WEST UNIVERSITY OF TIMIŞOARA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
BACHELOR STUDY PROGRAM: Computer Science

BACHELOR THESIS

SUPERVISOR:
Lect. Dr. Todor Ivașcu

GRADUATE:
Mihai Andrei Gherghinescu

TIMIŞOARA
2023

WEST UNIVERSITY OF TIMIŞOARA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
BACHELOR STUDY PROGRAM: Computer Science

Traffic Manager

SUPERVISOR:
Lect. Dr. Todor Ivașcu

GRADUATE:
Mihai Andrei Gherghinescu

TIMIŞOARA
2023

Abstract

Effective traffic management is a vital component in ensuring the smooth and efficient movement of vehicles, reducing congestion, and enhancing overall transportation systems. As urban populations continue to grow, the challenges associated with traffic congestion, environmental impact, and safety concerns become more pronounced. Therefore, the need for innovative and intelligent traffic management solutions is imperative.

This thesis provides an overview of various approaches and technologies employed to tackle traffic management challenges as well as a new traffic management system that would solve most of the traffic congestion problems in the near future. It highlights the significance of intelligent transportation systems (ITS), which leverage advancements in communication networks, sensors, and data analytics to enhance traffic flow and optimize resource allocation.

One key aspect of traffic management is traffic signal optimization. Traditional fixed-time signal systems are being replaced by dynamic adaptive systems that continuously monitor traffic patterns in real-time and adjust signal timings accordingly. By dynamically responding to traffic conditions, these systems optimize traffic flow, reduce delays, and minimize congestion.

The new system itself attempts to benefit on already existing V2V and V2I communication and provides an IPC based system with multiple clients, servers and proxies that can achieve data car collection and processing on a global scale. Also, the system bases on 8 traffic states that in normal conditions follow one after the another in a cycle, just like with any other junctions at the give moment, but the normal flow can be altered in specific scenarios to improve traffic flow.

Unlike other traffic systems, that try to predict the traffic, a simpler, more effective approach was provided that dynamically adapts based on traffic conditions. To be more precise green light time and waiting time are determined based on the number of vehicles that crossed or are waiting at the junction. The data is collected in various ways, either from the vehicles themselves or with the help of additional hardware like microcontrollers, cameras, GPS, ZigBees, radios etc. To achieve that we defined a new message standard and trained an object-detection neural networks model that could recognize our vehicles.

Rezumat

Lucrarea are ca scop prezentarea unui nou sistem de semaforizare ce combină mai multe idei deja în uz. Ca și o parte introductivă, se va face o prezentare asupra tehnologiilor dezvoltatea de-a lungul timpului. Vom aborda in special teme precum sisteme de semnaforizare pe baza detecției de obiecte, sisteme ce folosesc senzori, sisteme ce se bazează pe algoritmi de planificare și in cele din urmă sisteme pe baza de semnale radio cu rază mica de acoperire(DSRC). Vom face o comparație evidențiând avantajele și dezavantajele acestora.

Odata prezentate, vom sublinia de ce este necesar să cream un nou sistem in momentul de față, anume unul care este suportat de infrastructura traficului din momentul scrieri lucrări.

În urmatoărele capitole vom descrie cum va funcționa sistemul nostru mai exact, de ce credem că este o mai bună alternativă de gestionare a traficului și ce avantaje/dezavantaje aduce. În sine acest sistem se bazează pe 2 mari sisteme deja dezvoltate și aduce la rândul său îmbunătățiri. Arhitectura sa presupune crearea mai multor servere ce colectează date, redirecționează dar și gestionează traficul pe baza informațiilor primite. De asemenea, pentru a procura datele am ales să avem doua tipuri de clienți: unul staționar, conectat direct la un singur semafor și unul mobil.

Pentru a putea face corespondența dintre clientul mobil și serverele ce gestionează traficul și pentru a putea identifica care este următorul server care urmează a fi intersectat, am decis să folosim un proxy ce acționează ca și un fel load balancer și permite sistemului a fi scalat la nivel global. Există un server principal de unde pleacă interogarea clientului, dar acestuia i se poate fie răspunde cu următoarul server în ruta sa sau poate fi redirecționat la alt proxy din calea de acoperire a celui curent. De asemnea interacțiunea cu server-ul principal este minimizată, deoarece dupa fiecare request clientul își stochează și salvează toate proxi-urile vizitate de acesta, interogând mereu utlimul proxy vizitat. Vom vorbi despre serverele ce vor capta date și vor gestiona traficul, ce vor fi amplasate la fiecare intersecție în parte, despre clientul staționar ce este legat la o cameră și detectează mașinile ce urmează a trece prin intersecția noastră, despre clientul mobil instalat direct pe automobil, ce va parsa date de la GPS și va determina coordonatele cât si directia vehiculului și le va transmite prin intermediul radioului către server.

Odată descris se va face o scurtă prezentare asupra software-ului dezvoltat ce alcătuiește sistemul, dar și asupra modelului antrenat pentru a recunoaște autovehiculele. Vom sublinia de ce credem că traficul nu poate fi prezis ți în același timp vom argumenta de ce credem că adaptarea dinamică a timpului de așteapare și a fazelor de trafic este cea mai bună solutie de gestionare a traficului din punctul nostru de vedere.

Într-un final, vom avea o privire critică asupra sistemului conceput, interpretând datele obținute și vom propune potențiale îmbunătățiri ale acestuia.

Contents

List of Figures

Chapter 1

Introduction

1.1 Current state of traffic

Intersections are the points where two or more routes overlap with one another. As a result, for drivers, they act as an obstruction for smooth traffic flow especially within urban areas where the infrastructure demands them the most. This leads to unwanted traffic scenarios like interruptions, congestions, and overall poor control and management of the traffic.

To minimize the amount of time lost at crossroads while maintaining the safety of the drivers, Intelligent Transportation Systems (ITS) were developed. One major impact that can be seen as a result is the reduction of CO_2 car emissions. You may think that this wouldn't be relevant when the era of combustion engines comes to an end, but there are so many other benefits apart from this. For example, the less time it takes for resources to travel from a provider to a manufacturer, the faster the production can start, so it also has a huge impact on the global economy.

On a global scale, traffic congestion is a prevalent issue in many major cities and urban areas. As urbanization and population growth continue, the number of vehicles on the roads increases, leading to higher traffic volumes and a greater likelihood of congestion.

Some regions with particularly notorious traffic congestion include heavily populated areas such as Tokyo, Beijing, Mumbai, Los Angeles, New York City, São Paulo, and many other major cities around the world. These cities often face significant traffic challenges due to their large populations, limited road capacity, and high levels of economic activity.

The frequency of traffic jams(1.1) can vary throughout the day and week. Rush hours, typically occurring during morning and evening commute times, tend to experience the most congestion as people travel to and from work or school. Similarly, certain events or holidays can cause increased traffic and congestion in specific areas.

Various factors contribute to traffic congestion, including inadequate infras-

Figure 1.1: Traffic Jam Example (Image Source ©)

tructure, insufficient public transportation options, inefficient traffic management systems, and a high reliance on private vehicles. Additionally, accidents, road maintenance, and adverse weather conditions can exacerbate congestion levels.

As its occurrence is noticed the most in urban areas, the problem is currently being tackled in concordance with the smart city concept. A smart city 1.2 is defined as an ultra-modern urban area that aims to improve the overall life quality of citizens. It is based on different architectural approaches that involve modernizing and upgrading our current environment by applying new concepts and technologies on top of the ones currently in use.

1.2 Smart City Concept

Smart city traffic management refers to the implementation of advanced technologies and intelligent systems to optimize the flow of traffic within urban areas. It involves the use of various sensors, data analytics, and communication networks to monitor, control, and manage traffic conditions efficiently. The goal is to improve traffic flow, reduce congestion, enhance safety, and minimize environmental impacts. Smart city traffic management often includes the following key elements: ITS, Data Analytics, Adaptive Traffic Control Systems, Integrated Traffic Management, Connected Vehicles, Multi-modal Transportation, Smart Parking Systems, Sustainable Transportation.

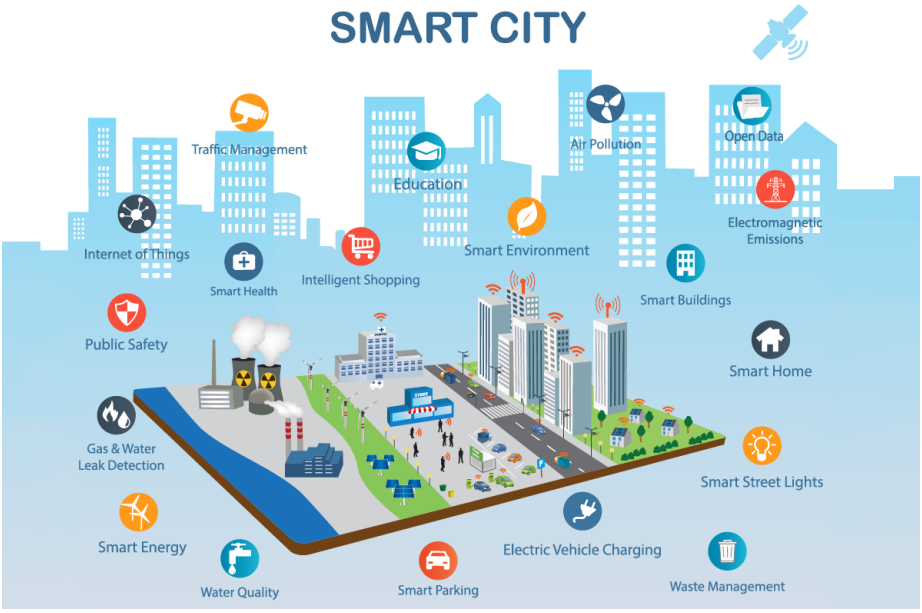

Figure 1.2: Smart city concept (Image Source ©)

1.3 Motivation, further goals and objectives

What determined us to write this thesis is the traffic that we experienced in Timişoara, Romania at the time writing. We believe that the traffic system here is based on traffic light synchronization systems due to the fact the most of the time, when you are able to catch the green light at a given junction, you will catch it for all the junctions that follow the given one. What is problematic and the main issue that we think bother most of the drivers are the rush hours, when traffic is so poorly managed that you will have to wait most of the time at least 2 or 3 or even more times for the green light at the same junction. This is due to the high amount of volume of traffic through the main routes. In this scenario the vehicles will no longer be able to travel at constant speeds and as a result the traffic itself will be asynchronized.

This is just one example of bad traffic condition, but this problem persists on a global scale, no matter the system that has been used to manage the traffic, there will always be one scenario when the traffic is completely jammed.

Our main goal is creating a system that would be able to not only reduce the occurrences of traffic jams but also adapt on those scenarios, and dynamically improve traffic flow. We want to let it adapt on each individual scenarios as we believe there isn't a way to precisely predict traffic. So every single one of our

systems will learn from the environment that is being installed on and evolve to the point where it perfectly suits that given city/junction/road system traffic conditions.

We also want to create an affordable system that can be improved if needed because we want to launch this system on a global scale, but not everyone has the financial power to support new infrastructural requirements. Along with the upgradability of the system we will also guarantee that if any of our components will fail, the system will be able to continue working and not cause drastic traffic disturbance. One thing that needs to be taken in consideration as well is the fact that some components can single-handedly manage traffic if the traffic volume is relatively low so in many scenarios you won't even need to upgrade the system.

One final goal is to make the system deployable on a global basis. For that we will need to develop a some kind of way to scale our system horizontally.

1.4 Topics Covered

In the next chapters we will discuss the following subjects: the technologies that already have been developed, the reason the current systems are not performant enough, a new way of handling traffic, the way we implemented the system and further directions.

We will start analyzing traffic management from the most antique ways all the way to the latest systems that have been developed at the time writing with focus on DSRC based systems huge potential of improving traffic flow. We will also describe the benefits/drawbacks of other systems that have been implemented through the time like object recognition-based systems, sensor based systems and many others.

The current flaws of the traffic management system will be highlighted and a new IPC system will be presented to fix the given issues. It will be based on already existing traffic system and has the possibility to be globally scaled and deployed. We will further talk of the importance of DSRC systems and the need of maintaining stable traffic flow even if our systems somehow fail.

After describing the system, we will present the implementation details, the model that we have trained for detecting vehicles, the way we handle IPC, describe the servers and clients responsibilities based on their type, the GPS data handling, and overall how things work within our systems.

Chapter 2

Overview of the Technology Developed

To better understand the problem, we must first know what fix attempts were used to minimize the negative effects of this issue throughout the time and how they evolved.

The first fix attempt that was put in use was created long time ago and required the use of actual human resource to control the traffic. Policeman were responsible to manage the traffic at junctions. Once with the technological era, we were able to remove the need of this kind of resource by creating autonomous systems that alternate between STOP and GO phases. The design of the systems was pretty simple and straightforward, we would use lights to signal those phases. Red lights were matched with the STOP phase, green ones with the GO phase. To prevent collisions and maintain drivers safety amber was introduced as well. It notifies the drivers that STOP and GO phases will soon switch, and they should take the corresponding measures.

As the time passed by and the number of cars on the road increased, drivers started experiencing traffic jams scenarios. Also there was clearly a better option then having a standard fixed time to change the lights. Many times the green signal was turned on even though there were no cars crossing that given road while on the other crossing roads there were actual drivers waiting. Something had to be done, a more efficient way had to be found, but at what cost? If we wanted to develop better solution we would need the help of expensive hardware components.

We reach the point in time when a new discovery was made: budget units based on microcontrollers, sensors and receivers [2] that were cable of running advanced algorithms (Figure 2.1). So all that was left was to develop new solution and use them to design an intelligent Traffic Light Control System (TSC)

By the time writing, TSC is an active research topic that proved to be really challenging. Continuous work has been done on designing and developing intelligent traffic signal control systems that could address the issue.

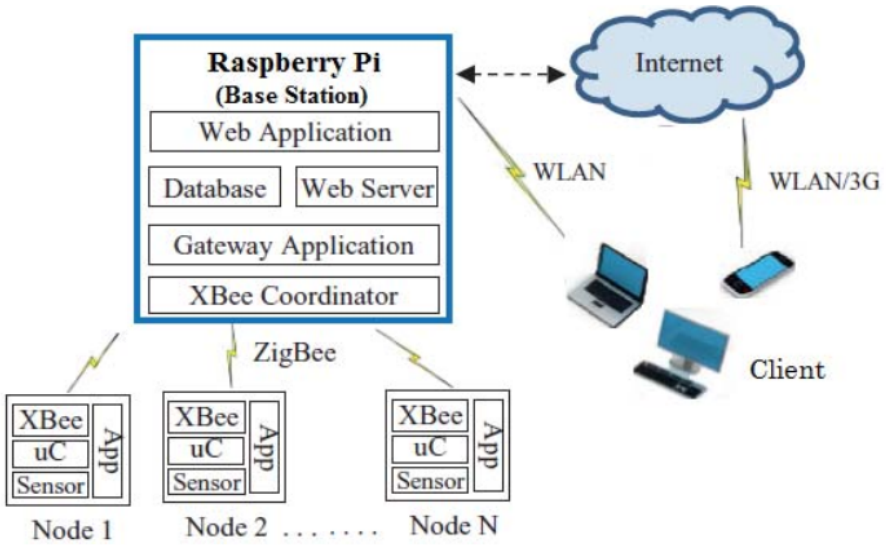

Figure 2.1: Raspberry PI based system (Image Source ©)

So new methods and advanced systems and many other algorithms have been proposed by the researchers to solve this TSC problem, mostly being based on fuzzy logic, evolutionary algorithms, image processing, neural network, etc. [7]. All the methods aimed to reduce the overall waiting time and prevent traffic jams from occurring while maintaining drivers safety. In this section we will discuss some of the methods that were put in to use and their ups and downs. After describing each and every technology we will have a brief comparison to highlight some of their characteristics.

2.1 Object recognition-based systems

One way to determine traffic scenarios was done with the help of cameras. To be more specific we wanted to calculate the traffic waiting queue by detecting cars present int the images provided by our hardware components (Figure 2.2). This problem was addressed by the help of vehicle tracking and image segmentation algorithms.

However, the techniques used to solve the problem proved to be ineffective in real-time scenarios due to its high computational complexity that made the system unable to keep up with the traffic flow especially at high speeds. Also, some are not even able to operate in bad weather conditions, so they were doomed to failure. Studies had shown that whenever visibility is reduced due to rain, fog or other external factors that the percentage of vehicles that will be detected will significantly drop. [5]

Even if the problems stated above were known, neural networks systems

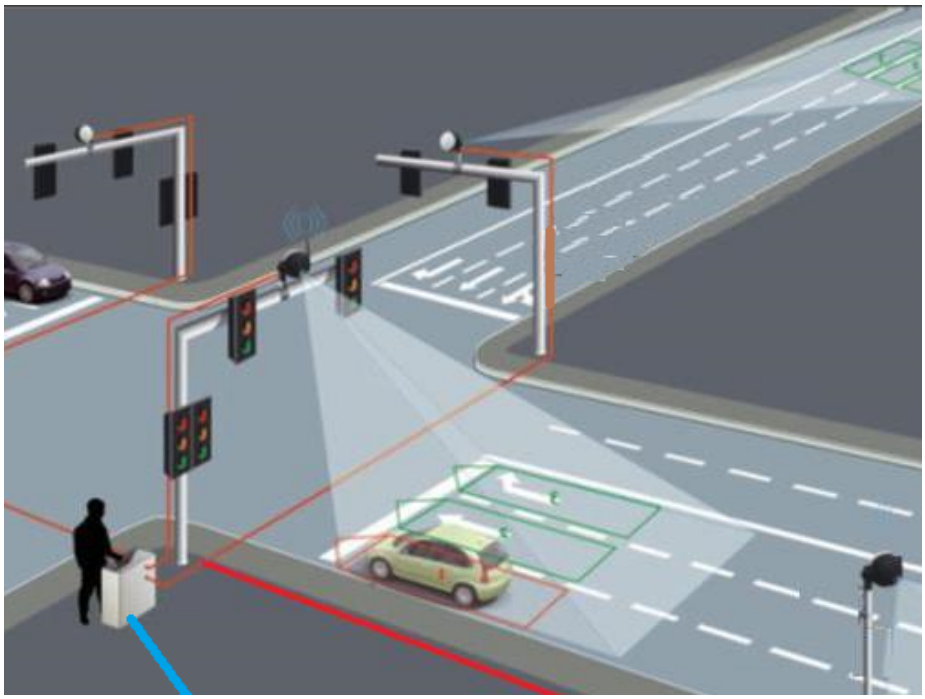

Figure 2.2: Object recognition-based Traffic System Model (Image Source ©)

have been developed with the help of object recognition algorithms. They were designed to predict the upcoming traffic volume from an X-min daylight traffic flow on different traffic conditions. Like so, we would be able to overcome the high computational complexity of the algorithm by avoiding recalculations. This lead to high memory needs, that could not be overcome, so this approach is unsuitable for real-time systems. Also, we believe that it is almost impossible to predict traffic on a constant manner, and our best way to handle it is to dynamically adapt to it. We can still make use of a model that pools all traffic states for detection, but the actual "prediction" of the next traffic state should be calculated dynamically from real-time data.

2.2 Vehicle sensor detection-based systems

Another approach would be to collect data about the vehicles approaching junctions with the help of GPS sensors. (Figure 2.3)

One way to do it is to track the position, speed and direction of the given vehicles. Sadly, this method would work only for a road network, it can not solely control the traffic signal timing for just one junction. This is due to the high speeds of travel and the time complexity of the main algorithm.

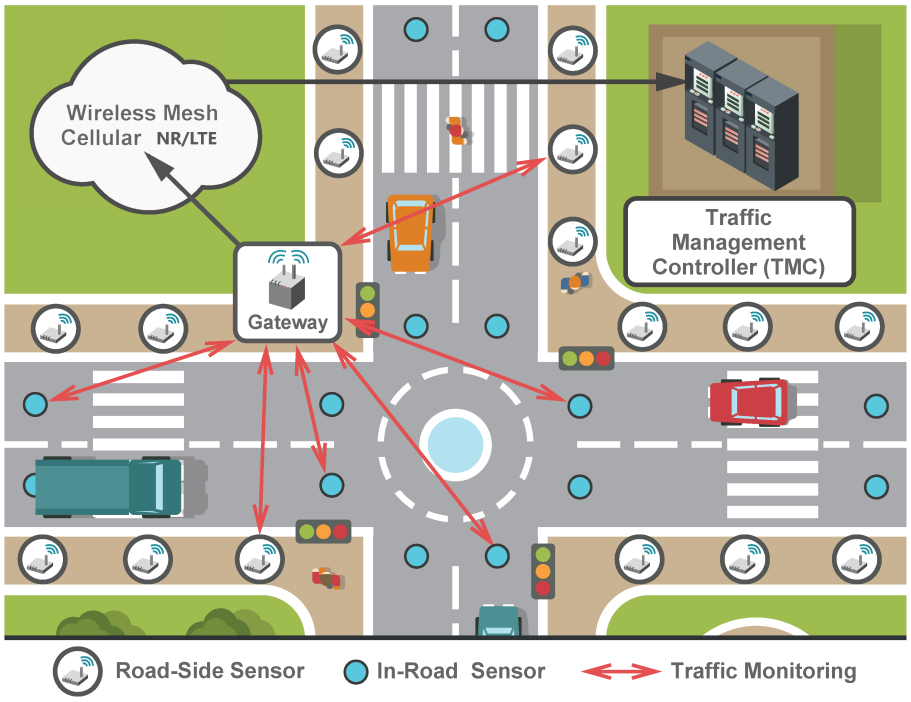

Figure 2.3: Sensor Based Traffic System Model (Image Source ©)

Another way to do it is to monitor the arrival and departure of vehicles at a junction. This can be done with the help of sensors and traffic servers. Like so, we could use embedded technology to record the GPS data and send it to the traffic monitoring system through GSM/GPRS. The drawbacks of this method are the fact that it involves very high implementation cost and, sadly, some vehicles can not be tracked using radio detection systems. This problem can be also approached with the help of in-road sensors, but it would require even higher costs as you would need to often change the sensors because roads need to be rebuilt due to wear or ongoing constructions periodically.

2.3 Traffic Lights Synchronization based systems

Traffic Lights Synchronization (TSC) [7] systems aim to minimize the number of STOP and GO occurrences by adapting the traffic light phases at junctions. This technique involves all vehicle maintaining a consistent speed while traveling on one side of the road, allowing them to continue indefinitely to the opposite end of the road by consistently encountering green lights at intersections by maximizing their number. Studies had shown that in comparison with other fixed time and non-synchronized traffic control strategies it reduces overall

travel time by up to 39%. [**ALEKO2019**]

Just like most of the others methods, it collects and process data from real traffic scenarios to determine the green light timer. When vehicles pass through a junction the green light timer for the following junctions decreases by a certain amount (Figure 2.4). As with the other roads present at the junction, the red light time increase to guarantee the continuous motion of already travelling cars. This can be done for multiple levels of traffic depending on how many junctions do you want to synchronize with one another, but must of the times it was handled as a 2 level system.

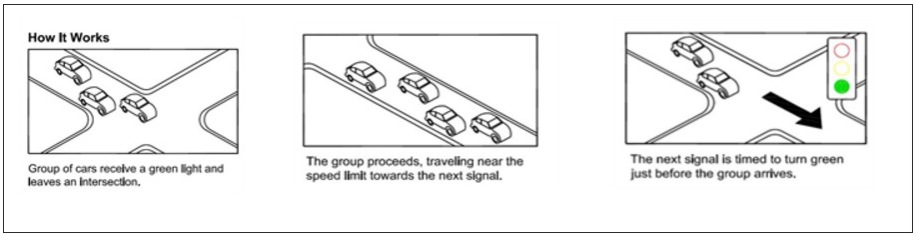

Figure 2.4: Traffic Signal Synchronization Model (Image Source ©)

Multiple approaches to implement this kind of systems were made with the use of various AI algorithms and already known data collecting methods but the result are mostly the same. This method would lead to longer red signal time, but will reduce the overall waiting time when travelling long distances. One other advance to keep in mind is that, as result of this phenomenon, the wear of the vehicles will be drastically reduced.

The main disadvantages of this method is the fact that you will eventually have to prioritize one route. This would be problematic if multiple main roads collide with one another. Also, the algorithm will not always provide an optimal solution as drivers following side roads may have longer waiting times. One more thing to keep in mind is that this method is reliant on drivers speed consistency. Most of the cases, there will be a decent amount of speeding vehicles that will not be able to pass multile junctions without any kind of stops as the algorithm itself does not intend to handle those edge scenarios.

2.4 Fuzzy Intelligent Traffic Signal Control System

Fuzzy Intelligent Traffic Signal Control (FITS) [6] [3] systems were originally designed to mimic a human policeman in controlling traffic lights at an intersection. They are systems that take different types of traffic input and apply fuzzy rules to manage the traffic. Like so data is converted into fuzzy truth values between 0 and 1. For instance, the green light extension time can be modeled by a number of fuzzy sets including “none”(1), “short”(2), “moderate”(3) and

"long"(4). The only thing that you are left to do is to define your membership functions for the given set. For example, we can use the following functions to do so:

$$"none" - f(q) = max(min((10 - q)/10, 2), 0) \tag{2.1}$$

$$"short" - f(q) = max(min(q/10, (20 - q)/10), 0) \tag{2.2}$$

$$"medium" - f(q) = max(min(q/5, (30 - q)/5), 0) \tag{2.3}$$

$$"long" - f(q) = max(min((50 - q)/10, q/2), 0) \tag{2.4}$$

Like so, we would alternate base on other fuzzy values or randomly, between choosing "none", "short", "moderate" and "long" fuzzy values, to determine the time that our green light should last.

Depending on the actual improvements of the traffic flow the algorithm will adapt, altering its member functions and decreasing/increasing the odds to chose one specific fuzzy value from the set. One big advantage that this would lead to is that FITS mitigates the negative effects of detection malfunction by predicting the traffic states at the whole intersection by simulating real time traffic scenarios. So FITS may be able to run without any kind of additional hardware, leading to huge cost advantages.

Despite all the benefits presented above, as with any AI based algorithm, it takes a lot of time for the algorithm to improve itself and the upgrade. Also, the chance to upgrade is very reliant on traffic conditions. Furthermore, even if the algorithm has been running for a long time, it still can choose a non-optimal solution that may or may not lead to traffic jams.

2.5 Dedicated Short-Range Communications systems

To explain why this kind of systems would work, first we have to understand what are Dedicated Short-Range Communications (DSRC) [8] [7]. DSRC are one-way or two-way wireless communication channels specially designed for use in automobiles that are mostly used by ITS to communicate with other vehicles or infrastructure technology. They operate on the 5.9 GHz band of the radio frequency spectrum and are effective over short to medium distances.

One technique that was developed with the help of DSRC is the Virtual Traffic Light (VTL) approach. VTL is a biologically-inspired approach to traffic control that relies on Vehicle-to-Vehicle (V2V) communication by using the Signal Phase and Timing (SPaT) message and Basic Safety Message (BSM) from the DSRC OBU. (Figure 2.5)

Figure 2.5: Dedicated Short-Range Communications Radio (Image Source ©)

The radio is capable of broadcasting several messages defined by the SAE 2735 [1] protocol with the most important being BSMs. This kind of messages contain vehicle's current information (GPS location, speed and heading) that are associated with a temporary ID. As a result, by broadcasting BSM messages we are able to detect the vehicles approaching the intersection in a continuous manner, unlike traditional methods that only detect the presence of the vehicle using loop detectors.

You can image the cars being "moving routers", and the junctions being "stationary routers". Whenever one "gateway" or in our case one traffic route is overfilled with vehicles it blocks the others and starts letting them pass just like RIP protocol. Like so, we are not actually introducing new sorts of technologies that may or may not fail, we just adapt already existing algorithms, that proved to be great solutions, to fit one specific use case.

With the help of this approach we can drastically reduce the implementation costs as it does not require any kind of additional expensive hardware components. Furthermore, it is robust against weather conditions and easy to maintain. Extensive simulations have shown that this kind of technology can reduce daily commute time in urban areas by more than 30%. Different aspects of VTL technology, including system simulation, carbon emission, algorithm design and deployment policy have been researched in the last few years. [4]

Figure 2.6: DSRC Based System Model (Image Source ©)

The main drawback of this approach is that it requires vehicles to support DSRC technologies, which might be problematic at the given time. Moreover, within V2V communications in VTL systems, there is a possibility of encountering partial obstruction by physical object present in the innermost Fresnel zone (non-LoS conditions). This presents a significant challenge in terms of making quick decisions effectively. However, even if DSRC technology isn't totally supported by vehicles, and we might collide with non-Los conditions, it proved to be effective in reducing the average waiting time at junctions. So, maybe the next step on improving traffic flow, would be to create an infrastructure for this kind of technology, but only time will tell. In the meantime the best thing we can do is to develop an economically efficient plan for transitioning from existing traffic control systems to VTL systems.

Chapter 3

A new flexible economic approach

We believe that the future of traffic light management will be based on DSRC like signals. No matter if it will still be based on DSRC, 5G or even 6G will take lead, we aim to provide a some kind of software system that would be able to handle any kind message based technology. Also, because at the given moment, the infrastructure for DSRC systems hasn't been fully developed and deployed we want to provide an economic approach to the given problem that would benefit on the already working systems, but also can simulate the already working traffic light management protocol.

Even if DSRC will be fully deployed on a global basis, there will still be a lot of poor countries that will have a hard time upgrading their infrastructure. Because of that, we wanted to be able to have a working system even without investing any kind of money. Think about a PC for example, even one of the worst ones can run an OS and if you want to upgrade them, to make it run faster or handle a newer software, you would just add or replace a component with a better performing one. Not only you will be able to add "performance boosts" as needed to your given infrastructure but think about the scenarios when something goes bad? What if one of your component goes down and for example the traffic light will be stuck on red until someone comes and fixes it. That would lead to a HUGE traffic jam.

We also want to be able to take advantage of already working DSRC compatible vehicles without adding any kind of new hardware and be able to disable the additional protocols and move to a fully DRSC based system if needed.

Sounds great, right? This is how we designed it. The system will have **4 major components**: **2 types of clients**: one stationary one and one moving one, "attached" to our vehicles; **2 types of servers**: one that would manage the position of the clients and redirect them to the corresponding servers and one that would handle all the messages and change the actual traffic lights. For

the simplicity on things we will name them as follows:

- **Traffic Observer (TO)** - the stationary client.
- **Vehicle Tracker (VT)** - the moving client installed on our vehicles
- **Junction Main Server (JMS)** - the server that handles the traffic lights
- **Proxy** - the server that will guide the clients to the corresponding servers

The system would work as follows: We would constantly have a TCP-IP connection between our clients and the corresponding JMS. Once they are connected clients would send various data about the traffic that would be further processed by the JMS and traffic lights would be changed accordingly. The TO would be responsible for sending data about the traffic on its road and the VT would be responsible for sending its own data to the server (Figure 3.1). We also handled the cases when there is an emergency ongoing like any kind of special vehicles crossing the junction, but we will talk later about this because it requires some more attention as it can be easily exploited if not handled right. Just imagine any kind of cars being treated like an ambulance on duty, you would just be able to pass freely any kind of junction without ever waiting at the red light.

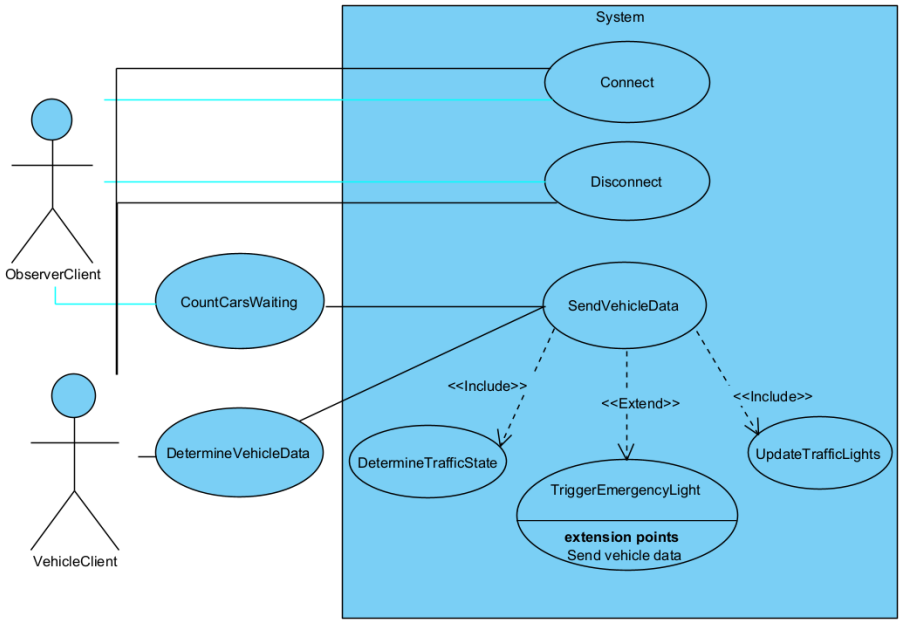

Figure 3.1: All UC Diagram

3.1 Proxy

The first problem that needs to be addressed is the fact that VTs will constantly have to switch between JMSs, but how do we know what server to connect to? We do not want to connect to each and every server in the covered area and start broadcasting mostly useless message. The answer would be that each client will know the main server off the system, that would act just like a proxy and redirect them to the corresponding JMS. For that we would need to store the JMS coordinates in one or multiple database. The VT would need to just send its own coordinates and the direction they are heading, as a result the proxy would provide the IP address of the corresponding JMS (Figure 3.2).

As we want this to be scaled on a global level, the system supports horizontal scaling, as we can just stack proxies one on top of the other and instead of having stored only the JMS coordinates we will also store all the proxies that are in the subarea of the owner. To not overload any kind of servers we also implemented a load balancing mechanism that prevents this kind of issue.

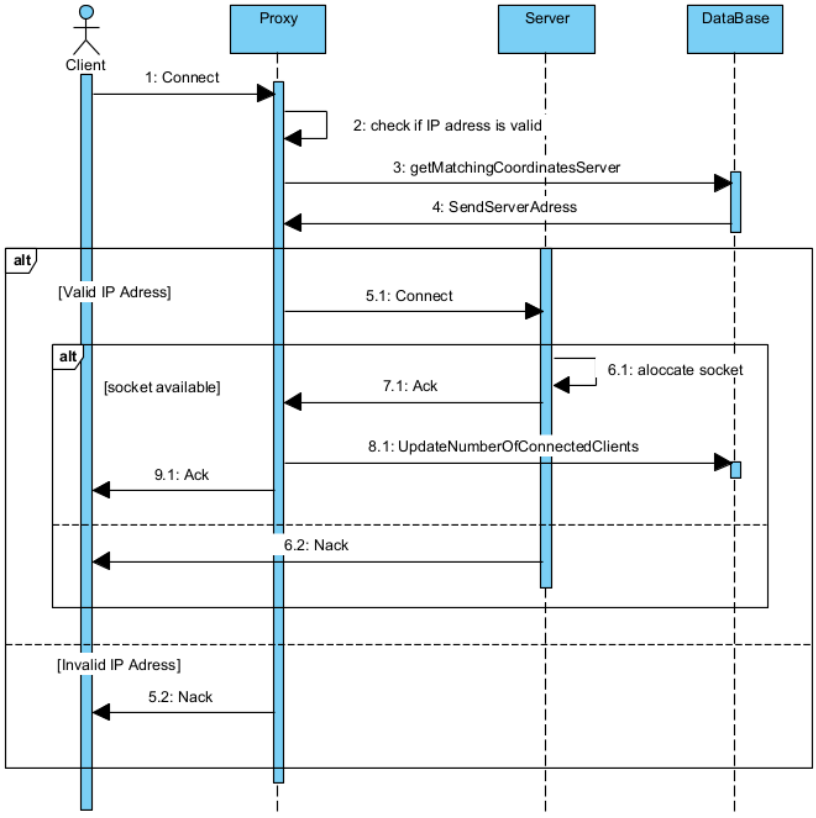

Figure 3.2: UC: Connect

One thing this can lead to is a better surveillance of the vehicles that aren't supposed to be on the road in the first place. The IP address of a given vehicle can be "banned" and if it tries to connect to the JMS the message will be ignored and not only the JMS will not count the waiting vehicle, sometimes leading to longer waiting times for the individual that is driving the vehicle and broke the law, but also like this we would have the proof that he broke the law, so we would be able to further punish him. We can lower the number of cars driving on the public roads that do not respect safety norms and reduce the number of accidents. For this to work we will need to also disallow owners to change their hardware without any kind of approval as a new radio would lead to a different serial number and this would be problematic.

Once passing the area of coverage of that given junction we would also need to disconnect from that given JMS (Figure 3.3). By doing this we will be able to always track the vehicles and react accordingly when one or multiple leave the junction. One thing that we have to mention is that if we further think about this we could in theory keep a constant track of ALL the vehicles ON A GLOBAL BASIS and no longer require license plates, if, of course all the vehicles would support sending DSRC signals.

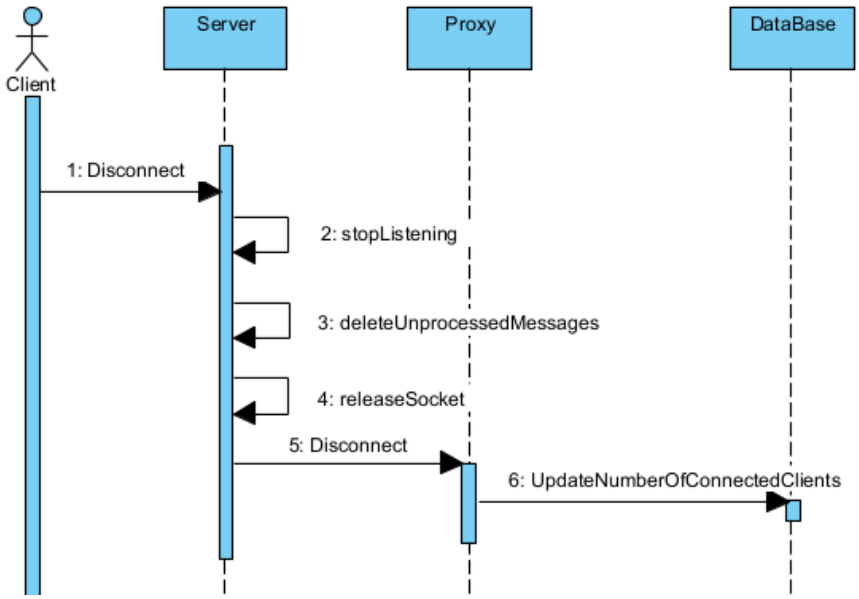

Figure 3.3: UC: Disconnect

3.2 Traffic Observer

The second thing we need to do is provide a way to detect the waiting vehicles without the help of DSRC messages. This can be done in multiple ways, but for the simplicity of things and to keep costs as low as possible we went with a camera based system. The way it works is as follows: multiple cameras are mounted on the road that are connected to microcontrollers, we take the images provided and apply object recognition algorithms to detect the passing cars.

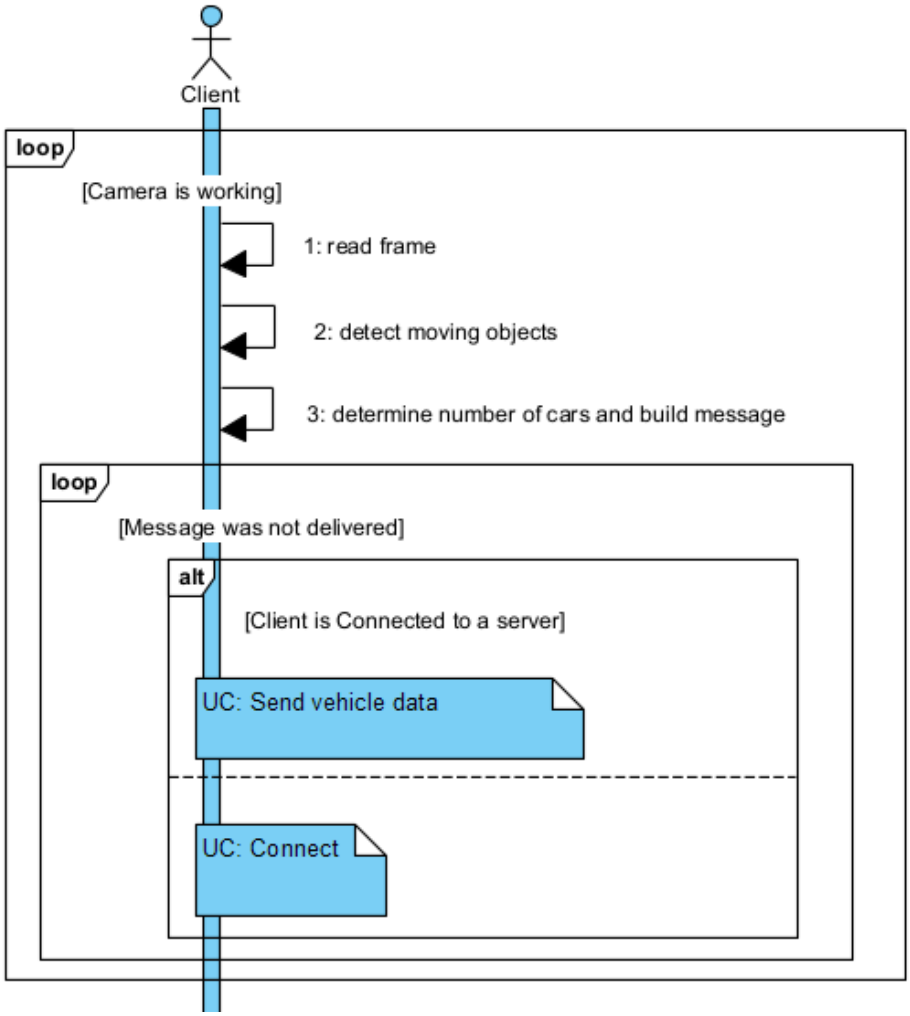

Figure 3.4: UC: Find cars from provided images

The first thing we actually do is preprocess the given images and detect the moving objects inside our frames and second of all determine if that moving object is or is not a vehicle (Figure 3.4). Like this we can speed up the image detection process as we will have fewer pixels to keep track off. This overall speeds up the system as can be seen in Figure 3.2. The blue line represents the amount of cars detected with it running and the red one without it.

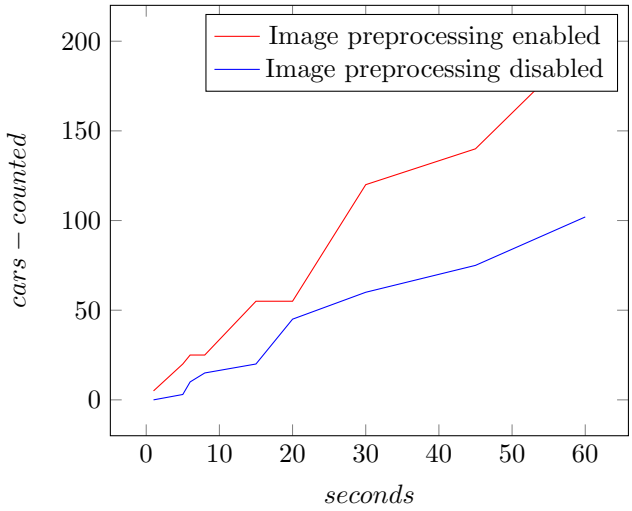

Figure 3.5: Car tracking performance

3.3 Vehicle Tracker

We want to be able to be able to send DSRC signals from any given vehicle. To do that, we make use of already integrated hardware components, to be more specific we make use of the radios and GPS installed on our vehicles. Like this, we get the coordinates of the vehicle as well as its heading, and the radio allows us to establish TCP-IP connection and send messages. So the only things that is left to do is to download our service and keep it running while travelling.

Whenever a vehicle wants to connect to a junction, it will have to specify its serial number, like this we will be able to deny/accept its connection. The only tricky part that is left to do is prevent anyone from changing their serial number. This can be done in many ways, like changing the ACL of the executable to be ran or overwritten only by the system or simply putting a flag in memory to prevent any kind of access in the location where the executable itself is stored or just encrypt the whole storage space of the vehicle. The security part of the system itself is not yet defined, it will be developed only if the project will gain momentum.

3.4 Junction Main Server

To handle all of those messages and change the actual flow of traffic we need to have a main command unit that acts as a server. Now that we know about all the components of the system we can start visualizing it(Figure 3.6). Clients will query the 2 time of servers and connect their corresponding junction. Once connected, they will send various type of data to the junction server and the server itself will change the traffic phases based on the data recieved.

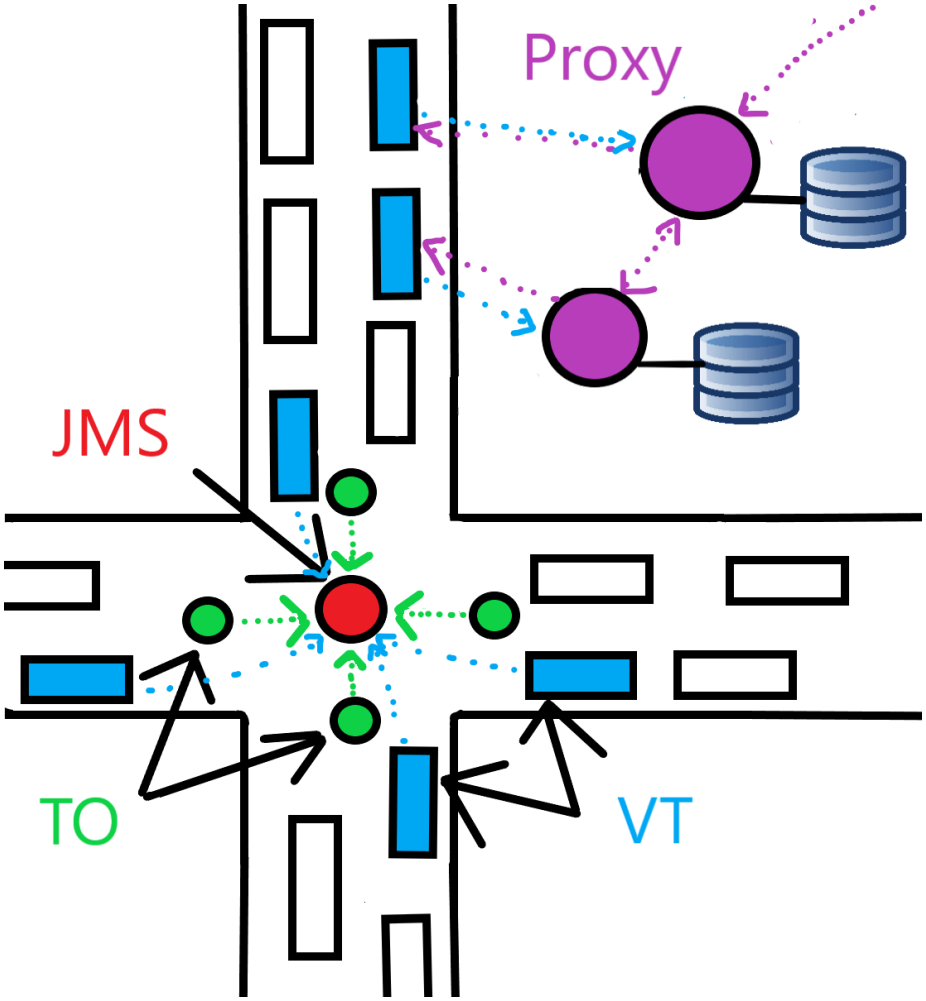

Figure 3.6: System sketch

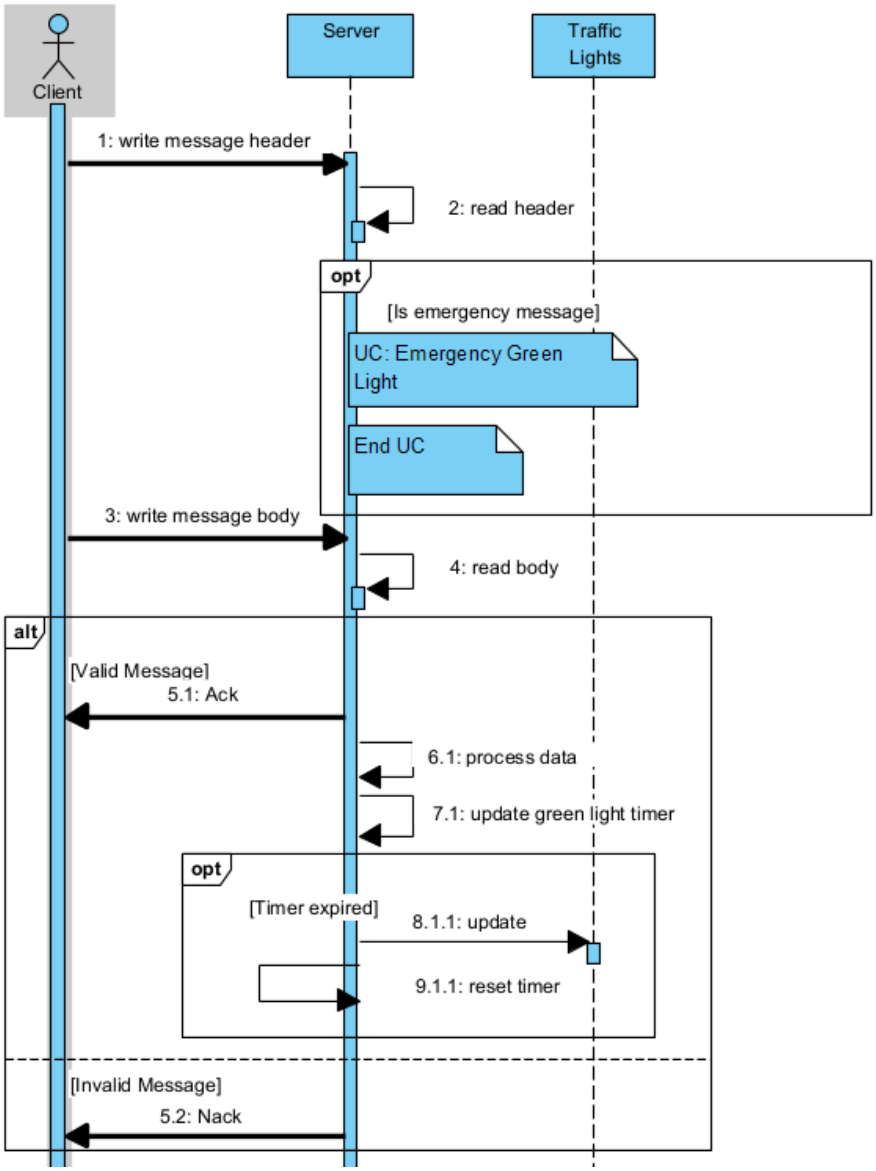

Figure 3.7: UC: Send vehicle data

The mechanism works mostly like any other TCP-IP mechanisms. We read the header of the message to determine the type of message received and the size of the actual message then based on the info we process the data and update the traffic state. You may ask what happens if a car is detected by the TO as well as its data is sent to the JMS through a VT. Well, because of this scenario we chose to approximate the traffic based on the number of cars determined by the

TO and the number of VT that sent messages to the JMS. Like this we make sure that, if one of the 2 components or both fail the system will still work. Also, because no one is supposed to wait to long for the green light we can set a maximum timer for the red light. Imagine being alone on one road waiting for the green light and having thousands of cars crossing the intersection from another road, you would have to wait ages for you to be able to cross. This prevents that and acts just like an "aging mechanism" (Figure 3.7).

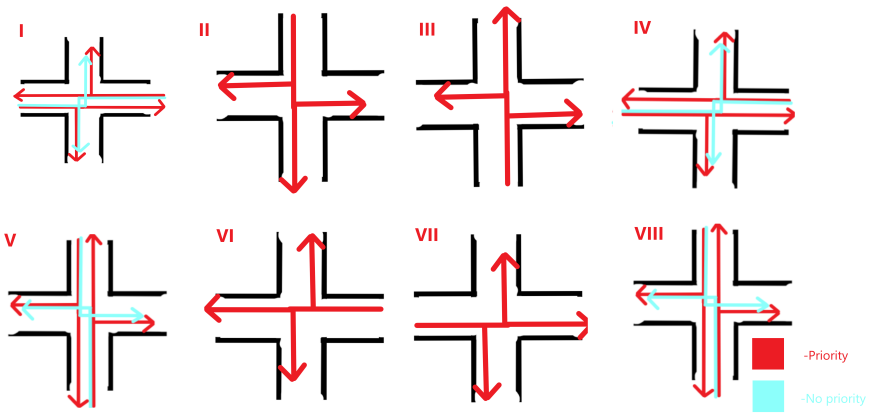

Figure 3.8: Junction phases

As far as the main algorithm for the traffic management goes we have 8 phases of the traffic (Figure 3.8). The most basic scenario is that the traffic will follow that 8 fazes going through fazes 1 to phase 8 in order and cycle. That's the implementation for the way the traffic works, but we want to also be able to jump from one phase to any other at any given time, to be able to shortcut the system. For that we are basing on several things.

First and foremost we need to take into account the number of cars waiting. We do that by combining the provided data from the VT and TO and calculate an average of cars waiting.

Second of all we will need to fully understand our traffic fazes, why the are required and how to determine the phase we want to switch to. For that we first described the phases based on the acting vehicles.

- PHASE I: E + W waiting vehicles
- PHASE II: N waiting vehicles
- PHASE III: S waiting vehicles
- PHASE IV: E + W waiting vehicles (same with phase I)
- PHASE V: N + S waiting vehicles

- PHASE VI: E waiting vehicles=
- PHASE VII: W waiting vehicles
- PHASE VIII: N + S waiting vehicles (sane with phase V)

We want to know when to switch from one phase to another. For that we will have timers for each direction: E, W, N and S. At first all the timers will be set to a costume amount, but the waiting time will be changed dynamically based on the traffic conditions as well as the green light waiting time.

To start the whole mechanism we want to initiate the traffic as usual, having the normal order of fazes running and for us to switch to and abnormal phase we'll just need one or more timers to expire. The check for the jump to an abnormal phase will be done ONLY when GREEN LIGHT ENDS.

Last but not least we need to define the way our algorithm jumps from one phase to another, for that we will use the following example (Figure 3.9):

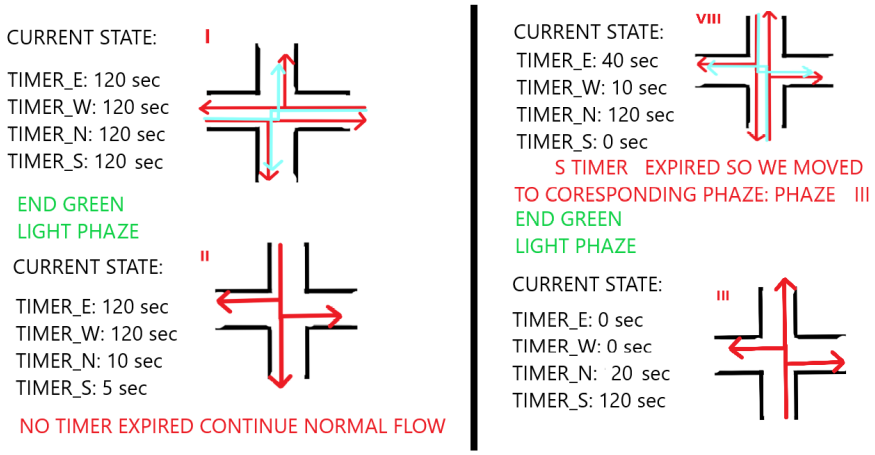

Figure 3.9: phase switching example

At the end of every green light phase we check the timers. If any of them expired we jump to it's corresponding phase. If, for example the following state corresponds to the phase we want to jump to, we do not short circuit the normal flow of things. Every time we are trough green light phases the timers corresponding to each lane that has a green signal is frozen and afterwards reset.

There are also "traffic conflicts" that can occur, for example, what if both the timers from E and S lanes has expired, we can not give a green light to both lanes, there isn't a corresponding traffic state. Those are the edge scenarios that we want to avoid the most. The way we handle them at the given moment is that we just use FIFO logic, take the timers that expired first and build an existing traffic phase. When this happens we afterwards update the green and red light timer's duration accordingly to prevent anymore occurrences. An example of the given scenario can be seen in Figure 3.10.

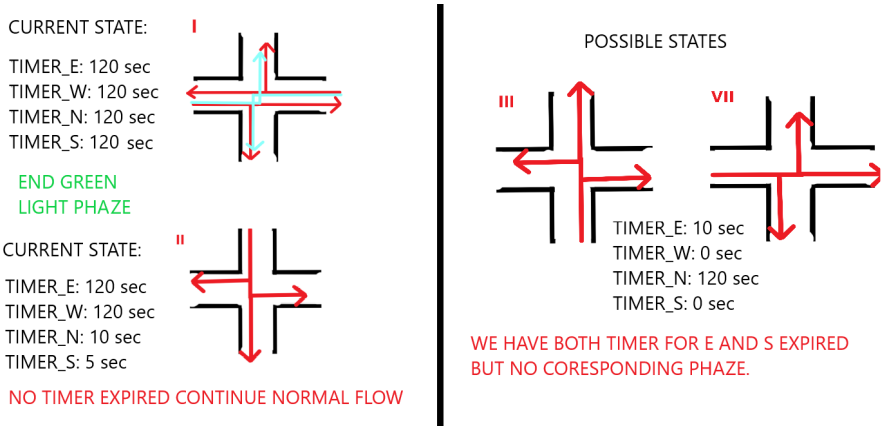

Figure 3.10: Faulty phase switching

3.5 Emergency states

The system also recognizes and treats accordingly special vehicles like ambulances, police cars and fire trucks crossing the junctions. Whenever one of those vehicle is detected our server turns on the green light corresponding to the lane they are following and keeps it on until it crosses the junction. This is called and emergency green light state. If we are already in one to begin with, we will queue the next state by using FIFO rule. The whole communication is done only by using VTs as the process of detecting special vehicles in mission would take too much time when it comes to object detection algorithms and time is key for us in those critical moments. When no more emergency vehicles are present, traffic will just go back to its original flow, from where it left off.

Chapter 4

Implementation details

The whole system was developed using C++17, Python, Boost, OpenSSL, GLFW, MSVC WinAPI, OpenCV, Tensorflow, YoloV8 and MySQL. The system itself is treated as a big project and split into multiple submodules:

- static librarys
  - Common
  - Car Detector
  - IPC
  - GUI
- executables
  - servers
    - Proxy
    - Junction Main Server
    - Object Detection Server
  - clients
    - Vehicle Tracker
    - Traffic Observer
  - testing

This section aims to explain in detail what each one of these submodules were meant to do, how they were implemented some examples of using them and the main features that they provide. The project is not cross-platform, it's Windows oriented at the moment because during the development of the whole system Microsoft Visual Studio 2022 MSVC was used as it provided a really quick and easy way of debugging my applications, but the code itself can be later converted to be cross-platform using CMake, apart from the testing module, as it is using WinAPI to spawn and handle processes.

4.1 Common

The main idea of the common submodule is to act as a helper library. It provides solutions to the following commonly found problems:

Multi threaded concurrency: Thread safe structures:
- Thread Safe Priority Queue
- Thread Safe Queue

Handling GPS output: NMEA coordinates data parsing:
- GPGLL, lat, N/S, lon, E/W, time, A/V, A/D/E/M/N * checksum
- GPGGA, time, , N/S, lon, E/W, Position Fix Indicator, Satellites used, HDOP, MSL Altitude, Units, Geoid Separation, Units, Age of diff. corr., Diff. ref. station ID * checksum
- GPRMC, time, A/V, lat, N/S, lon, E/W, Speed over ground, Course over ground, Date, Magnetic Variation, A/D/E * checksum

MySQL interactions: C++ class table encapsulation and query handling

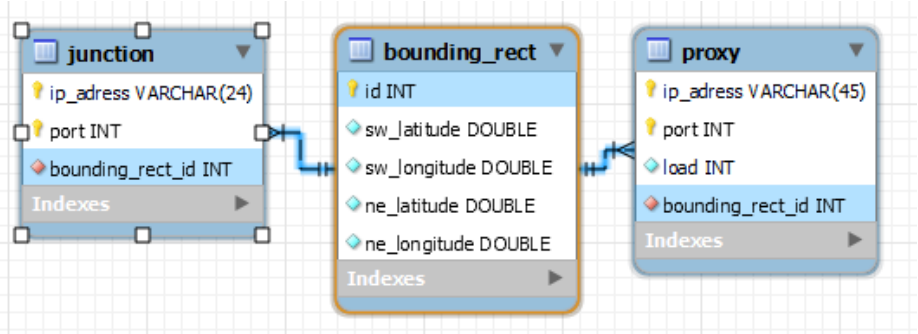

Figure 4.1: Database Schema

Parsing config files: Specific JSON parser, data type converter

Creating preplanned timed actions: Observers, Timers

Handling user input command line parser, signals handler

Synchronously logging on multiple threads: Customizable logger

4.2 IPC

To establish the communication among all the executables a costume made IPC system was developed, while using Boost Asio for socket handling. The logic itself runs on 4 threads: one for keeping the context running, one for reading, one for writing and one for notifying progress.

Listing 4.1: Client template

```
template<typename T>
class Client
{
private:
    ThreadSafePriorityQueue<OwnedMessage<T>>
        incomingMessages_;

protected:
    boost::asio::io_context context_;
    std::thread threadContext_;
    std::mutex mutexUpdate_;
    std::mutex mutexGet_;
    std::shared_mutex mutexConnection_;
    std::condition_variable condVarUpdate_;
    std::unique_ptr<Connection<T>> connection_;
    std::atomic<bool> shuttingDown_;
    boost::asio::io_context::work idleWork_;
    std::unique_ptr<IClientDisconnectObserver<T>>
        observer_;
    LOGGER("CLIENT");
public:
    bool connect(const IP_ADRESS& host, const PORT port);
    void disconnect();
    bool isConnected();
    void send(const Message<T>& msg);
    bool waitForAnswear(uint32_t timeout = 0);
    std::optional<std::pair<OwnedMessage<T>, bool>>
        getLastUnreadAnswear();
}
```

The client has 4 main methods, all of them being run synchronously: "connect", "disconnect", "send", "waitForAnswear". Every time a new connection is established with the server a new "Connection" object will be created that will be responsible for reading, writing and providing messages. To be able to send request over the network we use boost asio context that is initially provided with idle work so that it doesn't go out of scope before sending/receiving messages. The actual waiting for the messages is done through the use of unique locks and condition variables so that busy waits are avoided.

Listing 4.2: Connection template

```
template <typename T>
class Connection
: public std::enable_shared_from_this<Connection<T>>
{
protected:
    const Owner owner_;
    std::thread threadRead_;
    std::thread threadWrite_;
    std::mutex mutexRead_;
    std::mutex mutexWrite_;
    std::condition_variable condVarRead_;
    std::condition_variable condVarWrite_;
    std::condition_variable& condVarUpdate_;
    boost::asio::io_context& context_;
    boost::asio::ip::tcp::socket socket_;
    ThreadSafePriorityQueue<OwnedMessage<T>>&
        incomingMessages_;
    ThreadSafePriorityQueue<Message<T>>
        outgoingMessages_;
    Message<T> incomingTemporaryMessage_;
    std::atomic_bool isReading_;
    std::atomic_bool isWriting_;
    std::atomic_bool shuttingDown_ = false;
    uint32_t id_; std::string ipAdress_;
    std::unique_ptr<IClientDisconnectObserver<T>>&
        observer_;
    LOGGER("CONNECTION-UNDEFINED");
private:
    bool readData(std::vector<uint8_t>& vBuffer,
        size_t toRead);
    void readMessages();
    void writeMessages();
public:
    Owner getOwner() const;
    bool connectToServer(const results_type& endpoints);
    bool connectToClient(uint32_t id);
    void disconnect();
    bool isConnected() const;
    void send(const Message<T>& msg);
}
```

The connection is responsible for reading and writing messages. It acts like a middle point between the client and the server. It shared a queue of incoming messages and a condition variable with any of the other to classes so that it can notify the owner that a new message has arrived.

Listing 4.3: Server template

```
template<typename T>
class Server
{
protected:
    ThreadSafePriorityQueue<OwnedMessage<T>>
    incomingMessagesQueue_;
    boost::asio::io_context context_;
    std::thread threadContext_;
    std::thread threadUpdate_;
    std::condition_variable condVarUpdate_;
    std::mutex mutexUpdate_;
    std::mutex mutexMessage_;
    boost::asio::ip::tcp::acceptor connectionAccepter_;
    common::ThreadSafeQueue<uint32_t> availableIds_;
    std::map<uint32_t,
        std::shared_ptr<Connection<T>>>connections_;
    std::atomic<bool> shuttingDown_ = false;
    LOGGER("SERVER");
private:
    void update();
public:
    bool start();
    void stop();
    void waitForClientConnection(); //ASYNC
    void messageClient(
        std::shared_ptr<Connection<T>> client,
        const Message<T>& msg);
protected:
    virtual bool onClientConnect(
        std::shared_ptr<Connection<T>> client);
    virtual void onClientDisconnect(
        std::shared_ptr<Connection<T>> client);
    virtual void onMessage(
        std::shared_ptr<Connection<T>> client,
        Message<T>& msg);
}
```

The server is running on 2 threads, one for handling the boost asio operations and one for processing the incoming messages. It has a list of already connected clients and the connection itself the only operation that is done asynchronously. It has 3 main methods that can be overwritten that define the actual behavior of the server: "onClientConnect", "onClientDisconnect", "onMessage" that are actual callbacks to IPC events. If, for example, and integrator would want to use our IPC framework all that he would need to do would be do overwrite those given callbacks.

Listing 4.4: Message template

```
template <typename T>
struct MessageHeader
{
    T type{};
    uint16_t id{};
    bool hasPriority = false;
    size_t size = 0;
};

template <typename T>
struct Message
{
    MessageHeader<T> header{};
    std::vector<uint8_t> body;

    size_t size() const;
    void clear();
    Message<T> clone();

    friend std::ostream& operator << (
        std::ostream& os,
        const Message<T>& msg);

    template<typename DataType>
    friend Message<T>& operator << (
        Message<T>& msg,
        const DataType& data)

    template<typename DataType>
    friend Message<T>& operator >> (
        Message<T>& msg,
        DataType& data)
}
```

To define our message structure we also created a message class. The header of message contains its size, if it has priority(is from an emergency vehicle) and the type of message that is being sent as well as and ID. The data saved inside our body is in byte format, and we defined costume addition/extraction operations for any type of data. To do so we just memcopy the contents of the data as bytes and push them to the end of our vector. Also, the extraction can be done only from the back of the vector. So if for example we add and variable A and variable B to the message in this specific order we would need to first read variable B then variable A.

4.3 GUI

To be able to visualize the actual data, a GUI module was created. It is only for demonstrative/testing purposes, and it can be disabled using a flag. To achieve this we used GLFW, to be more precise we pre drawn the junction and linked to our actual timers. When it comes to the cars we had a queue of waiting actions and whenever a car was detected by the JMS, the action of drawing was queued. The action itself will queue another action if the car hasn't reached the end of the screen. Because we also wanted the scene to act just like a junction, whenever we were about to cross the junction when drawing, we would stop and delay the action if the red light persisted.

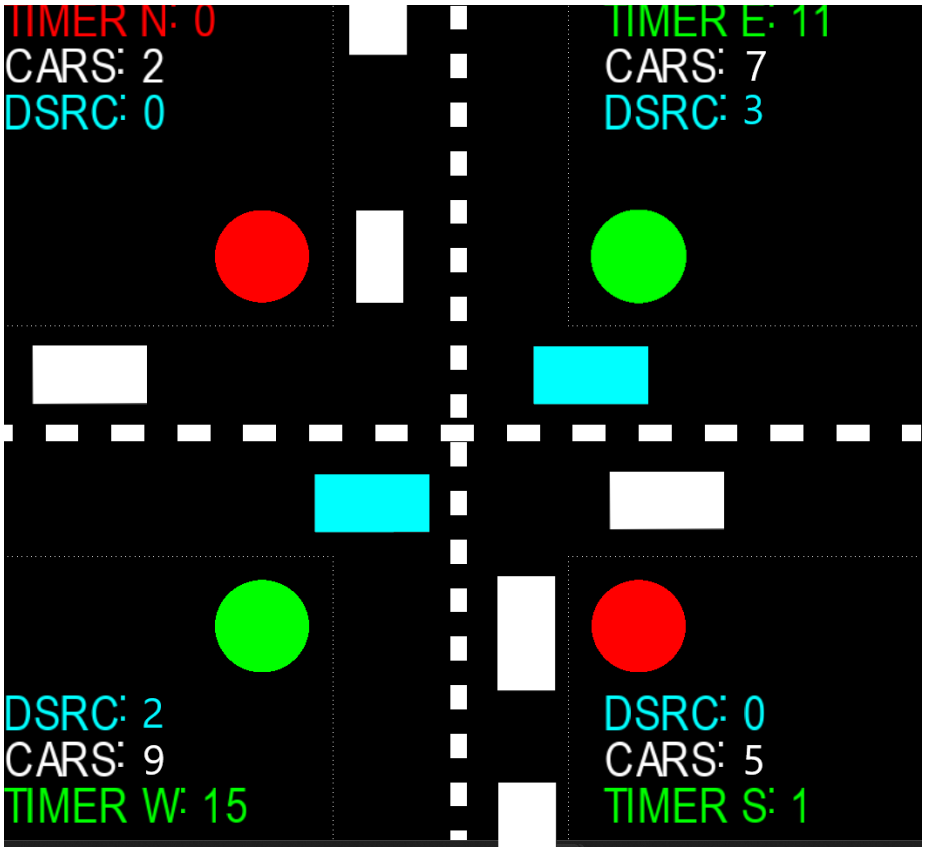

Figure 4.2: GUI JMS

In figure 4.2 the cars that support DRSC are marked with blue and the cars detected by the TO are marked with white. For each lane there is an associated description of the current state with the corresponding timer as well as the number of vehicles waiting to pass the junction. Due to limited space, we decided to allow vehicles to overlap and just add to the number of cars waiting.

4.4 Car Detector

To be able to connect to the road traffic cameras and count the number of incoming/outgoing vehicles wrappers over OpenCV library were made. The whole process works as follows: it starts detecting the moving objects inside a give frame, try to predict their next position and once they cross a imaginary lines inside the picture they are accounted if they were matched to be cars. To improve changes of actual registering the passage of a car, we considered cars punctiform objects by only taking into account the center point of the bounding rect as actual cars.

To boost performance, the module runs on 4 threads: providing images from the camera, detecting moving objects, classifying objects as cars and displaying the bounding boxes. The last one can be disabled, as it is just for illustrative purposes.

The car classification itself is not done by the given module due to current C++ Tensorflow - OpenCV outdated compatibility, but by the Object Detection Server written in Python. The Car Detector module just acts like a client and sends the actual bytes of the cropped image of the detected moving objects to the server. It is important to note that if, in the near future, the compatibility issues will be fixed, it would serve as a great performance upgrade to move this logic inside the module and remove any kind of process interactions.

4.5 Testing

To be able to test our whole system we created an executable that would emulate the traffic conditions and run our system. The executable takes multiple config files as input and is able to run multiple servers (JMS/proxies) and multiple clients at the same time. It is highly customizable as you can choose to run any kind of combination regarding the number of each client/server. To emulate the movement of the moving clients we generate GPS data with the help of NMEAGEN as for the actual camera input multiple videos were collected.

We make use of WinAPI functions for handling processes so sadly the module itself can only be run on Windows platforms. The way we do it is by parsing each and every config provided and spawning process accordingly. For example, for VTs we have a set of files containing GPS input. Every 1-3 seconds we spawn a new VT with a randomly picked input to emulate the passing of a car.

To be able to collect actual data we made a plugin that will periodically register the number of vehicles that passed as well as the average time waiting at the junction.

4.6 Object Detection Server

To be able to detect the presence of cars we made tried using 2 different architectures: one based on YOLOv8 and one using Tensorflow Object Detection API, while using the same dataset for training.

YOLO, short for You Only Look Once, is a popular real-time object detection system. It revolutionized the field of computer vision by introducing a unified approach to object detection that combines object localization and classification in a single neural network. The YOLO algorithm divides an input image into a grid and predicts bounding boxes and class probabilities for each grid cell. These predictions are based on anchor boxes, which are pre-defined bounding box shapes of different sizes and aspect ratios. YOLO predicts the offsets and sizes of anchor boxes relative to the grid cells, as well as the probabilities of different classes for each bounding box.

After training our model we achieved a precision of 75% but the detection, proved to be to slow, so we further looked for another architecture that could fit our objectives.

The architecture that we settled on was the one that was provided by Tensorflow Object Detection API. It is a powerful framework provided by Google's TensorFlow library that facilitates training, evaluating, and deploying object detection models. It offers a collection of pre-trained models and tools for building custom models to detect and localize objects in images and videos.

Due to the fact that the model will be loaded by a microcontroller a SSD MobileNet model was trained by using a public dataset. SSD MobileNet is a combination of two popular deep learning architectures: Single Shot MultiBox Detector (SSD) and MobileNet.

SSD is an object detection algorithm that provides real-time object detection in images. It achieves this by predicting the bounding boxes and class labels of multiple objects in a single pass through a neural network. Unlike other object detection methods that require region proposal generation, SSD directly predicts object detections at different scales and aspect ratios.

MobileNet is a lightweight convolutional neural network architecture designed for efficient use on mobile and embedded devices. It utilizes depth wise separable convolutions, which separate the spatial filtering and channel-wise filtering operations, reducing the computational complexity and model size. MobileNet achieves a good balance between accuracy and efficiency, making it suitable for resource-constrained environments.

By combining SSD and MobileNet, the SSD MobileNet model achieves real-time object detection on mobile and embedded devices with limited computational resources. It provides a good trade-off between accuracy and efficiency, making it a popular choice for various applications such as autonomous vehicles, surveillance systems, and mobile applications requiring object detection capabilities.

The training was done with the use of Tensorflow repository, and it took about 15h on an RTX 3060 and the model is capable of detecting cars, streetlights and other traffic related objects, but it will be used just for detecting

cars. The model speed is the most important thing here being able to evaluate 22 frames/s, as its accuracy is not that great averaging 60%. The overcome the low accuracy any object that is more than 50% likely to be a car will be taken into account.

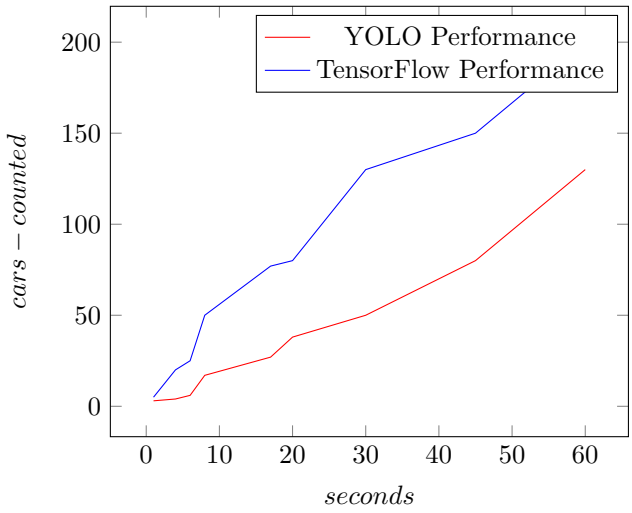

Figure 4.3: Comparison Tensorflow-YOLO

A comparison between the speed of the Tensorflow model and YOLO can be seen in figure 4.6 the blue line representing the amount of cars detected during a given period of time for the Tensorflow model and the red line represents the same ratio but for YOLO. It is clear that the speed performance of the Tensorflow model is superior.

The model itself is loaded just once on a server, because it is a really time-consuming operation, and used whenever a new valid message is received. Once the predictions are calculated, we remove the overlapping objects detected with the help of OpenCv NMSBoxes and return just the size of the remaining list of objects, not the bounding boxes themselves, representing the amount of cars detected in the given image. To keep the consistency of the IPC implementation I imported the C++ message definitions and implemented the created a simple implementation of a server using socket package. To further improve the response time of the server, the request handling can be done in an asynchronous manner, having multiple threads working on different messages for the same client.

Related to the actual device that will run the server, it is better to just have them tied down to every proxy, due to the high resource need of loading the actual model that can not be achieved on the microcontrollers.

4.7 Proxy

The proxy is a wrapper over the server implementation that assures the connection between the vehicle trackers and the next junction main server they will encounter. Every single one is connected to their own database, that contains a list of junctions, proxies and the area covered by them.

They way it works is pretty straight forward, if a client queries the proxy for the next junction it searches the database. If it finds a suiting junction then it sends the connection data, otherwise it redirects the vehicle to the closest proxy. This process repeats itself until it finally reaches one valid junction or the car is out of coverage area (for example if the car is somewhere on a ship in the middle of the sea there is no reason for us to connect it to a junction).

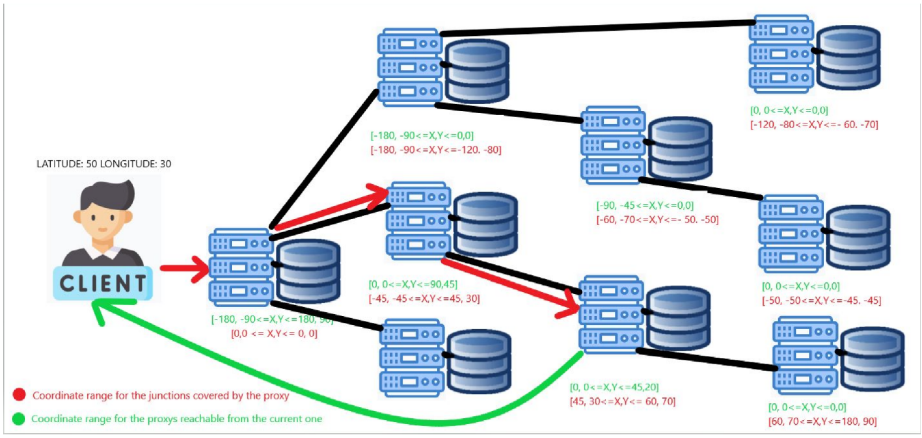

Figure 4.4: Proxy queries flow

The servers themselves have a range of coordinate values for the covered junctions and a range of coordinate values for the known connected sub-proxies. The coordinate range of the know connected proxies lessens the more you go down the connection tree until an endpoint is reached, marked with range 0,0 <= >0,0. (Figure 4.4). If we still don't get an answer from the proxy containing a junction that means that we are out of reach and the client will just stop. Like this, the more servers we will have to spread our junction data, the faster the communication will be, as the client doesn't necessarily need to interrogate the main server, it can interrogate any of the servers.

4.8 Junction Main Server

The executable is a wrapper over the server implementation that manages the traffic states. It keeps track of the incoming vehicles and times out each and every traffic state. The implementation itself is thought to be a state machine, with the states represented by the traffic states and the events being timer expire events. Each state has associated to it a timer that decreases normally or when an incoming vehicle is detected. Whenever we are in a given state, it's corresponding timer is frozen and afterwards reset.

```
{
    "server": { "ip": "127.0.0.1", "port": 5000 },
    "maxWaitingTime": 2,
    "usingLeftLane": true,
    "lanes": {
        "top": "keyword1", "down": "keyword2",
        "left": "keyword3", "right": "keyword3"
    }
}
```

Both the timer for the green light and the ones attached to each lane dynamically update based on the traffic. To describe the way this is done we must first annalise what increasing/decreasing the waiting time means for our system and also what could cause this scenario. What we would wish to achieve is that every time a state ends X drivers have passed the junction. To do so, we control the amount of time a driver has to pass the junction. Also, the most important part is to minimize the amount of time those cars have to wait to pass the junction. Like so, we would want to increase/decrease the green light timer as well as the driver waiting timer attached to each lane. Figure 4.5 illustrates what negative effects playing around with those timers would bring to our system.

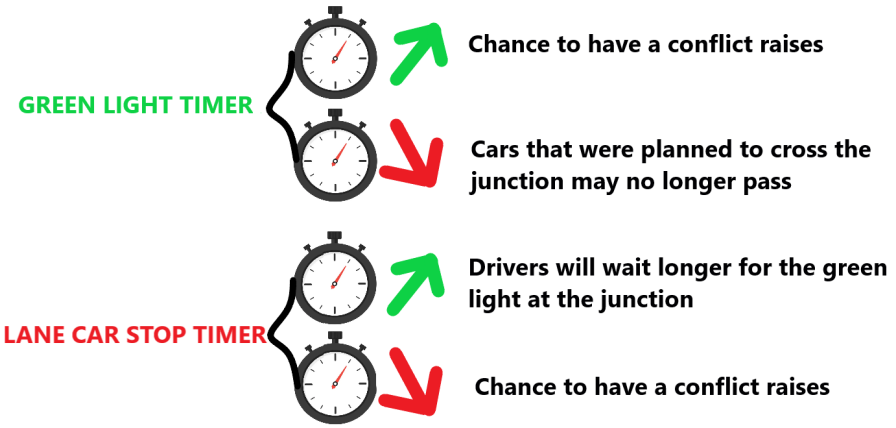

Figure 4.5: Timer Increase/Decrease Impact

To achieve the desired traffic flow described above while minimizing the waiting time as much as we can, we must be aware of 2 things: avoiding fewer cars passing the junction; avoiding conflicting states

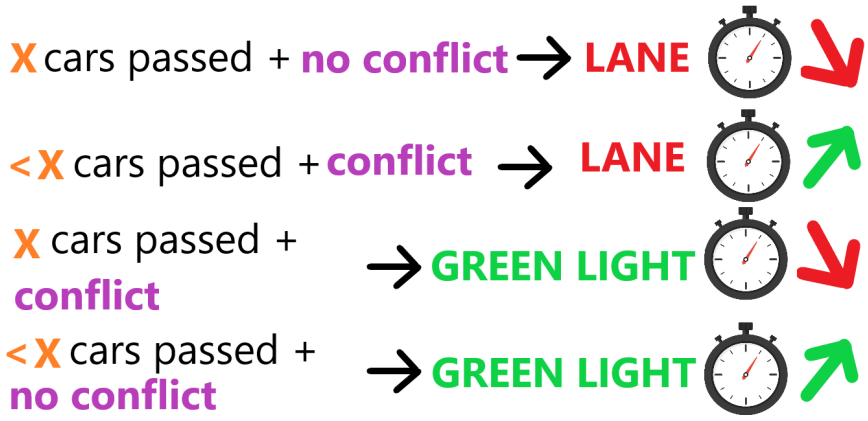

Figure 4.6: Faulty Scenarios Handling

Figure 4.6 illustrates how we actually handle each combination of those scenarios. The way we determine if enough vehicles passed is by tracking the actual cars. Due to high complexity of image comparison, this number is based only on the Vehicle Trackers input, so Traffic Observer client does not contribute to this process and significant decreases in waiting time can be seen only for the cars that are capable of using the Vehicle Tracker client.

As for the way the lane timer decreases when a new vehicle is detected we will have a value that constantly increases the more cars are waiting. We also keep in mind an average of cars waiting to pass the junction and whenever the value is surpassed we start to drastically increase the amount of time decreased for the corresponding timer.

During the testing of the given module it was observed that traffic tends to keep it's normal flow, there is no jumping between traffic states unless a large inflow of cars is detected. Even if this happens, after the required jump transitions, traffic states will be normalized back to their initial flow definition.

'' We mainly focused on the performance part to be able to provide the best traffic conditions that we could offer at minimal costs. To be able to display it, we compared it with the normal flow of traffic by keeping the tracking capabilities of our system and removing timers dynamic adaptations based on traffic inflow. The red line (Fig 4.8) represents the average amount of time cars had to wait when traffic was calculated statically with a set waiting time and fixed traffic state transitions. The blue line (Fig 4.8) represents the average amount of time cars had to wait when the waiting time was calculated dynamically, and we could jump through the traffic states based on traffic inflow. We also wanted to present a scenario when our system will not perform optimally, so in Fig. 4.8

we presented the performance difference when it's raining. As you can see, it's clear that cars had to wait less time when the optimizers were enabled.

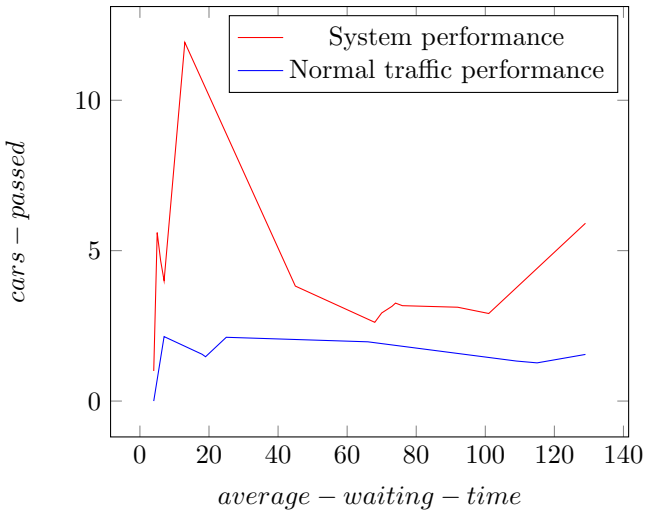

Figure 4.7: Performance comparison to regular traffic systems

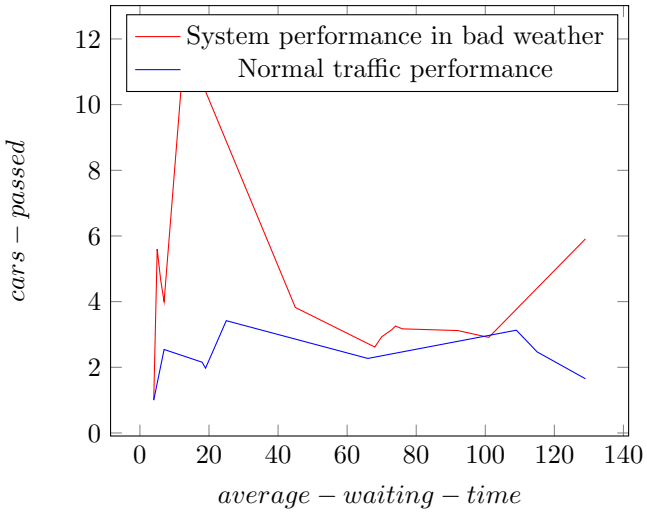

Figure 4.8: Performance comparison during bad weather conditions

4.9 Traffic Observer

The executable is a wrapper over the client implementation that combines car-detection functionalities. On startup, it receives a keyword for the junction to be able to determine what lane provided the incoming car message.

To be able to run the executable itself you will need to provide a camera as well as a way to connect to the network, for example a Zigbee. For testing purposes a video was provided instead of linking directly to a camera. In Figure 4.9 we can see in yellow bounding boxes the objects that were detected to be moving and in red bound boxes the moving objects that proved to be actual vehicles.

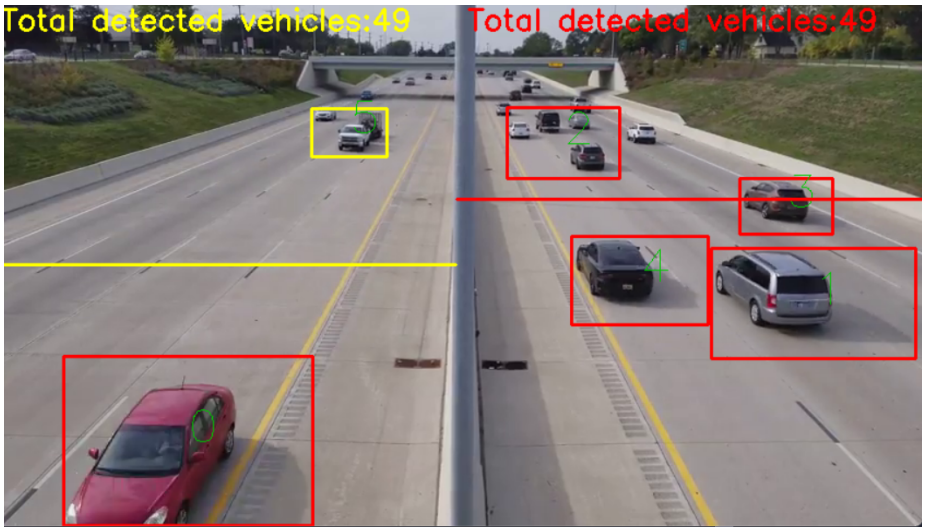

Figure 4.9: Real time traffic detection examples

The way it works is as follows: It starts off by loading the car-detection module and attempting to connect to the object detection server. After the connection is established it reads the keyword provided when starting the executable so that it can connect to the JMS. For security purposes we added a public private key exchange using RSA(from OpenSSL). So, before sending the actual keyword over the network, the public key of the JMS is requested. Only after this operation finished successfully we encrypt the keyword and send it over the network to the JMS so that it can be decoded on the server side and determined if the keyword matches any kind of available lanes.

Once everything is set up, it will periodically send the number of cars that have been detected so that the traffic state can be updated accordingly from the JMS side.

4.10 Vehicle Tracker

The executable is a wrapper over the client. On startup, it tries loading a config file that contains an ordered list with the previous queried proxies. If not present, it will just interrogate the main server. Once the application shuts down a config file will be saved with all the queried proxies, so it will almost always load a config file. It contains a list of proxies represented by their connection data.

```
{
    "proxies": [
    { "ip_adress": "127.0.0.1","port": 6000},
    { "ip_adress": "127.0.0.5","port": 5000}]
}
```

The executable aims to be downloaded on each car system by the manufacturer, and it requires a GPS module to be present as well as a Wi-Fi module. Messages start to be sent only when the vehicle itself is moving. To be able to determine this as well as it's heading GPS data is manually parsed, extracting latitude and longitude from NMEA GPGLL, GPGGA and GPRM inputs. For testing purposes data was already written inside a file and passed through a pipe to the application. The data itself was taken from a public repository.

We first query the last visited proxy. If we are out of reach, we move up the proxy list until eventually we get valid junction connection data 4.10 or we get back to the root of the connection tree(the main server). If we have reached the root node, the process of searching for a junction without a config file restarts. Once we got the connection data we notify the junctions whenever we approach/disconnect from it. When we have passed a junction the whole process restarts from the last queried proxy.

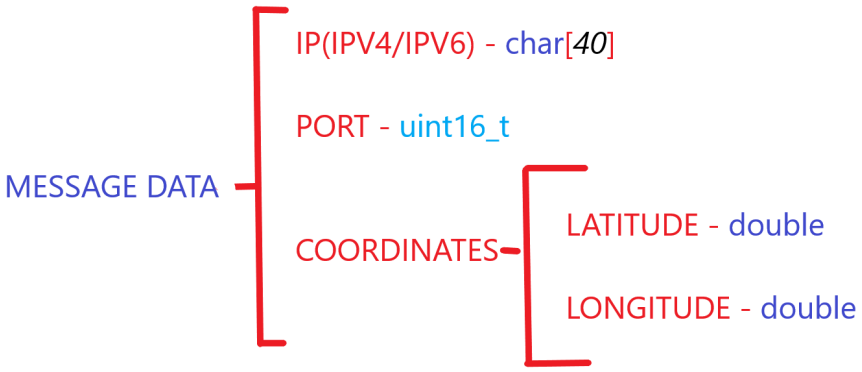

Figure 4.10: Proxy junction data reply

Chapter 5

Conclusions

5.1 A new way to manage traffic

In this thesis, through an analysis of existing research, implementation examples, and emerging trends, we have gained valuable insights into the effectiveness and potential of traffic management systems. With the data collected, we have provided a new alternative way of managing traffic by defining and implementing a new traffic system that we believe will significantly improve traffic flow.

The new system itself attempts to benefit on already existing V2V and V2I communication and provides an IPC based system with multiple clients, servers and proxies that can achieve data car collection and processing on a global scale. It servs as a new economic approach, due to the low hardware requirements, and acts as a bridge between current traffic flow handling and a fully DSRC signal based traffic control system that we believe will be the future of traffic handling systems.

5.2 System Flaws and Further Directions

The main disadvantage of our method is the fact the green/red light time duration updates only when a DSRC vehicle passes the junction, so if there isn't any support from the infrastructure, then we will assume to be left with no green light/red light time increase/decrease. This hypothesis is wrong, because timer will still decrease based on the number of vehicles that arrive at the junction. So actually we would just not have any kind of timer increase. As a result we would have much higher odds of encountering traffic state conflicts described in chapter 4. The difference in performance can be seen in Fig 5.2, the green line representing the average car waiting time when no DSRC vehicles were detected.

The first improvement that needs to be done before using our systems is improving the security layer. As for now, we did not focus on it that much, all that was implemented as a security measure was a public private key exchange

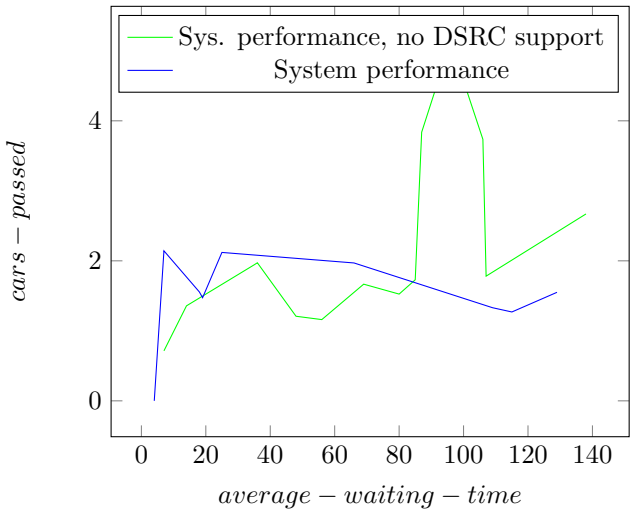

Figure 5.1: Performance comparison to none DSRC traffic

with the help of RSA between the JMSs and the TOs so that the connection key can not be intercepted and used maliciously. Regarding the other messages, we did not encode them as they do not contain any kind of sensitive data, and it would have just lead to an overall slower performance. The problem the needs to be addressed is the connectivity to the JMS of the VTs, so that there can't be a malicious entity that can fake the arrival of multiple VTs. One way to do it is to approximate the average size of a vehicle and map out the area of already connected VTs, but this would lead to a high computational complexity as we would have to verify if any kind of bounding rects of our vehicles would overlap with one another. We would need to consider if we really want to add this kind of security measure when there is a such high cost regarding performance, or, of course, a better way to do it needs to be found.

Another improvement that should be done before deploying our system on a global level is to create a real time test environment, as for now, no real-time testing was done, so the system may or may not fail. We tried to provide as close to reality data to our system, but there are almost always room for error when dealing with such vast subjects like traffic management. Right now the system is emulated for just one junction located in Timişoara, near West University of Timişoara. The GPS data is generated with the help of NMEAGEN and the actual camera input are videos as relevant as possible, so we can handle as many traffic scenarios as we can.

One last improvement that could be done before utilizing our systems would be to improve our model capabilities of detecting vehicles by gathering more data or even changing the whole architecture of the CNN(Convolutional Neural Network) model as for now we used already pre-trained models provided by the Tensorflow Object Detection library.

Bibliography

[1] Dex R. Aleko and Soufiene Djahel. "An IoT Enabled Traffic Light Controllers Synchronization Method for Road Traffic Congestion Mitigation". In: *2019 IEEE International Smart Cities Conference (ISC2)*. IEEE, Oct. 2019. DOI: 10.1109/isc246665.2019.9071667.

[2] Akshay D. Deshmukh and Ulhas B. Shinde. "A low cost environment monitoring system using raspberry Pi and arduino with Zigbee". In: *2016 International Conference on Inventive Computation Technologies (ICICT)*. IEEE, Aug. 2016. DOI: 10.1109/inventive.2016.7830096.

[3] Junchen Jin, Xiaoliang Ma, and Iisakki Kosonen. "An intelligent control system for traffic lights with simulation-based evaluation". In: *Control Engineering Practice* 58 (Jan. 2017), pp. 24–33. DOI: 10.1016/j.conengprac.2016.09.009.

[4] Till Neudecker et al. "Feasibility of virtual traffic lights in non-line-of-sight environments". In: *VANET '12: Proceedings of the ninth ACM international workshop on Vehicular inter-networking, systems, and applications* (June 2012). DOI: 10.1145/2307888.2307907.

[5] Marcel Sheeny et al. "RADIATE: A Radar Dataset for Automotive Perception in Bad Weather". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2021. DOI: 10.1109/icra48506.2021.9562089.

[6] K.T.K. Teo, W.Y. Kow, and Y.K. Chin. "Optimization of Traffic Flow within an Urban Traffic Light Intersection with Genetic Algorithm". In: *2010 Second International Conference on Computational Intelligence, Modelling and Simulation*. IEEE, Sept. 2010. DOI: 10.1109/cimsim.2010.95.

[7] Ishu Tomar, Indu Sreedevi, and Neeta Pandey. "State-of-Art Review of Traffic Light Synchronization for Intelligent Vehicles: Current Status, Challenges, and Emerging Trends". In: *Electronics* 11.3 (Feb. 2022), p. 465. DOI: 10.3390/electronics11030465.

[8] R. Zhang et al. "Increasing Traffic Flows with DSRC Technology: Field Trials and Performance Evaluation". In: *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, Oct. 2018. DOI: 10.1109/iecon.2018.8591074.