CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Dynamic Edge-Server Data Processing for Vehicle Geofencing

Master's thesis in Computer science and engineering

Agnes Brogeby, Victor Ebbesson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's thesis 2026

Dynamic Edge-Server Data Processing for Vehicle Geofencing

Agnes Brogeby, Victor Ebbesson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Dynamic Edge-Server Data processing for Vehicle Geofencing
Agnes Brogeby, Victor Ebbesson

Supervisor: Marina Papatriantafilou, Computer Science and Engineering
Advisor: Victor Jarlow, AstaZero
Examiner: Marina Papatriantafilou, Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Dynamic Edge-Server Data processing for Vehicle Geofencing
Agnes Brogeby, Victor Ebbesson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Geofencing can be used to contain vehicles to a virtually defined area. A type of geofence that does this is the Model Predictive Geofence (MPG), which provides a way to contain vehicles within a geofence by using precomputed safety information from Hamilton-Jacobi reachability analysis. This enables predictive safety checks during operation, where a vehicle can determine whether its current state will lead to leaving the geofence before the geofence boundary is actually crossed. However, the resulting data can require large amounts of memory, making it difficult to store the full MPG on resource-constrained vehicle computers.

This thesis investigates whether predictive geofencing can be used on vehicle computers without storing the complete geofence data onboard. To address this, Dynamic Adaptive Loading of Spatial Datasets (DALOSD) is proposed, where the data of the MPG is divided into smaller segments that are requested from a remote server based on the vehicles current position and predicted future movement. The loading scheme uses a spatial window to determine which segments to load, keep, and discard, and introduces prioritization and speed limitation mechanisms to reduce the risk of missing required data during operation. The results show that DALOSD significantly reduces onboard memory consumption while still providing access to the required MPG data under suitable network conditions.

Keywords: Computer, science, computer science, engineering, thesis, dynamic data loading, Model Predictive Geofence, autonomous vehicles, Hamilton-Jacobi reachability, geofencing.

Acknowledgements

We would like to express our sincere gratitude to our supervisor and examiner, Marina Papatriantafilou, as well as our industry advisor, Victor Jarlow. Their continuous support, valuable guidance, and constructive feedback have been greatly appreciated throughout the work on this thesis. Their insights and encouragement have played an important role in shaping both the direction and the final outcome of this work.

We would also like to thank our families for their continuous support, patience, and encouragement throughout the work on this thesis. Agnes would especially like to express her deepest gratitude to her husband, Jeremiah Danielsen, whose unwavering support have been invaluable during this process.

During the work on this thesis, AI-based tools have been used as support for discussing ideas, identifying bugs and syntax errors in code, and improving the phrasing of selected parts of the text. All code of DALOSD as well as text in the report have been written, reviewed, and revised by the authors. The authors take full responsibility for the content, results, and conclusions presented in this thesis.

Agnes Brogeby, Gothenburg, 2026-07-30.

Victor Ebbesson, Gothenburg, 2026-07-30.

Contents

List of Figures

List of Tables

Glossary

loading scheme latency is defined by the average time it takes from the point it is noted that a data segment is needed, to when the data segment is in storage within the vehicle. 14, 15, 26, 27, 33

loading scheme stop a loading scheme stop is an emergency brake due to a data segment not being loaded on time. ix–xii, 15, 16, 22, 24, 30, 31, 33–39, 41–45, 49, 50, 53

memory consumption refers to the amount of working memory needed to store the MPG in the vehicle. 15, 19, 23, 27, 30, 33, 34, 38, 39, 46–49, 52, 55

state is defined as (x, y, v, θ), where x, y is the vehicles position, v the current speed and θ the current rotation. x, 5, 7, 8, 13–15, 18, 20, 21, 24–26, 28, 52

Acronyms

ABS Anti-lock Braking System. 19

BRT Backwards Reachable Tube. x, 5–9, 11, 13–16, 18–21, 24–26, 29, 34, 38, 39, 41, 45–48, 52, 53, 55

DALOSD Dynamic Adaptive Loading of Spatial Datasets. iv, vi, viii–x, 2, 3, 14–18, 21–28, 30, 32, 34, 38–42, 44–46, 48–55

MAD-C Multi-stage Approximate Distributed Cluster-combining. 10, 23

MPG Model Predictive Geofence. iv, viii, x, xi, xiii, 7–9, 11, 13–16, 18–22, 24, 27, 28, 31, 32, 38–41, 45–51, 53, 55, 56

PDE Partial Differential Equation. 5, 6

RSU roadside unit. 12, 51

RTT Round Trip Time. x, xi, 30, 32, 40–44, 48, 50, 51, 53

SAT Separating Axis Theorem. 9, 10, 23

SDM Spatial Decomposition Method. viii, 11, 18–20, 28, 29, 51, 52

1

Introduction

Future transportation systems are expected to include a wide range of automated units, from personal vehicles to large industrial machinery, operating with increasingly higher levels of autonomy. Such systems have the potential to improve productivity, increase safety, and reduce the need for humans to perform repetitive or hazardous driving tasks.

To safely deploy autonomous vehicles, it is necessary to define and enforce operational boundaries that prevent vehicles from entering unsafe or unauthorized areas. A commonly used approach is geofencing, where virtual boundaries are established, and actions are triggered when a vehicle approaches or crosses those boundaries [1]. Geofences can be implemented in several ways depending on the application and safety requirements.

The simplest form of geofencing relies on static boundaries that remain unchanged over time. An example is autonomous lawnmowers, which operate within predefined areas using GNSS and RTK positioning systems [2]. While such approaches are sufficient for low-speed applications in controlled environments, they become increasingly challenging to apply in road-traffic scenarios due to positioning uncertainty, higher vehicle speed, and the need to account for vehicle dynamics [3].

To address these limitations, dynamic geofences have been proposed, where the position or shape of the geofence changes over time. One example is movable safety zones around emergency vehicles, enabling nearby vehicles to receive warnings and thereby reducing collision risks [4].

More advanced approaches incorporate predictive capabilities that evaluate future vehicle dynamics and provide formal safety guarantees. One such framework is Hamilton-Jacobi reachability analysis [5], which computes the set of states from which a vehicle can be guaranteed to remain within a safe operating region despite disturbances and uncertainties. In simpler terms, the Hamilton-Jacobi approach looks at all positions in the geofence and calculates, using a model for vehicle dynamics, whether it can continue to operate for all rotations and speeds or whether it needs to perform evasive action such as emergency braking. As a result of the mathematical guarantee for when evasive actions are required, Hamilton-Jacobi reachability has emerged as a promising foundation for safety-critical geofencing and autonomous vehicle supervision.

Despite these advantages, Hamilton-Jacobi reachability suffers from a major practi-

cal limitation. Computing reachable sets requires solving high-dimensional partial differential equations, resulting in significant computational and memory requirements that grow rapidly with system complexity [5]. Although these computations can be performed offline and the resulting safety data stored for later use, deploying such solutions on resource-constrained edge computers remains challenging. In particular, storing the entire result from the computation locally may exceed the available memory, while repeatedly transferring large datasets can introduce communication delays and reduce availability.

This challenge motivates the need for mechanisms that enable efficient access to precomputed reachability data without requiring the complete dataset to be stored onboard the vehicle. Therefore, this thesis investigates methods for dynamically loading Hamilton-Jacobi backwards reachability results from remote storage to resource-constrained edge platforms. The objective is to reduce local memory requirements while maintaining the availability and safety guarantees required for real-time geofencing applications. This is done by answering the following research questions throughout the work:

1. Is it possible to use geofencing with predictive components in vehicle computers?
2. Is it possible to reliably have access to a geofence without keeping all geofence information onboard the vehicle?

The main contribution of this thesis therefore lies in addressing one of the main practical barriers to using predictive geofencing in real vehicle systems, which is the large amount of onboard memory required to store the precomputed reachability data. To solve the issue, this thesis proposes *Dynamic Adaptive Loading of Spatial Datasets (DALOSD)*, which significantly reduces the memory requirement onboard the vehicle by dynamically loading smaller segments of the pre-calculated reachability data over network, depending on the vehicle's current and predicted future movement.

The main challenge of the work lies in upholding safe operations. Storing safety critical data elsewhere than locally on the vehicle introduces safety risks due to lack of access, as access to the data now becomes reliant on data transfers over the network. If data saying whether a vehicle state is safe or not is inaccessible, the state must be treated as unsafe until proven otherwise. This means that lack of access to data could cause a vehicle to perform an unnecessary emergency brake, which could introduce hazardous situations where accidents may occur due to unpredictable behavior from the vehicle. To avoid these situations to occur DALOSD contains a mechanism to slow the vehicle's speed to a level where access to the reachability data is more likely to be upheld.

The thesis is structured into nine chapters that progressively develop the approach for DALOSD. Chapter 2 provides the theoretical foundation necessary for understanding the work, while Chapter 3 discusses related work. Chapter 4 formally defines what problem this thesis aims to solve as well as formally defines the challenges of dynamically loading reachability data from server to vehicle. Chapter 5

presents the method for implementing DALOSD while Chapter 6 explains how the different parameters of DALOSD have been selected. Evaluation is conducted in Chapter 7, and further discussed in Chapter 8. Finally, Chapter 9 concludes the work by summarizing the main contributions and findings of this thesis.

2

Background

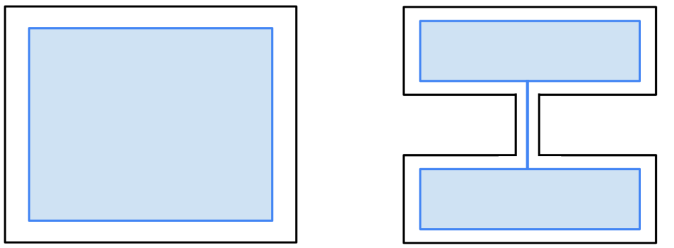
This section describes the necessary background knowledge needed for understanding the context of the thesis. It begins by outlining different ways of implementing geofences in Section 2.1, then describes Hamilton-Jacobi reachability analysis in Section 2.2, followed by an implementation of a geofence through reachability analysis in Section 2.3, and finally describing methods for collision detection in Section 2.4.

2.1 Geofencing used for containment of vehicles

Geofencing is a software-defined virtual boundary based on maps or coordinates. It allows for monitoring vehicle movements, making it possible to intervene when a boundary is crossed, or about to be crossed. This creates the possibility to contain a vehicle within a predefined or dynamically defined area. One approach to implement geofencing is by comparing the collected GPS coordinates of an entity and investigate if the coordinates are within the statically defined area. This approach can be used to alert if an entity leaves or enters a designated area, such as a car leaving city boundaries or a freight truck entering the proximity of a drop-off point [1].

Using a geofence to alert any crossing of its boundary can seem straightforward enough. However, if the geofence is meant to be used as a hard boundary, not allowing any vehicle to either enter or exit, the implementation of it becomes harder. If the geofence only alerts once its boundary has been crossed, the vehicle will not be strictly confined to the geofence area, but instead stay within its vicinity [6]. To avoid this, and securely contain a vehicle within a geofence, there needs to be a mechanism able to anticipate the approach of the boundary [7].

One solution is defining different sets of boundaries, as done by Steven and Atkins in [6]. Instead of only using the actual geofence boundary, they define extra sets of boundaries at a fixed distance from the geofence boundary. These new boundaries create an operational area, and crossing them triggers a warning that the vehicle is getting close to the actual geofence boundary, making it possible to stop the vehicle before it exits the geofence area. Having these boundaries at fixed distances can, however, make it impossible to utilize the full geofence for some shapes. An example of this can be seen in Figure 2.1, where Figure 2.1b shows how the inner boundary at a fixed distance hinders access to the full geofence by not giving a wide enough operational area to utilize the path between the two larger squares.



(a) Square geofence. (b) Non square geofence.

Figure 2.1: Depictions of two geofences with an inner operational area in blue at a fixed distance from the geofence boundary.

Another solution is to constrain the vehicle's velocity depending on how close it is to the geofence border, Cavanini et al [8] propose an algorithm for Model Predictive Control of the vehicle that calculates speed bounds over a specified time horizon, depending on the vehicle's deceleration ability. However, continuously performing these calculations in real time is heavy and takes up computational resources. An alternate solution, which allows for a similar result without the heavy calculations for the onboard computer in the vehicle, can be found through Hamilton-Jacobi reachability analysis.

2.2 Hamilton-Jacobi reachability analysis

Hamilton-Jacobi reachability analysis is a method for analyzing dynamical systems to determine whether a system can reach a specified set of states within a given time horizon [9]. For example, the *state* of a vehicle may consist of its position, velocity, and heading. Intuitively, consider a vehicle driving toward a tree. Hamilton-Jacobi reachability analysis can determine which vehicle states will inevitably result in a collision with the tree within a specified time horizon.

Formally, reachability analysis seeks to determine the set of states M that can be reached from an initial state x_0 during a time interval T. Hamilton-Jacobi reachability analysis extends this concept by computing the set of states R from which the system is guaranteed to reach a specified target set L within the time horizon T. This set R is commonly referred to as the backwards reachable set or *Backwards Reachable Tube (BRT)*.

If the target set L represents unsafe states, then all states belonging to R are also considered unsafe, since trajectories originating from these states will inevitably reach the unsafe region within the considered time horizon [5].

To compute R, the Hamilton-Jacobi *Partial Differential Equation (PDE)* is solved backward in time. Consider a controlled dynamical system described by

$$\frac{dx}{dt} = f(x, u)$$

where x is the system state, u denotes the control input and $f(x, y)$ describes how the state evolves over time. The target set L is represented by an implicit surface function $\phi(x)$, defined such that

$$\phi(x) \leq 0$$

for all states inside the target set and

$$\phi(x) > 0$$

for all states outside the target set.

The value function $V(x, t)$ represents the signed distance to the BRT. States satisfying

$$V(x, t) \leq 0$$

belong to the BRT and are therefore guaranteed to reach the target set L within the considered time [5].

The Hamiltonian $H(x, \nabla V)$ characterizes the optimal instantaneous change of the value function along system trajectories and is defined as:

$$H(x, \nabla V) = \min_u \nabla V \cdot f(x, u).$$

The minimization selects the control input that minimizes the rate of change of the value function. When the target set represents unsafe states, this formulation captures the states from which reaching the unsafe region cannot be avoided, making it useful for safety analysis and emergency intervention systems [5]. Combining these components yields the Hamilton-Jacobi PDE

$$\frac{\partial V}{\partial t} + H(x, \nabla V) = 0, \qquad V(x, 0) = \phi(x).$$

Solving this PDE backward in time over the chosen time horizon produces a BRT, consisting of all states satisfying

$$V(x, t) \leq 0.$$

The BRT therefore represents all states from which the system is guaranteed to enter the target set within the specified time horizon [5].

An example illustrating the use of a BRT is shown in Figure 2.2. In this example, the target set consists of all states outside the yellow geofenced area. For simplicity, assume that no control inputs are available and that the arrows indicate the trajectories followed by the vehicle during the considered time horizon. Under these assumptions, state v does not belong to the BRT because its entire trajectory remains within the geofence. In contrast, state u belongs to the BRT because the trajectory originating from u enters the unsafe target set during the time horizon, even though the final state of that trajectory lies outside the target set. Since all states belonging to the BRT are considered unsafe, the vehicle state corresponding to u would require an avoidance action, such as evasive steering or emergency braking.

An issue with Hamilton-Jacobi reachability analysis is that the computational complexity grows rapidly with the dimensionality of the state space. More specifically, the memory requirements scale as $O(N^d)$, where N is the discretization resolution per dimension and d is the number of state dimensions. Consequently, high-dimensional systems may require several gigabytes, or terabytes, of memory and computation times ranging from hours or days to years, making some problems impractical to solve [5].

This scalability challenge is particularly problematic for edge computing, which typically has limited computational resources and memory capacity. Numerous approaches have been proposed to reduce the computational burden through numerical optimizations and approximation techniques. However, despite these improvements, Hamilton–Jacobi reachability analysis remains computationally demanding for high-dimensional systems [10]. As a result, real-time computation onboard vehicles is generally infeasible for many practical applications. A solution to this is however found in the *Model Predictive Geofence*.

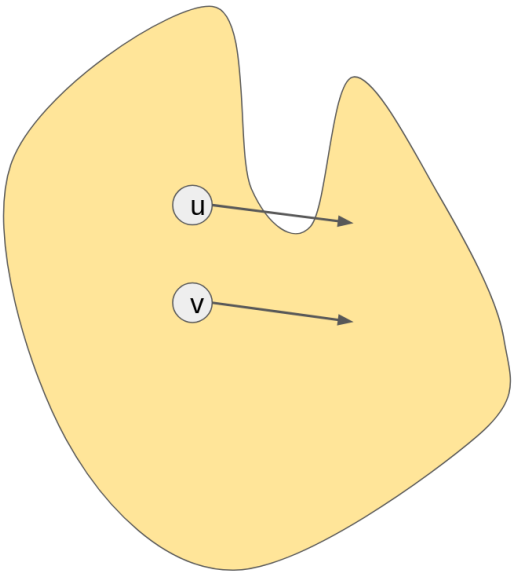


Figure 2.2: An example of a geofence. The yellow area defines the geofence with two states u and v with possible travel paths.

2.3 The Model Predictive Geofence

Thorén et al [11] present the *Model Predictive Geofence (MPG)*, where a geofence is implemented through the use of Hamilton-Jacobi reachability analysis. In their method, vehicle states depending on position, speed, and direction are used to calculate all possible future states within a specified time horizon by using the BRTs from Hamilton-Jacobi reachability analysis. During operations, a vehicle will compare its current state to those stored in the BRT of the MPG. If the current state will lead to the vehicle crossing the geofence boundary, action can be taken to avoid it.

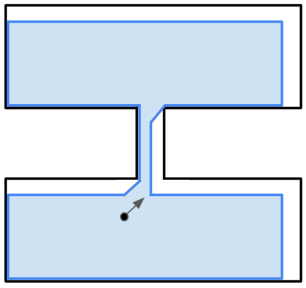


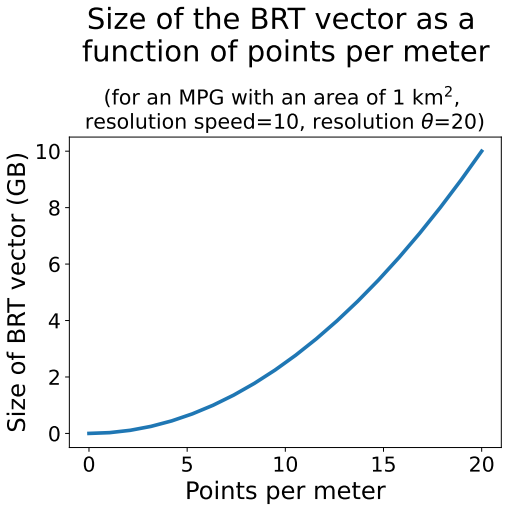
Figure 2.3: An example of the operational area of the model predictive geofence in the instance a vehicle is traveling toward the upper right corner for one specific state.

The MPG has two phases: first, an offline phase where the BRT for each state, depending on position, speed, and direction, is calculated. These calculations generate a set of states leading to a vehicle exiting the area. The BRT is stored as a four-dimensional tensor, which is then stored as a vector to further decrease space requirements. Each entry, which corresponds to a lookup for a single state, is stored as a single bit, which means eight states can be stored in a single byte. Secondly, there's an online phase where the vehicle's current state is compared against the previously computed BRT. This is done by calculating an offset derived from the current state by adding the state's components. While the calculations of the offline phase are heavy, utilizing the MPG in the online phase is a lot less so. Since all sets are precalculated and stored in the BRT vector, the only computations necessary onboard the vehicle are a series of lookup operations with a complexity of $O(1)$.

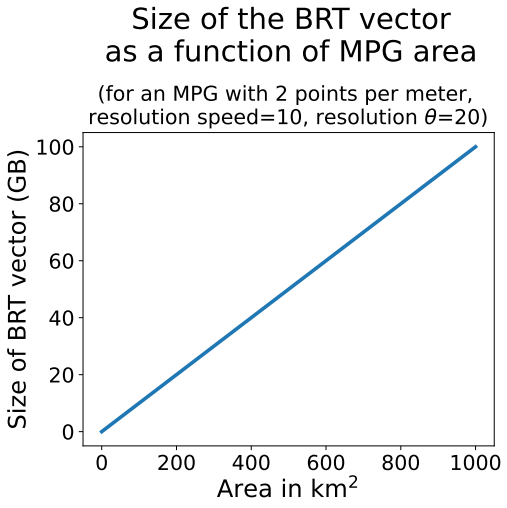
The analysis done in the online phase leads to a high utilization of the entire fenced area, while still guaranteeing the vehicle stays within the geofence boundaries as long as it adheres to the results of the MPG lookup operations. Figure 2.3 shows a visualization of the operational area of the MPG for a vehicle in the instance it is travelling at a specific speed toward the upper right corner of the geofence. Compared to the previously presented geofences with an operational area at a fixed distance from the geofence boundaries, the MPG has a higher utilization of the area, as it takes the current state of the vehicle into account. As seen in Figure 2.3, the narrow path between the two squares, previously inaccessible in Figure 2.1b, is now accessible with this type of geofence.

One problem of the MPG is found in the calculations of the offline phase having very high memory requirements [11]. As previously mentioned in Section 2.2, calculations for high resolutions and a high number of dimensions can sometimes be impossible to calculate on regular computers. Depending on the size of the geographical area, as well as what resolution it is calculated in, the BRT vector of the MPG can also require a lot of storage space in the online phase, which is not suitable for vehicle computers that typically have limited resources [5].

Two examples of how the size of the BRT vector increases is shown in Figure 2.4, where Figure 2.4a shows how the size of the BRT vector grows when the area and resolution in speed and direction as θ is fixed, but the resolution in points per meter



(a) The size of the BRT vector as a function of points per meter.



(b) The size of the BRT vector as a function of MPG area.

Figure 2.4: The size of the BRT vector.

increases. Figure 2.4b shows how the size of the BRT vector increases when the area increases but the resolutions for speed and direction θ as well as points per meter is fixed.

An MPG covering the area between Gothenburg and Alingsås in Sweden, two cities at a distance of 50 km of each other, needs an area of around 900 km^2. That area corresponds to a BRT vector size of 90 GB. If the MPG were to instead cover the area between Gothenburg and Stockholm in Sweden, two cities at a distance of around 480 km from each other, the size of the BRT vector increases to around 6.4 TB.

2.4 Methods for collision detection

When using geofences it can be relevant to examine whether two objects are overlapping. This can be done through collision detection. A foundation for many such algorithms is the *Separating Axis Theorem (SAT)* [12].

The Separating Axis Theorem is a fundamental result in computational geometry and forms the basis of many efficient collision detection algorithms. SAT follows from the hyperplane separation theorem and provides a practical criterion for determining whether two convex sets intersect [12].

Theorem 1 (Separating Axis Theorem [12]) *Let A and B be two convex sets in $\mathbb{R}^2$. The sets A and B are disjoint if and only if there exists a direction (an axis) such that the orthogonal projections of A and B onto that axis do not overlap.*

In collision detection, SAT is commonly applied to convex polygons in two-dimensional space. Rather than directly testing for geometric intersection, the problem is reduced

to a sequence of one-dimensional overlap tests. Each polygon is projected onto a set of candidate axes, and the resulting projection intervals are compared. If a separating axis exists, the intervals on that axis will not overlap, proving that the polygons are disjoint. Conversely, if the projections overlap on all candidate axes, the polygons intersect.

For convex polygons in two dimensions, it is sufficient to consider only the axes defined by the outward normals of the polygons' edges. This significantly reduces the computational complexity of the test while maintaining correctness. The overlap test on each axis is performed using simple comparisons between the minimum and maximum projection values [12].

SAT has proven efficient in a wide range of narrow-phase collision detection applications. For example, Gottschalk et al. [13] demonstrated its usefulness by integrating SAT-based tests with bounding volume hierarchies for efficient interference detection. Due to its limited set of candidate axes and computationally inexpensive projection operations, SAT remains a robust and efficient framework for determining whether two convex polygons overlap, making it well suited for fast and reliable collision detection in two-dimensional environments [14].

Another possible solution for collision detection within geofences is *Multi-stage Approximate Distributed Cluster-combining (MAD-C)*, which operates by abstracting unstructured, high-volume data into local clusters, which are then summarized using constant-sized bounding ellipsoids. These ellipsoidal models are particularly advantageous for certain geofencing applications [15]. By evaluating the relative position of an ellipsoid against the bounding planes of a geofence, MAD-C can determine spatial violations with linear computational complexity, significantly reducing both processing time and communication bandwidth in distributed edge-computing networks.

3

Related works

This chapter presents the related work of the thesis. It starts by describing work done to allow generation of MPGs for larger areas, and then continues describing High-definition maps and work regarding transferring these from infrastructure to vehicles.

Spatial decomposition of the MPG. A solution to the problem of needing large amounts of memory for calculating the MPG has been formulated by Hedlund and Färenmark in their master's thesis *Efficient Backwards Reachability Computation for Vehicle Geofencing* [5]. They present the *Spatial Decomposition Method (SDM)*, which partitions the operational area into overlapping segments. These segments are used to create several small MPGs whose BRT vectors are then merged to create the final MPG. The overlap is set to contain the maximum braking distance of the vehicle, as well as some safety margin. This is done to ensure that a state will not get falsely flagged as leaving the operational area, and allowing the vehicle to enter a new MPG segment. The SDM significantly reduced the memory consumption in the offline phase, allowing the generation of MPGs for larger areas and higher resolutions while maintaining a high quality of the resulting MPG. In the thesis, the BRT vector segments are merged to create a full BRT vector for an MPG of the area, but it is stated that the BRT vector can be stored in segments instead, at the cost of a bigger total file size due to overlapping data points between segments. Storing the BRT vector of the MPG as segments could make it possible to only load the parts currently needed for vehicle operation in real-time.

High-definition maps. Another related area of research is that of *High-definition (HD) maps*. An HD map provides highly detailed environmental representations of the world, providing insights about road layouts in real time, and is essential for autonomous vehicle navigation [16]. The concept of HD maps is still under research, and even though HD maps are considered important for the development of cooperative automated vehicles, there is no standard or guidelines of what constitutes an HD map [17]. Most existing literature regarding the transfer of HD maps consists of moving information from vehicles to central servers, but less work can be found about dynamically moving parts of an HD map to vehicles in need of the information.

Broadcasting HD maps. An example of information transfer from server to vehicle is provided by Annavazzala et al. [16], who developed a broadcast scheduling scheme for HD maps in vehicular networks. The scheme accounts for the bandwidth

of the broadcasting *roadside unit (RSU)*, vehicle speed, and the size of the HD map. In the paper, an HD map is vertically partitioned into 'tiles', where one tile contains layers with different levels of priority. Highly dynamic data, such as pedestrians and vehicle paths, are contained in the highest layer and deemed as low priority. The lowest layer contains highly static data, such as buildings and road segments, and is of the highest priority as it is needed for basic navigation. If a vehicle lacks information about an upcoming road segment, it will monitor the RSUs broadcast channel to check if the data is currently being broadcast, otherwise it sends a request for the tile to the RSU. The request contains the vehicle's location and speed as well as the tile's location. The RSU will process the request and, if the requested tile is of high enough priority or is not already scheduled to be broadcast, schedule it to be broadcast before the vehicle physically reaches the requested tile. The goal of the algorithm is not to answer every request with the corresponding tile, but to make sure that the tiles with the highest priority are available to vehicles in time, meaning that requests of lower priority tiles might go unanswered. The algorithm also only concerns the server side of operation, and does not touch on how a vehicle will select which tiles are needed.

HD maps and ontologies. Another example of server-to-vehicle information transfer is presented by Qiu et al. [18] who study the problem by using ontologies. HD maps require, because of their extreme detail, a lot of processing power compared to traditional digital maps. This makes it impossible to store the entire HD map within a vehicle, and the vehicle needs to constantly request new parts of the map as it is progressing along a route. Since HD maps is an area of research that is still relatively fresh, there is no standard format for them. The problem of dynamically updating the local map data within the vehicle, while dealing with heterogeneous formats, is combated by using ontologies. An ontology provides a structured description of the concepts and relationships needed to model a particular domain. The ontology approach is used to model spatial reasoning that considers vehicle motion events, and the paper produces a use case where a car following a road segment will pre-fetch map tiles depending on its current position and remaining distance to the next tile. This is done by using the concept of a spatial window, a moving window defined by a forward- and backward parameter from the moving vehicle, where tiles overlapping the spatial window get prefetched.

4

Problem definition and aim

This chapter summarizes the Model Predictive Geofence, describes the problem definition and the aim of the thesis, challenges and scope of the thesis, as well as ethical considerations.

4.1 Model Predictive Geofence summary

This section is a brief summary of Section 2.3, which describes the *Model Predictive Geofence (MPG)*, a geofencing approach that ensures a vehicle remains within a defined operational area by using precomputed information derived from Hamilton-Jacobi reachability analysis. Instead of reacting when a boundary is crossed, the MPG predicts whether a vehicle's current *state*, defined by its position, speed and direction, will inevitably cause the vehicle to leaving the geofence, enabling preventive action such as braking before a violation occurs [11].

Using the MPG requires operations in two phases. In an offline phase, the computationally intensive reachability analysis is performed to generate a *Backwards Reachable Tube (BRT)*, representing all unsafe states that would lead to exiting the geofence within a given time horizon. These results are then stored as a BRT vector where each entry corresponds to a specific state and tells whether a state is safe for a vehicle to continue operating or not. In the online phase, the vehicle performs simple lookup operations, where the lookup offset is derived from the current vehicle state, in the stored BRT vector to determine whether its current state is safe. This allows for predictive decision-making with minimal onboard computation [11]. The online phase of the MPG can however need large amounts of storage due to the size of the BRT vector, which is not suitable for environments with limited resources [5].

4.2 Problem definition

There is a need to use geofences to contain vehicles to an operational area. Using the Model Predictive Geofence method in real-life scenarios faces problems due the heavy calculations needed to produce it, as well as the memory needed to store it.

One possible solution is to split the BRT vector of the MPG into smaller segments that can be transferred to the vehicle on demand. To reiterate, the BRT vector is a four-dimensional matrix represented as a vector, which is indexed by the vehicle's

state (x, y, v, θ) where x, y represents the vehicle position, v its current speed, and θ its current heading. Each position in the vector contains a single bit that tells if a state is safe or not. The BRT vector originally covers the full area of the MPG, but could be partitioned into smaller segments in such a way that the x and y coordinates are grouped together to form smaller spatial segments, in essence creating a number of smaller MPGs. This would be similar to how a large digital map can get divided into smaller, more manageable tiles. This leads to the possibility of instead of storing the complete BRT vector onboard, only the segments relevant to the vehicle's current and predicted positions in the near future can be requested from a remote server. This introduces the problem of how to reliably and efficiently provide the vehicle with the required MPG data during operation.

The aim of this work is therefore to design a loading scheme capable of *dynamically transferring precomputed BRT vector segments of an MPG from a remote server to a moving vehicle* and will hence forth be named by *Dynamic Adaptive Loading of Spatial Datasets (DALOSD)*. DALOSD needs to determine which segments are required based on the vehicle state. For *dynamic loading* the scheme needs to ensure that the BRT vector segment of the MPG containing the current position is always loaded onto the vehicle. To ensure this there needs to be a predictive component anticipating what segments will be needed in the near future. DALOSD must also be able to synchronize data transfers, memory updates and lookup operations, allowing these to happen concurrently without significantly affecting the performance of the system.

How fast the server can fulfill requests for BRT vector segments might differ depending on conditions such as current bandwidth. The system therefore needs to be able to adapt to these conditions to avoid needing to stop the vehicle due to missing BRT vector segments.

The operational output of DALOSD consists of the BRT vector segments of the MPG currently stored in the vehicle's memory. These segments are then used for lookup operations that determine whether the vehicle's current state is safe with respects to the geofence boundaries, allowing the vehicle to either continue operation or initiate an emergency brake.

4.3 Properties of interest

The potential impact of this work is enabling the use of the MPG in resource-constrained vehicle computers, in turn making it possible to use Hamilton Jacobi reachability analysis in an environment where calculating it in real time might put strain on the system. This his however done at the cost of reduced access to the full BRT vector of the MPG. To evaluate its feasibility it is important to identify what properties are of interest, which in this thesis are *data availability, balancing memory and latency* and what *trade-offs* DALOSD introduces.

Data availability. In DALOSD, the most important property is having the data accessible when needed. This is the foundation of being able to utilize the MPG within the vehicle. The data availability is closely connected to the *loading scheme*

latency and the *memory consumption* within the vehicle. The loading scheme latency stems from two different components: the complexity of computations within DALOSD, in other words the amount of time it takes to calculate what data is needed, as well as the network conditions.

Balancing memory and latency. Allowing a high memory consumption can increase data availability and combat high loading scheme latency, since more segments can be kept in memory, and segments can be loaded further ahead, thus giving a greater overhead for peaks of latency. Minimizing the memory consumption can however align well with meeting the requirements of using as little resources as possible, but comes at the price of not being able to load data as far ahead. This can create a situation where there is a higher risk of not having the needed data available in the case of a sudden drop in network connectivity.

Trade-offs. Here, some trade-offs arise when trying to minimize loading scheme latency to ensure the required data is available while also minimzing memory consumption to ensure DALOSD does not use to much resources. A simple algorithm reduces the amount of computations needed when deciding what data to load, but can be coarser in its decision making and thus result in a high memory consumption. The higher memory consumption can increase the probability of having the needed data available, but can also strain the system by increasing the amount of data needed to transfer over the network. A more complex algorithm with higher accuracy in what data to load can increase the number of computations needed, thus adding time to the loading scheme latency, but also reducing the memory consumption by being more granular. The lower memory consumption can however come at the price of reducing the possibility of having the needed data available while also reducing the systems resilience against poor network conditions.

4.4 Challenges of dynamically loading safety critical data

For this thesis, safety will be defined as avoidance of *loading scheme stops*, where a loading scheme stop is defined as a stop due to lacking access to a necessary part of the BRT vector of the MPG in the onboard computer. To ensure safe operations, it is vital to guarantee that the necessary operational data is accessible when needed. Since data is sent over a network, some latency is to be expected, and the scheme needs to leave enough latency overhead to avoid loading scheme stops due to missing data.

This definition of safety comes from the fact that the MPG data contains information regarding the safety of a vehicle state, meaning the data is critical for safe operations. A challenge for DALOSD lies in defining what safe operations actually are, and how to evaluate and guarantee safety under different definitions of safety. For example, it can be a definition of safe to always ensure the vehicle never leaves the dedicated area. With this definition, safety can be guaranteed by placing the vehicle within the geofence and not allow it to move. This is then a safe solution, but not a practical one, as it hinders the vehicles normal operations. Another definition can involve

risks associated with stopping a vehicle. It might not be safe to stop the vehicle, as is the case while traversing a highway, where an emergency brake can cause a rear end collision from a car traveling behind.

Another challenge lies in the fact that the process of checking if the vehicle is inside the geofence is considered a background process while the vehicle is performing other duties. DALOSD can therefore not take up too many resources, so as not to hinder the main operations of the vehicle. This makes it necessary for any solution to minimize the amount of working memory needed.

This summarizes to two main challenges:

1. Avoid loading scheme stops due to missing data.
2. Minimize the amount of working memory needed.

4.5 Thesis scope

A fully operational scheme for dynamically loading parts of the BRT vector of an MPG needs to work for a high number of vehicles. If many vehicles request information at the same time the network might become congested and the server might run into scheduling issues. This is, however, deemed to be out of the scope of this thesis, which will instead be limited to only consider one vehicle and one server.

The modeling of network conditions will be simplified and will not account for real-world fluctuations, such as distance to the transmission source, weather conditions, or other environmental factors that may influence available capacity. For evaluation network conditions, such as bandwidth, latency and packet loss, will generally be represented by different constant values.

A system limitation is that the MPG relies on precise positioning, and does not take road and weather conditions into account. Because of that this thesis will assume that the vehicle is operating under ideal conditions with precise positioning. Adapting the system to take unreliable positioning, road and weather conditions into account is left for future work.

4.6 Ethics and risk

When it comes to automated vehicles, safety is a major concern. In the case of DALOSD in this thesis, the most apparent goal is to achieve high reliability, which ultimately points to a guarantee of safe operations. In this thesis, safe operations have been defined as a guarantee that the vehicle will stay within the bounds of the geofence and avoid loading scheme stops. If those criteria were to fail, and an accident occurs, what ethical implications would that have?

Hansson et al. [19] discuss the topic of responsibility for safety in the case where the vehicle is fully automated. Traditionally, responsibility for vehicle accidents has been attributed to the driver. In the case of automated vehicles, that responsibility

is instead largely transferred to the constructors and maintainers of the vehicles, as well as the road infrastructure and communication systems that support them. Automation of vehicles has the potential to majorly increase safety through features such as short reaction time, as well as communication with surrounding vehicles and infrastructure.

The ethical implications for this thesis becomes a need to clarify what limitations DALOSD have. In a system meant to provide a guarantee of safe operations it is paramount to explore under what conditions this can be upheld.

5

Method

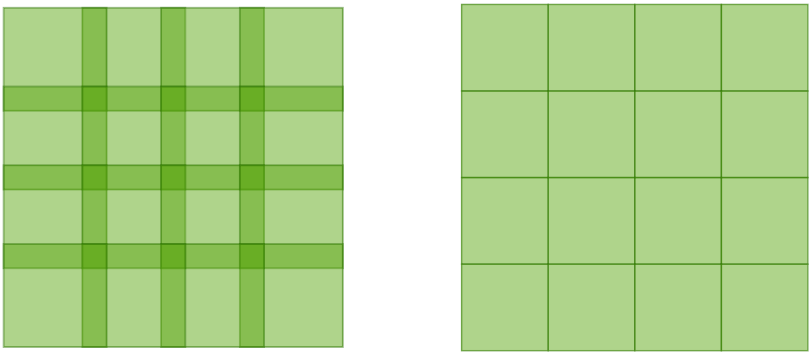
This chapter presents the proposed method for dynamically transferring MPG data between a remote server and a moving vehicle. The approach is centered around Dynamic Adaptive Loading of Spatial Datasets that determines which parts of the MPG are required for the vehicle, based on the current vehicle state. The method aims to ensure that the necessary data is available in time for safe operation, while minimizing memory usage and communication overhead.

5.1 Loading segments or streaming data

To be able to generate an MPG of a large area, the Spatial Decomposition Method [5] previously presented in Section 3 is used. When further deciding how the BRT vector of the generated MPG should be transferred to the vehicle, two main solutions have been identified. One solution is to continuously stream individual bytes of the needed parts of the BRT vector. Another is to segment the BRT vector and transfer larger chunks that get stored in memory, a method similar to the ones used in the studies [16], [18] presented in Section 3 related works.

One of the advantages of the MPG lies in its efficient lookup operation, as previously mentioned in Section 2.3. The lookup operation, in turn, relies on the offset for a state in the BRT vector being a simple operation of adding the components of the state. Using streaming either moves this calculation to the server, which would be bad practice in real-time autonomous vehicle calculations [20], or it would create significant overhead in storage management onboard the vehicle, as the onboard system would need to track a large amount of entries separately. Further, it would need to align its local BRT vector with the offset calculation. For this reason, this thesis uses segments instead of streaming individual bytes.

The segment approach can still become similar to streaming individual bytes with a small enough segment size, while also minimizing the amount of computations wasted on storage management. If the segments become too small, they can, however, introduce significant overhead on the network layers, making it costly for little gain. This increased cost comes from extra information connected to tracking, which information belongs to which coordinates.



(a) Segmentation with the SDM method.

(b) The proposed method for segmentation.

Figure 5.1: The two different methods for segmenting the MPG.

5.1.1 Using the segmentation of the SDM

As stated in Section 3, a way to create BRT vector segments already exists from the previous work done in the master's thesis presenting the SDM [5], where the full area of the MPG is partitioned into smaller segments, which are then merged to form the full MPG after calculating the BRT vector of each individual segment. These segments include an overlap set to contain the maximum braking distance of the vehicle, as well as some safety margin, to allow the vehicle to enter a new segment when calculating the BRT vectors. The overlap in these segments do however, as visualized in Figure 5.1a, introduce major overhead with regard to memory consumption while storing the data.

One example showing the excessive memory consumption is illustrated through an MPG covering a rural road, on which the maximum speed v_{max} is 90 km/h. The overlap is derived in part from the vehicle's breaking distance, which in the SDM [5] is estimated as

$$breakingDistance = \left\lceil -1 \cdot \frac{v_{max}^2}{2 \cdot (-5.25)} \right\rceil.$$

In this equation, the maximum speed is squared and then divided by two times the deceleration constant, which here is set to -5.25 [5]. The deceleration constant will vary depending on the road condition, the conditions of the vehicle and whether *Anti-lock Braking System (ABS)* brakes are used or not, but is generally in the range of 5-6 m/s in good condition with ABS brakes [21].

The segment overlap is then calculated by creating two squares. The first square is the actual segment without overlap accounted for as

$$S_{original} = segmentSize^2,$$

which is equal to the squared segment size. The second square is the original segment size plus overlap factors as

$$S_{overlap} = (segmentSize + 2 \cdot buffer + breakingDistance)^2,$$

where the buffer is a safety margin whose size will depend on the specific shape of the geofence, but generally around 6-10 meters is considered a good value [5].

To create an example with actual numbers, let us assume the $segmentSize$ to be 300 meters and the $buffer$ to be 6 meters. This gives us the values $S_{original} = 90000$ and $S_{overlap} = 138384$. To calculate how much larger $S_{overlap}$ is relative to $S_{original}$ we divide $S_{overlap}$ by $S_{original}$. The difference is about 1.5, where the extra space is wasted on overlap which increases the total amount of data that needs to be transferred over the network. If smaller segment sizes are used the problem becomes even more pronounced. If the $segmentSize$ is set to 100 meters, and the $buffer$ is kept at 6 meters, then $Square_{overlap} = 29584$ and $Square_{simple} = 10000$, which gives a factor of almost 2.96.

5.1.2 An alternative way of segmenting the MPG

Another solution to create BRT vector segments, which is proposed in this thesis, is to create them from a fully merged BRT vector. This solution still utilizes the SDM, and by extension its segmentation, to be able to create MPG's of larger areas. However, instead of directly using the SDM segments containing overlaps, the SDM is allowed to complete, meaning its segments are merged to form the full BRT vector of the full MPG. This merged BRT vector can then be partitioned into segments, without needing to take any breaking distance or safety margins into account, as seen in Figure 5.1b. As the safety of a state is reached through the offset derived from the vehicle's current state, it is possible to break the merged BRT vector into smaller vectors by reordering data. Since the MPG only deems a state as safe if the vehicle is contained within the geofenced area, it is deemed safe to use these segments to provide information regarding the safety of a state. If no information about a state can be found, the state is automatically deemed to be unsafe.

As previously stated, the BRT vector is a four-dimensional matrix represented as a vector indexed by coordinates x, y, speed v and rotation θ, each position containing a single bit that tells if a state is safe or not. A segmentation of the full MPG needs to reorder the data in such a way that the x and y coordinates get grouped together, as to form smaller spatially coherent segments, similar to creating a number of smaller MPGs. This is done by splitting the BRT vector of the full MPG into many smaller BRT vectors depending on the x and y positions. Each small BRT vector gets a unique starting position and covers a distance S in both x and y's directions, meaning S becomes equal to the size of the segment.

A segment is thus created by taking the starting coordinates x and y of the segment, collecting the ranges x to $x + S$ and y to $y + S$, and then selecting all rotations and speeds belonging to every coordinate within the selected ranges. Below is an example line for creating a segment from the BRT vector of the full MPG in Python:

$$Segment = BRT_{vector}[x : x + S, y : y + S, :, :].$$

This line gets repeated over the entire BRT vector, looping over all combinations of x and y with a difference of S, effectively segmenting the full MPG. Each segment

does not contain any overlap, and when all of them are combined, they recreate the original BRT vector of the full MPG.

5.2 DALOSD outline

The proposed DALOSD takes a vehicle state (x, y, v, θ) as input and loads the needed segments. Coordinates x and y provide the vehicle's current position, v its current speed, and θ its current heading. Drawing inspiration from the work done regarding HD-maps and ontologies in [18], the scheme utilizes a spatial window to decide which segments to load, keep, and discard. The spatial window can be seen in Figure 5.2 and contains a triangle in front of the vehicle, deciding which segments to load, as well as a rectangle around the vehicle, deciding which segments to keep. Everything outside of the spatial window is eligible for removal. The size of the triangle is decided in part by a lookahead forward parameter L_f, which decides how far ahead in the spatial dimension the vehicle should load segments.

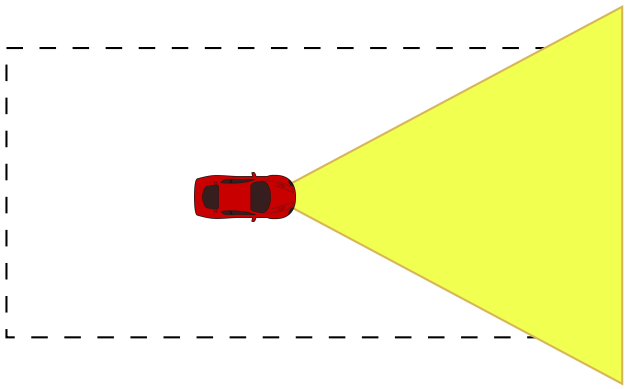


Figure 5.2: The spatial window, where the cone decides what segments to load and the rectangle what segments to keep.

The lookahead forward parameter L_f is dependent on both the maximum speed v_{max} of the geofence as well as how far ahead in time $t_{lookahead}$ DALOSD should look. It is thus defined as

$$L_f = v_{max} \cdot t_{lookahead}.$$

During operations, DALOSD periodically examines the loading triangle, request the needed segments from a server, and integrates these segments into its own storage. DALOSD periodically updates the internal storage to remove segments outside of the spatial window. The management of the storage is parallelized to allow for efficient data insertion and data lookup operations inside it. This means that a lookup operation determining if a vehicle state is safe or not will not need to wait for any ongoing data transfer operations to finish executing.

5.3 Modeling safety

Previous work has concluded the MPG will contain the vehicle within the geofenced area [11]. However, ensuring safe operations also includes preventing loading scheme stops, as previously mentioned in Section 4.4. Loading scheme stops may occur if necessary segments have not been successfully loaded to the vehicle. One potential cause of missing segments is the available bandwidth not being high enough to sustain the vehicles current speed. To combat this, DALOSD proposes two solutions, where the first implements *loading priority* to the requested segments, and the second imposes a *speed reduction* to the vehicle.

5.3.1 Safety by loading priority

The first proposed solution implements a loading priority, where the segments closest to the vehicle have the highest priority, as they are the segments the vehicle will reach first. After that, the segment straight ahead has the highest priority, and the segments placed to the sides of the triangle have the lowest priority. A visualization of the priority within the loading triangle can be seen in Figure 5.3.

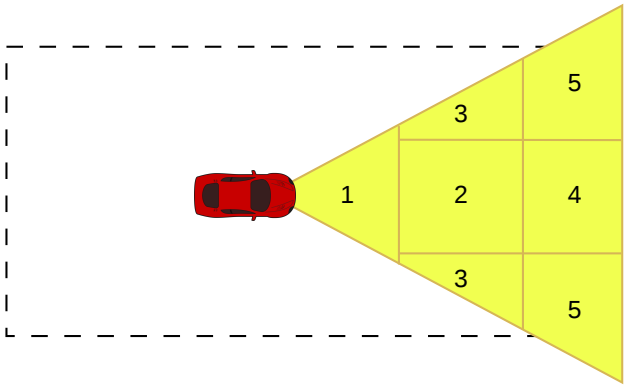


Figure 5.3: The spatial window with priority marked in the loading triangle.

5.3.2 Safety by speed reduction

The second proposed solution calculates a speed v_{limit}, which defines the *maximum speed where DALOSD estimates the necessary segments will get loaded in time* for operation. v_{limit} is derived from a lookahead available parameter L_a calculated from the number of segments currently loaded into the vehicle, as seen in Figure 5.4, where the orange cone shows how many segments have successfully been transmitted to the vehicle.

Since needing to emergency brake due to a missing segment is considered a failure of DALOSD, then v_{limit} can be likened to using the seatbelt in a car. It is good that it is there, but it should not have to be used during normal operations. Similarly to congestion control in a network protocol, it reduces throughput to provide stable operations. To avoid v_{limit} getting activated, it is necessary to increase the margin

by increasing the size of the spatial window. A larger spatial window will allow the scheme to look further into the future, giving the vehicle a larger amount of stored future possible positions. This becomes a tradeoff where safer operations lead to a higher memory consumption for DALOSD.

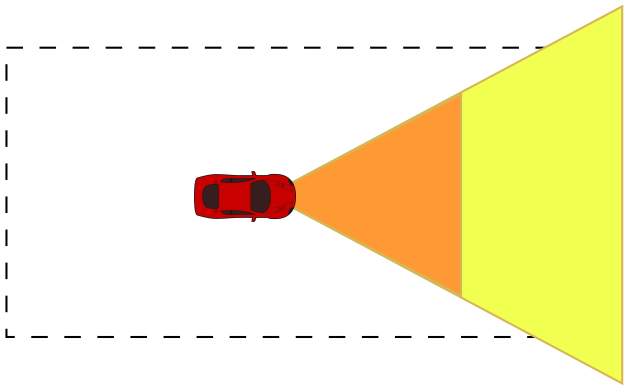


Figure 5.4: The spatial window, where the orange triangle shows how many segments have currently been loaded to the vehicle.

5.4 Collision detection within DALOSD

Deciding what segments overlap the spatial window is an essential part of DALOSD. This can efficiently be done through collision detection, where collisions between specific segments and the spatial window are examined. Section 2.4 presented two possible candidates for collision detection: MAD-C and SAT, and out of these two SAT was deemed to be the more appropriate solution.

Despite the efficiency of MAD-C for distributed sensor fusion, the specific problem trying to be solved by this thesis favors a more direct geometric approach. The collision is between statically uniformly defined segments in a grid and a distinct spatial window [15]. Applying MAD-C in this context would require modeling uniform grid segments as ellipsoids, which introduces geometric approximation errors and unnecessary computational overhead related to covariance matrix calculations.

Because the system requires exact intersection queries between a convex polygon and a tightly bounded subset of the axis-aligned grid of squares, SAT provides a superior solution. SAT guarantees exact collision detection between convex shapes. Adding a bounding box for broad-phase collision checking, with the box being at most one segment length outside, restricts candidates to check to a tight local area, while SAT as narrow-phase collision detection remains computationally lightweight, exact, and avoids the probabilistic abstraction necessary for unstructured point clouds [13].

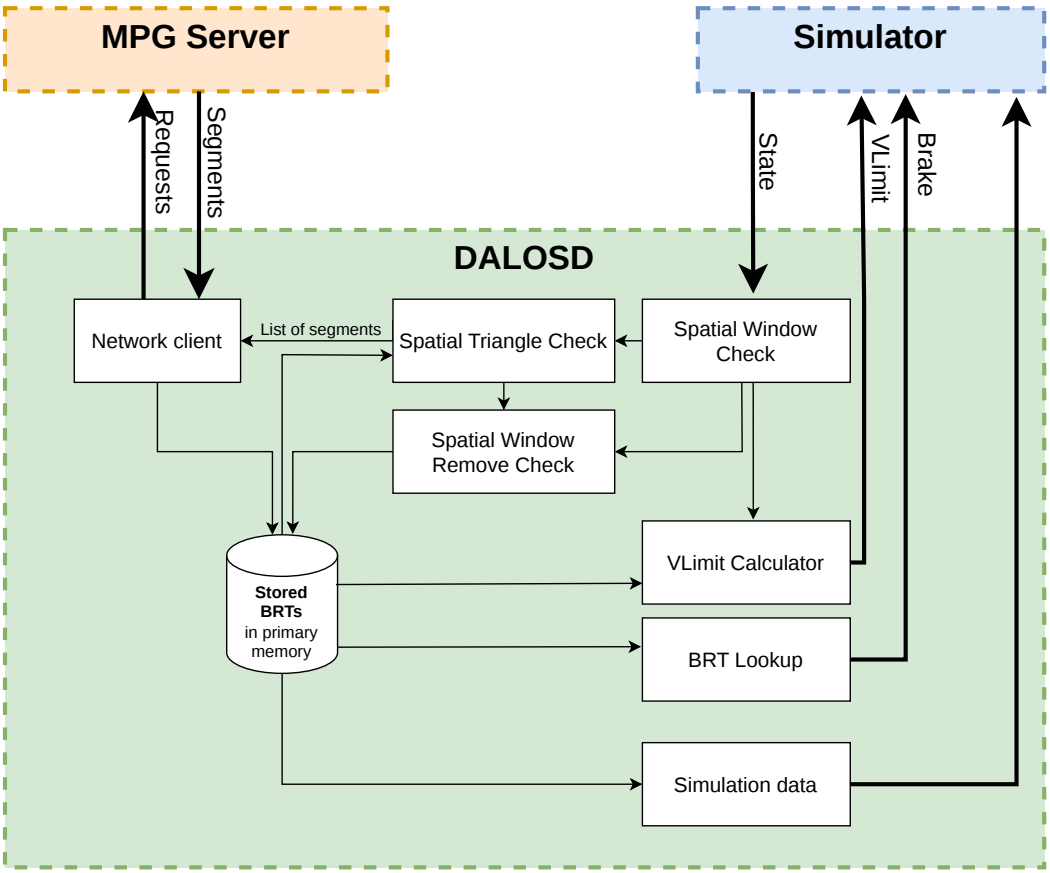


Figure 5.5: Overview of the different components of DALOSD. Green, orange, and blue boxes are the major components. Everything inside of the green box is done on the edge computer.

5.5 High level overview of DALOSD

DALOSD is divided into multiple smaller components, as shown in Figure 5.5. The major components are the MPG server, a simulator, and DALOSD itself. The server is an HTTP server that takes a list of segments and transfers them in the order they were requested. The simulator is a very simple simulator that will track a state for a car following the dubinscar4d model [5]. It has a communication interface that sends the current state to DALOSD, similar to what a GPS or location system would provide. Further, the simulator has added functionality to account for when the maximum speed is reduced and can visualize the state of the vehicle's storage of BRT vector segments. The last major component is DALOSD itself, which is built up from the following components: *StateCheck*, *VLimit calculator*, and *BRT Lookup*. Each part performs a key task of DALOSD, aiming at keeping safe operations both regarding keeping the vehicle within the geofence boundaries, as well as avoiding loading scheme stops due to missing data. Both the *Simulation data* and *Network client* are simple communication interfaces to the server and simulator.

5.5.1 Pseudo code for DALOSD

This section provides high-level pseudo-code of the algorithms within DALOSD.

SpatialWindowCheck is responsible for determining which BRT vector segments are required for continued operation based on the vehicle's current state. It periodically examines the spatial window loading triangle and identifies all intersecting segments not currently in local storage. The missing segments are forwarded to the *Network client*, which requests them from the remote server. The *SpatialWindowCheck* also ensures that the segments not within the spatial window are removed, as to free up memory. Its pseudo code can be seen in algorithm 1.

Algorithm 1 SpatialWindowCheck - Perform SATcollision check to find all segments within the two parts of the spatial window

```
1:  if newState != oldState && time since last check > constant_value then
2:
3:      ### Spatial triangle check ###
4:
5:      triangle ← Create triangle polygon
6:      BB_triangle ← Create bounding box for triangle + 1 segment size in each direction
7:      for segment in BB_triangle do
8:          if SATCollision(triangle, segment) then
9:              if segment not loaded then
10:                 Add segment to segments to be loaded
11:             end if
12:         end if
13:     end for
14:     segments to be loaded.sort(distance)
15:     if segments to be loaded != ∅ then
16:         Network Client(segments to be loaded)
17:     end if
18:
19:     ### Spatial windows replace check ###
20:
21:     BB_rectangle ← Create bounding box for rectangle
22:     for segment in Storage do
23:         inTriangle = (segment ∈ segments to be loaded)
24:         inRectangle = SATCollision(rectangle,segment)
25:         if !intTriangle or !inRectangle then
26:             Remove segment
27:         end if
28:     end for
29:     send segments to be loaded to vLimit
30: end if
```

VLimit calculator. The *VLimit calculator* determines whether segments are being loaded quickly enough to sustain the maximum speed permitted by the geofence

and is performed on the edge device. If the loading process falls behind, the *VLimit calculator* computes a reduced speed that can be maintained while still ensuring continued access to the required BRT vector segments. To estimate a sustainable speed, the algorithm measures how far ahead within the loading triangle segments have been continuously loaded at a given moment. This available look ahead distance is then divided by a slowdown period to produce a temporary velocity limit. To reduce the impact of fluctuations in loading scheme latency, the calculated v_{limit} is averaged over multiple iterations, creating a moving average. The pseudo code for *VLimit calculator* is shown in Algorithm 2, while Figure 5.4 illustrates how the available look ahead distance is determined.

Algorithm 2 VLimit Calculator - Calculate how far ahead has been loaded and figure out a safe speed to maintaine

```
persistent vlimitbuffer (ring buffer with 5 entries)
maxDistance ← 0
for Segment in SegmentsInSpatialWindow do
    if Segment is in Storage then
        distance = segement.distanceToCenterFromPos()
        maxDistance ← max(maxDistance, distance)
    else
        break
    end if
end for
vlimitbuffer.append(maxDistance/slowdownPeriod)
return mean(vlimitbuffer)
```

BRT Lookup. The *BRT Lookup* component performs the safety evaluation for the vehicle's current state. It first identifies the BRT vector segment corresponding to the vehicles current position. If the required segment is not available in local storage, the state is immediately considered unsafe, since the safety of the state cannot be verified, and a signal to emergency brake is issued. If the segment is available, the algorithm computes the corresponding index within the segment based on the vehicle's current state and a segment reference point taken from the specific segment. In the case of this DALOSD the reference point is the coordinate of the lower left corner. The stored value at this index is then evaluated to determine whether the state is safe for continued operation. If the lookup indicates that the state is unsafe, an emergency brake signal is issued, otherwise, the vehicle is allowed to continue operations. The pseudo code for the *BRT Lookup* is shown in Algorithm 3.

Algorithm 3 BRT Lookup - Check if a segment is loaded and if the state in the segment is safe and send emergency signal if not

```
1: currentSegment ← calculate segment containing value for current position
2: if currentSegment is not loaded then
3:     Send emergency brake signal
4: else
5:     i ← Calculate offset based on state and segment reference point
6:     if currentSegment[i] == 0 then
7:         Send emergency brake signal
8:     end if
9: end if
```

5.6 Segment size

Segment size is an important factor affecting the performance of DALOSD. Smaller segments increase the granularity of the spatial window, allowing the scheme to load data more precisely and thereby reducing unnecessary memory usage. However, smaller segments also increase the number of computations required to determine which segments intersect the spatial window. In addition, they increase the overhead associated with requesting, transmitting, and integrating segments into local storage.

Larger segments instead reduce the number of intersection checks and integration operations required during each iteration of DALOSD. They also reduce the total number of requests sent to the server. However, larger segments increase memory consumption, since portions of a segment may extend outside the spatial window and contain data unlikely to be used during operation.

Increasing the segment size therefore introduces the risk of over-fetching data while also increasing the transfer time for each individual segment, which in turn increases the loading scheme latency. Smaller segments reduce this effect, but at the cost of increased computational and communication overhead. Selecting an appropriate segment size thus becomes a balance between minimizing memory consumption, communication overhead, and lookup latency while still ensuring reliable access to the required MPG data.

5.7 Simulation of DALOSD

To visualize and evaluate the performance of DALOSD, a simple vehicle simulator was developed as part of the thesis. The decision to implement a custom simulator instead of using existing simulators, such as *CARLA* or *SUMO*, was motivated by the evaluation scope as well as avoiding integration complexity [22], [23]. The mentioned existing simulators provide extensive functionality for simulating vehicle dynamics, traffic flows, and sensors, but many of their features are not required for the evaluation of this work, whose focus is on network conditions and resource consumption by DALOSD. Detailed modeling of traffic flows, sensors, and vehicle

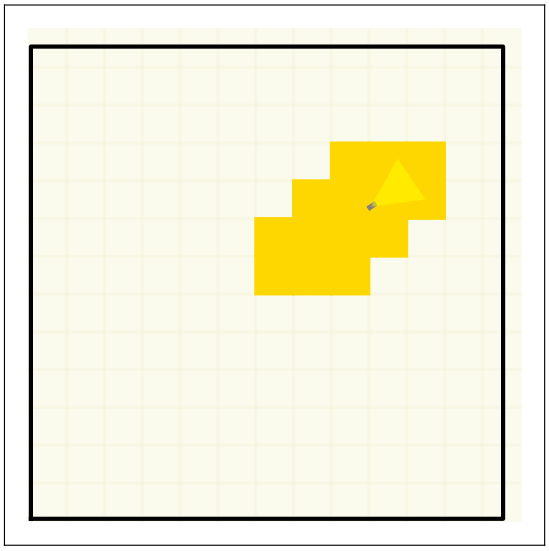


Figure 5.6: The simulation used for running DALOSD. The inner black line is the geofence area, and the pale yellow shows the area covered by the MPG, which extends beyond the geofence area to make every segment even in size for simplicity. The outer border is the border of the window for the simulation.

dynamics is not necessary to achieve this. The features inspired by CARLA and SUMO are the vehicle dynamics, but the implementation ideas come from the paper about SDM [5].

The simulator created for this thesis continuously updates a vehicle state while also providing a visual representation of DALOSD's internal storage state. One run from the vehicle simulator can be seen in Figure 5.6, where each tile represents a segment and different colors represent the segment's status compared to the vehicle's internal state. The loading cone is shown in yellow, whereas the darker yellow color represents segments loaded to storage.

With this simulator, it becomes possible to run DALOSD in real-time and utilize the system clock to calculate time differences between when requests are sent and received, allowing tracking of time in *ms*.

The vehicle in the simulation is initially manually operated by the user, but the simulator has an option of recording a taken path, which can be replayed at a later point to create reproducibility during tuning and evaluation of DALOSD.

5.8 Use cases

To tie usage of DALOSD to real world applications, two use cases have been identified. Each use case has different needs, thus producing different optimal settings

regarding segment sizes and resolution.

- **Case Quarry.** This use case is defined to be a quarry with autonomous mining vehicles. The maximum vehicle speed is low at 30 km/h, and the vehicles drive around freely within the area, not being confined to specific roads. For this use case, the resolution is set to four points per meter, the speed resolution is set to three different speeds, and the resolution for heading θ is set to 20.
- **Case Rural Road.** This use case is defined to be a rural road with a higher maximum speed of 90 km/h. The overall area is big, and cars are confined to specific roads. Since the vehicle travels at a greater speed, the number of segments needed to get loaded during specific time windows becomes greater. For this use case, resolution is set to two points per meter, the speed resolution is set to ten different speeds and the resolution for heading θ is set to 20.

The resolution is what is used when calculating the BRT vector and tells the SDM generator how many entries per meter that the BRT should have, as well as how many different speeds and headings each coordinate holds. This means that with a resolution of four points per meter, a speed resolution of three and a θ resolution of 20, there is an entry every 0.25 meter, each containing the safety statues of three different speed calculations for 20 different headings that can be looked up when using the BRT vector.

5.9 DALOSD source code

The source code, experiment scripts, and data-analysis notebook developed for this thesis are publicly available in the DALOSD repository [24].

6

Methodology for parameter tuning

DALOSD has a number of tunable parameters, such as spatial window size and segment size, which all affects the performance. To reduce the number of configurations used in evaluation, an experiment was designed to find different configurations targeting low memory consumption, low latency, and a low impact from v_{limit} on vehicle speed while minimizing the number of loading scheme stops.

The experiment had a vehicle follow a prerecorded path in a large area to ensure the experiment was reproducible and deterministic. The experiment was run on one computer running both the server and DALOSD client using *Netem* to control the network [25].

The path the vehicle followed during the experiments can be seen in Figure 6.1, and was made to traverse several curves to give an overview of how DALOSD performs. This path is built using two big curves, two medium curves, and one small curve, made to stress DALOSD and test how well configurations handle different curves.

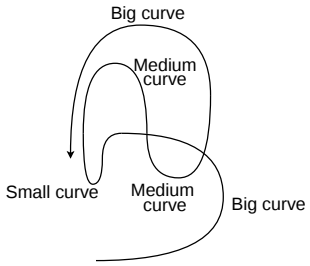


Figure 6.1: The path used when running the experiment for tuning DALOSD configurations.

6.1 Static parameters

The parameters that have been identified as affecting the performance of DALOSD are a combination of configuration and network parameters. These can be seen in Tables 6.1 and 6.2. Even though some of these naturally vary over time, such as *bandwidth, packet loss* and *Round Trip Time (RTT)*, they are treated as static during the experiments and evaluation.

6.1.1 Configuration parameters

The configuration parameters are a combination of parameters tied to the spatial window as well as the segment size. They can be seen in Table 6.1. The size of the spatial window combined with the size of the segment size has a direct impact on how much memory is needed for storing the MPG on the vehicle. For automated vehicles the recommended minimum time to look ahead is 7 to 8 seconds, and 11.8 seconds for a smoother experience. For ultimate comfort it is however best to utilize a 15 second time window [26]. For a vehicle driving at 30 km/h this corresponds to roughly 76, 98 and 125 meters distance respectively.

The table shows the range of values tested for each use case. For both use cases the experiment was run for five different values of L_f, five different values of ϕ and two different values of A. The Quarry case was run for six different segment sizes, and the Rural Road case was run for seven different segment sizes. This creates a total of 300 different test runs for use case Quarry, and 350 different test runs for use case Rural Road.

Table 6.1: Table of configuration parameters containing their information as well as their range for each use case.

Configuration parameters				
Name	**Symbol**	**Description**	**Range use case quarry**	**Range use case rural road**
Lookahead forward	L_f	The length of the loading triangle	67-209 m	200-600 m
Cone width angle	ϕ	The angle deciding the width of the triangle at its base	30-120 °	30-120 °
Area of box	A	The area of the spatial window box	2520-8400 m^2	23500-75000 m^2
Segment size	S	The segment size	10 - 100 m	5 - 50 m

6.1.2 Tying network parameters to real life conditions

To tie both the tuning experiments and evaluation experiments to real world conditions the network parameters are set from data provided by AstaZero[1], where they have measured representative network conditions on national road 40, a stretch of road that passes between Gothenburg and Borås in Sweden. A table containing information about the network parameters, as well as the measured range of values can be seen in Table 6.2.

Since challenge 1 presented in Section 4.4 focuses on avoiding loading scheme stops, it is important to focus on resilience against bad network conditions, as these can hinder the flow of data from server to vehicle. When tuning the configuration parameters, the experiment is therefore run with the worst measured conditions

[1]AstaZero : https://astazero.ri.se/

Table 6.2: Table of network parameters

Network parameters			
Name	**Symbol**	**Definition**	**Measured value(s)**
Round Trip Time	RTT	The time for data to go from source to destination and back again	20 - 1300 ms
Bandwidth	B	Available bandwidth	1-10 Mbps
Packet loss	ℓ	The average amount of packets lost in transit	3%

provided from the AstaZero national road 40 dataset for the Quarry use case, which gives a bandwidth of 1 Mbps, a packet loss of 3% and an RTT of 1300 ms.

Since it was estimated that the Rural Road use case would have a harder time to combat bad network conditions it was decided to use the mean values of the measured values of both bandwidth and packet loss, and the best possible value was used for RTT.

The experiments were conducted on a single computer running Ubuntu 24.04.3 LTS using gcc v.13.3.0, having a total of 16.0 GB RAM. The CPU was an Intel® Core™ i5-1035G4 which ran at 1.10 GHz, and has a total of 4 cores and 8 threads. Network communication was run locally and the tool *NetEm* provided network emulation in the experiment setup by limiting bandwidth, inserting delays, and simulating packet loss [25]. This makes it possible to reproduce different network conditions in a deterministic and repeatable manner during the experiments.

6.2 Assessing impact of DALOSD speed limitation

To measure the impact v_{limit} has on vehicle speed, a penalty system was developed. The penalty describes how much vehicle speed has been reduced over time due to segments not getting loaded fast enough to sustain the maximum speed allowed by the geofence. The penalty ranges from 0 to 5, where a penalty of 0 means that v_{limit} has no impact on the vehicle's speed, and a penalty of 5 means that v_{limit} is constantly set to 0, not allowing the vehicle to move.

The penalty is calculated by summarizing the time duration a certain v_{limit} speed has been active, where n is the number of different speeds measured during an evaluation round, and duration is the amount of time that speed was active. A weight is added reflecting how big of an impact v_{limit} has on the vehicle speed, with an increasing penalty for lower speeds. For example, if a car is traversing a road where the maximum speed is set to 90 km/h, then driving at 80 km/h has a small impact on traffic flow, whereas lowering the speed to 30 km/h has a major impact to the point where the low speed could be considered a safety risk. A v_{limit} speed that is the same as v_{max}, where v_{max} is the maximum speed allowed by the MPG,

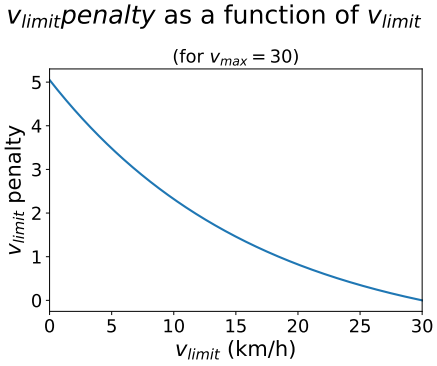


Figure 6.2: Graph showing v_{limit} penalty where the maximum speed is set to 30 km/h.

thus has no effect on the penalty. A v_{limit} speed that is in the vicinity of v_{max} has a low impact on the penalty and a v_{limit} that is close to 0 has a major impact on the penalty. The v_{limit} penalty is given by

$$v_{limit} \text{ penalty} = \frac{\sum_{i=1}^{n}(e^{1.8(1-\frac{v_{limit[i]}}{v_{max}})} - 1) \cdot duration_i}{\text{total time}}. \tag{6.1}$$

The v_{limit} penalty equation is constructed to give a big penalty to really low speeds and exponentially decrease the penalty for speeds closer to the maximum speed. To achieve this, we first normalize the speed $\frac{v_{limit[i]}}{v_{max}}$, then to get a decreasing curve, we subtract one from this. After, take this as an exponent to e to get a range from 6-1, downshift by taking all -1. The result of this becomes a penalty that punishes very low speeds. Multiply by the duration for the current v_{limit} and divide by the total time of the run, to weigh the score by the duration. Repeat for the value the v_{limit} has entered for a run to get the final v_{limit} penalty. A graph of the v_{limit} penalty where the maximum allowed speed is set to 30 km/h can be seen in Figure 6.2.

6.3 Selecting configurations to plot

When analyzing the results, a smaller number of configurations was selected from the total of 300 and 350 different configurations. For each segment size, four representative configurations were selected with different targets: **minimum amount of loading scheme stops**, **minimum memory consumption**, **minimum loading scheme latency**, and **minimum v_{limit} penalty**. These cases are selected to make the data observable and to try to extract the best possible configuration for spatial windows. In cases where multiple configurations achieved the same result, which occurred frequently for memory consumption, the configuration with the lower number of loading scheme stops was selected. In some instances, a single configuration was preferred for more than one of the evaluation targets.

The selected configurations for use case Quarry can be seen in Table 6.3, and the selected configurations for use case Rural Road can be seen in Table 6.4. Both

tables contain configuration names, spatial window parameters, and what target, or targets, each configuration minimizes.

Table 6.3: Table of the selected configurations for use case Quarry. L_f and ϕ is the length and angle of the loading triangle, and A shows the area of the rectangular spatial window. Target to minimize shows which metric was minimized for corresponding segment sizes S.

Selected configurations: use case Quarry		
Name	**Spatial window parameters**	**Target to minimize**
SW config Q1	L_f=67, ϕ=30, A=8400	Latency: S=10, 20, 30, 40, 50 No of stops: S=10, 30 v_{limit} penalty: S=10, 30
SW config Q2	L_f=67, ϕ=45, A=2520	Peak memory: S=20
SW config Q3	L_f=67, ϕ=90, A=2520	No of stops: S=100 v_{limit} penalty: S=100
SW config Q4	L_f=67, ϕ=90, A=8400	No of stops: S=20 v_{limit} penalty: S=20
SW config Q5	L_f=67, ϕ=120, A=8400	Latency: S=100 Peak memory: S=100
SW config Q6	L_f=84, ϕ=30, A=2520	Peak memory: S=10, 30, 50
SW config Q7	L_f=84, ϕ=90, A=8400	No of stops: S=40, v_{limit} penalty: S=40
SW config Q8	L_f=209, ϕ=30, A=2520	Peak memory: S=40
SW config Q9	L_f=209, ϕ=45, A=8400	No of stops: S=50 v_{limit} penalty: S=50

6.4 Parameter tuning results

This section presents the results for the parameter tuning. Memory consumption is analyzed with respect to how much memory is required to store the BRT vector segments on the vehicle, since this will allow running the scheme on a more constrained system or to increase margins of safety. The number of loading scheme stops are high for each configuration, which is due to the fact that DALOSD is run with the worst network parameters measured in the Asta Zero national road 40 dataset. The worst-case network parameters were chosen to ensure configurations that survive in the worst case, as determining the best configurations for the worst case ensures they will not collapse under poor network conditions. Even if the selected configurations might not be the best performers for good network conditions, it is more important to maintain stability in the bad cases to guarantee the system is as safe as possible.

Use case Quarry. Figure 6.3 shows the tuning results from use case Quarry. Figure 6.3a shows the v_{limit} penalty for different configurations and different segment sizes, and Figure 6.3b shows the number of loading scheme stops for different configurations and different segment sizes.

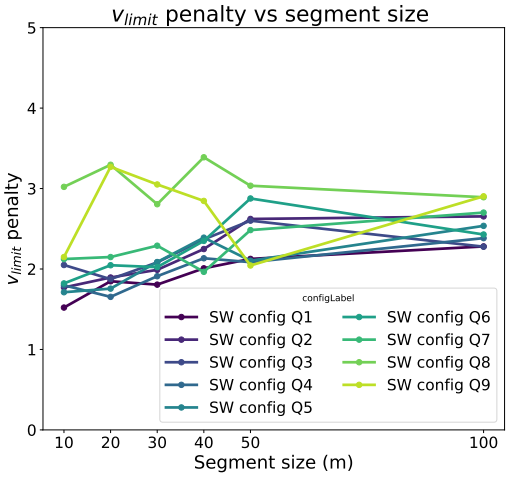
Table 6.4: Table of the selected configurations for use case Rural Road. L_f and ϕ is the length and angle of the loading triangle, and A shows the area of the rectangular spatial window. Target to minimize shows which metric was minimized for corresponding segment sizes S.

Selected configurations: use case Rural Road		
Name	**Spatial window parameters**	**Target to minimize**
SW config RR1	L_f=200, ϕ=30, A=23500	Latency: S=10, 100 Peak memory: S=20
SW config RR2	L_f=200, ϕ=30, A=75000	Latency: S= 5, 20, 30, 40, 50 No of stops: S=40 v_{limit} penalty: S=40
SW config RR3	L_f=200, ϕ=45, A=23500	Peak memory: S=40
SW config RR4	L_f=200, ϕ=45, A=75000	No of stops: S=20, 50 v_{limit} penalty: S=20, 50
SW config RR5	L_f=200, ϕ=120, A=23500	Peak memory: S=5
SW config RR6	L_f=250, ϕ=30, A=23500	Peak memory: S=10
SW config RR7	L_f=250, ϕ=30, A=75000	No of stops: S=30 v_{limit} penalty: S=30
SW config RR8	L_f=250, ϕ=120, A=75000	No of stops: S=100 v_{limit} penalty: S=100
SW config RR9	L_f=300, ϕ=30, A=23500	Peak memory: S=50
SW config RR10	L_f=600, ϕ=30, A=23500	Peak memory: S=30
SW config RR11	L_f=600, ϕ=30, A=75000	No of stops: S=5 v_{limit} penalty: S=5
SW config RR12	L_f=600, ϕ=70, A=23500	Peak memory: S=100
SW config RR13	L_f=600, ϕ=120, A=75000	No of stops: S=10 v_{limit} penalty: S=10

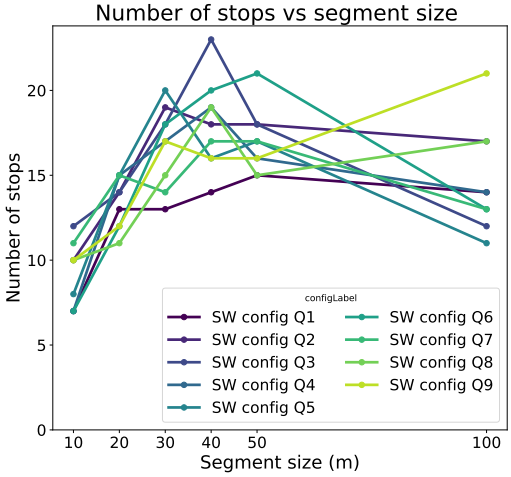
Use case Rural Road. Figure 6.4 shows the tuning results from use case Rural Road. Figure 6.4a shows the v_{limit} penalty for different configurations and different segment sizes, and Figure 6.4b shows the number of loading scheme stops for different configurations and different segment sizes.

6.4.1 Selecting configurations for evaluation

The experimental results in Section 6.4 show nine different spatial window configurations for the Quarry use case. For five different segment sizes, this gives a total of 54 possible configurations to evaluate. For the rural road case, the experiments in Section 6.4 showed 11 different configurations for 6 different segment sizes, which gives a total of 66 possible configurations to evaluate. To further narrow the number of configurations for evaluation, six configurations per use case are selected for further study.

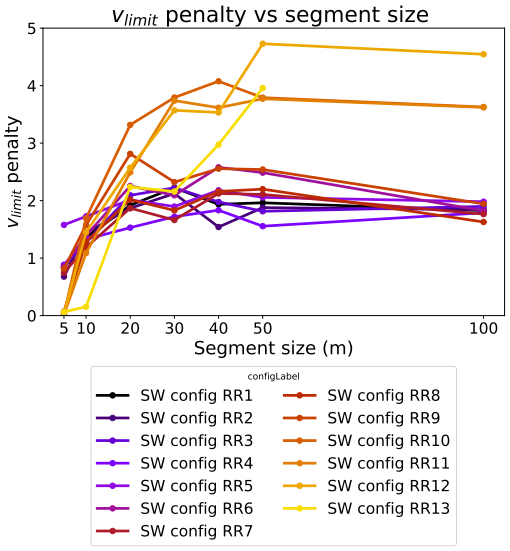


(a) v_{limit} penalty vs segment size.

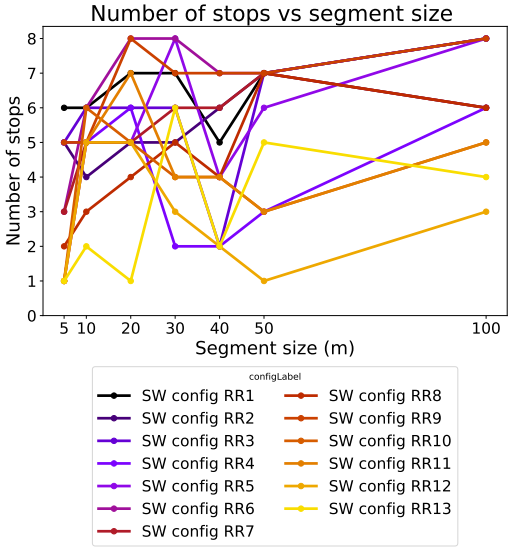


(b) Number of loading scheme stops vs segment size.

Figure 6.3: Parameter tuning results for use case Quarry.



(a) v_{limit} penalty vs segment size.



(b) Number of loading scheme stops vs segment size.

Figure 6.4: Parameter tuning results for use case Rural Road.

An important part of the decision-making in selecting configurations for evaluation was to select configurations with a low v_{limit} penalty and a low number of loading scheme stops. Generally, configurations with a low value of L_f seemed to perform the best. To allow for a richer evaluation, it was decided to select two configurations with higher values of L_f as well, which stems from the recommended distance from the paper by Sánchez et al. [26]. The configurations selected for evaluation can be seen in Table 6.5, where six configurations per use case has been selected.

Table 6.5: Table of configurations selected for further evaluation, with a general target to minimize the v_{limit} penalty and loading scheme stops. Column *SW config* corresponds to the previously presented configurations from Tables 6.3 and 6.4. L_f and ϕ is the length and angle of the loading triangle, A shows the area of the rectangular spatial window, and S the segment size.

Use case Quarry		
Name	**SW config**	**Configuration**
Eval config Q1	SW config Q1	L_f=67, ϕ=30, A=8400, S=10
Eval config Q2	SW config Q3	L_f=67, ϕ=90, A=2520, S=100
Eval config Q3	SW config Q4	L_f=67, ϕ=90, A=8400, S=10
Eval config Q4	SW config Q5	L_f=67, ϕ=120, A=8400, S=100
Eval config Q5	SW config Q7	L_f=84, ϕ=90, A=8400, S=10
Eval config Q6	SW config Q8	L_f=209, ϕ=30, A=2520, S=20
Use case Rural Road		
Name	**SW config**	**Configuration**
Eval config RR1	SW config RR4	L_f=200, ϕ=45, A=75000, S = 30
Eval config RR2	SW config RR6	L_f=200, ϕ=120, A=75000, S = 5
Eval config RR3	SW config RR6	L_f=200, ϕ=120, A=75000, S = 10
Eval config RR4	SW config RR11	L_f=600, ϕ=30, A=75000, S = 5
Eval config RR5	SW config RR11	L_f=600, ϕ=30, A=75000, S = 10
Eval config RR6	SW config RR13	L_f=600, ϕ=120, A=75000, S = 10

7

Evaluation

Accessing the MPG through dynamic loading enables usage of the MPG in resource constrained environments by reducing the memory usage. This does however come at a cost of reduced availability, since availability is dependent on the transfer of data over network. The reduced availability can lead to missing segments, speed limitations or loading scheme stops due to missing data. An evaluation of DALOSD therefore needs to explore the implications of this tradeoff by comparing vehicle operations while using the full MPG and when using the MPG through dynamic loading.

During evaluation the different configurations identified in the previous chapter are used. The evaluation targets memory consumption as well as access to the full MPG. This is done by performing a longer test run for each configuration and for each use case. The amount of memory used for storing the MPG on the vehicle is measured, as well as the number of loading scheme stops and the speed impact of v_{limit}.

The evaluation thus aims at exploring the following evaluation questions:

1. Does DALOSD provide access to the full MPG by
   (a) not having a major impact on the vehicles allowed speed, compared to using the full MPG onboard the vehicle.
   (b) having a low number of loading scheme stops, compared to using the full MPG onboard the vehicle.
2. Does DALOSD lower the amount of memory needed for storing the BRT vector of the MPG on the vehicle computer, compared to using the full BRT vector in the vehicle.

For both evaluation questions the baseline is the full MPG. For questions 1a and 1b using the full MPG would mean no impact on the vehicles speed and no loading scheme stops. For question 2 using the full MPG would mean loading the entire BRT vector onto the vehicle computer.

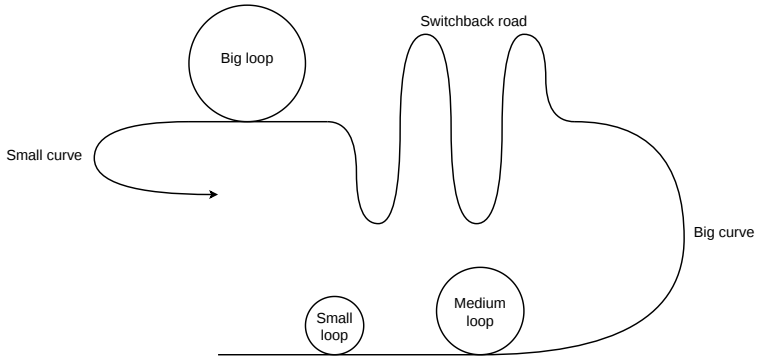


Figure 7.1: The driving scenario used for evaluating DALOSD.

7.1 Evaluation design

The evaluation was designed to examine whether the proposed DALOSD provides reliable access to the required BRT vector segments while reducing onboard memory consumption. To ensure reproducibility, all experiments were conducted using the simulator described in Section 5.7. The vehicle followed a prerecorded path shown in Figure 7.1, ensuring identical driving behavior across all test runs. The evaluation path was selected to include both straight segments, curves, and loops in order to give an overview of how DALOSD performs.

The path is designed to test DALOSD's ability to handle a varying number of segments needing to get loaded at any one point. The loops, the switchback road, and the curvature are designed to increase the number of segments the vehicle needs to fetch, which tests the responsiveness of DALOSD and how much prefetching of data is needed to avoid loading scheme stops. The evaluation thus investigates how DALOSD handles sharp turns, or if the sharp turns cause interruptions for vehicle operations. Further, not reusing the path from the parameter tuning experiments allows the evaluation to be conducted with no bias, since reusing the same path could give an unfair advantage to configurations that performed the best on that specific path.

The evaluation was conducted using the configurations from Table 6.5, which were identified during the parameter tuning process in Chapter 6. These configurations represent different trade-offs between low memory consumption and high data availability. During each test run the vehicle accelerates as fast as possible to the maximum speed allowed by the MPG, and then maintains the maximum speed allowed by v_{limit} throughout the test run, which in the best case scenario is the same as the maximum speed allowed by the MPG.

For each experiment run, the amount of memory needed for storing the MPG on the vehicle, the number of loading scheme stops, and the values and duration of each v_{limit} invocation are logged and analyzed. This data aims to answer evaluation questions 1 and 2 presented in the introduction of this chapter.

7.1.1 Network setup for evaluation

Network conditions were modelled using static bandwidth, latency, and packet loss. To evaluate how each network parameter impacts DALOSD, the network parameter being tested is varied, while the other network parameters are set to the best observed values from the data presented in Table 6.2, which reflect representative conditions taken from the AstaZero national road 40 dataset previously presented in Section 6.1.2. In the evaluation both bandwidth and RTT are investigated, while the packet loss is set to an optimal value of 0%.

Each configuration was run for optimal network conditions, and then run again with repeatedly worsening conditions for the selected network parameter to be investigated. This simulates issues DALOSD will have to effectively manage. The evaluation thereby assesses the reliability and robustness of the system under constrained network conditions.

In a realistic setting the RTT values would fluctuate over time. Even if the worst measured RTT value is 1300 ms, it is highly unlikely that the RTT would hold that value over long periods of time. A more realistic scenario would be a lower RTT value, and then sudden jumps to higher values. Still, it was decided to use constant values for the RTT to fully test what DALOSD would be capable of in the worst possible case, and investigate if it can withstand those conditions. To further enrich the evaluation an evaluation round with more realistic network conditions containing sudden spikes of poor network connectivity was conducted and presented in Section 7.2.4.

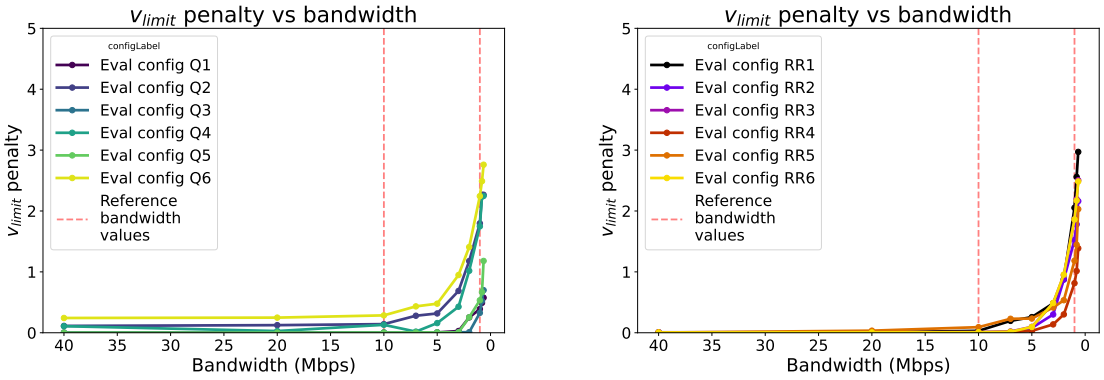
7.2 Evaluation of MPG access

This section evaluates how DALOSD affects access to the full MPG, thereby answering evaluation question 1. Each graph presented in this section has two red dashed lines representing the measured network values in the AstaZero national road 40 dataset presented in Table 6.2, to show what conditions DALOSD is expected to handle.

7.2.1 Evaluation of speed impact on vehicle operation

This section evaluates how DALOSD impacts vehicle operations through speed reduction by v_{limit}, measured by the v_{limit} penalty presented in Section 6.2, thereby answering evaluation question 1a. Since the full MPG has access at all times, its v_{limit} penalty will be a constant 0. It is thus not of interest to measure, and is not shown in the graphs.

Impact from bandwidth. The results of the speed impact for decreasing bandwidth can be seen in Figure 7.2, where Figure 7.2a shows the results for use case Quarry and Figure 7.2b shows the results for use case Rural Road. Both use cases exhibit similar curves, and for both use cases, the v_{limit} penalty keeps a mostly consistent value for high bandwidths. As soon as the lines reach the first dashed line, which represents the representative conditions measured in the AstaZero national



(a) Use case Quarry.

(b) Use case Rural Road.

Figure 7.2: v_{limit} penaltyagainst bandwidth for the two different use cases.

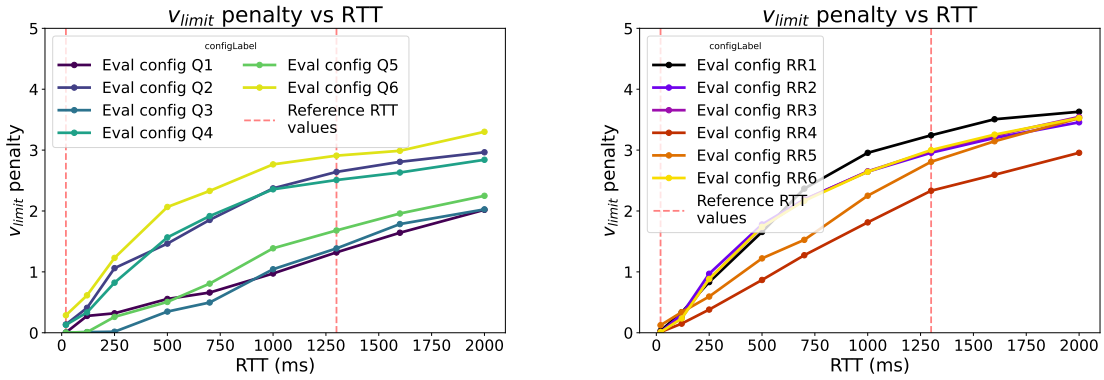
road 40 dataset, a degradation can be seen for a lot of configurations. For all configurations, the speed restriction increases exponentially as the network conditions worsen, after a breaking point is reached. Both use cases have configurations that perform steadily down to 7 Mbps, but after that the v_{limit} penalty gets more active for the Rural Road use case. The same thing happens in the Quarry use case, but first after reaching a bandwidth of 3 Mbps. Overall, the Quarry use case has several configurations that give good access to the MPG for bandwidths from 3 Mbps and above, and the Rural Road use case has one configuration that gives good access to the MPG for bandwidths from 7 Mbps and above.

Impact from RTT. The results of the speed impact for decreasing RTT can be seen in Figure 7.3, where Figure 7.3a shows the results for use case Quarry and 7.3b shows the results for use case Rural Road. From both graphs, it is clear that DALOSD's resilience against increasing RTT is a lot worse than its resilience against decreasing bandwidth. As in the case of the bandwidth evaluation, the Quarry use case has a smaller impact from v_{limit} penalty. This is probably due to the fact that the Quarry use case has a lower speed limit, giving DALOSD a lot more time to load BRT vector segments to fill up DALOSD triangle before v_{limit} becomes active.

7.2.2 Evaluation of access through loading scheme stops

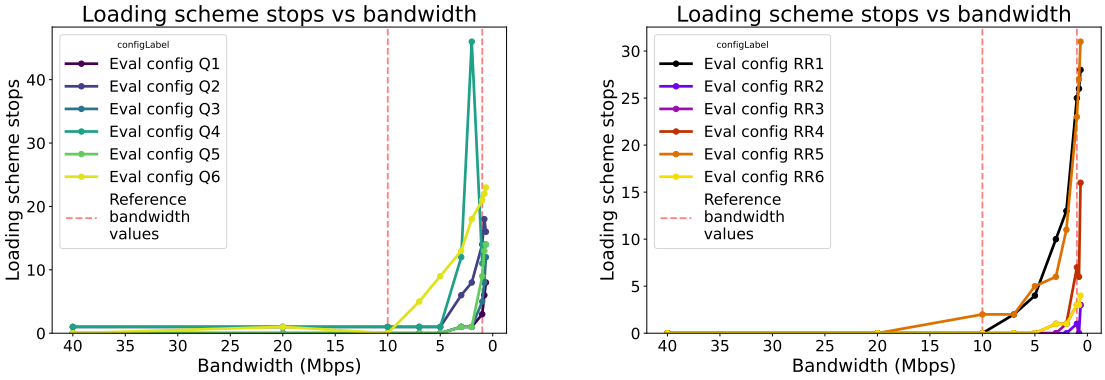
This section evaluates how dynamic loading affects access to the MPG in terms of loading scheme stops, thereby answering evaluation question 1b. Since the full MPG has access at all times, it will have 0 loading scheme stops and is not represented in the graphs.

Impact from bandwidth. Figure 7.4 shows the number of loading scheme stops measured against decreasing bandwidth for both use cases. Figure 7.4a shows the results for use case Quarry, where it is apparent that bandwidths lower than 5 Mbps has an adverse impact on each configuration's ability to avoid loading scheme stops. Figure 7.4b shows the results for use case Rural Road, where most configurations are majorly impacted by bandwidths lower than 7 Mbps. However, some configurations



(a) Use case Quarry. (b) Use case Rural Road.

Figure 7.3: v_{limit} penaltyagainst RTT for the two different use cases.

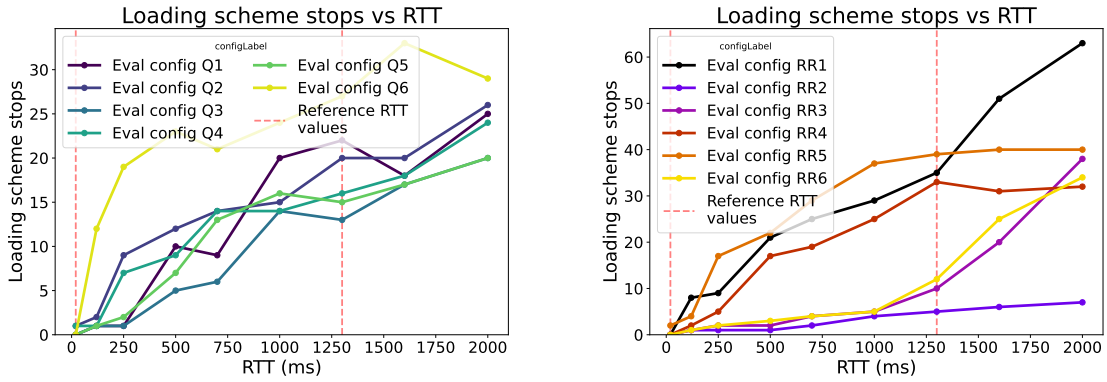


(a) Use case Quarry. (b) Use case Rural Road.

Figure 7.4: The measured number of loading scheme stops measured against decreasing bandwidth for both use cases.

perform surprisingly well even for low bandwidths, specifically Eval Config RR2 does not give a single loading scheme stop until the bandwidth drops to the lowest measured value from the AstaZero national road 40 dataset.

Impact from RTT. Figure 7.5 shows the number of loading scheme stops measured against increasing RTT for both configurations. These graphs show that the RTT has a major impact on DALOSD's ability to avoid unnecessary stops. Surprisingly, the use case Rural Road shown in Figure 7.5b has configurations that are more resilient against loading scheme stops than the configurations shown for use case Quarry in Figure 7.5a.



(a) Use case Quarry.

(b) Use case Rural Road.

Figure 7.5: The measured number of loading scheme stops measured against increasing RTT for both use cases.

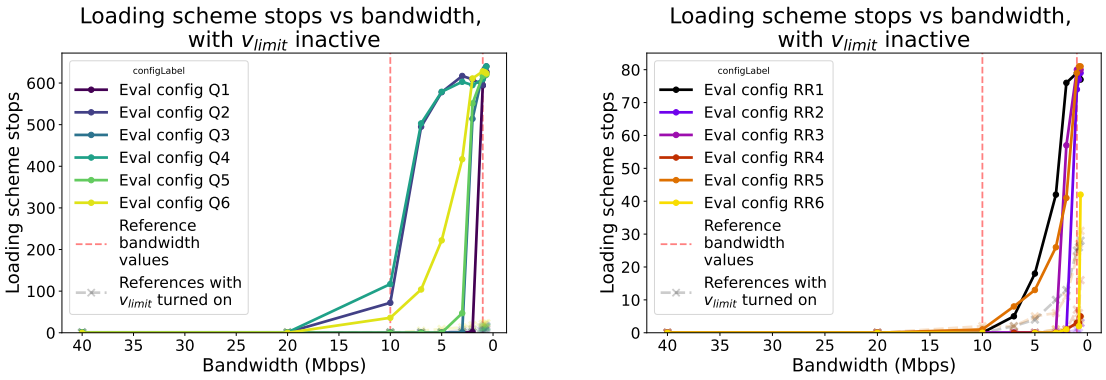
7.2.3 Effect of v_{limit} on the number of loading scheme stops

To evaluate the effect v_{limit} has on safety, tests were also conducted with v_{limit} inactivated. The results can be seen in Figures 7.6 and 7.7. To make it easier for the reader to understand the impact v_{limit} has on the number of loading scheme stops, the original data from Figures 7.4 and 7.5, depicting the number of loading scheme stops when v_{limit} is active, is plotted as transparent, dashed lines.

All graphs show a major impact from v_{limit} on the number of loading scheme stops, especially in use case Quarry, where the highest number of loading scheme stops in Figures 7.6a and 7.7a move from close to 40 with v_{limit} active, to around 600 with v_{limit} inactive. For use case Rural Road the difference is less stark. This is probably due to the differing speeds and braking distance in the two different use cases.

To avoid counting a loading scheme stop multiple times, the registered emergency brake signals are grouped together according to the respective braking distance of each use case. For example, if a signal to emergency brake is registered and counted as a loading scheme stop at coordinate (10.0, 11.0), there is no reason to register a new loading scheme stop from an emergency brake signal issued at coordinate (10.1, 11.0), since the vehicle is already expected to perform the emergency brake at that point. To avoid counting these multiple emergency brake signals as individual loading scheme stops it was decided to group all emergency brake signals that happen within the braking distance of the original signal into one loading scheme stop.

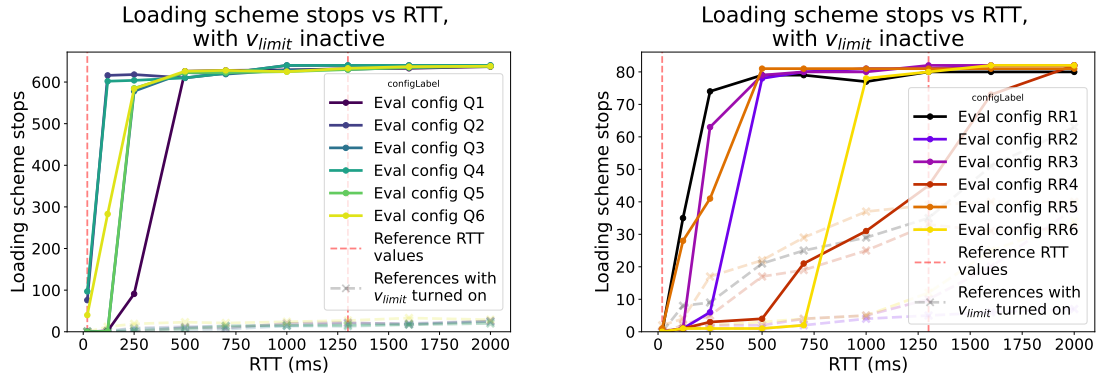
This grouping affects the number of loading scheme stops possible to count at a given distance. If a vehicle is driving at a high speed it is going to have a much higher braking distance than a car driving at a lower speed. If both vehicles are traversing the same distance, the vehicle with the lower speed is going to have a higher number of possible stops due to its grouping distances being much lower. Both use cases have roughly the same length of evaluation path, meaning that a use case with greater speed (and by extension a greater breaking distance), will have a lower number of counted loading scheme stops, as the emergency signals get



(a) Use case Quarry.

(b) Use case Rural Road.

Figure 7.6: The measured number of loading scheme stops plotted against decreasing bandwidth with v_{limit} inactivated.



(a) Use case Quarry.

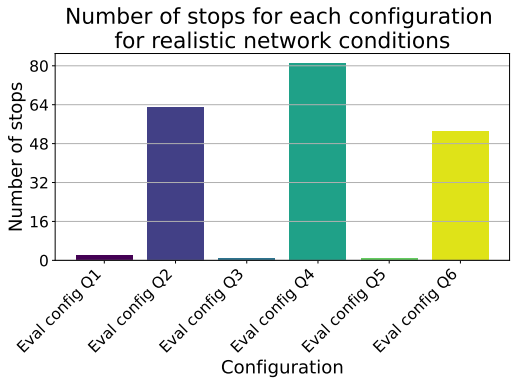
(b) Use case Rural Road.

Figure 7.7: The measured number of loading scheme stops plotted against increasing RTT with v_{limit} inactivated.

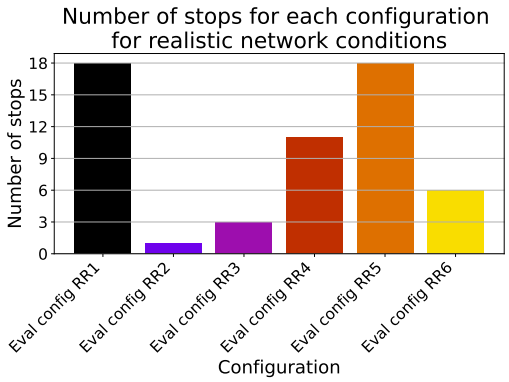
grouped over a greater distance. This is why the difference in the number of loading scheme stops is lower for use case Rural Road, than for use case Quarry when v_{limit} is activated or inactivated.

7.2.4 Realistic network conditions

To test DALOSD in more fair conditions for what it might face in a real world scenario, an evaluation round was performed with varying bandwidth, RTT and loss. The Varying conditions were modeled with a similar distribution as measured in the AstaZero network measurements. The results can be seen in Figures 7.8 and 7.9. From the test evaluating the number of loading scheme stops in Figure 7.8 it can be seen that DALOSD performs decently for three configurations for the Quarry case, and for two configurations for the Rural Road case. However, no tests yielded zero stops. A common denominator for the three use cases that performed the best in the Quarry case is that they all have a smaller segment size of 10.

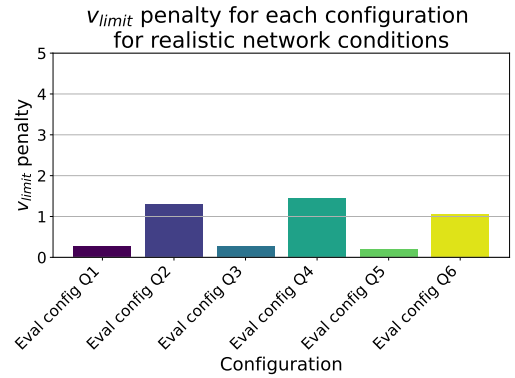


(a) Use case Quarry.

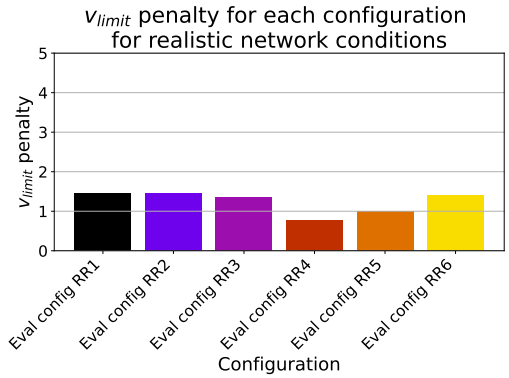


(b) Use case Rural Road.

Figure 7.8: The measured number of loading scheme stops for each configuration during more realistic network conditions.



(a) Use case Quarry.



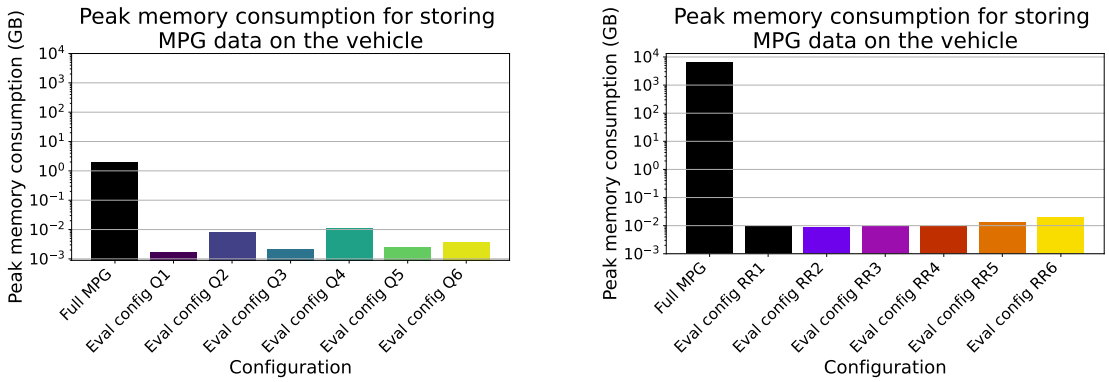
(b) Use case Rural Road.

Figure 7.9: v_{limit} penalty for each configuration during more realistic network conditions.

Evaluating how the implementation of v_{limit} performs for a realistic network simulation can be seen in Figure 7.9. In the Quarry use case, the v_{limit} penalty stays very low for a few cases and high for others, while staying consistently high for almost all cases in the Rural Road use case. These spikes are likely caused by the configurations inability to handle spikes in bad network conditions, where the spikes cause the v_{limit} to significantly halt the vehicles speed. To combat this DALOSD would need to load data further ahead, which would in turn stabilize the changes in v_{limit}.

7.3 Evaluation of memory consumption

Peak memory consumption for storing the BRT vector segments on the vehicle is measured for the different configurations of DALOSD, and compared to the BRT vector of the full MPG. This illustrates the difference in peak memory usage and answers evaluation question 2. The results can be seen in Figure 7.10, where Figure



(a) Use case Quarry.

(b) Use case Rural Road.

Figure 7.10: Graphs showing peak memory consumption for the full MPG and different configurations.

7.10a shows the results for use case Quarry and 7.10b the results for use case Rural Road. The graphs illustrate the difference in memory consumption between the left-most column, representing the memory needed for storing the full BRT vector of the MPG on the vehicle, compared to the rest of the columns, which represent different configurations of DALOSD.

For each use case DALOSD configurations are compared to a full MPG that matches the use case itself. For the Quarry use case the size of the BRT vector matches that of LKAB's quarry Svappavaara, in north of Sweden. For the Rural Road use case the size of the BRT vector matches that of an MPG covering the area between Gothenburg and Stockholm in Sweden.

To make all bars visible it was necessary to plot with a logarithmic scale on the y-axis, and to make an easier comparison between the two use cases the y-axis's of the two graphs were aligned with each other. From the two graphs, it is clear that DALOSD does have a major impact on memory consumption on the onboard computer of the vehicle.

The Quarry use case has the smaller difference of the two, but the difference is still big. The full MPG needs two GB for storing the BRT vector, and the best configuration of DALOSD only needs around two MB. The difference here is three orders of magnitudes. The worst Quarry configuration needed more memory for storage, but the consumption is still low at 11 MB. For the Rural Road use case the difference is almost 6 orders of magnitudes between the full MPG and DALOSD configurations. Here the full MPG needs 6.4 TB of storage, and each DALOSD configuration only needs around 11 MB.

8

Discussion

The evaluation results show that dynamically loading the BRT vector segments can significantly reduce onboard memory consumption while still allowing a level of continuous access to the full geofence during vehicle operation. This suggests that predictive geofencing based on Hamilton-Jacobi reachability analysis can be used on vehicle computers through segmented storage and dynamic loading.

The rest of this chapter discusses how the properties of interest of the thesis have been addressed and how its challenges have been met. Two of the properties of interest were those of balancing memory and latency, as well as identified trade-offs. These are discussed in section 8.1. The final property of interest was that of data availability, which also ties to the identified challenge of safety. This is discussed in section 8.2.

8.1 Tradeoffs

The thesis showed that many of the trade-offs between two variables also have an impact on another variable, resulting in the fact that one trade-off impacts other trade-offs. The trade-offs discussed in this chapter are the following:

- Computation vs Availability
- Memory vs Resilience to dropped connection
- Overfetching of data vs Segment size
- Network requirement vs Availability

Computation vs Availability. One important insight about the availability of the MPG was discovered during the tuning process through the dependency between the maximum speed of the system and the burden of computations. As the speed increases, the total area covered by the spatial window increases with it, as the length of the triangle needs to be increased to follow the recommended time windows to look ahead previously presented in Section 6.1.1. If a vehicle is expected to look seven seconds ahead into the future, the length of the loading triangle will move from 67 meters for the Quarry use case to 200 meters for the Rural Road use case. The increased size of the triangle will in turn increase the burden of calculating intersecting segments.

The mathematics for an isosceles triangle tells that the area of a triangle with constant apex will scale quadratically with the triangles height. This means that the number of collisions to check will increase by roughly nine times when changing speed from 30 km/h to 90 km/h. This means that segment sizes that work well for slow speeds can cause computation problems for use cases with higher speeds. If the system needs a long time for computing which segments overlap the loading triangle, it takes longer to send the requests to the server and the availability of the MPG is affected.

Memory vs Resilience to dropped connection. Lowering the amount of memory needed for storing the MPG comes at the trade-off of needing a certain quality in network connectivity. Either an increased RTT, a dropped bandwidth, or other interruptions to the connection can cause problems with access to the MPG. A solution to increase the resilience to deteriorating network conditions is to increase how far ahead the algorithm preloads segments of the MPG. This makes it possible to maintain a large buffer of loaded segments, allowing DALOSD to handle sudden drops in network connectivity, but comes at a cost of increased memory usage.

One major gain of utilizing DALOSD is the low memory usage compared to when having the entire BRT vector loaded in memory, as can be seen Figure 7.10. The memory usage through DALOSD goes way below the amount of memory a system today can be expected to have access to, which generally is multiple *GB* [27]. It is also important to note that the memory needed for DALOSD is not tied to the size of the full MPG. As previously shown in Figure 2.4b in Section 2.3, the size of the BRT vector of the MPG scales with the size of the geofence area, meaning the memory consumption of utilizing the full BRT vector in the vehicle will increase as the geofence area increases. The memory consumption from using the MPG through dynamic loading will, however, stay the same, no matter the size of the full BRT vector.

Furthermore, the system could adapt the lookahead distance based on network conditions. In poor or unstable network conditions, the spatial window could be increased to load data further ahead and improve resilience against sudden delays or drops in bandwidth. In areas with reliable connectivity, the spatial window could instead be reduced to save memory. The same principle could be applied when moving between servers or coverage areas, where the spatial window could be increased in advance to prepare for possible delays during the transition.

Overfetching of data vs Segment size. During tuning, it became clear that larger segment sizes cause the needed storage space for the BRT vector segments in the vehicle to increase. This is due to the spatial window only overlapping some segments by a tiny margin, but still needing to load these into storage, causing data to get loaded without ever being used. As a result of this, the amount of data loaded increases with segment size, which increases memory usage more than necessary and, in a limited bandwidth scenario, will cause problems with wasting the already limited bandwidth on transferring data that is not really needed. Essentially, larger segments lead to data outside of the spatial window being loaded due to alignment between the travel path and segments, and thus, for large segments, loading a lot of

data outside of the spatial window. As can be seen in Figure 6.4a and Figure 6.3a, the larger segment sizes generally tend to be worse than small segment sizes.

Network requirement vs Availability. The evaluation demonstrates that the reduction in memory usage comes at a cost of availability of the full MPG, thus introducing a dependency on network connectivity. DALOSD therefore needs to balance minimizing memory consumption against maintaining resilience against unfavorable network conditions.

Since the available bandwidth generally becomes the biggest limitation of the availability of the full MPG, as seen in the evaluation, a generally good idea for improving performance is to utilize as much of the other resources as possible to increase the utilization of the limited bandwidth. One possible way to do this is to increase memory consumption, but it is currently impossible to increase how far ahead the scheme looks due to the amount of bandwidth needed to further prefetch. To find a good configuration with a high MPG availability the general recommendation is to utilize small segment sizes, with a long cone length and small or medium width. This can be seen in Figures 7.3 and 7.5 where configurations such as *Eval config Q5* and *EvalConfig RR2* are some of the best performing configurations. This trade-off becomes especially important in a real deployment where the network will not operate with constant conditions: handovers, temporary outages, and changing conditions will create temporary but extreme conditions. To ensure that DALOSD can maintain safe operation under such conditions, it is necessary to identify the network requirements needed for sustained operation. These requirements would make it possible to determine when DALOSD can be used reliably, and when the system should instead transition to an alternative approach, initiate a safe shutdown, or safely transition to manual driving.

8.2 Discussion of challenges and research questions

A major challenge of the thesis was that of safety. Safety was introduced in Section 4.4 as avoidance of loading scheme stops, where a loading scheme stop was defined as a stop due to lacking access to a necessary part of the MPG in the onboard computer. Storing the full MPG on the vehicle provides full access at all times, providing a complete lack of loading scheme stops. Section 7.2 provided insights into how well DALOSD was able to meet the challenge of avoiding loading scheme stops under the evaluated conditions.

Another challenge was that of minimizing the needed resources by minimizing the amount of working memory needed to sustain DALOSD. The evaluated configuration that uses the largest amount of memory still only needed 24 MB of working memory. While 24 MB of memory is not a large amount of memory for a system today since they generally have multiple *GB* available [27], it leaves a lot of room to improve the scheme and ensures it stays as a subsystem that can be utilized on top of other techniques.

8.2.1 Safety of DALOSD

For the v_{limit} to be considered a safe option to use, it needs to change the speed of the system with a small enough change to ensure that it does not go slower than absolutely needed to avoid abruptly stopping. Speed changes relative to the allowed speed increase the risk of being involved in an accident and increase the risk of the accident being fatal. However, as long as the speed of the system remains within 0-20 km/h of the allowed speed, the effect on the risk of an accident remains virtually unchanged [28]. The 20 km/h difference means roughly 0.5 v_{limit} penalty for the 90 km/h maximum speed in the Rural Road use case, and for the Quarry case the 20 km/h difference becomes 2.32. Further, the likelihood of a sudden emergency brake causing a crash is higher than a slow and controlled slowdown, and should allow for other systems to take over control while the connection remains below the requirements of DALOSD.

As shown in Figures 7.4 and 7.5, six configurations, *Eval config RR2*, *Eval config RR3*, *Eval config RR6*, *Eval config Q1*, *Eval config Q3*, and *Eval config Q5*, successfully avoid all stops across both test types within parts of the evaluated representative network conditions, which demonstrates that operation without loading scheme stops is achievable. Some configurations also get close to no loading scheme stops even under poor network conditions, which could indicate that further study could push DALOSD to perform with zero stops within the entire interval of representative network conditions.

Among these configurations, *Eval config Q1*, *Eval config Q3*, and *Eval config Q5* stand out as the only alternatives that combine operation with no loading scheme stops with an acceptable v_{limit} penalty under relatively good network conditions. While *Eval config RR2*, *Eval config RR3*, and *Eval config RR6* also eliminate stops and maintain a good speed when testing 3 Mbps and above, they fail when the average network delay rises beyond 20 ms. This can be considered a failure to drive safely when the delay increase to what can be realistically measured. Consequently, configuration *Q1*, *Q3*, and *Q5* satisfy both criteria: avoiding stops while maintaining a speed reduction within an acceptable safety margin. Further, *Q1*, *Q3*, and *Q5* are also shown to be fully stop-free within a range of the representative network conditions, which means they would, with a high likelihood, survive in a dedicated network environment.

DALOSD can thus uphold safe operations, as defined by this thesis, for the Quarry use case for bandwidths higher than 5 Mbps and RTT values lower than 20 ms. For the Rural Road use case DALOSD can uphold safe operations for bandwidths higher than 3 Mpbs and RTT values lower than 20 ms. It is also worth reiterating what has already been mentioned in Section 4.5, that the MPG is reliant on precise positioning for safety to be guaranteed.

8.2.2 Answering the research questions

This section aims to answer the initial research questions presented in Section 1.

Question 1 asked if it is possible to use geofencing with predictive components in

vehicle computers, and this thesis has shown that it is indeed possible. Specifically, the Model Predictive Geofence has been used as an example in this work onboard the vehicle, showing how predictive components can be used onboard the vehicle without needing to perform predictive computations in real time.

Question 2 asked if it is possible to reliably have access to a geofence without keeping all geofence information onboard the vehicle, and this thesis has shown that it is indeed possible. This is done by utilizing DALOSD devised in this paper. There are some limitations to using this technique, such as having to assume the RTT never rises above 20 ms, and bandwidth never drops below 20 Mbps.

8.3 Expansion of studies

The thesis only evaluates one vehicle communicating with one server. In a real-world implementation, multiple vehicles would simultaneously request segments, potentially causing congestion, scheduling delays, and contention for bandwidth. While the proposed scheme of this thesis demonstrates feasibility in a controlled setting, additional work is still needed to determine how well the approach scales under real-world conditions.

To further investigate the impact of adding more vehicles to the equation one could re-examine the adaptive broadcast scheduling scheme proposed by Annavazzala et al. in [16], previously presented in Section 3. Another study that investigates what scheduling of requested data transfers from RSUs to vehicles can look like is performed by Zhang et al. in [29], where they investigate the implications of vehicles moving in and out of range of the RSU, and propose a several scheduling schemes to address the issue of bandwidth contention. Both Zhang et al. and Annavazzala et al. note that broadcasting information increases efficiency when transferring data from RSUs to several vehicles, however, neither of these studies take into account the case where it is necessary to successfully serve each data request, as is the case in DALOSD.

Another expansion of studies is that of adding more servers to the equation. Currently DALOSD only uses one server, but a real-world implementation would most likely use more than one server as the number of vehicles requesting data increases, especially if RSUs are used. Especially in the Rural Road use case it would be infeasible to use a single RSU to distribute the data of the MPG to multiple vehicles travelling from Gothenburg to Stockholm.

The thesis also simplifies several real-world conditions. The experiments assume static network conditions during the majority of the evaluation. In reality, the bandwidth would fluctuate, as well as packet loss and delay. The current evaluation therefore primarily demonstrates the feasibility of DALOSD under controlled conditions rather than guaranteeing real-world performance.

Further, the MPG is simplified to a large square, which makes the data as dense as possible. The SDM, however, introduced an optimization where some segments that are not within drivable range would not be generated, as there is no path to

those areas. This would probably improve the amount of data needed to transfer for certain cases, especially for medium segment sizes, since a result from this would be a drastic decrease in data needed to be loaded for certain use cases. Further, it would allow for the utilization of compression in a more realistic scenario.

Although DALOSD was created to work specifically for BRTs and utilize the SDM, DALOSD's core structure and ideas can be utilized in a wider context, such as loading HD maps or other cases where data needs to be loaded dynamically based on spatial positions.

Further, these experiments need to be redone with network configurations that use new and upcoming state-of-the-art networks dedicated to automated vehicles rather than general networks, such as *Ultra-Reliable Low-Latency Communication (URLLC)*, which is a method of communication that is part of 5G that provides low latency and high reliability with a guaranteed quality of service. This could address the unreliability and safety problems caused by RTT [30]. This would also be a more realistic scenario for the Quarry use case since such a site, or similarly small dedicated areas, most likely can support dedicated infrastructure that is faster than the multipurpose infrastructure the AstaZero dataset is measured on.

8.3.1 Expansion of DALOSD

In the current version of DALOSD, full BRT vector segments are transferred. It would, however, be possible to take inspiration from streaming when transferring BRT vector segments. The resolution could be lowered depending on both speed and rotation. If the vehicle is travelling at a certain speed in a certain direction, it might not need the part of the geofence that accounts for speeds much different than the current speed, or rotations in the opposite direction of the current rotation. This would not lower the memory consumption of the scheme in the vehicle, as the lookup function still requires states to be at a fixed offset, but it would lower the amount of data needed to be transferred over the network, allowing for continued operation under poor network conditions.

Another solution for lowering the amount of data to transfer is to divide each segment into multiple overlapping segments with differing speed limits. A segment could then be partitioned to have a version calculated for lower speeds, a version for medium speeds, and a version for high speeds. This partitions the segment into three different versions, and a vehicle can request which one it needs, depending on its current speed, effectively cutting the segment size to about the number of partitions one decides to create. A caveat to this is that its usability depends on the segment size, where this solution is only useful if the segment size stays relatively small. For example, if the segment size is 20 meters, it is reasonable to dynamically load depending on the current speed. If the segment size is instead more than 300 meters, the vehicle needs all available speed calculations, since a lot can happen speed-wise in 300 meters.

One other possible expansion is to explore other shapes for the spatial window, such as an oval or ellipse curve, or more likely to be successful, a bell-shaped curve whose shape is created based on the reasonable movements of the vehicle. Another

extension could be to move from using a fixed shape of the spatial window to instead have a dynamic spatial window, which changes its shape depending on the road geometry or the vehicles planned trajectory. While these solution increases the complexity of inspecting what is within the spatial window by moving from simple polygons, it might help solve the problem of unneeded data getting loaded. Further, when combined with other improvements, it might yield good results.

A further possible expansion could be to go beyond the concept of a spatial window altogether and preload data further ahead depending on the vehicles planned route. This could possibly allow a more narrow spatial window to reduce the amount of unnecessary data transfers, as the major decision factor on what segments to load now depends on the planned route instead, with the spatial window allowing the vehicle to deviate from its planned route without losing access to the MPG data.

Another potential improvement would be to utilize compression to minimize the amount of data needed to transfer over the network. The segments should be stored compressed on the server, and be decompressed onboard the vehicle. Which compression algorithm is best to utilize for minimizing the segment size needs to be studied further, but it can be heavy during compression since it is done once on the server side before storing the BRT vector segments, and it should be lightweight when decompressing onboard the vehicle.

Further, due to the BRT vector being split into multiple smaller parts which can easily be indexed due to each one being positioned by x and y coordinates, it is possible to calculate parts of the geofence dynamically. It will require a meshing algorithm to combine the old BRT vector with the new partial BRT vector but this can be streamlined and will require significantly less computations than recalculating the entire BRT vector. It might also be possible to dynamically do this in real time.

The second to last possible major improvement would be to exchange the server in DALOSD with a broadcast server. The current server establishes a connection with the vehicle, which provides the benefit of being able to use network protocols which guarantees the integrity of the transferred data. A broadcast server would instead broadcast segments without responses, which would greatly improve resilience against the high RTT at the cost of higher complexity when dealing with invalid packets. It would need to be studied further to determine how large of an improvement this would provide. A broadcast server would also provide the added benefit of making it possible to transfer the same segment to more than one vehicle at the same time.

The last, and probably one of the biggest improvements would be to change the loading priority of the scheme to, instead of using a static priority, use a dynamic priority order that uses the expected travel path to limit the number of loading scheme stops from data not being loaded. This could further be combined with the previously mentioned idea of a dynamic spatial window to be able to take advantage of information regarding expected or planned routes. This would increase the cost of computation within DALOSD, but since the network is generally the limitation of data availability, this is a trade-off that in most scenarios would provide a significant gain in performance.

8.3.2 Security

One topic this thesis has not covered is the domain of cybersecurity, where DALOSD has to be built to account for integrity protection, authentication, and resilience against different malicious actors. The problem arising from this is the increased overhead that different security protocols have to add to ensure non-tampering. Further expansion upon this is the privacy concern that comes from sending unencrypted data containing coordinates, currently creating a tracking system where malicious actors can follow either the vehicle or specific people. This can be circumvented with encryption, but this adds a lot of overhead and needs to be studied to make sure the overhead does not consume too much bandwidth.

9

Conclusion

DALOSD splits the BRT vector of the MPG into smaller parts, and then dynamically loads these to the vehicle depending on the vehicle's current position and what it is predicted to need in the near future. This allows the MPG to be used on the onboard computer of a vehicle without requiring the full BRT vector to be stored locally. As a result, the scheme reduces the memory requirements of utilizing the MPG onboard the vehicle, while preserving its functionality of containing the vehicle within the geofenced area.

DALOSD substantially reduced the memory consumption onboard the vehicle. In the Quarry use case, the full MPG required approximately 2 GB of onboard storage, while the evaluated DALOSD configurations only required roughly between 2 and 11 MB. In the Rural Road use case, the difference was even larger, with the full MPG requiring approximately 6.4 TB of onboard storage, compared to around only 11 MB for DALOSD configurations. This shows that dynamic loading does not only reduce memory usage, but does so in quantities large enough to change the feasibility of using the MPG onboard vehicle computers.

DALOSD has been evaluated on a single computer, and real-life network conditions have been emulated through the tool *netEm*. The evaluation performed in this thesis suggests that DALOSD would be a viable solution for using the MPG in real-life situations, but further testing and evaluation is needed to fully assess how it performs in real-life scenarios.

DALOSD works under ideal network conditions but is greatly limited in how well it performs if network conditions are not optimal, and is not recommended for use in its current form for high-speed traffic. It does however work satisfactory for lower speeds such as 30km/h. With a bit more tuning and improvements, it might be feasible to run the scheme for higher speeds.

The practical significance of this work is that it reduces one of the main barriers of deploying predictive geofencing in real vehicle systems, namely the need to store large precomputed reachability datasets onboard. By dynamically loading only the BRT vector segments that are relevant to the vehicles current and predicted future movement, the proposed scheme makes it possible to use Hamilton-Jacobi reachability analysis in real-time through the MPG on resource-constrained edge computers.

While this thesis has focused on making it possible to utilize the MPG in vehicle computers by lowering the necessary amount of memory needed through the use

of dynamic loading, the concepts explored in this thesis could extend beyond the MPG itself. The work may also be applicable to other systems requiring dynamic transfer of large spatial datasets to vehicles, such as HD maps or other predictive safety systems.

Overall, the results indicate that dynamic loading can help make predictive geofencing with reachability analysis feasible in practice, especially for lower speeds or environments where network performance can be monitored, controlled or predicted.

Bibliography

[1] F. Reclus and K. Drouard, "Geofencing for fleet freight management," in *2009 9th International Conference on Intelligent Transport Systems Telecommunications, (ITST)*, 2009, pp. 353–356. DOI: 10.1109/ITST.2009.5399328.

[2] Husqvarna Group, *Husqvarna epos: Exact positioning operating system*, Accessed: 2026-05-05, 2024. [Online]. Available: https://www.husqvarna.com/ca-en/learn-and-discover/husqvarna-epos/.

[3] P. D. Groves, *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems (2nd Edition)*. Artech House, 2013, ISBN: 978-1-60807-005-3. [Online]. Available: https://app.knovel.com/hotlink/toc/id:kpPGNSSIM9/principles-gnss-inertial/principles-gnss-inertial.

[4] K. Weibull, T. Kunclová, B. Lidestam, and E. Prytz, "Geofencing to prevent collisions in drivers interactions with emergency vehicles," *Transportation Research Interdisciplinary Perspectives*, vol. 28, p. 101 297, 2024, ISSN: 2590-1982. DOI: https://doi.org/10.1016/j.trip.2024.101297. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590198224002835.

[5] J. Hedlund and J. Färenmark, "Efficient backwards reachability computation for vehicle geofencing," Master's thesis, Chalmers University of Technology, 2025.

[6] M. Stevens and E. Atkins, "Layered geofences in complex airspace environments," in *2018 Aviation Technology, Integration, and Operations Conference*, Jun. 2018. DOI: 10.2514/6.2018-3348.

[7] P. Thomas and P. Sarhadi, "Geofencing motion planning for unmanned aerial vehicles using an anticipatory range control algorithm," *Machines*, vol. 12, p. 36, Jan. 2024. DOI: 10.3390/machines12010036.

[8] L. Cavanini, F. Ferracuti, G. Ippoliti, and G. Orlando, "Model predictive control for uav geofencing," in *2023 9th International Conference on Control, Decision and Information Technologies (CoDIT)*, 2023, pp. 573–578. DOI: 10.1109/CoDIT58514.2023.10284450.

[9] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin, "Hamilton-jacobi reachability: A brief overview and recent advances," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, 2017, pp. 2242–2253. DOI: 10.1109/CDC.2017.8263977.

[10] M. Chen and C. J. Tomlin, "Hamiltonjacobi reachability: Some recent theoretical advances and applications in unmanned airspace management," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. Vol-

ume 1, 2018, pp. 333–358, 2018, ISSN: 2573-5144. DOI: https://doi.org/10.1146/annurev-control-060117-104941. [Online]. Available: https://www.annualreviews.org/content/journals/10.1146/annurev-control-060117-104941.

[11] S. Thorén, L. Wikander, V. Jarlow, and T. Kero, "Model predictive geofencing for vehicle containment," in *2024 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2024, pp. 65–70. DOI: 10.1109/SMC54092.2024.10831796.

[12] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004, ch. 1.

[13] S. Gottschalk, M. C. Lin, and D. Manocha, "Obbtree: A hierarchical structure for rapid interference detection," in *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, 1st ed. New York, NY, USA: Association for Computing Machinery, 2023, ISBN: 9798400708978. [Online]. Available: https://doi.org/10.1145/3596711.3596791.

[14] C. Ericson, *Real-Time Collision Detection*. CRC Press, 2005. [Online]. Available: https://research.ebsco.com/linkprocessor/plink?id=a215a778-2bc4-3d16-960e-5abaf8d89012.

[15] A. Keramatian, V. Gulisano, M. Papatriantafilou, and P. Tsigas, "Mad-c: Multi-stage approximate distributed cluster-combining for obstacle detection and localization," *Journal of Parallel and Distributed Computing*, vol. 147, pp. 248–267, 2021, ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2020.08.013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731520303610.

[16] M. Annavazzala, S. D. Tambe, A. F. A., and B. R. Tamma, "Adaptive broadcast scheduling scheme for high-definition map tile dissemination in vehicular networks," in *2024 IEEE 99th Vehicular Technology Conference (VTC2024-Spring)*, 2024. DOI: 10.1109/VTC2024-Spring62846.2024.10683544.

[17] G. Elghazaly, R. Frank, S. Harvey, and S. Safko, "High-definition maps: Comprehensive survey, challenges, and future perspectives," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 4, pp. 527–550, 2023. DOI: 10.1109/OJITS.2023.3295502.

[18] H. Qiu, A. Ayara, and B. Glimm, "Ontology-based processing of dynamic maps in automated driving," in *International Conference on Knowledge Engineering and Ontology Development*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:227129433.

[19] S. O. Hansson, M.-A. Belin, and B. Lundgren, "Self-driving vehiclesan ethical overview," *Philosophy Technology*, vol. 34, Dec. 2021. DOI: 10.1007/s13347-021-00464-5.

[20] A. Abouaomar, S. Cherkaoui, Z. Mlika, and A. Kobbane, "Resource provisioning in edge computing for latency-sensitive applications," *IEEE Internet of Things Journal*, vol. 8, no. 14, pp. 11 088–11 099, 2021. DOI: 10.1109/JIOT.2021.3052082.

[21] Conference of European Directors of Roads (CEDR), "European sight distances in perspective (eusight): Driving experiment report (deliverable d5.2)," CEDR Transnational Road Research Programme (Call 2013: Safety), Amers-

foort, Netherlands, Deliverable Report D5.2, 2015, Funded by National Road Administrations of Germany, Ireland, Netherlands, and UK. [Online]. Available: https://www.cedr.eu/download/other_public_files/research_programme/call_2013/safety/eusight/Eusight_D5_Driving_experiment.pdf.

[22] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[23] P. Alvarez Lopez et al., *Simulation of urban mobility (sumo)*, version 1.27.0, May 2026. DOI: 10.5281/zenodo.20312733. [Online]. Available: https://doi.org/10.5281/zenodo.20312733.

[24] A. Brogeby and V. Ebbesson, *DALOSD: Dynamic adaptive loading of spatial datasets*, Source code, accessed 29 July 2026, 2026. [Online]. Available: https://github.com/DALOSD/DALOSD.

[25] V. Jarlow, R. Brenick, S. Bach, and M. Westling, "Scenario-driven evaluation of in-vehicle connectivity using a programmable connectivity controller," in *2026 IEEE 103rd Vehicular Technology Conference (VTC Spring)*, Accepted for publication, Nice, France: IEEE, 2026.

[26] M. M. Sánchez, C. van der Ploeg, R. Smit, J. Elfring, E. Silvas, and R. van de Molengraft, *Prediction horizon requirements for automated driving: Optimizing safety, comfort, and efficiency*, 2024. arXiv: 2402.03893 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2402.03893.

[27] N. Aliane, "A survey of open-source autonomous driving systems and their impact on research," *Information*, vol. 16, no. 4, 2025, ISSN: 2078-2489. DOI: 10.3390/info16040317. [Online]. Available: https://www.mdpi.com/2078-2489/16/4/317.

[28] J. Stuster, Z. Coffman, and D. Warren, "Synthesis of safety research related to speed and speed management," Federal Highway Administration, Tech. Rep. FHWA-RD-98-154, Jul. 1998, Accessed: 2026-05-28. [Online]. Available: https://www.fhwa.dot.gov/publications/research/safety/98154/.

[29] Y. Zhang, J. Zhao, and G. Cao, "On scheduling vehicle-roadside data access," in *Proceedings of the Fourth ACM International Workshop on Vehicular Ad Hoc Networks*, ser. VANET '07, Montreal, Quebec, Canada: Association for Computing Machinery, 2007, pp. 9–18, ISBN: 9781595937391. DOI: 10.1145/1287748.1287751. [Online]. Available: https://doi.org/10.1145/1287748.1287751.

[30] P. Popovski et al., *Wireless access in ultra-reliable low-latency communication (urllc)*, 2018. arXiv: 1810.06938 [cs.IT]. [Online]. Available: https://arxiv.org/abs/1810.06938.