Emergency Vehicle Priority-based Traffic Management using AI Planning and IoT

IMT2022033 Sourav M Dileep IMT2022071 Samyak Jain

May 16, 2025

All the codes can be found here

Introduction

Efficient urban traffic management is one of the most critical challenges in modern smart cities. Among the various scenarios that strain existing infrastructure, prioritizing the movement of emergency vehicles, such as ambulances, fire trucks, and police cars, through congested intersections is particularly crucial. Delays in emergency response not only hinder public safety but can result in irreversible outcomes such as loss of life.

Traditional traffic control systems rely heavily on pre-programmed signal cycles and static rules that do not adapt well to dynamic, real-time situations. While some advanced cities have integrated sensor-based or camera-based traffic control mechanisms, these systems are often expensive, hardware-dependent, and lack intelligent, context-aware decision-making.

As shown in the image, the traffic signal remains green on an empty road due to fixed, pre-programmed cycles, while on the opposite side, a long queue of vehicles is waiting at a red signal, highlighting the inefficiency of static traffic control systems.

We aim to solve these problems through our project by introducing an AI-driven, context-aware traffic management system.

In this project, we explore a software-based, AI-driven approach to emergency traffic management using **symbolic planning** and **IoT simulation techniques**. Our focus lies in dynamically adjusting traffic signal behaviors in response to real-time high-level inputs (e.g., “An ambulance is approaching signal2”) and using **automated planning** to determine a sequence of actions (like signal changes and vehicle movements) to optimize emergency vehicle passage.

This system integrates three key components:

1. A **natural language interface** that accepts high-level traffic events.
2. A **PDDL-based planning engine** using the Fast Downward planner to compute safe and efficient signal operations.
3. A **Python-based simulation** that visualizes vehicle movement across a 4-way intersection based on the computed plan.

The primary goal of this reading elective project is to bridge the gap between human-level event understanding and AI-level traffic control, all within a software-only, scalable architecture. Through this, we demonstrate how AI planning and IoT principles can be harmonized for real-world impact, without relying on any physical sensors or hardware infrastructure.

Background

The field of Artificial Intelligence (AI) planning focuses on generating sequences of actions to achieve specific goals from a given starting state. In recent years, planning has seen increasing relevance in real-world applications such as robotics, autonomous navigation, and smart infrastructure. One such critical application domain is **urban traffic management**, where AI can be used to dynamically adapt signal behaviors based on real-time context.

In traditional traffic systems, signal cycles are either fixed or respond to low-level sensors (e.g., timers, inductive loops, or cameras). While these solutions have brought some adaptability, they often fail in emergency situations where static rules and delays can lead to significant consequences. The challenge lies in bridging high-level human instructions (e.g., “An ambulance is at signal2”) with machine-executable plans that reconfigure traffic flow safely and efficiently. This is where symbolic AI planning using **PDDL (Planning Domain Definition Language)** and **IoT** becomes powerful.

During the course of this project, we studied the theoretical foundations of **classical planning** and **probabilistic planning**, exploring how planners model real-world actions and transitions. We understood the difference between **offline planning** (precomputed plans) and **online planning** (real-time replanning), and how concepts like **finite state automata** and **Markov Decision Processes (MDPs)** can represent state transitions in uncertain environments.

To realize the planning component of our system, we explored **Fast Downward**, a state-of-the-art planner that accepts domain and problem files written in PDDL. We learned how to define custom domains using:

- **Types** (e.g., vehicle, segment, signal),
- **Predicates** (e.g., (*at ?v ?s*) to represent vehicle location),
- **Actions** (e.g., *move-vehicle, change-signal*), and
- **Preconditions and effects** following STRIPS-style planning.

We also examined how planning trade-offs (e.g., prioritizing emergency vehicles over normal cars) can be represented using action order or separate goals.

Additionally, our study involved exploring how **IoT systems** — typically defined by their ability to sense and react to environmental stimuli — can be simulated in software. In our case, the IoT aspect was represented virtually: the environment receives high-level event inputs (e.g., "A car is arriving at segment1") and reacts by generating planning problems that mimic a real-world intersection scenario.

We reviewed academic resources such as:

- Papers on goal-directed self-assembly (e.g., magnetic alignment through local forces),
- The architecture of dynamic intent parsing in smart homes (used by our seniors),
- PDDL-based planners like FF and Fast Downward,
- Concepts like **heuristic search**, including *h^FF*, and *lazy greedy* strategies.

These studies helped us understand how symbolic AI systems process goal states and generate optimal paths while respecting logical and spatial constraints. We concluded that a hybrid approach — using **symbolic planning to generate signal actions** and **Python-based IoT simulation for execution and visualization** — would offer both clarity and feasibility within the scope of our reading elective.

Methodology

In order to address the problem of intelligent traffic control and emergency vehicle prioritization without hardware dependencies, we adopted a **symbolic AI planning-based approach** integrated with a lightweight **IoT-inspired simulation**. The key idea was to represent vehicle movement and traffic light control as a planning problem in PDDL, and to simulate the resulting plan through software.

Our methodology can be broken into four main phases: high-level input parsing, problem generation, AI planning, and simulation. The system is designed to take natural language inputs (e.g., "An ambulance is at signal2 on segment4") and translate them into a symbolic form that a planner like **Fast Downward** can understand and solve.

Overall architecture

Here is the high-level flow of our system

High-Level Input → NLP + Python Parser → PDDL Problem File →
(input.txt) (preprocessing.py) (traffic_problem.pddl)

→ Fast Downward Planner → Python Simulation
(plan.txt) (simulate_crossroad.py)

High-Level Input Parsing and PDDL Generation

To enable a user-friendly interface between real-world traffic descriptions and machine-understandable planning logic, we developed a custom Python script (*preprocessing.py*) to convert high-level natural language inputs into a structured PDDL problem file.

The system reads plain English instructions from an input file (*input.txt*). Each line in this file describes an event, such as a vehicle arriving at or reaching a specific segment and signal. For example:

"An ambulance has reached signal2 at segment4"

This line is automatically parsed by the script to extract:

- The **vehicle type**: ambulance or car
- The **vehicle identifier**: dynamically generated (e.g., *car1*, *amb1*)
- The **starting segment** (e.g., *seg4*)
- The **signal associated with the segment** (e.g., *sig2*)

Using regular expressions, the script converts each of these sentences into PDDL predicates such as:

(at amb1 seg4)

(signal_status sig2 red)

These predicates are then used to populate the *(:init ...)* section of the PDDL problem file. Additionally, all segments mentioned are marked either as *free* or *occupied* based on initial vehicle positions.

To simulate a realistic urban intersection, we defined a fixed topology with five segments (*seg1* to *seg5*), where *seg2* acts as the central intersection. This segment is specially marked with the predicate (*intersection seg2*) to allow multiple vehicles to enter it without mutual exclusion constraints.

Static connections between segments, such as (*connected seg1 seg2*), are also included to define the road network. Traffic signals are placed between specific segment pairs using the *signal_between* predicate.

The final problem file (*traffic_problem.pddl*) is automatically generated with the following components:

- **Object declarations**: all vehicles, segments, signals, and colors (red, green, yellow)
- **Initial state**: locations of vehicles, signal states, free segments, and road connections
- **Goal**: a default setting where all vehicles aim to reach the central intersection segment (*seg2*)

This preprocessing step ensures that domain experts or city operators can describe traffic conditions in simple English while the system internally translates it into a machine-executable planning problem without requiring any manual PDDL editing.

Planning with Fast Downward

Once the PDDL problem file has been generated from high-level input, the next step in our methodology is to compute a valid sequence of actions that transitions the system from the initial state to the defined goal state. For this, we utilize the **Fast Downward planner**, a widely-used automated planning system that supports STRIPS-style planning and various heuristic-based search strategies.

Fast Downward requires two input files:

- *traffic_domain.pddl*: Defines the types of objects, predicates, and actions available in the system.
- *traffic_problem.pddl*: Specifies the current state of the world and the goal conditions.

The core of this phase lies in the correct modeling of **actions** within the domain file. For example, a simplified version of the *move-vehicle* action is defined as:

```
(:action move-vehicle
 :parameters (?v - vehicle ?s1 ?s2 - segment)
 :precondition (and
  (at ?v ?s1)
  (connected ?s1 ?s2)
  (or (intersection ?s2) (free ?s2)))
 :effect (and
  (not (at ?v ?s1))
  (at ?v ?s2)
  (free ?s1)
  (when (not (intersection ?s2)) (not (free ?s2))))
)
```

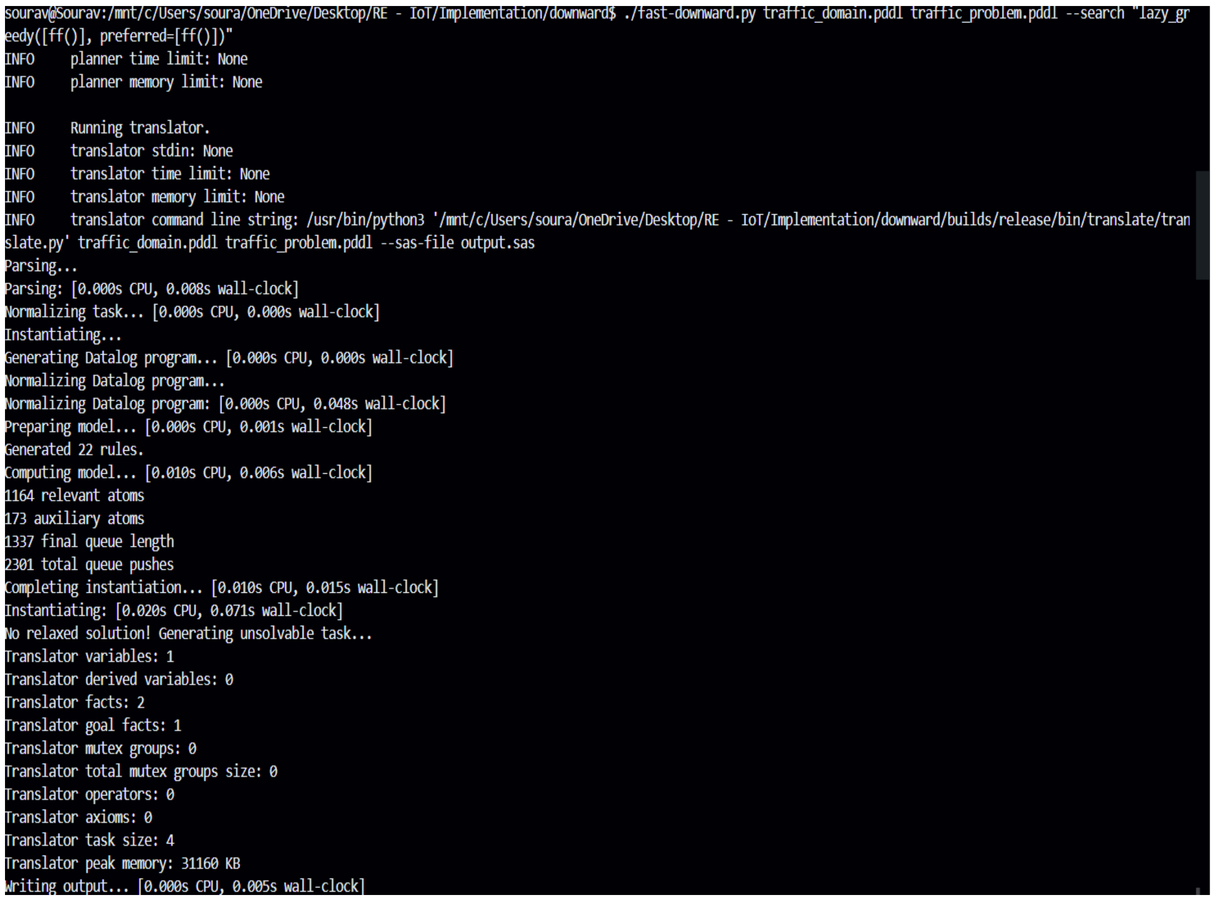
This action allows a vehicle to move from one segment to another if:

- It is currently at the source segment (?s1).
- The segments are connected via (connected ?s1 ?s2).
- The target segment is either free or marked as an intersection that permits multiple vehicles.

Similarly, traffic signals are controlled through a change-signal action:

```
(:action change-signal
  :parameters (?sig - signal ?current ?new)
  :precondition (signal_status ?sig ?current)
  :effect (and
    (not (signal_status ?sig ?current))
    (signal_status ?sig ?new))
)
```

This enables us to simulate signal behavior dynamically, with planners deciding the optimal order in which signals should be changed.



```
sourav@Sourav:/mnt/c/Users/soura/OneDrive/Desktop/RE - IoT/Implementation/downward$ ./fast-downward.py traffic_domain.pddl traffic_problem.pddl --search "lazy_gr
eedy([ff()], preferred=[ff()])"
INFO     planner time limit: None
INFO     planner memory limit: None

INFO     Running translator.
INFO     translator stdin: None
INFO     translator time limit: None
INFO     translator memory limit: None
INFO     translator command line string: /usr/bin/python3 '/mnt/c/Users/soura/OneDrive/Desktop/RE - IoT/Implementation/downward/builds/release/bin/translate/tran
slate.py' traffic_domain.pddl traffic_problem.pddl --sas-file output.sas
Parsing...
Parsing: [0.000s CPU, 0.008s wall-clock]
Normalizing task... [0.000s CPU, 0.000s wall-clock]
Instantiating...
Generating Datalog program... [0.000s CPU, 0.000s wall-clock]
Normalizing Datalog program...
Normalizing Datalog program: [0.000s CPU, 0.048s wall-clock]
Preparing model... [0.000s CPU, 0.001s wall-clock]
Generated 22 rules.
Computing model... [0.010s CPU, 0.006s wall-clock]
1164 relevant atoms
173 auxiliary atoms
1337 final queue length
2301 total queue pushes
Completing instantiation... [0.010s CPU, 0.015s wall-clock]
Instantiating: [0.020s CPU, 0.071s wall-clock]
No relaxed solution! Generating unsolvable task...
Translator variables: 1
Translator derived variables: 0
Translator facts: 2
Translator goal facts: 1
Translator mutex groups: 0
Translator total mutex groups size: 0
Translator operators: 0
Translator axioms: 0
Translator task size: 4
Translator peak memory: 31160 KB
Writing output... [0.000s CPU, 0.005s wall-clock]
```

We use the *ff()* (Fast Forward) heuristic in combination with a *lazy_greedy* search strategy to quickly compute a feasible plan without exploring the full search space.

The output of this phase is a *plan.txt* file that contains a sequence of actions in the format:

0: (change-signal sig3 red green)

1: (move-vehicle car1 seg5 seg2)

2: (change-signal sig3 green red)

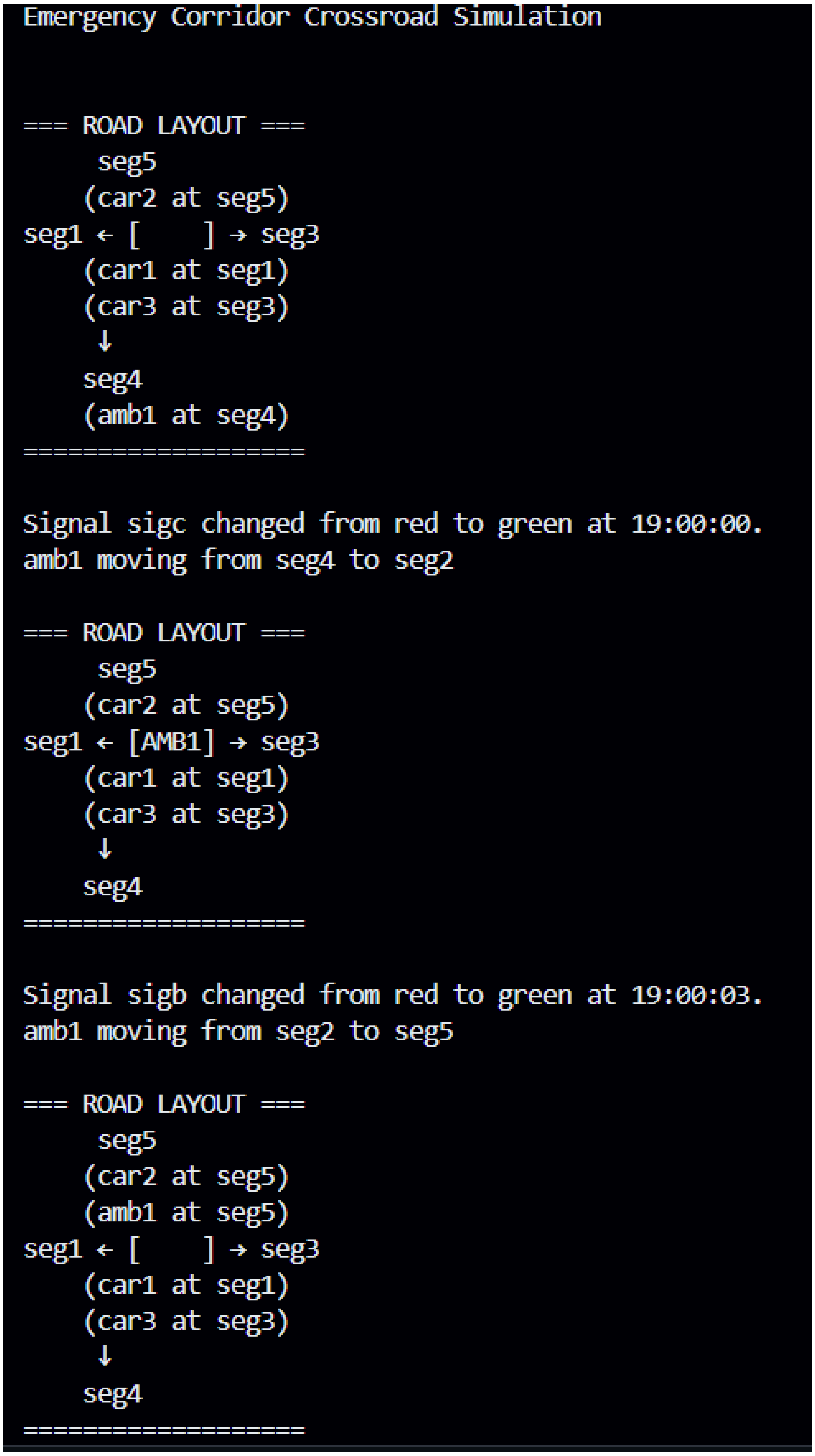
Each step corresponds to a discrete action that must be executed to transition from the initial state to the goal. This plan serves as the direct input for our simulation engine, allowing us to visualize how traffic flows are managed and how emergency vehicles are prioritized through dynamic signal control.

This modular separation between problem definition and planning computation makes our system highly reusable, flexible, and scalable to larger intersections or multi-signal networks.

Simulation in Python

To bring the planned output to life, we built a text-based simulation in *simulate_crossroad.py*. This script:

- Reads *plan.txt* line-by-line
- Tracks vehicle positions
- Changes signal states with timestamps
- Displays a visual layout of the intersection


```
Emergency Corridor Crossroad Simulation


=== ROAD LAYOUT ===
     seg5
    (car2 at seg5)
seg1 ← [    ] → seg3
    (car1 at seg1)
    (car3 at seg3)
     ↓
    seg4
    (amb1 at seg4)
===================

Signal sigc changed from red to green at 19:00:00.
amb1 moving from seg4 to seg2

=== ROAD LAYOUT ===
     seg5
    (car2 at seg5)
seg1 ← [AMB1] → seg3
    (car1 at seg1)
    (car3 at seg3)
     ↓
    seg4
===================

Signal sigb changed from red to green at 19:00:03.
amb1 moving from seg2 to seg5

=== ROAD LAYOUT ===
     seg5
    (car2 at seg5)
    (amb1 at seg5)
seg1 ← [    ] → seg3
    (car1 at seg1)
    (car3 at seg3)
     ↓
    seg4
===================
```

The simulation logic also ensures:

- Ambulances override red signals automatically
- Only one signal is green at a time (safety constraint)
- Final signal states and vehicle positions are displayed at the end

Positives of the Approach

Our approach to traffic signal management and emergency vehicle prioritization offers several key advantages, particularly in the context of a software-only implementation that does not rely on physical infrastructure. Some of the major strengths include:

1. Software-Only Implementation

The entire system is built using standard programming and planning tools (Python, PDDL, Fast Downward) and does not require any hardware like cameras, sensors, or IoT modules. This makes it cost-effective and easy to deploy in a simulated or educational setting.

2. Natural Language Input Compatibility

By designing a lightweight parser for high-level English commands, we bridge the gap between human operators and symbolic planning. Users can describe scenarios in plain text, which the system automatically converts into structured PDDL problems.

3. Scalable Planning Logic

The domain model and planning architecture are flexible and can easily accommodate more vehicles, segments, or signal configurations without needing significant changes in logic. Adding more road segments or intersections would simply involve modifying the connections and signal definitions.

4. Realistic Signal Handling

Through explicit signal modeling, we simulate how real-world traffic lights behave, including turning signals green or red before allowing vehicle movement.

Additionally, mutual exclusion of green signals ensures safety, mimicking real traffic controller constraints.

5. Emergency Vehicle Priority Handling

The planning logic can naturally accommodate prioritization, such as letting ambulances proceed first or override red signals. This feature is crucial in real-world scenarios where time-sensitive routing is necessary.

6. Modular Architecture

Each component — input parsing, problem generation, planning, and simulation — is decoupled from the others. This allows independent development, testing, and enhancement of each part of the system.

Challenges Faced

Despite the overall success of the implementation, several challenges were encountered during the course of the project. These limitations stemmed both from the constraints of the planning tools used and from modeling complexities inherent in traffic management scenarios.

1. Incompatibility with Durative Actions

Our initial domain design used *:durative-actions* to model timed behaviors (e.g., signal changes taking 5 seconds). However, Fast Downward does not support durative planning, requiring us to remove these features and simulate timing externally in Python.

2. Handling Multi-Vehicle Occupancy

Classical planning typically assumes exclusivity — that one object occupies one resource at a time. Since our intersection (*seg2*) was designed to hold multiple vehicles simultaneously, we had to explicitly mark it as (*intersection seg2*) and adjust planning preconditions to relax mutual exclusion constraints.

3. Planner Failures Due to Unreachable Goals

We observed that poorly defined goals (e.g., asking all vehicles to reach a segment that is blocked or unreachable) would cause the planner to fail or return no solution.

Ensuring goal feasibility through proper connectivity and initial conditions became a critical part of the preprocessing step.

4. Signal Synchronization Constraints

In real intersections, only one direction is allowed to proceed at a time. Enforcing this in planning required careful sequencing of *change-signal* actions to prevent multiple signals from being green concurrently. While this was handled procedurally in our simulation, it added complexity to our planning assumptions.

5. Debugging PDDL Syntax and Semantics

Even minor issues like undefined types, improperly scoped predicates, or misordered arguments in PDDL files often led to cryptic planner errors. A significant portion of development time was spent debugging such issues to ensure the planner correctly parsed and interpreted the domain.

6. Lack of Built-in Feedback Loops

Our system follows an offline planning model, where plans are computed once and then executed. In contrast, real-world traffic systems often benefit from online feedback loops that continuously adapt to new inputs. This is a potential future improvement.

Future Work

While the current system successfully demonstrates AI-based traffic signal planning and emergency vehicle prioritization in a controlled intersection, several enhancements can be made to extend its capabilities and realism:

1. Support for Dynamic Replanning

Currently, our system performs offline planning — a plan is computed once and executed as-is. Future extensions can incorporate **online planning**, where the system continuously monitors incoming traffic and replans in real-time to accommodate changes like unexpected congestion or new emergency inputs.

2. Multi-Intersection Integration

The current architecture is limited to a single intersection (centered around seg2). Future versions could simulate an entire urban grid with interconnected intersections, requiring more sophisticated coordination between local planners or a centralized global planner.

3. Traffic Volume and Timing Optimization

By incorporating traffic density data and signal timing constraints, the system can be extended to perform **traffic load balancing** across multiple routes, aiming to minimize congestion and improve flow for all vehicles, not just emergencies.

4. Learning-Based Signal Adaptation

Integration of **reinforcement learning techniques** could allow the system to learn optimal signal strategies over time based on vehicle arrival patterns and reward functions (e.g., reduced waiting time for ambulances).

5. Visual Dashboard and Analytics

Adding a graphical dashboard to display simulation outputs, traffic states, and vehicle flows in real-time would enhance usability for traffic controllers and researchers alike. Exporting logs for further analysis can also be supported.

6. Natural Language Interface Expansion

The natural language input system can be expanded to handle more complex instructions, queries, and even spoken input via speech-to-text APIs. This would make the system usable in practical control rooms by non-technical personnel.

7. Simulation of Pedestrian Crossings and Lane Rules

Introducing pedestrians, lane-specific rules (e.g., left-turn only segments), or vehicle priorities based on arrival time would create a more robust urban traffic simulation environment

Conclusion

In this project, we have developed a complete software-based pipeline for intelligent traffic signal control using AI planning and IoT-inspired simulation techniques. By combining natural language input parsing, symbolic planning with Fast Downward, and Python-based visualization, we have demonstrated how emergency vehicle prioritization can be achieved without relying on physical infrastructure or hardcoded logic.

Our system successfully translates high-level, human-readable traffic events into formal PDDL problem statements and computes optimal action plans to dynamically control traffic signals and vehicle movement. The inclusion of intersection logic and mutual signal exclusivity ensures safety, while the modular architecture allows for future expansion and integration with real-world data sources.

Through this reading elective, we not only gained a deeper understanding of AI planning concepts like STRIPS, heuristics, and domain modeling, but also experienced the challenges and trade-offs involved in building real-time reactive systems. The experience reinforced the value of combining formal AI models with practical simulation frameworks to prototype solutions that could have real-world impact.

This project serves as a stepping stone toward building smarter, context-aware transportation systems, and lays the foundation for future work in scalable, intelligent urban traffic management.

References

- ***Collaborative Decision Making in IoT Network for Sustainable Smart Cities: An Artificial Intelligence Planning Method Based Solution*** **[_link_]**
- **Google**