2022 6th CAA International Conference on Vehicular Control and Intelligence (CVCI)*, Nanjing, China, 2022, pp. 1-6.
- [9] S. K. Tse, Y. Ben Wong, J. Tang, P. Duan, S. W. W. Leung and L. Shi, "Relative State Formation-based Warehouse Multi-robot Collaborative Parcel Moving," *2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*, Victoria, BC, Canada, 2021, pp. 375-380.

- [10] J. Luo, J. Jin and F. Shan, "Standardization of Low-Latency TCP with Explicit Congestion Notification: A Survey," in *IEEE Internet Computing*, vol. 21, no. 1, pp. 48-55, Jan.-Feb. 2017.
- [11] M. Masirap, M. H. Amaran, Y. M. Yussoff, R. A. Rahman and H. Hashim, "Evaluation of reliable UDP-based transport protocols for Internet of Things (IoT)," *2016 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, Penang, Malaysia, 2016, pp. 200-205.
- [12] T. Yokotani and Y. Sasaki, "Comparison with HTTP and MQTT on required network resources for IoT," *2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, Bandung, Indonesia, 2016, pp. 1-6.
- [13] F. Fernández, M. Zverev, P. Garrido, J. R. Juárez, J. Bilbao and R. Agüero, "And QUIC meets IoT: performance assessment of MQTT over QUIC," *2020 16th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Thessaloniki, Greece, 2020, pp. 1-6
- [14] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, et al., "The QUIC transport protocol: design and internet-scale deployment," in: *ACM SIGCOMM Computer Communication Review*, 2017, pp. 183- 196.
- [15] L. Eggert, "Towards securing the Internet of Things with QUIC", *Proc. Workshop Decentralized IoT Syst. Security (DISS)*, pp. 9, 2020.
- [16] P. Kumar and B. Dezfouli, "Implementation and analysis of QUIC for MQTT", *Comput. Netw.*, vol. 150, pp. 28-45, Feb. 2019.
- [17] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley and H. Tokuda, "Is it still possible to extend TCP?", *Proceedings of the ACM SIGCOMM Internet Measurement Conference IMC*, 2011, pp. 181-194.
- [18] Y. Han, Z. Wang, M. Jiang and L. Zhang, "Simulation and Test of Vehicle Global Routing Planning Algorithm Based on Autonomous Driving Simulator," *2022 34th Chinese Control and Decision Conference (CCDC)*, Hefei, China, 2022, pp. 604-609.
- [19] X. Li, X. Hu, Z. Wang and Z. Du, "Path Planning Based on Combination of Improved A-STAR Algorithm and DWA Algorithm," *2020 2nd International Conference on*

*Artificial Intelligence and Advanced Manufacture (AIAM)*, Manchester, United Kingdom, 2020, pp. 99-103.

- [20] S. Choi and W. Yu, "Any-angle path planning on non-uniform costmaps", *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 5615-5621.
- [21] C. Liu and Y. Liu, "Robot planning and control method based on improved time elastic band algorithm," *2023 4th International Conference on Computer Engineering and Application (ICCEA)*, Hangzhou, China, 2023, pp. 911-915.
- [22] F. Ropero, P. Munoz and M. D. R-Moreno, "TERRA: A path planning algorithm for cooperative UGV-UAV exploration", *Engineering Applications of Artificial Intelligence*, vol. 78, 2019, pp. 260-272.
- [23] R. Baich and C. Arkin, "Behaviour-based formation control for multi-robot teams[J]", *IEEE Transactions on Robotics & Automation*, vol. 14, no. 6, pp. 926-939, 1998.
- [24] R. N. Haksar and M. Schwager, "Distributed deep reinforcement learning for fighting forest fires with a network of aerial robots", *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.
- [25] Y. C. Chang and Y. Yamamoto, "Online dead-lock avoidance scheme of wheeled mobile robot under the presence of boxlike obstacles", *IEEE/ ASME International Conference on Advanced Intelligent Mechatronics*, pp. 1535-1540, 2005.

## A Appendices

### **multi\_robot\_gazebo.launch**

<launch>

<!-- Set initial parameters before loading the simulation environment. These are set to their default values. -->

<arg name="paused" default="false"/>

<arg name="use\_sim\_time" default="true"/>

<arg name="gui" default="true"/>

<arg name="headless" default="false"/>

<arg name="debug" default="false"/>

<!-- Launch the Gazebo simulation environment using the parameters defined above. Starts with an empty world. -->

<include file="\$(find gazebo\_ros)/launch/empty\_world.launch">

<!-- Specify the world file to be loaded in the Gazebo simulator. -->

<arg name="world\_name" value="\$(find sim\_env)/worlds/warehouse.world" />

<!-- Pass the arguments to the empty world simulation. -->

<arg name="paused" value="\$(arg paused)"/>

<arg name="use\_sim\_time" value="\$(arg use\_sim\_time)"/>

<arg name="gui" value="\$(arg gui)"/>

<arg name="headless" value="\$(arg headless)"/>

<arg name="debug" value="\$(arg debug)"/>

</include>

<!-- After launching the empty world and setting the parameters, include the launch file for spawning individual robots. -->

<group ns="robot1">

<!-- Define a namespace for this group, allowing multiple robots to operate within the same simulation environment without interference. -->

```

<include file="$(find multi-robot-gazebo)/launch/one_robot.launch">
  <!-- Set initial pose arguments for the robot in this namespace. -->
  <arg name="init_pose_x" value="0"/>
  <arg name="init_pose_y" value="0"/>
  <arg name="init_pose_z" value="0"/>
  <arg name="init_pose_a" value="0"/>
  <arg name="multi_robot_name" value="robot1"/>
</include>
</group>

<group ns="robot2">
  <!-- Similar to the first group, define a separate namespace for another robot. -->
  <include file="$(find multi-robot-gazebo)/launch/one_robot.launch">
    <!-- Set different initial pose arguments for this robot to avoid overlap with others. -->
    <arg name="init_pose_x" value="-0.87"/>
    <arg name="init_pose_y" value="0.5"/>
    <arg name="init_pose_z" value="0"/>
    <arg name="init_pose_a" value="0"/>
    <arg name="multi_robot_name" value="robot2"/>
  </include>
</group>

<group ns="robot3">
  <!-- Define another unique namespace for the third robot to ensure isolated control
within the same simulation. -->
  <include file="$(find multi-robot-gazebo)/launch/one_robot.launch">
    <!-- Set initial pose arguments for this robot, distinct from the others to maintain
separation. -->
    <arg name="init_pose_x" value="-0.87"/>
    <arg name="init_pose_y" value="-0.5"/>

```

```

    <arg name="init_pose_z" value="0"/>
    <arg name="init_pose_a" value="0"/>
    <arg name="multi_robot_name" value="robot3"/>
  </include>
</group>
</launch>

```

### **follower\_formation.py**

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Import necessary libraries
import rospy
import math
import sys
import tf2_ros
from typing import cast
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist, TransformStamped
from nav_msgs.msg import Odometry
from quicross import *

# Constants for angle conversions
DEG2RAD = math.pi / 180
RAD2DEG = 180 / math.pi

# Constants for robot states
CENTER = 0
LEFT = 1
RIGHT = 2

```

```

LINEAR_VELOCITY = 0.3
ANGULAR_VELOCITY = 1.5
GET_TB3_DIRECTION = 0
TB3_DRIVE_FORWARD = 1
TB3_RIGHT_TURN = 2
TB3_LEFT_TURN = 3

# APF constants
ATTRACTIVE_FORCE = 1.0
REPULSIVE_FORCE = 0.8
REPULSIVE_RANGE = 0.7

class Turtlebot3Drive:
    def __init__(self, source_frame, target_frame):
        # Initialize the hybrid node
        rospy.init_node('turtlebot3_drive', anonymous=True)

        c = cast(QuicClient(IP = '1.14.33.50', port = 4433, client_key=None), 'turtlebot3_drive')
        c.connect()

        quic_client = quicros.QuicClient.registration(c)

        self.source_frame = source_frame
        self.target_frame = target_frame
        self.scan_data = [0.0, 0.0, 0.0]

        # Set up publishers and subscribers

        self.cmd_vel_pub = rospy.Publisher('{} /cmd_vel'.format(source_frame), Twist,
queue_size=10)

        self.laser_scan_sub = rospy.Subscriber('{} /scan'.format(source_frame), LaserScan,
self.laser_scan_msg_callback, queue_size=10)

        self.odom_sub = rospy.Subscriber('{} /odom'.format(source_frame), Odometry,
self.odom_callback, queue_size=10)

```

```

# Initialize TF2 buffer and listener for transformations
self.tf_buffer = tf2_ros.Buffer()
self.tf_listener = tf2_ros.TransformListener(self.tf_buffer)

# Navigation and turning parameters
self.escape_range = 30 * DEG2RAD
self.check_forward_dist = 0.7
self.check_side_dist = 0.6
self.tb3_pose = 0.0
self.prev_tb3_pose = 0.0
self.tb3_state = GET_TB3_DIRECTION
self.robot1_to_robot2 = None

def laser_scan_msg_callback(self, msg):
    # Parse laser scan data
    scan_angle = [180, 210, 150]
    for num in range(3):
        if math.isinf(msg.ranges[scan_angle[num]]):
            self.scan_data[num] = msg.range_max
        else:
            self.scan_data[num] = msg.ranges[scan_angle[num]]

def odom_callback(self, msg):
    # Calculate orientation from quaternion
    sin_y = 2.0 * (msg.pose.pose.orientation.w * msg.pose.pose.orientation.z +
msg.pose.pose.orientation.x * msg.pose.pose.orientation.y)
    cos_y = 1.0 - 2.0 * (msg.pose.pose.orientation.y * msg.pose.pose.orientation.y +
msg.pose.pose.orientation.z * msg.pose.pose.orientation.z)
    self.tb3_pose = math.atan2(sin_y, cos_y)

```



```

def update_command_velocity(self, linear, angular):

    # Publish new velocity commands

    cmd_vel = Twist()

    cmd_vel.linear.x = linear

    cmd_vel.angular.z = angular

    self.cmd_vel_pub.publish(cmd_vel)


def calculate_potential_field(self):

    # Calculate forces for potential field navigation

    attractive_force_x = ATTRACTIVE_FORCE *
self.robot1_to_robot2.transform.translation.x

    attractive_force_y = ATTRACTIVE_FORCE *
self.robot1_to_robot2.transform.translation.y

    repulsive_force_x = 0.0

    repulsive_force_y = 0.0


    for i, distance in enumerate(self.scan_data):

        if distance < REPULSIVE_RANGE:

            angle = math.radians(i * 60 - 30)

            force = REPULSIVE_FORCE * (1.0 / distance - 1.0 / REPULSIVE_RANGE)

            repulsive_force_x += force * math.cos(angle)

            repulsive_force_y += force * math.sin(angle)


    total_force_x = attractive_force_x - repulsive_force_x

    total_force_y = attractive_force_y - repulsive_force_y


    force_magnitude = math.sqrt(total_force_x ** 2 + total_force_y ** 2)

    force_angle = math.atan2(total_force_y, total_force_x)


    return force_magnitude, force_angle

```

```

def update_tb3_state(self):
    # Determine the state of the robot based on scan data and potential field
    force_magnitude, force_angle = self.calculate_potential_field()
    if selfscan_data[CENTER] > self.check_forward_dist:
        if self.scan_data[LEFT] < self.check_side_dist:
            self.tb3_state = TB3_RIGHT_TURN
        elif self.scan_data[RIGHT] < self.check_side_dist:
            self.tb3_state = TB3_LEFT_TURN
        else:
            self.tb3_state = TB3_DRIVE_FORWARD
    else:
        self.tb3_state = TB3_RIGHT_TURN

    if self.tb3_state == TB3_DRIVE_FORWARD:
        self.update_command_velocity(min(force_magnitude, LINEAR_VELOCITY),
force_angle)
    elif self.tb3_state in [TB3_RIGHT_TURN, TB3_LEFT_TURN]:
        if abs(self.prev_tb3_pose - self.tb3_pose) >= self.escape_range:
            self.tb3_state = GET_TB3_DIRECTION
        else:
            angular_direction = -ANGULAR_VELOCITY if self.tb3_state ==
TB3_RIGHT_TURN else ANGULAR_VELOCITY
            self.update_command_velocity(0.0, angular_direction)

def control_loop(self):
    # Main control loop running at 50Hz
    rate = rospy.Rate(50)
    while not rospy.is_shutdown():
        try:

```

```

        # Update transformations and robot state continually
        self.robot1_to_robot2 = self.tf_buffer.lookup_transform(self.source_frame,
self.target_frame, rospy.Time(0), rospy.Duration(1))

        self.update_tb3_state()

        rate.sleep()

    except (tf2_ros.LookupException, tf2_ros.ConnectivityException,
tf2_ros.ExtrapolationException) as e:

        # Handle TF2 exceptions
        rospy.logerr("TF2 exception: %s", e)

        continue

if __name__ == '__main__':
    try:
        # Check for correct number of command line arguments
        if len(sys.argv) != 3:
            rospy.logerr("Invalid number of parameters")
            sys.exit(1)

        target_frame = sys.argv[1]
        source_frame = sys.argv[2]

        # Initialize and start the Turtlebot3 driving control
        tb3_drive = Turtlebot3Drive(source_frame, target_frame)
        tb3_drive.control_loop()

    except rospy.ROSInterruptException as e:
        rospy.logerr(e)

```

### **frame.py**

```

import enum

from dataclasses import dataclass

from typing import Type

import genpy

```

```

@dataclass
# Defines parameters for publishing messages on a ROS topic
class AdvertiseReq:
    name: str = ""
    data_class: Type[genpy.Message] = None
    queue_size: int = 0

```

```

@dataclass
# Parameters for subscribing to a ROS topic
class SubscribeReq:
    name: str = ""
    data_class: Type[genpy.Message] = None

```

```

@dataclass
# Service request to create real-time responsive services in ROS
class ServiceReq:
    name: str = ""
    service_class: Type = None

```

```

@dataclass
# Service proxy for making service requests from a client node
class ServiceProxyReq:
    name: str = ""
    service_class: Type = None

```

```

# Enumerates types of message requests within ROS
class ReqType(enum.IntEnum):
    INVALID = 0

```

```
ADVERTISE = 1
SUBSCRIBE = 2
SERVICE = 3
SERVICE_PROXY = 4
```

```
@dataclass
# General container for different types of ROS requests
class Req:
    type: ReqType = ReqType.INVALID
    advertise_req: AdvertiseReq = None
    subscribe_req: SubscribeReq = None
    service_req: ServiceReq = None
    service_proxy_req: ServiceProxyReq = None
```

```
@dataclass
# Data class for response with error handling
class Rsp:
    err_code: int = 0
    err_msg: str = ""
```

```
@dataclass
# Represents a generic ROS message
class ROSMessage:
    msg: genpy.Message = None
```

```
codec.py
import enum
import importlib
import inspect
```

```

import io
import json
from dataclasses import is_dataclass
from typing import Type, Any
import genpy
from .frame import *

_IMPORTED_MODULES = {} # Cache for dynamically imported modules

def load_class(module_path: str, class_name: str) -> Type:
    # Dynamically loads a class from the specified module.
    global _IMPORTED_MODULES
    if module_path not in _IMPORTED_MODULES:
        _IMPORTED_MODULES[module_path] = importlib.import_module(module_path)
    return getattr(_IMPORTED_MODULES[module_path], class_name)

def _marshal(obj: Any) -> Any:
    # Serializes various types of objects to a format suitable for transmission or storage.
    if isinstance(obj, type):
        return {"__module": inspect.getmodule(obj).__name__, "__class": obj.__qualname__}
    elif isinstance(obj, genpy.Message):
        buf = io.BytesIO()
        obj.serialize(buf)
        return {
            "__module": inspect.getmodule(type(obj)).__name__,
            "__class": type(obj).__qualname__,
            "__content": base64.standard_b64encode(buf.getvalue()).decode("utf-8"),
        }
    elif isinstance(obj, ReqType):

```

```

        return obj.value
elif is_dataclass(obj):
    return {k: _marshal(getattr(obj, k)) for k in obj.__dataclass_fields__}
elif isinstance(obj, bytes):
    return base64.standard_b64encode(obj).decode("utf-8")
else:
    return obj

def _unmarshal(src: Any, data_class: Any) -> Any:
    # Constructs an object from its serialized form.
    if src is None:
        return src
    elif data_class is ReqType:
        return ReqType(src)
    elif data_class in (Type[genpy.Message], Type):
        return load_class(src["__module"], src["__class"])
    elif issubclass(data_class, genpy.Message):
        instance = load_class(src["__module"], src["__class"])()
        instance.deserialize(base64.standard_b64decode(src["__content"]))
        return instance
    elif is_dataclass(data_class):
        instance = data_class()
        for k in instance.__dataclass_fields__:
            setattr(instance, k, _unmarshal(src[k], instance.__dataclass_fields__[k].type))
        return instance
    elif data_class is bytes and isinstance(src, str):
        return base64.standard_b64decode(src)
    return src

```

```
def marshal(obj: Any) -> bytes:
    # Converts a Python object to a JSON byte string.
    return json.dumps(_marshal(obj), ensure_ascii=True).encode("utf-8")

def unmarshal(data: bytes, data_class: Any) -> Any:
    # Converts a JSON byte string back to a Python object of the specified type.
    return _unmarshal(json.loads(data), data_class)
```



## **B University's Plagiarism Statement**

This statement is reviewed annually and the definitive statement is in the [University Calendar](#).

The University's degrees and other academic awards are given in recognition of a student's personal achievement. All work submitted by students for assessment is accepted on the understanding that it is the student's own effort.

Plagiarism is defined as the submission or presentation of work, in any form, which is not one's own, without acknowledgement of the sources. Plagiarism includes inappropriate collaboration with others. Special cases of plagiarism can arise from a student using his or her own previous work (termed auto-plagiarism or self-plagiarism). Auto plagiarism includes using work that has already been submitted for assessment at this University or for any other academic award.

The incorporation of material without formal and proper acknowledgement (even with no deliberate intent to cheat) can constitute plagiarism. Work may be considered to be plagiarised if it consists of:

- a direct quotation;
- a close paraphrase;
- an unacknowledged summary of a source;
- direct copying or transcription.

With regard to essays, reports and dissertations, the rule is: if information or ideas are obtained from any source, that source must be acknowledged according to the appropriate convention in that discipline; and any direct quotation must be placed in quotation marks and the source cited immediately. Any failure to acknowledge adequately or to cite properly other sources in submitted work is plagiarism. Under examination conditions, material learnt by rote or close paraphrase will be expected to follow the usual rules of reference citation otherwise it will be considered as plagiarism. Departments should provide guidance on other appropriate use of references in examination conditions.

Plagiarism is considered to be an act of fraudulence and an offence against University discipline. Alleged plagiarism, at whatever stage of a student's studies, whether before or after graduation, will be investigated and dealt with appropriately by the University.

The University reserves the right to use plagiarism detection systems, which may be

externally based, in the interests of improving academic standards when assessing student work.