ÇANKAYA UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

CENG 407

Innovative System Design and Development I

Project Report

Team ID: 202311

AI-based Firefighting Vehicle

Ceren ERDOĞAN-201911401

Efekan KARAKOÇ-201911208

Esra YALÇIN-201911069

Şeyma SERT-201911055

Ömer Faruk YILDIRIM-202011062

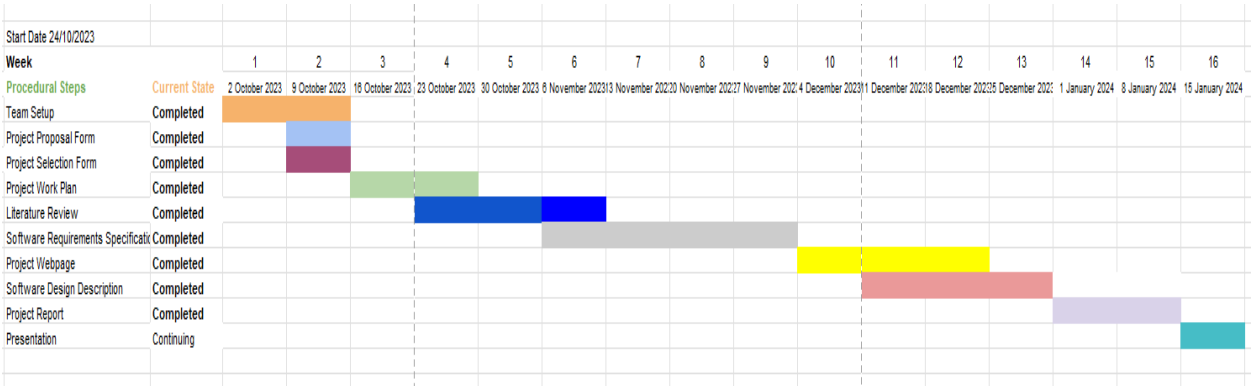
Advisor: Dr. Abdül Kadir GÖRÜR

Table of Contents

Introduction

For the AI-based Firefighting Vehicle project, this document includes a literature review, Software Requirements Specification (SRS), and Software Design Document (SDD). An overview of the body of knowledge regarding the project's advanced features and use cases can be found in the literature review. The SRS section then clarifies the goal, parameters, and overall specifications of the AI-based Firefighting Vehicle. The SDD, which comes after the SRS, carefully explains the p's architecture, features, and design elements from the viewpoint of the user. This document offers a thorough overview of the AI-based Firefighting Vehicle project by narrating the project's overall structure and content step-by-step.

Project Plan



Literature Review

Abstract

The effectiveness of emergency responses has a critical role in providing fast and reliable fire services. Especially in heavy urban traffic, fire trucks must reach fire scenes quickly and safely for life-saving interventions. This study presents a literature review of a simulation project that uses the Unity game engine and applies reinforcement learning (RL) techniques. The simulation focuses on the adaptive and rational response of fire trucks to various dynamics in traffic. It focuses on how these vehicles can effectively navigate in traffic.

Öz

Acil durum müdahalelerinin etkinliği, hızlı ve güvenilir yangın hizmetlerinin sağlanmasında kritik bir role sahiptir. Özellikle yoğun şehir içi trafikte hayat kurtarıcı müdahaleler için itfaiye araçlarının yangın mahalline hızlı ve güvenli bir şekilde ulaşması gerekmektedir. Bu çalışma, Unity oyun motorunu kullanan ve takviyeli öğrenme (RL) tekniklerini uygulayan bir simülasyon projesinin literatür taramasını sunmaktadır. Simülasyon, itfaiye araçlarının trafikteki çeşitli dinamiklere uyarlanabilir ve rasyonel tepkisine odaklanmaktadır. Bu araçların trafikte nasıl etkili bir şekilde gezinebileceğine odaklanılıyor.

1.Introduction

Unity The literature review examines the fundamentals of RL algorithms and their application to the simulation environment, traffic simulations and navigation strategies, used RL algorithms and models, performance evaluation metrics, current challenges, future directions, and potential real-world applications. The main reason why Unity was chosen is that it has the flexibility and power to simulate real-world conditions, thanks to its advanced graphics capacity, comprehensive physics engine, and AI integration tools such as the ML-Agents Toolkit. This simulation environment in Unity can model real-time traffic flow, various road and weather conditions, and emergency scenarios in detail. These features provide an ideal platform to analyze the adaptation and performance of RL algorithms to complex problems.

This literature review reveals the evolution of simulation-based fire truck navigation studies from the past to the present. It provides a comprehensive look at how RL algorithms can be used to provide safe and effective navigation in interaction with traffic. The practical challenges of implementing RL algorithms in the Unity environment and research to overcome these challenges are also discussed. The simulation is designed to improve behavior in accordance with traffic rules and the ability to make quick decisions in emergency scenarios, and the methods and algorithms used to achieve this goal are examined in detail.

2. Unity and Simulation Environments

Thanks to its extensibility and versatility, Unity can also be used as a testing laboratory. The design of test environments is critical for developing and validating RL algorithms. In particular, when creating a fire truck simulation, it offers a flexible platform that can emulate various traffic and environmental conditions.

Testing Environment: In the project, firstly, the test environment will be used for the initial training and validation phases of RL algorithms was developed. This testbed will conduct controllable experiments on traffic scenarios, road conditions, and vehicle behavior models. By creating complex cityscapes and traffic flow models, Unity provides an ideal framework for evaluating the agent's adaptation to different scenarios. The fire truck's agent will encounter increasingly complex tasks in this test environment, starting with simple scenarios. Simple scenarios may involve moving directly on a single road or while remaining in a specific lane. Complex scenarios include making the right turns at intersections, prioritizing other vehicles in traffic, and reacting to sudden obstacles.

City Simulation: After successfully completing the test environment, the second phase of the project, a realistic city simulation, is commissioned. This phase aims to demonstrate potential real-world applications of the project using the advanced features Unity offers. City simulation includes realistic city plans, various vehicle and pedestrian traffic densities, and various emergency scenarios. This simulation environment will test whether the fire truck's agent can internalize traffic rules and apply the navigation strategies it has learned in real-world conditions. For example, it includes scenarios where the agent is randomly launched at different points in the city and is expected to find the fastest and safest path to the fire. Unity's graphics and physics engines provide rich visual feedback and detailed analysis to evaluate the performance of the RL agent. In addition, Unity allows real-time data collection and monitoring features to examine the agent's decision-making processes and the results of these processes in detail.

In conclusion, a Unity-based testbed and city simulation provide the opportunity to evaluate whether RL-based navigation strategies are effective in theory and practical and realistic scenarios. The literature review should include research on how such simulations can be used for agent training and evaluation and guidance for the transition of simulations to real world applications.

3. Fundamentals of Reinforcement Learning:

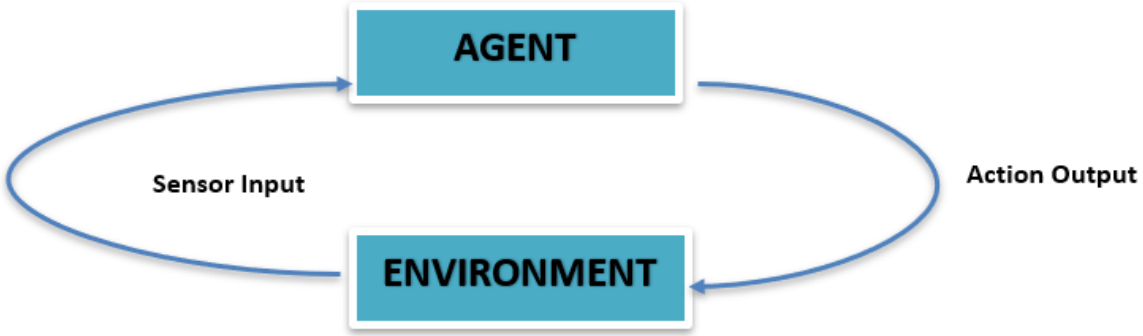
Reinforcement Learning (RL) is a paradigm in the fields of artificial intelligence and machine learning that relies on an agent learning optimal action strategies through trial and error. In this process, the agent acts according to the feedback from its environment, and this feedback is usually in the form of reward or punishment. The main purpose of RL is to develop behavior policies that will maximize the agent's total reward.

Basic Concepts

- **Agent:** An entity that learns and makes decisions.

- **Environment:** The physical or virtual space in which the agent interacts and receives state information.
- **State:** All observable and/or unobservable information the environment has at any moment.
- **Action:** The intervention that the agent can apply to the environment in any situation.
- **Reward:** The feedback the agent receives as a result of an action. Positive rewards indicate correct behavior, and negative rewards indicate incorrect behavior.
- **Policy:** A set of rules that determine what action the agent will choose in a situation.
- **Value Function:** It refers to the expected total reward to be obtained under a policy, starting from a certain situation or situation-action pair.
- **Q-Value:** It represents the value of a specific situation-action pair and is used to find the optimal policy.
- **Learning process :** In the RL process, the agent interacts with the environment and receives a reward after each interaction. Based on these rewards, the agent tries to measure the long-term consequences of its actions and learns the strategy that will achieve the maximum total reward. Some of the basic RL algorithms used in this process are: Monte Carlo Methods: Methods that learn at the end of each episode and try to maximize the average reward.
- **Temporal Difference (TD) Learning:** Methods that are a combination of Monte Carlo and dynamic programming methods and use partial results to update predictions.
- **Q-Learning**: It is a model-free algorithm in which the agent learns the best Q-value regardless of the policy.
- **Deep Q-Network (DQN):** By combining Q-Learning and deep learning methods, it makes it possible to learn from high-dimensional data in complex environments. Policy Gradients: Methods that directly optimize policy parameters and are generally preferred for continuous action areas.
- **Actor-Critic Methods:** It is a two-component method that improves both value function and policy.
- **Application and Fire Truck Scenario:** Reinforcement Learning provides an effective solution to the problem of finding the optimal path for the vehicle to reach the fire in traffic in the fire truck scenario. The agent learns how to reach the fire scene as quickly as possible, obeying city traffic rules and without damaging other vehicles. During the training process, the agent's interactions with other vehicles in traffic in every situation

it encounters are shaped by the rewards and punishments it receives from the environment. This learning takes place in simulation environments designed taking into account complex traffic dynamics and emergency scenarios.



4.Traffic Simulations and RL

Traffic simulations are computer-based systems for analyzing the movement of vehicles and traffic control mechanisms by modeling real-world traffic flow. Reinforcement Learning (RL) is used in traffic simulations to develop strategies that optimize traffic flow, reduce traffic congestion, and clear paths for emergency vehicles.

- **Role of RL in Traffic Simulations**: Traffic simulations are ideal environments for developing and testing RL algorithms because they offer the opportunity to model complex traffic scenarios in real time safely. In these simulations, RL provides a flexible learning mechanism for the agent (fire truck) to learn the dynamics within the traffic and determine the optimal path. In particular, in the traffic way finding problem for emergency vehicles, RL is investigated under the following headings.
- **Control at Traffic Lights**: Algorithms that speed up the passage of emergency vehicles by optimizing traffic lights.
- **Road and Lane Selection**: Policies that determine the fastest route, considering factors such as traffic density and road condition.
- **Agile Maneuvering:** Algorithms that learn to maneuver without hitting other vehicles or violating traffic rules.

- **Simulation Tools and RL Algorithms:** Traffic simulation tools, software such as SUMO (Simulation of Urban Mobility) and VISSIM, are capable of performing traffic simulations on a large scale and offer rich visual experiences by integrating with Unity. RL algorithms are incorporated into these simulation platforms and used for agents to learn complex scenarios in traffic.
- **Using RL for Fire Truck Scenario:** In a fire truck scenario, the RL algorithm might follow these steps: Observation: The agent observes information such as the traffic situation in its environment, nearby vehicles, the status of traffic lights, and road conditions.
- **Action:** The agent selects a movement from a predetermined set of activities (e.g., speed up, slow down, change lanes).
- **Reward:** The agent receives a positive or negative reward due to the action he chooses. Getting to the fire faster receives a positive reward, while breaking traffic rules or causing accidents gets a negative reward.
- **Learning:** The agent updates its action strategies using the reward information it has collected.

This process allows the fire truck to navigate effectively in the complex traffic and emergency scenarios of the simulation. In particular, it aims to proceed quickly and safely in traffic, not to endanger other vehicles and to disrupt the traffic flow as little as possible. Reinforcement Learning has the potential to produce innovative solutions to road-clearing and way-finding problems in traffic simulations, especially for emergency vehicles such as fire trucks. The development of RL algorithms and their integration with traffic simulations can provide valuable insights and strategies for real-world traffic management applications.

5.Fire Truck Navigator Strategies

Navigation strategies of fire trucks focus on determining the shortest and safest route to optimize response time to emergencies. Shortest path algorithms play a fundamental role in solving this type of navigation problem.

- **Shortest Path Algorithms:** Dijkstra's Algorithm: Finds the shortest distance from the starting point to the destination, considering the distance from each node to neighboring

nodes. Dynamically changing the weights according to the traffic situation allows this algorithm to be used effectively in traffic simulations.

- **Algorithm A:** Predicts the least-cost path from the origin to the destination using a heuristic function. Fire trucks often prefer to minimize time and find an effective route without hitting obstacles.
- **Bellman-Ford Algorithm:** It allows finding paths with negative edge weights in weighted graphs but works slower. It can be used when unexpected situations in traffic (for example, the sudden closure of a road) need to be taken into account.
- **Floyd-Warshall Algorithm:** Finds the shortest paths between all pairs of nodes and provides a general solution for complex city maps, but may not be practical in large graphs due to its computational intensity.
- **Dynamic Routing Algorithms:** They constantly update the best route using real-time traffic information. These algorithms can instantly react to dynamic events such as traffic congestion and road works.
- **Simulation and Real-Time Application:** In the Unity simulation environment, the application of the shortest path algorithm for the fire truck takes place on the simulated city map. These algorithms are integrated with Unity's physics engine and map data structures, serving as real-time navigation and decision-making mechanisms.

Particularly in traffic simulations, the vehicle's environmental sensing capabilities (e.g., a virtual fire truck equipped with cameras and sensors) and artificial intelligence control points that manage traffic flow are critical to improving the algorithm's accuracy. Shortest route finding for fire trucks is a critical factor in reducing emergency response times. Applying these algorithms in the Unity simulation environment can mimic real-world conditions and effectively model the challenges and response strategies that fire department responders may face. Thus, it increases the operational efficiency and effectiveness of the fire truck both in simulation and in real-world applications.

6. RL Algorithms and Models Used

In this project, we will try various Reinforcement Learning algorithms to enable the fire truck to navigate traffic effectively. These algorithms learn optimal decisions in complex traffic scenarios and adapt over time.

Basic RL Algorithms

- **Q-Learning:** It is one of the oldest and most common RL algorithms. It aims to maximize the reward obtained from actions taken from a certain situation. For the fire truck, the table of Qvalues is updated for each condition (vehicle position, traffic situation, etc.) and action (change of direction, speed adjustment, etc.).
- **Deep Q-Networks (DQN):** Adding deep learning techniques to the classical Q-Learning algorithm increases its applicability for large state spaces and complex tasks. It plays an essential role in helping a fire truck learn the complex traffic dynamics.
- **Policy Gradients:** Optimizes a strategy that determines what action to take in each situation by making direct improvements over learned policies. It is helpful for fire trucks to learn traffic rules and yield scenarios.
- **Actor-Critic**: It consists of two separate components: Actor and Critic. While the Actor determines the actions to be taken, the Critic estimates the value of these actions. This approach helps the fire truck not ignore its long-term goals while making instant decisions in traffic.
- **Proximal Policy Optimization (PPO):** It is an advanced algorithm from the Policy Gradients family that increases the stability and reliability of the algorithm. It makes it easier for the fire truck to adapt to sudden changes in traffic quickly.

- **Twin Delayed Deep Deterministic Policy Gradients (TD3):** It is an algorithm that combines the advantages of DQN and Actor-Critic methods and provides more stable learning by reducing noise in action selection. It is especially useful in situations that require sudden maneuvers in traffic.
- **Model-Based RL Approaches:** Model-based RL algorithms predict the consequences of actions using a model of the environment. They can thus provide learning with fewer trials. This is important for practical applications in real traffic because every wrong action can have serious consequences.
- **Simulation Integration:** Implementing these algorithms in the Unity environment allows the fire truck to experience realistic traffic conditions in a virtual environment and develop optimal decision making mechanisms. Tools such as Unity's ML-Agents Toolkit provide the necessary interface and infrastructure for the integration and optimization of RL algorithms. RL algorithms have the potential to improve the performance of fire trucks in traffic. Choosing the right algorithm and training it effectively is the key to successfully applying the strategies learned in the simulation environment in the real world. These algorithms are critical for developing robust and flexible decision-making capabilities for various situations to be encountered in traffic scenarios.

7. Performance Evaluation and Metrics

A.Performance Metrics:

- **On-Time Arrival Rate (TVO):** The percentage of the fire truck arriving at the fire scene on time is the most direct indicator of success.
- **Average Travel Time (AVT):** The average time it takes to reach a fire is a measure of the efficiency of traffic flow.
- **Number of Collisions:** Collisions between vehicles and the environment are an important metric in terms of safety.
- **Compliance Rate with Rules:** The rate of compliance with traffic rules shows how much the vehicle complies with traffic laws.
- **Ride Smoothness (JS):** Sudden accelerations, braking, and lane changes reflect the ride quality and impact the vehicle has on other drivers around it.

- **Response Time:** The time it takes for the fire truck to reach the fire scene is one of the most critical performance metrics. A short response time ensures quick and effective fire response.
- **Priority Situations:** The fire truck's ability to prioritize emergency vehicles can increase the speed of reaching the fire scene. This metric is used to evaluate the vehicle's interaction with other emergency vehicles.
- **Optimal Route Use**: The fire truck's ability to choose the shortest and fastest route in traffic increases its effectiveness. This metric evaluates the distance of the vehicle's chosen route to the fire scene.
- **Disrupting Traffic Flow:** How much the fire truck disrupts traffic flow is an important factor for other drivers. The ability to not disrupt traffic flow can increase the efficiency of urban traffic.
- **Total Rewards:** The amount of rewards collected by RL algorithms during the learning process is a metric that reflects the performance of the agent. A higher reward total indicates better performance.

B. Simulation Based Evaluation:

- **Scenario Replay:** Specific traffic and fire scenarios can be played multiple times in the simulation to evaluate the model's decision-making ability.
- **Multi-Environment Tests:** The model should be tested in various conditions, such as different traffic density and different city maps.
- **Realism:** How well the simulation reflects real-world conditions in terms of traffic flow, driver behavior and emergency procedures should be evaluated.

C. Statistical Analysis

Decision Distribution Analysis: The statistical distribution of which decisions the fire truck prefers in which situations is valuable for understanding policy. It is especially important to examine under what conditions decisions are made, such as complying with traffic rules or reaching the fire zone quickly, to understand the priority order of decisions. This analysis will help us better understand the fire truck's behavior and make improvements where necessary. Confidence Intervals and Variance: Statistical confidence intervals and variance of performance

metrics indicate the consistency and reliability of the model. This statistical analysis helps us determine how much the simulation results are affected by random variations. Narrower confidence intervals and lower variance indicate that the model is more stable and reliable. This helps better predict the real-world performance of the fire truck.

D. Comparative Analysis

- **Benchmarking:** The model's performance should be compared with industry standards or results from similar studies. These comparisons help us identify areas where the developed model can be improved. It is also used to measure the ability to perform above or below industry standards. Benchmark results help us better understand the competitiveness and feasibility of the model.
- **A/B Tests:** Comparative analysis of results obtained using different RL algorithms or hyperparameter settings. These tests help us determine where different model configurations perform better. For example, one algorithm may have reached the fire zone more quickly than others, while another algorithm may have followed traffic rules more strictly. This type of analysis helps us understand how the model should be tuned to achieve the best performance.

E. Feedback from Human Experts

- **Expert Evaluation:** Fire experts and traffic managers can review the simulation results and evaluate them for realism and feasibility. Additionally, these experts can provide important insight into aligning fire truck behavior with real-world fire crews. These experts can offer important perspectives on how the fire truck can be better integrated into firefighting operations.
- **Surveys and Snapshots:** Surveys and snapshots from drivers in real traffic situations can be used to understand the impact of the model. Surveys and interviews can be conducted to help drivers evaluate how they perceive the fire truck's behavior in traffic and the impact of these behaviors on their safety. This can be an important source of feedback on how the model can better serve the needs of society. A comprehensive evaluation of the fire truck's performance reveals the strengths and weaknesses of the developed model and enables necessary improvements to be made before

implementation in the real world. The metrics measured must accurately reflect the vehicle's effectiveness in traffic and its ability to react in emergency situations.

8. Challenges, Future Directions and Applications

A. Current Challenges

- **Simulation Realism:** Virtual environments created in Unity do not fully reflect real-world complexity and unpredictability, which can make the transfer learning process difficult. Computational Resources: High-resolution simulations and complex RL algorithms require significant processing power and can occasionally lead to resource constraints. Sensing and Sensor Integration: Integrating precise sensor data that mimics real-world conditions into simulation remains a significant engineering challenge.
- **Algorithm Stability:** The stability and reliability of RL algorithms are critical, especially in dynamic and constantly changing traffic environments.

B. Potential Directions for Future Research:

- **Transfer Learning:** Developing methods that will enable successful transfer of the model from the simulation to the real world.
- **Multi-Agent Systems:** Researching strategies that will enable multiple fire trucks to act in a coordinated manner.
- **Deep Learning Integration:** Combining RL algorithms with deep learning techniques can improve perception and decision-making mechanisms.

C. Transfer Potential to Real World Applications:

- **Emergency Management:** Integration of RL-based solutions in real-time traffic management and emergency response strategies.
- **Intelligent Transportation Systems:** Using RL algorithms to optimize traffic flow and improve safety.

D. Technological Advances:

- **Artificial Intelligence and Machine Learning:** Advances in AI models will increase the applicability of RL for traffic navigation. Computer Hardware: Advanced hardware such as GPUs and dedicated processors will speed up simulation and model training processes.
- **Shortest Path Algorithms: Dijkstra's Algorithm:** Finds the shortest distance from the starting point to the destination, taking into account the distance from each node to neighboring nodes. Dynamically changing the weights according to the traffic situation allows this algorithm to be used effectively in traffic simulations.
- **Algorithm A:** Predicts the least-cost path from the origin to the destination using a heuristic function. It is often preferred for fire trucks to minimize time and find an effective route without hitting obstacles.
- **Bellman-Ford Algorithm:** It allows finding paths with negative edge weights in weighted graphs, but it works slower. It can be used when unexpected situations in traffic (for example, the sudden closure of a road) need to be taken into account.
- **Floyd-Warshall Algorithm:** Finds the shortest paths between all pairs of nodes and provides a general solution for complex city maps, but may not be practical in large graphs due to its computational intensity.
- **Dynamic Routing Algorithms:** They constantly update the best route using real-time traffic information. These algorithms can instantly react to dynamic events such as traffic congestion and road works.

9. Conclusion

This review guides future research on how RL algorithms can be applied in simulation and real-world scenarios in the Unity environment and provides a bibliography of essential studies in the field.

Software Requirement Specification (SRS)

1. Introduction

1.1 Purpose of this Document

This SRS document is a detailed guide for developing an AI-driven fire engine system, using Unity and ROS 2. It provides a comprehensive framework, detailing design, functionality, AI integration, and the application of advanced reinforcement learning techniques. The document serves as an essential blueprint, guiding each stage from conceptualization to implementation, and ensuring clarity and accuracy in the development process. It's a pivotal reference that outlines system specifications, design considerations, functional and non-functional requirements, and testing methodologies, aiming to align the development team with the project's core objectives and ensuring a cohesive approach to integrating advanced simulation technologies with AI algorithms.

1.2 Scope of The Project

This project is centered around creating an advanced AI-driven fire engine control system in the Unity environment, utilizing reinforcement learning for effective autonomous decision-making. The focus is on training the system in a simulated environment to respond to emergency scenarios with precision and safety. Unity offers a realistic simulation platform, while ROS 2 is used for processing vital Lidar data, enhancing the AI's perception capabilities. The project also involves generating and visualizing detailed point cloud data through ROS 2, adding to the realism and effectiveness of the AI's training. Developed on the Ubuntu operating system, the

project combines

AI autonomous control strategies with a high-fidelity simulation environment, aiming to significantly improve emergency response in urban settings and advance autonomous vehicle technology.

2. General Description

The project involves developing an AI-driven fire engine control system in Unity, focusing on autonomous navigation in emergency scenarios. The simulation environment in Unity will create detailed settings for the AI to learn and adapt. Key to this system is reinforcement learning, enabling the AI to navigate and respond to various emergency situations effectively. Integration with ROS 2 will facilitate the processing and visualization of Lidar-generated point cloud data, enhancing the AI's environmental perception.

User interfaces will be included for interaction, performance observation, and training scenario adjustments. The project, developed on Ubuntu, aims to establish strategies applicable to real-world emergency response, ensuring seamless integration of Unity and ROS 2 for effective simulation and data handling.

2.1 Glossary

Term	Definition
AI (Artificial Intelligence)	A branch of computer science focused on creating systems that can perform tasks that typically require human intelligence.
RL (Reinforcement Learning)	An area of machine learning where an AI agent learns to make decisions by performing actions and receiving feedback from the environment.
Unity	A cross-platform game engine widely used for developing interactive 3D applications, including simulations and games.
ROS (Robot Operating System)	An open-source robotics middleware suite, used for building complex robotic applications.
Lidar (Light Detection and Ranging)	A remote sensing method that uses light in the form of a pulsed laser to measure distances.
Point Cloud	A set of data points in space, often produced by 3D scanners like Lidar, representing a physical object or space.
Ubuntu	An open-source operating system based on Linux, known for its stability and widespread use in various computing environments.
User	An individual or a system that interacts with the software application, in this context, the personnel or systems engaging with the fire engine navigation system.
Actor	In software and system engineering, an actor represents an entity that interacts with the system (e.g., a user, a device, or an external system).
Agent	In the context of AI and machine learning, particularly in reinforcement learning, an agent is an entity that makes decisions based on observations from the environment, seeking to achieve certain goals.
Python	A high-level, interpreted programming language known for its simplicity and readability, widely used in software development, including AI and machine learning applications.

2.2 User Characteristics

- **Technical Proficiency**: Users are expected to have a basic understanding of AI and machine learning concepts, particularly in reinforcement learning, to effectively interact with the system.
- **Familiarity with Simulation Tools:** Users should be comfortable working with simulation environments, especially those familiar with Unity for real-time simulations.
- **Understanding of Robotics Concepts:** Knowledge of robotics, particularly familiarity with ROS (Robot Operating System), is beneficial for users, especially when dealing with point cloud data and Lidar information.
- **Problem-Solving Skills:** Users should possess strong analytical and problem-solving skills to interpret the system's outputs and make informed decisions during simulations.

- **Adaptability:** As the system involves a blend of advanced technologies, users should be adaptable and open to learning new tools and interfaces.

2.3 Overview of Functional Requirements

- **AI-Driven Autonomous Navigation**: The system must be able to autonomously navigate a fire truck through urban environments using reinforcement learning techniques.
- **Scalability and Upgradability**: Design the system to be scalable for different urban environments and upgradable as new AI techniques and simulation technologies evolve.
- **Reliable Communication Protocols**: Establish reliable communication protocols between the Unity environment, ROS 2, and any other integrated systems or sensors, ensuring seamless data transfer and synchronization.

2.4 General Constraints and Assumptions

- **Constraints:** Hardware Limitations: The system's performance is subject to the capabilities of the hardware used for simulation and data processing, particularly for handling complex AI algorithms and high-resolution simulations in Unity.
- **Software Compatibility**: The integration of Unity with ROS 2 and other software tools must be compatible with the chosen Ubuntu operating system. Real-Time Processing: Achieving real-time data processing and decision-making within the simulation can be constrained by computational power and system optimization.
- **Scalability Limits:** The scalability of the system may be limited by technological and resource constraints, affecting its adaptability to different urban environments or more complex scenarios.

Assumptions:

- **User Proficiency**: It is assumed that users interacting with the system will have a basic understanding of AI, Unity, and ROS 2.

- **Availability of Data**: The system assumes the availability of necessary data, including traffic patterns and Lidar sensor data, for effective simulation and training.
- **Consistency in Simulation Environment:** The simulation environment in Unity is assumed to consistently replicate real-world traffic conditions and scenarios for effective AI training.

3.Specific Requirements

3.1 Interface Requirements

3.1.1 User Interface

- **Real-Time Data** Visualization: Implement features for real-time visualization of the simulation, including the movement of the fire truck and Lidar-generated point cloud data.
- **Performance Metrics Dashboard:** A dashboard displaying key performance metrics such as response time, route efficiency, and compliance with neural networks and algorithms, allowing users to assess the AI's performance.

3.1.2 Hardware Interface

- **Computing Power:** Adequate computing power is required to handle the AI algorithms, Unity simulations, and ROS 2 data processing, which may involve high-performance CPUs and GPUs.
- **Sensor Integration:** Compatibility with virtual and, potentially, real Lidar sensors for gathering environmental data within the simulation and in real-world testing scenarios.
- **Network Capabilities:** Adequate network hardware to ensure stable and fast data transfer, especially important for real-time simulations and cloud-based data processing.
- **Peripheral Support:** Support for various peripherals such as monitors for visualization, input devices (keyboard, mouse).

3.1.3 Software Interface

- **Unity Engine Compatibility:** Ensure compatibility with the latest or most stable version of the Unity engine for creating and running the simulations.

- **ROS 2 Integration:** Seamless integration with ROS 2 for handling Lidar data and point cloud processing.
- **AI and Machine Learning Libraries:** Incorporation of necessary AI and machine learning libraries, particularly those relevant to reinforcement learning.
- **Operating System Support:** Ensure system compatibility with the Ubuntu operating system, including necessary drivers and software dependencies.

3.1.4 Communication Interfaces

- **Data Transfer Protocols**: Establish robust data transfer protocols for the seamless flow of information between Unity, ROS 2, and any other integrated systems.
- **API Integration:** Develop or integrate APIs (Application Programming Interfaces) for efficient communication between different software components and external data sources.
- **Security Protocols:** Implement secure communication protocols to protect data integrity and privacy, especially when transferring data over networks.
- **Real-Time Communication Support:** Ensure that communication interfaces can handle real-time data exchange with minimal latency for effective simulation and AI performance monitoring.

3.2 Detailed Description of Functional Requirements

3.2.1 Template for Describing Functional Requirements

For each functional requirement of the AI-driven fire engine navigation system, the following template will be used:

- **Requirement Name**: A clear and concise name for the requirement.
- **Description:** A detailed description of the requirement, explaining what it entails and its purpose within the system.
- **Priority:** The level of importance of the requirement (e.g., High, Medium, Low).
- **Inputs:** Any inputs required for the requirement to function (e.g., data types, user actions).
- **Processing Steps:** A step-by-step breakdown of how the requirement will be processed within the system.

- **Outputs:** The expected outputs or results from the requirement once processed (e.g., visual data, decision outcomes).
- **Dependencies:** Any other system components or requirements that this requirement depends on to function properly.

3.3 Non-Functional Requirements

- **Performance:** The system must process data and perform simulations in real-time, maintaining high responsiveness and accuracy in AI-driven navigation.
- **Reliability:** The system should function consistently under varying conditions, providing stable and dependable AI navigation capabilities.
- **Usability:** The user interface and overall system design should be user-friendly, allowing easy navigation and interaction for users of varying technical backgrounds.
- **Scalability:** The system should be scalable, capable of handling increased loads, and adaptable to various urban environments and emergency scenarios.
- **Maintainability:** The system should be easy to maintain and update, with clear documentation and modular design for future enhancements.
- **Security:** Data security measures must be in place to protect sensitive information processed by the system, including user data and simulation results.
- **Compatibility:** The system must be compatible with the necessary hardware and software environments, ensuring seamless integration and operation.
- **Portability:** The design should allow for the potential portability of AI strategies from the simulated environment to real-world applications.
- **Compliance:** The system should comply with relevant legal and ethical standards, particularly concerning data handling and AI deployments.
- **Accessibility:** The system should be accessible to users with different abilities, incorporating design considerations for broader user inclusivity.

4.Analysis-UML

4.1.1 Use Cases Diagrams

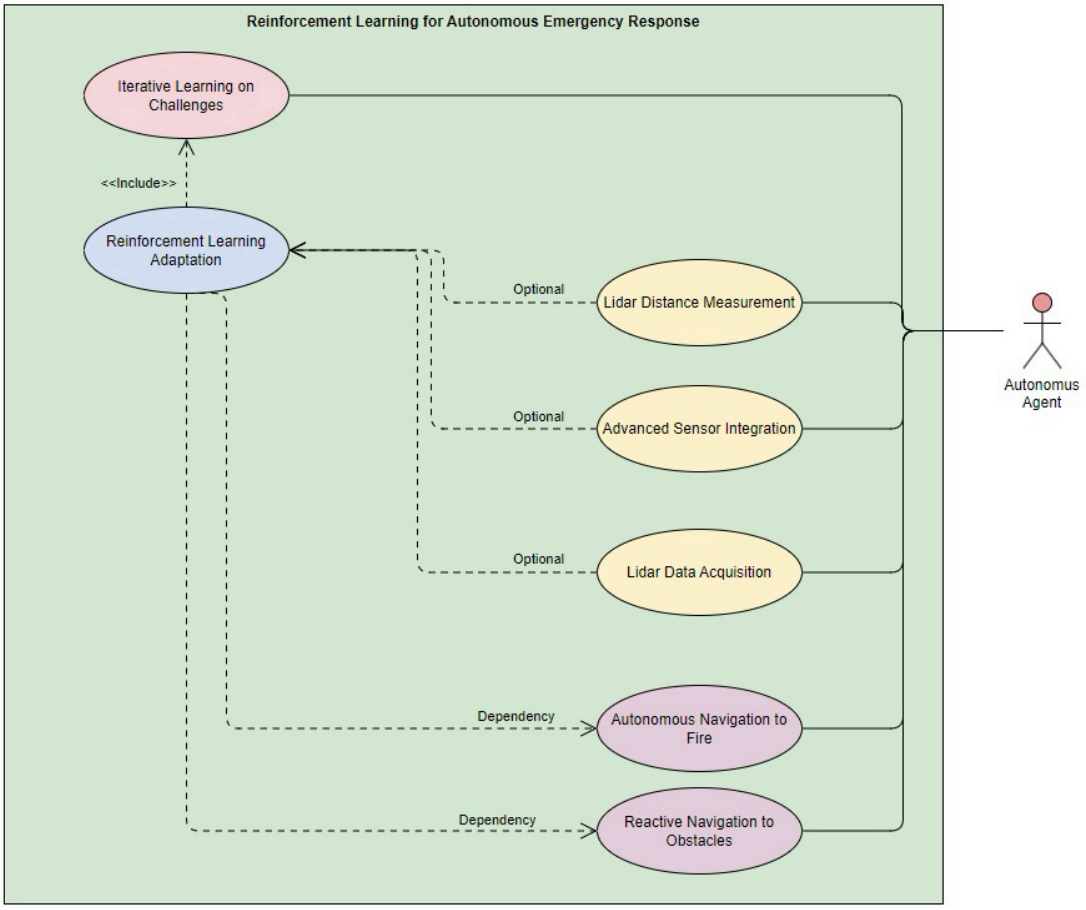


Figure 1. Use Case Diagram

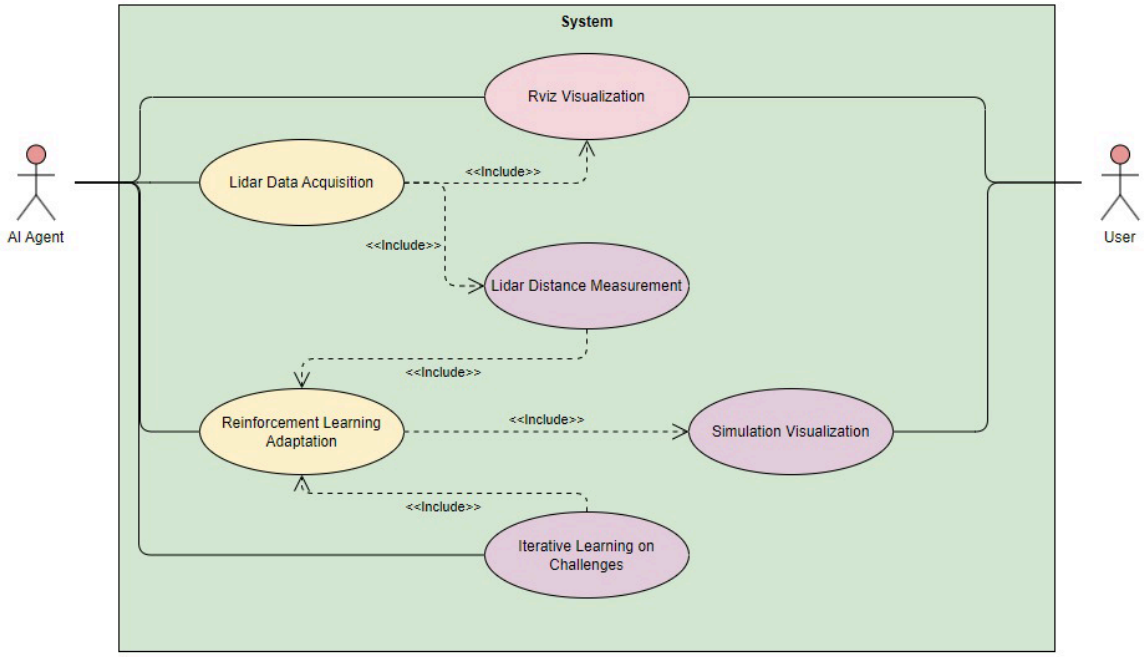


Figure 2. Use Case Diagram

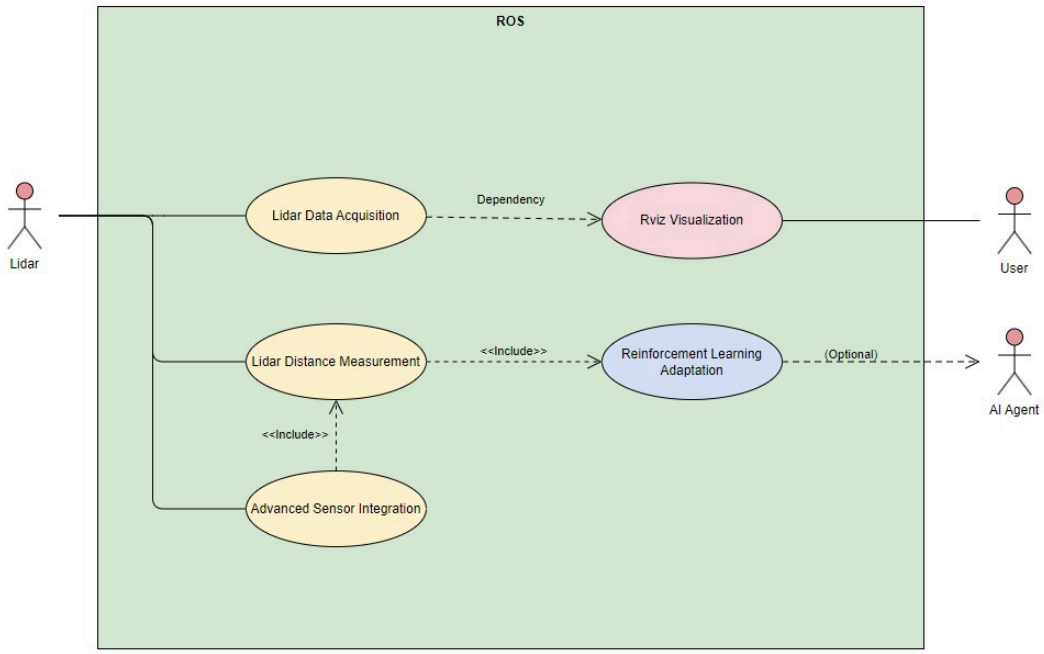


Figure 3. Use Case Diagram

4.1.2 Describe Use Cases

Field	Description
Use Case Number	Use Case 1
Use Case Name	Autonomous Navigation to Fire
Actor	Autonomous Agent
Description	The AI agent autonomously navigates to the location of a fire within the simulated city environment, using reinforcement learning to choose the best path without deviating from its lane and adhering to traffic rules.
Precondition	Simulation running, fire location identified.
Scenario	Agent receives fire alert, calculates optimal route, navigates to location.
Postcondition	Agent arrives at fire location.
Exceptions	Route blocked, sensor malfunction.

Field	Description
Use Case Number	Use Case 2
Use Case Name	Lidar Data Acquisition for City Modeling
Actor	Autonomous Agent with Lidar
Description	While navigating, the agent uses its Lidar sensor to collect point cloud data, which is then used to model the city within the simulation. This data collection is critical for demonstrating the feasibility of RL techniques in real-world applications.
Precondition	Lidar operationaln identified.
Scenario	Agent navigates city, Lidar scans environment, data processed for 3D city model (Point Clouds).
Postcondition	3D city model updated.
Exceptions	Data corruption, Lidar failure.

Field	Description
Use Case Number	Use Case 3
Use Case Name	Reinforcement Learning Adaptation
Actor	AI System
Description	The system utilizes reinforcement learning to adapt the agent's navigation strategies in response to dynamic urban layouts, improving its decision-making over time.
Precondition	Initial learning parameters set
Scenario	Agent learns and adapts behavior
Postcondition	Improved navigation efficiency
Exceptions	Learning algorithm converges poorly.

Field	Description
Use Case Number	Use Case 4
Use Case Name	Rviz Visualization of Point Cloud
Actor	User, System
Description	Users can visualize the point cloud data generated by the agent's Lidar sensor in Rviz running on ROS 2 within an Ubuntu environment, allowing for detailed observation and analysis of the simulated city's structure.
Precondition	Rviz running on Ubuntu with ROS 2
Scenario	User requests point cloud display, Rviz renders Lidar data
Postcondition	User views and analyzes point cloud
Exceptions	Incompatible data format, visualization errors

Field	Description
Use Case Number	Use Case 5
Use Case Name	Emergency Route Optimization
Actor	Autonomous Agent
Description	This use case finds the most efficient path to the emergency by observing the various paths of the AI system.
Precondition	Fire reported
Scenario	Agent computes fastest safe route
Postcondition	Optimal path taken, fire reached promptly
Exceptions	Unexpected road closures

Field	Description
Use Case Number	Use Case 6
Use Case Name	Reactive Navigation to Sudden Obstacles
Actor	Autonomous Agent
Description	The AI agent must dynamically adjust its path in real-time upon encountering unexpected obstacles to maintain safety and continue towards its destination.
Precondition	Agent en route to fire
Scenario	Sudden obstacle detected, Agent recalculates route dynamically
Postcondition	Obstacle avoided, route readjusted
Exceptions	Sensor blind spots, delayed obstacle detection

Field	Description
Use Case Number	Use Case 7
Use Case Name	Integration with Real-World Mapping Systems
Actor	Autonomous Agent
Description	Testing and ensuring that the simulated environment can integrate with real geographic data to accurately reflect real-world locations and enhance training fidelity.
Precondition	Simulated environment ready
Scenario	Integrator tests compatibility with real-world mapping systems, aligns simulation data with real geography
Postcondition	Simulation accurately reflects real-world locations
Exceptions	Misalignment issues, geographic data discrepancies

Field	Description
Use Case Number	Use Case 8
Use Case Name	Lidar-Based Distance Measurement for Emergency Navigation
Actor	Autonomous Agent
Description	The AI agent uses its Lidar sensor to measure distances and detect the proximity of objects while navigating to an emergency. This functionality is crucial for avoiding collisions and determining the best path in real-time.
Precondition	The agent is activated, and the Lidar sensor is operational.
Scenario	1. The agent receives a navigation request. 2. Lidar sensor initiates scanning. 3. Distance and proximity data are collected. 4. The agent processes this data to navigate safely.
Postcondition	The agent successfully avoids obstacles and reaches the destination without incident.
Exceptions	Lidar sensor failure, inaccurate data due to environmental factors (e.g., smoke, heavy rain).

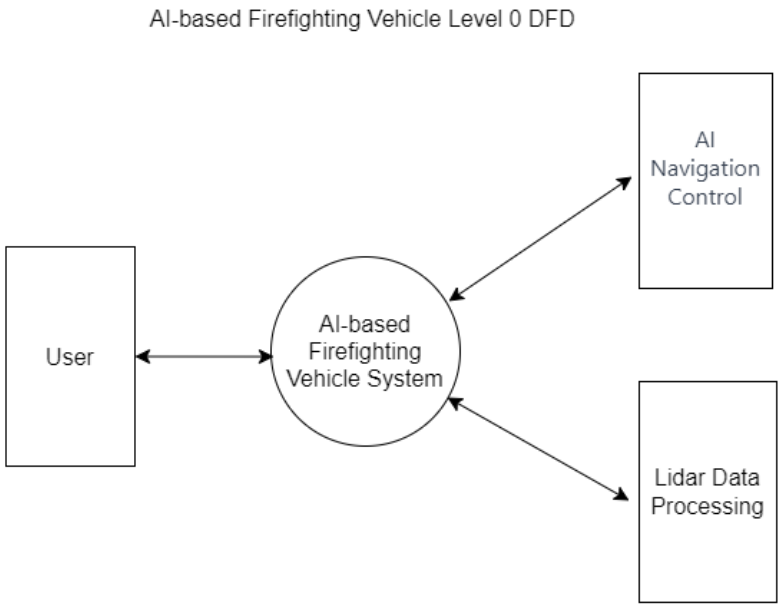
Field	Description
Use Case Number	Use Case 9
Use Case Name	Advanced Sensor Integration for Obstacle Detection and Analysis
Actor	AI Agent
Description	This scenario details the AI agent's process of detecting environmental obstacles using advanced sensors (e.g., Lidar, infrared) and calculating a safe path based on this information. The agent processes environmental data in real-time and uses a complex decision tree to determine the optimal route, including detailed analysis of the size, location, and potential impact of obstacles.
Precondition	AI agent active, sensors operational.
Scenario	1. Agent collects sensor data. 2. Data processed through predetermined algorithms. 3. Optimal route calculated and applied.
Postcondition	Agent safely navigates around detected obstacles.
Exceptions	Sensor malfunctions, data processing errors.

Field	Description
Use Case Number	Use Case 10
Use Case Name	Iterative Learning on Encountered Challenges
Actor	AI Agent
Description	The AI agent applies an iterative learning process to each challenge encountered. In each new scenario, the agent improves its decisions based on past experiences using its algorithm (e.g., Q-learning or Deep Q Networks). This process involves analyzing the relationship between each decision and its outcome and optimizing its strategy for similar future situations.
Precondition	AI agent encounters various challenges.
Scenario	1. Agent records encountered challenges. 2. Learning process occurs after each event. 3. Agent improves its strategy for future similar situations.
Postcondition	Enhanced decision-making ability of the AI agent.
Exceptions	Situations where the algorithm falls short, insufficient training data.

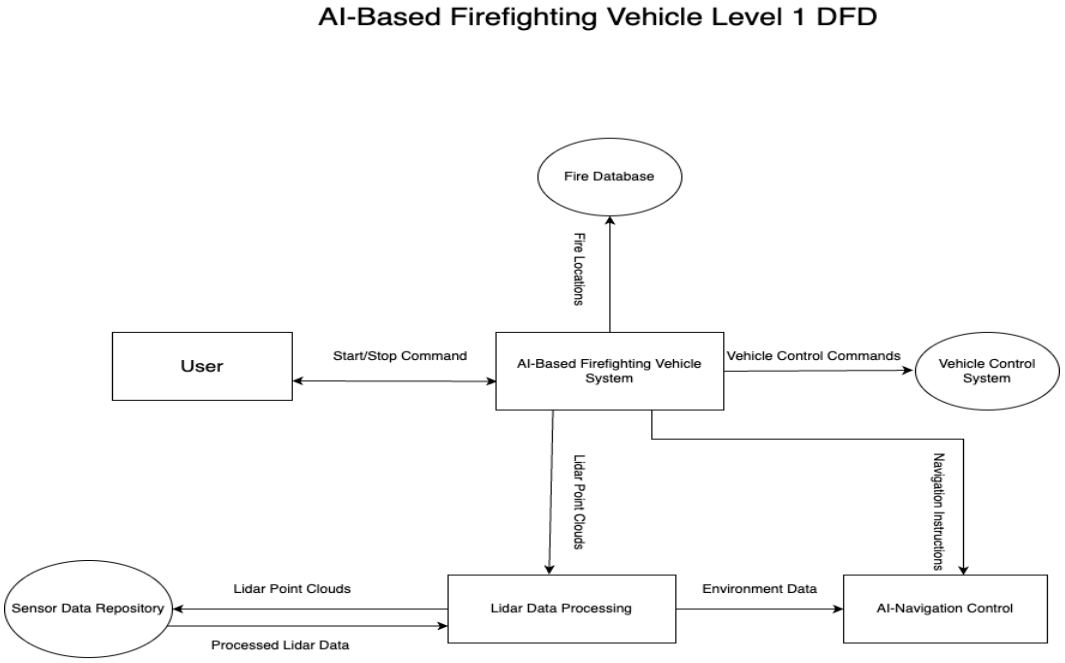
4.2 Functional Modeling (DFD)

4.2.1 DFD Diagrams (Level-0, Level-1)

Level-0 DFD Diagram



Level-1 DFD Diagram



4.2.2 Data Dictionary

1. Terms and Abbreviations:

- **AI (Artificial Intelligence):** A field of computer science focused on creating systems that can perform tasks requiring human intelligence.

- **RL (Reinforcement Learning):** A machine learning area where an AI agent learns to make decisions by interacting with its environment.

- **Unity:** A cross-platform game engine widely used for developing interactive 3D applications, simulations, and games.

- **ROS (Robot Operating System):** An open-source robotics middleware suite used for building complex robotic applications.

2. Data Elements and Descriptions:

- **Lidar (Light Detection and Ranging):** A remote sensing method used for distance measurement. In the project, it's used for collecting environmental data with 3D scanners.

- **Point Cloud:** A set of data points in space, often produced by 3D scanners like Lidar, representing a physical object or space.

- **AI Agent:** An entity controlled by artificial intelligence in your project.

- **Ubuntu:** An operating system; your project will be developed on this OS.

- **User Interface:** The interface allowing users to interact with the system.

3. Data Stores and Descriptions:

- **Sensor Data Repository:** Stores raw and processed sensor data such as Lidar data.

- **Fire Database:** Stores data related to fires, such as fire locations.

4. Workflows and Descriptions:

- **AI-Driven Autonomous Navigation:** The system's ability to autonomously navigate to a fire location.

5.Conclusion

The conclusion of this project on AI-driven fire engine navigation underscores its comprehensive development and future potential. By integrating Unity and ROS 2, and utilizing Lidar sensor point cloud data, the project lays a foundation for effective autonomous navigation in emergency scenarios. This integration facilitates the transition from theoretical models to practical applications. The application of AI and reinforcement learning techniques, especially in emergency response, paves the way for groundbreaking advancements. The project thoroughly addresses both functional and non-functional requirements, optimizing performance, reliability, and scalability. This endeavor represents a significant stride in autonomous navigation and emergency management, establishing a robust foundation for future technological advancements.

Software Design Description (SDD)

1.Introduction

This section serves as the gateway to the Software Design Document (SDD) for our AI-driven fire engine project. It elaborates on the context and the high-level view of the project, providing an overview of the document's layout and summarizing the primary goals, scope, and objectives of the project. This introduction sets the stage for a detailed exploration of the system's design.

1.1 Purpose

The purpose of this document is to meticulously outline the design and architectural blueprint of the AI-driven fire engine control system. It acts as a pivotal guide for the development team, stakeholders, and any involved parties, providing a clear understanding of the system's design philosophy, architectural decisions, and the integration of critical technologies such as Unity and ROS 2.

1.2 Definitions

This section will provide clear definitions for key technical terms and concepts that are crucial for understanding the SDD. It will include terminology related to artificial intelligence, fire engine operations, emergency response scenarios, the Unity game engine, ROS 2 middleware, Lidar data processing, and other relevant technical jargon. This glossary aims to ensure clarity and a common understanding of the terms used throughout the document.

2.System Overview

This section will present an expansive overview of the AI-driven fire engine system. It will detail the system’s composition, highlighting key components such as the AI and machine learning modules, the simulation environment created in Unity, and the data processing and visualization mechanisms facilitated by ROS 2. The overview will also address how these individual elements synergistically interact within the framework to provide a sophisticated solution for autonomous fire engine navigation and emergency response scenarios.

3. System Design

The system design section will offer an in-depth analysis of the architectural choices and design intricacies of the system. This includes a thorough examination of the AI model's structure, focusing on the intricacies of the reinforcement learning algorithms, the integration and processing of Lidar data, and the implementation of these technologies within the simulated environment. Additional focus will be given to the user interface design, ensuring a seamless and intuitive interaction for users. The system's data flow, integration methods, and overall architecture will be dissected to highlight the system's robustness, efficiency, and scalability in real-world applications.

3.1 Architectural Design

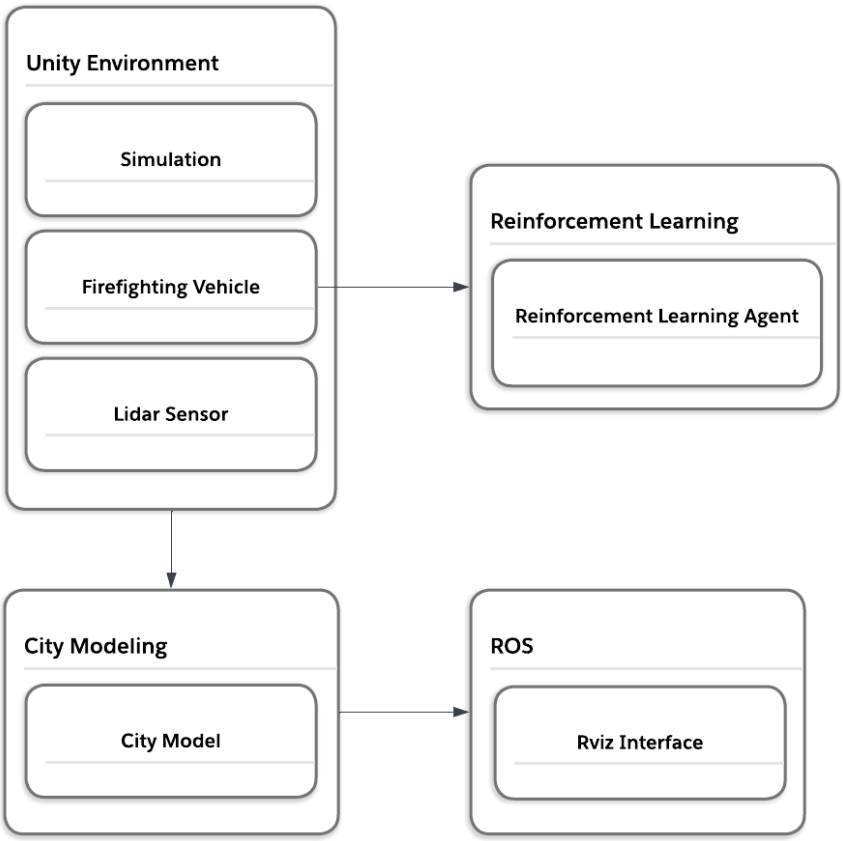
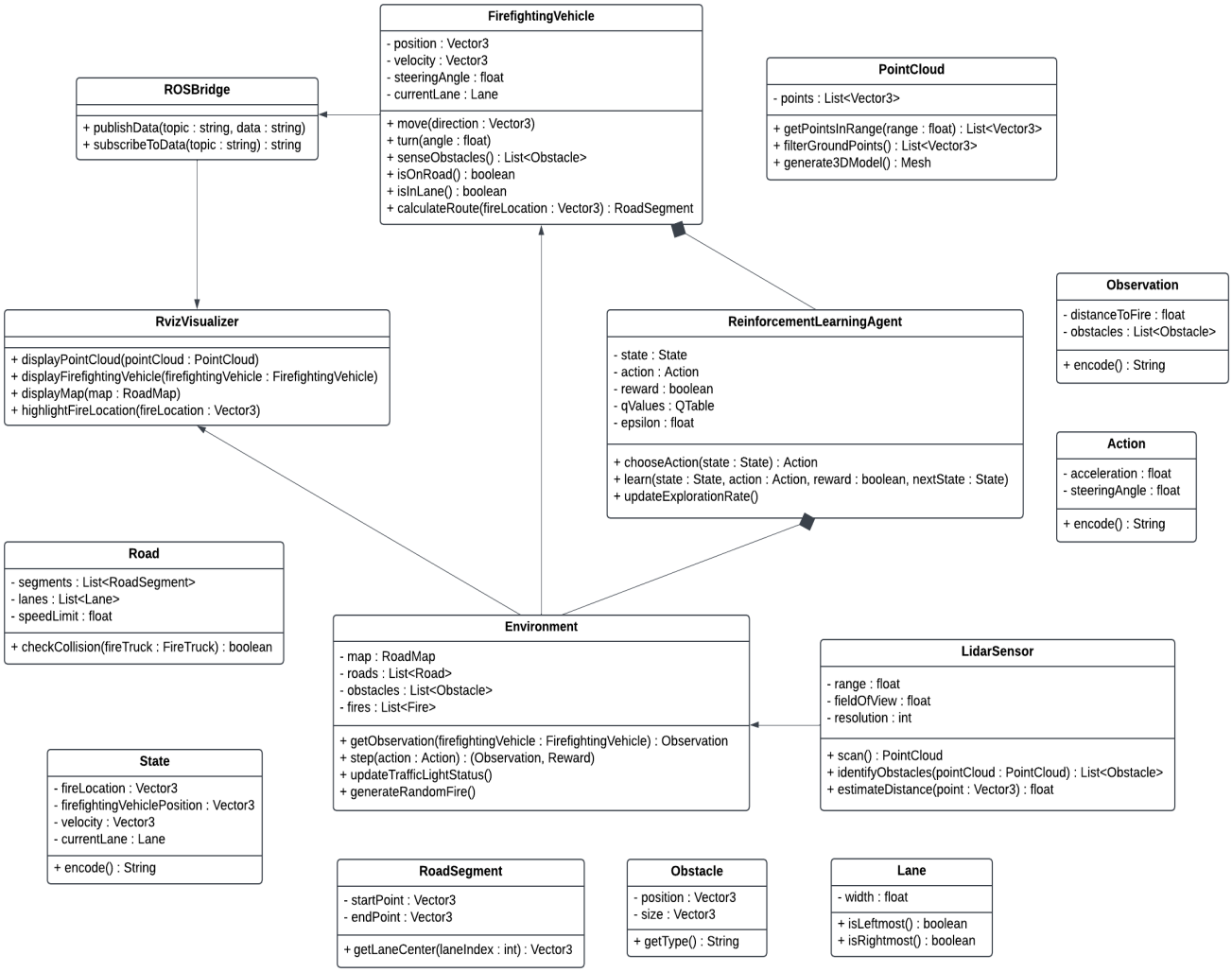


Figure 4. Architecture Design

3.2 Decomposition Description: Class Diagram

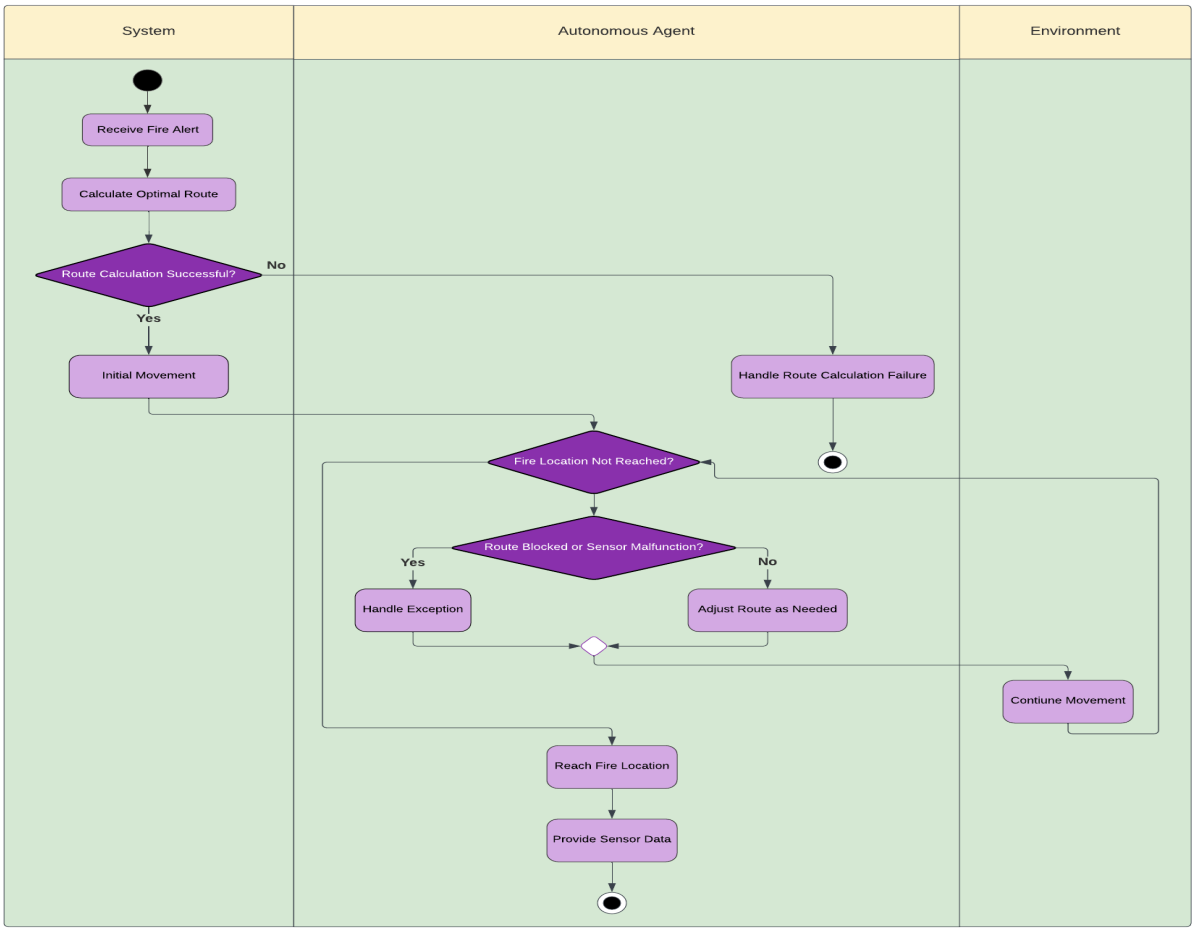
Figure 5. UML Class Diagram



3.3 System Modeling

3.3.1 Autonomous Navigation to Fire

Figure 6. Activity Diagram of Autonomous Navigation to Fire



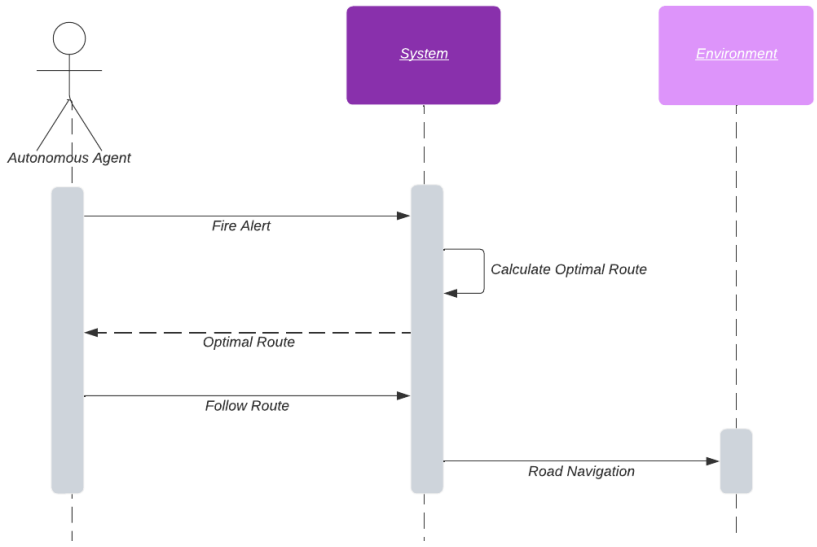


Figure 7. Sequence Diagram of Autonomous Navigation to Fire

3.3.2 Lidar Data Acquisition for City Modeling

Figure 8. Activity Diagram of Lidar Data Acquisition for City Modeling

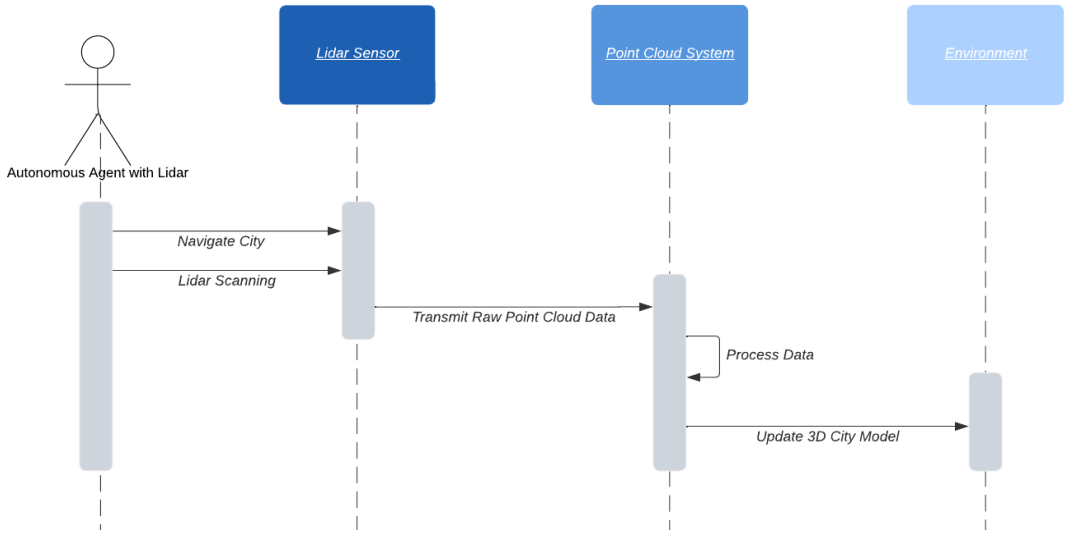
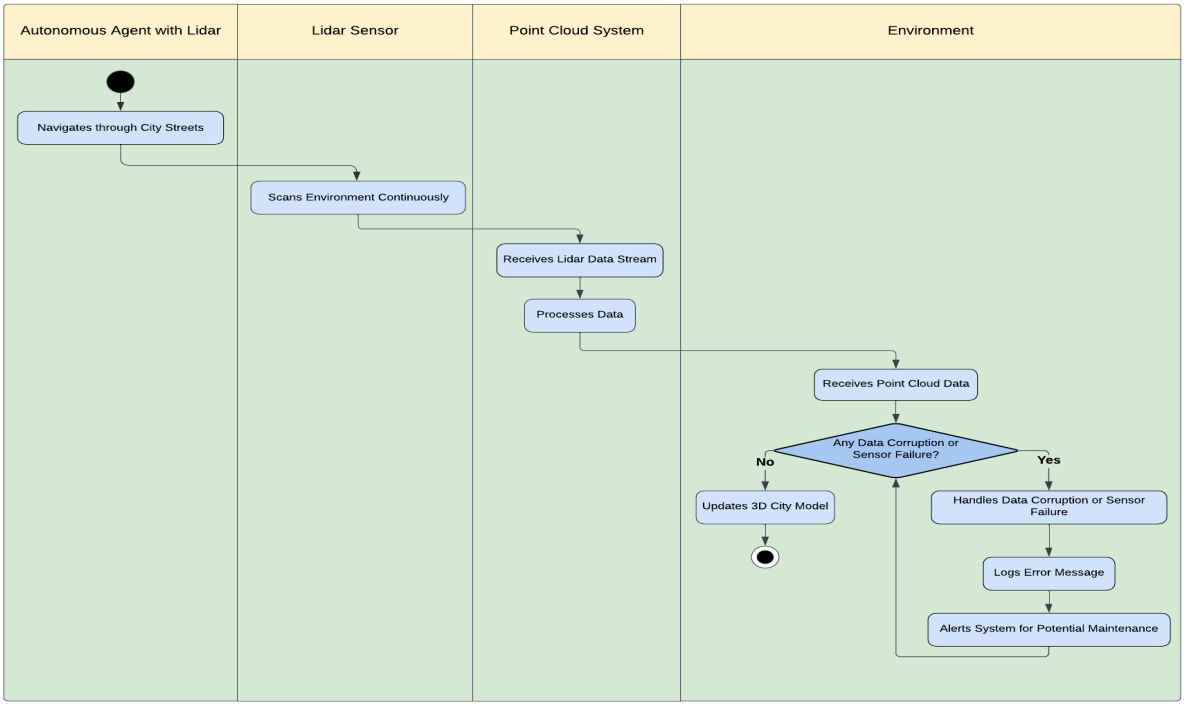


Figure 9. Sequence Diagram of Lidar Data Acquisition for City Modeling



3.3.3 Rviz Visualization of Point Cloud

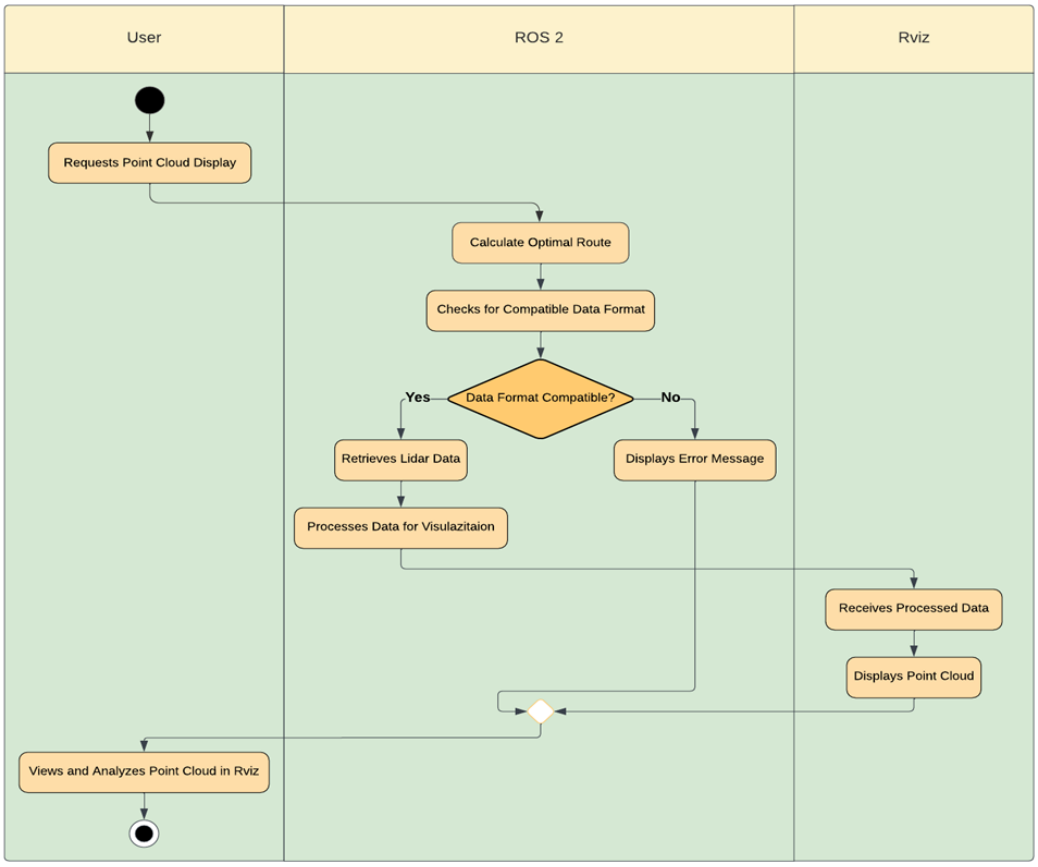
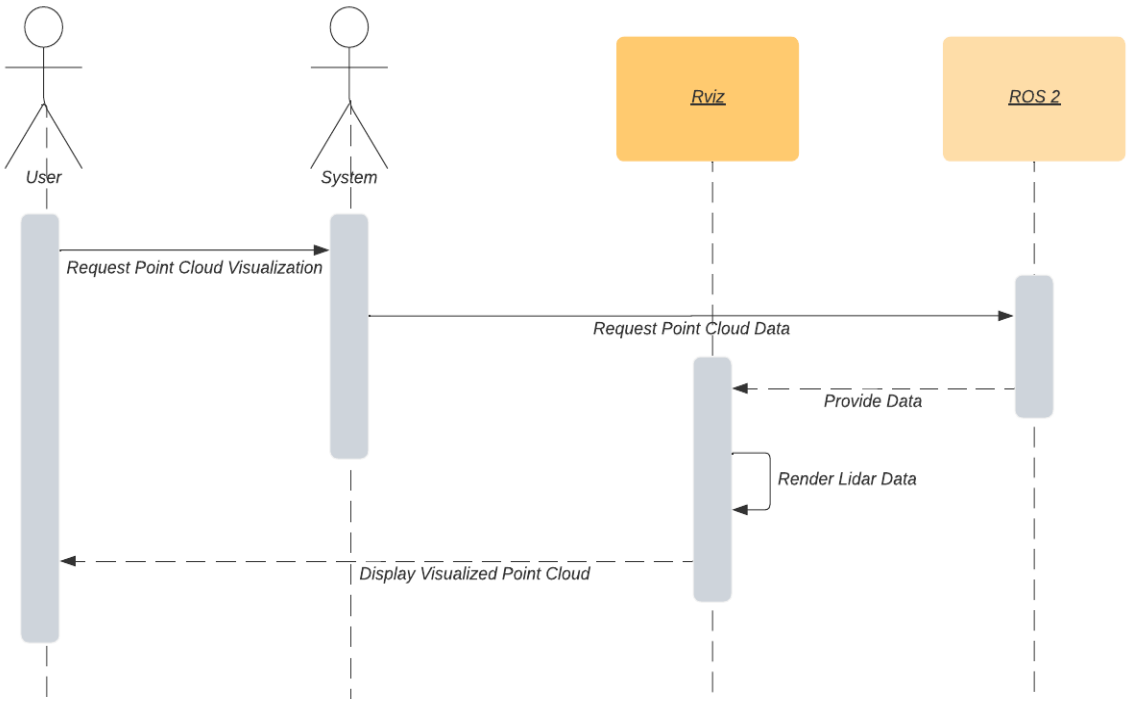
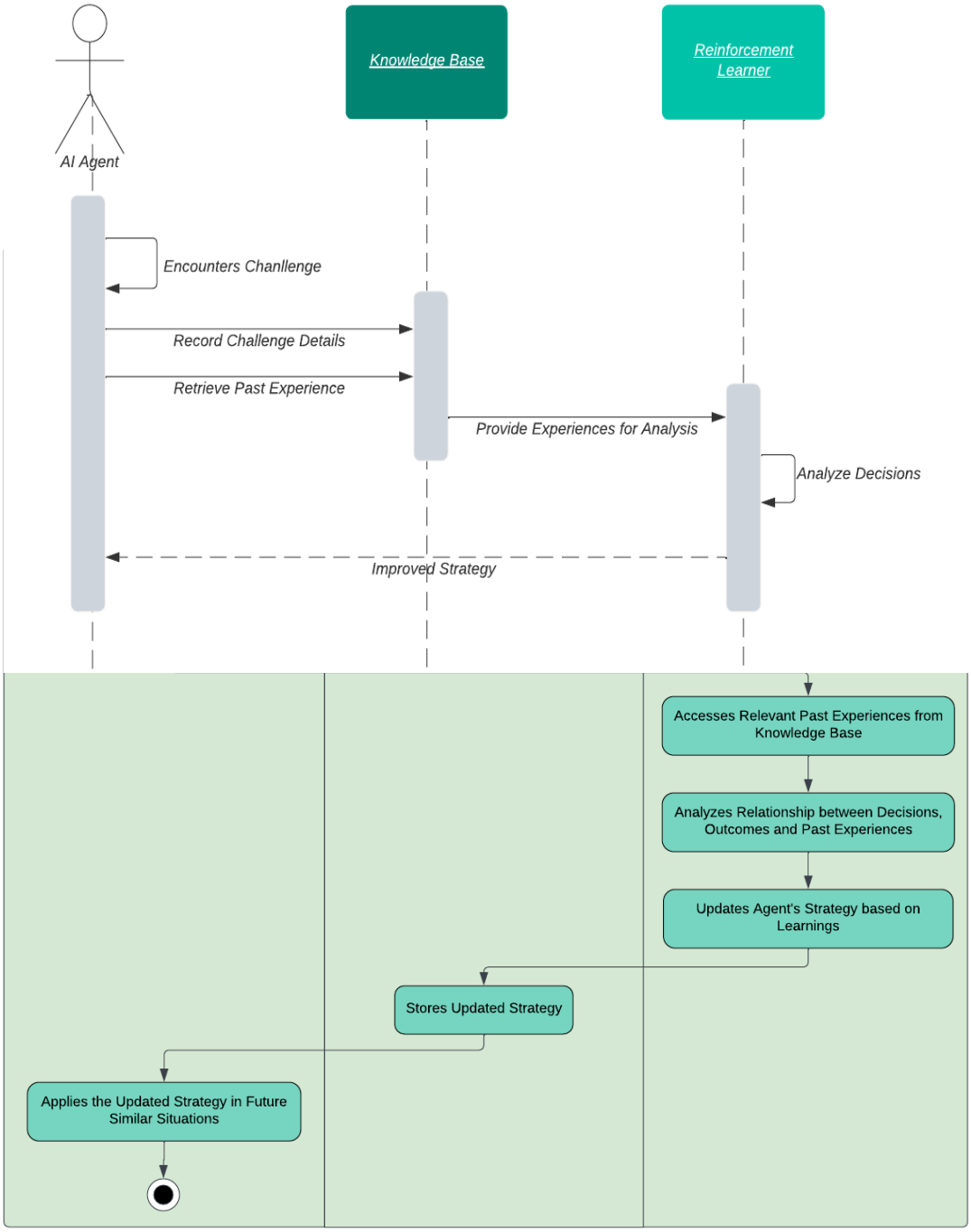


Figure 10. Activity Diagram of Rviz Visualization of Point Cloud



F

Figure 11 *Sequence Diagram of Rviz Visualization of Point Cloud*



3.3.4 Iterative Learning on Encountered Challenges

Figure 12. Activity Diagram of Iterative Learning on Encountered Challenges

Figure 13. Sequence Diagram of Iterative Learning on Encountered Challenges

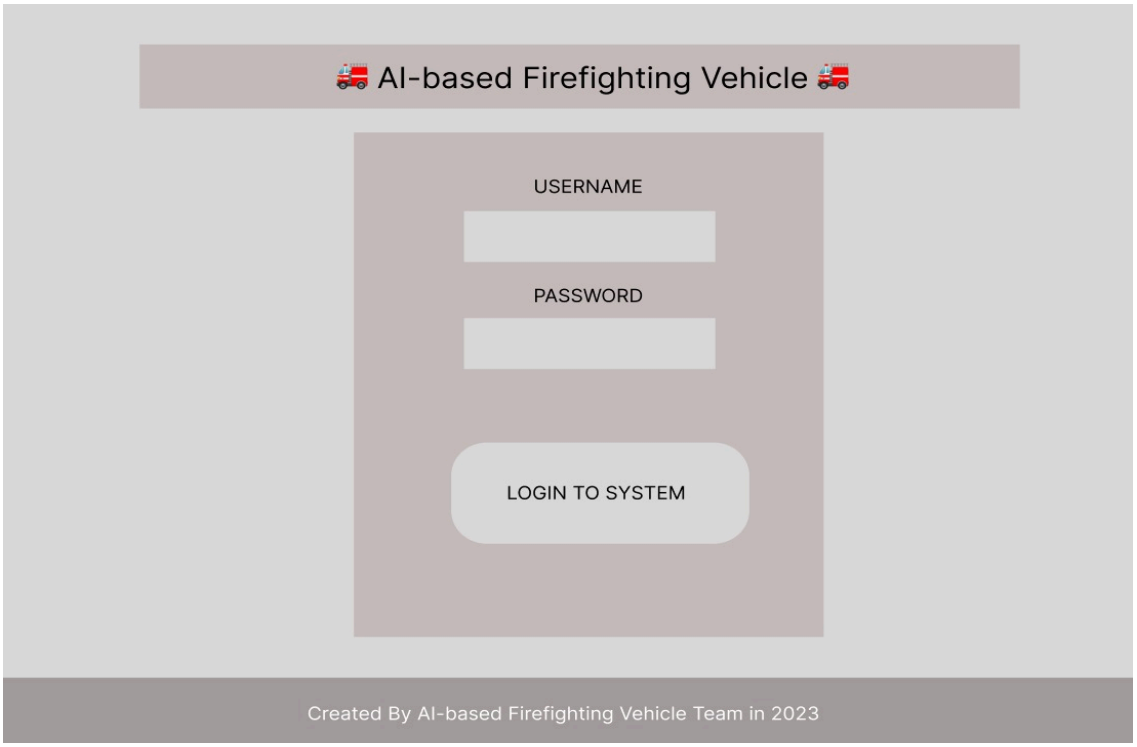


Figure 14. Login Page

4.User Interface Design

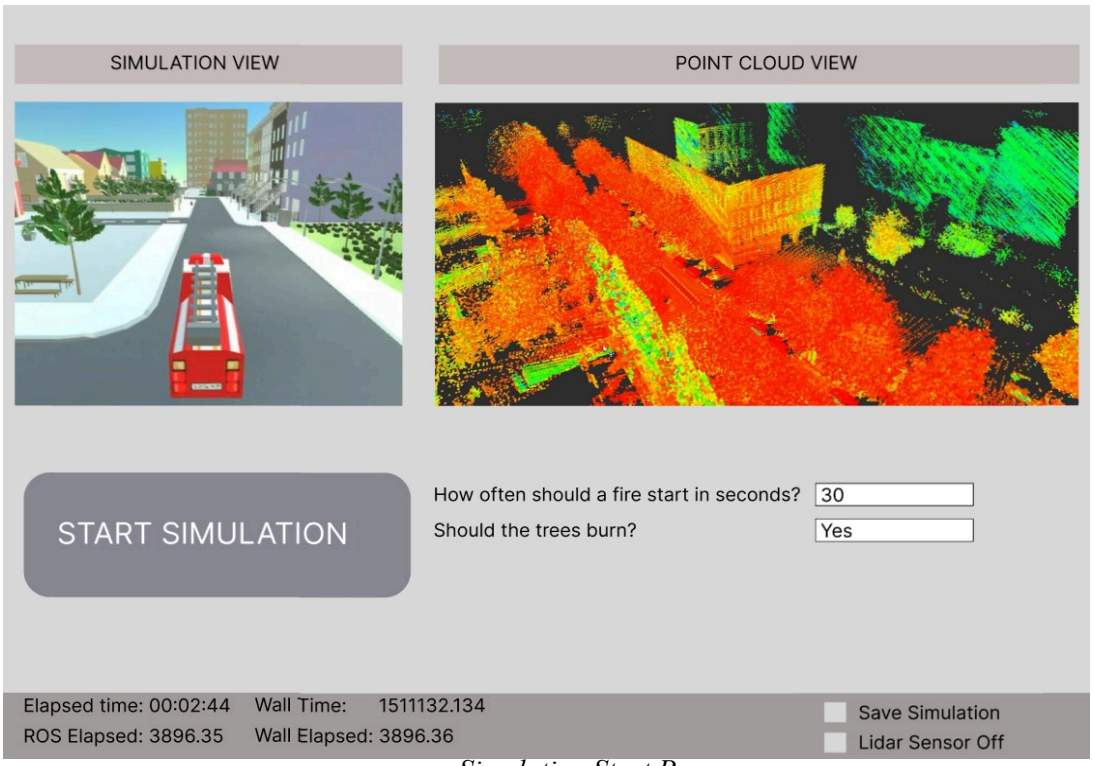


Figure 15. Simulation Start Page

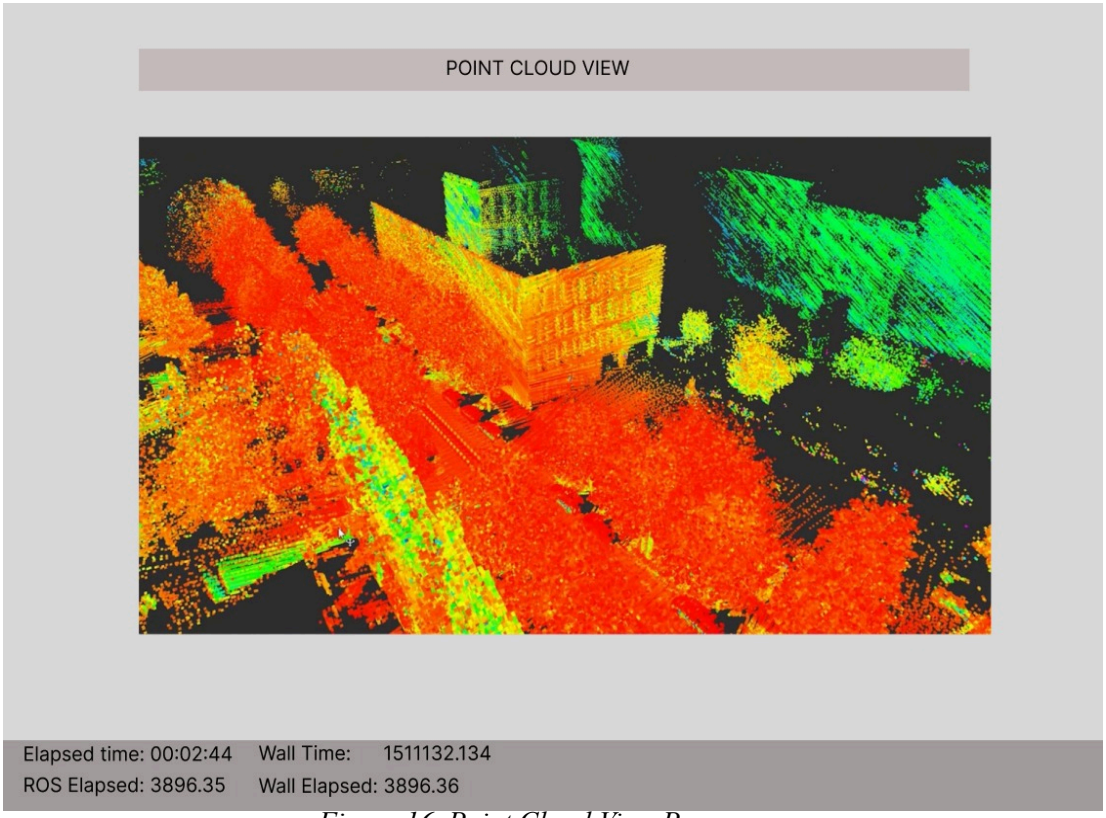
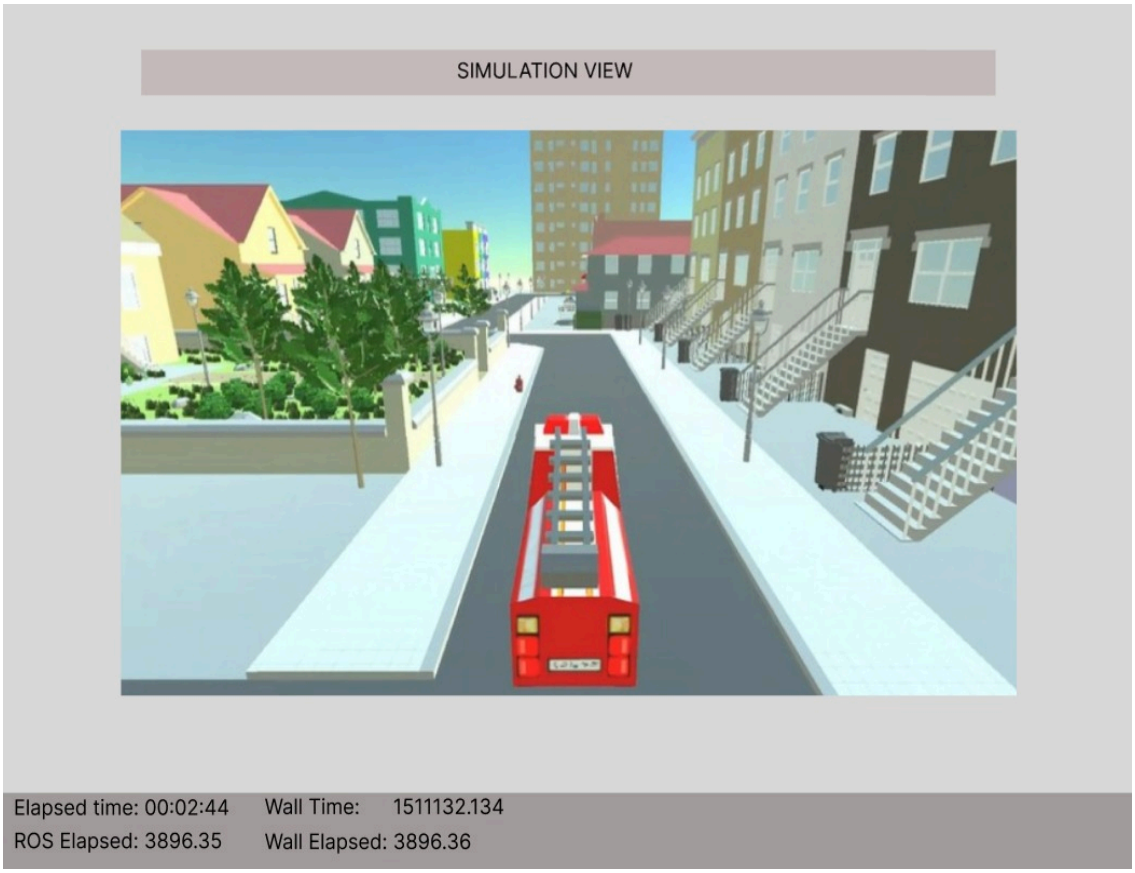


Figure 16. Point Cloud View Page

Figure 17. Simulation View Page



References

[1] Sutton, R.S., and Barto, A.G. (latest edition). Reinforcement Learning: An Introduction. MIT Press.

[2] Mnih, V. et al. (2013). "Playing Atari with Deep Reinforcement Learning". DeepMind Technologies.

[3] Silver, D. et al. (2016). "Mastering the game of Go with deep neural networks and tree search". Nature.

[4] Lillicrap, T.P. et al. (2015). "Continuous control with deep reinforcement learning". arXiv.

[5] Pan, Y., et al. (2017). "Virtual to Real Reinforcement Learning for Autonomous Driving". arXiv.

[6] Dosovitskiy, A., et al. (2017). "CARLA: An Open Urban Driving Simulator". arXiv.

[7] Goodfellow, I., Bengio, Y., Courville, A. Deep Learning.

[8] Russell, S., Norvig, P. Artificial Intelligence: A Modern Approach.

[9] McKinney, W. Python for Data Analysis.

[10] Géron, A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.

[11] Unity Official Documentation and User Guides.

[12] Google AI Blog and DeepMind Publications.

[13] OpenAI Research Blog and Publications.

[14] IEEE Xplore Digital Library.

[15] arXiv e-Print archive.

[16] Advance your robot autonomy with ROS 2 and Unity

[17] ROS2Learn: a reinforcement learning framework for ROS 2

[18] Unity ML-Agents

[18] Reinforcement Learning: A Survey