UNIVERSITÀ DI PARMA

UNIVERSITA' DEGLI STUDI DI PARMA

DOTTORATO DI RICERCA IN
"MATEMATICA"

CICLO XXXV

Control, Perception and Coordination algorithms for Advanced Driver-Assistance System: optimized implementations and simulations

Coordinatore:
Chiar.ma Prof.ssa Alessandra Lunardi

Tutore:
Chiar.mo Prof. Nicola Capodieci

Dottorando: Filippo Muzzini

Anni Accademici 2019/2020 – 2021/2022

Abstract

The future of urban mobility is undergoing changes with the development of intelligent cities and the increased use of autonomous vehicles. The transition to this new paradigm is a gradual process over several years or decades, but progress has been made through the implementation of smart sensors and communication infrastructure. The development of Advanced Driver-Assistance Systems (ADAS) with growing autonomy is also in progress.

In this thesis, two main aspects are addressed: *coordination* algorithms to manage the smart city traffic flow and the *perception-control* pipeline of autonomous vehicles with specific emphasis on the *localization* and *planning* phases.

With regards to the first aspect, several novel algorithms are proposed that exploit the new smart city capabilities to address typical problems such as Traffic Lights and Intersection Management, Parking Management, and Emergency Vehicles Management. The work proposed in this thesis is a study of the current situation in which autonomous or able-to-communicate vehicles and *traditional* vehicles that are not able to communicate with city infrastructure co-exist. This is a crucial aspect since mixing ADAS and *traditional* vehicles impacts the algorithm design. The proposed algorithms are tested in a simulated scenario in order to study unexpected behaviors since traffic flow is a complex system and some events can trigger unpredictable consequences. The results show that the proposed algorithms improve the city's livability by decreasing the waiting time at traffic lights, reducing the parking search time, and the emergency vehicle response time.

In regards to the *localization* and *planning* stages, the emphasis is placed on the execution time of the algorithms, as it is a critical aspect. If the *perception* and *control* pipeline takes too long, the intended maneuver may become outdated due to changes in the environment, potentially causing safety risks. In light of this, novel implementations for *localization* and *planning* algorithms are proposed, which make extensive use of the GPU as an accelerator in order to reduce computational time. The GPU is leveraged to parallelize the algorithm and minimize memory access. Additionally, the use of different floating-point precision types is investigated to assess the impact on the results. The proposed implementations of the ORB-SLAM algorithm for the *localization* phase and the *Frenet Path Planner* algorithm for the *planning* phase show a consistent speedup, compared to the previously published CPU-based implementations of the algorithms.

Contents

Acknowledgements

I want to give a big thank you to everyone at the HipeRT lab who helped me out throughout this journey, especially my mentor Nicola for all the great advice he gave me during my Ph.D.

I also want to thank my grandparents, my parents Dirce and Eugenio, and my brother Andrea for always being there for me and supporting me along the way.

And a big thank you to Susanna for being by my side and supporting me through most of this journey.

And last but not least, a huge thank you to all of my friends for always being there to encourage me.

Chapter 1

Introduction

1.1 Motivation

The future of urban mobility is poised to undergo significant transformations, as the implementation of smart cities and the widespread adoption of autonomous vehicles are expected to occur. This transition is likely to be a gradual process, spanning several years or even decades. However, some efforts have already been made in this direction, with the deployment of smart sensors such as cameras for traffic monitoring and smart actuators like traffic lights being implemented. Furthermore, efforts are being made to develop infrastructure that enables the communication between vehicles (V2V) and between vehicles and smart sensors or actuators (V2I) as shown in Figure 1.1. While fully autonomous vehicles are not yet a reality, recent advancements in technology have led to the development of Advanced Driver-Assistance Systems (ADAS) that utilize embedded boards to run algorithms that assist drivers during their trips. These systems can have varying levels of autonomy, depending on the maneuvers they can perform without human intervention.

The infrastructure that supports smart cities and autonomous vehicles, such as the equipment used within urban environments and in-vehicle embedded computational boards, is a critical element in the successful deployment of these technologies. However, the algorithms that govern these systems are also of paramount importance. Algorithms that are inadequately designed can result in safety hazards and accidents in the case of autonomous vehicles, or disruptions to city life and traffic flow in the case of smart city management. As such, it is essential that the algorithms employed in these dynamic and complex contexts possess robustness to handle unexpected sit-

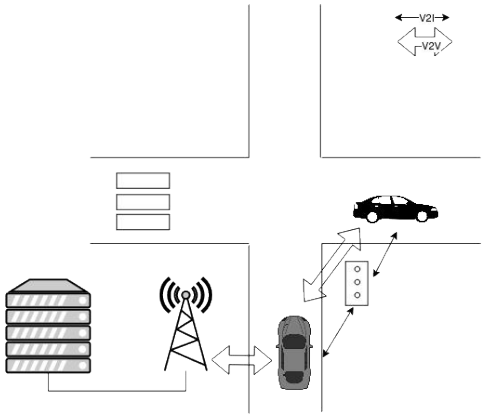


Figure 1.1: Communication between vehicles (V2V) and between vehicles and city infrastructure (V2I).

uations.

In an imaginary scenario in which all vehicles are autonomous and have the capability to communicate with each another, *coordination* algorithms can be designed to take into account other vehicles' behavior and thus derive appropriate plans. However, such a scenario also presents the challenge of incorporating unpredictable elements, such as pedestrians, into the decision-making process. In reality, the situation becomes even more complex when not all vehicles are autonomous and connected, as human-driven vehicles introduce an additional layer of unpredictability. Furthermore, the traffic flow system is highly complex, where a single event can initiate a cascade of responses from other vehicles, making it difficult to develop algorithms that can effectively prevent negative outcomes.

When focusing on the behavior of individual autonomous vehicles, it is important to consider the *control* algorithms that plan and execute their movements. The entire pipeline can be split into two phases that are more focused on environmental *perception* and on actual *control* of the vehicle. Vehicle movements are dependent on the current state of the environment, thus *perception* algorithms must have the ability to perceive the environment, while *control* algorithms have to plan the vehicle movements based on the current and potential future states of the environment. This entails considering the actions and behaviors of other vehicles and pedestrians, both in the present moment and in the foreseeable future, in order to guarantee the safety of the intended movements.

To guarantee the safety and efficacy of *coordination* algorithms in smart

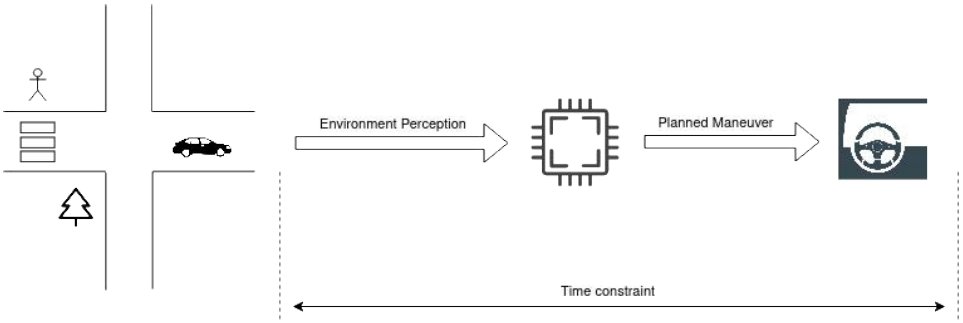


Figure 1.2: Time Constraint in algorithms execution from the environment perception to the planned maneuver execution.

cities and *control* algorithms in autonomous vehicles, it is essential to thoroughly test them in simulated environments before implementation in real-world scenarios. Additionally, it is crucial that *control* algorithms are able to perform their functions in a timely manner as the environment is dynamic and information that was relevant at one point may rapidly become outdated. If a *control* algorithm has a large processing time, it may be applied to a situation that has since changed, potentially reducing its effectiveness. Hence, to ensure safety, *control* algorithms must be able to execute efficiently and quickly or anyway at a frequency close to the actuator rate (see Figure 1.2).

In summary, there are two main aspects of the security of smart cities and the vehicles that inhabit them: *coordination* among vehicles, and the time constraints of the single vehicle *control* algorithm execution. The first aspect involves accounting for the unpredictable behavior of non-connected vehicles and ensuring that the interactions enhance the livability of the city. Thus, it is essential to test the effect of *coordination* algorithms in a simulated urban environment. The second aspect is related to the optimization of *control* algorithms and the capabilities of the hardware mounted inside the vehicle. Due to space and weight limitations, vehicles cannot be equipped with high-end hardware [1]. To address this, a possible solution is to use embedded boards that are compact yet have limited resources, which implies that execution times become a critical consideration. Therefore, *control* algorithms must leverage the hardware capabilities of compute accelerators in order to minimize execution times.

1.2 Scopes

The primary objective of this thesis is to investigate the capabilities of Advanced Driver-Assistance Systems (ADAS) and connected vehicles within the context of smart cities. The aim is to harness the city's infrastructure and embedded boards to improve both the livability of the city and the security of ADAS algorithms. The aforementioned considerations are taken into account in the presented analysis and proposals.

The overall focus is on addressing two main areas: the development of *coordination* algorithms for managing the flow of traffic in urban environments, and the optimization of *control* and *perception* algorithms for individual vehicles. To achieve the aforementioned scopes the thesis is organized into two parts: the first is dedicated to improve the livability of the smart city using smart algorithms (Chapters 3 and 4); the second part is dedicated to improve the security of ADAS algorithms (Chapters 6 and 7).

In the context of smart cities addressed in the first part of the thesis, the focus is on addressing the co-existence of connected and non-connected vehicles in the same scenario, and the proposed *coordination* algorithms are evaluated in a simulated environment to assess their impact on traffic flow.

In the first portion of the thesis, the focus is on the smart city scenario and the development of *coordination* algorithms among connected vehicles to address common issues such as Parking Management, Traffic Light Management, Intersection Management, and Emergency Vehicle Management.

In the second part, the proposed solutions for Advanced Driver Assistance Systems (ADAS) and autonomous vehicles are geared towards reducing the computational time of *control* and *perception* algorithms through the utilization of parallelization and hardware acceleration on embedded boards.

In particular, the focus shifts to the individual vehicle, where specific tasks within the *perception* and *control* pipeline, such as *localization* and *planning*, are addressed with the goal of reducing execution time through the use of accelerators in the algorithms used to address these tasks.

1.3 Methodologies

To achieve the fist scope, novel *coordination* algorithms to manage the smart city crucial aspects are proposed. Moreover, they are tested in a simulated scenario to see their effectiveness.

In particular, a new auction-based system for Traffic Lights Management is proposed (Section 3.2.1) and its effectiveness is evaluated by comparing it to the *traditional* Fixed-Time traffic lights policy. The proposed system is suitable to reduce the waiting time at traffic lights compared to the Fixed-Time policy. Additionally, the system is tested in a simulated scenario in which some vehicles are equipped with the proposed system while others are not.

Similarly, for the Intersection Management problem, a novel management system based on auction is proposed (Section 3.2.2). Several variants that consider different strategies to bid and different policies to choose which vehicle will pass the intersection are evaluated through various simulations. Moreover, the system is designed to be easily adopted by human-driven vehicles with minimal equipment, as it takes into account the limitations of human capabilities to carry out very precise instructions.

With regards to Parking Management, a new, smart system is proposed that enables vehicles to select and reserve parking spots (Section 4.2.1). The system is engineered to monitor the utilization of parking areas and provide vehicles with information about parking spot availability. Additionally, it is capable of reserving the parking spot selected by the vehicle. The co-existence of smart vehicles and *traditional* vehicles is also taken into account in the design of the system. The system is resilient to the presence of *traditional* vehicles and has been shown to reduce the time spent searching for a parking spot by up to 76% when compared to baseline strategies utilized by human drivers. The impact of the presence of *reservable* parking spots is also investigated.

Considering Emergency Vehicle Management, a new system is proposed that aims to decrease emergency vehicles response time by up to 36.7% with respect to a scenario in which no such system exists (Section 4.2.2). The system notifies other non-emergency vehicles to alter their routes in order to avoid congestion on streets traversed by emergency vehicles. Additionally, the system informs vehicles of accidents as they occur, allowing them to reroute and avoid the accident site, resulting in a reduction of trip time by up to 36.1% compared to the trip time of the same vehicle that does not interact with the system.

To achieve the second scope, two important task (*localization* and *planning*) of ADAS vehicles are analyzed and optimized to achieve better execution times.

To optimize the performance of the *localization* task, the ORB-SLAM algorithm is addressed and a novel GPU-based implementation using CUDA

is proposed (Section 6.3). This new implementation demonstrates a significant increase in performance, with a speed-up of up to 3x compared to the original baseline CPU-based implementation of ORB-SLAM, achieved through the optimal utilization of concurrency and the implementation of novel methods for the *Pyramid* construction and point filtering phases of the algorithm.

With regards to the *planning* task, the proposed novel implementation of the *Frenet Path Planner* algorithm, which is designed to exploit the capabilities of a CUDA-capable GPU, demonstrates a significant increase in computational efficiency (Section 7.3). Through the optimal utilization of GPU resources, such as *shared memory*, the implementation achieves a speed-up of up to 28x compared to the baseline CPU-based implementation. Additionally, the impact of various types of floating-point precision on the computational time and trajectory precision is also evaluated.

In Chapter 2 of this thesis, the concepts related to the analysis of smart cities and connected vehicles are presented. The proposed algorithms for addressing the *coordination* of vehicles at both a local level, such as Traffic Light Management, and a global level, such as Emergency Vehicle Management, are discussed in Chapters 3 and 4 respectively. These algorithms are evaluated using an urban simulator and the extension implemented in the simulator to enable the simulation of smart cities is described in Chapter 2 (Section 2.2). Chapter 5 shifts the focus to the individual vehicle, discussing the role of ADAS vehicles and the importance of *control* algorithm execution time and embedded boards. In particular, the topics of *localization* (Chapter 6) and *planning* (Chapter 7) in the *perception* and *control* phases are examined, algorithms for these tasks are presented, and novel implementations proposed. These implementations are evaluated on an embedded board commonly used in ADAS and autonomous vehicles. Finally, the thesis concludes with a summary of key findings and recommendations in Chapter 8.

Chapter 2

Connected and Autonomous Vehicles in Smart Cities

2.1 Introduction

The rapid growth of the Internet of Things (IoT) [2] presents new opportunities for smart mobility [3, 4], which can be realized through the implementation of smart city infrastructure and systems. By leveraging these services, the overall driving experience can be improved, resulting in reduced travel times, decreased traffic congestion, reduced pollution, and a reduction in driver stress. Ultimately, these efforts aim to make cities more efficient, safe, and livable for both residents and workers. V2I and V2V communication will allow for the exchange of valuable information [5] and the ability for vehicles to make autonomous decisions on the most efficient routes based on factors such as distance, cost, driver needs, and real-time traffic conditions [6].

However, the realization of smart mobility requires not only a robust infrastructure that connects vehicles and the environment but also sophisticated algorithms that effectively manage and coordinate the movements of these vehicles [7].

Another important consideration in the implementation of smart mobility is the heterogeneity of vehicles, with some featuring advanced communication capabilities or driving assistance systems (ADAS) while many others currently do not. It is likely that in the future, all vehicles will be equipped with these advanced systems, however, in the present and near future, it is necessary to account for scenarios where both connected and non-connected vehicles exist. Furthermore, in the implementation of autonomous vehicles,

it is also necessary to consider the co-existence of human-driven cars. This means that it will be necessary to account for the interactions between both connected and non-connected cars as well as autonomous and human-driven cars. In the end, vehicles can be classified into four distinct groups (Figure 2.1):

- **Equipped** or connected: Vehicles able to communicate with city infrastructure or other vehicles;
- **non-Equipped** or non-connected: Vehicles not able to communicate with city infrastructure or other vehicles;
- **Autonomous** or self-driving: Vehicles able to drive themselves without human intervention;
- **non-Autonomous** or human-driven: Vehicles are not able to drive themselves without human intervention.

It is important to note that a vehicle may be equipped to communicate, but not necessarily be fully autonomous, and vice versa since an autonomous vehicle can leak in the connection capabilities or not support the communication protocol adopted by the city infrastructure. This type of autonomous vehicle must be treated as non-equipped since they are not able to exchange information within the city infrastructure. In certain contexts, it is also common to refer to *traditional* vehicles, which encompasses both *non-connected* and *non-autonomous* vehicles.

This heterogeneity of vehicles has implications for the design of solutions for smart mobility. Solutions developed for scenarios in which all vehicles are autonomous or connected may not be appropriate during the transition period where non-connected and connected vehicles will coexist. It is important to take into account that human drivers cannot be relied upon to follow instructions as precisely as autonomous vehicles. For example, at intersections, human drivers may be able to execute simple instructions such as "stop and wait for your turn" or "go, it's your turn", but not more complex sequences of instructions or precise maneuvers such as to pass close to other vehicles with a few cm of precision or follow a precise speed or acceleration. Additionally, it is not feasible to rely on human drivers to substitute for sensors and cameras in collecting data for implementing *coordination* policies. Therefore, it is necessary to design algorithms that can account for the co-existence of both types of vehicles and are able to take advantage of the capabilities of smart cities.

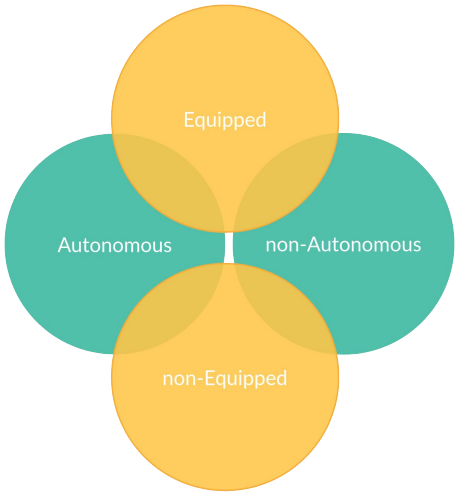


Figure 2.1: Vehicles Categories.

The implementation of new algorithms for traffic management has the potential to revolutionize the way in which we experience the city. However, due to the complexity of traffic flow, these algorithms may exhibit unexpected emergent behaviors that could have negative impacts on urban life. To mitigate this risk, it is essential to thoroughly test proposed traffic management approaches before their implementation in real-life scenarios. This necessitates the use of an accurate urban traffic simulator for the analysis of the complex dynamics arising from the interactions between the smart city infrastructure and vehicles. The unintended consequences of the implementation of new algorithms can have serious negative impacts on the city. Miscommunication with vehicles or improper management of traffic lights, for instance, could lead to collisions. An inaccurate algorithm can also create traffic congestion, resulting in increased travel times for drivers and significantly hindering the ability of emergency vehicles to respond to rescue operations. These examples serve to underscore the significance of conducting a simulation phase prior to the actual deployment of systems and algorithms in the city. For the use cases discussed in Chapters 3 and 4, the MATSim urban simulator[1] was used to perform the necessary testing. The MATSim Simulator [8] is a mesoscopic simulation platform for urban transportation that is based on the principles of Multi-Agent Systems (MAS) [9].

The standard version of MATSim is restricted in its ability to simulate in-

[1]https://www.matsim.org/

teractions among road users, including communication between vehicles and the surrounding infrastructure. To address this limitation, a new extension to MATSim (described in Section 2.2) has been developed, which allows system engineers to simulate the interactions among sets of connected vehicles within a Smart City. This extension facilitates a more thorough evaluation of proposed traffic management approaches before their implementation in actual settings.

The Smart City MATSim extension is composed of multiple modules that simulate cameras, sensors, servers, and communication systems. This enables the simulation of algorithms that leverage smart city hardware.

Given the issues discussed above, in Chapter 3 and 4, the intersections, emergency vehicles, traffic lights, and parking problems are examined. For each of these issues, a system that utilizes smart city infrastructure is proposed. Furthermore, the aspect of vehicle heterogeneity is also taken into consideration when testing the proposed systems in simulation environments.

2.2 MATSim extension

MATSim, or Multi-Agent Transport Simulation, is a sophisticated urban simulation software that utilizes a multi-agent paradigm to simulate the daily traffic of a city. The software evaluates the actions of individual vehicles, taking into account the interactions between them as they share the same traffic network. These interactions result in the emergence of traffic flow patterns and potential congestion.

The simulation can be divided into two levels of abstraction: the microscopic level, which pertains to the behavior and plans of individual vehicles, and the macroscopic level, which pertains to the aggregate traffic flow patterns that result from the interactions of these vehicles.

The primary goal of MATSim is to provide a comprehensive simulation of traffic flow at the macroscopic level, using the daily plans of vehicles as inputs. The resulting simulation is based on the collective behavior of individual vehicles and their interactions, rather than relying solely on mathematical models.

Through this approach, MATSim allows for the testing of various scenarios and the examination of how changes in the environment, vehicle plans, and behavior can affect traffic flow patterns.

On the other hand, the basic form, MATSim does not cover the typical

capabilities of a smart city. In the smart city scenario vehicles and infrastructure are able to exchange information, and the city is equipped with sensors and actuators to perceive and manage crucial aspects such as traffic flow or parking. In order to add the possibility to simulate a city with these capabilities, the MATSim modularity is exploited. The new modules are built without a specific algorithm in mind but they are designed to be ready to permit the programmer to write algorithms that have to exploit smart city capabilities. In particular, the new modules give the possibility to change the vehicle's route in response to other events (Section 2.2.1), design custom smart algorithms for traffic lights and intersection management (Section 2.2.4), and manage parking areas with smart algorithms (Section 2.2.5). These modules require a substrate of simulated sensors and communication stuff, for this reason, modules to add communication and perception capabilities to MATSim are built (Sections 2.2.2 and 2.2.3). Eventually, the new MATSim extension adds to the basic form of the simulator the communication and perception capabilities and a smart structure that is used to implement smart algorithms for parking, emergency vehicles, intersection, and traffic lights management as in Chapters 3 and 4.

2.2.1 Dynamic Vehicle Routing

In its basic form, MATSim simulates vehicles that follow pre-determined plans and routes without deviation. However, in the context of simulating a smart city scenario, it is necessary to extend this concept by introducing the notion of interconnected vehicles that are able to retrieve real-time traffic information and adjust their routes accordingly.

Previous work has proposed extending MATSim to enable dynamic routing [10], which allows vehicles to change their routes at runtime in response to events that occur during the simulation.

The new MATSim extension used in this thesis builds upon the previous work mentioned above by extending MATSim to introduce dynamic behavior for vehicles in a smart city scenario. This new extension allows for the variation of vehicles' behavior in response to real-time smart city events and further enables vehicles to communicate with a central city server in order to retrieve and utilize relevant information for decision-making and planning.

An example of the practical application of the new extension to MATSim is presented in Figure 2.2. This use case pertains to the ability of vehicles to dynamically reroute in response to congestion in urban traffic flow. In the standard version of MATSim, when a vehicle reaches an intersection, the simulator determines the next street (referred to as a "link" in MATSim

terminology) the vehicle will take based on the predefined route. However, in this use case, the vehicle's route is subject to change if the central server notifies of congestion in the traffic flow. With the new extension, combined with the communication extension described in Section 2.2.2, vehicles are able to compute a new route using the information provided by the server, allowing for real-time adjustments to their planned route when the simulator requests the next link.

2.2.2 Communication

An integral component of smart city systems is the ability for vehicles to communicate with one another and with city infrastructure. However, MATSim, as it currently stands, does not include this capability. To address this limitation, a new extension has been developed to add communication functionality to MATSim, enabling the simulation of smart city scenarios.

This extension is divided into two primary components: the communication functionality itself, and a discovery phase in which vehicles can identify other communicating entities within their vicinity. The first component is implemented through the use of a Java interface, which a vehicle class must implement in order to be able to communicate. This interface defines methods for sending and receiving messages, as well as for discovering other reachable communicating entities.

The second component is implemented by leveraging events emitted by the simulator engine. The goal is to maintain an up-to-date map of the positions of vehicles and other communicating entities. This map is used to identify reachable entities within communication range when a vehicle wishes to communicate. To accomplish this, the extension includes a wrapper that maintains the map as an internal state, updating it when the simulator engine notifies of a vehicle movement. The wrapper also includes a method that returns the reachable entities, which is called by a vehicle when it wishes to initiate communication (Figure 2.3).

2.2.3 Perception

Another key aspect of smart cities is the use of smart sensors to gather information about the city's current state, which serves as the foundation for decision-making. For example, smart cameras that are able to count the number of vehicles present on a particular street can be used to monitor traffic flow and detect congestion.

In its current form, MATSim does not include this concept, so a new

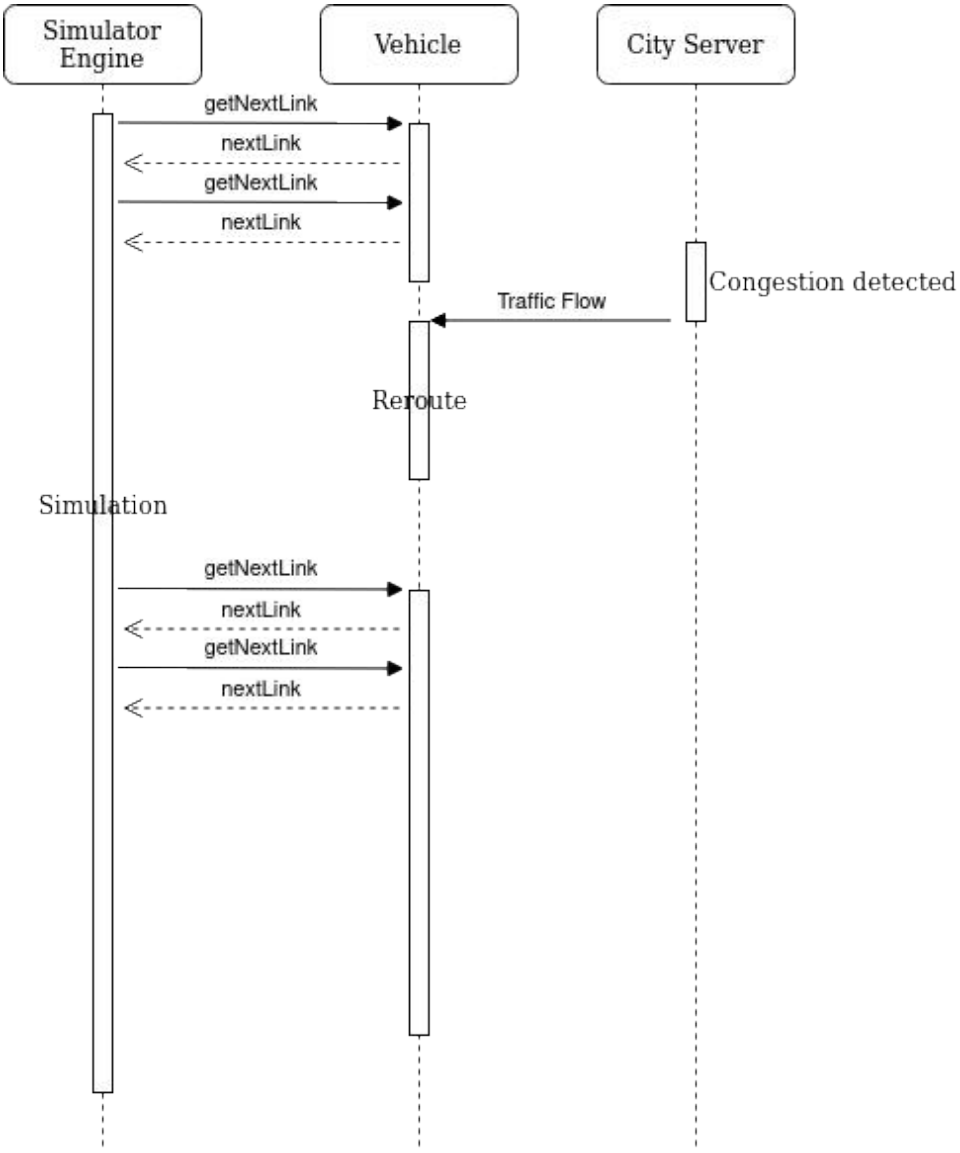


Figure 2.2: Flow of events for a vehicle that dynamically adjusts its route in response to congestion notifications received from the central server.

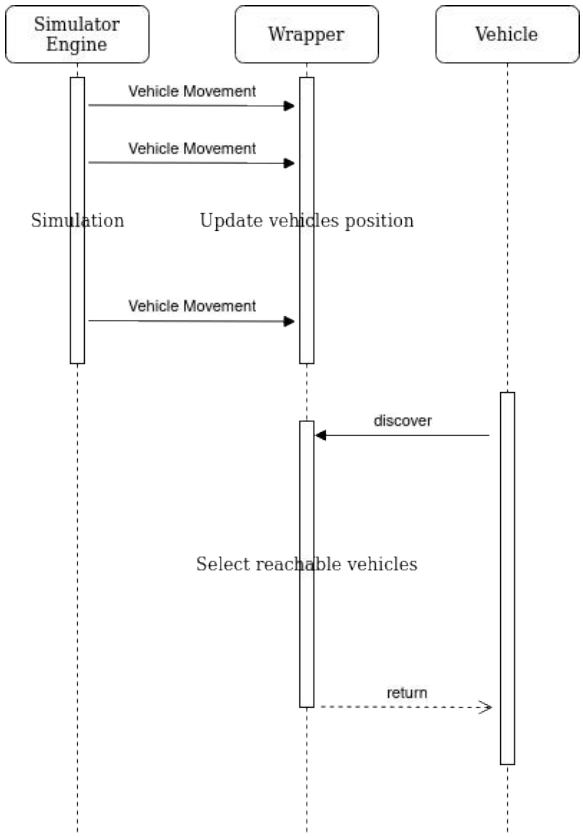


Figure 2.3: Functionality of the communication wrapper in the MATSim extension for smart city scenarios.

extension was developed to simulate this type of sensor. The new extension was implemented using a wrapper as in the communication extension (described in Section 2.2.2). The wrapper captures events emitted by the simulator engine regarding the movement of vehicles, and in particular, is interested in the information about when vehicles enter or leave a link. This information is used to update the internal state of the wrapper, which consists of a vehicle counter for each link present in the city. By incrementing or decrementing the counters based on vehicle movement, the wrapper is able to provide real-time information on the number of vehicles present on each link. The wrapper can work in a passive or active way: it can notify a change in a link but it exposes also a method to retrieve link information by calling it.

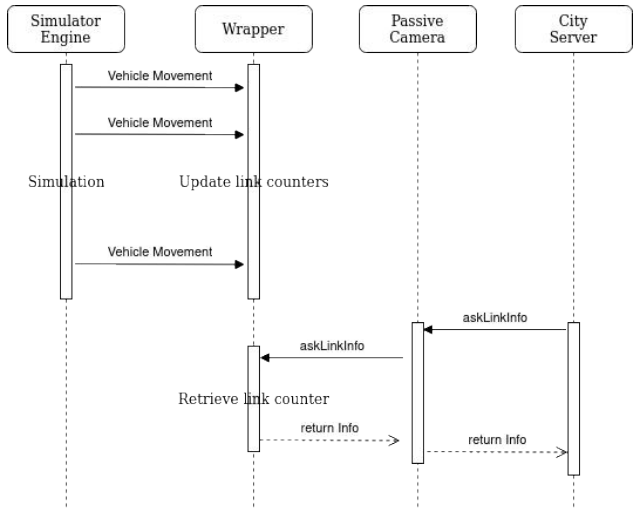
To make the extension more realistic, a class representing a camera was implemented. This class uses the wrapper's information to retrieve the number of vehicles present on the street within its field of view, simulating the behavior of a real-world camera. Additionally, by decoupling the wrapper and camera concepts, it is possible to simulate areas of the city that are not covered by sensors.

In particular, two types of cameras are implemented in the extension: one passive and one active. The passive camera does not actively monitor the street, it waits for a hypothetical city server to request information about the street, at this point the camera retrieves the information from the wrapper and sends it to the requester. The active camera actively monitors the streets, using the active version of the wrapper to be notified of link changes. When there is a change it notifies the change to the city server. The two versions of the perception extension are illustrated in Figure 2.4.

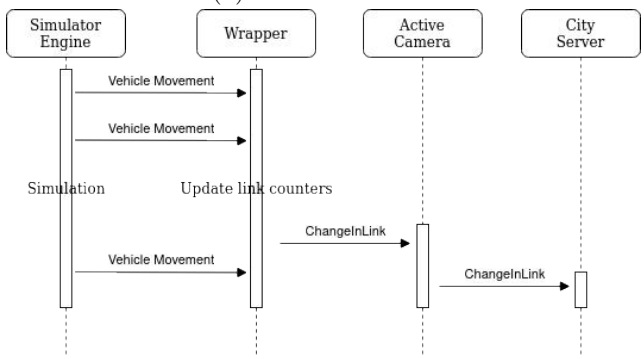
2.2.4 Traffic Lights

A complementary concept to that of sensors in smart cities is the concept of actuators, which are hardware devices capable of performing actions in order to achieve a specific goal. In a smart city scenario, an example of an actuator is a traffic light. While a standard traffic light can manage traffic based on a predefined sequence of states (green and red), a smart traffic light can dynamically adjust its behavior based on information about the status of intersections and adjacent streets, or by receiving instructions from a city server.

In its current form, MATSim does not include a concept of traffic lights. A module for simulating standard traffic lights was added in [11, 12]. This module implements a system that can be summarized in two main aspects:



(a) Passive version.



(b) Active version.

Figure 2.4: Functionality of the perception MATSim extension for smart city scenarios.

it allows only vehicles on streets with a green light to pass through an intersection, and it includes a class that represents the traffic light controller. Each signalized intersection has its own controller, which defines the timing of the green and red light phases.

To extend this functionality, a new extension has been developed to simulate smart traffic lights. By integrating this class with the communication extension (described in Section 2.2.2), it is possible for the traffic light controller to dynamically adjust its behavior in response to real-time information about the city's status. For example, by querying cameras for information on street overcrowding or giving priority to emergency vehicles as indicated by the city server (Figure 2.5). The extension allows the traffic lights to change their behavior dynamically in response to the city's status. In the end, the new extension adds to the previous one described in [11, 12] the possibility to change dynamically the traffic lights' behavior exploiting information that the smart city can give. Since the previous extension allow only to implement statically defined algorithm for traffic lights management, the new extension can be considered as the smart city version of the previous one.

2.2.5 Parking

As reported in the MATSim book [8] the baseline version of the simulator does not take into account the parking aspect. On the other hand in [13] and [14] the authors have extended MATSim to simulate parking. Specifically, the baseline version assumes that a vehicle will always find a parking spot near its destination. With the extensions mentioned above, a vehicle is only able to end its trip if there is an available parking spot. Each link can have an associated parking area with a specific capacity, and a vehicle can park in that link only if the capacity has not been reached. There is a manager that tracks the number of vehicles parked in each area and determines whether a vehicle can park in a particular link. Moreover, in [14], the author proposes an algorithm for parking choice that uses static information that the driver is supposed to know. This parking choice is performed before the driver starts the trip and then is not changed. This is a limitation in a smart context in which the vehicle can retrieve information from the city infrastructure and can dynamically change the parking choice by reacting to parking availability changes.

Indeed, in a smart city scenario, parking areas can be more complex than traditional parking areas. They can have monitoring capabilities, such as the ability to count the number of available spots, and they can control

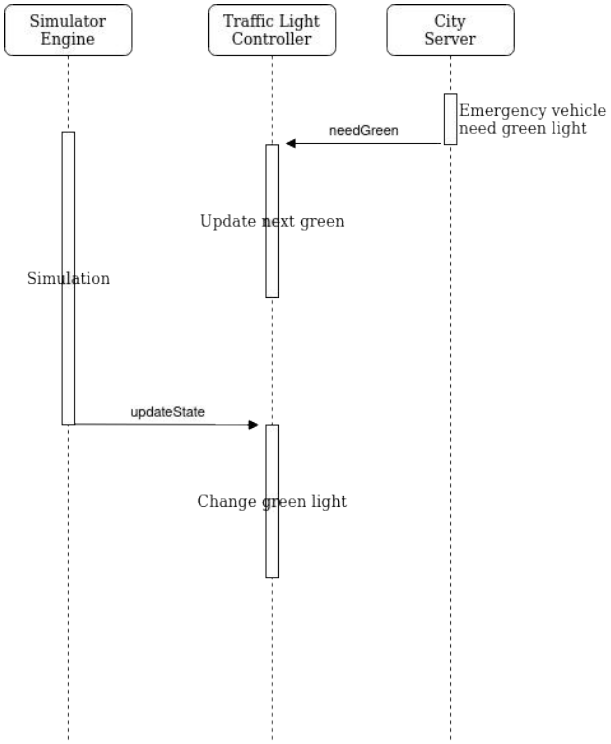


Figure 2.5: Illustration of a smart traffic light responding to a request from the city server to give priority to an emergency vehicle.

which vehicles are able to park in the area. Moreover, they can share with the vehicles their parking availability to facilitate parking choice.

The new module for smart parking is built to allow these smart capabilities to MATSim giving the programmer the possibility to implement more advanced algorithms with respect to the parking extensions of [13] and [14] exploring useful information that the smart parking areas can give. Similar to the perception mechanism described in Section 2.2.3, a class was implemented to represent a camera specifically for parking monitoring. It can notify a hypothetical server of changes in parking occupancy, for example when a vehicle enters or exits a parking area. To retrieve this information, the camera interacts with the parking manager to get the current parking occupancy. The manager is also extended to manage which vehicles are allowed to park, simulating a smart city scenario in which parking spots can be reserved, and only the vehicle that has reserved the spot is allowed to park.

Chapter 3

Local Coordination: Traffic Lights and Intersections

In a hypothetical smart city where vehicles are able to communicate with the city's infrastructure, *coordination* algorithms can involve various entities such as sensors, actuators, and vehicles themselves. In certain situations, the interaction may be localized, as only entities within a certain communication range can participate in the *coordination* algorithm. In these instances, the algorithm can be designed to consider the sensors, actuators, and vehicles within a specific area. This implies that the algorithm does not have a global view of the city, making it difficult to design certain types of algorithms that require such information. On the other hand, an algorithm designed in this way is distributed and does not rely on a central server, which means that there is no single point of failure for the entire city. A failure in a specific area would only result in the inability to utilize the algorithm in that area. Therefore, if the algorithm is designed to consider only the local area, it minimizes the possibility that inefficiencies will affect the entire city.

In this chapter, two *coordination* algorithms that can be designed for a local area are discussed. The first algorithm addresses the Management of Traffic Lights, while the second algorithm addresses the Management of vehicles at unsignalized Intersections.

3.1 Local Coordination Related Work

3.1.1 Traffic Lights Related Work

Traditionally, traffic lights at signalized intersections were managed using static and Fixed-Time Control (FTC) systems [15]. With the emergence of smart cities and connected vehicles, there has been an opportunity to improve the management of signalized intersections through the use of communication. The proposed approach described in [16] and evaluated in [17], builds on the concept of auctions and takes into account the co-existence of vehicles that are capable of communicating with city infrastructure and those that are not. This aspect, which has not been given much attention in the literature, is crucial in scenarios where *traditional* and connected vehicles will co-exist.

Moreover, the proposed system is distributed. Indeed, each traffic light acts as a single entity and does not require a central server or other traffic lights. This means that the system can be deployed in the whole city or only in some critical intersections. In this manner, the city authorities of both large and small cities can find an effective tradeoff between the cost of the deployment of the new system and the benefit that can provide to the city.

Most of the research in this field has been focused on payment-based strategies. For example, in [18], the authors propose a strategy to maximize the utility function reported by drivers at intersections. In [19, 20] a different approach is presented, in which two traffic lights compete to attain the highest value of green time. Micro-auctions are used in a decentralized manner to determine the next signal phase in [21]. However, these approaches primarily rely on communication through smart city infrastructure and do not involve vehicles in a central role. This is an important consideration as drivers could provide information about their priorities and their value of time (VOT) as in [22, 23]. Reinforcement learning is adopted in [24] to obtain an optimal bidding strategy, and in [25] the goal is to minimize the personal delay of drivers through an auction for green time. Another important aspect to consider is incentive compatibility, that is, ensuring that drivers are incentivized to adopt the system and not cheat to gain a personal benefit at the expense of overall performance [18, 26]. This has been demonstrated in [27] where the authors show that overall performance degrades if incentive compatibility is not considered.

Regarding the co-existence of *traditional* and connected vehicles, in [28], the authors propose a method for coordinating the movements of autonomous and human-driven vehicles through traffic lights using a First-Come-First-

Served policy. Similarly, in [29], the authors examine the impact of the penetration rate of an auction system for traffic lights, implying scenarios in which connected and non-connected vehicles will co-exist, however, differently from the proposed approach, they use a second-price auction i.e the highest bid win but the winner has to pay only the price of the second highest bid.

3.1.2 Intersection Related Work

Intersections present a significant challenge in the context of traffic flow [5]. Yield rules have traditionally been implemented to coordinate vehicles at intersections, taking into account the characteristics of both human drivers, such as unpredictable behaviors and reaction times, and the characteristics of the intersections themselves, such as assigning priority to larger and highly trafficked lanes.

However, the scenario changes significantly when vehicles are autonomous and can communicate with one another and acquire information about their surrounding environment. This opens up a wide range of possibilities for improved traffic coordination, overcoming the limitations of traditional traffic lights in allocating resources and preventing the starvation of vehicles on roads without yield.

Several approaches have been proposed for the collaborative management of intersections [30, 31]. One common method is to use auctions to dynamically manage resources [32], as they allow for the pricing of goods based on customer interest [33]. Auctions were largely exploited to manage negotiations between autonomous entities modeled as agents [34].

A number of papers have been published on the use of auctions in Intersection Management, including [32], which uses a wallet system for automatic bidding based on trip characteristics, driver-specified budget, and remaining distance to the destination. It also addresses the optimization of overall traffic.

Another approach is proposed in [35], which uses a two-step auction mechanism. In the first step, only the first vehicles of the lanes are involved. They place the bid following the second-price sealed-bid auction rule [36]: the system allocates the next time slot to the vehicle with the highest bid, while this vehicle needed only to pay the second highest bid placed in the auction. In the second step the winner of the first step can acquire a longer time slot initiating another auction, in this auction also the vehicle following the winner of the first step can bid with the other vehicles placed in the first places of lanes.

Finally, Vasirani and Ossowski [37] proposed a novel approach to manage urban intersections, utilizing a reservation-based intersection control model [38] and on market-inspired rules. They evaluated two different scenarios, one with a single intersection and the other with a network of intersections. The first scenario was used to analyze the performance of a policy that employs combinatorial auctions to allocate slot reservations. The second scenario was used to assess the impact of a traffic assignment strategy, which is inspired by competitive markets, on drivers' route choices. The authors then combined these two strategies to propose an adaptive management mechanism that combines the auction-based traffic control policy and the competitive traffic assignment strategy.

More in general, have also been utilized to manage resources other than intersections, such as parking slots and fleets. In particular, managing parking spaces for private and individual vehicles is a challenging problem of resource allocation coordination. Various solutions have been proposed [39], including the use of auctions to implement a negotiation approach in which each vehicle actively participates in the coordination process by advancing its proposals [40, 41]. Additionally, for fleets of special vehicles such as emergency vehicles or taxis, which typically have a limited number of units, auctions have been used to determine which users to serve first when demand exceeds the available resources [42].

In this body of work, the presence of both human-driven and autonomous or connected vehicles is taken into account. Specifically, the proposal described in Section 3.2.2 accounts for the simultaneous presence of these two types of vehicles, which is crucial in addressing the transition from an exclusively human-driven vehicular environment to one that is fully autonomous [38]. This co-presence of vehicles necessitates the consideration of various constraints arising from the unpredictability and potential untrustworthiness of human-driven vehicles. Additionally, the proposed approach in Section 3.2.2 allows for the participation of all vehicles present in a lane in the auction, rather than just those at the head of the lane. This is a critical aspect in ensuring the efficient and effective management of intersections.

The approach presented in the subsequent section builds upon the research presented in [43] and addresses the challenges associated with the co-existence of human-driven and autonomous vehicles. The approach accounts for the limitations of human-driven vehicles and outlines methods for equipping these vehicles to interact with autonomous and connected vehicles. Furthermore, the interactions and actions are designed to accommodate the limitations of human-driven vehicles.

3.2 Proposals for Local Coordination

3.2.1 Proposal for Traffic Lights Management

In the proposed scenario, it is assumed that both autonomous and human-driven vehicles co-exist. The proposed crossing management system is designed for intersections that are already equipped with traffic lights. The system aims to convert the *traditional* functionalities of traffic lights into a new, more efficient system. The decision to install a traffic light at a particular intersection can be based on the fact that the intersection is frequently congested or poses a significant safety risk. Therefore, it is deemed beneficial to implement a controlled crossing management system at the intersection.

Auctions

The proposed system utilizes an auction mechanism to regulate access to a crossing for equipped vehicles. Each equipped vehicle approaching the crossing participates in an auction to secure the next green light for its lane. The system collects bids from all lanes throughout the duration of the current green light period. The bids are then summed per lane, and this value represents the lane's bid b_l. To ensure fairness, the module scales down the total bid from the lane currently displaying the green light by a factor σ. Formally, each b_l is calculated as in eq. 3.1 where B_l is the set of bids placed by vehicles in lane l.

$$b_l = \sum_{b \in B_l} \frac{b}{\sigma} \tag{3.1}$$

The lane with the highest bid is then selected as the winner of the auction and is granted the next green light for a predefined period of time, denoted as Γ.

Vehicles

The proposed system consider all the vehicles categories described in Section 2.1. In particular, the difference between *equipped* and *non-equipped* is important since the former are equipped by means of an integrated software or a physical ad-hoc device, to autonomously participate in the proposed auction mechanism without the intervention of a human, and will always behave according to the mechanism. The latter will not directly intervene in the auction and will behave by following the directions given by traffic lights.

Vehicle budgets

In this proposed system, equipped vehicles are given a budget in virtual coins that will be used to place bids when approaching traffic lights. The budget allocated to each trip is determined by the vehicle based on the importance or urgency of the trip. The method of obtaining virtual coins and the decision-making process for allocating the budget for a specific trip are not the focus of this proposal. However, it can be noted that in a real-world scenario, virtual coins can be implemented by local administrations using various policies such as being bought with real money, given as an incentive for good driving practices like driver's adherence to driving regulations, or even proportionally to the time the vehicle is idle to encourage the use of alternative means of transportation, participation in carpooling, or the adoption of low-emission or environmentally-friendly vehicles. Ultimately, the implementation of virtual coins and budgets is a political, social, economic, and ethical issue related to fair and equitable mobility regulations.

The choice of budget allocation for a specific trip can be made by drivers based on their needs, such as time constraints or the need to make other trips before obtaining more budget. Over time, drivers will learn to manage their budget in a similar manner to managing data usage on smartphones.

Bids

Equipped vehicles, which have the capability to communicate with the city infrastructure, participate in these auctions by computing their route and allocating a budget for the trip analogously to [32]. The bid placed by each equipped vehicle is determined by dividing the allocated budget by the number of intersections on the route. Formally if the route goes through $\mathcal{I}$ intersections for which a bid is necessary, and let $\mathcal{B}_t$ be the trip budget allocated for a such route, then each bid is set to $\frac{\mathcal{B}_t}{\mathcal{I}}$. This ensures that vehicles do not run out of budget before reaching their destination. The number of intersections for which a bid is necessary is known at departure time since the final destination is defined by the user and the route to the destination is computed.

On the other hand, non-equipped vehicles, which lack the capability to communicate with the city infrastructure, are unable to place bids themselves. Therefore, the system places bids for these vehicles and for autonomous vehicles that have exhausted their budget. The system calculates the average bid placed by equipped vehicles participating in a specific auction and uses this value as the bid for non-equipped vehicles. This allows

non-equipped vehicles to behave as an average equipped vehicles. In case there are only non-equipped vehicles at the crossing, the system selects the lane with the larger number of vehicles as the winning lane.

Traffic Lights Management System

The proposed approach for managing signalized intersections in a smart city scenario involves the use of an auction system described above, in which vehicles make bids for the right to access the intersection. This approach is designed to accommodate the co-existence of both autonomous and human-driven vehicles.

Vehicles in the lanes make bids for the next green light based on the budget allocated for their current trip. The lane with the highest total bid is granted the next green light by the crossing management system. This system is implemented on a physical device placed at the crossing site, which controls the traffic light and regulates access to the crossing.

The proposed system comprises two main functions: the management of the auction and the selection of the next green phase, and the management of interactions between vehicles and traffic lights. The first function is responsible for collecting bids placed by involved vehicles, placing bids for non-equipped vehicles, processing this information, and determining the next green lane based on predefined rules. The second function is responsible for receiving bids from vehicles, utilizing sensors to count the number of vehicles in the lane (both equipped and non-equipped), and controlling the traffic lights by setting the green light for the lane that has been selected as the winner of the auction and setting the red light for all other lanes.

Traffic lights

The proposed system utilizes traffic lights as a key component. As in traditional traffic lights, the light turns green in a lane if vehicles in that line are allowed to pass the crossing. However, it should be noted that the system only allows one lane at a time to display the green light, while all other lines display the red light. This ensures that vehicles in lanes with the red light must wait until their turn to pass the crossing, avoiding any potential conflicts or accidents. So the system does not take into consideration the possibility of allowing vehicles from different lanes to pass simultaneously, as is the case with the "right turn on red" rule in the United States, as it is not the focus of the proposed system and is not investigated in the experiments.

Both green and red lights are displayed even when all vehicles in the

lanes are autonomous. This serves to inform passengers in autonomous vehicles of their vehicle's future behavior, as well as to provide guidance to human-driven vehicles that may be approaching or are already in the lane.

The green light has a minimum display time, denoted as Γ, that is set to ensure safe crossing for both autonomous and human-driven vehicles. The minimum display time has been predetermined, and although it is possible to establish a proportional relationship with the winning bid, investigating the influence of this parameter on the system's outcome is beyond the scope of this thesis and will be examined in future research.

3.2.2 Proposal for Intersections Management

The proposed scenario involves the deployment of a management system at each intersection, which is responsible for determining the priority of vehicles crossing the intersection through the use of an auction mechanism. Vehicles located at the front of each lane participate in the auction by submitting bids, the amount of which is based on their budget. Additionally, the bid submitted by the front lane vehicle may be augmented through the consideration of factors such as the number of vehicles in the lane or by allowing vehicles in the rear of the lane to contribute to the bid.

In this scenario, the coexistence of autonomous and human-driven vehicles is taken into account, and all vehicles are equipped to participate in the intersection management mechanism through an auction process. The vehicles at the front of each lane make bids, which are determined based on factors such as their budget and the number of vehicles in the lane. Additionally, the bids of vehicles at the back of the lane may also be taken into consideration. Importantly, the participation of human-driven vehicles in the auction process does not require direct human intervention, as they are equipped with technology such as mobile apps or infotainment systems that facilitate communication with the management system and provide instructions to drivers. In this scenario, the co-existence of human-driven and autonomous vehicles is considered, and all vehicles are equipped to participate in the intersection management mechanism through an auction process. However, due to the unpredictable and untrustworthy nature of human-driven vehicles, coordination mechanisms that assume knowledge of vehicles' trajectories or forecasts of behaviors cannot be implemented. This includes policies that aim to make the clearing of intersections more efficient by allowing multiple vehicles to access the intersection simultaneously if their trajectories do not conflict, as previously proposed in literature such as [32]. The lack of knowledge and trust in the behavior of human-driven

vehicles makes such strategies infeasible.

Therefore, in the following, it is assumed that only one vehicle at a time is granted access to the intersection and that the only instructions provided to human drivers, upon reaching the start of a lane, are to either "stop" or "proceed" through the intersection.

The proposed system aims to improve latency at intersections by utilizing auction mechanisms for autonomous vehicles. The system aims to demonstrate the effectiveness of auction mechanisms in reducing vehicle latencies compared to standard traffic yield rules. Additionally, the system aims to investigate the feasibility of achieving differentiated latencies, where higher bids result in smaller latencies. However, it is not immediately clear that simply placing high bids will result in reduced latencies, as vehicles must wait in a first-in-first-out (FIFO) manner to approach an intersection.

In the following of this section, it is introduced the mechanisms that are used in the experiments.

Bidding Strategies

The proposed system utilizes virtual coins as a means for setting bids in an auction-based approach for managing intersections. Vehicles are assigned a budget in virtual coins upon joining the system. As for the traffic lights scenario (specifically Section 3.2.1), the implementation of virtual coins and the assignment of budgets is not the primary focus of the proposed system and it can be influenced by various factors such as regulations set by local authorities, social and economic considerations, and ethical considerations.

Two distinct bidding strategies have been formulated in the design of the system:

- **Randomized (Rand):** vehicles place a random bid that falls inside the range of their budgets. If a vehicle runs out of budget, the system will let it make a minimal bid anyway.
- **Route dependent (Prop):** vehicles allocate their budgets by dividing them by the total number of intersections they will pass through during their trip. Bids are placed at each intersection based on this calculation. It is assumed that if routes are fixed and do not change during the trip, vehicles will not run out of budget. To ensure human-driven vehicles adopt this strategy, it is required for them to input their final destination into their navigation system at the start of their

trip. The auction mechanism will then calculate the shortest route from the starting point to the destination, considering it the expected route, and will compute the bid amount accordingly. If the driver chooses to deviate from the expected route, the bid amount will be adjusted accordingly.

In the experiments, it is assumed that virtual coins are indivisible, thus bids placed by vehicles will be in integer form. However, if in a real-world scenario, virtual coins are divisible, bids may be made with a precision that reflects the number of decimal places of sub-multiples of virtual coins.

A budget recharge mechanism is also devised in which the amount of the bid placed by the winner of an auction is redistributed among the other vehicles that participated in the auction. This mechanism is designed to prevent vehicles from running out of budget early in their trip, particularly for the Rand bidding strategy, but it will also be tested for the Prop bidding strategy. While it may seem unnatural or unfair to redistribute a vehicle's budget to others, particularly if budgets are acquired with real money, it could be deemed worthwhile if there is a common benefit to doing so.

Auction Resolutions

It is devised two different approaches for auction resolution:

- **Cooperative approach (COOP):** all the vehicles at the front of the lanes at the intersection make their bid, and all bidding vehicles will go through the intersection according to the bid order (highest first).
- **Competitive approach (COMP):** all the vehicles at the front of the lanes at the intersection make their bid, but only the vehicle that wins the auction gets to pass the intersection. Vehicles that lost the auction will have to attend and win a successive auction before being able to go through the intersection. Note that this may imply *starvation*.

Moreover, it is considered two different methods for the bid payment:

- **All-Pay (AP):** All bidding vehicles are charged for their bid.
- **Only-Winner-Pays (OWP):** only the vehicle that wins the auction will be charged for its bid, while vehicles that lost the auction are not charged.

In a previous study presented in [44], the effectiveness of four strategies resulting from all possible combinations of auction resolution approaches and

payment methods was investigated, using simple randomized bidding strategies. The study found that two of the four combinations, namely strategies COOP-AP and COMP-OWP, were of particular interest and deemed more natural to interpret. For this reason, the experiments will focus exclusively on these two strategies. These strategies are complementary in nature, with the first strategy requiring all vehicles to pay their bids in exchange for the opportunity to proceed through the intersection after the auction is complete. However, the order of passage through the intersection is not determined in advance. This round-robin-like approach ensures that all lanes proceed at each auction round. Conversely, in the second strategy, the movement of lanes is dependent on the lane of the winning bid, thus resulting in different speeds for each lane. The effectiveness of these strategies is evaluated by measuring the average time to clear intersections, calculated as the elapsed time from the moment vehicles reach the front of the lane and the moment they exit the intersection, and the average waiting time in the lane, under varying traffic conditions. The results of the study demonstrated that the average time to clear the intersection was minimal for the strategy COOP-AP, while it slightly increased for the strategy COMP-OWP. Additionally, the variance of the average values for both strategies was found to be small. With respect to the average waiting time in the lane and its corresponding variances, there was no significant difference between the strategies COOP-AP and COMP-OWP under the same traffic conditions, with the latter being slightly higher than the former.

Finally, in the competitive approach, two mechanisms have been devised to allow vehicles that are not at the front of a lane to participate in the auctions: the "enhancement" and "sponsorship" mechanisms, which will be described in the following paragraphs.

Enhancement. The *enhancement* mechanism aims to balance the waiting times of lanes with a large number of vehicles in line with those of lanes with a smaller number of vehicles in line. This is achieved by adjusting the bid placed by the front vehicle of a lane based on the number of vehicles waiting in that lane. Specifically, for each lane ℓ with n_ℓ vehicles in line and a bid b_ℓ placed by the front vehicle, the enhancement $en(\ell)$, that is the actual bid for the front vehicle, is mathematically defined in equation (3.2).

$$en(\ell) = b_\ell(\ln n_\ell + 1) \tag{3.2}$$

The result of *enhancement* mechanism is a real number instead an integer number as the bid since it is performed after the bid placing. It should be

noted that the enhancement value is equal to the bid placed by the front vehicle when there is only one vehicle in the lane. As the number of vehicles waiting in the lane increases, the logarithmic multiplicative factor becomes more significant. This mechanism allows for implicit vehicle involvement in the auction, as their presence in a lane influences the bid placed by the front vehicle without requiring them to place a bid themselves.

Sponsorship. The *sponsorship* mechanism allows vehicles behind in lanes to contribute to the bid of the vehicle at the front of the lane. Vehicles that are not at the front of the lane can place bids that will be added to the bid of the front vehicle. Only the sponsored vehicle will be charged for the bid if they win the auction. This mechanism is intended to allow vehicles to speed up their lanes at the expense of others. In the cooperative approach, enhancement and sponsorship are not used, as all vehicles are guaranteed to pass through the intersections at each auction round, regardless of their bid outcome. On the contrary, in the competitive approach, lanes may experience starvation, thus these mechanisms are implemented to mitigate this issue.

3.3 Local Coordination Experiments

3.3.1 Traffic Lights Management Experiments

The goal of the experiments is to evaluate the effectiveness of the proposed system in reducing waiting times at crossing sites by incorporating the auction mechanism for traffic light control. This is accomplished by comparing the performance of the proposed system with that of the traditional Fixed-Time Control (FTC) approach [15] in managing the traffic flow at crossing sites. The results of these experiments will provide insights into the feasibility and potential benefits of incorporating the auction mechanism for traffic light control in smart cities.

The objective of the experiments is to quantitatively evaluate the time vehicles spend traversing intersections controlled by traffic lights during their trips. Specifically, the focus is on measuring the time duration between a vehicle's entry into a road segment immediately preceding an intersection with a traffic light and its exit from the same segment. This approach enables the assessment of the performance of the proposed coordination system in terms of reducing waiting times at intersections, by comparing the results

obtained from using traditional Fixed-Time Control against the proposed auction-based system.

To understand how the approach performs, it is set up three experiments within the scenario described in Section 3.3.1:

Experiment 1. In this experiment it is measured delays of 2000 equipped and non-equipped vehicles when using our crossing management system and varying the percentage of equipped vehicles. As a baseline it is considered vehicle delays, under the same experimental settings, occurring when traffic light systems use the FTC policy.

Experiment 1bis. Similar to Experiment 1 but the number of vehicles is decreased to 500, so to be able to study the impact on traffic light waiting times with a significantly lower number of road users.

Experiment 2. In this experiment it is studied how delays vary when varying the budget of a population sample and the size of the population with varied budgets, under different conditions:

1. 2000 vehicles, half equipped and half not equipped, with 200 equipped vehicles getting their budget raised gradually up to twice the initial budget;

2. 5000 agents half equipped and half not equipped, varying the percentage of agents that increase their budget and gradually double their budget;

3. 2000 agents, gradually varying the percentage of agents that increase their budget up to 40% and gradually increasing their budget up to 2000%.

Scenario

The proposed experiments take place within a 1-kilometer-wide smart urban area located in the city of Modena, Italy. This area is part of the CLASS Horizon 2020 project[1] and is equipped with advanced smart sensors, actuators, and communication infrastructure. The area's infrastructure allows for the collection and real-time processing of vast amounts of data on urban

[1] https://class-project.eu

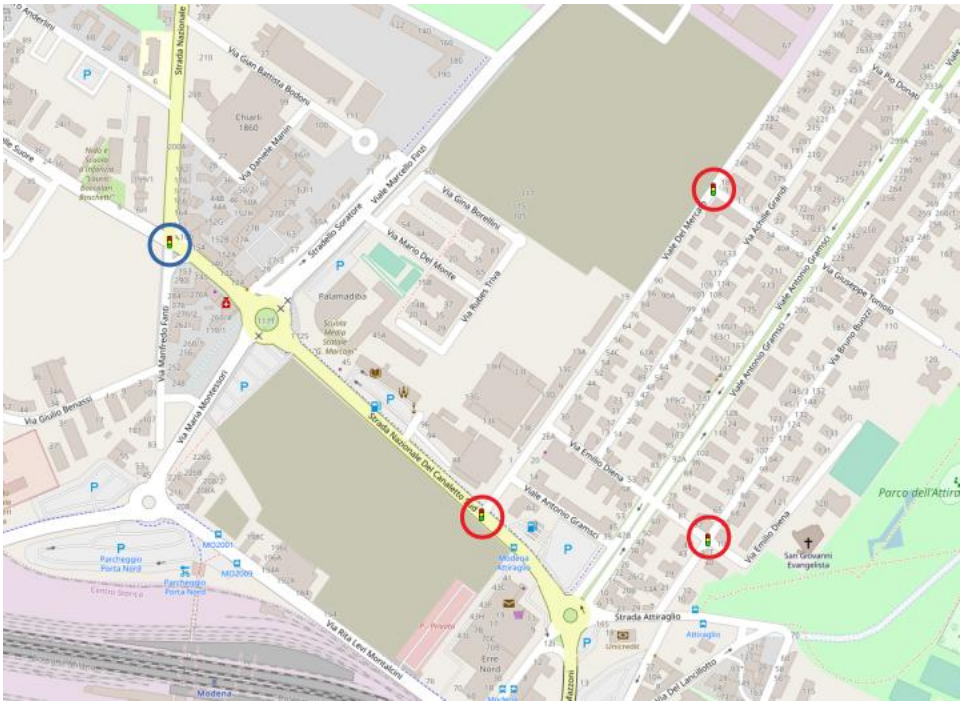


Figure 3.1: Modena Automotive Smart Area (MASA). The traffic light circled in blue is the only one really exists, those circled in red have been added for the sake of experiments.

traffic, which is used to communicate with connected vehicles. These vehicles are equipped with various sensors, actuators, and V2X connectivity, which enhances both the driving experience and the overall safety of the city. Advanced urban mobility applications are deployed to efficiently coordinate vehicles and city resources through the use of data-in-motion and data-at-rest analytics.

Map. The part of the city described above, and that will be considered for the experiments, is depicted in Figure 3.1. From now on, this area will be referred to with the acronym *MASA* (Modena Automotive Smart Area).In order to study the agents' behavior on a more complex scale, three traffic lights that are not present in the considered area are artificially added (see Figure 3.1), and it is reached a total of four traffic lights.

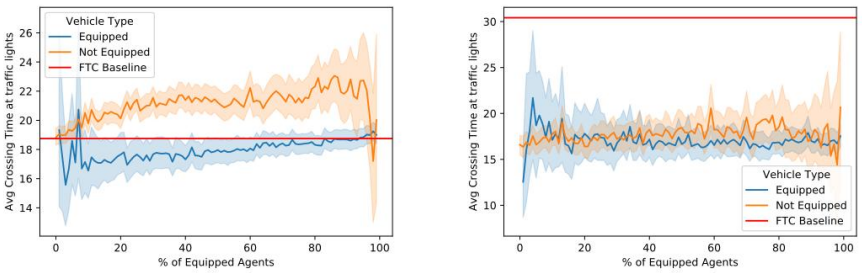
Simulator. The MASA area was reproduced within the MATSim multi-agent simulator for urban transportation [45]. Furthermore, it is implemented and added to the MATSim simulator four additional modules for enhancing its ability to simulate smart city related scenarios. In particular, the *communication module* described in Section 2.2 is used in order to allow agents to send bids to traffic lights, the *perception module* is used in order to allow traffic lights to locate not equipped agents, and finally, an *analysis module* is implemented to register values needed for the analysis.

Vehicles. The standard simulation population consists of 2000 agents with departure times spread during the 24 hours of the day. Each vehicle performs two trips a day, from home to work and back, and facilities (i.e. departure and destination locations) are randomly distributed all over the MASA map. Departure times follow a Gaussian distribution (when departing from home the distribution has its peak at 9 AM and on the way back the rush hour is set to 6 PM), to simulate variations in daily traffic.

Budget. Concerning vehicles' individual budget, to simulate variability in budget disposal, each vehicle is associated with a class of budget among the following: *low budget*, with the interval of possible values [1..28]; *average budget*, with the interval of possible values [38..65]; *high budget*, with the interval of possible values [75..101]. In the simulations, it is set to 1/3 of the vehicles in each budget class. The actual trip budget is randomly chosen within the previously assigned budget class range. The scaling factor σ for the bid of the lane with green light is set to 10.

3.3.2 Traffic Lights Management Results

Experiment 1. The graph in Figure 3.2a illustrates the results of an experiment that measures the time spent by vehicles crossing intersections with traffic lights during their trips. The x-axis of the graph represents the percentage of equipped vehicles, and the y-axis indicates the average delay caused by waiting in line before crossing intersections with traffic lights. The blue line pertains to equipped vehicles, the orange line pertains to non-equipped vehicles, and the red line represents the average delay when traffic lights are managed using the Fixed-Time Control (FTC) policy. The results indicate that, with the exception of situations in which the percentage of equipped vehicles is very low or very high, equipped vehicles experience shorter delays than non-equipped vehicles. Furthermore, the latency of equipped vehicles in auction-managed crossings is shorter than when using



(a) 2000 vehicles.

(b) 500 vehicles.

Figure 3.2: Average delays (in sec) incurred by vehicles to cross the intersections with traffic lights during their trip, by varying the rate of equipped agents, grouped by Vehicles Type (equipped and non-equipped). The red line indicates the average delay in the scenario with FTC policy. Bands around average delays represent the average confidence.

the FTC policy. These findings suggest that during the transition from non-equipped to equipped vehicles, it is advantageous to be equipped, providing an incentive for vehicles to become equipped.

On the other hand, when the percentage of equipped vehicles is low, it may be difficult to determine if there is a clear advantage to being equipped. The delay experienced by equipped vehicles may be affected by various factors such as the positioning of the bidding vehicle within the lane, the presence of other actively bidding vehicles in the same lane, and so on. It is important to note that in scenarios with a low percentage of equipped vehicles, the results may be highly dependent on the specific traffic conditions at the intersection sites.

In the case when the percentage of equipped vehicles is high, the results show that the average delay for equipped vehicles is worse than when using the Fixed-Time Control (FTC) policy. This is likely due to the fact that the auction strategy may cause delays for a lane for more than one round of green lights, which does not occur in the FTC strategy. Additionally, it can be observed that in this scenario, the few non-equipped vehicles experience significant advantages due to their small numbers. This outcome is not unexpected, as the high number of vehicles in the simulation is likely to fill all lanes at each intersection, leading to the aforementioned issues with the auction strategy.

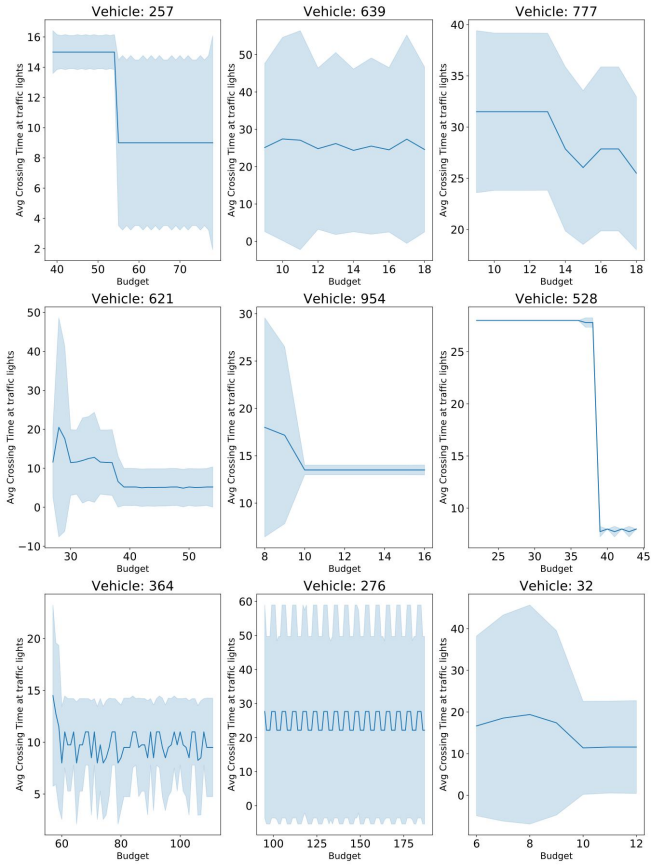


Figure 3.3: Avg waiting time (in sec) at traffic lights (y-axes) during the whole trip, when varying their budget (x-axes), for single vehicles.

Experiment 1bis. To test the last statement, the experiment was repeated with 1/4 of vehicles, namely 500, to see if intersection congestion might be the cause of such a small difference between the two policies when the percentage of equipped vehicles is high. Results in Figure 3.2b show that, with lighter traffic, the auction policy always outperforms the FTC approach. Moreover, there is no great difference between equipped and not equipped vehicles. Motivation is to be sought in the fact that with light traffic and auctions, vehicles do not have to wait for green lights whenever the other lanes are empty.

Experiment 2.1 Simulations were conducted by incrementally increasing the budgets of 10% of vehicles and analyzing the variations in delays experienced by the same vehicle on the same trip at the same time of day. Not completely unexpected, it was observed that only 10% of vehicles with increased budgets experienced benefits in terms of reduced delays. It can be inferred from the results that, even when a single vehicle has a high budget, it is unable to gain priority at the intersection due to the serialized nature of access. This is because, if other vehicles in the same lane have placed low bids, a high bid alone may not be sufficient to win the auction against other lanes, even if those lanes have no vehicles with a high budget. This outcome suggests that devising bidding policies to ensure that vehicles traveling with different levels of urgency are able to increase their travel speed solely through increasing their bids may not be straightforward.

In order to ascertain the reasons for their favorable outcomes, the cases in which delays were reduced were further investigated. Figure 3.3 presents some representative examples of variations in delays. The x-axis of each plot displays the trip budget, while the y-axis shows the average waiting time at intersections with traffic lights.

The majority of plots not shown exhibit similar behavior to that of vehicle 639, in which there is no significant variation. However, for some vehicles, an increase in budget results in a reduction in delay, while for others, the behavior does not stabilize as effectively. However, the results of the experiments indicate that there are variations in the benefits obtained by increasing the budget of vehicles. While it is observed that only a small percentage of vehicles experience a reduction in delays, the reasons for this phenomenon are not entirely clear. Further investigation of the number of intersections, starting budget, and travel times may provide insights into the underlying causes. However, it is noted that in situations where the number of intersections is smaller, the behavior of the system becomes more predictable and stable when the budget is increased. This is evident in the cases of vehicles 257, 777, 954, and 528, which pass through only one intersection per trip, compared to vehicles 364 and 276, which pass through two and three intersections, respectively. Despite the fact that both vehicles 257 and 777 are traveling in heavy traffic (around 9 AM and 6 PM), their starting and final budgets are vastly different (40 and 10, respectively). On the other hand, vehicles 639 and 954 have similar initial budgets (9 and 8, respectively) and depart at similar times of the day (leaving home at 11 AM and 10 AM, respectively, and leaving work at 3 PM and 4 PM, respectively). Despite this, the former exhibits consistent behavior while the latter receives a benefit. The only discernible difference is that vehicle 639 has one

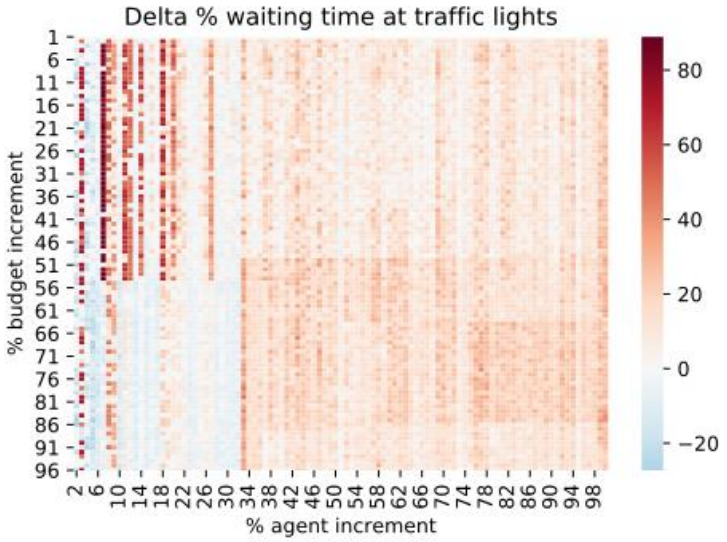


Figure 3.4: % of increase of waiting time at traffic lights.

additional intersection to traverse.

Experiment 2.2 In the second experiment, the time a vehicle waits for a green light during its trip and the total duration of the trip is measured. The results of vehicles that traverse at least three traffic lights during their trip are presented, as these intersections have the most substantial impact. Figures 3.4 and 3.5 are heat maps that display the average percentages of increase in waiting times of vehicles and the percentage of increase in trip time, respectively, using a sample of 5000 vehicles over a 24-hour period. The x-axis represents the percentage of agents that have an increased budget, while the y-axis represents the percentage of budget increment. It is observed that beyond 30%-40% of vehicles having an increased budget, there is no significant benefit in further augmenting budgets. On the other hand, when up to 30% of vehicles have an increased budget, an unexpected outcome is observed. It appears that a small increase in budget leads to increased waiting times and total trip times, while larger increments result in some benefits. This may be due to the high traffic conditions in the simulation, the fact that the heat maps display average delays, or even that the budget increase was not sufficient to produce a significant impact in certain situations. As such, further experimentation is conducted to gain a deeper

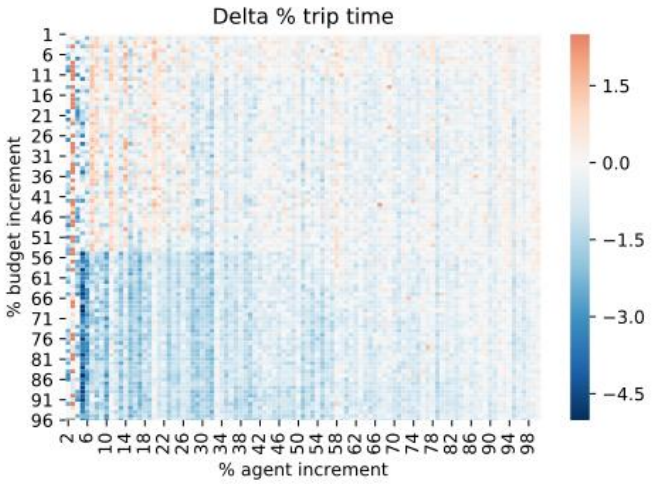
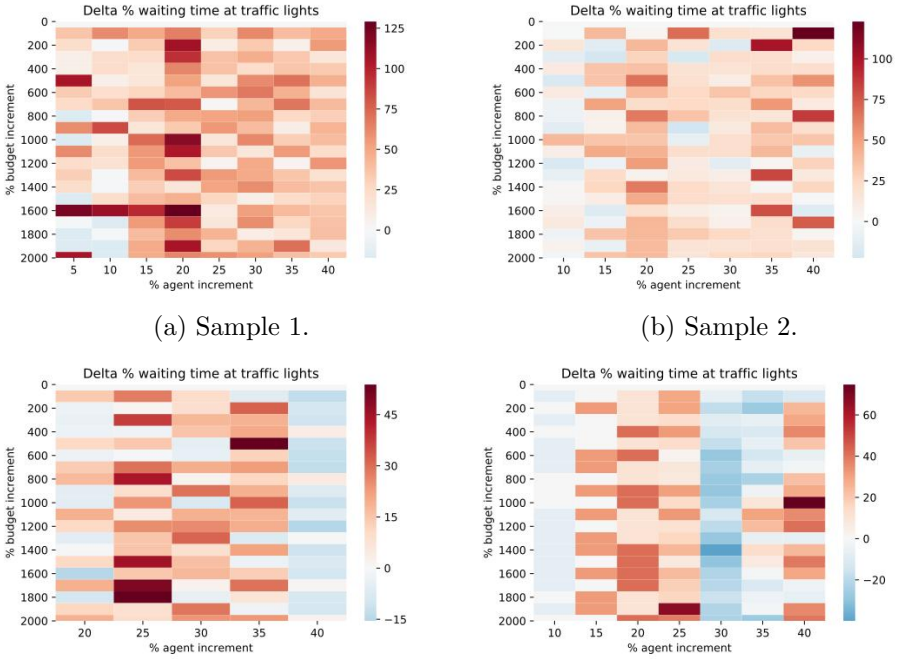


Figure 3.5: % of increase of trip time.

understanding of these results.

Experiment 2.3 The waiting times at traffic lights were further analyzed by conducting experiments where up to 40% of vehicles have an increased budget, and progressively increasing the budget up to 2000%. Subsequently, the results for individual vehicles were examined. Figure 3.6 illustrates heat maps for four representative samples of the results, each corresponding to a different vehicle. It is apparent that various situations may arise, with no clear patterns evident. Instead, the results appear to be random: (1) vehicles with high budgets sometimes experience large positive delta delays (e.g., Figure 3.6c with 1800% budget increase); (2) vehicles with a small increase experience large negative delta delays (e.g., Figure 3.6d with 200% budget increase); (3) when a limited number of vehicles have increased budgets, some vehicles may experience benefits (e.g., Figure 3.6b with 10% increase), while others do not (e.g., Figure 3.6c with 30% increase). Similarly, when a greater number of vehicles have increased budgets, some vehicles may benefit (e.g., Figure 3.6d with 30% increase) while others do not (e.g., Figure 3.6a with 30% increase). Further investigation will be needed, but these results suggest that traffic conditions and vehicle routes are more determinants than budget.



(a) Sample 1.

(b) Sample 2.

(c) Sample 3.

(d) Sample 4.

Figure 3.6: Single vehicle heat map, with an increasing budget.

3.3.3 Intersection Management Experiments

The experiments are conducted using MATSim urban simulator.

The study utilized a Manhattan grid map with 8×8 intersections, each having one lane for each direction. The routes for the vehicles were randomly generated, with starting and ending points selected at random on the map and the route being determined as the shortest path between the two points. The departure times for the vehicles were modeled using two Gaussian distributions, one peaking at 9 AM and the other at 6 PM, to simulate daily traffic patterns. Each vehicle was assigned an initial budget with a default value. The experiments were conducted with varying numbers of vehicles to simulate different traffic conditions.

In each experiment, a specific scenario was defined by selecting one of the two auction resolution approaches (COOP-AP or COMP-OWP) and one bidding strategy (Rand or Rd), with or without budget redistribution. Additionally, for the COMP-OWP approach, the effectiveness of the enhancement and sponsorship mechanisms were separately evaluated by enabling or disabling them (respectively Enh and Spon). In order to provide a comparison, experiments using only standard yield rules were also conducted (referred to as Priority in figures). The waiting times were recorded in seconds.

For each experiment, the following measures are reported:

- Statistics on vehicles *average waiting times at intersection* before they are given the right to pass the intersection: from the moment vehicles reach the front of the lane, to the moment they win an auction, in the COMP-OWP case, or to the moment it is their turn to pass through after the auction is over, for the COOP-AP case.

- Statistics on *average times vehicles spend waiting in lanes*: compute the average time each vehicle spends waiting in the lane, from the moment it queues up, to the moment it has cleared the intersection.

In the considered scenario, if a vehicle's destination lane is filled, the vehicle is required to wait at the front of its current lane before crossing the intersection, and no other vehicle is permitted to proceed. This is deemed the safest solution, as it prevents any potential incidents that may arise from human-driven vehicles deviating from predetermined trajectories. Alternative solutions, such as allowing other vehicles to pass the intersection even if they did not win the auction, may be viable when all vehicles are autonomous, however, given the unpredictability of human-driven vehicles, it is not feasible for the intersection management system to know in advance their

trajectories. Furthermore, making real-time changes to the order in which vehicles are permitted to pass through intersections may lead to confusion and errors on the part of human drivers, increasing the risk of incidents.

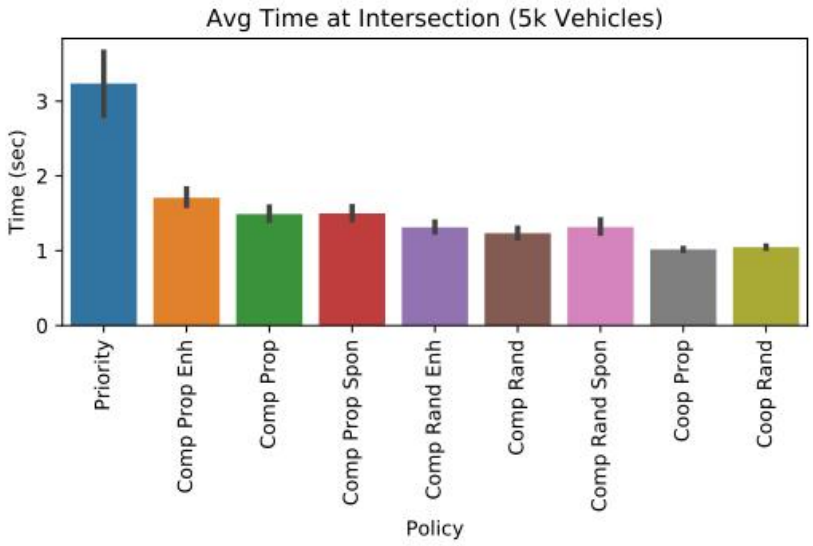
3.3.4 Intersection Management Results

In this section, the results of the experiments outlined in the previous section are presented. In Section 3.3.4, the results of the comparisons between different scenarios and the latency experienced when using standard yield rules are presented. In Section 3.3.4, results related to achieving differentiated latencies are presented.

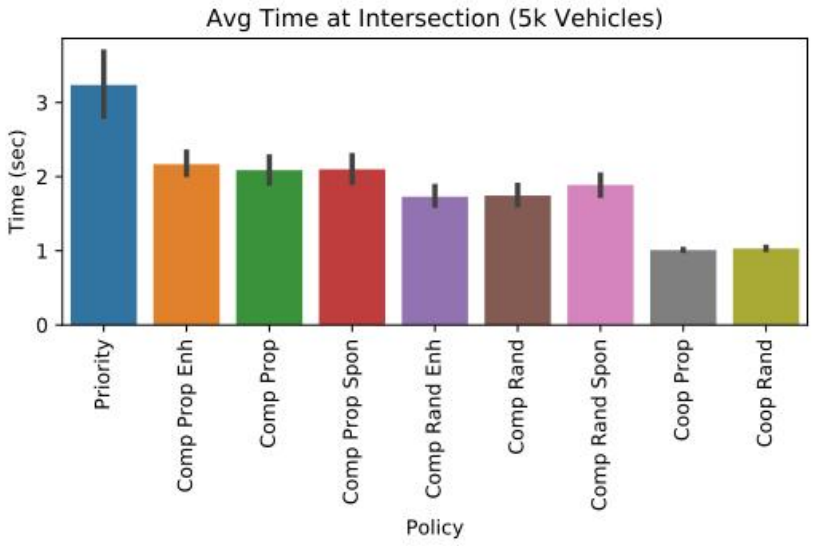
About reducing latencies

Simulations were conducted with 5,000 vehicles operating on the map to simulate heavy daily traffic conditions. The results of the experiments are presented in Figures 3.7, which show the average waiting times for vehicles at the front of a lane, or the average time before being granted the right to cross the intersection. The results are presented for two cases: when the budget recharge mechanism is adopted and when it is not. The first observation is that, for any combination of strategies, on average, vehicles are able to cross the intersection faster than when using standard yield rules (as indicated by "Priority" in the figures).

As previously established in the preliminary experiments presented in [44], it is expected that among all auction policies, the COOP-AP strategy would result in the smallest waiting times across all alternatives. This is due to the fact that under this strategy, vehicles only participate in one auction per intersection and will never have to wait for more than one vehicle per lane. This is evident in the results, as the variance in waiting times is observed to be very small for all COOP-AP cases. In the case of the COMP-OWP approach, it has been observed that the difference between redistributing or not the budget is evident only in terms of absolute waiting times, with the former resulting in smaller times. However, the relative behavior of different policies remains unchanged. Additionally, it has been found that the average waiting times achieved with the Rand bidding strategy are slightly lower than those achieved with the Prop strategy. This is due to the fact that the bids of the same vehicle in the former strategy range across the entire budget interval, leading to a lower likelihood of losing several auctions in a row. In contrast, the bids of the same vehicle in the latter strategy are always very similar. Finally, when implementing the enhancement and sponsorship



(a) With budget redistribution.



(b) No budget redistribution.

Figure 3.7: Average waiting times at intersections before the right to pass is granted, with 5k vehicles running on the map. Vertical black lines represent the standard deviation. (a) Budget recharge mechanism is adopted. (b) Budget recharge mechanism is not adopted.

mechanisms, it is observed that there is a slight increase in average waiting times compared to not implementing them. This is due to the fact that while some vehicles experience lower latencies with the implementation of these mechanisms, a larger number of vehicles experience longer waiting times.

The total waiting times on lanes, including the time spent queuing up until clearing the crossing by entering the next lane, were also measured for the cases where the budget is redistributed and not redistributed. The results indicate that there is no significant difference between the two scenarios. Unexpectedly, any benefit exhibited when evaluating only auction times at intersections is not reflected in the larger picture of the vehicles' journey. The average, standard deviation, and maximum times for vehicles to go from one lane to the successive are similar across all strategies. As the differences are insignificant and not evident when plotted, no plots are provided. One possible reason for this outcome could be that lane latencies are more influenced by overall traffic conditions rather than the speed at which a vehicle is able to go through intersections via auctions.

Hence, in subsequent experiments, the number of vehicles running on the map varied. The results are shown in Figures 3.8 and Figure 3.9. The plots in (a) and (b) refer to average traffic conditions (2500 vehicles), while the plots in (c) and (d) refer to light traffic conditions (200 vehicles). It can be observed that the overall trend does not change, with the exception of the absolute values, which are smaller for lighter traffic conditions.

With light traffic conditions, a slight increase in standard deviation is observed, indicating a larger variation in waiting times between different configurations of lanes and vehicles at intersections. However, on average, the waiting times are similar across all policies, as there are a limited number of vehicles, and any policy works effectively. Additionally, it is observed that all policies incur comparable maximum waiting times, which can be attributed to the limited number of configurations that occur with light traffic, resulting in similar worst-case scenarios.

As for the average traffic condition setting, it is observed that there are some differences in maximum waiting times. These differences can be explained by the fact that such values may be experienced by only one or a few vehicles. For example, in the case of COOP-AP-Rand, it is possible that a vehicle experienced a longer wait time due to a larger number of small bids placed by the front vehicles in that lane. Similarly, for the COMP-OWP with the enhancement mechanism, a longer maximum waiting time may have occurred because a vehicle had to wait longer due to a long lane that was advantaged by the mechanism. When analyzing average waiting times, it can be observed that there is little variation among the different

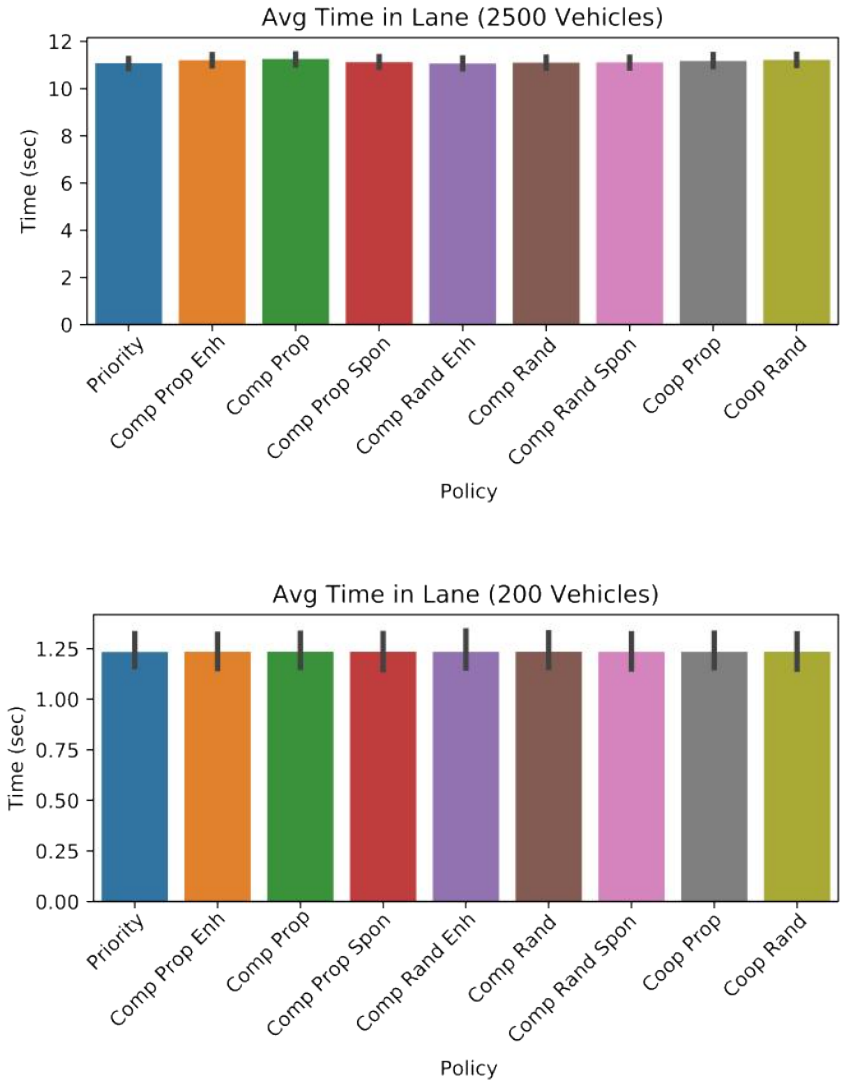


Figure 3.8: Average waiting times spent waiting in the lane, with 2500 (a) and 200 (b) vehicles respectively.

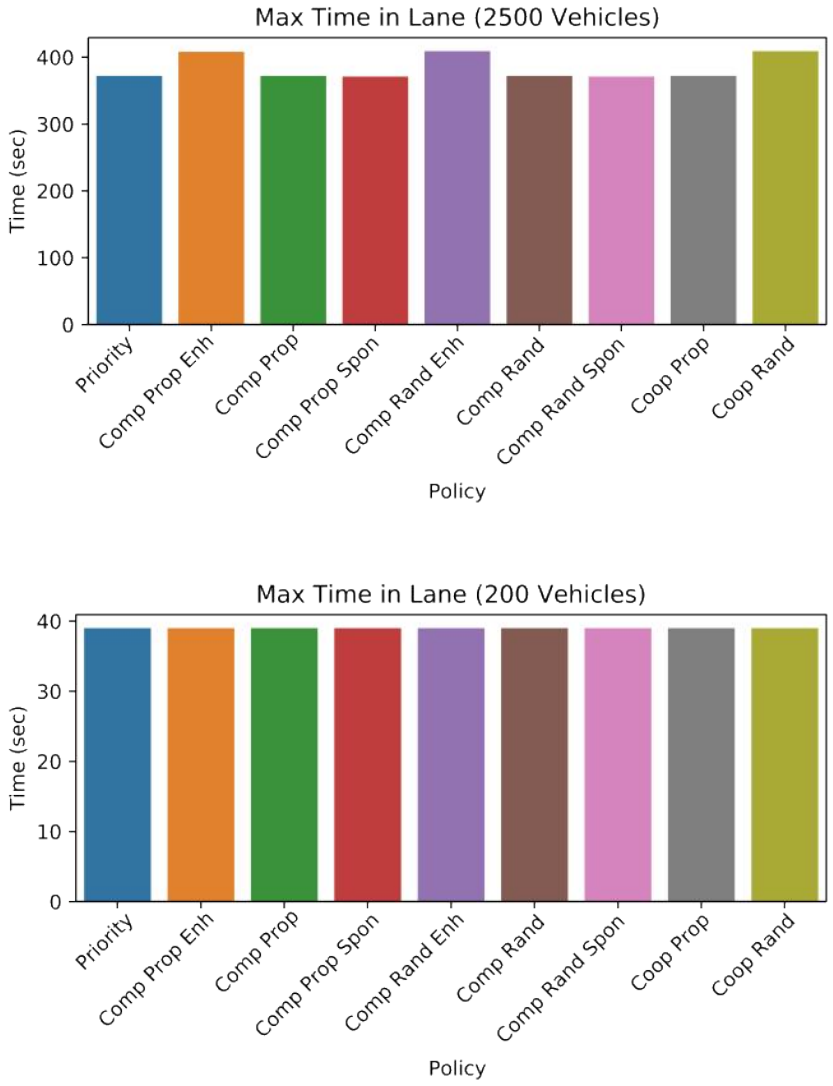


Figure 3.9: Maximum time spent waiting in the lane, with 2500 (a) and 200 (b) vehicles respectively. In both cases budget is not redistributed.

scenarios. This is likely due to the fact that when traffic is heavy, the waiting time is primarily determined by the time required to clear the lane ahead of the vehicle. As the map used in the experiments is a grid, all lanes have the same length, which would contribute to the similar average waiting times across all scenarios.

It should be noted that similar results were obtained from a separate set of experiments conducted using an independent simulator and a smaller 3×3 Manhattan-style map. Specifically, in terms of total waiting times on lanes, minor differences were observed only in the case of maximum waiting times under average traffic conditions (with one hundred vehicles).

In conclusion, the results of the experiments indicate that none of the proposed strategies result in significant improvements in terms of latency when compared to the adoption of standard yield rules. These findings suggest that simple auction mechanisms may not be an effective coordination policy for reducing vehicle latency during the transition period towards exclusively autonomous vehicles, as traffic flow and lane congestion have a greater impact on waiting times than the advantages introduced by auctions.

On the other hand, while the results of the experiments indicate that auction mechanisms do not provide significant improvements in terms of reducing latencies when compared to standard yield rules, they do offer an advantage in terms of traffic coordination and the potential to reduce the number of accidents among vehicles. The clear indication of which vehicle has the right to pass through an intersection provided by auction mechanisms eliminates misunderstandings and miscalculations that often lead to accidents among human-driven vehicles under traditional yield rules. However, it is important to note that auctions are not the only solution to achieve this goal.

About differentiating latencies

In this section, the specific case of allowing a single vehicle, such as an emergency vehicle, to definitively speed up its lane is addressed. In current urban environments, where streets are primarily populated by human-driven vehicles, emergency vehicles announce their urgency using colored flashing lights and acoustic signals. Other vehicles on the street attempt to pull over to allow the emergency vehicle to pass. It is important to understand if it is possible to exploit the auction mechanism, without the need to implement specific ad-hoc rules for such cases, in order to allow emergency vehicles to speed up their lane.

Therefore, experiments were conducted in which there was only one ve-

hicle with a potentially infinite budget, allowing it to place extremely large bids that are incomparable to any other vehicle's bids. The difference in waiting times for the vehicle when endowed with an infinite budget and when given a standard budget was measured, while always traveling at the same time of day. Various measurements were made by varying traffic conditions and the emergency vehicle's trajectory. In particular, one trajectory was convoluted and went through many intersections, while the other was a straight route from one side of the map to the opposite side.

The results indicate that the intended outcome is not achieved under any combination of strategies. The reasons for these results are subsequently analyzed.

Cooperative approach. Under the assumption of the COOP-AP strategy, there is no significant advantage in providing the emergency vehicle with an infinite budget. This outcome is not surprising as this policy is extremely fair and there is no way for a single vehicle with a high budget to speed up its own lane. While the emergency vehicle may be fortunate to have its lane cleared first at the intersection, all other vehicles must wait for all vehicles at the front of their lanes to clear the intersection before the next auction.

In more formal terms, the results indicate that under the COOP-AP approach, providing the emergency vehicle with an infinite budget does not yield a significant improvement in waiting times. This is because the COOP-AP approach prioritizes fairness, and thus, even with an infinite budget, the emergency vehicle must still wait for all other vehicles in the lane to clear the intersection before proceeding, regardless of its position in the lane. Furthermore, in the worst-case scenario where all lanes are crowded and the emergency vehicle is at the back of its lane, the waiting time is determined by the number of lanes (L), the average time required for a vehicle to clear the intersection (Δ), and the position of the emergency vehicle (p) in its lane, resulting in a minimum waiting time of $\Delta \cdot L \cdot p$ time units.

Competitive approach. Under the COMP-OWP approach, it was expected to see evidence of the benefits of providing an emergency vehicle with a high budget. However, no significant improvements were observed. When traffic is light, waiting times are generally low because many links are free and, if not free, lines are always very short. As the emergency vehicle also has to wait for vehicles ahead in the lane to pass the intersection before it, having a high budget is not decisive in achieving significantly lower

waiting times. Therefore, the experiments were focused on heavy traffic conditions, which is the real scenario in which one would like emergency vehicles to be able to move faster than others.

The results of the experiments with the COMP-OWP approach and the Rand bidding strategy did not demonstrate significant improvements in reducing waiting times for the emergency vehicle, as the randomly chosen bids were not consistently large enough to outweigh the benefits of having a potentially infinite budget. In contrast, the Prop bidding strategy ensured that the emergency vehicle's bids were consistently much larger than those of other vehicles, however, even this strategy did not result in significant improvements in reducing waiting times.

Under the COMP-OWP approach, the use of the *enhancement* mechanism did not result in significant improvements in reducing waiting times for the emergency vehicle. This is due to the fact that the mechanism only takes into account the number of vehicles in the lanes and not their budget. As such, when lanes are crowded, the emergency vehicle is treated as any other vehicle in the lanes, and the winner is determined based on the bid of the front vehicle. To ensure that the emergency vehicle's bids are taken into consideration, even when it is located towards the back of the lane, the *sponsorship* mechanism could be adopted.

When utilizing the budget recharge mechanism in combination with an infinite budget for the emergency vehicle, it is observed that the emergency vehicle experiences even longer waiting times. The reason for this is that when the emergency vehicle wins an auction, all front lane vehicles are also given an infinite budget, resulting in these vehicles also acting as emergency vehicles and interfering with subsequent auctions involving the original emergency vehicle. This effect is particularly pronounced in the experiments with a convoluted emergency vehicle trajectory, as the emergency vehicle is required to cross multiple intersections, leading to repeat interactions with the same vehicles at different intersections.

Finally, the most promising COMP-OWP approach, which includes the use of the Prop bidding strategy, no recharge mechanism, and *sponsorship*, did not demonstrate significant improvements in reducing waiting times for emergency vehicles. The main issue is that, in heavy traffic conditions, even if the emergency vehicle's lane wins the auction, it is unable to clear the intersection if its destination lane is occupied, and the emergency vehicle has no ability to influence auctions at other intersections.

To conclude, none of the proposed combinations of bidding and auction strategies effectively minimize latencies for vehicles with a potentially infinite budget under any traffic condition. The results of the experiments

reveal that even the most promising approach, which is the COMP-OWP strategy combined with the Prop bidding strategy, no recharge mechanism, and *sponsorship*, fails to achieve the desired outcome. The reason for this failure is rooted in the fact that heavy traffic can cause an emergency vehicle to become stuck even when it is given an infinite budget. This situation is not dependent on the specific auction policy adopted at the intersection, but rather on the fact that each intersection acts independently and the emergency vehicle is not able to overtake other vehicles in its line.

3.4 Conclusion

In this chapter, two key issues in smart mobility were addressed: Traffic Light Management and Intersection Management. The proposed *coordination* algorithms leveraged smart city communication capabilities and sensors to enable local decision-making, without the need for a centralized server. This approach has the advantage of avoiding a single point of failure, but it also has the limitation of not having a global view of the traffic flow.

Both algorithms were based on auction mechanisms, allowing vehicles to bid based on their individual needs, such as a vehicle in a hurry bidding more to prioritize its passage. However, due to the complexity of the traffic flow, the results of the algorithms were not always predictable. To evaluate their effectiveness, the algorithms were tested in a simulator to determine the potential benefits for vehicles and the impact of different bidding strategies.

The experiments revealed that the proposed coordination algorithms were able to reduce waiting times at intersections in most cases. However, in high traffic scenarios, the benefits were limited due to the potential negative impact on other vehicles. Additionally, it was found that a higher bid did not always result in lower waiting times or trip times, as other factors such as the overall traffic flow also played a significant role.

Overall, the results of the experiments highlighted the complexity of traffic flow and the limitations of an algorithm designed as a local coordinator, which is unable to consider the effects of its decisions on other areas or take information from other systems. The conclusion is that, in order to achieve optimal traffic management, a more holistic approach is needed, which takes into account the entire city state, and communication and cooperation among the different systems.

Chapter 4

Global Coordination: Parking and Emergency Vehicles

In contrast to the *coordination* algorithms discussed in Chapter 3, some *coordination* algorithms require a comprehensive understanding of the state of the entire city and must be centrally managed. These algorithms can involve entities from various areas of the city, thus the central server must have knowledge of the state of all sensors and the ability to communicate with all actuators and vehicles. This is necessary for the algorithm to make decisions that affect the entire city, and thus the algorithm must be designed with a global perspective.

In this chapter, two *coordination* algorithms that are designed with a global perspective are discussed. The first algorithm addresses the Management of Parking areas, while the second algorithm addresses the Management of Emergency Vehicles.

4.1 Global Coordination Background

4.1.1 Parking Related Work

The problem of limited parking availability is classified as a resource-oriented problem in [46]. The literature on smart city parking has extensively discussed various aspects such as implementation, metering, billing, and reservation (e.g. [47, 48, 49]). Surveys such as [50] and [51] have focused on park-

ing system algorithms and vehicle detection techniques. In [52], the authors propose a smart parking system that manages reservations and simulates a scenario in which all vehicles are IoT-capable. The system presented in Section 4.2.1 suggests available parking options by taking into account reserved but free slots and notifies vehicles if their destination parking space is taken before they arrive. Furthermore, it studies the impact of the co-existence of both IoT-capable vehicles and *traditional* vehicles and compares the proposed smart system with *traditional* strategies commonly used by humans to find parking.

4.1.2 Emergency Vehicles Background

The problem of limited parking availability is classified as a resource-oriented problem in [46]. The literature on smart city parking has extensively discussed various aspects such as implementation, metering, billing, and reservation (e.g. [47, 48, 49]). Surveys such as [50] and [51] have focused on parking system algorithms and vehicle detection techniques. In [52], the authors propose a smart parking system that manages reservations and simulates a scenario in which all vehicles are IoT-capable. The system presented in Section 4.2.1 suggests available parking options by taking into account reserved but free slots and notifies vehicles if their destination parking space is taken before they arrive. Furthermore, it studies the impact of the co-existence of both IoT-capable vehicles and *traditional* vehicles and compares the proposed smart system with *traditional* strategies commonly used by humans to find parking.

4.2 Global Coordination Proposals

4.2.1 Parking Managment Proposal

The proposed strategy for smart parking is referred to as the *Smart Shortest Distance* approach and is detailed in [53]. This strategy is specifically designed for equipped vehicles, and its effectiveness will be evaluated in a scenario where both equipped and non-equipped vehicles co-exist. The assumption is that equipped vehicles have the capability to communicate with IoT city infrastructure and a city server devoted to managing parking areas. On the other hand, non-equipped vehicles are not able to communicate with these systems. Additionally, it is assumed that the drivers of equipped vehicles will follow the recommendations provided by their equipment.

The system utilizes a centralized city server infrastructure that is constantly aware of the availability and occupancy of parking spots. This can be achieved through the use of IoT-enabled on-site parking monitors that communicate with the server. It is assumed that all parking spots are monitored, however, further research can examine scenarios where only some parking spots are monitored. Additionally, it is assumed that some parking spots may be reserved for vehicles that are still en route, and once reserved, these spots cannot be occupied by other vehicles. These types of parking spots will be referred to as *reservable* parking spots. The reservation of these spots can be implemented through the use of automatic road bollards or signal lights at the spots, to prevent or discourage non-equipped vehicles from parking there.

For parking spots that are not *reservable*, it is possible that other vehicles may occupy a spot that was communicated as available to an equipped vehicle by the centralized city server infrastructure. However, it can be assumed that the city's IoT-enabled systems will be able to detect this occurrence and promptly notify the equipped vehicle that the spot is no longer available.

When an equipped vehicle starts searching for a parking spot, the following steps are performed:

1. The vehicle contacts the city server to get the positions of parking areas with available spots. Let P the set off all parking, $P_a \in P$ is the subset of parking with available spots.

2. The vehicle computes its distance to each parking area $p \in P_a$ then selects the available one closest to the destination, communicates the choice to the city server, and moves towards the selected parking area. During the trip, the vehicle regularly checks for messages coming from the city server.

3. The city server updates the parking occupancy and, if the spot is *reservable*, signals that the parking spot is reserved. Meanwhile, it waits for an "end of trip" message from the vehicle. If the previously chosen parking spot gets occupied by another vehicle, then the server sends a notification to the traveling vehicle together with its current knowledge of still available parking areas.

4. If the vehicle arrives at the destination with no further notification from the city server, then it occupies the chosen parking spot and notifies the city server with an "end of trip" message. Otherwise, it will

select a new parking spot using the new information sent by the city server. Instead, if the vehicle receives a message from the server that indicates that the parking destination was stolen by another vehicle, it computes the new destination as in the first step.

In order to determine the proximity of parking spots, vehicles use the Euclidean distance metric instead of the shortest paths as in eq. (4.1) where d_p is the resulting distance for the p parking, $(g.x, g.y)$ is the destination position and $(p.x, p.y)$ is the parking position. This choice is made due to its computational efficiency, which allows for quick computation even on devices with limited processing power and memory, such as those commonly used in equipped, non-autonomous vehicles.

$$d_p = \sqrt{(g.x - p.x)^2 + (g.y - p.y)^2} \tag{4.1}$$

4.2.2 Emergency Simulation Model

The Smart City area of interest is represented as a network of interconnected nodes, formally represented as a graph $G = (V, E)$, where the vertices denote traffic intersections and the edges represent the connecting roads. The graph is directed, allowing for the simulation of permitted flow directions. Utilizing the MATSim extension, it is possible to enhance the baseline representation of the urban area by incorporating turning bans and one-way streets that reflect the actual area selected for simulation. Each link in the road network is associated with a First In First Out (FIFO) queue, where vehicles entering the link are automatically inserted. It is clear that a link affected by an accident will quickly fill its queue, resulting in congestion that will greatly reduce or even halt the speed of the vehicles involved (as discussed in Section 4.2.2).

Vehicles categories

In the simulation scenario, various types of vehicles are considered, and each vehicle is characterized by a starting location, one or more activities (in terms of destinations to be reached within specific time intervals), and a final location. More formally, each vehicle A is associated with a path $P \subset \Gamma$.

The *traditional* vehicle in the scenario being considered is one that is unable to communicate with other vehicles or the surrounding infrastructure.

These vehicles can only become aware of a traffic accident when they are in close proximity to the location of the accident. In the scenario, traditional vehicles are further classified as either *autochthonous* or *allochthonous*. *Autochthonous* vehicles are those that are familiar with the city and are able to find an optimal rerouting strategy without the need for assistance from the Smart City infrastructure. This is achieved by using the Dijkstra algorithm on the road network, with the accident link removed. In contrast, the *allochthonous* user will attempt to find alternative routes by randomly trying nearby roads. More specifically, if an allochthonous user with a specified *destination* sees an accident at a particular location $siteOfAccident$, a list of connected links from that site ($listOfConnectedLinks$) is constructed. A random outgoing link is then selected from that list and checked against the link corresponding to the accident's location. If the selected link is different from the accident link, the vehicle will move toward the selected link and check if the new arrival link belongs to its original route to the destination. If it does, the rerouting plan is complete. Otherwise, this process is repeated until a suitable route is found. This process is explained in Algorithm 1.

Algorithm 1 Allochthonous user rerouting plan.

```
1: if destination == siteOfAccident then
2:     enqueue(siteOfAccident)
3:     return
4: end if
5: while true do
6:     if siteOfAccident is the only member of listOfConnectedLinks
   then
7:         enqueue(siteOfAccident)
8:         return
9:     else
10:        vehiclePosition ← link ← pickRnd(listOfConnectedLinks)
11:        update(listOfConnectedLinks)
12:        if link ∈ originalPath then
13:            return reroutingPlan
14:        end if
15:    end if
16: end while
```

The *smart vehicle* is able to communicate with the city infrastructure and receive real-time updates about traffic conditions, including accidents. This allows the smart vehicle to proactively plan an optimal rerouting strategy

and avoid congestion caused by the accident. This approach results in a faster response time for the smart vehicle compared to traditional vehicles, which rely on visual cues or a pre-planned route. Algorithm 1 can be used to describe the rerouting process for the smart vehicle, where the vehicle receives updates about the accident location and computes the optimal route in real-time.

The emergency vehicle represents a vehicle acting on behalf of a critical emergency response organization, e.g. police, fire brigade, or ambulance. In the considered scenario, the aim is to prioritize their ability to move within the city during accidents. The emergency vehicle is able to receive real-time updates on traffic conditions, including the location of accidents, and can make informed decisions about the most efficient route to reach its destination. Additionally, the smart vehicle and the city infrastructure can make adjustments to traffic flow, such as rerouting other vehicles, to make way for the emergency vehicle and minimize its response time. An emergency vehicle entering a queue of a link is modeled as an occupant of a *Seepage queue*. Instead of lining up [54], a Seepage queue model allows the emergency vehicle to bypass congestion caused by accidents and reach its destination as quickly as possible, increasing the chances of survival for individuals involved in the emergency. Also, there is the assumption that this vehicle can communicate with other smart vehicles and with the city's infrastructure.

Even if a smart vehicle can immediately reroute as soon as a central authority is aware of an accident, V2V communication can be exploited to avoid smart vehicle routes to conflict with the emergency vehicle routes to/from the accident site. Given that the path of the emergency vehicle ($emergencyPath$) is composed of N links, it is defined a *Look Ahead Window* (LAW) of links represented by the next M links within the emergency vehicle path starting from its current position. The LAW is communicated to the nearby smart vehicles so that they can compare this newly received list of links with a subset (P^*) of their current plan (P); such a subset is composed of the first K subsequent element starting from the current position of the smart vehicle. If a potential conflict is detected then the smart vehicle will reroute (i.e. by removing the overlapping links and recomputing Dijkstra). Implementation-wise, constraints in equation 4.2 apply.

$$
\begin{aligned}
&M = |LAW|,\ N = |emergencyPath| \\
&LAW \subset emergencyPath \\
&K = |P^*|,\ P^* \subset P \\
&K \leq M \ll N, M > \epsilon \\
&if\ P^* \cap LAW\ \neq \emptyset\ then\ conflict\ is\ detected
\end{aligned}
\tag{4.2}
$$

The Look Ahead Window, as defined in equation 4.2, is constrained to have a relatively small number of links compared to the entire emergency vehicle path. This is done to minimize the unnecessary rerouting operations for the smart vehicles while still providing enough information for path conflict detection. However, it should be noted that the cardinality of the window should not be so small that it would require an unrealistic level of precision in terms of the vehicles' location and timing. In a simulated environment, this level of precision may be achievable, but it would be impractical to expect the same level of accuracy in a real-world scenario.

Vehicles speed calculation

In order to model the speed of the cars within a queue, the Greenshield Linear model (equation 4.3) is exploited.

$$
v = max\left(v_m - \tfrac{v_m\ d}{c}, min_{speed}\right) \tag{4.3}
$$

In equation 4.3, v is the current velocity, v_m is the maximum velocity for that category of vehicle, d is the current vehicle density of that link [vehicles\mile or kilometer] and c is the link capacity (as in vehicle count). According to Equation 4.3, if the link's capacity is exceeded, vehicles are required to set their velocities to zero ($min_{speed} = 0$) to reflect congestion. The maximum speed, v_m, is determined by the local traffic authority, which allows emergency vehicles to travel at a higher speed than regular road users. To simulate the tendency of emergency vehicles to "seep" through traffic, a minimum speed of $min_{speed} > 0$ is imposed for this class of vehicles. Additionally, non-emergency vehicles traveling on non-congested links decrease their speed when an emergency vehicle is passing through, simulating the likely behavior of road users to yield and move to the side of the road to allow emergency responders to pass more easily. This speed, v^*, is calculated using Equation 4.4.

$$
v^* = min\left(R\ e^{-\frac{R}{l_{link}}}, v\right) \tag{4.4}
$$

In equation 4.4, the speed v^* of non-emergency vehicles is defined as the minimum value between v as calculated in equation 4.3 and a decreasing exponential function that is dependent on a constant value R, representing a walking-pace speed, and the length of the link l_{link}. This inverse proportionality to the length of the road is intended to simulate that the effect of an emergency vehicle passing through does not necessarily propagate to the entirety of the affected street.

4.3 Global Coordination Experiments

4.3.1 Smart Parking Management Experiments and Results

The objective of these experiments is to evaluate the effectiveness of the proposed smart parking strategy, referred to as *Smart Shortest Distance*, and to compare it against other non-smart strategies in a mixed scenario where both equipped and non-equipped vehicles co-exist. It is assumed that non-equipped vehicles will utilize a non-smart strategy. The non-smart strategies simulate the approaches that human drivers use to select the next link to traverse in order to reach a parking spot close to their destination. There are three non-smart strategies that are taken into consideration:

- *Random.* The *Random* strategy simulates the behavior of drivers who have limited knowledge of the city or available parking areas, such as a tourist in an unfamiliar city. At intersections, they make a *Random* selection of the next road to follow, avoiding roads that have already been traversed, unless all alternatives have been selected once. This strategy will serve primarily as a baseline, as it is unlikely that drivers travel without some form of a navigation system, whether integrated or not.

- *Benenson [55].* The *Benenson* strategy simulates the behavior of drivers who possess some level of familiarity with the city and the location of parking areas (i.e., drivers are able to orient themselves in the area). At intersections, these drivers will choose the road that gradually brings them closer to the parking areas.

- *Shortest Distance.* The *Shortest Distance* strategy simulates drivers who have a clear understanding of the area, including their preferred parking locations and the most efficient routes to reach them. However, they are not aware of the availability of parking spaces until they arrive at the parking area. This strategy involves identifying the

nearest parking area to their destination and searching for the closest available parking spot. If the selected area is fully occupied, the driver will proceed to the next closest parking area.

Illegal Parking. In this study, the scenario of illegal parking is also addressed and simulated. Illegal parking refers to the situation in which drivers, after a prolonged period of searching for an available parking spot, decide to park in a convenient location outside of designated city parking areas. This behavior can occur in both smart and non-smart strategies. For instance, in a non-smart strategy, vehicles may continue to search for parking in areas where the number of available parking spots is inadequate for the number of vehicles looking for parking. Similarly, a smart strategy may fail to find parking due to a lack of available spots or the potential for a selected spot to be occupied by another vehicle.

Experimental set up

The experiments were conducted using the MATSim simulator, which was extended with new functionalities to incorporate both smart and non-smart parking strategies (see Section 2.2.5). The use of MATSim was deemed appropriate as it is a flexible open-source platform that enables customization to meet specific research needs.

Map In the experiments, a Manhattan-style map, shown in Figure 4.1, was utilized to test the proposed system in a highly regular scenario. The map is an artificially constructed representation and not a representation of a real city area. Each intersection comprises four links, with eight horizontal bidirectional links intersecting nine bidirectional vertical links, resulting in a total of 72 four-way intersections. The parking areas were chosen arbitrarily, with no regular pattern, in order to diversify traveling distances and simulate a real scenario in which there are areas with more parking spots than others, specifically, more parking areas on the upper and left side of the map than on the bottom and right side.

Population Each vehicle is associated with two locations, namely the driver's residence and place of work, as well as two schedules consisting of a departure time from home and a return time from work. The locations are randomly selected on the map, with the constraint that a connecting path exists in both directions. The departure and return times are generated according to normal distributions with peaks at city rush hours, specifically

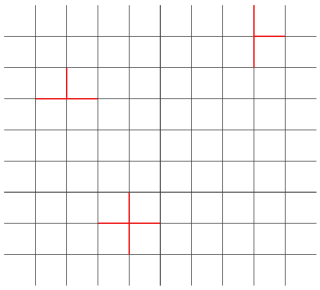


Figure 4.1: Manhattan map: parking areas are highlighted in red.

09:00 AM for departure and 06:00 PM for return. Illegal parking is simulated by setting a threshold of 26 minutes for the parking search time, which was determined empirically to ensure that the simulations always terminate.

Experiments

The aim of the experiments is to measure the duration of time required for vehicles to locate and reach an available parking spot. This duration is measured as the time elapsed between the moment when a vehicle begins its search for a parking spot and the moment it reaches an unoccupied spot. The starting point for this measurement is considered to be the instant in which a vehicle reaches its final destination and commences its search for a parking spot.

The proposed smart parking strategy is evaluated and compared to non-smart strategies in various scenarios. The impact of varying the number of *reservable* spots and the co-existence of smart and non-smart vehicles on the search time is also investigated.

Each scenario is evaluated through a series of 20 simulations, and the results are reported as averages across these runs. For each simulation, the initial conditions, such as the starting position, destination, and schedules of the vehicles, are kept constant across all strategies being compared. This allows for fair and comparable results to be obtained.

Experiment 1 In this experiment, the performance of the smart parking strategy is evaluated by varying the percentage of *reservable* parking spots, which are set to 0%, 25%, 50%, and 100%. The smart strategy is adopted by all vehicles in the simulation, and two cases are considered: (1.1) when there are more parking slots (1000) than vehicles (800); and (1.2) when there are

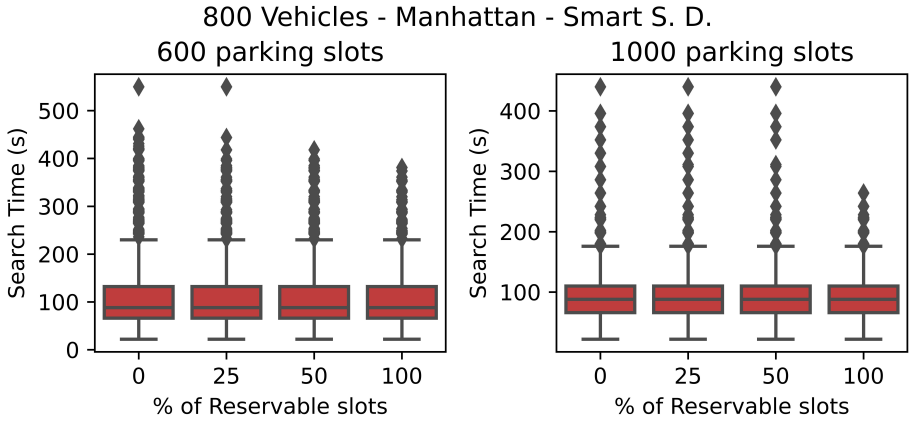


Figure 4.2: **Experiment 1.** Smart strategy search time when varying the percentage of *reservable* slots. There are 800 vehicles and on the left (resp. on the right) 600 (resp. 1000) parking spots.

fewer parking slots (600) than vehicles (800). The results of this experiment are presented in Figure 4.2.

It is evident from the results that the proposed smart parking strategy demonstrates robustness across varying traffic and infrastructure conditions.

Specifically, the average search time and standard deviations remain relatively consistent when the percentage of *reservable* parking spots is varied. Specifically, the average search time is found to be within the range of [88, 89] seconds for experiment 1.1 and [103, 106] seconds for experiment 1.2. Additionally, the standard deviation is found to be within the range of [40, 41] seconds for experiment 1.1 and [57, 66] seconds for experiment 1.2.

On the other hand, when analyzing the outliers, it can be observed that the number of *reservable* spots does have an impact on the maximum search times. In Experiment 1.1, the maximum search time was 440 seconds when the percentage of *reservable* spots was 0%, 25%, and 50%, and 264 seconds when it was 100%. Similarly, in Experiment 1.2, the maximum search time was 550 seconds when the percentage of *reservable* spots was 0%, 25%, 418 seconds when it was 50%, and 318 seconds when it was 100%.

Furthermore, it is observed that the smart strategy performs better in scenarios where there is an abundance of parking spots, with average search times being 20% less, the standard deviation being 30-38% less, and maximum search times being 20-31% less, than when there is a lack of parking

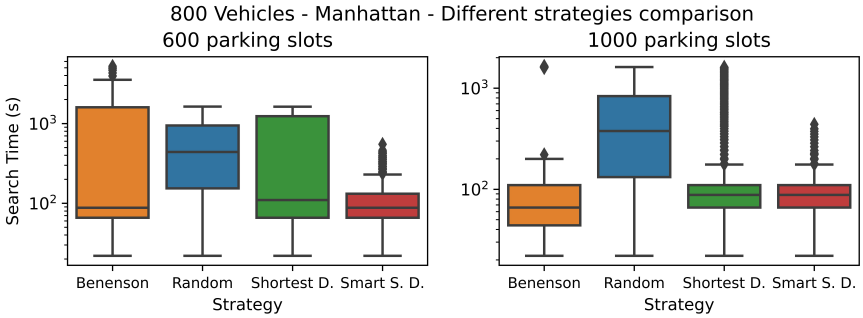


Figure 4.3: **Experiment 2.** Search time (in log scale) when varying the parking strategy. There are 800 vehicles and on the right (resp. on the left) 1000 (resp. 600) parking spots.

spots.

The results obtained from the experiments are indeed encouraging, particularly in scenarios where the number of parking slots is limited. It was observed that all vehicles were able to find a free parking spot within 9 minutes, thus avoiding the need for illegal parking. It is worth noting that the vehicles do not circulate simultaneously, and this could explain why they were able to find a parking spot even in scenarios where the number of parking slots is less than the total number of vehicles.

Experiment 2 In this experiment, the performance of the proposed smart parking strategy is compared against three non-smart strategies, under the assumption that all vehicles adopt the same strategy. Two scenarios are considered, where the number of available parking slots is 1000 (experiment 2.1) and 600 (experiment 2.2). For the smart strategy, it is assumed that there are no *reservable* parking spots, which represents the worst-case scenario. To ensure comparability, the same initial conditions are used for each strategy in each run of the experiment. The results are presented in Figure 4.3.

It is clear from the results of the experiment that the proposed smart parking strategy outperforms the non-smart strategies in both scenarios of ample and scarce parking availability. Specifically, when there is an abundance of parking spots, the average search time for the smart strategy is 89 seconds, which is significantly lower than the average search times for the *Random*, *Benenson*, and *Shortest Distance* strategies, which are 540 sec-

onds, 155 seconds, and 117 seconds, respectively. Similarly, when there is a scarcity of parking spots, the smart strategy still performs well with an average search time of 106 seconds, while the other strategies have an average search time of 598 seconds, 472 seconds, and 513 seconds, respectively.

It is apparent from the results that the *Smart Shortest Distance* strategy exhibits superior performance in comparison to other non-smart strategies. Specifically, the results indicate that the *Smart Shortest Distance* strategy leads to a reduction in searching time for 82% of the vehicles when compared to the *Random* strategy, on average achieving a reduction of 76%. Similarly, the strategy results in a reduction of searching time for 51% of the vehicles when compared to the *Benenson* strategy, on average resulting in a 57% reduction. Additionally, the strategy shows a 7% reduction in searching time for vehicles when compared to the *Shortest Distance* strategy, on average resulting in a 71% reduction in searching time.

As anticipated, when the availability of parking spots decreases, all strategies perform worse, but the proposed *Smart Shortest Distance* strategy still remains the most competitive. Specifically, the mean search time increases from 540 to 598 seconds for the *Random* strategy, from 154 to 472 seconds for the *Benenson* strategy, from 117 to 513 seconds for the non-smart *Shortest Distance* strategy, and from 89 to 106 seconds for the *Smart Shortest Distance* strategy. Similarly, the maximum search time increases from 1621 to 1632 seconds for the *Random* strategy, from 1650 to 5237 seconds for the *Benenson* strategy, from 1617 to 1630 seconds for the non-smart *Shortest Distance* strategy, and from 440 to 550 seconds for the *Smart Shortest Distance* strategy. It's worth noting that the *Random* strategy has the smallest percentage increase among the non-smart strategies when reducing the number of parking spots, while the *Benenson* and non-smart *Shortest Distance* perform much worse in this scenario.

Experiment 3 In this experiment, a scenario is simulated in which equipped and non-equipped vehicles co-exist, and the percentage of *reservable* parking spots is varied. The simulation includes 800 vehicles, with 200 vehicles adopting each strategy. The number of *reservable* parking spots varied at 0%, 25%, 50%, and 100%. The simulation is run with 1000 and 600 parking slots. Non-equipped vehicles are only permitted to park in *reservable* spaces if they have not been reserved. Results of this experiment are presented in Figure 4.4.

It is observed that the smart parking strategy remains competitive even when in the presence of non-equipped vehicles, particularly when there is a

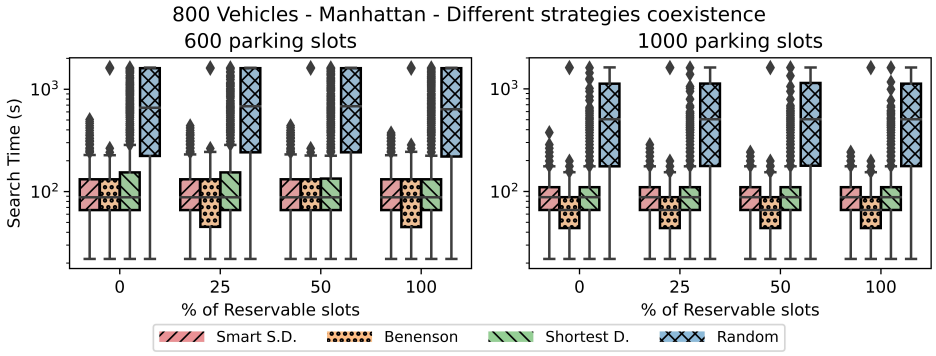


Figure 4.4: **Experiment 3:** Parking search time (in log scale) when smart and non-smart strategies co-exist and vary the percentage of *reservable* parking spots. There are 800 vehicles and on the right (resp. on the left) 1000 (resp. 600) parking spots.

lack of available parking spots. Notably, there is little variation in performance when varying the percentage or number of *reservable* parking spots, and the search times are comparable to those observed in the previous experiment where all vehicles adopted the smart strategy.

Observe also that the smart strategy allows all vehicles to find a parking spot in a timely manner, well before the point at which they would resort to illegal parking. Furthermore, it is observed that all other strategies exhibit a significant number of outliers, with very large searching times, particularly in the case of the *Random* strategy. Additionally, non-smart strategies tend to result in more instances of illegal parking. Even though the *Benenson* strategy may perform better in certain situations, it still exhibits a notable number of outliers, which are not present in the *Smart Shortest Distance* strategy.

4.3.2 Emergency Experiments and Results

The initial representation of the MASA area street network was created by importing data from the OpenStreetMap (OSM) database. This process involved the use of several plugins to augment the network with additional information such as capacities for the road links, turning bans, and parking spaces. The MASA area, which is a 1-square kilometer wide area centered around coordinates (44.65632, 10.93150) in OSM[1], was chosen as the simula-

[1]https://www.openstreetmap.org/

tion area. The simulated population size was estimated using regional traffic flow data made available online[2]. The data indicated that the rush-hour road users population in the area ranges between 10,000 and 20,000 users. Constants in equations 4.3 and 4.4 were set to $v_m = 50\ Km/h$, $min_{speed} = 0$ for non-emergency vehicles, $v_m = 70\ Km/h$, and $min_{speed} = 10\ Km/h$ for emergency vehicles, $R = 5\ Km/h$. These values are in compliance with Italian law. The Look Ahead Window (LAW) cardinality for smart vehicle rerouting was set to 15, and $K = 8$ from equation 4.2.

The vehicles in the experimental scenarios are provided with initial plans, which include their departure and arrival locations. These plans are set for all the users in the different experimental scenarios so that the users' plans remain consistent while different behaviors are being simulated. The plans are constructed by selecting two random locations on the MASA map as the home and work locations for the vehicles. The vehicles perform two trips per day, starting from home and ending at work in the morning, and then reversing the route in the evening. The departure times are chosen based on a normal distribution, with the peak during typical rush hours (8:00 AM-9:00 AM for home departure and 06:00 PM-07:00 PM for departing from work).

The goal of the traffic modeling in this simulation is to investigate the mechanisms behind emergency response while varying the capabilities of the involved vehicles within a specific city area. For this purpose, a representation of the MASA area street network was created, utilizing information from the OpenStreetMap database and augmented with additional information such as capacities for the road links, turning bans, and parking spaces. The simulated population was estimated using regional traffic flow data, and the vehicles' routes were calibrated to depict realistic traffic situations [56, 57]. However, it should be noted that the objective of these simulations is not to provide an accurate reconstruction of the traffic in the MASA area, but rather to study the proposed emergency response strategies.

Emergency vehicles response time

In this set of experiments, the primary focus is on analyzing the response time of emergency vehicles, specifically ambulances, when traveling to and from the site of an accident within the MASA area. The simulation model outlined in section 4.2.2 highlights that the travel time of emergency vehicles can be influenced by the ratio of smart vehicles to the total number of road

[2]https://servizissiir.regione.emilia-romagna.it/FlussiMTS/

	Smart Vehicle percentage	**Ambulance travel Time [s]**	**% difference from Scenario 1**
Scenario 1	0	109	-
Scenario 2	33	102	6.4
Scenario 3	50	89	18.3
Scenario 4	66	80	26.6
Scenario 5	100	69	36.7

Table 4.1: Ambulance average travel times with different kind of vehicles.

users in the simulation. This ratio serves as a parameter in the experiments and can provide insight into the impact of smart vehicles on urban mobility during the transitional period from *traditional* vehicles to fully equipped vehicles. It is important to note that during this transition, both equipped and non-equipped vehicles will coexist, and thus, understanding the effects of varying this ratio is crucial for evaluating the feasibility and potential benefits of smart vehicle deployment in urban environments.

To determine the ambulance travel time with an increasing number of smart vehicles, multiple scenarios were run. In all these scenarios, it is reported the average travel time over a set of 30 ambulance trips. The first scenario consisted of 12000 users with a scaling number of smart vehicles and results are depicted in table 4.1.

In summary, the results presented in table 4.1 indicate that the average response time of emergency vehicles improves as the percentage of smart vehicles increases. The improvement is particularly significant when the percentage of smart vehicles exceeds 50%, with a decrease in response time of up to 36.7%. Additionally, the results also suggest that the population size of road users within the simulated area has an impact on the response time improvement, with larger populations resulting in a smaller improvement. Additional tests are performed increasing the number of vehicles to 16000, hence getting closer to the inevitable congestion of the MASA. In such a scenario, the response time improvement within a population characterized by 75% of smart vehicles would range from 4.3% to 8.33% with respect to the baseline scenario of no smart vehicles at all. Overall, the results suggest that an increase in the percentage of smart vehicles in a population can significantly improve the response time of emergency vehicles in the event of an accident.

Accident Scenario	Classic	Allochthonous	Autochthonous	Smart
Prev. intersection has multiple outgoing links	1	0.7385	0.6906	0.6724
Prev. intersection has no other outgoing links	1	1	1	0.6392

Table 4.2: Normalized travel times of different kind of vehicles.

Accidents' impact on non-emergency vehicles

In this section, the impact of accidents and related road closures on travel times for vehicles within the MASA region is analyzed. As previously discussed in Section 4.2.2, regular road users are classified into different vehicle categories, including *traditional*, *allochthonous*, *autochthonous*, and *smart* vehicles. The *traditional* vehicles are modeled as the MATSim baseline implementation of road users who will not attempt to circumvent the roadblock caused by an accident. These vehicles serve as a baseline for the proposed simulation. In this experiment, accidents are artificially introduced at random locations within the MASA area and their effect on the travel plans of vehicles is recorded. The average travel times for each vehicle category are then calculated. It is assumed that the entire population of users belonging to the indicated categories is included in the experiments. Another crucial aspect to consider in such an experiment is the number of outgoing links from the link preceding the blocked road. If there is only one outgoing link, it is likely that most road users will be stranded on the blocked road, whereas having multiple options for rerouting can help drivers avoid being blocked. The results of the experiment can be observed in Table 4.2.

Table 4.2 illustrates that when it is possible to reroute at the intersection preceding the accident link, travel times can be improved by up to 33.6% for *smart* vehicles and 26.1% for *allochthonous* vehicles. These results indicate that a more realistic approach to modeling accidents leads to less pessimistic traffic estimates compared to the MATSim baseline approach, particularly for allochthonous vehicles. More specifically, *traditional* vehicles will attempt to find alternative routes without the aid of smart city infrastructure, even if it means risking getting lost. While the performance difference between *allochthonous* and *autochthonous* vehicles can be explained by the fact that the latter category of vehicles cannot get lost, in a scenario where there are multiple outgoing links, there is relatively little difference in travel time improvements between *autochthonous* and *smart* vehicles (about 2The benefits of *smart* vehicles become more pronounced in the second scenario, where

the intersection preceding the accident site does not allow for last-minute rerouting. Specifically, *allochthonous* and *autochthonous* vehicles can only react to an accident after they see it, resulting in performance that is similar to *traditional* vehicles modeled in the MATSim simulations. In contrast, *smart* vehicles, as modeled in the simulations, are immediately alerted to the accident by the city's infrastructure, leading to significant improvements in performance.

4.4 Conclusion

In this chapter, the importance of a central server in a smart city is emphasized through two examples: Parking Management and Emergency Vehicle Management. The proposed *coordination* algorithms for these two problems are designed to exploit information about the entire city and are centrally managed as a central entity must have knowledge of the entire city state in order to effectively manage these issues and provide the necessary information to vehicles.

The Parking Management algorithm is able to suggest parking areas with available spots for vehicles and notify them if a chosen spot will be occupied by another vehicle. Additionally, if the parking spot has the capability to be reserved, the vehicle is able to reserve it.

The Emergency Vehicle Management algorithm is also centrally managed, with a central server requesting non-emergency vehicles to change their routes if they will pass through the emergency vehicle's route, in order to prevent congestion. The server also informs connected vehicles of the location of any accidents that occur in the city, allowing them to adjust their routes accordingly.

The results of these algorithms demonstrate that they are effective in reducing parking search time and emergency vehicle response time, even in situations of high traffic flow or limited available parking spots. Additionally, non-emergency vehicles also benefit from information about accident locations. Specifically, the time spent searching for a parking spot is reduced by up to 76% when compared to baseline strategies utilized by human drivers and the proposed Emergency Vehicle Management System is able to reduce both the Emergency Vehicle response time by up to 36.7% and non-emergency Vehicle trip time by up to 36.1% with respect to the scenario in which no such system exists.

It is important to note that the central design of these algorithms is crucial for their effectiveness, as they rely on information about the entire

city state. A distributed design would not be able to achieve the same benefits. While a central design does present the potential for a single point of failure, the benefits outweigh this potential drawback.

Chapter 5

Hardware Accelerated Planning and Localization

5.1 Introduction

In recent years, there has been a marked increase in interest and research in the field of automotive technology, specifically in the areas of Advanced Driver Assistance Systems (ADASs) and autonomous vehicles. Many research and development efforts have been focusing on these areas.

ADAS are vehicles that require human supervision but can perform certain maneuvers such as steering or acceleration autonomously. The level of autonomy of the system determines the type of maneuvers that can be performed and the role of the human driver. The Society of Automotive Engineers (SAE) defines different levels of autonomy:

- **Level 0.** The vehicle is not able to perform any maneuvers in itself. The driver has to perform all maneuvers. *Traditional*, human-driven, vehicles are examples of this level.

- **Level 1.** The vehicle can perform a single maneuver autonomously; the driver must perform other maneuvers. An example of this level is Cruise Control, which is a system that controls only vehicle acceleration.

- **Level 2.** The vehicle can perform autonomously more maneuvers. For example, the vehicle can both steer and accelerate.

Autonomous vehicles are the evolution of ADAS, they do not need human intervention and can reach a destination independently. In the SAE categorization, autonomous vehicles occupy the last three levels:

- **Level 3.** The vehicle can drive itself in some situations. The human driver has to supervise the system and take control of the vehicle if the system fails or if the system itself alerts that is not able to manage the situation.
- **Level 4.** The vehicle can drive itself in a definite situation (such as a specific area) without human intervention.
- **Level 5.** The vehicle can drive itself without human intervention, in all situations.

The main distinction between Level 2 and Level 3 vehicles is that Level 2 systems primarily provide assistance to the human driver, who is still considered the primary operator of the vehicle. On the other hand, Level 3 systems are designed to be capable of performing most driving tasks, but with the expectation that a human driver will be available to take over in certain situations or scenarios.

Both ADAS and autonomous vehicles rely on the ability to perceive the environment in order to control their maneuvers. The *perception* system is responsible for determining the vehicle's location and detecting obstacles. The *control* system is responsible for *planning* and executing the maneuvers.

The input required by *perception* and *control* algorithms can be obtained in two ways: through the use of sensors or by leveraging smart city infrastructure. The former method, which is the most commonly used, has several advantages. It does not rely on external infrastructure and the sensors can be selected and installed directly on the vehicle by the manufacturer. However, the raw information obtained must be pre-processed, and it can contain noise. By utilizing smart city infrastructure, vehicles can access more reliable data, as other vehicles can communicate their exact position and intended trajectory. The smart city authority can also provide precise information about the environment, such as the location of static obstacles or the roadway geometry. However, this approach also has its drawbacks, such as the potential for data to arrive too late to be processed and not all vehicles being equipped to share their position and intentions.

More specifically, *perception* algorithms require data to reconstruct the environment, which includes the vehicle's position, the roadway geometry,

and the location of obstacles. The rest of the pipeline, including the *control* algorithm, then uses this reconstructed environment. It uses the reconstructed environment in conjunction with other vehicles' trajectories and positions to avoid collisions. This data can be obtained from the smart city infrastructure or by using sensors to reconstruct the environment. The smart city authority can provide roadway information, obstacle positions, and vehicle *localization* within the city. Other vehicles can also share their position and intentions. Sensors such as GPS receivers or odometry systems can be used for *localization*, and cameras and Lidar can be used to perceive and reconstruct the environment to identify other vehicles and obstacles' positions. However, it is challenging to infer the intentions of different vehicles from sensors, as there is a degree of unpredictability in the maneuvers of other vehicles that can only be estimated probabilistically. Some recent studies have used deep learning techniques for this task [58, 59, 60], while different approaches have assumed that vehicles only follow physical rules [61] or take into account the driver's possible intentions [62, 63].

As previously stated, the most common method for retrieving information is through the use of sensors. As a result, most algorithms are based on sensor data. This thesis focuses on the *localization* and *planning* phases, for which there are a variety of algorithms that use sensors. For example, GPS [64] or odometry sensors [65] can be used to determine the vehicle's position. Other approaches, such as Simultaneous Localization and Mapping (SLAM) [66], use sensors such as Lidar or cameras to construct a map of the environment, with the latter approach being referred to as Visual SLAM [67]. Cameras can also be used to retrieve other data required for the *planning* phase, such as reconstructing roadway geometry [68] and detecting the positions of other objects [69].

The pipeline from *perception* to actuation must be completed in a timely manner to avoid safety hazards. As the surrounding environment is constantly changing, if the pipeline takes too long to complete, the selected maneuver may no longer be appropriate. For example, if the pipeline takes too long to complete, and an obstacle enters the scene, the new trajectory that avoids the obstacle may be ready too late, resulting in a collision as the vehicle is still following the original trajectory. It is crucial that the pipeline is optimized to complete quickly in order to maintain safety.

For this reason, it is essential that the algorithms used in the pipeline are optimized for speed. This can be achieved by improving the algorithms themselves or by using hardware accelerators (such as a CUDA-enabled GPU) as outlined in Section 5.2. This will help ensure that the pipeline completes quickly and does not pose a safety hazard.

The following chapters of this thesis will address the *localization* and *planning* problems. It will elaborate on the specific algorithms and techniques that can be used to achieve fast and accurate *perception* and *control*, and the research on how to optimize them.

In particular, Chapter 6 is dedicated to the *localization* problem. The chapter will provide an in-depth examination of the problem and will describe an algorithm for *localization* in Section 6.2. Additionally, in Section 6.3, a novel implementation of this algorithm will be proposed to improve its performance and efficiency.

In contrast, Chapter 7 will focus on the *planning* problem. It will describe an algorithm for addressing this problem in Section 7.2. And, similar to *localization*, a novel implementation of the algorithm will be proposed in Section 7.3 to enhance its performance and efficiency.

5.2 Hardware Accelerators

In recent years, the use of hardware accelerators has become increasingly popular in the field of autonomous driving systems. Hardware accelerators are specialized computer hardware that can perform certain operations much faster than general-purpose processors. These accelerators are used to speed up computationally intensive tasks, such as *perception* and *control* algorithms, which are critical to the operation of autonomous vehicles.

There are several types of hardware accelerators that have been used in autonomous driving systems. Graphics Processing Units (GPUs) are widely used to accelerate the processing of sensor data, such as images and point clouds. Field-Programmable Gate Arrays (FPGAs) are also used to accelerate the processing of sensor data and to perform tasks such as object detection and tracking. Digital Signal Processors (DSPs) are used to perform tasks such as sensor fusion, and Neuromorphic processors are used to accelerating deep learning-based algorithms.

The use of hardware accelerators can significantly improve the performance and efficiency of autonomous vehicles, by reducing the computational time of the *perception* and *control* pipeline. This allows the system to respond more quickly to changes in the environment and make decisions in real time, which is essential for maintaining the safety of the vehicle and its passengers.

In particular, Nvidia embedded boards, due to their compact dimensions, lightweight design, and low power consumption, are well-suited for use in vehicles. These boards are equipped with various accelerators, such

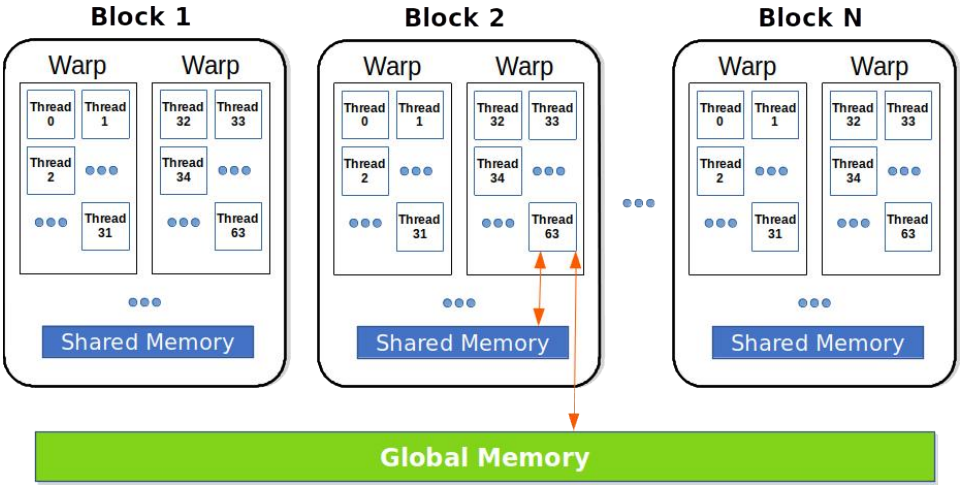


Figure 5.1: Illustration of a CUDA kernel launch configuration, where threads within the same block access *shared memory* more efficiently compared to global memory, and threads are grouped in *warps* that execute the same instruction.

as the GPU, Deep Learning Accelerator (DLA), and Programmable Vision Accelerator (PVA), which is specifically designed to accelerate visual algorithms.

The Nvidia boards are used as reference hardware for the optimization of the proposed algorithms. In particular, the Nvidia Xavier AGX board, which has all the aforementioned accelerators, is used. The GPU serves as the accelerator to enhance the performance of the algorithms.

In order to utilize the GPU accelerator for general-purpose computation and accelerate the algorithms, the CUDA programming model is employed. The usage and implementation of CUDA are thoroughly discussed in Section 5.2.1.

5.2.1 GPU and CUDA

The GPU (Graphics Processing Unit) was originally designed to accelerate graphic workloads, but it can now also be used for general-purpose computing (GPGPU). A GPU is a SIMD (Single Instruction Multiple Data) processor that is capable of processing large amounts of data in a highly parallel manner. To execute a compute task on the GPU, it is necessary to allocate memory in GPU-visible memory spaces, copy the input data from

the host (CPU) to the GPU device memory, execute a *kernel*, which is a program specifically coded for the GPU instruction set architecture, and then copy the results back to the host memory.

Kernels are dispatched through a launch configuration, which is a grid specified by the programmer in which parallel GPU threads are logically organized [70]. The launch configuration describes the degree of parallelism in which the work must be computed over a grid of parallel threads. Depending on the type of problem and the size of the input data, a kernel may be invoked with a launch configuration that spans the X, Y, and Z dimensions. A grid is composed of blocks of threads, and threads within a single block are grouped into *warps*, which are sets of 32 threads that can execute the same instruction on different data items in a lock-step fashion (see Figure 5.1). Divergence, or conditional instructions within a warp, causes serialization of all the conditional branches.

Moreover, threads in the same block can access can synchronize themself using the *synchthread* primitive, and share data using a special memory called *shared memory*.

CUDA *shared memory* is a special type of memory that is shared among threads within the same thread block as illustrated in Figure 5.1. It allows threads to read and write to the same memory location, allowing them to coordinate and communicate with each other. *Shared memory* is faster than global memory (which is the memory in which data are copied from the CPU), as it has lower access latency, and can be accessed by all threads in a block simultaneously.

The *syncthreads* primitive is a CUDA primitive function that synchronizes all threads within a thread block. When a thread reaches a *syncthreads* instruction, it will wait until all other threads in the same block have reached the same instruction. This allows for coordination and communication among threads, as they can be sure that all threads have completed a certain operation before proceeding. It can also be used to implement barrier synchronization, where all threads must wait at a barrier before proceeding.

Shared memory and *syncthreads* are powerful features that allow for efficient data sharing and coordination among threads. They enable the optimization of parallelization and computation, particularly for applications that require large amounts of data to be shared among threads, such as image processing, machine learning, and scientific simulations.

To simplify the process of utilizing the GPU capabilities, NVIDIA developed a proprietary standard called CUDA (Compute Unified Device Archi-

tecture). CUDA provides APIs and libraries that can be accessed through high-level programming languages such as C++. In other words, CUDA abstracts much of the hardware complexity that is typical of GPU applications, making it easier for programmers to work with the GPU.

In addition, the CUDA programming model allows the programmer to express an additional layer of parallelism through CUDA *Streams*. A CUDA *Stream* is a sequence of commands (compute kernel invocations and memory movements) that must be executed in the order in which they are enqueued. As a result, a single program that manages multiple *Streams* is able to submit copy and compute commands concurrently, so that their execution overlaps in time. For instance, while a kernel is performing calculations for a specific set of data, memory transfers can be overlapped with such kernels in a buffered flow of execution. To synchronize the execution of commands among different *Streams*, a mechanism called CUDA *Events* can be used.

The proposed implementation of the *localization* algorithm (Section 6.3) heavily relies on CUDA *Streams* and *Events*. This approach was taken to optimize performance by overlapping copies and kernel execution, as well as processing multiple kernels launched in different *Streams*. By doing so, according to the hardware features of the GPU, and depending on the requirements (in terms of computing and memory resources) of each kernel, it allows for parallel execution of the kernels.

The majority of the kernels used in the image processing phase of the proposed implementations are invoked through a three-dimensional grid, where the X and Y dimensions are associated with the width and height of the input images. The Z dimension is used to access the differently scaled versions of the input images. This will be discussed in more detail later in section 6.3.1.

In regards to the proposed implementation of the *planning* algorithm (Section 7.3), *Streams* are utilized to overlap memory copies and kernel execution. Additionally, the grid dimensions are associated with different parameters used to compute the trajectory points. The grid is also utilized to group threads that need to synchronize themselves and use the *shared memory* in the same block. The synchronization and *shared memory* are used to compute the cost component in each thread and then combine the values into a single result.

Chapter 6

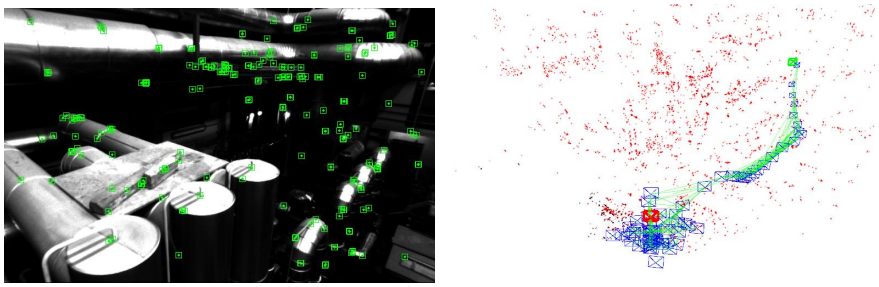
Localization - ORB-SLAM

6.1 Localization

One of the primary tasks for autonomous vehicles is navigation, and determining the vehicle's location is a crucial aspect of this task. The term "autonomous vehicle" refers to all types of vehicles that are capable of performing tasks involving locomotion with minimal human intervention. Examples of autonomous vehicles include Unmanned Aerial Vehicles (UAVs), which are used in various fields such as structural inspection [71, 72], environmental monitoring [73, 74], and surveillance [75], autonomous cars [76] used for emergency response [77] and everyday city transportation [5], as well as autonomous underwater vehicles (AUVs) [78] and autonomous surface vehicles (ASVs) [79].
Autonomous vehicles must use sensors and algorithms to perceive the external environment to localize themselves in the world.

There are various types of sensors that can be used for *localization* in autonomous vehicles, such as GPS receivers and Inertial Measurement Units (IMUs) which are specifically designed for *localization* purposes. Other sensors, such as Lidar and cameras, are primarily used for the *perception* of the environment, but the data they collect can also be utilized by certain algorithms to estimate the movement of the vehicle and determine its location. These types of sensors are used to extract features from the environment and use them as a reference for the *localization* problem.

The selection of sensors for autonomous vehicles is a complex process that depends on multiple factors. The cost of the sensor is a significant consideration, as it can have a significant impact on the overall cost of the robot. The precision of the sensor is another important factor to consider, as



(a) Points extracted to map and recognize the environment

(b) Reconstructed trajectory based on *localization* of the vehicle

Figure 6.1: An example of ORB-SLAM system in UAVs.

different sensors have varying levels of precision, and even the same sensor type can have different levels of precision based on construction quality. It is also usually correlated with the sensor price, as more precise sensors tend to be more expensive. Additionally, the sensor selection must be done in light of the algorithms that will be used to achieve the desired results. The algorithm's computational and memory requirements must also be taken into account, as not all algorithms can be run on all types of robots, which typically have an embedded board to run the system. If the algorithm required by a sensor is not compatible with the robot, it would be more efficient to consider another sensor.

6.1.1 Simultation Localization And Mapping (SLAM)

The Simultaneous Localization and Mapping (SLAM) algorithm is a *localization* technique that combines sensor data with mapping capabilities to determine the vehicle's location and construct a map of the environment. This algorithm can be used to explore unknown environments and determine the vehicle's location when it revisits a previously mapped area (see Figure 6.1). SLAM is applicable to a wide range of scenarios such as automated parking [80], swarm systems [81], self-driving cars [82], and in rescue or other extreme situations [83, 84].

SLAM algorithms can integrate a variety of sensors such as Lidar, IMU, GPS, and cameras. Among these sensors, cameras have been extensively studied in the context of SLAM, due to their low cost, compact size, and ease of setup [85]. The visual information provided by cameras is particularly useful for SLAM, as it can be used to extract features from the environment,

which can be used to determine the vehicle's location. Camera-based SLAM systems are particularly useful in applications where cost, size, and ease of installation are important considerations.

The *Visual SLAM* (V-SLAM) systems are camera-based SLAM systems that utilize point features across consecutive images to recognize the environment, making use of camera sensors as input. Additionally, by utilizing stereo cameras, the system can extract depth information by processing two images simultaneously, the left and the right image, providing the system with additional information for the *localization* and mapping process. This approach allows for a more robust and accurate *localization* and mapping.

Recently, there has been a significant amount of research on the SLAM problem, with a particular focus on *Visual SLAM* (V-SLAM) systems. The main objectives of a SLAM system are to estimate the camera pose and construct a map of the environment. Different V-SLAM approaches tackle these objectives in various ways, making use of different types of information extracted from the input images. Some systems can handle both monocular and stereo camera configurations, while others are specifically designed for monocular cameras. The choice of sensor type and the algorithm used can have a significant impact on the performance and the accuracy of the SLAM system, hence it is an important factor to consider when designing a V-SLAM system. One of the main approaches used in V-SLAM systems is *Direct* methods. These methods use the camera input image pixels directly to estimate the motion of the camera and the structure of the environment by minimizing a photometric error between two consecutive images. This type of approach is based on the idea that by comparing the intensity of the pixels in two images, it is possible to determine the relative motion of the camera and the environment. The Direct method is considered to be very efficient and can be used in real-time applications, but it is sensitive to lighting variations and can be affected by changes in the texture and the geometry of the environment. One example of a Direct method approach is the LSD-SLAM algorithm [86, 87]. This algorithm is capable of building large-scale semi-dense maps using high gradient pixels and can work with both monocular and stereo cameras. Another example is the DSM algorithm [88] that focuses on the idea of map reusing and highlights the importance of mid-term data association. However, it is only designed to be used with monocular cameras. An alternative approach to Direct methods in V-SLAM systems is *feature-based* methods. These methods extract specific features from the images and use them to compute the estimated camera pose. One of the first examples of this type of approach is the MonoSLAM

algorithm [89, 90]. This algorithm tracks points in consecutive images by performing a guided search based on correlation and, as the name suggests, it is designed to work exclusively with monocular cameras. This type of approach is considered to be more robust to lighting variations and changes in the environment, but it can be affected by the presence of repetitive patterns or textures. Another type of Visual SLAM is the *keyframe-based* approach, which estimates the map using only a select number of frames, commonly referred to as keyframes, that are considered to be more informative. This approach leads to more accurate results at the same computational cost [91] compared to the previously mentioned approaches. One example of this methodology is the PTAM algorithm [92], which separates the tracking and mapping parts into different threads. Like MonoSLAM, PTAM is designed to work exclusively with monocular cameras. Keyframe-based approaches are known for their high accuracy, as it only uses salient frames.

Starting from the ideas above, the ORB-SLAM [93] system was built. ORB-SLAM is a feature-based visual simultaneous localization and mapping (SLAM) algorithm that uses ORB descriptors as features extracted from images, so it is a *feature-based* SLAM. The algorithm is designed to construct the map using only selected frames, making it a *keyframe-based* method. The first version of ORB-SLAM was only able to manage monocular cameras.

Subsequently, the ORB-SLAM2 [94] system was proposed, which introduces several improvements and new features to the original ORB-SLAM algorithm. The system features a more robust and efficient loop closing mechanism, and it also supports stereo and RGB-D cameras as input sensors. Additionally, the ORB-SLAM3 [95] system was developed, which integrates Inertial Measurement Unit (IMU) data and allows for the use of fish-eye cameras as input sensors. These advancements in the ORB-SLAM algorithm further enhance its ability to accurately localize and map environments, making it a valuable tool for various applications such as autonomous navigation and robotics. The detailed workings of the ORB-SLAM algorithm will be further explored in Section 6.2 of the thesis.

A novel approach is proposed that builds upon the ORB-SLAM family of algorithms. Specifically, Section 6.3 describes a method that utilizes GPU acceleration and introduces novel methods for filtering the extracted features and constructing image *Pyramid*, which differs from the methods used in previous works. From the implementation point of view, the GPU is exploited using an optimal task execution concurrency through CUDA streams (Section 5.2.1) and a parallel GPU implementation of the more computationally intensive task od ORB-SLAM, that, with a preliminary ex-

periment, results to be in the *Tracking* phase (see Section 6.2).

To evaluate the performance of ORB-SLAM and other SLAM and *Visual Odometry* systems, several commonly-used datasets are employed. One such dataset is the KITTI [96] dataset, which is a road dataset that is used for various purposes including *Visual Odometry* and SLAM systems. Another dataset is the NewCollege [97] dataset, which is a stereo camera dataset that also includes laser-scan data of the environment. The Euroc [98] dataset is designed for use with UAVs and drones and is useful for evaluating *Visual Odometry* and SLAM performance. Lastly, the TUM [99] dataset consists of RGB-D images and ground truth data for evaluating SLAM systems.

The proposed novel implementation is evaluated against the state-of-the-art CPU and GPU implementations, as well as on both versions of ORB-SLAM 2 and 3. The experiments demonstrate that it achieves superior performance in terms of execution latencies and trajectory errors, with a latency speed-up of up to 3x compared to the original implementation. Given its improved performance over the initial CPU version, the proposed solution establishes itself as a new benchmark for further research in this field.

The rest of this chapter is organized as follows: in Section 6.1.2 are presented works that use accelerators to implement ORB-SLAM. In Section 6.2 the ORB-SLAM algorithm is presented, then in Section 6.3 the novel proposed implementation is described and in Section 6.4 is evaluated. Finally, in Section 6.5.

6.1.2 Use of Accelerators in ORB-SLAM

The timely retrieval of the autonomous vehicle's position is a critical aspect of the *localization* task, as it is closely related to the safety of the vehicle. The *localization* information must be obtained within a reasonable time frame to avoid becoming obsolete, as it is used in subsequent driving tasks such as planning and actuation. An outdated position may result in incorrect planning and pose a safety risk, such as a possible crash. To meet these real-time requirements, various methods can be utilized such as utilizing powerful multi-core CPUs, implementing compute accelerators, and fine-tuning the algorithm's parameters and implementation details. These performance-related aspects play a crucial role in ORB-SLAM based approaches, which is why several versions of ORB-SLAM that exploit accelerators have been proposed in the literature, such as in references [100] and [101].

The computational time required for ORB-SLAM is a significant con-

cern, particularly in the context of autonomous vehicles, where real-time performance is essential for safety. To address this, various efforts have been made to optimize the performance of ORB-SLAM by leveraging computational accelerators. In [102], a comparison of different acceleration methods for ORB-SLAM and other Visual Odometry techniques is presented, using three CUDA-enabled devices: the Nvidia Jetson TX2, Xavier NX, and Xavier AGX. However, it is worth noting that the process of adapting SLAM-based algorithms for use on GPU is not a straightforward task. In [103], the authors proposed an acceleration method for ORB-SLAM2 using OpenCL for FPGA (Altera DE1-Soc board) and CUDA for NVIDIA (Jetson TX1) platforms, with a focus on accelerating the *Map Initialization* phase of the algorithm. Both OpenCL and CUDA are programming models and languages that exploit the parallel computing capabilities of accelerators. OpenCL is an open standard, while CUDA is a proprietary technology developed by NVIDIA.

Similarly, in [101], the authors used CUDA to accelerate ORB-SLAM2 on NVIDIA platforms by utilizing CUDA and OpenCV GPU versions to extract and match features in Stereo camera systems. In [104], the authors employed CUDA to accelerate the initial stages of feature extraction in the ORB-SLAM algorithm and also utilized OpenVX to model the computational graph and offload operations to the GPU accelerator. Additionally, in [100], the authors used OpenCV to perform image scaling on the GPU, which is an important phase of the ORB-SLAM approach, and only implemented GPU acceleration on select parts of the feature extraction phase.

In summary, various works have proposed methods to accelerate ORB-SLAM by utilizing various acceleration technologies such as CUDA, OpenCL, and OpenVX on different platforms like FPGA, NVIDIA Jetson, and NVIDIA GPU. All of these works aim to improve the performance of ORB-SLAM by reducing its computational time.

The proposed novel approach builds upon the ORB-SLAM family of algorithms and utilizes GPU acceleration to improve performance. It introduces novel methods for filtering the extracted features and constructing image *Pyramids* and is integrated into both ORB-SLAM2 and ORB-SLAM3. It is designed to minimize data transfers between the CPU and GPU and to optimally exploit CUDA features such as streams and events, which allows the algorithm to minimize CPU-GPU interactions and fully exploit the parallel compute potential of integrated GPU devices. This is in contrast to previous works that have focused on ORB-SLAM2 and have used other acceleration methods such as OpenCL and OpenVX without explicitly managing CUDA streams and events.

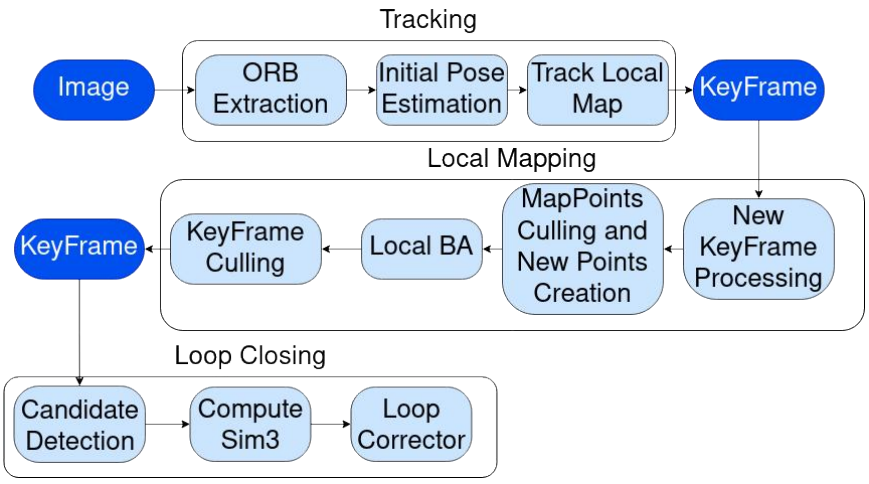


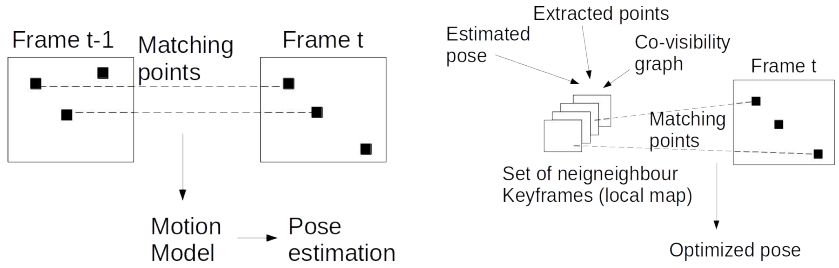
Figure 6.2: ORB-SLAM flow diagram.

6.2 ORB-SLAM

As seen in Figure 6.2, the ORB-SLAM method consists of three main phases: *Tracking*, *Local Mapping*, and *Loop Closing*. In the original ORB-SLAM implementation, these three consistent parts of the algorithm are executed in parallel by three different threads. This allows the system to establish a pipeline where, while a new frame is being processed, the *Local Mapping* and/or the *Loop Closing* operations are running on the data related to the previous frame(s). Synchronization is performed using locks and variables that indicate when the input used by a phase is ready to be processed by the following step of the algorithm. These three phases are performed for both the *Monocular* and the *Stereo* version of the ORB-SLAM; the difference lies in the *Tracking* phase, particularly in the *ORB Extraction* phase. In the *Stereo* version, the *ORB Extraction* phase is performed on both the left and right input images and the results are then merged with an additional *Stereo Matching* phase.

6.2.1 Tracking

The primary goal of the *Tracking* phase is to process each incoming image, also known as a frame, from the camera and calculate the estimated camera pose. Additionally, this phase determines whether or not to insert a *Keyframe* into the system. A *Keyframe* is a data structure that stores the camera pose and the salient image pixels, referred to as points, that



(a) Initial Pose Estimation. (b) Track Local Map.

Figure 6.3: Tracking phase: schema of Initial Pose Estimation and Track Local Map.

were extracted during the processing. Additionally, for each point, its ORB descriptor [105] is recorded.

The ORB descriptor is a scale and rotation-invariant fingerprint of the point, which is used to recognize the same point across different frames. This enables the linking of different *Keyframes* that share points and the construction of a co-visibility graph [106]. This graph can then be used to extract a *local map* that is employed to localize the camera within the environment.

The Tracking phase of ORB-SLAM is composed of three main procedures: *ORB Extraction*, *Initial Pose Estimation*, and *Track Local Map*. The first procedure involves extracting relevant points from the input image and computing the ORB descriptor for each point. The second procedure utilizes these extracted points to estimate the camera pose by matching them with points from previous iterations and utilizing a velocity motion model, as illustrated in Figure 6.3a. The final procedure, *Track Local Map*, uses the estimated camera pose and extracted points to find additional correspondences in the local map constructed from the co-visibility graph of previous iterations, as shown in Figure 6.3b. The decision of whether to save new information as a *Keyframe* to improve the co-visibility graph is made in this phase. In cases where *Initial Pose Estimation* is unable to compute the camera pose, due to occlusion for example, a global re-localization is performed by querying all *Keyframes* to find the best match among the extracted points.

The *ORB Extraction* phase is a crucial component of the Tracking phase, and it is detailed in Section 6.2.4. The proposed implementation, presented in Section 6.3, specifically focuses on optimizing this phase of the ORB-SLAM algorithm.

6.2.2 Local Mapping

The *Local Mapping* phase is activated when a new *Keyframe* is inserted. This phase is responsible for maintaining and updating the co-visibility graph of the environment. The first step is to insert the new *Keyframe* into the graph and link it to other *Keyframes* that share similar points. To ensure accurate associations, a point must be observed for three consecutive frames before it is considered valid; otherwise, it is discarded. Next, the *Map Points Culling and New Points Creation* phases are executed, where new points are projected onto connected *Keyframes* and existing points are culled if they are no longer valid.

A *Local Bundle Adjustment* (BA) [107] is then performed, where the new *Keyframe* and connected *Keyframes* are used to minimize the re-projection error of points. This results in the adjustment of the position of the points, taking into account all the *Keyframes* in which they were detected. Finally, the *Keyframe Culling* phase is executed, where redundant *Keyframes* are identified and removed. A *Keyframe* is flagged for deletion if 90% of its points are present in at least three other *Keyframes*.

6.2.3 Loop Closing

The *Loop Closing* phase is responsible for detecting when the camera reaches an already visited location by searching for loops in the new *Keyframes* processed by the *Local Mapping* phase. The process begins by extracting a set of candidate *Keyframes* by comparing their similarity with the new *Keyframe*. A candidate is considered valid if there are at least three other connected *Keyframes* that have been detected consecutively. Next, a similarity transformation is calculated between the new *Keyframe* and the candidates. If there are candidates with sufficient common points, then it is considered a loop *Keyframe*. The similarity function is then used to correct the current pose of the new *Keyframe* and the connected *Keyframe* in order to align both sides of the loop.

6.2.4 ORB Extraction Details

The *ORB Extraction* part, which is the main focus of the proposed implementation, is composed of six sub-parts: *Pyramid*, *FAST*, *Distribute Octree*, *Orientation*, *Gaussian Blur*, and *ORB Descriptor*.

Pyramid

The purpose of the *Pyramid* construction phase is to generate multiple versions of the input image at different levels of scaling, as described in [108]. Typically, this is achieved through a sequential scaling process, where each level is constructed from the preceding one. For example, level 1 is generated from the original image scaled by a predefined factor, and level 2 is constructed from level 1, and so forth. As a result, this phase produces a set of scaled versions of the input image as its output.

FAST

The FAST algorithm [109] is a technique for detecting corners within an image, which are defined as pixels that have significantly different luminance values compared to their neighboring pixels. The algorithm assigns a score to each detected corner based on the values of the surrounding pixels and uses a threshold value to determine which pixels are considered corners. Additionally, a *non-maximum suppression* step is applied to eliminate adjacent pixels that are also flagged as corners, leaving only the one with the highest score.

In the ORB-SLAM system, the FAST algorithm is applied to the images at each level of the *pyramid*, generated in the previous step. To handle images with varying levels of luminosity, the system applies the FAST algorithm multiple times, using decreasing threshold values if no high-threshold points are found. The output of this phase is a set of points representing the corners and their associated scores, one set for each level of the *pyramid*.

Distribute Octree

The ORB-SLAM method requires a minimum number of extracted features for each level, as the features are the points with their descriptors, and it is important to keep the number of these points to a minimum in order to balance accuracy and computational load. To achieve this, a filter phase is applied. The aim of the filter is to preserve isolated points and prune less significant points in dense areas of the image. In other words, the filter attempts to prune as many points as possible while still ensuring the minimum number of points required by the system.

The ORB-SLAM system uses the Octree distribution [110] as a filtering mechanism. This approach preserves the points distribution, meaning that areas of the frame in which more points have been detected will still feature

higher point counts after filtering. Additionally, if an area of the input image only displays a single point, such a point will not be erased.

The method recursively divides the image into four areas and counts the points in each area. If there is only one point, it will not be filtered out. If there are more points, the area is recursively divided into four sub-areas. If the minimum number of points required by the system is reached and there are areas with more than one point, the point with the maximum score, as computed in the FAST operation, is selected from each area.

The method is applied to each set of points (one for each level) returned by the FAST operation. The sets containing the filtered points are then returned as the result.

Orientation

The orientation is computed as the angle of the *intensity centroid* [111], which is the mean direction of the gradient on the patch of pixels surrounding the point. The intensity centroid is calculated as the first moment of the gradient vector in that patch.

Considering a round patch of surrounding pixels on the considered point, and considering the value of the pixel at (x, y) as $p(x, y)$. the centroid is computed as in eq. (6.1), where m_{pq} is computed as in eq. (6.2).

$$\theta = atan2(m_{01}, m_{10}) \tag{6.1}$$

$$m_{pq} = \sum_{x,y} x^p y^q I(x, y) \tag{6.2}$$

This orientation value is then assigned to the point, and it will be used in the following steps of the ORB-SLAM pipeline, such as the computation of the ORB descriptor.

Gaussian Blur

A blur filter is applied to the *Pyramid* images as required by the computation of the further point descriptors.

ORB Descriptors

The ORB descriptor is based on the BRIEF descriptor [112], which is a binary string representation of an image patch constructed from a set of binary intensity comparisons. The ORB descriptor is generated by selecting a set of pixel pairs, and for each pair comparing the intensity value of one

pixel to the other. The result of these comparisons is concatenated to form the final descriptor, which is a compact and efficient representation of the image patch. Equation (6.3) shows the mathematical formulation of the BRIEF binary test.

$$\tau(p, k_1, k_2) = \begin{cases} 1 : p(k_1) < p(k_2) \\ 0 : p(k_1) \geq p(k_2) \end{cases} \tag{6.3}$$

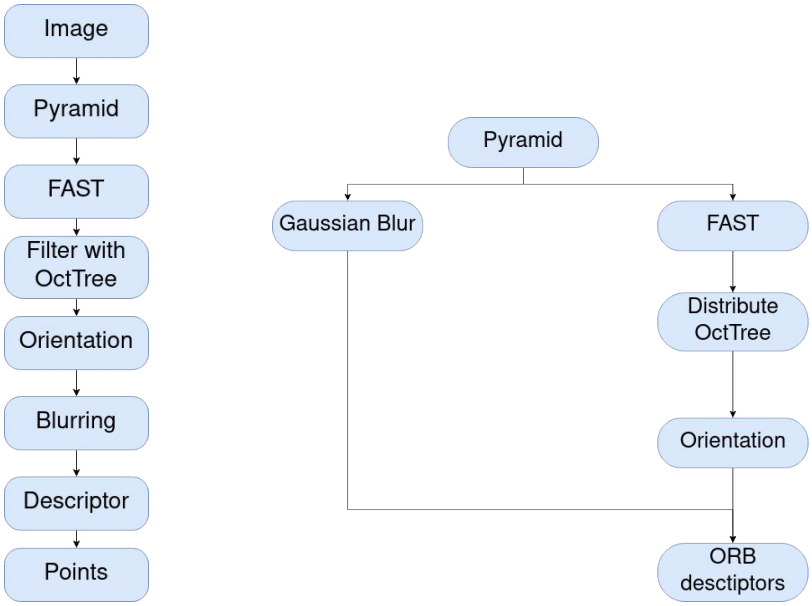
where p is the image patch around the point and $p(k)$ denotes the intensity of the pixel in such position $k = (x, y)$. However, BRIEF descriptors have a limitation in that they do not take into account point orientation, which can result in failure to recognize the same point across multiple frames if it is rotated even by a few degrees. To address this, the ORB descriptor [105] includes a correction that considers the orientation θ computed during the *Orientation* phase. This is done by using a rotation matrix R_θ and applying it to a matrix composed of the coordinates of the selected points within the patch used to compute the BRIEF descriptor. The rotated coordinates are then used to compute τ, which is the string bit associated with the considered patch as a function of the considered pixel position and its intensity value.

The ORB Descriptor phase computes a descriptor for each point, considering the orientation information obtained from the previous Orientation phase. The computation is performed independently for each level of the image *pyramid*, as produced by the *Pyramid* construction phase.

6.3 ORB-SLAM Proposed Implementation

The proposed GPU-based implementation [113] of the ORB Extraction phase aims to improve computational efficiency by exploiting the parallel processing capabilities of a GPU. As described in Section 5.2.1, many of the operations within the ORB Extraction phase, such as the FAST corner detection, Octree distribution filtering, and Orientation computation, can be executed in parallel.

The implementation's flow is structured to leverage this parallelism to optimize the overall performance of the *ORB Extraction* phase, as illustrated in Figure 6.4b that defers from the baseline sequential flow illustrated in Figure 6.4a. Specifically, the *Gaussian Blur* task, which is only required by the ORB descriptor task, is executed concurrently with the execution of the *FAST*, *Distribute Octree*, and *Orientation* phases, thus allowing for overlapping computation and reducing overall execution time.



(a) Sequential Flow. (b) Concurrent Flow.

Figure 6.4: *ORB Extraction* flows.

Many tasks in the baseline, sequential CPU implementation such as *Pyramid*, *Gaussian Blur*, *FAST*, *Orientation*, and *ORB descriptor* can be implemented as CUDA kernels. However, the *Distribute Octree* task, which is used to filter points and preserve their distribution, is not found to be suitable for GPU implementation. This is because, while there are various GPU implementations for Octree construction [114, 115], they have been found to show performance benefits only for a large number of input points, and the FAST corner detection with Non-max suppression returns significantly fewer points in this specific application. Moreover, in the baseline Octree implementation within ORB-SLAM, the Octree is constructed and traversed at the same time: since the Octree construction must stop when the desired number of features is reached (see Section 6.2.4), it is necessary to introduce synchronization among GPU threads to signal the end event and code divergence that would significantly hamper the GPU throughput, i.e. causing violation to the lock-step execution model. An alternative would be to complete the Octree construction to then traverse it at a later stage: in this case, it must be forced the GPU to perform additional unnecessary work. To conclude, the convenience of a parallel GPU implementation is

lost.

Once potential CUDA kernels have been identified, it is possible to define the optimal number of CUDA *streams* considering the tasks that can be computed concurrently. At the end the resulting graph of CPU and GPU interactions is shown in Figure 6.5. First, the input image is copied on the device (GPU) memory; then a kernel constructs the *pyramid* image levels (see Section 6.3.1 for more details); at this stage, three different CUDA *streams* are used: one to compute the blurred images, one to copy the images to the CPU host and the last one to perform the *FAST* algorithm. These three tasks only depend on the *pyramid* construction so all of them can be executed concurrently. After the *FAST* operation, the *Orientation* step can be computed: the output of this phase as well as the blurred image will be the input for the following *ORB Descriptor* phase. Synchronization for these stages is achieved through CUDA *events*. After the *ORB Descriptor* task, the extracted points must be copied to the Host memory and translated to the OpenCV[1] data structure used for the rest of the SLAM system; after this phase, the *Distribute Octree* task is performed. Note that the sequence of operations compared to the sequential flow is different, as the *Distribute Octree* is now at the end of the extraction phase, whereas in the sequential flow, it is just after the *FAST* algorithm. The reason for this change is that is it more convenient to compute *Orientation* and *ORB Descriptor* before the filtering operation operated by the *Distribute Octree*. This might be counter-intuitive from the point of view of maximizing performance, as in this way, we are forcing the system to process (at the *Orientation* and *ORB Descriptor* stages) more points compared to the baseline implementation. However, by proceeding with a different order of operations the number of data transfers from CPU memory and GPU memory is minimized. This is a critical aspect to consider when porting non-trivial CPU applications to GPU, as memory copies might account for a non-negligible chunk of the total execution time. Memory copies from GPU memory to CPU memory are necessary as, in this version, *Distribute Octree* executes on the CPU whereas the other kernels are on GPU. So looking at Figure 6.4b, and assuming there is a CUDA kernel for both *FAST* and *Orientation*, two memory copies between CPU and GPU address spaces are mandatory: one just after the *FAST* (copy device to host) and one before the *Orientation* phase (copy host to device). Conversely, in Figure 6.5, the only memory copy needed is a device to host memory transfer operated at the end of the *ORB Descriptors* step.
Considering integrated GPU, in which the memory is shared between CPU

[1] https://opencv.org

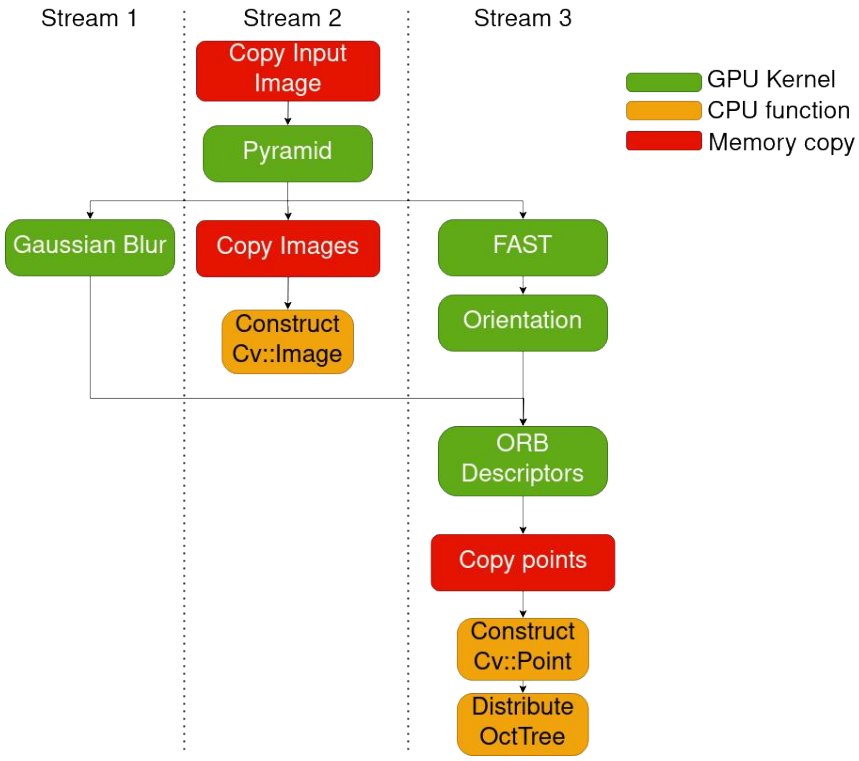


Figure 6.5: GPU Flow.

and GPU, CUDA UM (Unified Memory) can be exploited but, as reported in different works [116] [117] [118], each application have different behavior, so it is not guaranteed that UM improves the performances and it must be investigated. This aspect can be investigated in the future but it is not the focus of this proposal and relative experiments.

6.3.1 ORB-SLAM Kernels descriptions

In this section, the details of the individual ORB-SLAM operations that have been ported to the GPU as CUDA kernels are discussed.

Image Pyramid

The GPU implementation of the image *pyramid* construction differs from the baseline method in that it computes each level simultaneously, starting from the original image. Each pixel of each level is assigned to a CUDA GPU

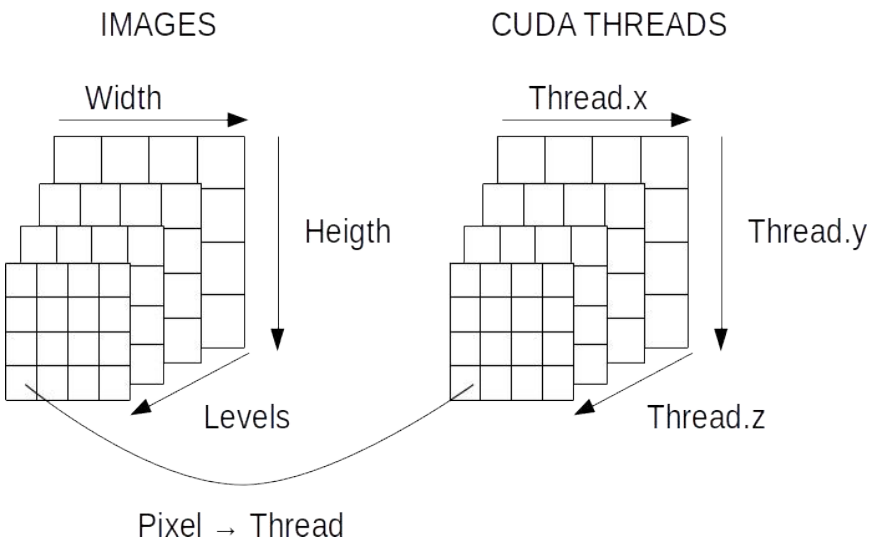


Figure 6.6: CUDA Threads mapping: each pixel of each level is assigned to one CUDA Thread.

thread, which computes its value independently, resulting in a reduction of dependencies among the *pyramid*'s subsequent levels. The scaling factor f of the current pixel level is used to determine the reference pixels from the original image, as specified in equation (6.4).

$$\begin{aligned} x_{up} &= \lceil x * f \rceil \\ x_{low} &= \lfloor x * f \rfloor \\ y_{up} &= \lceil y * f \rceil \\ y_{low} &= \lfloor y * f \rfloor \end{aligned} \tag{6.4}$$

Using the reference pixels, the thread computes the new pixel value q as in eq. 6.5.

$$\begin{aligned} q_1 &= v_1 * \frac{x_{up} - (x * f)}{x_{up} - x_{low}} + v_2 * \frac{(x * f) - x_{low}}{x_{up} - x_{low}} \\ q_2 &= v_3 * \frac{x_{up} - (x * f)}{x_{up} - x_{low}} + v_4 * \frac{(x * f) - x_{low}}{x_{up} - x_{low}} \\ q &= q_1 * \frac{y_{up} - (y * f)}{y_{up} - y_{low}} + q_2 * \frac{(y * f) - y_{low}}{y_{up} - y_{low}} \end{aligned} \tag{6.5}$$

where v_1, v_2, v_3 and v_4 are respectively the pixel value at coordinates (x_{low}, y_{low}), (x_{up}, y_{low}), (x_{low}, y_{up}) and (x_{up}, y_{up}) from the original image.

Because each level is constructed from the input image, the results are qualitatively equal with respect to the baseline method, however, by removing the chain of dependencies we gain in execution latencies and GPU resource utilization. Eventually, the main difference between the novel method and the baseline method lies in the reference pixel selection. The impact on the accuracy of this change will be discussed later in section 6.4.

Gaussian Blur

The Gaussian Blurring algorithm applies a filter to each pixel of an image. To implement this operation on a GPU, it is straightforward to assign a single thread to the computation of each pixel. Additionally, as the blurring must be applied to images belonging to each level of the pyramidal image, it is possible to compute each level concurrently. Therefore, a GPU kernel is launched, with a thread assigned to each pixel of each image level, as illustrated in Figure 6.6.

FAST

This kernel is based on the kernel implemented by [100] in which each kernel block is a tile of image and GPU shared memory is exploited to check if points are found and eventually decrease the threshold.
The proposed modified version introduces the concurrency also in the level dimension and a limit to the number of detected points is not forced as authors did in [100]. In the end, the launch configuration for this kernel remains the same as that depicted in Figure 6.6, utilizing the same block configuration as in [100] to take advantage of shared memory for point checking and threshold decreasing.

Orientation

For each point extracted by the FAST algorithm (and filtered by the *Distribute Octree* phase), it is necessary to compute the *Orientation*. This phase was implemented as a kernel that launches a thread for each point of each level and computes the orientation (as illustrated in Figure 6.6). Each thread computes the orientation associated with its point as described in Section 6.2.4.

ORB Descriptor

After the *Orientation* phase, the *ORB Descriptor* phase is executed.

To perform this computation, a CUDA kernel is implemented that launches a thread to compute the descriptor for each point on each level of the *pyramid*. The thread configuration follows the same pattern as previous kernels, with a thread assigned to each point of each level, as illustrated in Figure 6.6. The descriptor calculation is performed according to the method described in Section 6.2.4.

6.3.2 Distribute Octree Variant: a novel clustering-based filter

A different approach for point filtering was implemented to achieve the same properties as the *Distribute Octree* method, as it was deemed not suitable for GPU implementation.
The purpose of this variant is to apply a filter to the detected points directly on GPU. By applying the filter on the GPU it is possible to save data transfers (fewer points are copied from the device to host memory) and computational time since the following phases *Orientation* and *ORB Descriptor* have to compute fewer points. Indeed, in the flow shown in Figure 6.5, these phases compute all points and then the filter is performed on the CPU; instead, filtering the points on GPU immediately after the *FAST* phase results in a less load for *Orientation* and *ORB Descriptor* phases.

The final flow of this variant is shown in Figure 6.7.
The novel method is based on a clustering technique and it is articulated in five phases:

Centroid The initial phase involves pre-computing the location of centroids based on the size of the image. These centroids are evenly spaced to cover the entire image surface, resulting in a uniform distribution across the image. The number of centroids corresponds to the number of features required for the given level, as discussed in Section 6.2.4.

Cluster Assignment In this phase, a kernel is launched with a thread for each point extracted by the FAST algorithm for each level in parallel. The kernel computes the distance of each point from every centroid and links the point to the nearest centroid. As a result, each point is assigned to a cluster, and multiple points can be assigned to the same cluster.

Compute the max score For each cluster it is necessary to only take the point with the highest score (calculated by FAST). In order to select the point with the highest score from each cluster, a kernel is launched with a thread for each point of each level in parallel. These threads are responsible for performing a CUDA *atomicMax* operation, comparing the point's score with the current maximum score of the cluster to which it belongs. As a result, the maximum score of all points within each cluster is obtained.

Max score point selection In order to determine the point with the highest score for each cluster, a kernel is launched with one thread for each point for each level in parallel. The thread retrieves the cluster assigned to the point and the maximum score of that cluster. The point score is then compared to the cluster's maximum score. If the two values match, the point is considered the best for that cluster. Otherwise, the point is marked as not selected. It should be noted that there may be cases where multiple points within a cluster have the same score, in which case a point is randomly chosen among those with the same score, as there is no specific tie-breaking mechanism in place at this stage.

Last Cluster Assignment In this phase, a mechanism is put in place to handle the possibility that certain clusters may not have any points assigned to them. This can occur, for example, if all corners in an image are located in a small region, resulting in all points being assigned to a single cluster. To ensure that the required number of features is met, an additional cluster assignment procedure is implemented. A CUDA kernel is launched, with a GPU thread for each cluster for each level. The thread checks if its corresponding cluster has at least one point assigned to it. If this is the case, the thread terminates. If the cluster does not have any points assigned, the GPU thread will iterate through all remaining unassigned points and calculate the distance between each point and the cluster centroid. The nearest point is then assigned to the cluster. This process is repeated until all clusters have at least one point assigned to them.

6.4 ORB-SLAM Experiments

In the experiments there are compared the two novel implementations (figures 6.5 and 6.7) with the implementations by [100][2] and [104][3], as authors of

[2]https://github.com/yunchih/ORB-SLAM2-GPU2016-final
[3]https://github.com/xaldyz/dataflow-orbslam

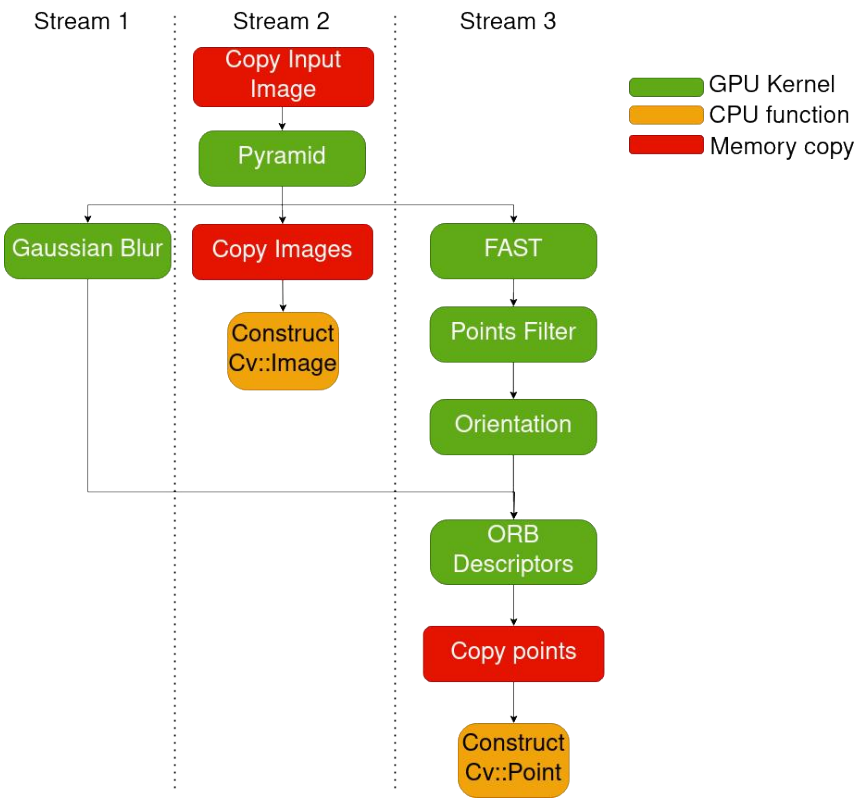


Figure 6.7: Our improved GPU version with clustering-based filtering instead of Octree.

those contributions have made publicly available their source codes. Results are reported in tables and figures as *ORB-GPU* for [100] implementation, as *ORB-dataflow* [104] implementation, as *O-nc* for the proposed novel implementation represented figure 6.5 and as *O-c* for the proposed novel implementation represented figure 6.7. There are also reported measures regarding the original ORB-SLAM2 [4]. The base code of all these implementations is derived from ORB-SLAM2 [94] and all of these implementations exploit the GPU acceleration, except for the original ORB-SLAM2 that is fully implemented on the CPU. In the experiments, there are used sequences 04, 06, and 07 from the Kitti dataset [96] as tests scenarios and it is measured the computational time needed by the *Tracking phase* of the ORB-SLAM to process a single frame in the *Monocular* and *Stereo* version. The system processes a single image in the first case, whereas there are two images in the second case. Also, the trajectory errors are measured using the tool released in [119] with the same scaling and alignment described in that paper (7dof for Monocular and 6dof for Stereo): this tool measures the average translational error $t_{err}(\%)$ and rotational error $r_{err}(^\circ/100\text{m})$, the root-mean-square error between predicted camera poses and ground truth ATE(m) and the frame-to-frame relative pose error RPE both in terms of meters (m) and degrees (°). Sequences 04, 06, and 07 were chosen because they are captured in very different situations: in sequence 04 there are mostly straight trajectories; sequence 06 features a large number of close turns, whereas sequence 07 represents an urban scenario that mixes both straightforward and curvy paths.

The two novel proposed implementations (*O-nc* and *O-c*) were also tested using the ORB-SLAM3 [95][5] as base code and they have compared it with the original version of ORB-SLAM3 using the first sequence of the Euroc Dataset that is a scenario designed for autonomous drones. The baseline ORB-SLAM3 implementation also runs exclusively on the CPU. In this case, we have used the tool released with the code of ORB-SLAM3 to measure the absolute translational error ATE(m) between predicted camera poses and ground truth. We report the mean, the standard deviation, and the root-mean-square error (RMSE).

6.4.1 Hardware setup

The experiments were run on the Nvidia Xavier AGX board. This embedded board is equipped with a CUDA-capable GPU (512 NVIDIA CORE) and

[4]https://github.com/raulmur/ORB_SLAM2
[5]https://github.com/UZ-SLAMLab/ORB_SLAM3

an ARM CPU (8 core); it has 32GB of memory. To get the maximum performance, it was set the fan at maximum speed and it was used the MAXN mode with the *jetsonclock* script. In this setting the board reaches the frequency of 2265.6 MHz for CPU, 1377 MHz for GPU, and 2133 MHz for memory; this is the maximum performance capabilities of this board.

6.4.2 ORB-SLAM2 Results

In Table 6.1 are reported the computation time results and in Table 6.2 the trajectory errors on the KITTI dataset in the *Monocular* version. In Table 6.3 and Table 6.4 the same metrics have been reported for the *Stereo* version. The ORB-dataflow version does not support *Stereo* mode so the results using this implementation are not reported in *Stereo* version.
In Figure 6.8, the relative performance gain of the proposed implementations is illustrated by comparing their execution time with that of the original ORB-SLAM2 implementation. The speedup is computed as the ratio of the mean execution time of the original ORB-SLAM2 implementation to that of the proposed implementations. The *O-c* implementation presents the best mean and median time in all situations. Moreover, *O-nc* implementation shows a promising computational time: it is higher than *O-c* version but still lower than all the other implementations. Comparing various GPU implementations of ORB-SLAM2 with respect to the original one, in the *Stereo* mode, the ORB-GPU implementation demonstrates a speedup of 1.48x, 1.13x, and 1.11x in the KITTI04, KITTI06, and KITTI07 sequences, respectively. The *O-nc* implementation shows a speedup of 1.7x, 1.28x, and 1.24x in the KITTI04, KITTI06, and KITTI07 sequences, respectively. Lastly, the *O-c* implementation exhibits a speedup of 1.8x, 1.38x, and 1.34x in the KITTI04, KITTI06, and KITTI07 sequences, respectively. In the *Monocular* mode, the ORB-GPU implementation demonstrates a speedup of 2.38x, 2.13x, and 2.32x in the KITTI04, KITTI06, and KITTI07 sequences, respectively. The ORB-dataflow implementation shows a speedup of 1.55x, 0.71x, and 0.74x in the KITTI04, KITTI06, and KITTI07 sequences, respectively. The *O-nc* implementation exhibits a speedup of 2.74x, 2.77x, and 2.80x in the KITTI04, KITTI06, and KITTI07 sequences, respectively. Lastly, the *O-c* implementation demonstrates a speedup of 3.06x, 2.96x, and 3.05x in the KITTI04, KITTI06, and KITTI07 sequences, respectively. The two novel implementations show the highest speed up in both modalities (*Monocular* and *Stereo*.

As far as tracking errors are concerned, for the monocular version, the original ORB-SLAM2 shows the best results for sequence 06 in ATE and t_{err},

		ORB-SLAM2 [94]	ORB-GPU [100]	ORB-dataflow [104]	*O-nc*	*O-c*
KITTI04	median	48.413	19.879	22.460	16.581	**15.023**
	mean	49.039	20.568	31.556	17.870	**15.979**
	std	**6.701**	8.065	138.247	10.928	7.752
	min	42.156	14.983	33.752	11.935	**11.900**
	max	145.387	141.620	44.609	131.922	**28.566**
KITTI06	median	47.514	20.167	21.253	14.845	**14.556**
	mean	49.719	23.369	70.273	17.979	**16.822**
	std	**8.094**	9.652	1628.210	11.571	8.399
	min	40.599	15.053	12.405	**10.601**	10.802
	max	207.565	214.655	194.517	235.658	**164.536**
KITTI07	median	49.120	19.739	21.193	16.776	**15.353**
	mean	50.077	21.559	67.521	17.887	**16.433**
	std	8.735	8.758	1535.450	**8.047**	8.107
	min	39.392	14.271	12.372	12.753	**11.160**
	max	260.271	240.451	276.189	**229.278**	235.644

Table 6.1: Computational time (ms) for KITTI dataset in Monocular version.

ORB-dataflow in r_{err} and *RPE(°)*, instead *O-nc* implementation show the best result for *RPE(m)*; for sequence 04 *O-c* implementation shows the best results for two measures (t_{err} and ATE), for the other three measures the best results are obtained by ORB-dataflow; for the sequence 07 *O-nc* shows the best result only for *RPE (m)* measure, instead the original ORB-SLAM2 works better in *RPE(°)*, the other best measures are obtained by ORB-GPU. Also in the stereo version *O-c* implementation shows the best computational times for all sequences and *O-nc* implementation keeps showing promising results. In this case, the two novel implementations perform notably better in trajectory errors: they achieve the best results in all metrics except for t_{err} and ATE in sequence 04 (that are achieved by ORB-GPU) and t_{err} in sequence 07 (that are achieved by original ORB-SLAM2).

6.4.3 ORB-SLAM3 Results

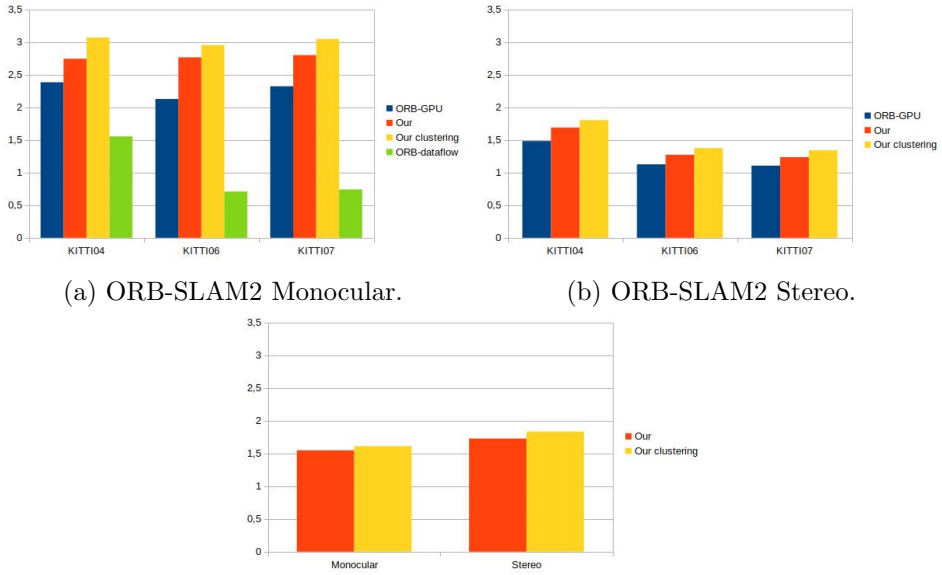
In Table 6.5 are reported the computational time comparison of the two novel implementations *O-nc* and *O-c* (with ORB-SLAM3 as code base) with the original ORB-SLAM3 and in Table 6.7 are reported the results errors

		ORB-SLAM2 [94]	ORB-GPU [100]	ORB-dataflow [104]	*O-nc*	*O-c*
KITTI04	t_{err}(%)	1.672	0.668	0.743	0.628	**0.570**
	r_{err}(°/100m)	0.286	0.404	**0.222**	0.293	0.299
	ATE(m)	1.645	0.552	0.633	0.495	**0.374**
	RPE(m)	0.060	0.069	**0.023**	0.039	0.034
	RPE(°)	0.121	0.138	**0.104**	0.122	0.119
KITTI06	t_{err}(%)	**6.901**	7.806	7.483	14.759	62.816
	r_{err}(°/100m)	0.440	0.715	**0.434**	2.960	27.497
	ATE(m)	**13.443**	16.172	16.229	22.953	110.478
	RPE(m)	0.977	0.578	0.665	**0.576**	0.800
	RPE(°)	0.138	0.196	**0.131**	0.367	0.912
KITTI07	t_{err}(%)	6.686	**6.509**	6.701	7.437	7.541
	r_{err}(°/100m)	2.299	**2.279**	2.301	2.687	2.691
	ATE(m)	4.299	**4.260**	4.438	4.768	4.551
	RPE(m)	0.463	0.464	0.569	**0.424**	0.471
	RPE(°)	**0.387**	0.390	0.388	0.423	0.423

Table 6.2: Trajectory error for KITTI dataset in Monocular version.

		ORB-SLAM2 [94]	ORB-GPU [100]	*O-nc*	*O-c*
KITTI04	median	113.179	76.412	67.006	**62.459**
	mean	113.471	76.560	67.189	**62.904**
	std	6.338	**4.592**	5.920	5.763
	min	101.737	68.430	58.278	**53.602**
	max	178.186	**110.765**	137.335	129.732
KITTI06	median	90.276	76.817	69.316	**64.874**
	mean	92.102	81.771	72.370	**66.981**
	std	**13.724**	14.674	15.542	14.990
	min	73.580	65.330	53.982	**50.345**
	max	391.772	**341.873**	453.502	462.290
KITTI07	median	86.445	77.942	69.205	**64.057**
	mean	87.546	79.150	70.828	**65.256**
	std	**7.692**	10.236	12.751	10.496
	min	72.527	67.656	57.499	**53.741**
	max	**238.742**	360.060	424.848	343.366

Table 6.3: Computational time (ms) for KITTI dataset in Stereo version.



(a) ORB-SLAM2 Monocular.

(b) ORB-SLAM2 Stereo.

(c) ORB-SLAM3.

Figure 6.8: Speedup respect to original ORB-SLAM2 and ORB-SLAM3 implementation.

		ORB-SLAM2 [94]	ORB-GPU [100]	*O-nc*	*O-c*
KITTI04	t_{err}(%)	0.444	**0.376**	0.387	0.479
	r_{err}(°/100m)	0.233	0.226	0.238	**0.155**
	ATE(m)	0.209	**0.162**	0.172	0.221
	RPE(m)	0.018	0.018	**0.016**	0.017
	RPE(°)	0.034	0.035	0.033	**0.032**
KITTI06	t_{err}(%)	0.499	0.565	**0.465**	0.507
	r_{err}(°/100m)	0.164	0.193	**0.131**	0.172
	ATE(m)	0.727	0.920	**0.609**	0.774
	RPE(m)	0.015	0.018	0.015	**0.014**
	RPE(°)	0.035	0.038	**0.035**	0.035
KITTI07	t_{err}(%)	**0.483**	0.514	0.493	0.578
	r_{err}(°/100m)	0.282	0.297	**0.253**	0.311
	ATE(m)	0.871	**0.805**	1.380	0.957
	RPE(m)	0.014	0.014	**0.013**	0.014
	RPE(°)	0.039	0.040	**0.038**	0.042

Table 6.4: Trajectory error for KITTI dataset in Stereo version.

for the Monocular version. In Table 6.6 and Table 6.8 there are reported the same measures but for the Stereo version. The speedup (considering the mean time) of the two novel implementations with respect to the original ORB-SLAM3 is reported in Figure 6.8c.
O-c implementation shows the best results in terms of computational time in both situations (Monocular and Stereo). The speedup with respect to original ORB-SLAM3 implementation is 1.55x (*Stereo*) and 1.73x (*Monocular*) considering *O-nc* implementation; 1.61x (*Stereo*) and 1.83x (*Monocular*) considering *O-c* implementation. In terms of trajectory errors, *O-nc* implementation shows the best results for the Monocular case; instead, for the Stereo version, the original version shows the best results for two metrics: std and RMSE, *O-c* implementation shows the best result for the last metric: mean.

6.4.4 Discussion

In the ORB-SLAM3 system, the two novel implementations perform well in terms of computational time, the *O-c* version is the fastest but also *O-nc* version is faster than the baseline version. In the Monocular case, the errors

	median	mean	std	min	max
ORB-SLAM3 [95]	28.377	29.555	8.427	19.388	118.630
O-nc	17.632	19.095	10.058	11.720	441.696
O-c	**16.305**	**18.352**	**5.229**	**11.041**	**67.279**

Table 6.5: Computational time (ms) for ORB-SLAM3 Monocular.

	median	mean	std	min	max
ORB-SLAM3 [95]	55.767	57.068	5.473	44.941	62.634
O-nc	31.015	33.021	16.888	21.843	65.219
O-c	**29.671**	**31.115**	**5.221**	**21.790**	**13.443**

Table 6.6: Computational time (ms) for ORB-SLAM3 Stereo.

	mean	RMSE	std
ORB-SLAM3 [95]	3.059	3.324	1.300
O-nc	**1.101**	**1.195**	**0.465**
O-c	3.170	3.444	1.347

Table 6.7: Error (m) for ORB-SLAM3 Monocular.

	mean	RMSE	std
ORB-SLAM3 [95]	0.037	**0.043**	**0.023**
O-nc	0.064	0.073	0.035
O-c	**0.035**	0.045	0.041

Table 6.8: Error (m) for ORB-SLAM3 Stereo.

are larger concerning the Stereo case; this is normal because with only one image it is more difficult to perceive the third dimension. Indeed, also the original version shows larger errors in the Monocular case. To summarize, *O-nc* implementation seems to be the best compromise for the Monocular case: it shows the best results in terms of trajectory errors and it is only 1ms slower than *O-c* version.
For the Stereo case, the *O-c* version seems to be preferable: it shows the best computational times and the trajectory error differs from the original version for just a few millimeters (*ATE1*) or centimeters (*ATE3)*.

In the ORB-SLAM2 system, *O-c* implementation is preferable in terms of computational time in both cases (Monocular and Stereo). On the other hand, *O-nc* implementation is preferable in terms of trajectory error in the Stereo case; its performance penalty concerning the *O-c* version accounts for about 5 ms only. However, it is still faster than the other implementations. It is important to notice that in the monocular scenario, no implementations can clearly outperform the others in terms of accuracy and related errors. Moreover, in the Monocular version, it is possible to see that the two novel implementations suffer in sequence 07 and mostly in sequence 06; in these sequences, there are many close turns (especially in sequence 06). In such situations, it is evident the accuracy deterioration compared to the other baseline approaches. A reason might be that there are intrinsic possibilities of different points choice for consecutive frames because the two novel methods are not deterministic in the choice of points with the same score (both *FAST* and *Octree variant* phases are affected by this behavior). This reduces the accuracy of the *localization* because a point extracted in the first frame might not be extracted in the second. Moreover, in contrast to the Cluster-filter method, which assumes that features in an image are equally distributed, the octree method takes into account the actual distribution of features by subdividing the image based on areas with higher concentrations of features. This can lead to a more effective selection of points in the octree method, as it ensures that the selected points come from the densest areas and will present higher scores. The Cluster-filter method may result in a sub-optimal feature selection, which can lead to larger errors when estimating the pose in the monocular version of the algorithm by comparison with the following frame. However, this issue is mitigated in the stereo version, as poorly selected features that do not match the other image are not considered. This effectively balances the overall accuracy of our proposed method, as shown in our experiments. Another source of discrepancies between novel approaches and the other baselines lies in the way in which it is optimized

the image *Pyramid* construction: the produced scaled versions are bound to be different compared to the serialized versions of the state-of-the-art papers. The parallel approach to computing the *pyramid* tends to produce less blurred scaled images. The results are good in a quality way but the single pixels can be different, so this can cause a different ORB descriptor that does not match the same point in the next frame.

It is also important to highlight that none of the above loss of accuracy issues is present in the stereo versions of the two novel algorithms, while still retaining better performance compared to the state-of-the-art papers.
In the end, the implementation choice is context-dependent: some implementations are better for situations in which it is possible to sacrifice precision over improved latencies; whereas in other situations it is preferable to the opposite solution.

6.5 Conclusion

This chapter addresses the problem of *localization* for autonomous vehicles. It highlights the various types of sensors that can be utilized to address this problem and presents a variety of algorithms that have been proposed in the literature to leverage these sensors. Of particular focus is the use of cameras as sensors, as they are cost-effective and easily integrated into a vehicle. Additionally, the need for optimizing these algorithms to reduce execution time on embedded boards was emphasized. The algorithm that was specifically targeted for optimization and utilizes cameras is ORB-SLAM.

In this chapter, two GPU-based implementations for the *ORB extraction* component of the *Tracking* phase in ORB-SLAM were proposed. These implementations were integrated into the ORB-SLAM2 and ORB-SLAM3 algorithms. The proposed evolution consists of the porting of most of the different steps of ORB-SLAM to the GPU. Additionally, the Pyramid algorithm is modified to be more amenable to parallelization, and a novel point filtering method is proposed. The proposed implementations are capable of supporting both *Monocular* and *Stereo* vision and are the first GPU implementation of ORB-SLAM3.

The proposed implementations were compared to state-of-the-art versions of ORB-SLAM3, ORB-SLAM2, and a GPU-enabled version of ORB-SLAM2. The results of this comparison demonstrate that the proposed implementations outperform the others in terms of computational time and,

in most cases, in terms of precision and accuracy. The improvements range from a minimum speedup of 1.24x and 1.55x up to 3.06x and 1.83x in ORB-SLAM2 and ORB-SLAM3 respectively. These performance improvements with respect to the other tested algorithms are attributed to the optimization of CUDA *streams*, leading to better utilization of GPU resources. Additionally, novel implementations for the Pyramid construction and Point filtering phases were proposed, which further contribute to the improved performance. Specifically, the proposed *Pyramid* construction method leads to a higher degree of parallelism within GPU threads, and the proposed clustering-based approach for point filtering is more amenable to GPU parallelization compared to the baseline version.

The achieved speed-up makes ORB-SLAM a more efficient algorithm for the *localization* problem and more suitable for real-time critical scenarios, such as autonomous vehicles. The reduced execution time, in combination with the use of simple sensors like cameras, makes it a viable candidate for the *localization* component of autonomous vehicles.

Chapter 7

Planning - Frenet

7.1 Planning

In the context of autonomous vehicles, the task of determining a safe and efficient path from the current location to the destination is known as *Planning*. This process involves the generation of a path that the vehicle can safely follow, which is typically represented as a list of discrete path points. The path can also be represented as a continuous function that links the starting point to the destination, following the road. This task is essential for the safe navigation of autonomous vehicles, as it allows them to make informed decisions about their movements and avoid potential hazards.

To accomplish this, the Path Planning phase is typically divided into two macro phases: the *Global Planner* and the *Local Planner*.

The *Global Planner* generates a reference path, which serves as a guide for the *Local Planner*. It is not detailed and does not take into account obstacles on the road or vehicle constraints, such as speed or acceleration.

The *Local Planner*, on the other hand, generates a detailed path by considering the reference path generated by the *Global Planner* and taking into account obstacles and vehicle constraints. Moreover, the generated path contains information about the speed or acceleration that the vehicle should try to follow.

7.1.1 Local Planner

There have been several *Local Planner* methods proposed in the literature. In [120] and [121], a path is generated while taking into account kine-

matic and dynamic constraints. The dynamic model is also considered in [122] where a Spatio-temporal lattice with feasible vehicle states is proposed. Similarly, in [123] and [124], kinematic quantities are optimized to find the path. A more complex approach, using model predictive control methods, is proposed in [125]; this approach is also used in [126] and [127]. In [128], the problem is formulated as an optimization problem and it is solved using the gradient descent method. In [129], the Euler spirals are used to describe paths of non-holonomic vehicles. A new curvature parametrization for this approach is presented in [130] and in [131] the model for generating velocity profiles is changed. The Rapid Exploring Random Tree algorithm [132] is used in [133] and [134]. In the first contribution, a closed loop system is simulated to sample a tree of trajectories, whereas in the latter the state space is explored along a given reference path. An alternative approach is to generate a single path and repeatedly refine it as per the method proposed in [135]. Other studies in the literature, such as [136, 137], focus on generating multiple paths and subsequently selecting the most suitable one based on predefined performance metrics.

The novel method presented in Section 7.3 also focuses on *Local Planner*. Specifically, it is based on the *Frenet Path Planner*.

The *Frenet Path Planner* was developed to address the limitations of previous works in specific situations, such as those related to nose-to-tail traffic [138]. Additionally, earlier research also faced the inherent limitations of working with complex formulations for curves and paths that arise when using Cartesian coordinates. To overcome these issues, recent work has focused on methods that take into account the time dimension to improve Path Planning algorithms, and also on the use of alternative coordinate systems that simplify problem formulation. For example, *Frenet Coordinates*, also known as *Frenet Frame*, can be used for this purpose. In [137] the authors used *Frenet Coordinates* to split the generation of lateral and longitudinal movements and also take into account obstacles to generate a collision-free trajectory. More recently, in [139] the authors considered dynamic objects in *Frenet Frame*. The novel implementation of *Frenet Path Planner* presented in Section 7.3 also considers obstacles as in the original *Frenet Path Planner* method, but with a focus on performance optimization. Finally, the novel implementation is tested across different data types (*double*, *float*, and *half*).

7.1.2 Use of Accelerators in Local Planner

The *Local Planner* problem can be computationally demanding, particularly for algorithms that generate multiple trajectories from which the best one is selected, such as the *Frenet Path Planner*. These algorithms can have a computational time that increases with the number of generated trajectories, which can become excessive. Furthermore, a prolonged computational time of this phase can negatively impact the entire autonomous driving pipeline. If the *Local Planner* is not able to produce a trajectory in a reasonable time frame, the vehicle may follow an outdated trajectory that does not take into account new obstacles, potentially resulting in a safety hazard. To address these issues, research has focused on optimizing the performance of *Local Planner* methods. For instance, in [140], the authors emphasized the importance of performance in the *Local Planner*. In [141], the authors parallelized the path generation on the multi-core CPU, while [130] and [142] used GPU acceleration for Path Planning. Other works have also used GPU acceleration for the A* algorithm [143] and its randomized variant [144], as well as for Unmanned Aerial Vehicle Path Planning [145] and [146]. In [147], the authors accelerated the path generation on *Frenet Coordinates* using GPU, but they did not consider obstacle avoidance. The novel proposed method, on the other hand, is not only designed for path generation but also for obstacle avoidance and it is accelerated on GPU. Moreover, it is optimally designed to overlap memory copies and kernel execution using CUDA *Streams*, and it exploits the GPU *shared memory* to compute the path cost.

7.2 Frenet Path Planner

The *Frenet Path Planner* [137] is a method used to compute a trajectory for an autonomous vehicle, by utilizing the *Frenet Coordinates* (Section 7.2.1). This method starts by considering the reference path, as generated by a *Global Planner*, represented by the blue line in Figure 7.1, which follows the center of the road. The vehicle, represented by the red box, must reach one of the possible endpoints (yellow dots).

The *Frenet Coordinates* are used to simplify the path generation process (see Section 7.2.2), by generating a set of possible paths that connect the current position of the vehicle to one of the endpoints.

Each path is then assigned a cost, which is a scalar value calculated based on a function that takes into account the path points. A common cost function is the cumulative jerk.

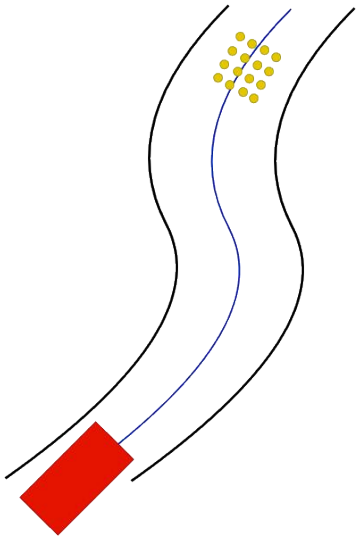


Figure 7.1: Frenet Path Planner.

Once the costs have been assigned, the paths are converted to the original *World Coordinates*, and checked for collisions with obstacles present in the environment. The path that has the lowest cost and is not obstructed by any collisions is then chosen as the final trajectory.

7.2.1 Frenet Coordinates

The *Frenet Coordinates*, introduced in [148, 149], describe the geometric properties of a curve. In the context of this work, the curve in question is the reference path that overlaps the road. By considering the position of a point on the curve at time t, denoted as $\vec{r}(t)$, and the arc length traveled by the point at time t, denoted as $l(t)$, it is possible to express the time as a function of the arc length, $t = f(l)$. This allows for the representation of the position of the point on the curve, $\vec{r}(t)$, to be rewritten as $\vec{r}(f(l))$, eliminating the need for the time variable.

The *Frenet Coordinates* are composed of three unit vectors: the tangent vector s, the normal vector d, and the bi-normal vector b ($s \times d$). These vectors are formally defined in equation (7.1).

$$s = \frac{\frac{d\vec{r}}{dl}}{\|\frac{d\vec{r}}{dl}\|} \quad d = \frac{\frac{ds}{dl}}{\|\frac{ds}{dl}\|} \tag{7.1}$$

In the *Frenet Path Planner*, the concept of *Frenet Coordinates* is used

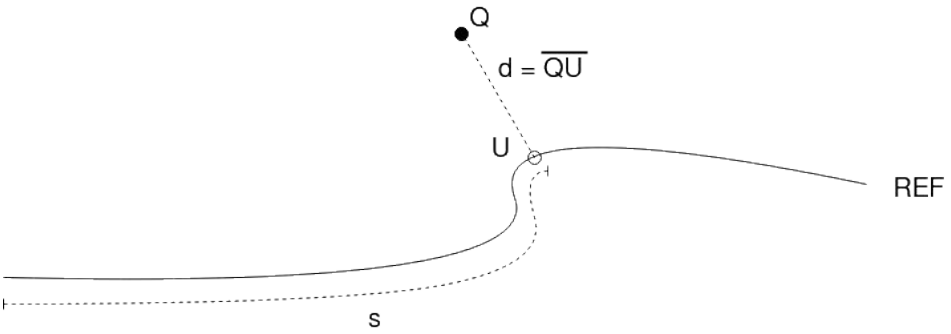


Figure 7.2: Frenet Coordinates Conversion.

to simplify the computation of the path. These coordinates provide a way to describe the geometric properties of a curve, specifically the reference path that overlaps the road. Using the *Frenet Coordinates*, the position of a point on the curve at time t can be represented by the traveled distance along the reference path, known as s, and the orthogonal displacement from the projection of the point onto the reference path, known as d.

The conversion from the original *World coordinates* (x, y) to the *Frenet coordinates*, as illustrated in Figure 7.2, is done by projecting a point Q onto the reference path REF, which is represented as a 2D spline (REF_x, REF_y). The projection U, as well as the value of s, can be computed using the Newton Method. The value of d is then obtained as the distance between the point and its projection on the reference path $\overline{QU}$. This allows for a simplified representation of the path in terms of the traveled distance and the orthogonal displacement, making the computation of the path more efficient.

7.2.2 Paths Generation

The generation of a single path in the *Frenet Path Planner* involves determining a contiguous course that connects the current position of the vehicle, represented in the *Frenet Coordinates* (s_0,d_0), to a final position (s_f,d_f). This path is represented as a curve, which can be defined by the coefficients of a fifth-degree polynomial for the lateral component (d) and a fourth-degree polynomial for the longitudinal component (s).

However, it is important to note that generating a single path is not sufficient for the planner. In order to ensure optimal performance, a set of paths that start from (s_0,d_0) and terminate at different possible endpoints

(s_f,d_f) should be generated. This set of paths can be generated by considering the initial state as defined in equation (7.2) and a possible endpoint as defined in equation (7.3).

$$S_0 =< s_0, d_0, \dot{s}_0, \dot{d}_0, \ddot{d}_0 > \tag{7.2}$$

$$S_f =< s_f, d_f, \dot{s}_f, \dot{d}_f, \ddot{d}_f > \tag{7.3}$$

In the initial state, the variables $\dot{s}_0$, $\dot{d}_0$, and $\ddot{d}_0$ represent the longitudinal velocity, lateral velocity, and lateral acceleration, respectively. Similarly, in the endpoint state, $\dot{s}_f$, $\dot{d}_f$, and $\ddot{d}_f$ represent the longitudinal velocity, lateral velocity, and lateral acceleration, respectively.

Considering the set of defined parameters:

- D_s: road width sampling length
- D_{max}: max road width
- D_{min}: min road width
- V_s: velocity sampling length
- V_{max}: max velocity
- V_{min}: min velocity
- V_{target}: desirable speed
- T_s: prediction time sampling length
- T_{max}: max prediction time
- T_{min}: min prediction time

the possible endpoint states variables can lead in a range defined by the aforementioned parameters. In particular, $\dot{s}_f$ leads in the range $[V_{min}, V_{max}]$ and d_f leads in the range $[D_{min}, D_{max}]$. Moreover, it is known that this state must be reached in the time t_f that leads in the range $[T_{min}, T_{max}]$. The sampling length of each of these values must be considered as in (7.4).

$$\begin{array}{rcl} \dot{s}_f \in [V_{min}, V_{max}] & | & \dot{s}_f = a \cdot V_s \\ d_f \in [D_{min}, D_{max}] & | & d_f = a \cdot D_s \\ t_f \in [T_{min}, T_{max}] & | & t_f = a \cdot T_s \end{array} \tag{7.4}$$

where $a \in N$.

It is possible to construct a set of possible end states SS_f that includes all possible S_f subject to the constraints expressed in (7.4). For each $S_f \in SS_f$, it is possible to compute a path that starts from S_0 and ends in S_f by resolving two different systems (one for d and one for s) as outlined in equations (7.5) and (7.6).

$$\begin{cases} d(t) = d_0 + \dot{d}_0 t + \frac{1}{2}\ddot{d}_0 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \\ \dot{d}(t) = \dot{d}_0 + \ddot{d}_0 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4 \\ \ddot{d}(t) = \ddot{d}_0 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3 \end{cases} \tag{7.5}$$

$$\begin{cases} \dot{s}(t) = \dot{s}_0 + \ddot{s}_0 t + 3a_3 t^2 + 4a_4 t^3 \\ \ddot{s}(t) = \ddot{s}_0 + 6a_3 t + 12a_4 t^2 \end{cases} \tag{7.6}$$

The coefficients a_3, a_4, a_5 in the first system and a_2, a_3 in the second can be determined by considering the variations in position, velocity, and acceleration from S_0 to S_f, resulting in the resolution of linear systems ((7.7)) and ((7.8)).

$$\begin{bmatrix} T^3 & T^4 & T^5 \\ 3T^2 & 4T^3 & 5T^4 \\ 6T & 12T^2 & 20T^3 \end{bmatrix} \times \begin{bmatrix} a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} d_f - (d_0 + \dot{d}_0 T + \frac{1}{2}\ddot{d}_0 T^2) \\ \dot{d}_f - (\dot{d}_0 + \ddot{d}_0 T) \\ \ddot{d}_f - \ddot{d}_0 \end{bmatrix} \tag{7.7}$$

$$\begin{bmatrix} 3T^2 & 4T^3 \\ 6T & 12T^2 \end{bmatrix} \times \begin{bmatrix} a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} \dot{s}_f - (\dot{s}_0 + \ddot{s}_0 T) \\ \ddot{s}_f - \ddot{s}_0 \end{bmatrix} \tag{7.8}$$

By utilizing these functions for the retrieval of position, velocity, and acceleration for the lateral component d, as well as velocity and acceleration for the longitudinal component s, it is possible to divide the path into discrete points p and define the path as a sequence of these points, denoted as $P = [p_0, p_1, ...p_f]$, with a discretization interval of T_s. Ultimately, the set of potential paths, PS, is computed through the implementation of Algorithm 2.

Each path $P = [p_0, p_1, ...p_f]$ has an associated cost C used to determine the optimal path, which is computed according to equation (7.9).

Algorithm 2 Frenet Path Generation.

Input: Initial state $S_0 =< s_0, d_0, \dot{s}_0, \dot{d}_0, \ddot{d}_0 >$
Output: Path set PS

for $d_f \leftarrow range(D_{min}, D_{max}, D_s)$ **do**
  for $t_f \leftarrow range(T_{min}, T_{max}, T_s)$ **do**
    $P_{tmp} = []$
    for $t \leftarrow range(0, t_f, T_s)$ **do**
      $p = pathPoint()$
      $p.t = t$
      $a_3, a_4, a_5 \leftarrow resolveLinearSystem(d_f, S_0)$ ▷ Eq. (7.7)
      $p.d = d_0 + \dot{d}_0 t + \ddot{d}_0 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$
      $p.\dot{d} = \dot{d}_0 + 2\ddot{d}_0 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4$
      $p.\ddot{d} = 2\ddot{d}_0 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3$
      $p.\dddot{d} = 6a_3 + 24a_4 t + 60a_5 t^2$
      $P_{tmp} \leftarrow p$
    end for
    for $\dot{s}_f \leftarrow range(V_{min}, V_{max}, V_s)$ **do**
      $P = []$
      for $t \leftarrow range(0, t_f, T_s)$ **do**
        $p \leftarrow P_{tmp}.get(t)$
        $a_3, a_4 \leftarrow resolveLinearSystem(\dot{s}_f, S_0)$ ▷ Eq. (7.8)
        $p.s = s_0 + \dot{s}_0 t + \ddot{s}_0 t^2 + a_3 t^3 + a_4 t^4$
        $p.\dot{s} = \dot{s}_0 + 2\ddot{s}_0 t + 3a_3 t^2 + 4a_4 t^3$
        $p.\ddot{s} = 2\ddot{s}_0 + 6a_3 t + 12a_4 t^2$
        $p.\dddot{s} = 6a_3 + 24a_4 t$
        $P \leftarrow p$
      end for
      $PS \leftarrow P$
    end for
  end for
end for

$$
\begin{aligned}
J_s &= \sum_{p \in P} p.\dddot{s}^2 \\
J_d &= \sum_{p \in P} p.\dddot{d}^2 \\
d_s &= (V_{target} - p_f.\dot{s})^2 \\
C_d &= k_j \cdot J_d + k_t \cdot t_f + k_d \cdot p_f.d^2 \\
C_s &= k_j \cdot J_s + k_t \cdot t_f + k_d \cdot ds \\
C &= K_{lat} \cdot C_d + K_{lon} \cdot C_s
\end{aligned}
\tag{7.9}
$$

The cost associated with each path $P = [p_0, p_1, ...p_f]$ is calculated using the function outlined in equation (7.9). The cost C is a combination of several factors, including the longitudinal jerk J_s, the lateral jerk J_d, the squared difference between the end speed and the desired speed V_{target}, and the distance of the final point p_f from the reference path. Additionally, both the lateral and longitudinal costs include a time component that penalizes paths that take longer to complete. The weighting of each component of the cost function can be adjusted through the use of multiplication factors, such as k_j for the jerk components, k_t for the time components, and k_d for the displacement components. Furthermore, the overall importance of the lateral and longitudinal costs can be adjusted with the use of multiplication factors K_{lat} and K_{lon}.

The path P, represented in the *Frenet Coordinates*, must be transformed back to the *World coordinates*. The conversion of each point $p \in P$, where $p \neq p_f$, is done using the equation outlined in (7.10).

$$
\begin{aligned}
I_x &= REF_x(p.s) \\
I_y &= REF_y(p.s) \\
yaw &= atan2(\dot{R}EF_y(p.s), \dot{R}EF_x(p.s)) \\
x &= I_x - (p.d \cdot sin(yaw)) \\
y &= I_y - (p.d \cdot cos(yaw))
\end{aligned}
\tag{7.10}
$$

Where REF_x and REF_y are the components of the reference curve as a Spline.

7.2.3 Collision Check

The final step of the *Frenet Planner* is to conduct a *Collision Check*. This involves determining whether a path P will collide with any obstacles in

the environment. Given a set of obstacles $OB = \{ob_1, ob_2, ...ob_n\}$ and an obstacle radius OR, a path P is considered safe if and only if each point on the path $p \in P$ is farther than a safe distance SD from each obstacle, as outlined in equation (7.11).

$$\sqrt{(p.x - ob.x)^2 + (p.y - ob.y)^2} - OR > SD \quad p \in P, ob \in OB \tag{7.11}$$

Finally, from the set of acceptable paths $PS' \subseteq PS$, the path with the lowest cost, as determined by the cost function outlined in equation (7.9), is selected as the optimal path for the vehicle to follow.

7.3 Frenet Path Planner Proposed Implementation

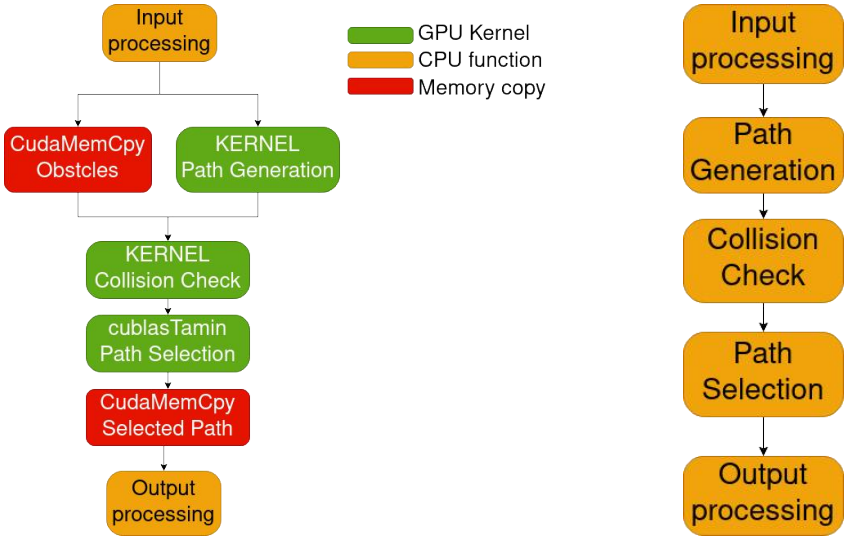
In this section, a novel GPU-based implementation of the *Frenet Planner* is proposed with the goal of reducing the computational time of the algorithm. Since both the *Path Generation* and *Collision Check* involve different independent data, it is reasonable to compute them in a parallel manner, utilizing the capabilities of a GPU. To this end, two CUDA kernels are implemented: one for *Path Generation* and one for *Collision Check*. Additionally, CUDA *Streams* are employed to overlap the execution of the kernels and memory copies as outlined in Figure 7.3a.

7.3.1 Paths Generation

The *Path Generation* phase presents several opportunities for parallel computing. By constructing the set of paths PS, it is possible to compute each path $P \in PS$ independently. Additionally, each point $p \in P$ can also be computed independently as it does not rely on other points. This allows for significant parallel computation and reduces the overall computational time.

Each path $P \in PS$ is computed by starting from specific values of d_f, t_f, and $\dot{s}_f$, as depicted in the three iterative constructs outlined in Algorithm 2 (lines 1, 2, and 14). The path P is composed of various points, each of which is calculated at different t values, as indicated by the innermost `for` loop in Algorithm 2 at lines 4 and 16. Given the independence of the values and computations described, GPU parallelism is exploited to compute them simultaneously, in order to optimize performance.

A CUDA kernel was configured with a three-dimensional launch grid that allows for concurrent computation of each path (as shown in Figure 7.4).



(a) Novel Implementation flow. (b) Baseline flow.

Figure 7.3: Implementation Flows.

The grid dimensions correspond to road width sampling d_f in the x-axis, time sampling t_f in the y-axis and velocity sampling $\dot{s}_f$ in the z-axis. Each block in the grid corresponds to a single path P, and the number of blocks present in the grid is shown in equation (7.12).

$$\frac{D_{max} - D_{min}}{D_s} \cdot \frac{T_{max} - T_{min}}{T_s} \cdot \frac{V_{max} - V_{min}}{V_s} \tag{7.12}$$

Furthermore, the computation of the points of a path is mapped to a different thread of the block associated with the path. This allows each thread to compute a point of the path considering a different value of t.

Each thread is responsible for computing the values of d, $\dot{d}$, $\ddot{d}$, $\dddot{d}$, s, $\dot{s}$, $\ddot{s}$, $\dddot{s}$, x, and y for its assigned point. Additionally, the thread calculates the cost components of its associated point ($p.\ddot{s}^2$ and $p.\dddot{d}^2$) which contribute to the final cost C of the path.

Since all cost components $p.\ddot{s}^2$ and $p.\dddot{d}^2$ must be computed in order to calculate J_s and J_d, the GPU's *shared memory* and the CUDA's *syncthreads* function are exploited. Each thread saves the $p.\ddot{s}^2$ and $p.\dddot{d}^2$ to *shared memory* and, before performing the sum over all points, the *syncthreads* function is called. After synchronization, it is guaranteed that all J_s and J_d parts are calculated, and thus it is possible to perform the sum. The sum is

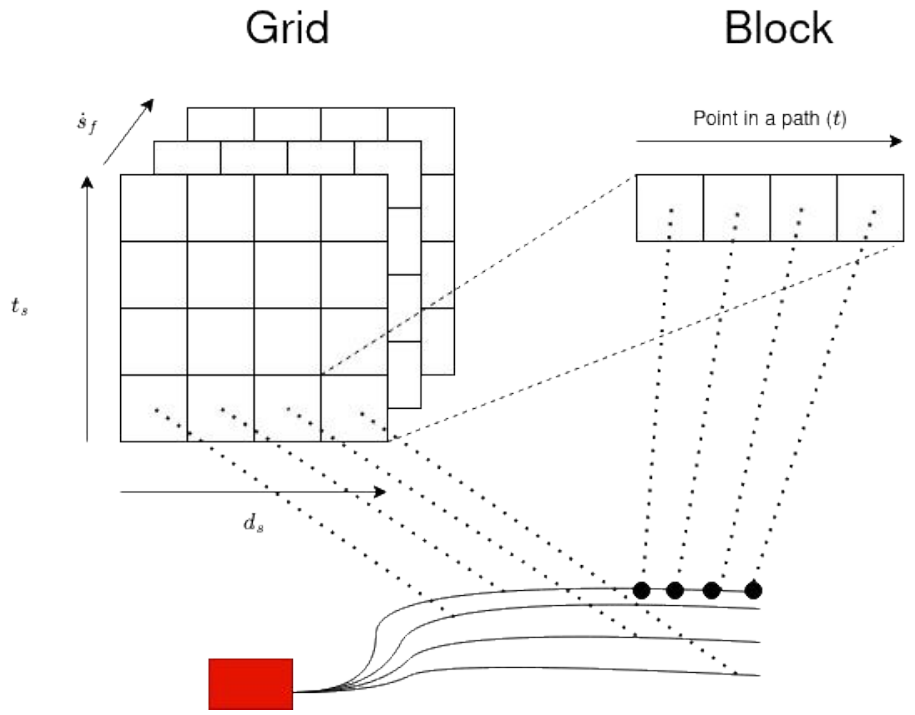


Figure 7.4: CUDA launch configuration - Paths Generation.

performed by a single thread, which is able to retrieve each part from *shared memory*.

In more detail, a thread computes all point components d, $\dot{d}$, $\ddot{d}$, $\dddot{d}$, s, $\dot{s}$, $\ddot{s}$, $\dddot{s}$, x and y (as outlined in Algorithm 2). In addition, it calculates $p.\dddot{s}^{2}$ and $p.\dddot{d}^{2}$. Then, the *syncthreads* function is called to ensure that all threads have completed the computation of these components. Finally, a designated thread performs the sum.

The use of *shared memory* to compute the cost allows for a reduction in access to GPU global memory. As *shared memory* is faster than global memory, this, combined with the parallelization described above, results in a reduction in the computational time of the algorithm.

7.3.2 Collision Check

The Collision Check kernel offers two degrees of parallelization: path points and obstacles. Each point in a path must be tested for collisions with every obstacle. As each collision check is independent of the others, all tests can

be computed by parallel GPU threads.

The kernel is launched by distributing all path points across different threads; each block has a fixed dimension of $16 \times 16 \times 4$ threads (on the x, y, and z axes respectively), for a total of 1024 threads per block (TPB). The grid is composed of an adequate number of blocks to cover the total number of points (TNP) on the x and y axes, the z-axis is used to map the obstacles (as seen in Figure 7.5). The number of blocks on the x and y axes is expressed in equation (7.13). Since each block is responsible for TPB points, in order to ensure that all points have an associated thread, the number of blocks must be greater than TNP/TPB. However, it is more efficient to distribute the needed blocks across both x and y axes, with each axis addressing the square root of the number of needed blocks. By rounding up the result, it is guaranteed that all points have an associated thread. Additionally, the number of blocks on the z-axis is equal to the number of obstacles.

$$\left\lceil \sqrt{\left\lceil \frac{TNP}{TPB} \right\rceil} \right\rceil \tag{7.13}$$

Each thread performs the collision test as detailed in equation (7.11) for one point and one obstacle. If the check fails, then the path of the point is marked as collided and the cost of the path is set to the maximum value. Once all paths have been checked, the path with the smallest cost is retrieved using the CUBlas API function call *cublasI<T>amin*[1], where T depends on the type of the array items: *d* for *double* and *s* for *float* and *half*. This function takes an array of values as input and returns the index of the smallest item.

7.4 Frenet Path Planner Results

Experiments were conducted to compare the novel implementation of the *Frenet Path Planner* with a CPU implementation[2] as a baseline. Various variants of the novel implementation were considered, including different precision types such as half, float, and double.

The CPU implementation was chosen as the baseline as its source code is publicly available, it implements the algorithm described in the original

[1]https://docs.nvidia.com/cuda/cublas/index.html#cublasi-lt-t-gt-amin
[2]https://github.com/arvindjha114/frenet_planner_agv

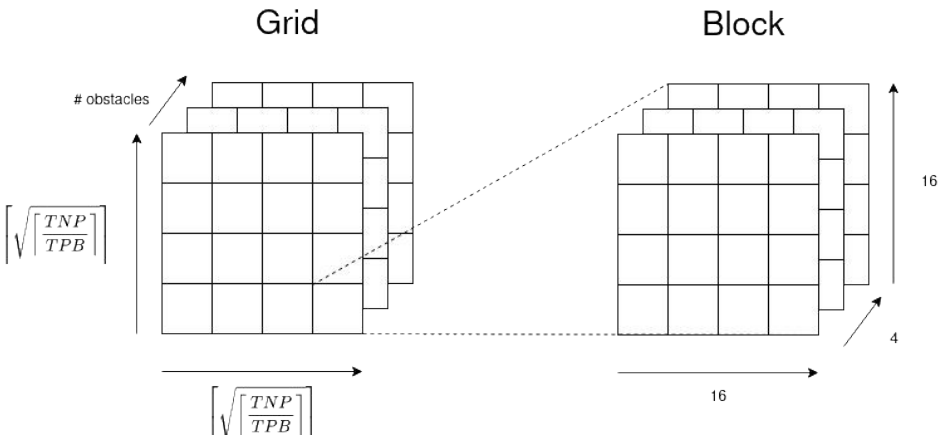


Figure 7.5: CUDA launch configuration - Collision Check.

Frenet Path Planner paper [137], and it is written in C++. Other implementations, such as the one reported in Table 7.1, cannot be fairly compared due to the unavailability of their source code. The only other previous contribution for which the code is publicly available is [139]. However, in this case, the code is written in Python, thus performance cannot be fairly compared to the proposed C++/CUDA implementation.

It is not possible to compare the novel implementation to the state of the art GPU implementation [147]. This is because they use a different board and do not exploit the GPU for the *Collision Check* phase, making the overall execution times not comparable. Additionally, the authors in [147] split the *Path Generation* phase into three kernels: one for computing the paths, one for computing the cost, and one for converting the path points from *Frenet Coordinates* to *World coordinates*. They report the times of each kernel separately. In contrast, the novel implementation uses only one kernel to compute the paths, cost, and perform the coordinate conversion, in order to reduce the overhead of kernel launch. As a result, the reported times include all of these tasks. In conclusion, it is not possible to reproduce and compare their work with the novel proposed implementation as they do not share the implementation code and report times measured in a different way and on a different board.

The tests are performed on an NVIDIA Xavier AGX. This embedded board is equipped with a CUDA-capable GPU (512 NVIDIA CORE) and an ARM CPU (8 core). It has 32GB of memory.

Two types of measures were performed. The first measure considered the

	[137]	[139]	[147]	Novel proposal
Path Generation	CPU	CPU	GPU	GPU
Collision Check	CPU	CPU	no	GPU
Best path choice	CPU	CPU	CPU	GPU
Available code	no	yes	no	yes
Programming Language	-	Python	C++/CUDA	C++/CUDA

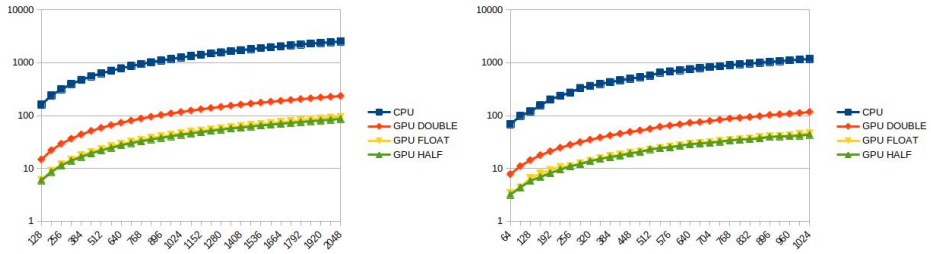
Table 7.1: Comparison of *Frenet Frame* based *Path Planners*.

overall time of the process, and the second measure considered distinct subparts. Specifically, in the latter case, the time of the *Path generation* phase and the time of the *Collision check* phase were measured separately. Since the novel implementation's kernel that computes the paths (Section 7.3.1) also performs cost computation and conversion of coordinates to *World coordinates*, it was compared to the union of these three tasks in the baseline implementation.

For each test, 100 iterations were performed, and the average time was reported in the plots. Two variants were performed, one varying the number of generated paths with a fixed path length, and one varying the path length with a fixed number of generated paths.

In Figure 7.6, the overall execution average time of two variants is presented: one where the number of generated paths is varied (Figure 7.6a) and another where the path length is varied (Figure 7.6b). The results indicate that the GPU implementation is faster than the CPU implementation. The precision type used also impacts the execution time, with the use of *double* precision resulting in longer execution times compared to the use of *float* or *half* precision. Although the execution times for the latter two types are similar, the use of *half* precision results in the lowest execution time. The trend for all implementations is linear, with the CPU implementation showing a steeper line. The first experiment (varying the number of generated paths) results in a constant speed-up of around 10x for *double* precision, 27x for *float* precision, and 28x for *half* precision (Figure 7.7a). In the second experiment (varying the path length), the speed-up increases since a path length of 288 and then reaches a plateau (Figure 7.7b).

The results of the execution time for the *Path generation* phase are consistent with those of the overall execution time. As shown in Figure 7.8a and Figure 7.8b, the GPU implementation is faster than the CPU implementa-

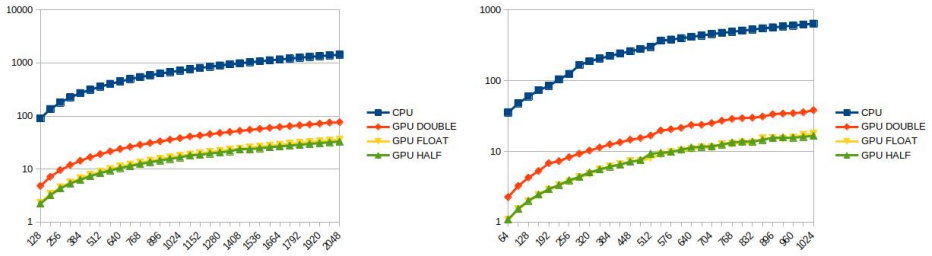


(a) Varying numbers of generated paths. (b) Varying the path length.

Figure 7.6: Overall Average Execution Time (ms) - X-axis represents the number of generated paths or the path length and the Y-axis represents the average execution time.

(a) Varying numbers of generated paths. (b) Varying the path length.

Figure 7.7: Overall speedup - X-axis represents the number of generated paths or the path length and the Y-axis represents the speedup.



(a) Varying numbers of generated paths. (b) Varying the path length.

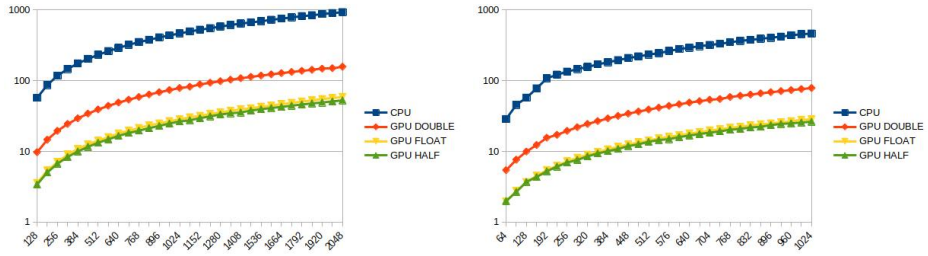
Figure 7.8: Average execution time (ms) of path generation phase - X-axis represents the number of generated paths or the path length and Y-axis represents the average execution time.

(a) Varying numbers of generated paths. (b) Varying the path length.

Figure 7.9: Path generation phase speedup - X-axis represents the number of generated paths or the path length and the Y-axis represents the speedup.

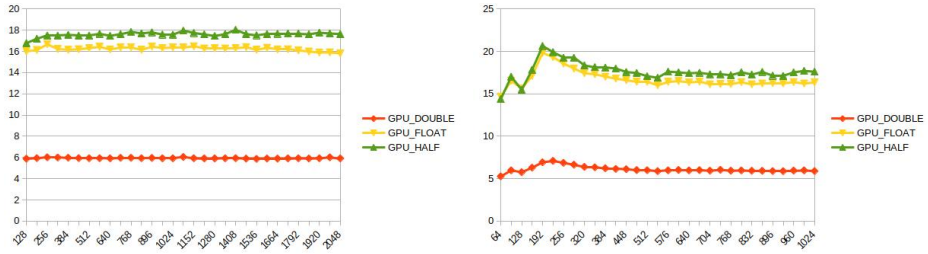
tion. The speed-up for this phase is also higher than the overall speed-up, with figures reaching 18x for *double* precision, 40x for *float* precision, and 46x for *half* precision when varying the number of generated paths, and reaching a plateau at a path length of 288 when varying the path length (as shown in Figure 7.9).

In regards to the *Collision Check* phase, the results show a similar trend as the overall and *Path Generation* phases. Figures 7.10a and 7.10b demonstrate the average execution time for this phase while varying the number of generated paths and the path length respectively. The speed-up for this phase is lower compared to the overall speed-up, as shown in Figure 7.11. When varying the number of generated paths, the speed-up is around 6x for double precision, 16x for float, and 17x for half precision. When varying the path length, the speed-up increases as the path length grows to 192, then



(a) Varying numbers of generated paths. (b) Varying the path length.

Figure 7.10: Average execution time (ms) of collision check phase - X-axis represents the number of generated paths or the path length and Y-axis represents the average execution time.



(a) Varying numbers of generated paths. (b) Varying the path length.

Figure 7.11: Collision check phase speedup - X-axis represents the number of generated paths or the path length and the Y-axis represents the speedup.

decreases a bit reaching a plateau near a path length of 544.

In Table 7.2, the percentage of execution time for the individual phases relative to the overall execution time is presented. It should be noted that other phases, such as the selection of the optimal path based on its cost, are also included in the overall execution time. The data shows that in the CPU implementation, the majority of execution time is spent on the *Path generation* phase, while in the GPU implementation, the highest percentage of execution time is allocated to the *Collision check* phase. The GPU implementation of the *Path generation* phase utilizes shared memory and minimizes critical operations, such as accessing global memory and branch divergence. Additionally, the integration of *World coordinates* conversion within the same kernel as the *Path generation* has reduced the overhead of kernel launch, leading to a significant reduction in execution time compared

	Path Generation	Collision Check	Other
CPU	56.9	36.9	6.2
GPU_DOUBLE	32.7	67.1	0.1
GPU_FLOAT	38.1	61.7	0.2
GPU_HALF	38.3	60.8	0.9

Table 7.2: Percentage of phases time over overall time.

to the CPU implementation. However, the kernel for the *Collision check* phase inherently includes branch divergence due to the differing instruction flow in cases of collision or non-collision and does not utilize shared memory. As a result, the reduction in execution time for this phase is not as significant compared to the *Path generation* phase. In conclusion, the reduction in time for the *Path Generation* phase is particularly significant due to the use of shared memory and minimizing critical operations. However, the reduction in time for the *Collision Check* phase is comparatively lower due to the intrinsic branch divergence and lack of shared memory exploitation. These observations are reflected in the percentage of execution time for each phase as shown in Table 7.2.

Typically, when a task is ported to a GPU, there is typically an overhead due to memory copies. In certain situations, such as on embedded boards, this overhead can be mitigated through the use of CUDA Unified Virtual Memory (UVM) or pinned memory [116, 117, 118]. However, this depends on the specific application. This overhead is included in the Other phases reported in Table 7.2. It should be noted that in the CPU implementation, this overhead is not present, and the Other phases are present due to different operations that are not memory copies, for example, the Other percentage includes the best path selection. The table shows that the percentage of Other phases is lower in GPU implementation. This is in contrast to what has just been said. This fact is due to two factors: first, the memory copies are minimized in the implementation, and by exploiting streams, they are done concurrently with the kernel computation. Second, as just said, the Other phases include also the computation of the best path, this is done by exploiting the CUDA *cublas* library and results in a better performance compared to the CPU implementation.

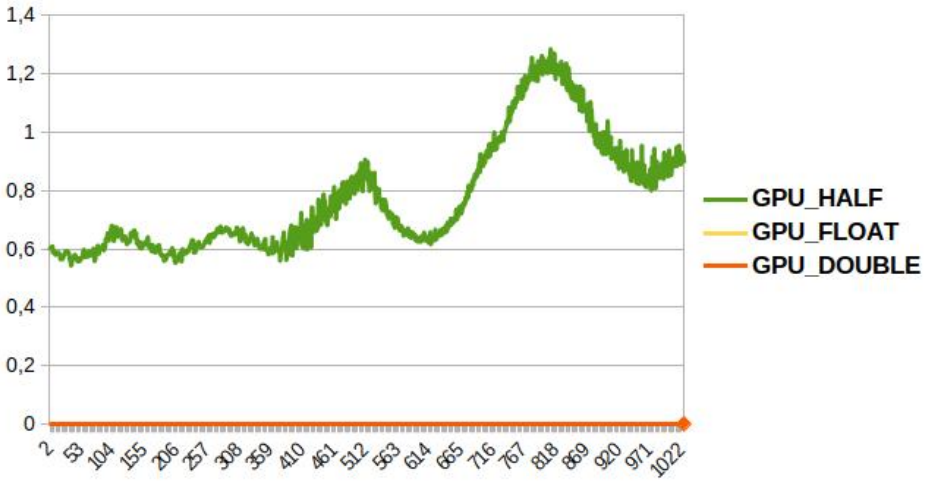
7.4.1 Precision error

In order to assess the impact of computation precision on the quality of the generated trajectories, a comparison was conducted between the trajectories generated by the CPU implementation (used as the baseline) and those generated by the GPU implementation with different precision types. The paths generated by the different precision types were compared while keeping the same parameters (D_s, D_{max}, D_{min}, V_s, V_{max}, V_{min}, V_{target}, T_s, T_{max}, T_{min}) and the *Frenet* algorithm was configured to generate 1024 different paths, each with 1024 path points. To obtain a larger number of paths to compare, a simulated vehicle was used which follows the selected path and generates a new path when it reaches a new location. This simulates the typical behavior of a vehicle that uses the *Frenet Path Planner*. The generation process was repeated 300 times and each point of each selected path was compared. The Average Trajectory Error (ATE) was used as the measure of comparison, which calculates the average displacement error of each point of the path, computed as the Euclidean Distance.

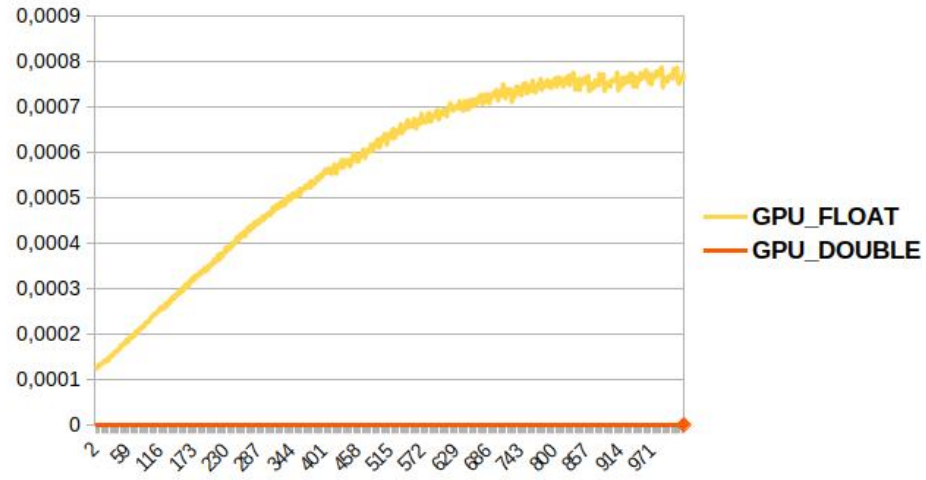
As expected that the ATE is 0m for *double* precision, indicating that the hardware (CPU or GPU) does not impact the result. In contrast, the use of *half* precision results in a larger error (0.7747m) due to the limited precision of the data type. However, the error in the *float* precision version is negligible (0.0005m). Furthermore, Figure 7.12 illustrates the errors for each point of the path, from 0 to 1024, for each of the 300 paths. It can be observed that the error within a trajectory increases as the vehicle moves further from the path starting point, indicating that the precision error tends to maximize towards the last points.

Given that the vehicle frequently re-computes its path based on its current position, it is unlikely that it will reach the final points of the initially generated path. This is an important consideration, as we have previously noted that precision errors tend to increase towards the final points of the path. To account for this, it is appropriate to measure the average error of the path traveled by the vehicle using the sequence of selected paths computed during the trip. By doing so, the Average Trajectory Error (ATE) of the trajectory of the simulated vehicle was obtained as follows: 0.5993m for *half*, 0.0001m for *float*, and always 0m for *double*.

It can be observed that the errors reported for *float* and *half* precision do not significantly impact the quality of the generated trajectory. In Figure 7.13, the generated trajectory can be seen in two scenarios: one involving a curve and one in the presence of obstacles. The trajectories are almost identical, but a closer examination reveals negligible differences (see

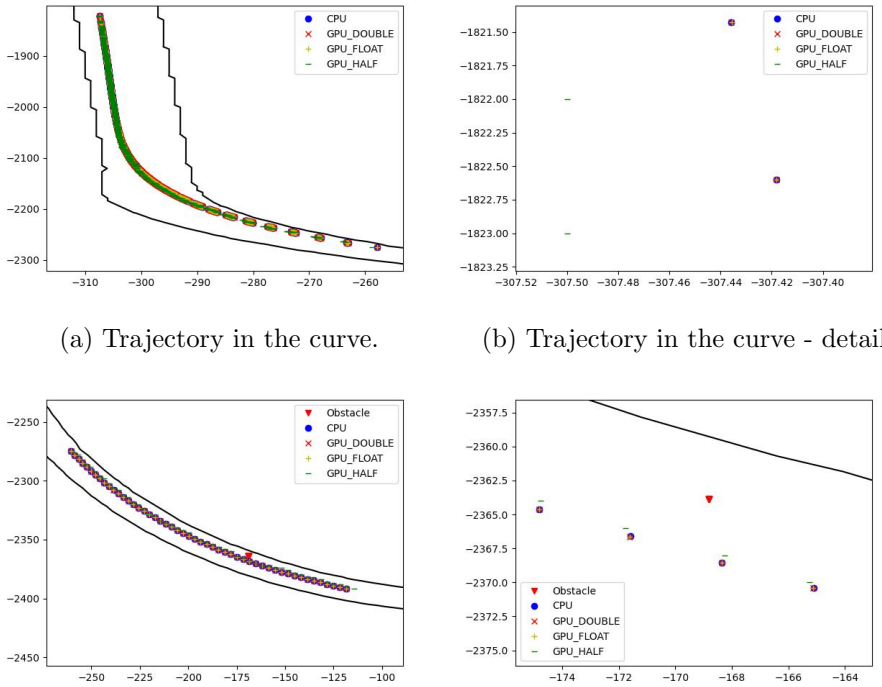


(a) All types



(b) Float and Double detail

Figure 7.12: Average Trajectory Error (ATE) in meters for each point of the path - X-axis represents the point number and Y-axis represents the ATE.



(a) Trajectory in the curve.

(b) Trajectory in the curve - detail.

(c) Trajectory with the obstacle.

(d) Trajectory with the obstacle - detail.

Figure 7.13: Generated trajectory: GPU with different data types vs. CPU.

figures 7.13 b and d). Given the small magnitude of the differences, it is evident that even in the *half* precision case, the trajectory is followed and the obstacle is avoided.

To conclude, by decreasing data precision in the GPU kernels we obtain noticeable performance improvements in terms of execution times. This comes at the cost of larger ATE values. The choice of precision within data types depends on the context. However, the experiment results clearly show how the *float* version represents the best choice, as it leads to an extremely low ATE compared to the huge execution time reduction (60% of time reduction with respect to *double* on GPU). The *half* version shows a higher ATE error towards the final points of generated trajectories, but the performance gain compared to the *float* version is not significant (only 4% of time reduction respect to *float* on GPU).

7.5 Conclusion

This chapter focuses on the problem of *planning* in autonomous vehicles, specifically addressing the *Local Planner* phase which is responsible for determining the most optimal trajectory for the vehicle to follow. The *Frenet Path Planner* algorithm is selected to address this problem, as its coordinate system facilitates the generation of a path based on various parameters. Additionally, since the algorithm generates multiple paths from which to choose the best, it is a suitable candidate for parallelization as it performs the same operation on different data. Furthermore, the *Collision Check* phase is also a valid candidate for parallelization for similar reasons.

In this chapter, a novel and optimized GPU implementation of the *Frenet Path Planner* algorithm is presented. To the best of current knowledge, there are no other existing implementations that have ported the entire pipeline to the GPU, making this the first implementation that fully implements the algorithm pipeline on GPU. The proposed implementation leverages CUDA streams to overlap memory copies and kernel execution. Additionally, it utilizes GPU *shared memory* and implements synchronization among threads within the same block, to compute the cost of the path. This is achieved through an optimized block construction, in which threads that compute points of the same trajectory share the same block. These enhancements contribute to a significant improvement in the execution time of the algorithm. In this study, the execution time of the proposed implementation was analyzed in comparison to a baseline CPU implementation, resulting in a speed-up of up to 28x. Additionally, the execution time was evaluated using different data types (i.e., *double*, *float*, *half*), and the impact on trajectory precision was also investigated. The results indicate that using float or half data types is advantageous as it leads to a 60% decrease in execution time compared to double data type, while the error in the trajectory is not significant enough to compromise the effectiveness of the algorithm.

The significant speed-up achieved makes the *Frenet Path Planner* algorithm more efficient for solving the *planning* problem, and more suitable for real-time critical scenarios, such as autonomous vehicles. The decreased execution time allows the algorithm to produce results faster, reducing the risk of safety hazards caused by a slow response that could result in an ineffective maneuver. The improved performance of the algorithm makes it more suitable for real-time scenarios where quick and accurate decision-making is crucial.

Chapter 8

Conclusion

In this thesis, two key areas of future mobility were explored: smart cities and Advanced Driver-Assistance Systems (ADAS) or autonomous vehicles.

The first area of focus was on traffic flow management in smart cities. Critical situations, such as Emergency Vehicle response, Parking Management, Intersection Management, and Traffic Light Management were examined and algorithms were proposed to manage these situations by utilizing the technologies and infrastructures present in smart cities, such as smart sensors, smart actuators, and communication capabilities. Additionally, emphasis was placed on simulating the proposed algorithms prior to deployment in the real city. This step was found to be crucial, as the results of the algorithm's decisions on the complex traffic flow system can be unpredictable and may pose potential safety risks for the city's inhabitants. By simulating the algorithms beforehand, potential problematic behaviors can be identified and addressed.

The proposed algorithms aim to optimize traffic flow in a smart city context by addressing critical situations such as the aforementioned Emergency Vehicle response, Parking Management, Intersection, and Traffic Light Management. These algorithms leverage the infrastructures present in smart cities to reduce waiting times at intersections and traffic lights, reserve parking spots, and improve the response time of emergency vehicles in case of accidents.

It is noteworthy that the proposed algorithms, while specifically tailored to facilitate *coordination* among connected vehicles, are also designed to account for the presence of non-connected vehicles in the traffic flow. This ensures that the algorithms can effectively manage traffic flow in a dynamic

environment, irrespective of the level of connectivity among the vehicles present in the city.

In particular, with regard to Traffic Lights Management problem, a novel auction-based system was proposed and its effectiveness was investigated in comparison to the standard Fixed-Time traffic light policy (FTC). The aim of the proposed system was to reduce the waiting time at intersections and achieve differentiated latencies. It was determined that there are advantages for vehicles that participate in the proposed system, as the average waiting time at traffic lights is reduced and the overall trip time is decreased with respect to FTC policy. However, it was also observed that the system was not able to attain differentiated latencies based on vehicle budgets. Further research is needed to determine whether the limitations in achieving differentiated latencies are due to the experimental settings or if the system inherently does not allow for them.

With regard to Intersection Management, a comprehensive study was conducted, focusing on the examination of various systems that utilize auction-based approaches. These systems were evaluated and compared in terms of their effectiveness in coordinating vehicles at intersections. The study also included an analysis of the standard yield rule method for coordinating vehicles at intersections, providing a point of comparison for the auction-based systems. The study aimed to provide valuable insights into the effectiveness and potential of auction-based approaches for managing and coordinating the movement of vehicles at intersections The co-existence of human-driven and autonomous vehicles was taken into account in the analysis of the proposed vehicle *coordination* strategies. It was determined that this aspect plays a crucial role in the design of such policies. The presence of human drivers in traffic poses unique challenges and limitations that must be taken into account when designing *coordination* policies. It was observed that human drivers are not able to follow complex rules in the same way that autonomous vehicles would, which precludes the adoption of many solutions that can be exploited when vehicles are exclusively autonomous. This highlights the importance of considering the co-existence of human-driven and autonomous vehicles in the development of effective *coordination* policies. The study focused on investigating the effectiveness of the proposed auction-based mechanisms for Intersection Management. These mechanisms have been previously shown to bring several benefits, such as reductions in latencies and the ability to differentiate latencies among vehicles according to their needs. However, the results of the study were not favorable, as the

experiments and the considered scenario did not yield any of these benefits when using auctions. This was primarily due to the limitations in design introduced by the presence of human drivers, which frustrated the potential benefits of the auction system. The study concludes that further research is needed to explore the potential benefits of auction-based mechanisms in a mixed traffic environment where human-driven and autonomous vehicles co-exist. In order to further understand the results of this study and to identify potential areas for improvement, future work will focus on investigating the motivations behind the results. This will include an examination of whether the results are primarily dependent on the specific scenario that was considered, or whether the strategies employed were too simplistic. Additionally, the study will investigate the impact of vehicles being necessarily serialized when in lanes on the effectiveness of the auction-based mechanisms. Subsequently, alternative *coordination* policies for the scenario of human-driven and autonomous vehicles co-existing will be studied.

As the results of Traffic Light Management were encouraging, it will be worth exploring if the difference in management leads to different results. In particular, it will be interesting to investigate if the longer green phase duration in traffic light management allows for more vehicles in the lane to pass, as opposed to the proposed Intersection Management where only one vehicle per lane can pass from one auction to the next. This could potentially be a source of degradation in throughput. Moreover, more experiments will be performed varying the number of vehicles present in the scenario in order to see the variation of the results at different system utilization. More complex algorithms can be designed, for example, traffic lights can consider, in addition to the auction, the load of other links to prevent congestion by denying green lights to those vehicles that would reach links close to saturation. Finally, The benefit of Smart Traffic Light systems can be measured not only against the FTC but also against other systems adopted in some cities, such as systems that sense when a lane is empty and skip the green light phase for this lane.

The study also examined the Parking Management problem in a smart city context, with a particular emphasis on addressing the co-existence of connected and non-connected vehicles. A system was proposed to effectively manage both types of vehicles in the context of parking management. The proposed approach was validated and benchmarked through the use of extensive simulations. The study aims to provide a solution that addresses the challenges associated with managing the co-existence of connected and non-connected vehicles in a smart city context, specifically in the context of

Parking Management. The study found that managing the co-existence of connected and non-connected vehicles in a smart city context is a complex task and the presence of non-connected vehicles has a significant impact on system design. Additionally, the presence of *traditional* vehicles also affects the performance of the system. For instance, a parking spot can be chosen by a connected vehicle, and then it can potentially be occupied by a non-connected vehicle that is unaware that the spot has already been selected by another vehicle. This situation does not occur in the scenario where only connected vehicles are present since connected vehicles are informed about which parking spots are free and which are already taken. Therefore, it is important to simulate the co-existence of these two types of vehicles to fully understand the challenges and design effective solutions. However, the proposed smart system is able to reduce the parking search time by up to 76% for the majority of vehicles that adopt it with respect to baseline strategies utilized by human drivers. This is a good incentive for *traditional* vehicles to switch to the smart system. In light of these findings, more sophisticated systems can be designed for Parking Management. For example, incorporating traffic information into the Parking Management system can improve its efficiency and effectiveness by suggesting the best parking spot for a vehicle based on not only the availability of parking spots but also the traffic on the surrounding roads and the likelihood of the spot being taken by another vehicle. This can improve the overall performance of the system.

With regards to Emergency Vehicles Management, the study aimed to investigate Emergency Vehicle Management in the context of a Smart City. To achieve this, the MATSim urban traffic simulator was extended to enable the testing and experimentation of different viability models in case of traffic accidents. While previous research on traffic congestion mitigation techniques [150, 151] has assumed that all road users can be informed of traffic disruptions in order to immediately reroute and avoid congestion, the proposed mechanism has a two-fold objective. The first is to increase the possibility of vehicles avoiding roads with accidents and rerouting before reaching them. The second is to decrease the emergency vehicle response time. To achieve this goal, the proposed system communicates with non-emergency vehicles about the specific roads that will be traversed by emergency vehicles in order to minimize congestion on the emergency vehicle's route. The non-emergency vehicles, upon receiving this information, are able to adjust their routes accordingly to avoid these roads and thus contribute to reducing congestion on the emergency vehicle's route. The study examined the impact of different types of agents with varying communication capabilities. The proposed mechanisms were able to significantly reduce emergency

vehicle response time by up to 36.7% with respect to a scenario in which no such mechanisms exist. This is of paramount importance as it increases the chance of survival for those involved in road accidents [152, 153]. Additionally, the proposed mechanism also resulted in a reduction in travel time for non-emergency vehicles as they are able to avoid roads with accidents, resulting in a decrease in trip time by 36.1% compared to the trip time of the same vehicle that does not join the proposed mechanism.

The aforementioned challenges were addressed through the use of two distinct types of algorithm design: local and global. The local design is a distributed approach in which there is no central authority, while the global design is a centralized approach where a central server possesses knowledge of the entire state of the city. Both the local and global algorithm designs have their own distinct advantages and disadvantages. It was observed that the global design, due to the central server's ability to manage the entire city, can provide vehicles with more comprehensive information and enable decision-making based on the overall state of the city. However, the central server also serves as a single point of failure, meaning that if there is an issue with the server or its connection, the entire system may become inoperable. The local design, on the other hand, does not have a single point of failure. However, the decision-making of the controller is based solely on local information, such as the status of streets and vehicles in the immediate vicinity. This can limit the decision-making capabilities of the controller. The study has demonstrated that Parking and Emergency Vehicle Management can benefit from a comprehensive view of the entire city, as it is important to have knowledge of the parking or traffic state of all areas to make informed decisions. On the other hand, Traffic Light or Intersection management can be designed as a local algorithm, as decisions can be made based on the status of the involved streets alone, thus avoiding the issue of a single point of failure.

In future research in this field, it would be useful to investigate whether a global design for Traffic Light and Intersection Management can improve system decision-making, as knowledge of the overall state of the city could facilitate the coordination of Traffic Lights. Additionally, it would be worth exploring if local management of parking and emergency vehicle management can be designed to mitigate the issue of a single point of failure.

When looking at the individual vehicle, i.e. ADAS and autonomous vehicles, the focus was on computation time. Two tasks of the *perception* and *control* pipeline were considered: *localization* and *planning*. For the *local-*

ization part, the well-known ORB-SLAM algorithm was described, which uses camera data to estimate the vehicle pose. For the *planning* part, the *Frenet Path Planner* algorithm was described.

Different implementations of these two algorithms using GPU acceleration have been proposed. The importance of fast execution is highlighted and comparisons are made with the baseline implementation of the algorithms, both in terms of execution time and precision of results.

The proposed implementation of ORB-SLAM uses CUDA *streams* and *events* to manage concurrency between kernels execution and memory copies. Moreover, the tasks inside the *ORB Extraction* phase have been parallelized on the GPU. In addition, a novel method for the *Pyramid* construction task and a novel cluster-based point filter method, more suitable for GPU parallelization, have been proposed. The combination of these improvements results in a speedup of up to 3x over the original baseline CPU-based implementation, as measured using the standard datasets in the *localization* field. As future research, It is feasible to study the capabilities of the Unified Memory feature on NVIDIA embedded boards and evaluate the potential impact of deploying it on a discrete GPU in terms of memory copies. It is also possible to quantify the energy consumption of the novel implementation and to create optimized GPU versions for the remaining algorithmic steps associated with SLAM-based approaches, such as the *Loop Closing* detection mechanism.

The proposed implementation *Frenet Path Planner* uses the CUDA *streams* to overlap memory copies and kernel execution. In addition, the *shared memory* and an optimal grid organization were designed to reduce access to global memory. Finally, the computation was tested using different floating-point precision types: *double*, *float*, and *half*. They were tested in the same condition to verify the reduction in execution time of the algorithm and the impact on the precision of the generated trajectories. The tests, performed in the simulator, show that the precision is not much affected (especially for *float* and *double*), while the execution time is consistently reduced, with a speed-up of up to 28x compared to the baseline sequential CPU-based implementation. In future work, it is possible to investigate, also in this case, whether the use of CUDA Unified Virtual Memory (UVM) instead of explicit memory copies can reduce the overhead of data transfers, since, especially on embedded boards, the use of one approach rather than another can reduce the overall execution time, but it depends on the specific application [116, 117, 118]. It is also possible to integrate the *Frenet Path*

Planner with a trajectory prediction for the surrounding vehicles to deal with dynamic obstacles.

In conclusion, the potential of smart cities and autonomous vehicles can be exploited to improve the quality of life of the city and its drivers. It is important to consider the co-existence of other non-autonomous and non-equipped vehicles, as the algorithm's design must take into account their unpredictability. In addition, the safety and consequently the execution time of the algorithms is a crucial points to consider. In order to avoid safety-threatening situations in the real scenario, it is a good practice to test the algorithms in a simulated scenario, where it is possible to see unexpected behaviors.

Bibliography

[1] N. Otterness, M. Yang, S. Rust, E. Park, J. H. Anderson, F. D. Smith, A. Berg, and S. Wang, "An evaluation of the nvidia tx1 for supporting real-time computer-vision workloads," in *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 353–364, IEEE, 2017.

[2] A. Bujari, M. Furini, F. Mandreoli, R. Martoglia, M. Montangero, and D. Ronzani, "Standards, security and business models: Key challenges for the iot scenario," *MONET*, vol. 23, no. 1, pp. 147–154, 2018.

[3] M. Furini, F. Mandreoli, R. Martoglia, and M. Montangero, "Iot: Science fiction or real revolution?," in *Proc. GOODTECHS 2016*, pp. 96–105, Springer International Publishing, 2017.

[4] A. Melis, S. Mirri, C. Prandi, M. Prandini, P. Salomoni, and F. Callegati, "Crowdsensing for smart mobility through a service-oriented architecture," in *2016 IEEE Int. Smart Cities Conference (ISC2)*, pp. 1–2, IEEE, 2016.

[5] M. Bertogna, P. Burgio, G. Cabri, and N. Capodieci, "Adaptive coordination in autonomous driving: motivations and perspectives," in *2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pp. 15–17, IEEE, 2017.

[6] A. Bujari, M. Ciman, O. Gaggi, and C. E. Palazzi, "Using gamification to discover cultural heritage locations from geo-tagged photos," *Personal and Ubiquitous Computing*, vol. 21, no. 2, pp. 235–252, 2017.

[7] A. Bujari, O. Gaggi, C. E. Palazzi, and D. Ronzani, "Would current ad-hoc routing protocols be adequate for the internet of vehicles? A comparative study," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3683–3691, 2018.

[8] K. W Axhausen, A. Horni, and K. Nagel, *The multi-agent transport simulation MATSim*. Ubiquity Press, 2016.

[9] J. Ferber and G. Weiss, *Multi-agent systems: an introduction to distributed artificial intelligence*, vol. 1. Addison-Wesley Reading, 1999.

[10] M. Maciejewski and K. Nagel, "Towards multi-agent simulation of the dynamic vehicle routing problem in matsim," in *International Conference on Parallel Processing and Applied Mathematics*, pp. 551–560, Springer, 2012.

[11] D. Grether, *Extension of a multi-agent transport simulation for traffic signal control and air transport systems*. PhD thesis, Technical University Berlin, 2014.

[12] D. Grether and A. Neumann, "Traffic light control in multi-agent transport simulations," in *Technical report, Transport Systems Planning and Transport Telematics*, Technical University Berlin, 2011.

[13] J. Bischoff and K. Nagel, "Integrating explicit parking search into a transport simulation," *Procedia computer science*, vol. 109, pp. 881–886, 2017.

[14] R. A. Waraich, *Agent-based simulation of electric vehicles: design and implementation of a framework*. PhD thesis, ETH Zurich, 2013.

[15] H. Prothmann, F. Rochner, S. Tomforde, J. Branke, C. Müller-Schloer, and H. Schmeck, "Organic control of traffic lights," in *Autonomic and Trusted Computing*, (Berlin, Heidelberg), pp. 219–233, Springer Berlin Heidelberg, 2008.

[16] G. Cabri, M. Montangero, F. Muzzini, and P. Valente, "Managing human-driven and autonomous vehicles at smart intersections," in *2020 IEEE International Conference on Human-Machine Systems (ICHMS)*, pp. 1–4, IEEE, 2020.

[17] F. Muzzini, N. Capodieci, and M. Montangero, "Exploiting traffic lights to manage auction-based crossings," in *Proceedings of the 6th EAI International Conference on Smart Objects and Technologies for Social Good*, GoodTechs '20, (New York, NY, USA), pp. 199–204, Association for Computing Machinery, 2020.

[18] M. O. Sayin, C.-W. Lin, S. Shiraishi, J. Shen, and T. Basar, "Information-driven autonomous intersection control via incentive compatible mechanisms," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, pp. 912–924, mar 2019.

[19] J. Raphael, S. Maskell, and E. Sklar, "From goods to traffic: First steps toward an auction-based traffic signal controller," in *Advances in Practical Applications of Agents, Multi-Agent Systems, and Sustainability: The PAAMS Collection*, pp. 187–198, Springer International Publishing, 2015.

[20] J. Raphael, E. I. Sklar, and S. Maskell, "An intersection-centric auction-based traffic signal control framework," in *Understanding Complex Systems*, pp. 121–142, Springer International Publishing, 2017.

[21] M. Covell, S. Baluja, and R. Sukthankar, "Micro-auction-based traffic-light control: Responsive, local decision making," in *International Conference on Intelligent Transportation Systems*, 2015.

[22] I. K. Isukapati and G. F. List, "Agent based framework for modeling operations at isolated signalized intersections," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, IEEE, sep 2015.

[23] I. K. Isukapati and S. F. Smith, "Accommodating high value-of-time drivers in market-driven traffic signal control," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1280–1286, IEEE, 2017.

[24] M. Mashayekhi and G. List, "A multiagent auction-based approach for modeling of signalized intersections," in *IJCAI workshops on synergies between multiagent systems, machine learning and complex systems*, pp. 13–24, 2015.

[25] C. Vilarinho, J. P. Tavares, and R. J. Rossetti, "Intelligent traffic lights: Green time period negotiaton," *Transportation research procedia*, vol. 22, pp. 325–334, 2017.

[26] D. Rey, M. W. Levin, and V. V. Dixit, "Online incentive-compatible mechanisms for traffic intersection auctions," *European Journal of Operational Research*, vol. 293, no. 1, pp. 229–247, 2021.

[27] D. Lin and S. E. Jabari, "Comparative analysis of economic instruments in intersection operation: A user-based perspective," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–6, IEEE, 2020.

[28] K. M. Dresner and P. Stone, "Sharing the road: Autonomous vehicles meet human drivers.," in *Ijcai*, vol. 7, pp. 1263–1268, 2007.

[29] C. Iliopoulou, E. Kampitakis, K. Kepaptsoglou, and E. I. Vlahogianni, "Dynamic traffic-aware auction-based signal control under vehicle to infrastructure communication," *Physica A: Statistical Mechanics and its Applications*, p. 128258, 2022.

[30] W. Chen, R. K. Guha, T. J. Kwon, J. Lee, and Y.-Y. Hsu, "A survey and challenges in routing and data dissemination in vehicular ad hoc networks," *Wireless Communications and Mobile Computing*, vol. 11, no. 7, pp. 787–795, 2011.

[31] J. Rios-Torres and A. A. Malikopoulos, "A survey on the coordination of connected and automated vehicles at intersections and merging at highway on-ramps," *IEEE Trans. on Intelligent Transportation Systems*, vol. 18, no. 5, pp. 1066–1077, 2017.

[32] D. Carlino, S. D. Boyles, and P. Stone, "Auction-based autonomous intersection management," in *Int. IEEE Conf. on Intelligent Transportation Systems*, pp. 529–534, IEEE, 2013.

[33] D. Bergemann and M. Said, "Dynamic auctions," *Wiley Encyclopedia of Operations Research and Management Science*, 2010.

[34] N. Capodieci, G. A. Pagani, G. Cabri, and M. Aiello, "Smart meter aware domestic energy trading agents," in *Proc. of the 2011 Workshop on E-energy Market Challenge*, IEEMC '11, (New York, NY, USA), pp. 1–10, ACM, 2011.

[35] H. Schepperle and K. Böhm, "Agent-based traffic control using auctions," in *Int. Workshop on Cooperative Information Agents*, pp. 119–133, Springer, 2007.

[36] W. Vickrey, "Counterspeculation, auctions, and competitive sealed tenders," *The Journal of finance*, vol. 16, no. 1, pp. 8–37, 1961.

[37] M. Vasirani and S. Ossowski, "A market-inspired approach for intersection management in urban road traffic networks," *J. of Artificial Intelligence Research*, vol. 43, pp. 621–659, 2012.

[38] K. Dresner and P. Stone, "A multiagent approach to autonomous intersection management," *Journal of artificial intelligence research*, vol. 31, pp. 591–656, 2008.

[39] G. Revathi and V. R. S. Dhulipala, "Smart parking systems and sensors: A survey," in *Proc. International Conference on Computing, Communication and Applications*, pp. pp. 1–5, 2012.

[40] B. R. Alves, G. V. Alves, A. P. Borges, and P. Leitão, "Experimentation of negotiation protocols for consensus problems in smart parking systems," in *International Conference on Industrial Applications of Holonic and Multi-Agent Systems*, pp. 189–202, Springer, 2019.

[41] S.-Y. Chou, S.-W. Lin, and C.-C. Li, "Dynamic parking negotiation and guidance using an agent-based platform," *Expert Systems with Applications*, vol. 35, no. 3, pp. 805–817, 2008.

[42] H. Billhardt, A. Fernández, M. Lujak, S. Ossowski, V. Julián, J. F. De Paz, and J. Z. Hernández, "Towards smart open dynamic fleets," in *Multi-Agent Systems and Agreement Technologies* (M. Rovatsos, G. Vouros, and V. Julian, eds.), (Cham), pp. 410–424, Springer International Publishing, 2016.

[43] G. Cabri, L. Gherardini, M. Montangero, and F. Muzzini, "About auction strategies for intersection management when human-driven and autonomous vehicles coexist," *Multimedia Tools and Applications*, pp. 1–16, 2021.

[44] G. Cabri, L. Gherardini, and M. Montangero, "Auction-based crossings management," in *Proceedings of the 5th EAI International Conference on Smart Objects and Technologies for Social Good*, GoodTechs '19, (New York, NY, USA), pp. 183–188, ACM, 2019.

[45] A. Horni, K. Nagel, and K. W. Axhausen, *The multi-agent transport simulation MATSim*. Ubiquity Press London, 2016.

[46] S. Mariani, G. Cabri, and F. Zambonelli, "Coordination of autonomous vehicles: taxonomy and survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 1, pp. 1–33, 2021.

[47] A. Khanna and R. Anand, "Iot based smart parking system," in *2016 International Conference on Internet of Things and Applications (IOTA)*, pp. 266–270, IEEE, 2016.

[48] P. Sadhukhan, "An iot-based e-parking system for smart cities," in *2017 International conference on advances in computing, communications and informatics (ICACCI)*, pp. 1062–1066, IEEE, 2017.

[49] L. Mainetti, L. Patrono, M. L. Stefanizzi, and R. Vergallo, "A smart parking system based on iot protocols and emerging enabling technologies," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pp. 764–769, IEEE, 2015.

[50] M. G. Diaz Ogás, R. Fabregat, and S. Aciar, "Survey of smart parking systems," *Applied Sciences*, vol. 10, no. 11, 2020.

[51] A. O. Kotb, Y.-c. Shen, and Y. Huang, "Smart parking guidance, monitoring and reservations: A review," *IEEE Intelligent Transportation Systems Magazine*, vol. 9, no. 2, pp. 6–16, 2017.

[52] T. N. Pham, M.-F. Tsai, D. B. Nguyen, C.-R. Dow, and D.-J. Deng, "A cloud-based smart-parking system based on internet-of-things technologies," *IEEE Access*, vol. 3, pp. 1581–1591, 2015.

[53] F. Muzzini, N. Capodieci, and M. Montangero, "Smart parking for all: equipped and non-equipped vehicles in smart cities," in *in Proc. Intelligent Distributed Computing Symposium,* to appear, 2022.

[54] A. Agarwal and G. Lämmel, "Modeling seepage behavior of smaller vehicles in mixed traffic conditions using an agent based simulation," *Transportation in Developing Economies*, vol. 2, no. 2, p. 8, 2016.

[55] I. Benenson, K. Martens, and S. Birfir, "Parkagent: An agent-based model of parking in the city," *Computers, Environment and Urban Systems*, vol. 32, no. 6, pp. 431–439, 2008. GeoComputation: Modeling with spatial agents.

[56] T. Toledo, M. E. Ben-Akiva, D. Darda, M. Jha, and H. N. Koutsopoulos, "Calibration of microscopic traffic simulation models with aggregate data," *Transportation Research Record*, vol. 1876, no. 1, pp. 10–19, 2004.

[57] R. Balakrishna, C. Antoniou, M. Ben-Akiva, H. N. Koutsopoulos, and Y. Wen, "Calibration of microscopic traffic simulation models: Methods and application," *Transportation Research Record*, vol. 1999, no. 1, pp. 198–207, 2007.

[58] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone, "Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data," in *European Conference on Computer Vision*, pp. 683–700, Springer, 2020.

[59] N. Sriram, B. Liu, F. Pittaluga, and M. Chandraker, "Smart: Simultaneous multi-agent recurrent trajectory prediction," in *European Conference on Computer Vision*, pp. 463–479, Springer, 2020.

[60] A. Kamenev, L. Wang, O. B. Bohan, I. Kulkarni, B. Kartal, A. Molchanov, S. Birchfield, D. Nistér, and N. Smolyanskiy, "Predictionnet: Real-time joint probabilistic traffic prediction for planning, control, and simulation," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 8936–8942, IEEE, 2022.

[61] R. Schubert, E. Richter, and G. Wanielik, "Comparison and evaluation of advanced motion models for vehicle tracking," in *2008 11th international conference on information fusion*, pp. 1–6, IEEE, 2008.

[62] D. Greene, J. Liu, J. Reich, Y. Hirokawa, A. Shinagawa, H. Ito, and T. Mikami, "An efficient computational architecture for a collision early-warning system for vehicles, pedestrians, and bicyclists," *IEEE Transactions on intelligent transportation systems*, vol. 12, no. 4, pp. 942–953, 2011.

[63] C. Hermes, C. Wohler, K. Schenk, and F. Kummert, "Long-term vehicle motion prediction," in *2009 IEEE intelligent vehicles symposium*, pp. 652–657, IEEE, 2009.

[64] J. Levinson, M. Montemerlo, and S. Thrun, "Map-based precision vehicle localization in urban environments.," in *Robotics: science and systems*, vol. 4, p. 1, Atlanta, GA, USA, 2007.

[65] P. Merriaux, Y. Dupuis, P. Vasseur, and X. Savatier, "Wheel odometry-based car localization and tracking on vectorial map," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pp. 1890–1891, IEEE, 2014.

[66] J. Aulinas, Y. Petillot, J. Salvi, and X. Lladó, "The slam problem: a survey," *Artificial Intelligence Research and Development*, pp. 363–371, 2008.

[67] A. Macario Barros, M. Michel, Y. Moline, G. Corre, and F. Carrel, "A comprehensive survey of visual slam algorithms," *Robotics*, vol. 11, no. 1, p. 24, 2022.

[68] H. He and B. Upcroft, "Nonparametric semantic segmentation for 3d street scenes," in *2013 IEEE/RSJ international conference on intelligent robots and systems*, pp. 3697–3703, IEEE, 2013.

[69] P. Panchal, G. Prajapati, S. Patel, H. Shah, and J. Nasriwala, "A review on object detection and tracking methods," *International Journal for Research in Emerging Science and Technology*, vol. 2, no. 1, pp. 7–12, 2015.

[70] N. Capodieci, R. Cavicchioli, and A. Marongiu, "A taxonomy of modern gpgpu programming methods: on the benefits of a unified specification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 6, pp. 1649–1662, 2021.

[71] S. Jung, S. Song, P. Youn, and H. Myung, "Multi-layer coverage path planner for autonomous structural inspection of high-rise structures," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–9, IEEE, 2018.

[72] E. Jeong, J. Seo, and J. Wacker, "Literature review and technical survey on bridge inspection using unmanned aerial vehicles," *Journal of Performance of Constructed Facilities*, vol. 34, no. 6, p. 04020113, 2020.

[73] S. Jung, H. Cho, D. Kim, K. Kim, J.-I. Han, and H. Myung, "Development of algal bloom removal system using unmanned aerial vehicle and surface vehicle," *IEEE Access*, vol. 5, pp. 22166–22176, 2017.

[74] H. Kim, J. Koo, D. Kim, S. Jung, J.-U. Shin, S. Lee, and H. Myung, "Image-based monitoring of jellyfish using deep learning architecture," *IEEE sensors journal*, vol. 16, no. 8, pp. 2215–2216, 2016.

[75] J. Scherer and B. Rinner, "Multi-uav surveillance with minimum information idleness and latency constraints," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4812–4819, 2020.

[76] R. Hussain and S. Zeadally, "Autonomous cars: Research results, issues, and future challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1275–1313, 2018.

[77] A. W. Obenauf, R. R. Souleyrette, R. M. Kluger, and A. Pratelli, "Impact of self-driving and connected vehicles on emergency response: The case of the usa and implications for italy," *WIT Transactions on The Built Environment*, vol. 189, pp. 101–112, 2019.

[78] A. Sahoo, S. K. Dwivedy, and P. Robi, "Advancements in the field of autonomous underwater vehicle," *Ocean Engineering*, vol. 181, pp. 145–160, 2019.

[79] Z. Peng, J. Wang, D. Wang, and Q.-L. Han, "An overview of recent advances in coordinated control of multiple autonomous surface vehicles," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 2, pp. 732–745, 2020.

[80] N. Tripathi and S. Yogamani, "Trained trajectory based automated parking system using visual slam on surround view cameras," *arXiv preprint arXiv:2001.02161*, 2020.

[81] M. Kegeleirs, G. Grisetti, and M. Birattari, "Swarm slam: Challenges and perspectives," *Frontiers in Robotics and AI*, vol. 8, p. 618268, 2021.

[82] T. T. O. Takleh, N. A. Bakar, S. A. Rahman, R. Hamzah, and Z. Aziz, "A brief survey on slam methods in autonomous vehicle," *International Journal of Engineering & Technology*, vol. 7, no. 4, pp. 38–43, 2018.

[83] K. Ebadi, L. Bernreiter, H. Biggie, G. Catt, Y. Chang, A. Chatterjee, C. E. Denniston, S.-P. Deschênes, K. Harlow, S. Khattak, *et al.*, "Present and future of slam in extreme underground environments," *arXiv preprint arXiv:2208.01787*, 2022.

[84] X. Zhang, J. Lai, D. Xu, H. Li, and M. Fu, "2d lidar-based slam and path planning for indoor rescue using mobile robots," *Journal of Advanced Transportation*, vol. 2020, 2020.

[85] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual slam algorithms: A survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, no. 1, pp. 1–11, 2017.

[86] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European conference on computer vision*, pp. 834–849, Springer, 2014.

[87] J. Engel, J. Stückler, and D. Cremers, "Large-scale direct slam with stereo cameras," in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 1935–1942, IEEE, 2015.

[88] J. Zubizarreta, I. Aguinaga, and J. M. M. Montiel, "Direct sparse mapping," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1363–1370, 2020.

[89] A. J. Davison, "Real-time simultaneous localisation and mapping with a single camera," in *Computer Vision, IEEE International Conference on*, vol. 3, pp. 1403–1403, IEEE Computer Society, 2003.

[90] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[91] H. Strasdat, J. M. Montiel, and A. J. Davison, "Visual slam: why filter?," *Image and Vision Computing*, vol. 30, no. 2, pp. 65–77, 2012.

[92] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *2007 6th IEEE and ACM international symposium on mixed and augmented reality*, pp. 225–234, IEEE, 2007.

[93] R. Mur-Artal, J. M. M. Montiel, and J. D. Tard'os, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[94] R. Mur-Artal and J. D. Tard'os, "ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[95] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021.

[96] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[97] M. Smith, I. Baldwin, W. Churchill, R. Paul, and P. Newman, "The new college vision and laser data set," *The International Journal of Robotics Research*, vol. 28, no. 5, pp. 595–599, 2009.

[98] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, 2016.

[99] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[100] Z. Huang, Y. Chen, and Y. Jiang, "Orb-slam2 gpu optimization," 2016.

[101] Z. Li, L. Zhang, C. Xu, F. Zou, J. Song, and D. Tian, "An acceleration method using cuda based on orb-slam2," in *2019 IEEE 9th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, pp. 1441–1446, 2019.

[102] J. Jeon, S. Jung, E. Lee, D. Choi, and H. Myung, "Run your visual-inertial odometry on nvidia jetson: Benchmark tests on a micro aerial vehicle," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5332–5339, 2021.

[103] Mamri, Ayoub, Abouzahir, Mohamed, Ramzi, Mustapha, and Latif, Rachid, "Orb-slam accelerated on heterogeneous parallel architectures," *E3S Web Conf.*, vol. 229, p. 01055, 2021.

[104] S. Aldegheri, N. Bombieri, D. D. Bloisi, and A. Farinelli, "Data flow orb-slam for real-time performance on embedded gpu boards," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5370–5375, 2019.

[105] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, pp. 2564–2571, 2011.

[106] H. Strasdat, A. J. Davison, J. M. Montiel, and K. Konolige, "Double window optimisation for constant time visual slam," in *2011 international conference on computer vision*, pp. 2352–2359, IEEE, 2011.

[107] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment—a modern synthesis," in *International workshop on vision algorithms*, pp. 298–372, Springer, 1999.

[108] M. Strengert, M. Kraus, and T. Ertl, "Pyramid methods in gpu-based image processing," *Proceedings vision, modeling, and visualization 2006*, pp. 169–176, 2006.

[109] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), (Berlin, Heidelberg), pp. 430–443, Springer Berlin Heidelberg, 2006.

[110] S. Peters, "Quadtree- and octree-based approach for point data selection in 2d or 3d," *Annals of GIS*, vol. 19, no. 1, pp. 37–44, 2013.

[111] P. L. Rosin, "Measuring corner properties," *Computer Vision and Image Understanding*, vol. 73, no. 2, pp. 291–307, 1999.

[112] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Computer Vision – ECCV 2010* (K. Daniilidis, P. Maragos, and N. Paragios, eds.), (Berlin, Heidelberg), pp. 778–792, Springer Berlin Heidelberg, 2010.

[113] F. Muzzini, N. Capodieci, R. Cavicchioli, and B. Rouxel, "Brief announcement: Optimized gpu-accelerated feature extraction for orb-slam," in *in Proc. Symposium on Parallelism in Algorithms and Architectures,* to appear, 2023.

[114] T. Karras, "Maximizing parallelism in the construction of bvhs, octrees, and k-d trees," in *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics*, pp. 33–37, 2012.

[115] P. Ajmera, R. Goradia, S. Chandran, and S. Aluru, "Fast, parallel, gpu-based construction of space filling curves and octrees," in *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pp. 1–1, 2008.

[116] Y. Gu, W. Wu, Y. Li, and L. Chen, "Uvmbench: A comprehensive benchmark suite for researching unified virtual memory in gpus," *arXiv preprint arXiv:2007.09822*, 2020.

[117] J. Choi, H. You, C. Kim, H. Young Yeom, and Y. Kim, "Comparing unified, pinned, and host/device memory allocations for memory-intensive workloads on tegra soc," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 4, p. e6018, 2021.

[118] S. Bateni, Z. Wang, Y. Zhu, Y. Hu, and C. Liu, "Co-optimizing performance and memory footprint via integrated cpu/gpu memory management, an implementation on autonomous driving platform," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 310–323, IEEE, 2020.

[119] H. Zhan, C. S. Weerasekera, J. Bian, and I. Reid, "Visual odometry revisited: What should be learnt?," *arXiv preprint arXiv:1909.09803*, 2019.

[120] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM*, vol. 40, pp. 1048–1066, Nov. 1993.

[121] S. M. LaValle and J. J. Kuffner, "Randomized Kinodynamic Planning," *The International Journal of Robotics Research*, vol. 20, pp. 378–400, May 2001.

[122] J. Ziegler and C. Stiller, "Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (St. Louis, MO, USA), pp. 1879–1884, IEEE, Oct. 2009.

[123] J.-w. Choi, R. E. Curry, and G. H. Elkaim, "Continuous Curvature Path Generation Based on Bézier Curves for Autonomous Vehicles.," *IAENG International Journal of Applied Mathematics*, vol. 40, no. 2, 2010. Publisher: Citeseer.

[124] J. Ziegler, P. Bender, T. Dang, and C. Stiller, "Trajectory planning for Bertha — A local, continuous method," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, (MI, USA), pp. 450–457, IEEE, June 2014.

[125] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, pp. 308–333, Mar. 2009.

[126] C. Gotte, M. Keller, C. Hass, K.-H. Glander, A. Seewald, and T. Bertram, "A model predictive combined planning and control ap-

proach for guidance of automated vehicles," in *2015 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, (Yokohama, Japan), pp. 69–74, IEEE, Nov. 2015.

[127] S. J. Anderson, S. B. Karumanchi, and K. Iagnemma, "Constraint-based planning and control for safe, semi-autonomous operation of vehicles," in *2012 IEEE Intelligent Vehicles Symposium*, (Alcal de Henares , Madrid, Spain), pp. 383–388, IEEE, June 2012.

[128] P. Fiorini and Z. Shiller, "Time optimal trajectory planning in dynamic environments," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 2, (Minneapolis, MN, USA), pp. 1553–1558, IEEE, 1996.

[129] A. Kelly and B. Nagy, "Reactive Nonholonomic Trajectory Generation via Parametric Optimal Control," *The International Journal of Robotics Research*, vol. 22, pp. 583–601, July 2003.

[130] M. McNaughton, C. Urmson, J. M. Dolan, and J.-W. Lee, "Motion planning for autonomous driving with a conformal spatiotemporal lattice," in *2011 IEEE International Conference on Robotics and Automation*, (Shanghai, China), pp. 4889–4895, IEEE, May 2011.

[131] Wenda Xu, Junqing Wei, J. M. Dolan, Huijing Zhao, and Hongbin Zha, "A real-time motion planner with trajectory optimization for autonomous vehicles," in *2012 IEEE International Conference on Robotics and Automation*, (St Paul, MN, USA), pp. 2061–2067, IEEE, May 2012.

[132] S. M. LaValle, "Rapidly-exploring random trees: a new tool for path planning," *The annual research report*, 1998.

[133] Y. Kuwata, G. Fiore, J. Teo, E. Frazzoli, and J. How, "Motion planning for urban driving using RRT," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Nice), pp. 1681–1686, IEEE, Sept. 2008.

[134] U. Schwesinger, M. Rufli, P. Furgale, and R. Siegwart, "A sampling-based partial motion planning framework for system-compliant navigation along a reference path," in *2013 IEEE Intelligent Vehicles Symposium (IV)*, (Gold Coast City, Australia), pp. 391–396, IEEE, June 2013.

[135] T. Heil, A. Lange, and S. Cramer, "Adaptive and efficient lane change path planning for automated vehicles," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, (Rio de Janeiro, Brazil), pp. 479–484, IEEE, Nov. 2016.

[136] F. von Hundelshausen, M. Himmelsbach, F. Hecker, A. Mueller, and H.-J. Wuensche, "Driving with tentacles: Integral structures for sensing and motion," *Journal of Field Robotics*, vol. 25, pp. 640–673, Sept. 2008.

[137] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a Frenét Frame," in *2010 IEEE International Conference on Robotics and Automation*, (Anchorage, AK), pp. 987–993, IEEE, May 2010.

[138] L. Fletcher, S. Teller, E. Olson, D. Moore, Y. Kuwata, J. How, J. Leonard, I. Miller, M. Campbell, D. Huttenlocher, *et al.*, "The mit–cornell collision and why it happened," *Journal of Field Robotics*, vol. 25, no. 10, pp. 775–807, 2008.

[139] M. Moghadam and G. H. Elkaim, "An Autonomous Driving Framework for Long-Term Decision-Making and Short-Term Trajectory Planning on Frenet Space," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, (Lyon, France), pp. 1745–1750, IEEE, Aug. 2021.

[140] M. J. Van Nieuwstadt and R. M. Murray, "Real-time trajectory generation for differentially flat systems," *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, vol. 8, no. 11, pp. 995–1020, 1998. Publisher: Wiley Online Library.

[141] E. Burns, S. Lemons, W. Ruml, and R. Zhou, "Best-First Heuristic Search for Multicore Machines," *Journal of Artificial Intelligence Research*, vol. 39, pp. 689–743, Dec. 2010.

[142] S. Heinrich, A. Zoufahl, and R. Rojas, "Real-time trajectory optimization under motion uncertainty using a GPU," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Hamburg, Germany), pp. 3572–3577, IEEE, Sept. 2015.

[143] A. Bleiweiss, "GPU accelerated pathfinding," in *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pp. 65–74, 2008.

[144] J. T. Kider, M. Henderson, M. Likhachev, and A. Safonova, "High-dimensional planning on the GPU," in *2010 IEEE International Conference on Robotics and Automation*, (Anchorage, AK), pp. 2515–2522, IEEE, May 2010.

[145] U. Cekmez, M. Ozsiginan, and O. K. Sahingoz, "A UAV path planning with parallel ACO algorithm on CUDA platform," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, (Orlando, FL, USA), pp. 347–354, IEEE, May 2014.

[146] D. Palossi, A. Marongiu, and L. Benini, "On the Accuracy of Near-Optimal GPU-Based Path Planning for UAVs," in *Proceedings of the 20th International Workshop on Software and Compilers for Embedded Systems*, (Sankt Goar Germany), pp. 85–88, ACM, June 2017.

[147] J. Fickenscher, S. Schmidt, F. Hannig, M. Bouzouraa, and J. Teich, "Path Planning for Highly Automated Driving on Embedded GPUs," *Journal of Low Power Electronics and Applications*, vol. 8, p. 35, Oct. 2018.

[148] F. Frenet, "Sur les courbes a double courbure.," *Journal de mathématiques pures et appliquées*, pp. 437–447, 1852.

[149] J.-A. Serret, "Sur quelques formules relatives à la théorie des courbes à double courbure.," *Journal de mathématiques pures et appliquées*, pp. 193–207, 1851.

[150] T. Cox and P. Thulasiraman, "A zone-based traffic assignment algorithm for scalable congestion reduction," *ICT Express*, vol. 3, no. 4, pp. 204–208, 2017.

[151] J. Li and N. Ferguson, "A multi-dimensional rescheduling model in disrupted transport network using rule-based decision making," *Procedia Computer Science*, vol. 170, pp. 90–97, 2020.

[152] R. Sánchez-Mangas, A. García-Ferrrer, A. De Juan, and A. M. Arroyo, "The probability of death in road traffic accidents. how important is a quick medical response?," *Accident Analysis & Prevention*, vol. 42, no. 4, pp. 1048–1056, 2010.

[153] C. S. Lim, R. Mamat, and T. Braunl, "Impact of ambulance dispatch policies on performance of emergency medical services," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 2, pp. 624–632, 2011.