



# CrowdNav

## Fundamentals of Adaptive Software

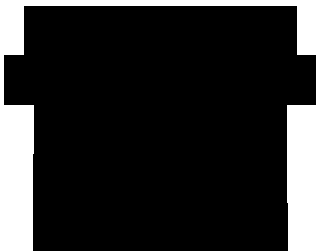


---

Name of the Assignment

Assignment 3: Adaptation Strategy - analysis, design,  
implementation and demo

## Report

Project group 7.1

Name	Student Nr	Email
		

Date:  
Amsterdam, Friday 22 December, 2023

# Contents

<b>1 Overall system description</b>	<b>3</b>
<b>2 Analysis of uncertainties</b>	<b>5</b>
<b>3 Requirements</b>	<b>5</b>
<b>4 Analysis of potential solutions</b>	<b>6</b>
4.1 Analysis . . . . .	6
4.2 Planning . . . . .	7
<b>5 Design of the proposed adaptation strategy</b>	<b>8</b>
<b>6 Implementation</b>	<b>11</b>
6.1 Monitor . . . . .	11
6.2 Analyse . . . . .	12
6.3 Plan . . . . .	12
6.4 Execute . . . . .	15
6.5 Alternative implementation . . . . .	15
6.6 Implementation process . . . . .	16
<b>7 Showcase and evaluation</b>	<b>16</b>
7.1 Training . . . . .	16
7.2 Testing . . . . .	16
7.3 Results and Evaluation . . . . .	17
<b>8 Reflection</b>	<b>18</b>
<b>9 Division of work</b>	<b>19</b>
<b>10 Links to private Github repositories</b>	<b>20</b>

# 1 Overall system description

The foundation of this project rests on the existing adaptive software solution, CrowdNav, which serves as a simulation platform for a dynamic system comprising multiple cars navigating through a city according to their drivers' itineraries. The system incorporates a centralized navigation service. The primary objective of self-adaptation within CrowdNav is aligned with optimizing user satisfaction, with a focus on dynamically adjusting parameters related to trip duration. The experimental setup involves simulating a medium-sized German city with approximately 600 cars, with 10% belonging to CrowdNav.

The system employs a Dijkstra algorithm to determine optimal routes based on a combination of static information (street length and maximum speed) and dynamic data (traffic conditions transmitted by cars). Since CrowdNav involves thousands of cars and drivers, it generates a substantial volume of data. Additionally, the system's self-adaptation requires real-time evaluation of metrics at scale, making it conducive to leveraging Big Data analytics tools. In the original implementation, the adaptation is provided by the Real-Time Experimentation (RTX) engine. The RTX engine integrates with the managed system through Kafka, enabling the system to adapt in real-time based on data analysis. The diagram below shows the overall system architecture with RTX. However, in our implementation, we substituted Kafka and RTX with the UPISAS library and HTTP server. Details of this implementation will be provided in further sections of this report.

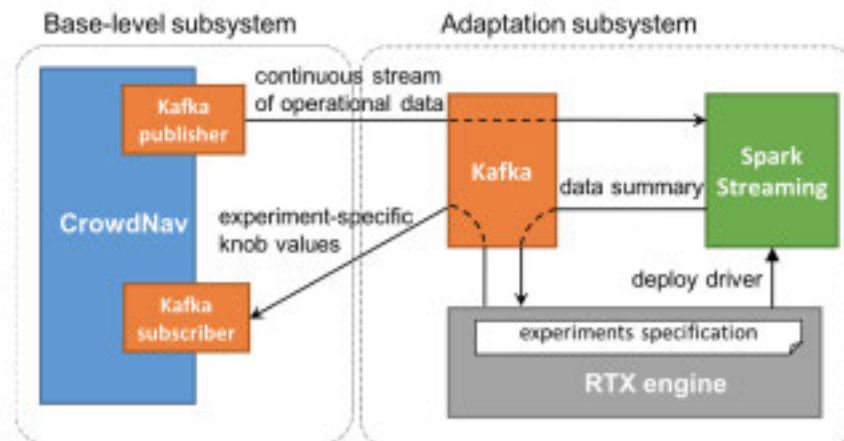


Figure 1: Original architecture of CrowdNav system with RTX.

Additionally in original implementation, SUMO is used for urban mobility simulation and TraCI for interface control. In the figure below, we can see the SUMO visualisation. Due to the limitations of this project in regard to Dockerisation, we decided to drop the visualisation in our adaptation. Although the original experimentation with visualisation allowed us to get familiar with the system and understand its nuances.

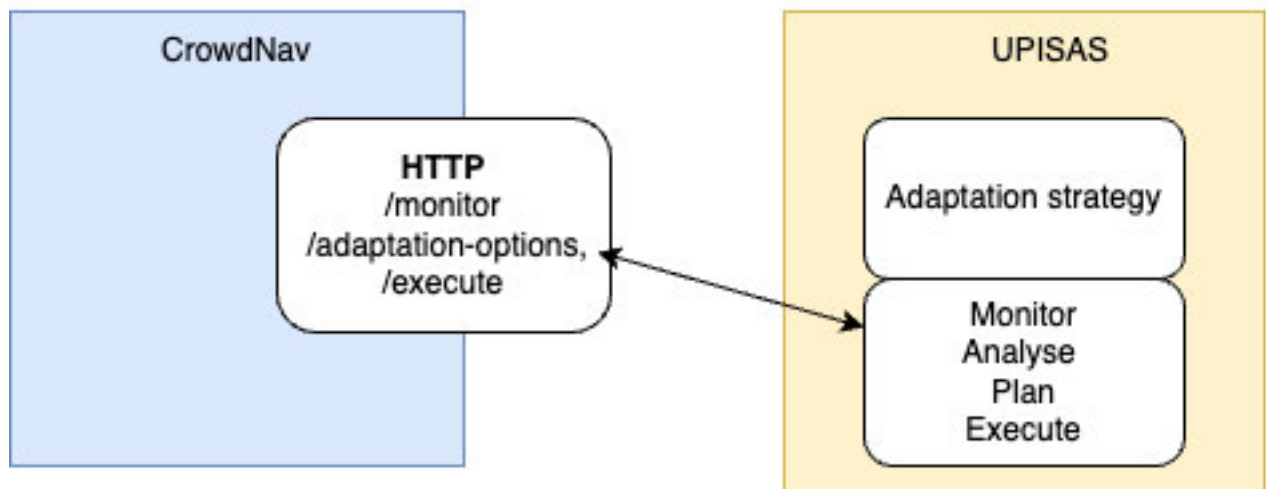


Figure 2: The architecture of CrowdNav system with UPISAS used in this project.



Figure 3: Sumo visualisation.

## 2 Analysis of uncertainties

The main goal of self-adaptation that we want to address relates to the non-functional requirement of optimizing time spent on travel, the adaptation happens around trip duration. Through this project exploration, we are trying to answer the question of what is the optimal set of parameters to minimize the median trip overhead of simulation. Since we want to normalize our results across all types of trips both the long ones that go across the whole city and the short ones, we decided to address the trip overhead, instead of regular time spent on travel.

**Name: Traffic Level Uncertainty**

**Classification:** Real-time Execution

**Context:** This uncertainty relates to the fluctuating and often unpredictable levels of traffic. Managing this uncertainty is essential to enhance route optimization and reduce travel times. Adaptation in this context aims to adjust routes dynamically in response to changing traffic loads. There have been 3 traffic loads identified as low, medium and high. Adaptation changes the set of parameters to manage the traffic the best in each of the given conditions.

**Impact:** The uncertainty in traffic levels significantly affects the system's route optimization capabilities. Effective management of this uncertainty can lead to improved trip efficiency and user satisfaction, by providing more accurate and timely route adjustments. The optimization around the median guarantees that the majority of city drivers would achieve a low difference between planned and actual arrival time.

**Degree of severity:** High, as traffic level fluctuations can alter expected travel times, impacting user experience.

**Sample illustration:** A car is expected to travel a certain route in 3 s under normal conditions. However, unexpected heavy traffic on this route could extend the travel time. The managing system, recognizing this change, recalculates and suggests an alternative route, effectively maintaining the original travel time estimation.

**Evaluation:** Comparing the performance of the self-adaptive system under the 3 distinguished traffic conditions with that of a non-adaptive system can highlight the benefits in terms of reduced travel time and increased reliability.

**Also known as:** Traffic Adaptability, Trip overhead minimalisation

## 3 Requirements

The below set of requirements addresses the managing system parts and considers the represented uncertainty. We want to emphasize again in this part that the goal of the project was to identify the best set of parameters for a given traffic not to minimize traffic, by changing the simulation.

**Traditional requirements:**

1. The system should minimize the average trip overhead time for all the cars in each simulation.

## 4 ANALYSIS OF POTENTIAL SOLUTIONS

2. The system should implement the optimization at run time.
3. The system should optimize the flow of traffic.
4. The managing system should monitor the system at all times.
5. The system should adapt to the number of cars in the traffic and offer the most optimal route for all the cars.

### Relaxed requirements:

1. The system shall minimise the average trip overhead time for most of the cars in each simulation.
2. The system shall implement the optimization at run time, as fast as possible to immediate response.
3. The system shall optimize the flow of traffic at most times.
4. The managing system should monitor the system most of the time, the one re-trial for every monitor is allowed.
5. The system should adapt to the number of cars in the traffic and offer the most optimal route for most cars, around 5 percent of outliers are allowed.

## 4 Analysis of potential solutions

Following the MAPE-K loop implantation, our managing system has five main components. Monitor, Analysis, Plan, Execute and Knowledge. The implementation of Monitor and Execute has been a part of our first assignment and was in the form of an HTTP server. This choice was a compulsory part of the course, therefore in this section, we do not explore the alternatives. Additionally, the Knowledge component is implemented as a part of UPISAS library which has also been a course requirement. Therefore this section is split into two sub-parts. Analysis and Planning part where we explored the potential paths of implementation.

### 4.1 Analysis

In the analysis section, we explore the potential alternatives for the analysis of the traffic. The traffic in the system is determined by the number of cars. The below table presents the potential solutions.

With the above analysis in mind we decided to choose clustering, as we believe that within the limitations of the scope of this project, as well as the complexity of the task, this method would allow us to obtain the required results. Additionally, we also based our decision on research done in the field, that also follows the same method[1].

Figure 4: Comparison of Data Analysis Methods

Method	Advantages	Limitations
Supervised Machine Learning Models	<ul style="list-style-type: none"> <li>- Highly accurate predictions</li> <li>- Suitable for complex pattern recognition</li> <li>- Possible improvement over time</li> </ul>	<ul style="list-style-type: none"> <li>- Requires a large amount of labeled training data</li> <li>- Very complex for our use case</li> </ul>
Clustering	<ul style="list-style-type: none"> <li>- Efficient in categorizing large datasets</li> <li>- Useful for identifying patterns and trends</li> <li>- Can simplify complex data for analysis</li> </ul>	<ul style="list-style-type: none"> <li>- Dependent on number of clusters</li> <li>- Sensitive to outliers</li> </ul>
Agent-Based Modeling	<ul style="list-style-type: none"> <li>- Captures individual behavior and interactions</li> <li>- Flexible for various scenarios and impacts</li> </ul>	<ul style="list-style-type: none"> <li>- Computationally intensive</li> <li>- Sensitive to assumptions and parameters</li> </ul>

However, our exploration has not stopped there. The below table also summarises the advantages and disadvantages of multiple clustering techniques, which were a part of our debate for the analysis section of this project.

Figure 5: Comparison of Clustering Techniques

Clustering Technique	Advantages	Limitations
K-Means Clustering	<ul style="list-style-type: none"> <li>- Simple and efficient</li> <li>- Well-suited for large datasets</li> <li>- Identifies natural groupings</li> </ul>	<ul style="list-style-type: none"> <li>- Assumes spherical, equal-sized clusters</li> <li>- Sensitive to initial cluster centers</li> </ul>
Hierarchical Clustering	<ul style="list-style-type: none"> <li>- Captures hierarchical relationships</li> <li>- No predefined cluster number required</li> </ul>	<ul style="list-style-type: none"> <li>- More intensive than K-Means</li> <li>- May not scale well for very large datasets</li> </ul>
DBSCAN	<ul style="list-style-type: none"> <li>- Identifies various shapes and sizes of clusters</li> <li>- Robust to outliers</li> </ul>	<ul style="list-style-type: none"> <li>- Sensitive to distance and density settings</li> <li>- Struggles with clusters of varying densities</li> </ul>

Our choice at the end was k-means clustering due to the simplicity of the use case and inspiration from the research. We end up using three clusters that were previously determined in [1]. This division splits our use case into low, medium, and high traffic, and allows the adaption to tweak the parameters as the response of this analysis.

## 4.2 Planning

In this section, we explore the alternative paths of Planning within our managing system. As a result of the analysis, we will receive the type of traffic classified as low, medium, or high and we need to repose to that. The paths that we potentially considered are RL, Dynamic Routing Algorithm, and Hybrid approaches. The advantages and limitations of these are described in the table below.

Figure 6: Comparison of Traffic Optimization Techniques

Optimization Technique	Advantages	Limitations
Trip Overhead Optimization using RL	<ul style="list-style-type: none"> <li>- Learns optimal trip planning strategies</li> <li>- Considers historical data and adapts to changes</li> </ul>	<ul style="list-style-type: none"> <li>- Complex reward function definition</li> <li>- Significant computational resources for training</li> </ul>
Dynamic Routing Algorithms	<ul style="list-style-type: none"> <li>- Adapts routes in real-time</li> <li>- Optimizes trip overhead dynamically</li> </ul>	<ul style="list-style-type: none"> <li>- Depends on accurate, up-to-date traffic info</li> <li>- Requires continuous updates and maintenance</li> </ul>
Hybrid Approaches	<ul style="list-style-type: none"> <li>- Integrates strengths of RL and routing algorithms</li> <li>- Balances adaptability with efficiency</li> </ul>	<ul style="list-style-type: none"> <li>- Increased system design complexity</li> <li>- Requires careful parameter tuning</li> </ul>

Overall in the planning phase, we decided to implement an approach that integrates RL for learning and adapting to dynamic traffic conditions for real-time optimization. This combination leverages the learning capabilities of RL while ensuring the efficiency of dynamic routing algorithms in responding to changes in traffic load during the planning phase. Effectively in the analysis phase, we determine what is the level of traffic, and in the planning phase, we execute the correct set of parameters that serve traffic the best in a given run time scenario.

We would like to emphasize here that the above exploration is only valid in the limited scope of the project without the possibility of changing the managed system. In the real-world scenario, multiple alternative implementations could be considered, as traffic is a complex system. A few of the potential solutions that we have discussed include energy optimization, avoidance of blocked streets, and introduction of fast lanes.

## 5 Design of the proposed adaptation strategy

Due to the time limitation of the project our work followed an Agile, fast-changing methodology and we did not execute the original design in the final version. Nonetheless, the following section captures the essential parts of the adaptation strategy design, from structural and behavioral contexts.

On a high level, the proposed adaptation reflects the following flow. Primarily we use the HTTP server to monitor the state of the traffic. Then, within the analysis part, we determine which situation low, medium, or high exists on the run time in our city. Then the planning phase chooses the correct set of parameters that have been obtained through pre-training for certain conditions, and we execute this parameter through the HTTP server. Then the cycle repeats. The details of the implementation will be included in the upcoming section.

The adaptation strategy, part of the UPISAS project, is structured for effective implementation in the CrowdNav system. The process begins with the HTTP server in CrowdNav observing key metrics such as Trip Overhead, Number of Cars, Max Speed and Length Factor, Freshness Update Factor, and Average Edge Duration Factor. The strategy component focuses on monitoring the Number of Cars, after which this data is stored in the knowledge base. The RL strategy then analyzes this information, prompting the planning phase. This phase enables a strategic response, updating crucial factors via the Execute port of the HTTP server. These updates include the Max Speed and Length Factor, Freshness Update Factor, and Average Edge Duration Factor. The figure below describes the structure of our changes. The description only concerns the parts that were changed in the original UPISAS and CrowdNav project, not the

From a behavioural point of view, we do operate in experimental conditions. The CrowdNav is a simulation software therefore it requires a human to trigger the simulation. This process can also be achieved with an automatic trigger from Upisas. Then we can see the 5 modules that correspond to the MAPE-K model. Additionally, there is also an interaction with the RL strategy Table that captures the learning taken from the training phase.



After the Simulation is initialized, the monitor module monitors the values through the HTTP endpoint. The values are saved to the knowledge module. The analysis module reads the values from knowledge at determines the current traffic. It passes that information to the Plan Module, which chooses the best option for the current condition based on results from the RL strategy table. Finally, the Plan module initiates the Execution module which executes the best parameters for the current condition. The process is repeated until the simulation ends as depicted on the activity diagram.

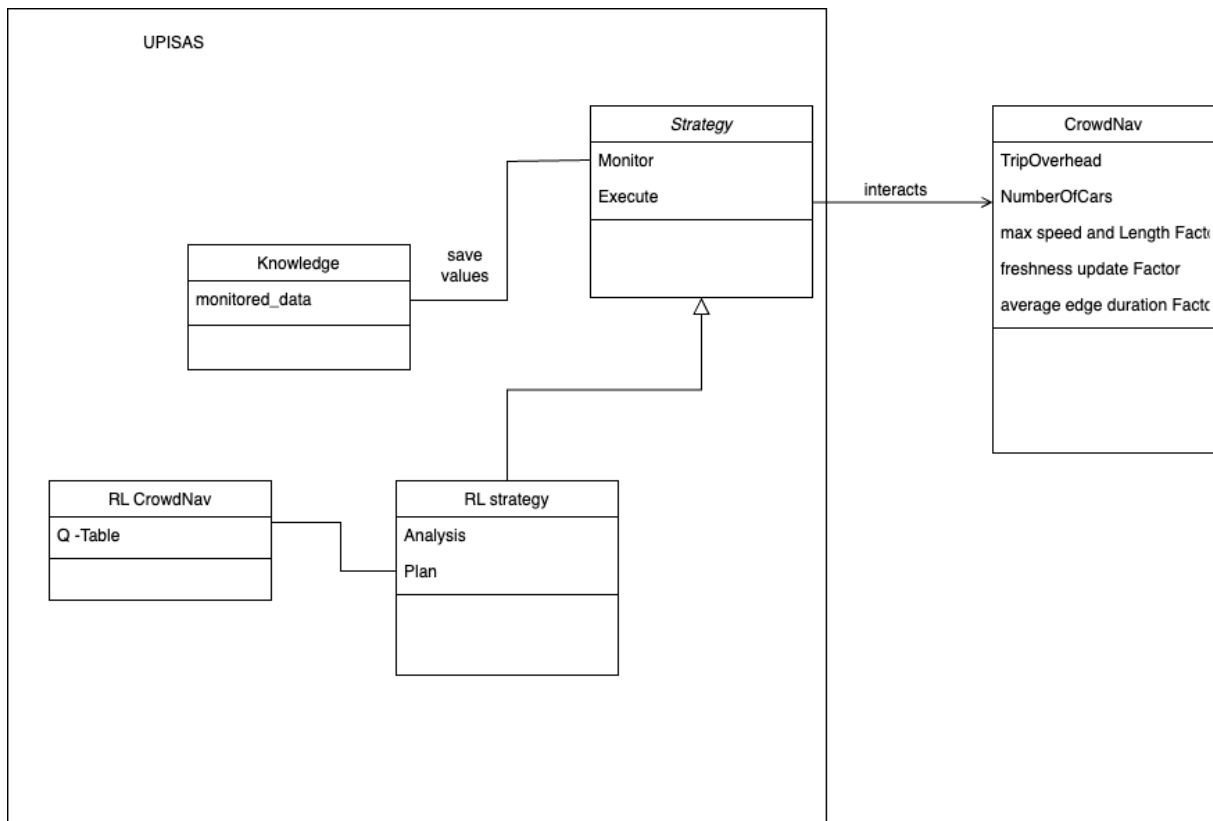


Figure 7: Class diagram representing the strategy of UPISAS interacting with CrowdNav.

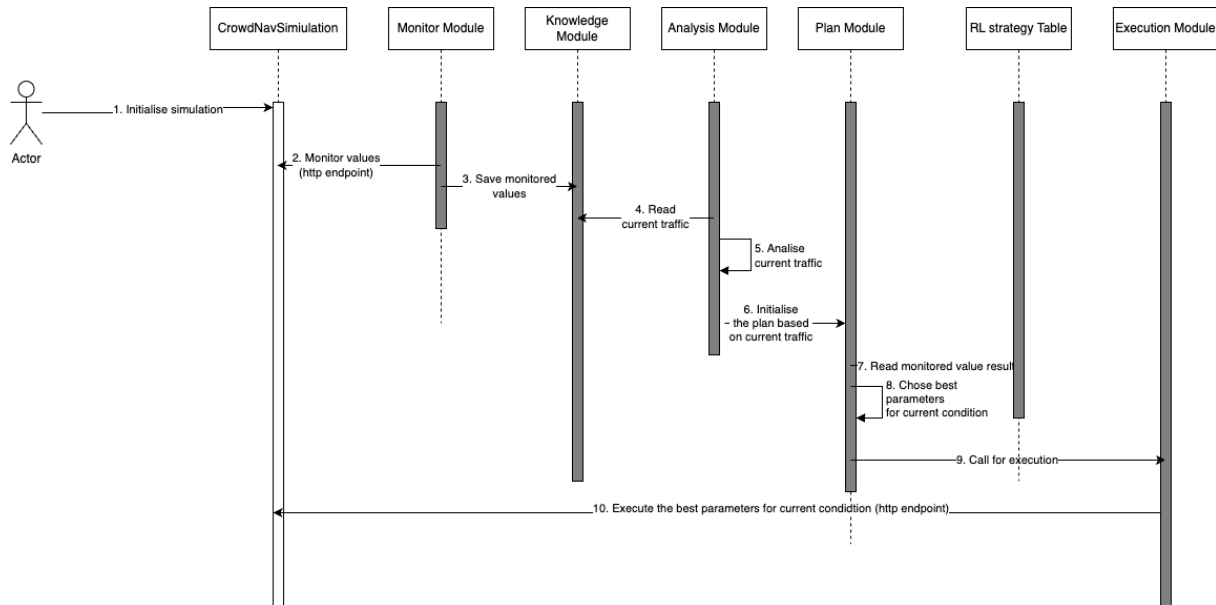


Figure 8: Sequence diagram representing the adaptation strategy.

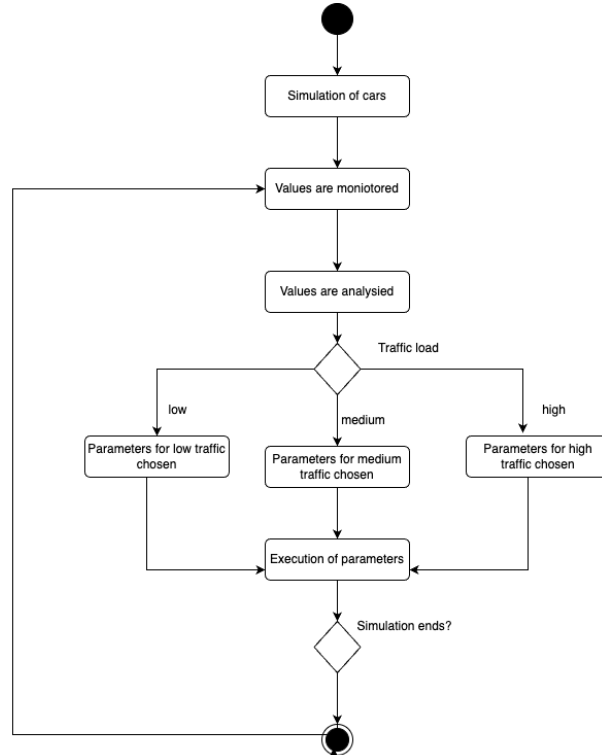


Figure 9: Activity diagram representing the adaptation strategy.

## 6 Implementation

The implementation of the software architecture as described in section 5 has been done systematically for the different components of the MAPE-K loop. Hence, this implementation section will outline the implementation process in the same order while delving deeper into the planning session and the used tools and methodologies for writing the adaptive planner utilizing reinforcement learning. Here, the technologies and libraries used will be discussed, while also pointing out the deployment of the developed software. Furthermore, alternative implementation options as well as the implementation process will be described in this section.

### 6.1 Monitor

In the implementation of the HTTP interface for the Monitoring part of the MAPE-K, we developed an HTTP server and corresponding endpoints to facilitate communication between Upisas and CrowdNav. To capture simulation data in CrowdNav and transmit it to our HTTP endpoints, we designed a singleton class named `SimulationData`. This class serves as a centralized repository for all required traffic data, ensuring consistency and providing controlled access through setter and getter methods. The use of a singleton class eliminates the risk of discrepancies arising from multiple instances and promotes scalability and extensibility. To address the challenge of HTTP data streaming between separate processes, we augmented the singleton class with a JSON file acting as middleware for persistent data storage. The JSON file serves as an intermediary for inter-process communication, ensuring that changes made to the data are consistently reflected and accessible across projects. This design decision enhances data consistency, synchronization, and scalability.

Two Python scripts, `Simulation.py` and `Car.py`, contain dynamic data for user monitoring have been deployed. We identified and updated the relevant points in these scripts, ensuring that the `SimulationData` class always transfers the latest simulation data to the HTTP endpoints. The monitoring capabilities are implemented through two endpoints: the monitor endpoint (`/monitor` - GET) and the monitor schema endpoint (`/monitor_schema` - GET). The former returns a JSON object with monitorable values, including data from the `knobs.json` file and additional fields from `Car.py` and `Simulation.py`. The latter provides a schema with descriptions and types for enforcing constraints during the validation process, enhancing robustness and manageability.

Moving on to execution capabilities, the first endpoint, `/execute` (PUT), allows the execution of one or multiple adaptation options in the system. The HTTP request's body includes a JSON object derived from the `/adaptation_options` endpoint, specifying the adaptations to be executed. All fields from the adaptation options endpoint must be present in the request body, with values reflecting changes if necessary. A successful execution results in a 200 OK response.

The second endpoint, `/execute_schema` (GET), provides a schema to enforce constraints during the validation of execution requests. This schema includes descriptions and types for each

field, offering users insights into the nature and type of execution options.

This adaptation and execution framework enhances the system's adaptability by providing clear interfaces for configuring and executing various options, contributing to the overall flexibility and user satisfaction optimization in the CrowdNav project.

Within UPISAS, additionally, a sliding window technique has been used so that multiple values are captured together. This is important for later reinforcement learning as rewards need to be calculated on the system response to the measures over time. Therefore, 30 values are always monitored and combined to a median value before going into the analysis phase.

### 6.2 Analyse

The Analyzer dynamically operates by utilizing real-time data on the number of monitored cars from the monitoring endpoint to smartly select the relevant context for planning activities within CrowdNav's optimization framework. This decision-making process is influenced by the extensive research conducted by Fredericks, Gerostathopoulos, Krupitzer, and Vogel in their work on Planning as Optimization [1], which explores different contextual scenarios. In their effort to enhance CrowdNav's optimization capabilities, the authors strategically integrate clustering as a pivotal technique for a nuanced analysis of contextual changes. The primary objective revolves around fine-tuning system configurations to achieve optimal performance across a range of runtime situations.

The methodology unfolds with the application of the k-means algorithm, a crucial component that continuously observes and compiles data on both context parameters and system outputs. This iterative process leads to the formation of distinct clusters, each encapsulating a unique scenario within the dynamic traffic environment. What sets this approach apart is the dynamic determination of the optimal number of clusters at runtime, achieved through the Silhouette method. This adaptive mechanism ensures that the clustering configuration aligns seamlessly with the evolving nature of the traffic system, fostering heightened adaptability and responsiveness.

After extensive experimentation, the authors discerned an optimal solution by identifying and leveraging three distinct clusters. This approach boosts the Analyzer's ability to adapt to traffic variations, making CrowdNav more resilient and effective in managing city-wide traffic.

### 6.3 Plan

The Planning part is the probably most important part for optimizing CrowdNavs performance adaptively within our implementation. After the endpoints have been monitored and 30 different values have been retrieved through the sliding window technique (as described in the monitor subsection), the plan gets executed. This Plan is positioned within our design as specified within the class diagram in figure 7. Hence, within the plan, a step function gets executed, utilizing the updated knowledge retrieved through the monitored data. Furthermore, for planning an Adaptive Planner class has been written to build on top of the

object-oriented architecture of UPISAS. This Adaptive planner class gets initialized with the respective (trained) Q\_table as well as information about the system settings (training or testing/exploration or exploitation).

The method used for the adaptive planner is reinforcement learning. Reinforcement Learning operates on the fundamental concept of an agent interacting with an environment (here Upisas interacting with CrowdNav), taking actions (execute), and receiving feedback (monitoring endpoints). This interaction is often modelled as a Markov Decision Process (MDP), which is characterized by states, actions, rewards, and transition probabilities. The agent's goal is to learn a policy, a mapping from states to actions, that maximizes the expected cumulative reward.

One of the defining features of RL is the balance between exploration and exploitation. The agent needs to explore the environment to discover optimal actions while exploiting its current knowledge to maximize immediate rewards. This trade-off is often formalized using exploration-exploitation strategies like epsilon-greedy, where the agent selects the best-known action with high probability but occasionally explores other actions.

Here, a specific form of Reinforcement Learning has been used - which is Q-learning. Q-learning is a fundamental reinforcement learning algorithm designed to enable agents to learn optimal decision-making policies through interaction with an environment. First introduced by Watkins and Dayan in 1992, Q-Learning stands out as a model-free, off-policy learning method [2]. At its core, the algorithm revolves around the concept of action-value functions, denoted as Q-values, representing the expected cumulative reward of taking a specific action in a given state.

For this, specific actions have been chosen by applying discretization as described in figure 10. Here, for the monitored value of type action, equal frequency binning has been used within the

Name	Type	Interval	Binning
maxSpeedAndLengthFactor	Action	1-5	5
averageEdgeDurationFactor	Action	1-5	5
freshnessUpdateFactor	Action	10-20	5
tripOverhead	State	1-3	50

Figure 10: States and Actions retrieved from the monitored values as well as their discretization options

described intervals for the specified number of bins.

Moreover, our state is defined by the discretized tripOverhead value, which means that we are training for 50 different states. From Actions and States, it was possible to generate an empty Q-Table including 50 different states (rows) and 15 different actions (columns) for storing the Q-Values.

Before The Q-Values can be created, however, action value mapping has been implemented together with the used policy (epsilon greedy in this case), which describes the exploration/exploitation trade-off in the following Python pseudo-code.

```

def choose_action(self, current__action_format, action_index):
    # Epsilon-greedy action selection
    if random() < self.exploration_prob:
        # Explore: choose a random action
        upper_bound = (self.num_actions * self.binning_action) - 1
        action = randint(0, upper_bound)

        # Decompose the action into meta action and magnitude
        meta_action = action // self.binning_action
        magnitude = action % self.binning_action

        # Extract the action space and update the current action format
        action_space_layered = [action['values'] for action in self.action_space]
        key = get_key_at_index(current__action_format, meta_action)
        current__action_format[key] = action_space_layered[meta_action][magnitude]
    else:
        # Exploit: choose the action with the highest Q-value
        path = f'./{self.q_table_filename}'
        q_table_updated = read_csv(path)
        q_values = q_table_updated.iloc[self.state_index]
        action = argmax(q_values)

        # Decompose the action into meta action and magnitude
        meta_action = action // self.binning_action
        magnitude = action % self.binning_action

        # Extract the action space and update the current action format
        action_space_layered = [action['values'] for action in self.action_space]
        key = get_key_at_index(current__action_format, meta_action)
        current__action_format[key] = action_space_layered[meta_action][magnitude]

    return current__action_format, action

```

For clarification, meta action is the actions described in figure 10 while the magnitude describes the binning number (interval start to interval end).

After an action has been determined in this way, the reward can be calculated as follows:

Let  $\text{min\_overhead} = 1$  and  $\text{max\_overhead} = 3$ .

$$\text{clipped\_overhead} = \begin{cases} \text{state} & \text{if } \text{state} > \text{min\_overhead} \text{ and } \text{state} < \text{max\_overhead} \\ \text{max\_overhead} & \text{if } \text{state} \geq \text{max\_overhead} \\ \text{min\_overhead} & \text{if } \text{state} \leq \text{min\_overhead} \end{cases}$$

Map the  $\text{clipped\_overhead}$  to a value between 0 and 1 using linear mapping:

$$\text{reward} = 1.0 - \frac{\text{clipped\_overhead} - \text{min\_overhead}}{\text{max\_overhead} - \text{min\_overhead} + 1e - 10}$$

From the reward, the Q-value as well as a defined learning rate and discount factor (here 0.1

and 0.9), Q-values can be calculated. This can however just be done after another step - so once the new state is known - as this is crucial for learning the optimal path after training.

The Q-values are iteratively updated based on the agent's experiences, incorporating immediate rewards and estimations of future rewards. The update equation, often referred to as the Bellman equation, expresses how Q-values evolve, balancing new information with existing knowledge. The learning process involves exploration and exploitation, where the agent explores new actions to discover optimal strategies while exploiting known information to maximize immediate rewards.

### 6.4 Execute

The execution is done - during exploitation - using the largest Q-value available for the given state. The successful execution of actions during this phase is vital for CrowdNav's adaptive performance, allowing UPISAS to navigate efficiently and make real-time decisions based on its learned experiences, ultimately contributing to the overall success of the system in dynamic and challenging scenarios.

### 6.5 Alternative implementation

Various alternative implementations could have been considered in the below list we describe a few of them.

1. The use of a Multi-armed Bandit algorithm, instead of using the Q-learning implementation

The Multi-Armed Bandit (MAB) algorithm is a simpler form of reinforcement learning that focuses on balancing exploration and exploitation but without considering the state of the agent. In our implementation, we could have replaced the state-action reward structure of Q-Learning with a reward-based approach for each action. Each action (or "arm") has its own estimated value, updated based on the reward function. In our system, each adaptation option in CrowdNav could be treated as an independent action. The algorithm would choose an adaptation option, observe the resulting reward, and update its estimation for that specific option. Even though this approach might have been relatively simpler to implement, there are challenges that arose with it that made us choose regular RL over MAB. Lack of state consideration might have reduced the effectiveness in our complex traffic environment.

2. The use of Deep Q-Networks (DQN)

DQN is an extension of Q-Learning that utilizes deep neural networks to approximate the Q-Value function. It is suitable for handling high-dimensional state spaces. In this scenario, we could have incorporated a neural network that takes the state as input and outputs Q-values for each action. The network could have been trained to approximate the optimal Q-value function. The state input could include various traffic parameters,

## 7 SHOWCASE AND EVALUATION

even the possible extension of the managed system and the network would output the Q-values for different traffic management actions. However, this solution required too much computational power for the scope of this project.

3. Policy gradient methods to directly learn a policy that maps states to actions, which can be advantageous in continuous action spaces (which could have been another alternative implementation we could have done).
4. Monte Carlo methods instead of Q-Learning to estimate the expected return by averaging over multiple trajectories. Monte Carlo methods estimate the expected return by averaging over multiple trajectories, rather than estimating Q-values based on immediate rewards and next-state values. This would be a replacement for the Q-Learning update rule with an approach that evaluates policies based on average returns from actions. In the context of CrowdNav, we could have evaluated traffic management strategies based on their long-term outcomes over several iterations, rather than immediate rewards. This could work for multiple simulations.

### 6.6 Implementation process

The facilitation of the project was through shared code repositories and to-do lists. The implementation process followed the prototype-based structure with shared GitHub repositories. During implementation, we had put a great emphasis on co-reviewing our implementation and peer-programming. The project has undergone multiple requirements changes due to the feedback, which were addressed on a small-sprint basis (weekly).

## 7 Showcase and evaluation

### 7.1 Training

We run UPISAS with our adaptation strategy to train our Q-tables for the 'low', 'medium' and 'high' traffic levels (500 cars, 700 cars and 800 cars respectively). Each scenario is run 300 times with the same seed value to maintain the integrity of the simulation. Each round consists of 30 consecutively monitored values, bringing up our training data to 9000 monitored values of trip overhead. In each round, we calculate the median of the latest 30 tripoverhead values.

For each level of traffic, we trained separate q-tables to ensure our learning is representative of our traffic situation.

### 7.2 Testing

For testing, we benchmarked our adaptation strategy against a baseline for different levels of traffic. In our baseline, we did not use any specific adaptation strategy and we executed random actions in each round. The baseline was also run with the same seed as the test runs



for our adaptation strategy to maintain the consistency of the simulations. The baseline was also run 300 times for all three levels of traffic. After training q-tables in the training phase, we performed testing of our adaptation strategy for all the traffic levels separately, with the corresponding q-tables. This clustering of levels of traffic is realized through our analyzer for choosing the adaptation strategy based on context. After running the adaptation strategy for different contexts for 300 rounds (each containing 30 median values of trip overhead), we store the results. The results include the median trip overhead in each round. Finally, for comparison, we took the median of all the medians of the trip-overheads collected through our baseline run as well as our adaptation strategy runs for all the different contexts. The results are discussed in the following section,

### 7.3 Results and Evaluation

The results are depicted in Table 11. As evident in the table, our adaptation strategy decreases the median of trip overhead in all three different traffic levels in comparison the the baseline runs for the same context. This shows that we successfully achieved our goal of mitigating the uncertainty of very high values of trip overhead which would make the drivers in the traffic dissatisfied. We observed that even with 300 rounds of training, our adaptation strategy was capable of optimizing the trip overhead values noticeably. We assume and argue if we prolong the training phase, the result might be even better. Another advantage of our adaptation strategy is that it performs contextually for different levels of traffic without generalisation. This clustering approach leads to better optimization of trip overhead due to considering the impacts of context shifts. However, it is worth mentioning that learning as well as the testing was performed only for one seed value of the traffic simulation. This was done to partially mitigate randomness and due to the scope of this project. This means that the results are not generalizable to all traffic simulations. However, we believe that the adaptation strategy can be utilised to optimize other seed values as well as even higher traffic levels as considered. Additionally, it is important to mention that our test cases were only run 300 times due to the limited scope and time of this project. However, a higher number of runs might cover more unseen situations within the simulation. Lastly, our adaptation strategy considered only the median trip overhead as the uncertainty. This might be a limitation as there are more uncertainties in the CrowdNav system such as user satisfaction, congestion etc. Furthermore, only three actions in knobs.json (maxSpeedAndLengthFactor, averageEdgeDurationFactor and freshnessUpdateFactor) are considered in our adaptation strategy. This might not be the best-case scenario as all values of knobs.json might affect the trip overhead values and we might lose some valuable information for better optimisation by restricting our scope.

From the values of the median in the table 11, it is clear that, with our managing system UPISAS performing the adaptation strategy, the CrowdNav system is better optimised leading to smaller values of median trip overhead over 300 runs compared to our baseline taking random actions without any adaptation strategy. Furthermore, for 500 cars ("low" traffic levels), there is a percentage improvement of around 8.37 in median trip overhead value which reaffirms that Q-learning could be performed to make adaptations for CrowdNav leading to better optimisation

## 8 REFLECTION

Levels of traffic	Median of medians of the 300 runs		Percentage improvement from Baseline
	Baseline	Strategy	
Low (500 cars)	1.9103656071496	1.7538545720886476	~8.37
Medium (700 cars)	2.0274640194182316	1.707927876669611	~15.84
High (800 cars)	2.101854585165429	2.0352597990225956	~3.33

Figure 11: Results of adaptation from baseline runs and our adaptation strategy for different levels of traffic for 300 rounds

of the trip overhead of the cars in the simulation. Furthermore, the same could be observed for other traffic levels as well with 700 cars leading to around a 15.84 percentage improvement in median trip overhead values and with 800 cars, our adaptation strategy led to a small 3.33 percent improvement in median trip overhead.

To add to this, it is clear that our adaptation strategy performs consistently with context changes in the three levels of traffic it was trained and tested for. But in the case of medium traffic levels with 700 cars, our adaptation strategy performs significantly better leading to around a 15 percent decrease in median trip overhead.

Furthermore, we argue that our managing system qualitatively also performs better because our adaptation strategy adapts to dynamic changes in the traffic conditions in the system and gives an optimised median overhead. Secondly, since we perform a reduction of adaptation spaces through discretization and learning while also making the adaptation consider context changes, we could surely say that with longer training and more contexts considered, our managing system is capable of learning and improving over time as it sees more situations and gets trained on them. This helps in the sustained improvement of the managing system. One more qualitative aspect concerning this is the scalability of the adaptation strategy. Our adaptation strategy further follows all considerations of a MAPE-K system thus enabling easy understanding of the adaptation strategy as well as help in further extension. Using the capabilities of UPISAS to the maximum, our adaptation strategy is modular and the solution can be scaled and extended effectively for other context changes and also could be easily modified to further include other uncertainties of the managed system to be considered.

## 8 Reflection

The project has been a learning curve for the whole group and gave us a meaningful insight into the world of adaptive systems. In the below section, we present the key takeaways and challenges that we faced during the process.

Primary, we had a chance to experience the MAPE-K loop development in the real project scenario. The work with the UPISAS library should use clear division with each of the MAPE-K component and their communication, with each other. We also followed the execution order and used the knowledge component to facilitate the flow.

Additionally, we also got familiar with the overall environment of the project. In the early stages, we needed to facilitate the work with the Docker environment and older versions

## 9 DIVISION OF WORK

---

of Python, both of which were new to some of the team members. Furthermore, we also experienced that UPISAS is a useful and handy tool to orchestrate the managed system, and from an overall course perspective, we saw its power of applied to various environments.

Nonetheless, we faced various challenges in the way of implementation. Primarily we faced various environmental issues at the early stages of the project that postponed some of the crucial work. Setting up the Docker environment and facilitating the HTTP server were the first two milestones in the project development. Later, we needed to narrow down the fewer knobs.json values that would affect the system. Due to the system being very sensitive to randomness this choice was not easy and required in-depth experimentation. On top of that, the simulation start wait time that was implemented as a mitigation strategy for monitoring (waiting till we actually can observe some values) significantly slowed down our development and debugging process.

The biggest challenge, however, was the fact that a CrowdNav was a black box for us to some extent. Not fully understanding each detail of our managed system, made it hard to see the effects of our changes. Also as mentioned already its sensitivity to randomness made it difficult to deduct the correct next steps in the implementation process.

Finally, the project had a very limited time scope that restricted our creativity and the possibility of prolonged training on various configurations. It served its purpose for the learning, but we believe that with longer training time the improvement of the performance could have been greater. Overall, we enjoyed working on the project and having this opportunity to apply the theory into practice.

## 9 Division of work

In the following section, the different responsibilities are displayed. We want to emphasize that we always worked and checked quality together, which is why the responsibilities have been shared very equally.

Brainstorming and ideation	X	X	X	X
RL Algorithm implementation	X	X	X	X
State space and action implementation			X	
Action implementation			X	
Reward implementation	X			
UPISAS to strategy communication work with it		X		X
Training for different scenarios	X	X		
Testing for different scenarios		X		X
Running baseline for different scenarios	X		X	
Github repo preparation for project	X			
Report section 1-4	X			X
Report Implementation details			X	
Report Showcasing Solutions		X		X
Demo video		X		X
Github repositories preparation		X		X

## 10 Links to private Github repositories

The following project can be tested by cloning two of the following repositories.



## References

- [1] Erik M Fredericks, Ilias Gerostathopoulos, Christian Krupitzer **and** Thomas Vogel. “Planning as optimization: Dynamically discovering optimal configurations for runtime situations”. *in 2019 IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*: IEEE. 2019, **pages** 1–10.
- [2] Christopher JCH Watkins **and** Peter Dayan. “Q-learning”. *in Machine learning*: 8 (1992), **pages** 279–292.