CHALMERS

Optimization of driver model parameters for Long Combination Vehicles

Master's thesis in Complex Adaptive Systems

IVO BATKOVIC

Department of Applied Mechanics
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2016

MASTER'S THESIS IN COMPLEX ADAPTIVE SYSTEMS

Optimization of driver model parameters for Long Combination Vehicles

IVO BATKOVIC

Department of Applied Mechanics
Division of Vehicle Engineering and Adaptive Safety
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2016

Optimization of driver model parameters for Long Combination Vehicles
IVO BATKOVIC

Master's thesis 2016:68
ISSN 1652-8557
Department of Applied Mechanics
Division of Vehicle Engineering and Adaptive Safety
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Chalmers Reproservice
Göteborg, Sweden 2016

Optimization of driver model parameters for Long Combination Vehicles
Master's thesis in Complex Adaptive Systems
IVO BATKOVIC
Department of Applied Mechanics
Division of Vehicle Engineering and Adaptive Safety
Chalmers University of Technology

Abstract

Long combination vehicles (LCVs) are modular combination vehicles that are longer and heavier than what currently is allowed on European roads. These vehicle combinations have the potential to cut down overall transportation costs, but also carbon dioxide emissions. Countries such as Canada and Australia already have these truck combinations driving on their roads, and their use on European roads is expected to increase in the near future. The LCVs however bring an undesired effect of increased difficulty of maneuvering on roads and in traffic. Thus, their increased complexity calls for driver assisting systems. The development of these systems leads to promising ways of improving traffic flow and increase the use of long combination trucks on current roads.

In this thesis an existing framework for automated driving has been used which utilizes driver models for the navigation of the LCV. The trajectories of the LCV are generated using numerical simulations of non-linear *ordinary differential equations* (ODEs). The actuation requests, which are front wheel steering, propulsion and braking are calculated using driver models. Up until now the parameters of the driver models have been fixed, and were set by fitting data after an on-road study with professional truck drivers.

An approach for optimization of driver model parameters has been proposed in this thesis, which involves *genetic algorithms* (GAs) and *particle swarm optimization* (PSO). In order to achieve a real-time feasible implementation, the highly parallel nature of the GA and PSO are utilized. OpenCL was used as a platform to implement the parallel processes for both algorithms which allowed for code execution on either CPU or GPU.

Optimzation of the driver model parameters showed that it could for a given dangerous scenario successfully abort or complete a driving maneuver within given safety limits. The use of stochastic optimization proved to be reliable and solutions were often found 100% of the time. As for the real time aspect of the optimization, the results hinted that by lowering the number of iteration steps, optimizing code and upgrading the used hardware, a real time implementation is within reach.

Keywords: long combination vehicles, driver models, OpenCL, genetic algorithm, particle swarm optimization

Preface

In the beginning of 2016 I started walking down the road towards graduation with this thesis as a goal. Everything presented here is the product of these last six months consisting of countless hours, both day and night, of intensive work.

A large extent of the time was invested being a team member of the Chalmers Truck Team for the *Grand Cooperative Driving Challenge* (GCDC) 2016. Throughout the period of a couple of months we managed to develop, to a large extent, an autonomous truck with both lateral and longitudinal control. The truck successfully performed the two GCDC scenarios, which consisted of a platoon merge and an intersection scenario. Other than helping the team with various tasks, my main responsibility was designing and implementing a lane detection algorithm for the truck.

The opportunity of being part of this fantastic and hard working team has brought everlasting memories that will be cherished and remembered for life.

Acknowledgements

My deepest gratitude goes towards my supervisor and examiner Dr. Ola Benderius. I was privileged to receive his advice, guidance and support in all kinds of discussions. He always managed to set aside time for questions, especially for times with great difficulties. His detailed critical reviews left out no details and helped push the work towards the right path.

This thesis was conducted with the help from Volvo Group Trucks Technologies. Therefore I wish to thank my supervisors from Volvo. Peter Nilsson and Dr. Leo Laine gave valuable input throughout the entire work and showed extraordinary enthusiasm and willingness to help. Their stream of new ideas and suggestions for improvement helped me along the way, and for this I am ever grateful.

My final acknowledgments goes to all forms of input from the Chassis Strategy & Vehicle Analysis group at Volvo, but also the division of Vehicle Engineering and Autonomous Systems at Chalmers. As a thesis worker to be able to take full advantage of their facilities has been more than a luxury.

Contents

1 Introduction

Long combination vehicles (LCVs) are here defined as modular combination vehicles that are longer and possibly heavier than what is currently allowed in Europe today. These are envisioned to be used widely on the roads in Europe in the near future and have the potential to cut down on the overall cost for road transportation, but also decrease carbon dioxide emissions [1]. Countries such as Canada and Australia already have these truck combinations operating on their roads. Surely, there are several advantages of LCVs compared to conventional combinations, but there are also some drawbacks. Perhaps, the main one is that with LCVs the maneuver complexity increases, thus limiting their use in different types of roads and traffic. However, a possible solution for this problem is to incorporate relevant driver assisting systems. The development of these systems and merging it with autonomous functionalities in trucks leads to promising ways of improving the traffic flow and safety. For example, Volvo GTT has focused on generating maneuvers for semi-automated LCVs for highway driving with emphasis on driver acceptance. This framework includes highway maneuvers as well as decisions for maintaining lane and lane changes [2].

In an existing framework for driving automation [2], the trajectories which the LCV should follow are generated using numercial simulations of non-linear *ordinary differential equations* (ODEs). The actuation requests, which are front wheel steering, propulsion and braking are calculated using driver models. The subject and surrounding vehicles are modelled in a closed-loop with the driver models to verify the feasibility of the trajectory generated by the requested actuation. This is sometimes referred to as *traffic situation predictions* (TSPs). The TSPs are integrated using the Euler forward method for a given prediction horizon with inclusion of constraints with respect to vehicle motion, collision to surrounding traffic and road properties. For each update step, the framework calculates a maximum of three TSPs. These are connected to the current and nearest adjacent lanes. If a TSP is feasible then the execution of the corresponding actuation is valid. The decision making architecture later decides how the TSPs are to be used to generate traffic maneuvers.

Currently the framework uses fixed parameters for the driver models. These were set for smooth driving conditions and give no assurance of their feasibility in more critical driving situations. A hypothesis is that this could be solved using online updating of the driver model parameters. A possible approach would be to use online optimization, which also includes constraints of various kinds, such as: vehicle motion, road boundaries and surrounding vehicles. As was mentioned above, the TSPs were calculated using non-linear, and non-convex ODEs. This implies that non-linear optimization methods must be considered. Combining non-linear optimization with short solution times is in general often hard to achieve.

The aim with this thesis is to propose an optimization scheme which will, using the already existing framework, allow for the generation of vehicle actuation for highway maneuvers of LCVs in safety critical scenarios, such as abort of safety critical lane changes. Improving and adding this functionality to the framework will enable LCVs to integrate themselves safely and efficiently on the roads.

This masters thesis will focus on proposing and implementing an optimization scheme that will allow real-time use. The development of the optimization scheme will be done with the assistance of simulations using an existing framework with a one-track plant model. In order to achieve a real-time implementation, it is envisioned to use optimization methods of highly parallel nature, e.g. *particle swarm optimization* (PSO) and *genetic algorithms* (GAs). Having algorithms that exhibit parallel behaviour are excellent to combine with parallel computation using graphics hardware [3]. Particle swarm optimization is also known to perform well for optimizing problems with high dimensionality. Other derivative-free optimization algorithms are to be considered when constructing the thesis work [4].

The interface for the parallel algorithms and the *graphics processing unit* (GPU) will be *Open computing language* (OpenCL) [5]. It is an open standard GPU library for C/C++ and is compatible with many of the major GPU developers, e.g. Intel, AMD and Nvidia. An alternative to OpenCL is CUDA [6] which is developed and only supported by Nvidia. Using OpenCL also brings the possibility of porting parallelized code from the

GPU immediately to the *central processing unit* (CPU), which is useful for comparison of run times.

This thesis will therefore address and try to answer the following research questions:

- Is it possible to optimize the driver model parameters to handle safety critical situations. What are those limits?
- Will biologically inspired algorithms be suitable and robust enough, given their stochasticity, for this given problem?
- Can the optimization be done within a time frame that will allow for real time execution?

2

Background

For an autonomous vehicle to be able to drive safely by itself it is, in many cases, necessary for it to have a path that it plans to follow. The previous three decades have brought increased research efforts, both in academia and industry, towards developing driverless vehicle technologies. Recently a survey was conducted [7] of the current state-of-the-art planning and control algorithms. By selecting certain proposed algorithms the authors discussed and analyzed their strengths and weaknesses. The side-by-side comparisons allows for easier review and gain of insight for the effectiveness and limitations of the reviewed approaches.

2.1 Motion planning

In robotics a common problem that needs to be solved is the path planning problem. It refers to the problem of finding a path $\sigma(s) : [0, 1] \to \mathcal{X}$ in the configuration space $\mathcal{X}$ for a robot, or possibly a vehicle, that starts with an initial configuration and reaching a final configuration. Most commonly the solution to this problem is to break down the process into discrete motions that satisfy local constraints, such as feasible kinetic movement, and global constraints, such as avoiding collisions with obstacles and the given environment. The terms feasible and optimal may be used to describe a path depending whether the quality of the solution is considered. Feasible path planning may be viewed as determining a path that only satisfies some given problem constraints, while optimal path planning takes also into consideration some quality criterion subject to some constraints, such as finding the shortest path between the configurations, or minimizing or maximizing a certain criterion.

There are several methods available [7] for the generation of valid collision free paths for a robot to follow. These usually consist of graph search methods such as Dijkstra (see Fig. 2.1), A*, D* or probabilistic sampling methods such as the *randomly exploring random trees* (RRT). Other approaches when the problem involves structured environments, lane graphs and geometric methods are commonly used. Prior work [8] using graph-based methods for discrete state spaces worked well and resulted in fast algorithms. However, it tends to produce paths that are non-smooth and do generally not respect the non-holonomic constraints of a vehicle. A non-holonomic system refers to, in simple terms, a system whose states depend on the path it has taken in order to acheive the target goal [9]. In another work [10], the authors used a Hybrid A* algorithm to generate paths that were kinematically feasible and took into account the non-holonomic properties of the vehicle. A different solution [11] is to incorporate a planner which first planned a path and later was driven by a local obstacle avoidance controller. The authors proposed an extension to the well-known RRT algorithm to allow integration with a trajectory parameter space in order to efficiently detect collision-free paths, but also kinematically feasible paths for arbitrarily shaped vehicles. However, solving the problem on this level alone is not always enough since using only the spatial information gives no guarantee how well this path will work after a time T in the future.

Path planning algorithms as those mentioned previously in this section, do not work very well in dynamic environments, since they lack forms of temporal information. Therefore it might be more suitable to formulate the motion of the robotic vehicle in a trajectory planning framework. That is to express the solution of the problem as a path, which is time-dependant. For instance a function with a time dependancy $\pi(t) : [0, T] \to \mathcal{X}$ which describes the configuration (such as position, velocity and heading) for the vehicle in time. With this formulation, it is possible to generate trajectories that react against a dynamic environment and it may plan around appearing, or disappearing, obstacles. In related work [12], the authors presented a strategy for trajectory planning and incorporated it in a vehicle that drove 103 km fully autonomously of the Bertha-Benz-Memorial-Route. Their solution used a local, continuous method that was derived from a variational formulation. Using a constrained objective function, the solution trajectory is obtained on the constrained extremum of said function. They considered both static and dynamic obstacle constraints and incorporated it within the system.

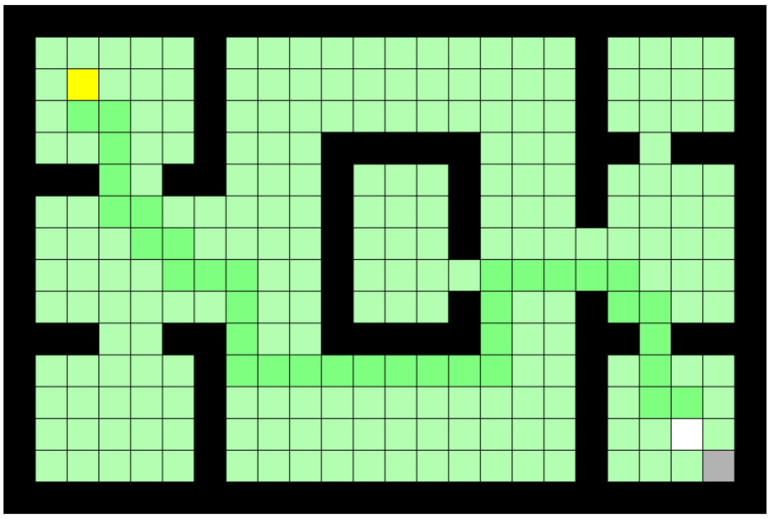


Figure 2.1: *Solution of a path problem using the Dijkstra algorithm. The yellow segment shows the initial position, and the white segment shows the target position. The surrounding area highlighted in light green denotes the explored environment, while the dark green segments show the optimal path towards the goal. The black segments are obstacles, or walls, and the single gray segment is a node never being considered.*

Even if the algorithms can react given a certain time step to a change in the environment, it is not always guaranteed to be able to handle the situation well. In order to get around this problem, predictions of the environment obstacles might be a necessity for certain problems.

Since the LCV kinematics is non-holnomic, planning and control methods that do not update their generated trajectories, namely the ones that do not predict future events, are not considered optimal for LCVs to use. With the increased vehicle complexity of a LCV the maneuvers need to be well planned. For instance, a car driver does not need to pay much attention of the surrounding vehicle when entering a corner. As for LCVs, or long trucks in general, there are the elements of needing to brake before a corner, or take wider corners to prevent its trailers from leaving the road or entering the oncoming lane. Therefore, the need of predictive techniques has emerged. Usually one makes a distinction between two kinds of predictive techniques. The first uses online optimization [13] whereas the other uses random sampling based methods such as the rapidly exploring random trees algorithm [14]. However, neither methods do not necessarily provide obvious ways how to choose constraints, parameter values or objectives to achieve user acceptable driving behaviour.

2.2 Online computation

In order to have a real system, such as a vehicle driving by itself, the solutions for the motion planning problem are required to be fast, reliable, and reactive. A delayed system will not be able to react properly to changes in the environment, and without the ability to react fast enough it might inevitably lead to great risks such as accidents or fatalities if used on the roads. The following will mention a few solutions that have been proven, in simulation, fast and reliable to generate trajectories for LCVs on highways.

As it has been stated in the section above, path planning in general is a field that is being researched quite extensively and there are multiple solutions to the problem. When it comes to LCVs and the problem discussed here, it has been shown [15] that this problem can be solved by using optimization-based receding horizon control. Formulating a trajectory generation problem as a *nonlinear program* (NLP), while also using state-of-the-art solvers, the trajectory generation problem can be solved in real-time. However, it showed that using kinematic vehicle models with low complexity could affect the stability of the performance.

Other work [16] proposed a nonlinear receding horizon trajectory generator for highway driving of an A-double combination LCV. With the use of mathematical solution strategies the problem was formulated into an *optimal control problem* (OCP) to define the open-loop constrained trajectories. For the solution technique a direct multiple-shooting solution with a piecewise constant control parametrization was used to obtain a nonlinear program. Then the ACADO Toolkit [17] was used to implement the so called Real-Time Iteration algorithm. By combining these methods it was shown that the system achieved stability and solution times were in fact real-time feasible. However, the solutions obtained by the system did not reflect well how real humans drive. This leads to the question whether a system like this will have a high driver acceptance when introducing automated driving functionalities for LCVs.

A solution that focuses on systems that target high driver acceptance for LCVs has previously been proposed [18]. The authors hypothesized that a high driver acceptance is achievable by applying driver models inspired by human perception. The proposed solution is based on *driver model control* (DMC) theory to direct the vehicle guidance. The method is based on human perception and optical flow theory and utilizes optical variables, so called aim-points, to steer and direct the LCV. Thus the trajectory generation follows implicitly with the proposed method. Numerical simulations proved that it was in fact possible to generate safe lane changing maneuvers with combined braking and steering. However, since this method does not utilize any optimization, parameter settings is a limiting factor for different driving conditions and scenarios.

Very similar methods [16, 18] were compared [2] for automated lane change maneuvers on highways for LCVs. Both methods have traffic situation predictions where motion variable constraints and actuation requests for steering, propulsion and braking are included. A simulation environment with a high-fidelity vehicle plant model of the LCV and models of surrounding vehicles was used in the comparison of the automated driving approaches. The results showed in general that the non-linear *model predictive control* (MPC) had shorter times for lane changes as well as lower magnitudes of lateral and longitudinal accelerations. However, the longitudinal vehicle speed of the non-linear MPC gave rise to unnecessary variation compared to the driver model control approach.

The mentioned methods above are proven to be robust and work well for highway driving. Unfortunately, the nonlinear receding horizon optimization [16] is computationally intensive, whereas the use of driver models [18] depends on parameter settings that are fixed. Reduction of solution times, but also updating of driver model parameters, could be improved by introducing parallel computation.

2.2.1 Parallel computation

A conventional way to speed up computation time is to find and reduce bottlenecks in the program code. This might be solved by either writing better and faster algorithms or finding processes which are independent of each other and try to run them in parallel. Over the last few years, increasing effort has been put into moving from a sequential to a parallel programming paradigm. An example of that has been the increased development in various famous frameworks such as CUDA, OpenCL and *Open Multi Processing* (OpenMP) [19]. All of these frameworks offer easy use of writing parallel executable code. OpenMP is allows for parallel code execution on CPU whereas CUDA is used for code execution on the GPU.

The open standard OpenCL, however, allows for execution on both CPU and GPU. Another positive feature is its compatibility on heterogeneous systems, which allows execution on devices made by such as Intel, AMD and Nvidia. Constant development of the frameworks is enabling more users to easily optimize and parallelize their code using the parallel computational power of GPUs using high level programming. With the advance in both CPU and GPU technology parallelism is becoming the more frequent and obvious choice.

For instance, work has been done where a combination of MPC and PSO with an open-loop simulation to generate swing-up trajectories for an under-actuated robotic arm in real-time [3]. The PSO is used as an optimizer for the MPC for a nonlinear system in real-time while considering constraints and handles modeling uncertainties well. The problem relates well with the open-loop approach used for the actuation and control for LCVs [16, 18, 2] and indicates to be feasible for a similar implementation approach.

Biologically inspired methods show great possibilities to solve problems with high complexity. An example is the work [20], where the authors used a GA and PSO to compute feasible and quasi-optimal trajectories in complex 3D environment for unmanned arial vehicles. All while taking into consideration the dynamic properties of the vehicle. They created paths using different line segments, vertical helices and circular arcs. Given the parallel nature of the used algorithms, it was possible with parallel programming to acheive a speed-up of factor 7.3 and allowing real-time performance with standard off-the-shelf commercial multicore CPUs.

3

Method

This chapter will bring to light the methods being utilized in this thesis. Starting from the existing simulating framework and then moving on to the theory behind the optimization. Lastly the implementation will be presented together with the inteded simulating scenario.

3.1 Existing framework for automated driving

This thesis has had the advantage to utilize and build upon an existing simulation framework for automated driving which was provided by VGTT. The following sections in this chapter will therefore describe how the most important parts of the framework are set up and how they intend to work.

3.1.1 Driver models

The driver model consists of a lateral and longitudinal control for the motion guidance of the LCV. Since the aim is to have a system that offers high driver acceptance one option is to have a driver model derived from human perception. In previous work [18], the authors used optic information for longitudinal and lateral control of a LCV.

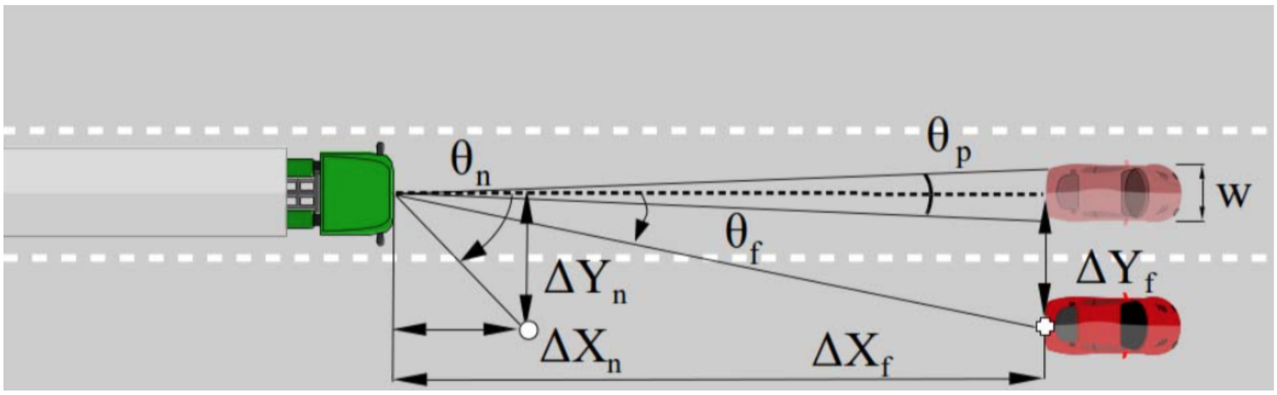


Figure 3.1: *Illustration over the optical variables used for the control algorithm. Figure taken from [2].*

Extensive research has been made on the topic of using optical information. The lateral control that will be used is the two-point Salvucci-Gray model [21], which takes into account the way humans drive. It demonstrates properties such as: curve negotiations, corrective steering after a lateral drift, lane changing and individual differences.

A two-point visual control model of steering The proposed model [21] is based on two perceived visual points, namely a 'near point' and a 'far point'. Thus, instead of utilizing only a single variable θ, the model takes into account θ_n representing the heading angle towards the near point and θ_f representing the heading angle towards the far point. An illustration of the concept is shown in Fig. 3.1, with the definitions of θ_n and θ_f shown. Based on this a control equation is derived as follows

$$\dot{\delta}_{ref} = k_f\dot{\theta}_f + k_n\dot{\theta}_n + k_I\theta_n. \tag{3.1}$$

The first two terms in Eq. 3.1 looks at the change in the far-point direction $\dot{\theta}_f$ and $\dot{\theta}_n$. The last term represents the visual direction to near point θ_n since it is the best relating term for the lateral positional error of the vehicle. By using this control algorithm the model will adjust its steering while trying to maintain the following criteria:

i Maintain a stable far-point, $\dot{\theta}_f = 0$.

ii Maintain a stable near-point, $\dot{\theta}_n = 0$.

iii Maintain a near point centered on the middle of the lane, $\theta_n = 0$.

The benefit with this control is that both the far point and near point do not have to be locked onto the center of a lane, but can be set to any visual point, thus creating lateral offsets from the center lane. The model was used by Markkula *et al.*, [22], and was performing very well in the modeling of human evasive maneuvering

Longitudinal control The longitudinal control used by the framework is based on information about time-to-collision of visual control [23]. It is a feed-forward controller that is updated iteratively using a reference acceleration $a_{x,ref}$ which can be formulated as

$$\begin{aligned} a_{x,ref} &= (1 + \dot{\tau}_m) \cdot \frac{\Delta v_x^2}{\Delta X_f - v_0 \cdot t_h} \\ \underline{a}_x &\leq a_{x,ref} \leq \bar{a}_x, \\ \underline{j}_x &\leq j_x \leq \bar{j}_x \end{aligned} \tag{3.2}$$

where Δv_x is the speed difference between the first axle of the LCV and the lead vehicle, ΔX_f is the far point distance and t_h is the desired final temporal headway towards the lead vehicle, $\dot{\tau}_m$ is the approximation of the time derivative of time-to-collision, and j_x is the longitudinal jerk.

With this model, the breaking and propulsion are initialized by utilizing margin values of the optical expansion rate $\dot{\theta}_{p,m}$ and the temporal headway to the leading vehicle $t_{hl,m}$.

$$\begin{aligned} \dot{\theta}_{p,m} &\leq \dot{\theta}_p \leq -\dot{\theta}_{p,m} \\ t_{hl,m} + \varepsilon_t &\leq t_h \leq t_{hl,m}. \end{aligned} \tag{3.3}$$

Here ε_t is a small constant parameter. The expansion rate $\dot{\theta}_p$ and the temporal headway t_h are calculated as

$$\begin{aligned} \dot{\theta}_p &= \frac{-4 \cdot w \cdot \Delta v_x}{w^2 + 4 \cdot \Delta X_f^2} \\ t_h &= \frac{\Delta X_f}{v_{x,1}} \end{aligned} \tag{3.4}$$

where w is the width of the leading vehicle and $v_{x,1}$ is the velocity of the front axle of the LCV.

3.1.2 Vehicle models

The vehicle of interest for this thesis has been an A-double LCV, which is presented in Fig. 3.2. In order to model a vehicle of this complexity, the vehicle dynamics has to be computed and solved. The following sections will thus briefly show how the vehicle prediction model and plant model are expressed and also how the surrounding vehicle in traffic are modelled. The derived vehicle models [24] that are being presented in the following sections, have already been used previously and showed promising results [2].

Subject vehicle prediction model

The prediction model consists of a one-track model of an A-double combination seen in Fig. 3.2. It was derived using Lagrangian formalism. The one-track model is presented in Fig. 3.3 where the left hand side shows the

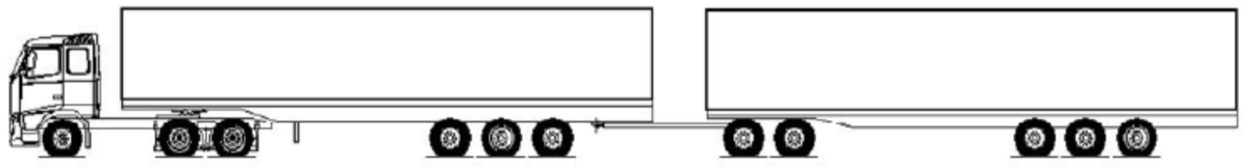


Figure 3.2: *Illustration of the A-double LVC based on the European modular system. Figure taken from [24].*

spatial parameters of the vehicle model and the right hand side shows the included motion variables and tyre forces. In the model all of the effect from tyres on an axle are combined into one virtual tyre and the concept of an equivalent wheel-base was used.

This model is used to describe the truck motion for the TSPs of the framework. It has been linearised regarding the kinematics, steering and tire slip using an assumption of small angles. The truck positioning with respect to the road is obtained by including parametrization of the road curvature and heading angle in the model equations. Finally, the model differential equations are formed, with the help of another work [25], which consists of the following

$$
\begin{aligned}
\dot{z}_1 =& 47.0 \cdot \delta - z_{10} \cdot z_3 + 1.9 \cdot z_4 + 0.9 \cdot z_6 - 0.002 \cdot z_8 + \\
& (-70.7 \cdot z_1 + 9.7 \cdot z_3 + 21.7 \cdot z_5 + 4.5 \cdot z_7 - 0.02 \cdot z_9)/z_{10} \\
\dot{z}_2 =& z_3 - \kappa_{R,1} \cdot (z_{10} \cdot \cos z_2 - (z_1 + z_3 \cdot 1.5) \cdot \sin z_2) \\
\dot{z}_3 =& 25.0 \cdot \delta - 1.9 \cdot z_4 - 0.8 \cdot z_6 + 0.002 \cdot z_8 + (27.6 \cdot z_1 \\
& - 174.2 \cdot z_3 - 20.8 \cdot z_5 - 4.3 \cdot z_7 + 0.02 \cdot z_9)/z_{10} \\
\dot{z}_4 =& z_5 \\
\dot{z}_5 =& -25.5 \cdot \delta - 4.0 \cdot z_4 + 2.5 \cdot z_6 - 0.007 \cdot z_8 + (-36.5 \cdot z_1 \\
& + 165.4 \cdot z_3 - 10.9 \cdot z_5 + 13.0 \cdot z_7 - 0.05 \cdot z_9)/z_{10} \\
\dot{z}_6 =& z_7 \\
\dot{z}_7 =& 0.6\delta + 2.3 \cdot z_4 - 22.9 \cdot z_6 - 0.9 \cdot z_8 + 19.9 \cdot z_1 \\
& - 216.8 \cdot z_3 - 169.7 \cdot z_5 - 125.8 \cdot z_7 - 7.2 \cdot z_9)/z_{10} \\
\dot{z}_8 =& z_{10} \\
\dot{z}_9 =& -0.19 \cdot \delta + 5.1 \cdot z_4 + 22.7 \cdot z_6 - 7.1 \cdot z_8 + (-12.5 \cdot z_1 \\
& - +195.8 \cdot z_3 + 168.6 \cdot z_5 + 68.2 \cdot z_7 - 54.7 \cdot z_9)/z_{10} \\
\dot{z}_{10} =& a_{x,1} \\
\dot{z}_{11} =& (a_{x,1,des} - a_{x,1})/\tau \\
\dot{z}_{12} =& 1/(1 - \kappa_{R,1} \cdot z_{13}) \cdot (z_{10} \cdot \cos(z_2) - (z_1 + z_3 \cdot 1.5) \cdot \sin(z_2)) \\
\dot{z}_{13} =& z_{10} \cdot \sin(z_2) + (z_1 + z_3 \cdot 1.5) \cdot (z_2) \\
\dot{z}_{14} =& 1/(1 - \kappa_{R,4} \cdot z_{15}) \cdot ((-24.6 \cdot z_3 - 22.7 \cdot z_5 - 12.3 \cdot z_7 - 7.7 \cdot z_9 \\
& - z_4 \cdot z_{10} - z_6 \cdot z_{10} - z_8 \cdot z_{10} + z_1) \cdot -\sin(z_2 + z_4 + z_6 + z_8 \\
& - \theta_{R,4} + \theta_{R,1}) + z_{10} \cdot \cos(z_2 + z_4 + z_6 + z_8 - \theta_{R,4} + \theta_{R,1})) \\
\dot{z}_{15} =& ((-24.6 \cdot z_3 - 22.7 \cdot z_5 - 12.3 \cdot z_7 - 7.7 \cdot z_9 - z_4 \cdot z_{10} - z_6 \cdot z_{10} \\
& - z_8 \cdot z_{10} + z_1) \cdot \cos(z_2 + z_4 + z_6 + z_8 \theta_{R,4} + \theta_{R,1}) + z_{10} \cdot \sin(z_2 + z_4 \\
& + z_6 + z_8 - \theta_{R,4} + \theta_{R,1})) \\
\dot{z}_{16} =& \dot{\delta} \\
z =& [\dot{y}_1, \phi, \dot{\phi}, \theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2, \theta_3, \dot{\theta}_3, v_{x,1}, a_{x,1}, s_1, e_1, s_{11}, e_{11}, \dot{\delta}] \\
u =& [a_{x,1,des}, \delta]
\end{aligned}
$$

where the states $\dot{y}_1, \phi$, and $\dot{\phi}$ are the lateral velocity, yaw angle, and yaw angle rate of the first axle of the vehicle, $\theta_1, \theta_2, \theta_3, \dot{\theta}_1, \dot{\theta}_2$, and $\dot{\theta}_3$ are the articulation angles and the respective articulation angle rates of the towed units, $v_{x,1}$ and $a_{x,1}$ is the longitudinal velocity and acceleration of the first axle, s_1, s_{11}, e_1, and e_{11} are the distances and perpendicular distances of the first and last vehicle axles projected on the road geometry,

$\kappa_{R,1}, \kappa_{R,4}, \theta_{R,1}$, and $\theta_{R,4}$ are the road curvature and road heading for the first and last axle. τ is a parameter denoting a time constant for the longitudinal dynamics. The input u to the model consists of the longitudinal acceleration of the first axles $a_{x,1,des}$ and the road wheel steering angle δ.

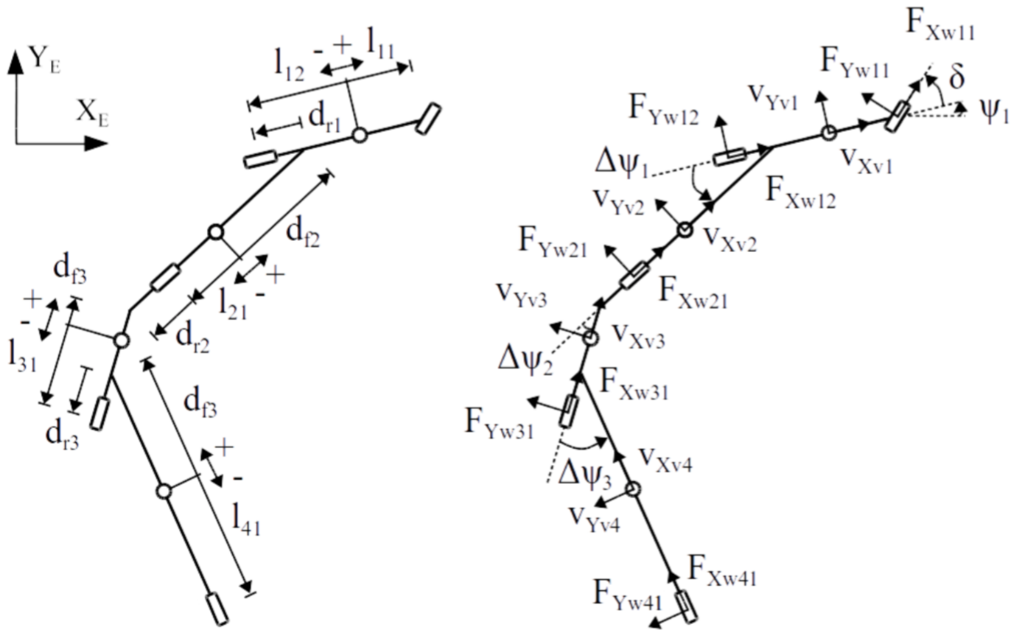


Figure 3.3: *Illustration of a one-track model of an A-double combination. The left hand side of the figure shows the spatial parameters of the vehicle model and the right hand side shows the included motion variables and tyre forces. Figure taken from [24].*

Surrounding vehicle prediction model

The surrounding vehicles were modeled in the most simple way using point-mass models. They were constrained to always follow the lane center, thus only the longitudinal dynamics were modeled along the road geometry. The motion of the vehicles can be expressed through the following equation

$$\frac{d}{dt}\begin{bmatrix} s_{o,n} \\ \dot{s}_{o,n} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{o,n} \\ \dot{s}_{o,n} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot \ddot{s}_{o,n} \tag{3.5}$$

where n is the index of the vehicle, and $s_{o,n}, \dot{s}_{o,n}$ and $\ddot{s}_{o,n}$ are the position, velocity and acceleration along the road geometry.

Together with the states, the vehicles store information about their own length and widths, but also in which lane they are currently in. When the models are used in traffic situation predictions, only a constant velocity is assumed throughout the entire prediction horizon.

Subject vehicle plant model

For the vehicle plant, a high fidelity two-track model was used in the simulating framework. Fig. 3.4 shows the spatial parameters and included motion variables together with tyre forces. It is visible from the figure that the more realistic two-track model has far more degrees of freedom compared to the one-track model.

The framework uses a Volvo in-house developed library to emulate this high fidelity plant model. It includes detailed sub-models of various components such as vehicle chassis, cab suspensions, steering system, powertrain, and brakes. The torsion flexibility of the frame of the tractor and semi-trailers is considered and taken into account by connecting multiple frame bodies through springs. Lastly, the magic formula tire model [26] is used for all tires in the vehicle combination. The model considers slip, dynamic relaxation and rolling resistance.

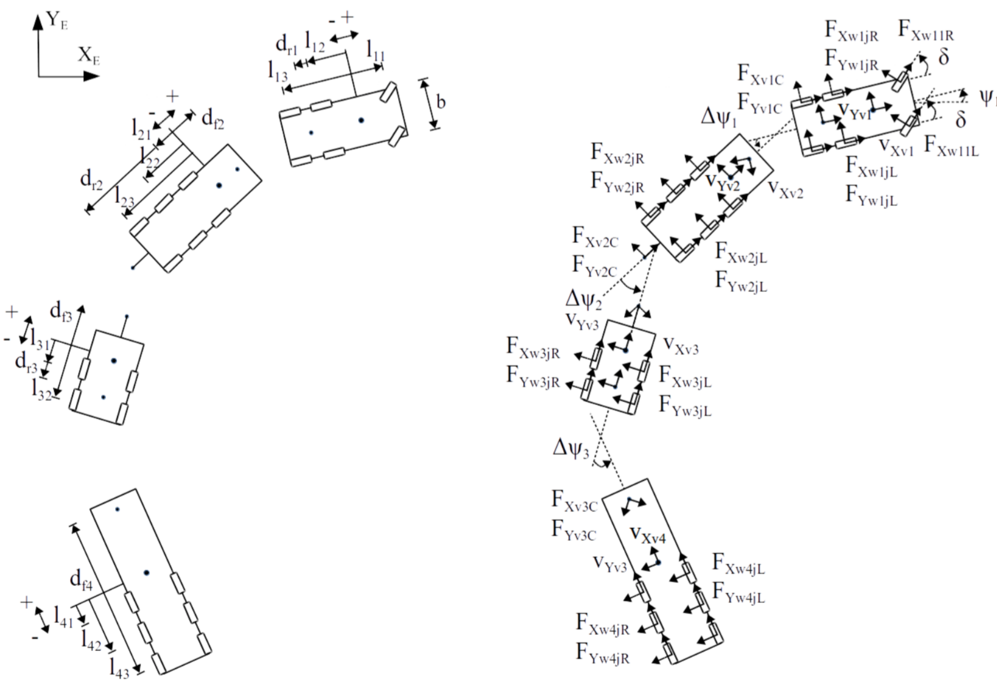


Figure 3.4: *Illustration of a two-track model of an A-double combination. The left hand side of the figure shows the spatial parameters of the vehicle model and the right hand side shows the included motion variables and tyre forces. Figure taken from [24].*

3.1.3 Road modelling

In order to evaluate the automated driving, it is necessary to have a clearly defined road model for the LCV to drive on. Considering then a global frame G, it is possible to define a road profile

$$\begin{pmatrix} X \\ Y \end{pmatrix} = f_r(s) \tag{3.6}$$

where s refers to the arc length of f_r for some arbitrary initial point $s = 0$. The road profile is constructed using clothoids, since they are widely used and are known to provide smooth driving associated with low jerk values [27]. Thus, the road profile f_r is constructed using clothoid segments in a similar way as Fig. 3.5.

With the road defined, it is possible to express the LCV positions with respect to the road. A certain point of the truck can be projected onto the road defined by a coordinate s and a perpendicular distance from the road geometry d. A coordinate (s, d) is not guaranteed to be unique for all coordinates (X, Y), nor is it guaranteed to exist for all (X, Y). During the simulations and evaluations of the vehicle dynamics, the framework uses a Frenét frame for each position s along the road geometry [16].

Lastly, the road model stores information about the lanes on the current road, providing a topology of current lane and adjacent lanes, whether there exists a lane to either of the sides, or if it is a road edge. The lanes contain information about their own widths, maximum distance, adjacent lanes, and clothoid properties such as curvatures, heading angles and arc lengths.

3.1.4 Traffic situation predictions

With the key components given above, the framework has most of the information present to form a traffic situation understanding. TSPs are formed for the current lane the LCV is driving in, but also for the adjacent left and right lanes. The predictions are simulated for a time horizon of $t_h = 3.5$s. During this time horizon, the surrounding traffic is predicted to continue moving along the road geometry with their current respective velocity by integrating their position using the simple forward-Euler integration with a step size of $h = 0.05$s.

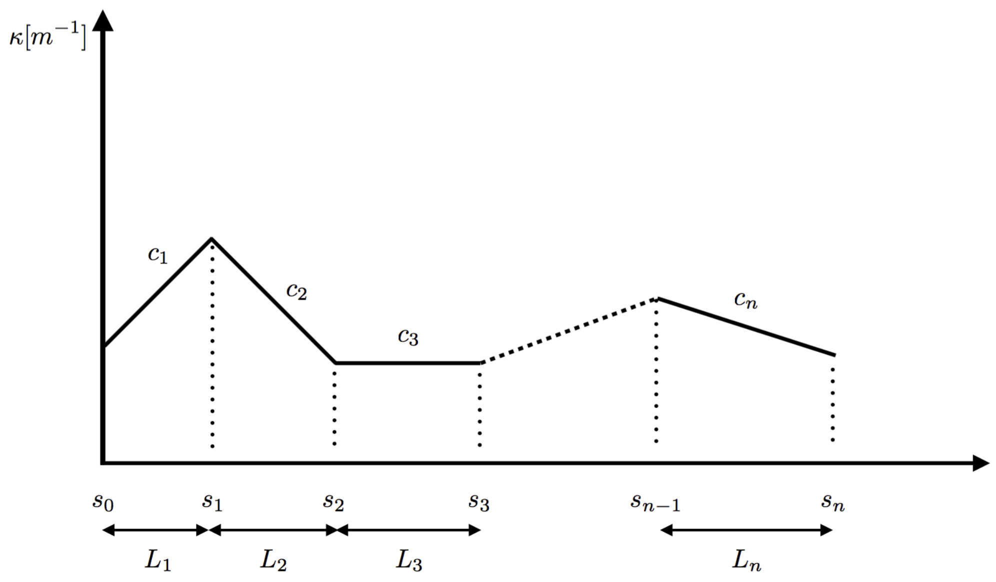


Figure 3.5: *The clothoid segments are presented above, which can be seen as a piecewise linear curvature function. Each clothoid segment is described by a pair of kink points, its curvature c_i and arc length L_i.*

Given this information, it is possible to integrate the LCV states together with the predictions of the surrounding traffic and its corresponding state equations. For each step in the integration the LCV is evaluated given a set of constratints that are related to the vehicle dynamics and lane boundaries

$$\underline{v}_x \leq v_{x,1} \leq \bar{v}_x \tag{3.7}$$
$$\underline{a}_y \leq a_{y,1} \leq \bar{a}_y \tag{3.8}$$
$$\underline{a}_y \leq a_{y,4} \leq \bar{a}_y \tag{3.9}$$
$$\underline{d} \leq d_1 \leq \bar{d} \tag{3.10}$$
$$\underline{d} \leq d_4 \leq \bar{d} \tag{3.11}$$

where $\underline{v}_x$ and $\bar{v}_x$ are the speed limits for the vehicle and, $v_{x,1}$ is the speed for the first axis of the LCV, $\underline{a}_y$ and $\bar{a}_y$ are the limits for the lateral acceleration, $a_{y,1}$ and $a_{y,4}$ are the lateral accelerations of the first and fourth axis, and $\underline{d}$ and $\bar{d}$ are the limits of the lateral offset towards the center lane. The limits for the lateral offsets change depending on the internal states of the vehicle.

A final constraint is added that checks collisions against vehicles. It is based by creating bounding boxes for the LCV units and the surrounding vehicles and each bounding box consists of four lines encapsulating the vehicle. Then by checking line intersections in 2D it can be decided whether any of the bounding box lines intersect.

This is solved by setting up the following equation system

$$\mathbf{P}_a = \mathbf{P}_{a,1} + s(\mathbf{P}_{a,2} - \mathbf{P}_{a,1})$$
$$\mathbf{P}_b = \mathbf{P}_{b,1} + t(\mathbf{P}_{b,2} - \mathbf{P}_{b,1})$$

and solving for s and t when $\mathbf{P}_a = \mathbf{P}_b$. $\mathbf{P}_{a,1}$, $\mathbf{P}_{b,1}$ and $\mathbf{P}_{a,2}$, $\mathbf{P}_{b,2}$ are the start and end points for each line a and b. A collision occurs between the lines only if $|s| < 1$ and $|t| < 1$. By solving this for the bounding boxes between two objects a collision may be detected.

If any constraints are violated during the iterations of the closed-loop predictions the plan is seen as infeasible, which the decision making algorithm takes into account [28]. However, take note that the simulation scenario in this thesis will not emphasize the decision making aspect of the driver model control.

3.1.5 Control hierarchy

The final control design of the framework is depicted in Fig. 3.6. The driver model control is fed inputs from measured vehicle states

$$z = [\dot{y}_1, \phi, \dot{\phi}, \theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2, \theta_3, \dot{\theta}_3, v_{x,1}, a_{x,1}, s_1, e_1, s_{11}, e_{11}, \dot{\delta}] \tag{3.12}$$

and the observations from the traffic environment; road curvature $[\kappa_r]$, road heading angle $[\theta_r]$, road distance $[s_r]$ and max road velocity $[v_{r,max}]$. The surrounding vehicles relative distance $[\Delta s_{o,n}]$, velocity $[\dot{s}_{o,n}]$, acceleration $[\ddot{s}_{o,n}]$ and lane $[l_{o,n}]$ is also fed. The driver also has the possibility to request a lane change externally from the framework.

Given the input, the driver model control sets up and evaluates the TSPs for the entire prediction horizon. After the evaluations are performed, the driver model control outputs the desired longitudinal acceleraton $a_{x,des}$ and road wheel steering angle rate $\dot{\delta}$. This is then fed as input to the vehicle motion management block which uses the control actuation signals and integrates them against a vehicle plant model. The vehicle plant model in this case might be a one-track model or the high fidelity two-track model mentioned in the previous section.

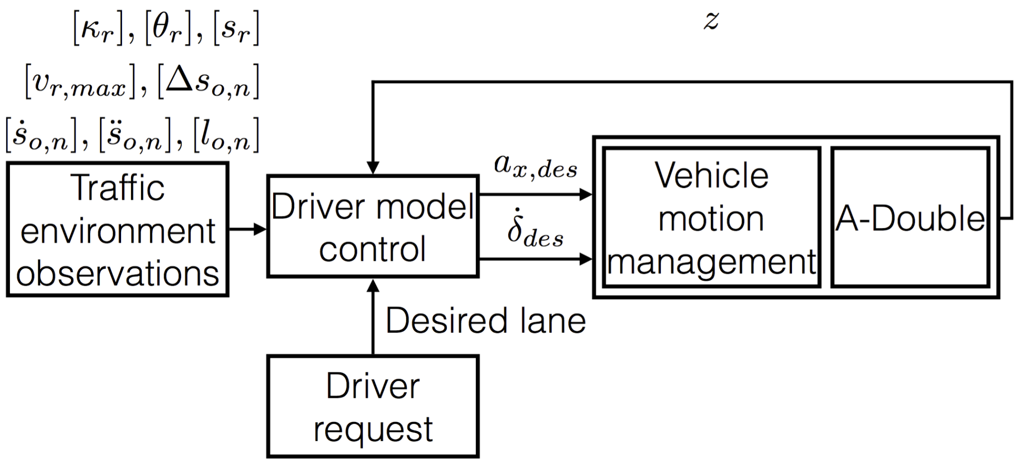

$[\kappa_r], [\theta_r], [s_r]$
$[v_{r,max}], [\Delta s_{o,n}]$
$[\dot{s}_{o,n}], [\ddot{s}_{o,n}], [l_{o,n}]$
z
Traffic environment observations
Driver model control
$a_{x,des}$
$\dot{\delta}_{des}$
Vehicle motion management
A-Double
Desired lane
Driver request

Figure 3.6: *The control design for the automated driving framework. The driver model control requires inputs from the vehicle state measurements and traffic observations, and outputs the desired longitudinal acceleration and road wheel steering angle rate.*

3.2 Biologically inspired optimization

When optimizing, or mathematically programming, one often manipulates a structure or a system to achieve a defined goal. The systems are often possible to define in terms of a mathematical function, and is then given the goal to either minimize or maximize the function value. A few fields that solve problems using optimization include decision-making, path planning, time-series prediction and various engineering problems such as construction of vehicles. This is just a very small list of many applications, but it is easily concluded that optimization plays a major role in science and engineering.

In most real-life applications you are faced with constraints, i.e limits on certain variable ranges. It gives rise to the question of what is the best thing to do, given a finite set of resources. As it was stated above, it is possible to formulate many problems as the minimization, or maximization, of a mathematical function, often denoted as the objective function. To solve problems such as these the wide field of mathematics comes to use.

3.2.1 Particle swarm optimization

PSO is based on the biological phenomenon of swarms. The algorithm is capturing mathematically the aspects of swarming which is namely its search efficiency [29]. In PSO the particles are encoded with a position and a velocity in the search domain, $\mathcal{R}^N$. Each individual is then evaluated against an objective function to obtain a performance. Therefore methods for updating the particles is based on the performance of the particle and that of the other particles.

Algorithm

The first step of the algorithm is to initialize the position $\mathbf{x}_i$ and velocity $\mathbf{v}_i$ of particles $p_i, i = 1, ..., N \in \mathcal{Z}$. The most suitable value of N varies often from problem to problem, however, typically the value of N is in the range of 20-40 particles [29]. The particle positions are often initialized randomly with a uniform distribution over the search space. Given a variable range the initialization can be written as

$$x_{ij} = x_{min,j} + r(x_{max,j} - x_{min,j}), \quad i = 1, ...N, \quad j = 1, ..., n \tag{3.13}$$

where x_{ij} denotes the j^{th} component of particle p_i, i.e the j^{th} variable. r denotes the uniform random number in range $r \in [0, 1]$, $x_{min,j}$ and $x_{max,j}$ denote the variable range for the j^{th} component. Finally N denotes the size of the swarm and n is the dimensionality, or number of variables, of the particle.

Particle velocities are also initialized randomly in a similar fashion

$$v_{ij} = \frac{\alpha}{\Delta t}\left(-\frac{x_{max,j} - x_{min,j}}{2} + r(x_{max,j} - x_{min,j})\right), \quad i = 1, ..., N, \quad j = 1, ..., n \tag{3.14}$$

where v_{ij} is the velocity of the j^{th} component of particle p_i, α is a constant in the range $\alpha \in [0, 1]$, Δt is the time step, and r is yet again a uniform random number in range $r \in [0, 1]$. In the special case where $x_{min,j} = -x_{max,j}$ the equation above can be reduced to

$$v_{ij} = \alpha\frac{x_{min,j} + 2rx_{max,j}}{\Delta t}, \quad i = 1, ..., N, \quad j = 1, ..., n. \tag{3.15}$$

Having initialized the particles, the next step is to evaluate and store the performance, often named cost, for each particle. How this is done depends on the given problem, but the evaluation is based on feeding the particle positions as input to an objective function that has the goal to be minimized or maximized.

In order to reach the optimal value of the objective function, the particles have to be updated, meaning that the positions and velocities need to be updated. The update rule relies on storing the personal best position $\mathbf{x}_i^{pb}$ of particle p_i and the global best position $\mathbf{x}^{gb}$ of any particle in the entire swarm. Thus, after evaluating particle p_i, two performance tests are needed. The first test is comparing the current performance with the previous best performance. If it is better then update $\mathbf{x}_i^{pb}$. The second test can be performed in multiple ways depending on the definition of the global best performance of all particles in the swarm. It is either assumed to be in the current swarm, or all particles evaluated thus far, but it also depends whether the global best is checked against all particles or within a neighbourhood. The latter will be discussed later in the text. After deciding on the approach for step two, $\mathbf{x}^{gb}$ is updated and stored.

With the updated values $\mathbf{x}_i^{pb}$ and $\mathbf{x}^{gb}$ the update rule is given as

$$v_{ij} \leftarrow v_{ij} + c_1 r\left(\frac{x_{ij}^{pb} - x_{ij}}{\Delta t}\right) + c_2 q\left(\frac{x_j^{gb} - x_{ij}}{\Delta t}\right), \quad j = 1, ..., n \tag{3.16}$$

where r and q are random numbers in the range $r, q \in [0, 1]$, and c_1 and c_2 are constants. The parameter c_1 is often referred as the cognitive component and c_2 as the social component. The cognitive component is seen as the degree of self-confidence of the particle in the swarm, whereas the social component is the particles trust in the ability of the swarm. Usually c_1 and c_2 are both set to 2, so that the mean of rc_1 and qc_2 is the same [29]. When updating the velocities it is necessary to make sure the velocities are restricted to the range $|v_{ij}| < v_{max}$, in order to keep it from expanding uncontrollably. This does not however imply that the position of particle p_i will be constrained within $[x_{min,j}, x_{max,j}]$. The final step is to update the positions as well which is done as

$$x_{ij} \leftarrow x_{ij} + v_{ij}\Delta t, \quad j = 1, ..., n, i = 1, ..., N. \tag{3.17}$$

This concludes one iteration of the algorithm. After updating the positions and velocities the particles are evaluated again and the steps above are performed again. For further clarity the PSO is presented in Alg. 1.

Algorithm 1 PSO algorithm

```
1: Initialize particle swarm, i.e positions and velocities of particle $p_i$:
        $x_{ij} = x_{min,j} + r(x_{max,j} - x_{min,j}), \quad i = 1, ...N, \quad j = 1, ..., n$
        $v_{ij} = \frac{\alpha}{\Delta t}\left(-\frac{x_{max,j} - x_{min,j}}{2} + r(x_{max,j} - x_{min,j})\right), \quad i = 1, ..., N, \quad j = 1, ..., n$
2: Evaluate each particle against the objective function $f(\mathbf{x}_i), \quad i = 1, ..., N$
3: Update the personal best $\mathbf{x}_i^{pb}$ position for all particles and and global best $\mathbf{x}^{gb}$
        if $f(\mathbf{x}_i) < f(\mathbf{x}_i^{pb})$
            $\mathbf{x}_i^{pb} \leftarrow \mathbf{x_i}$
        if $f(\mathbf{x}_i) < f(\mathbf{x}^{gb})$
            $\mathbf{x}^{gb} \leftarrow \mathbf{x_i}$
4: Update particle velocities:
        $v_{ij} \leftarrow v_{ij} + c_1 r\left(\frac{x_{ij}^{pb} - x_{ij}}{\Delta t}\right) + c_2 q\left(\frac{x_j^{gb} - x_{ij}}{\Delta t}\right), \quad j = 1, ..., n$
        if $|v_{ij}| > v_{max}$
            $v_{ij} \leftarrow v_{ij} v_{max} |v_{ij}|^{-1}$
5: Update paticle positions
        $x_{ij} \leftarrow x_{ij} + v_{ij}\Delta t, \quad j = 1, ..., n, \quad i = 1, ..., N$
6: Unless the termination criterion has been reached, repeat from step 2.
```

Best in current swarm vs best ever

In the section above we assumed that the x^{gb} denoted the best particle ever thus far in the swarm. It is possible to follow an alternate approach which is based on the best position $\mathbf{x}_c^{gb}$ in the current swarm, i.e the best of N particles in the current iteration. Other than that the algorithm is executed exactly the same. Taking computer programming into consideration, this modification is implemented with one line of code which resets the best in swarm position in each iteration.

Neighbourhood topology

Comparing the global best particle x^{gb} can be done in different ways, namely based on neighbourhoods. In the algorithm described above the neighbourhood included all of the particles, meaning that all particles were interconnected and that $\mathbf{x}_i^{gb} = \mathbf{x}^{gb}$. A visual representation is shown in Fig. 3.7a. There are, however, many alternatives to a fully connected neighbourhood. The fully connected neighbourhood is simply that each particle is connected to its $K = N - 1$ nearest neighbours. By letting $K = 2$ the neighbourhood seen in Fig. 3.7b. The effect of having a swarm that is has a neighbourhood structure that is less connected serves the purpose of preventing premature convergence [29]. As a general rule, introduction of restricted neighbourhood structures slows down the convergence of the algorithm, but it also allows for greater cover of the search space, which in some cases might lead to better results. It should be noted that the neighbourhood structures normally remain fixed during optimization.

3.2.2 Inertia weight

The last modification that will be discussed in this thesis is the inertia weight, w. The inertia weight determines the relative influence of previous velocities of a particle. Applying the concept of inertia the velocity update for

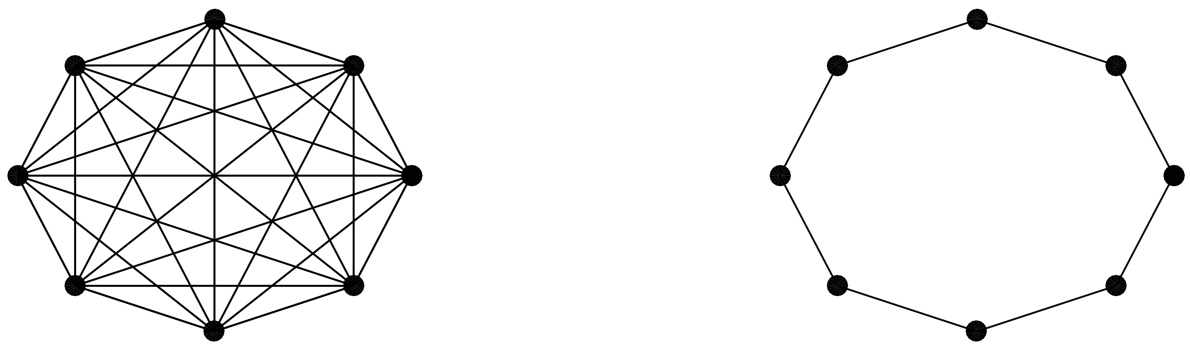

(a) *Fully connected neighbourhood.* (b) *Neighbourhood with $K = 2$ connections.*

Figure 3.7: *Two commonly used neighbourhood topologies. The figure to the left compares uses the global best particle in the swarm, whereas the figure to the left uses the global best particle compared to its closest neighbours.*

particle p_i becomes

$$v_{ij} \leftarrow wv_{ij} + c_1 r \left(\frac{x_{ij}^{pb} - x_{ij}}{\Delta t} \right) + c_2 q \left(\frac{x_j^{gb} - x_{ij}}{\Delta t} \right), \quad j = 1, ..., n. \tag{3.18}$$

If $w > 1$, the particle will favor exploration over exploitation, i.e the cognitive and social components will of be less value. However, if $w < 1$ the particle will be more attracted to the current best positions. Thus, a common strategy is to start with a w larger than 1 and reduce it by a constant factor $\beta \in]0, 1[$ each iteration until w reaches the desired lower bound.

3.2.3 Genetic algorithms

GAs are a type of evolutionary algorithms. This section will try to explain the basic parts of a GA and how it is implemented to optimize a given objective function. To solve this using a GA the information is encoded as a string of digits inside of a chromosome, which is essentially a vector holding digits from a programming point of view. Each of the digits are denoted as a gene, to keep the connection with the biological terminology. The genes encode the information of the chromosome and there are various encoding schemes. In the first introduced GA [30], a binary encoding scheme was proposed, where the genes took the values of 0 and 1 only.

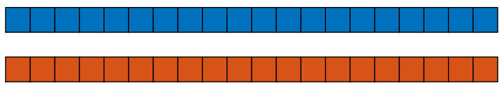
The algorithm is initialized by introducing a population of N chromosomes $c_i, i = 1, ..., N$ with random values with equal probability of the gene values 0 and 1. The value of a gene is often called an allele in GA terminology. The first initialized chromosomes constitute the first generation of the entire population. After the initialization, the N chromosomes have to decode its genes to form a corresponding individual consisting of n variables $x_j, j = 1, ..., n$. This can be done in various ways, but most common is to divide the chromosome into even bits $k = m/n$, where m denotes the number of genes in a chromosome. The decoding for each variable x_j can then be written as

$$x_{j,tmp} = \sum_{v=1}^{k} 2^{-v} g_{v+(j-1)k}, \quad j = 1, ..., n \tag{3.19}$$

where k is the number of bits of each chromosome. Thereafter the value of $x_{j,tmp}$ is simply scaled to cover the desired search region of $[x_{j,min}, x_{j,max}], j = 1, ..., n$ as

$$x_j = x_{min,j} + \frac{x_{max,j} - x_{min,j}}{1 - 2^{-k}} x_{j,tmp}, \quad j = 1, ..., n. \tag{3.20}$$

Usually in practical applications the decoding step might be seen as tedious and unecessary, thus it can be avoided by using real-number encoding instead of binary encoding. In real number encoding the alleles of each


(a) *Two selected chromosome pairs*

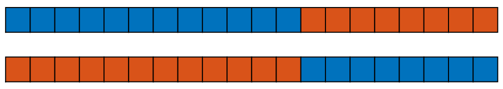

(b) *Chromosome pairs after the crossover operation.*

Figure 3.8: *An illustration of the crossover process. Figure 3.8a shows two chromosome pairs selected from the tournament selection, while figure 3.8b shows the resulting chromosomes after the crossover operation. The different colours represent the different alleles of the chromosomes.*

genes contain a floating-point number in the range of $[0, 1]$ the decoded variable is simply gotten from

$$x = x_{min} + g(x_{max} - x_{min}) \tag{3.21}$$

which results in x being in the range of $[x_{min}, x_{max}]$. There exist more encoding schemes than the two presented above, but they are out of the scope for this thesis.

Having decoded the chromosome, each individual i is evaluated against an objective function and assigned a corresponding fitness value F_i, which is later used to select the most fit individuals for reproduction. In GAs the fitness is usually seen as a goodness measure than an error measure, thus the goal is to maximize it. However, if one wishes to minimize a measure instead, it is simply equivalent to maximize its inverse. Therefore in the case of maximization, the objective function can be taken as the fitness.

Assuming that all N individuals have been evaluated and assigned a fitness measure it is time to form the next generation. In relation to what is observed in biology, there must be a sort of selection present, where the most fit individuals are selected. To avoid stagnation and resemble reality, this process is not made deterministic so that it does not always favour the fittest individuals. The most common ways of performing selection in GAs are through tournament selection and roulette-wheel selection. In roulette-wheel selection the individuals are selected by forming the cumulative relative fitness value

$$\Phi_j = \frac{\sum_{i=1}^{j} F_i}{\sum_{i=1}^{N} F_i}, j = 1, ..., N, \tag{3.22}$$

where F_i is the fitness value of individual i. Thereafter a random number $r \in [0, 1]$ is drawn uniformly and the individual that is selected is the one with the smallest j so that the following is satisfied

$$\Phi_j > r. \tag{3.23}$$

Since roulette-wheel selection is not a very likely biological process, an alternative procedure which resembles nature more is tournament selection. In essence, tournament selection is based on picking two individuals randomly, and at equal probability for all individuals, from the population. The individual with better fitness is then selected with probability p_{tour} and the worse individual with probability $1 - p_{tour}$. The parameter p_{tour} is called the tournament selection parameter, and is usually set around $0.7 - 0.8$. The tournament selection size can be generalized to include more than two individuals.

Sexual reproduction requires two individuals to produce an offspring with a combination of both parents genes. Thus, for the GA, individuals need to be selected in pairs using the selection method introduced above. Since the genetic information for each individual is stored in a single chromosome with fixed length, it is fairly easy to combine the genes of two individuals.

In GAs the crossover procedure consists of cutting the chromosomes at a randomly selected crossover point and then swap the end parts of each chromosome with each other. Fig. 3.8 shows an illustration of the crossover operation on two chromosomes. The crossover procedure is also, just like the selection procedure, non-deterministic. A parameter p_{cross} is defined which is the crossover probability, and decides the probability of two individuals actually mating.

The next step in the evolutionary process is to introduce mutations, which is crucial for evolution. Mutation is introduced to the GAs after the crossover operation by changing the alleles randomly. In the case of binary encoded chromosomes, this results in flipping the alleel from 1 to 0, or vice versa, with a mutation probability p_{mut}. If the chromosomes are encoded as floating point numbers, then the mutation process can be done by using creep-mutations, which will not be discussed in this text. In practice mutation is implemented by drawing a random number $r \in [0, 1]$ and mutating the allele if $r < p_{mut}$.

After performing all of the genetic operations on the chromosomes, the GA faces the final step, which is replacement. The simplest way is called generational replacement, where all of the individuals from the first generation are discarded and the N new individuals take their place. This concludes one generation of the GA and the new generation is taken back and evaluated and thus the cycle is closed. The entire algorithm is presented in steps in Alg. 2

Algorithm 2 GA algorithm

```
Initialize the population by randomly generation $N$ chromosomes $c_i, i = 1, ..., N$. If using binary encoding, generate chromosomes of length $m = kn$ where $k$ denotes the number of bits per variable.
Evaluate the individuals
    1. Decode chromosome $c_i$ to form $\mathbf{x}_i$.
    2. Evaluate each individual against the objective function and assign it a fitness value $F_i = f(\mathbf{x}_i)$.
    3. Repeat the two steps above until the entire population has been evaluated.
Form the new generation
    1. Select two individuals $i_1$ and $i_2$ from the evaluated population through means of non-deterministic selection. Individuals with better fitness should have greater probability of being selected than individuals with worse fitness.
    2. Generate two new chromosomes by crossing the selected chromosomes if $r < p_{cross}$, where $r \in [0, 1]$ is a random number.
    3. Mutate the alleles of each chromosome with probability $p_{mut}$.
    4. Repat the steps above until $N$ new individuals have been generated.
Unless the termination criterion has been reached, repeat from step 2.
```

3.2.4 Considered variables and cost function

This thesis will focus on optimizing driver model parameters for the lateral and longitudinal control. From the previous chapter it was stated that the control law for the lateral actuation was given by

$$\dot{\delta}_{ref} = k_f \dot{\theta}_f + k_n \dot{\theta}_n + k_I \theta_n \tag{3.24}$$

and the longitudinal control law was given by

$$a_{x,ref} = (1 + \dot{\tau}_m) \cdot \frac{\Delta v_x^2}{\Delta X_f - v_0 \cdot t_h}. \tag{3.25}$$

Depending on the choice of the parameters k_f, k_n, and k_I, it will result in different steering behaviour. One set of parameter may result in the characteristics of a calm driver, whereas other may reflect a more aggressive one. Therefore it is of great importance to be able to optimize these parameters on the fly, allowing for a more dynamic control system. As for the longitudinal parameter $\dot{\tau}_m$, it controls how strongly the system reacts to the acceleration commands.

The used framework currently has a set of preferred driver model parameters denoted: $k_{f,d}$, $k_{n,d}$, $k_{I,d}$, and $\dot{\tau}_{m,d}$. These were previously set where a genetic algorithm was used to fit the parameters after on-road study with professional truck drivers [18]. Therefore it is of interest to find solutions that do not deviate too much from the original desired parameters. Hence the cost function should include a term that benefits parameter values that are close to the original ones.

Cost function

As previously stated, the goal is to find parameters that deviate as little from the desired parameters, while still generating a feasible solution. The set of feasible solutions in this case depends on various complex factors such as the surrounding traffic observations and road profile. A solution that fulfills all the characteristics such as minimal lane off-tracking, abnormal lateral accelerations and no collisions would result in a low cost. Taking inspiration from similar work [20], the cost function was defined as

$$F = C_p + C_{o,1} + C_{o,4} + C_{a,1} + C_{a,4} + C_c, \tag{3.26}$$

where C_p penalizes deviance from the desired driver model parameters, $C_{o,1}$ and $C_{o,4}$ penalizes solutions where the LCVs lateral offset for the first and fourth axle becomes larger than the allowed limits, $C_{a,1}$ and $C_{a,4}$ penalizes solutions that generates large lateral acceleration for the first and fourth axle, and C_c penalizes collisions with surrounding traffic. The parameters C_p, $C_{o,1}$, and $C_{o,4}$ may be viewed as optimization criteria that will improve the solutions. The remaining three, however, are criteria that determine the feasibility and thus must be satisfied for a solution to be valid.

In the cost function the parameter term is given by

$$C_p = \sqrt{\frac{(k_{f,d} - k_f)^2 + (k_{n,d} - k_n)^2 + (k_{I,d} - k_I)^2 + (\dot{\tau}_{m,d} - \dot{\tau}_m)^2}{k_{f,d}^2 + k_{n,d}^2 + k_{I,d}^2 + \dot{\tau}_{m,d}^2}}, \tag{3.27}$$

Unless the driver model parameters are enforced to be bound within certain limits there is no guarantee that $C_p \in [0, 1]$. However, C_p will be in the range $[0, 1]$ only when the following is true

$$(k_{n,d} - k_n)^2 + (k_{n,d} - k_n)^2 + (k_{n,d} - k_n)^2 + (\dot{\tau}_{m,des} - \dot{\tau}_m)^2 \leq k_{f,d}^2 + k_{n,d}^2 + k_{I,d}^2 + \dot{\tau}_{m,d}^2 \tag{3.28}$$

The terms corresponding to the lateral off-tracking of the lane for axle i is defined as

$$C_{o,i} = \sum_{k=1}^{N} \left(\begin{cases} \frac{|d_{i,k} - \bar{d}|}{|w - \bar{d}| \cdot N} & \text{if } d_{i,k} > \bar{d} \\ \frac{|d_{i,k} - \underline{d}|}{|w - \underline{d}| \cdot N} & \text{if } d_{i,k} < \underline{d} \\ 0, & \text{else} \end{cases} \right), \tag{3.29}$$

where $d_{i,k}$ is the lateral offset from the center of road lane at time step k for the i:th axle, w is the maximum allowed off-tracking, in this case one full lane width, and N is the number of time steps of the closed-loop control algorithm. To get a better understanding, see Fig. 3.9. By designing the constraint in this way we ensure that $C_{o,i}$ will always be in the range of $[0, 1]$.

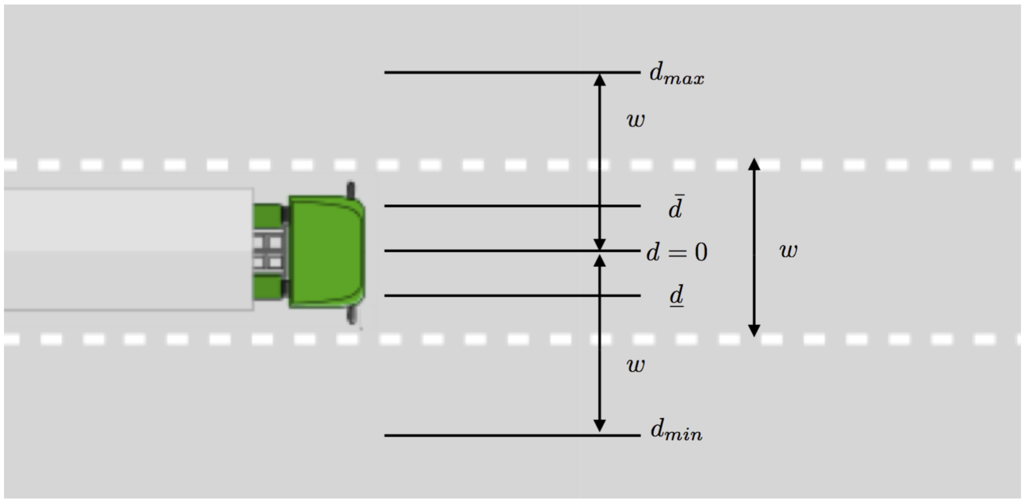


Figure 3.9: *Illustration of the lateral constraints.*

The acceleration term is a hard constraint, meaning that if it is violated the entire solution is infeasible. It is defined as the following for an axle i

$$C_{a,i} = \begin{cases} 0 & \text{if } \underline{a} \leq a_i \leq \bar{a} \\ P & \text{else} \end{cases} \tag{3.30}$$

therefore

$$C_{a,i} \in \{0\} \cup \{P\} \tag{3.31}$$

where $\underline{a}$ and $\bar{a}$ are the limits for the acceleration for the i:th axle. P is a penalty term that if set to $P = 2$ it will encourage feasible solutions over infeasible ones.

The collision term is also, in the same way as the acceleration term, a hard constraint and defined as

$$C_c = \begin{cases} 0 & \text{if no collision} \\ P & \text{else} \end{cases} \tag{3.32}$$

therefore

$$C_c \in \{0\} \cup \{P\} \tag{3.33}$$

where P is the same parameter defined for the acceleration term. The collisions were calculated in the same way as presented in section 3.1.4.

3.3 Implementation

In order to maximize the efficiency of the algorithms mentioned above, it is necessary to utilize the parallel nature they offer. This means simply that it is necessary to find a way to implement this in the software stack. The solution for this problem comes in the form of a framework called OpenCL, which was briefly mentioned in section 1 and 2.2. The framework allows for writing programs that are able to execute across heterogeneous platforms consisting of CPUs, GPUs, digital signal processors, field-programmable gate arrays and other similar processors. OpenCL uses a specific programming language that is based on C99, which is a standard C coding language. Thus, all devices or programs that are supposed to use the framework will have to be written in the C99 language. The parallel computing interface OpenCL offers is task-based and data-based parallelism.

The hierarchy of OpenCL may be viewed as if it consists of a number of compute devices and are passed instructions to perform specific tasks given by the framework. The number of available compute devices are hardware specific and thus GPUs generally have more devices than CPUs, which is an effect of their parallel architecture. A compute device is made up of several compute units, which in turn consists of multiple *processing elements* (PEs). Since OpenCL is an open standard, it is up to the hardware vendors how to divide up a compute device into compute units and PEs. To use the OpenCL framework, one has to write functions that are denoted as kernels. Depending on how the kernel is written, it is possible for it to run over all or many compute devices in parallel.

Alongside the C-like programming language, OpenCL has also defined an *application programming interface* (API) that allows regular programs running on the host to launch kernels on the compute devices and manage device memory which is separated from the host memory. OpenCL programs are mostly intended to depend on online compiling, meaning that it is compiled at run-time. This allows OpenCL to be portable between implementations for various host devices. The OpenCL standard defines host APIs for C and C++, where the latter will be used in this thesis. An illustration of the overview is shown in Fig. 3.10.

Driver model control interface

The interface with the host code and OpenCL for this thesis is a task-based parallel implementation. The way the host code uses the static driver model parameters to evaluate a plan each iteration can be viewed as a task. Therefore by porting the framework to OpenCL we allow for it to use its parallel structure to distribute and run the tasks in parallel, which means that it is possible to evaluate a different set of plans, given different sets of driver model parameters.

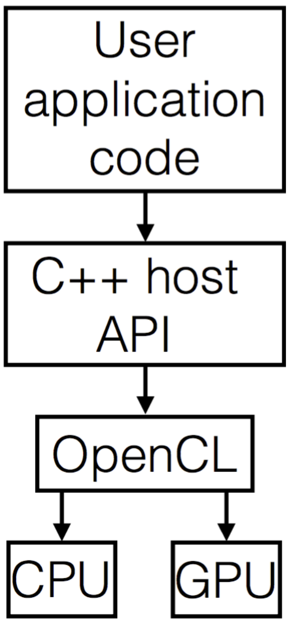


Figure 3.10: *A simple illustration over the program flow with OpenCL.*

However, for this to work all of the data concerning the current TSP has to be passed to the OpenCL device memory since it does not share the same memory with the host side. Thus, for each iteration it is necessary to set up a data transmission between the host and device side, which for a real time implementation can quickly become a bottle-neck.

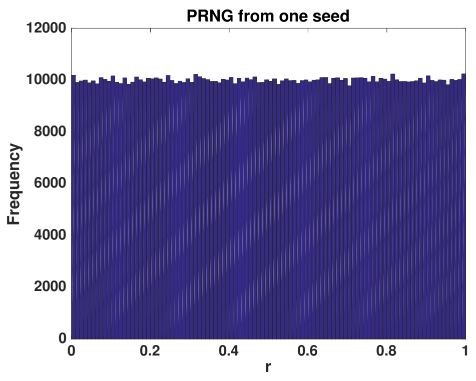
After the host side has sent all of the needed information to the OpenCL device side, the optimization is performed on the device side with instructions from the host side. Therefore the host side may tell the device that it should run N iterations of the PSO or GA. With the API, the host code then calls the OpenCL kernels to exectue the algorithm steps N times. No data needs to be sent during these steps since it is all still loaded and updated on the device side, only an instruction has to be sent from the host telling the device it should run the code. When the kernel finishes with its instructions, the output data of the kernel is read back to the host side. With the received data it is then possible to see wheather any feasible solution was found or not. If there does exist a new solution, the host side will update its parameters and simply run with the freshly updated ones. If not, then the framework will follow its decision making system to decide the next step.

3.3.1 Pseudo random number generation

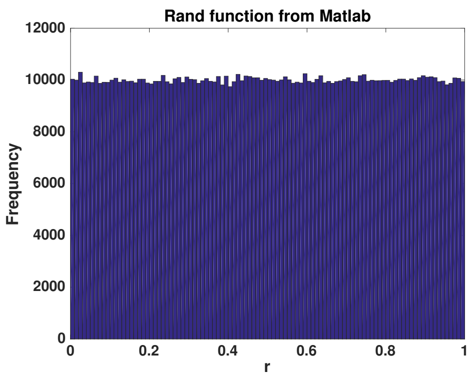
The proposed algorithms that will be used for optimization have one common factor, namely stochastic processes. Since the algorithms depend on randomness, it is of great essence to have a good random number generator. However, one problem is that creating purely random numbers in a computer is not possible. What is instead used are so called *pseudo random number generators* (PRNG). OpenCL by nature does not support, or have, any random generating function as one can find simply in most other programming frameworks. This creates a problem, since the algorithms to be used in the optimization rely on random numbers to be successful.

A solution for this problem is to implement a reasonably fast PRNG on the OpenCL device side. Existing work shows theoretic approaches for PRNGs and prove that they work well [31]. The algorithm itself is simple and an excerpt from the OpenCL code is presented below. In the original work by the authors, the algorithm generated pseudo random integers. Since the PSO and GA wants random variables in the range of $r \in [0, 1]$ the algorithm is just rescaled to fit the correct ranges.

```
// Pseudo random algorithm to generate random numbers on GPU.
// Input 0 < seed < 1, and get a new one within same range.
float rand(float seed)
```



(a) *PRNG algorithm from [32].*



(b) *Matlabs rand function.*

Figure 3.11: *Differences between the PRNG and Matlab* `rand()` *function.*

```
{
  unsigned int const a = 16807;
  unsigned int const m = 2147483647;

  int tmp = seed * m;
  tmp = (tmp * a) % m;

  return tmp / (float)m;
}
```

As a test of the algorithm a random seed is initialized using Matlab and the seed is used to generate a sequence of one million random numbers. The distribution can be seen in Fig. 3.11.

One could also solve the random number problem by generating random numbers for each iteration of the algorithm on the host side and keep sending random numbers to the buffer. This solution works fine in theory, but is in practice is very unpractical. It creates a dependency between the device and host side, where the device will have to wait for the host to send new random numbers before continuing. Due to the time consumption of transferring memory between each iteration, this solution is not good in a real-time application.

3.4 Scenario

In order to evaluate how well the optimization works, it is desired to try it on a rough scenario. Since the optimization parameters are including both the lateral and longitudinal parameters the scenario which utilizes a maneuver that takes into account both braking and steering, is namely a lane change. This section will paint the proposed scenario to test the functionality of the framework given the optimization scheme. It will therefore be investigated under which circumstances the system manages to perform safe lane maneuvers.

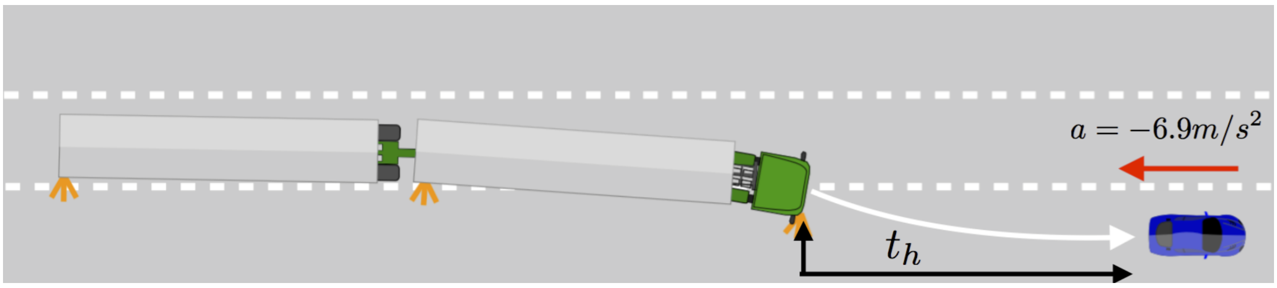


Figure 3.12: *Set up of the lane change scenario.*

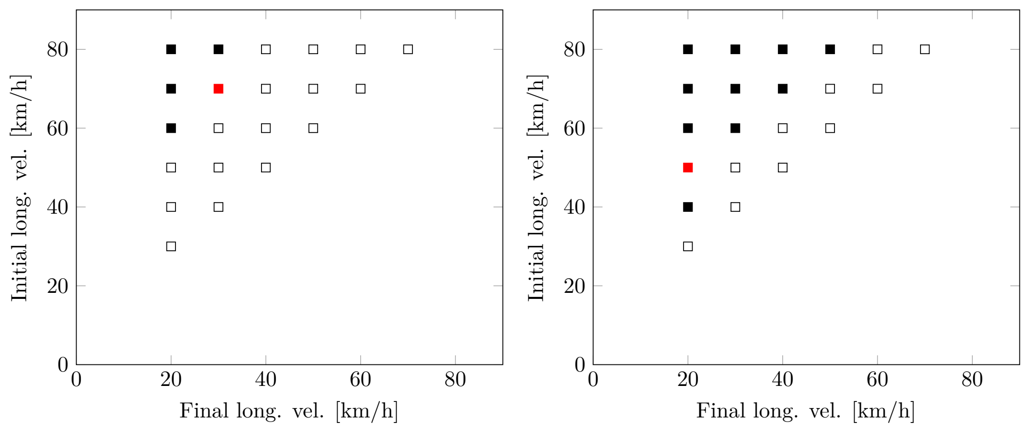
The start of the scenario consists of the LCV driving in a given lane, and an obstacle with a temporal headway t_h ahead is in the right lane of the LCV. At a given signal the LCV will get a request to make a right lane change. Thereafter a lane change will be initialized when the frameworks decides that the conditions for a lane change initialization is fulfilled, which is handled by the decision-making of the framework [28]. As the LCV passes into the target lane, the vehicle in front instantly starts to decelerate with $6.9\,\mathrm{m\,s^{-2}}$ to a target velocity v_f. The main illustration can be seen in Fig. 3.12. The possible solutions by the framework then are to either continue the lane change and brake behind the vehicle or abort the lange change and return to the original lane. In some cases the abort lane change is the only feasible solution. Actuator limits, together with small headways and large velocity differences makes it not possible to brake behind the vehicle.

The parameters that will be altered in the scenario are v_i, v_f and t_h. v_i denotes the intial velocities for the truck and the vehicle, and they will always be the same. v_f is specific for the braking vehicle. By changing these parameters it is easy to find situations where the scenario becomes too critical, in which it is not possible to complete a lane change given the dynamics of the LCV.

3.4.1 Previous results

The used framework has been evaluated on the same scenario [2], however at that time it was still using static driver model parameters. An extract from the results can be seen in Fig. 3.13, which shows how well the framework performed in more critical situations. It is visible that having a very low temporal headway, the framework did not manage to handle the cases very well with high differences in the initial and final velocities.

The interesting aspect here is that given the results from the braking lane change [2], they serve as a benchmark for the optimization. Thus, the expecting outcome is to clearly see the effect of optimization, and to see whether it is possible to find solutions that do not end up in an emergency braking or preparing for a collision.



(a) *Temporal headway* $t_h = 1.0$ (b) *Temporal headway* $t_h = 0.5$

Figure 3.13: *Performance of the framework while changing lane to the right. The unfilled boxes correspond to a succesful lane change, the red boxes correspond to a successful abort maneuver and the black boxes correspond to an emergency brake maneuver. The initial velocities of both the truck and vehicle were set in the range of* $v_i \in [20, 80]\ \mathrm{km\,h^{-1}}$ *and the final velocity of the vehicle was in the range of* $v_f \in [20, 70]\ \mathrm{km\,h^{-1}}$. *The two figures show the temporal headways for the initiation of the lane changes.*

4

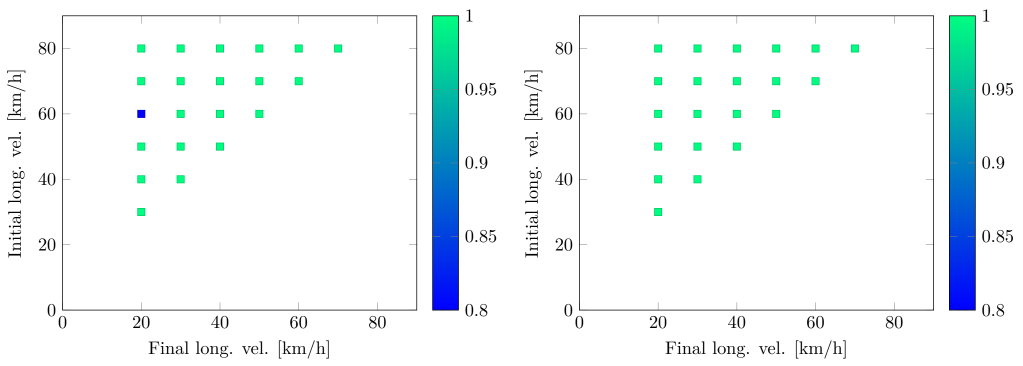
Results

The reason for simulating an emergency behaviour was to confirm and to answer the reasearch questions of the thesis. The results will show if optimizing the driver model parameters may lead to avoid dangerous situations which would end up in collisions if nothing were to be done. Thus, this chapter will show how well the framework performed given an emergency scenario. Simulation times of the optimization were tracked as well as different states throughout simulation.

Using the simulating framework the presented scenario described in the previous chapter was implemented. The results shown were gathered using a single track bicycle model throughout the closed-loop but also from Matlab/Simulinks outer-control loop. Therefore the validity of all results, at least in the extreme cases, should be taken with a bit of caution. Having a simplified bicycle model to describe the dynamics of a LCV might cause significant differences when putting actutation requests on the vehicle.

4.1 Lane changes with a lead vehicle braking

Figs. 4.1 and 4.2 presents how well the framework complied to a lane change where a vehicle in front brakes heavily as soon as the target lane is reached. The results were gathered using the proposed parallel optimization scheme where driver model parameters are optimized. The population and swarm size for the simulation was fixed to 128.

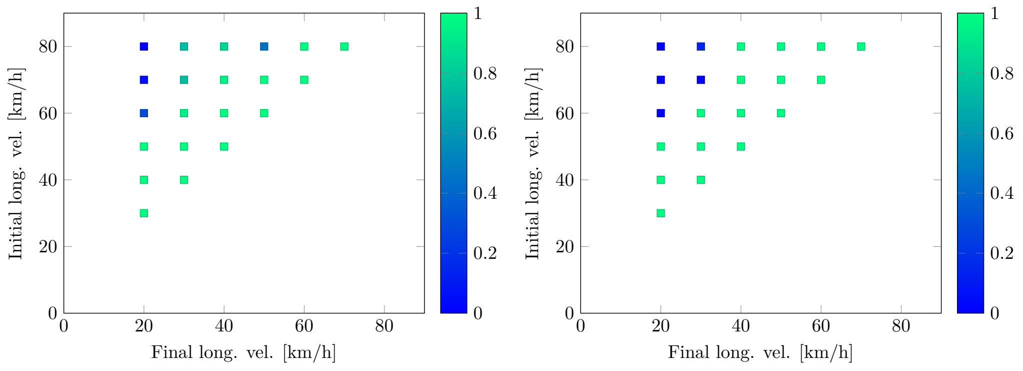


(a) *Temporal headway $t_h = 1.0$ for the GA* (b) *Temporal headway $t_h = 1.0$ for the PSO*

Figure 4.1: *Performance of the framework while changing lane to the right. The colored boxes represent the percentage of successful completed or aborted lane changes. A maneuver is seen as a failure if the system has to resort to an emergency brake solution. The initial velocities of both the truck and lead vehicle were set in the range of $v_i \in [20, 80]$ $\mathrm{km\,h^{-1}}$ and the final velocity of the vehicle was in the range of $v_f \in [20, 70]$ $\mathrm{km\,h^{-1}}$. Each box corresponds to 20 simulations where an average of the passed and failed maneuvers were taken. The lane change was initiated with a temporal headway of 1.0 s.*

Fig. 4.1 shows the simulation for when the LCV had a temporal headway of $t_h = 1.0$ s behind the leading target vehicle. It is visible that the GA only had a 80 % success rate for the lane change maneuver when the initial velocities were set to 60 $\mathrm{km\,h^{-1}}$ and the final velocity of the vehicle was set to 20 $\mathrm{km\,h^{-1}}$. It did however manage to successfully complete all other simulations. The PSO on the other hand showed to be successful for all test cases.

Studying the results in Fig. 4.2 it is visible that the difficulty of the problem has increased, thus the high



(a) *Temporal headway $t_h = 0.5$ for GA* (b) *Temporal headway $t_h = 0.5$ for PSO*

Figure 4.2: *Performance of the framework while changing lane to the right. The colored boxes represent the percentage of successful completed or aborted lane changes. A maneuver is seen as a failure if the system has to resort to an emergency brake solution. The initial velocities of both the truck and lead vehicle were set in the range of $v_i \in [20, 80]$ km h^{-1} and the final velocity of the vehicle was in the range of $v_f \in [20, 70]$ km h^{-1}. Each box corresponds to 20 simulations where an average of the passed and failed maneuvers were taken. The lane change was initiated with a temporal headway of 0.5 s.*

failing percentage of certain test cases. Neither of the two algorithms were able to find solutions in the range of having an initial velocity from 60 km h^{-1} to 80 km h^{-1} and the final velocity of the vehicle at 20 km h^{-1}. With the initial velocity of 60 km h^{-1} and a final velocity of 30 km h^{-1} the GA has a 95 % success rate. However, increasing the initial velocity to 70 km h^{-1} or 80 km h^{-1} brings the success rate down to 75 %. The PSO on the other hand manages always to find solutions when the initial velocity is 60 km h^{-1} and the final velocity is 30 km h^{-1}. Increasing the initial velocity further to 70 km h^{-1} or 80 km h^{-1} results in the algorithm always failing and having a success rate of 15 % respectively.

For the initial velocities of 70 km h^{-1} and 80 km h^{-1}, and final velocity of 40 km h^{-1} the GA has a success rate of 85 % and 95 % respectively. The last deviating result is seen at the initial velocity 80 km h^{-1} and final velocity of 50 km h^{-1} where the GA only finds a solution 45 % of the time. The PSO, while being more prone to errors in the more critical part of the test cases, shows a 100 % rate on the rest of the cases.

4.2 Run time of the optimization

When gathering results for the timing of the algorithm the following hardware was used:

- 2.6GHz Intel Core i5 CPU
- Intel Iris 1536 MB GPU

In order to gather run time data, only the part of the DMC with predictions was used. By enforcing that the prediction loop always ran for the entire prediction horizon, 75 time steps, a fair upper limit estimate could be obtained. Table 4.1 and 4.2 shows the average run times and the standard deviation of the GA when changing the number of iterations and population size. After passing the a population size of 256 the run times of the GPU is lower than of the CPU. Table 4.3 and 4.4 shows the average run times and the standard deviation of the PSO when changing the number of iterations and swarm size.

4.3 Swarm and population sizes

Here the number of iterations were constantly set to 20 and the population size and swarm size for the GA and PSO were changed in the range from 64 to 1024. Fig. 4.3 shows the outcome from the algorithms running a

Table 4.1: GPU run times for the GA.

(a) Average times across different population sizes and iteration numbers

Population size	20 iterations [s]	40 iterations [s]	60 iterations [s]	80 iterations [s]
64	0.145 79	0.276 43	0.384 82	0.563 59
128	0.146 32	0.282 79	0.418 45	0.570 13
256	0.268 31	0.516 75	0.850 16	0.997 97
512	0.365 55	0.641 13	0.916 31	1.306 37
1024	0.467 32	0.904 24	1.290 23	1.707 90

(b) Standard deviation across different population sizes and iteration numbers

Population size	20 iterations [s]	40 iterations [s]	60 iterations [s]	80 iterations [s]
64	0.016 782	0.036 27	0.003 60	0.091 52
128	0.003 222	0.023 17	0.050 15	0.043 12
256	0.006 189	0.022 07	0.111 01	0.008 22
512	0.047 193	0.024 64	0.018 63	0.060 45
1024	0.013 336	0.032 84	0.012 38	0.007 52

Table 4.2: CPU run times for the GA.

(a) Average times across different population sizes and iteration numbers

Population size	20 iterations [s]	40 iterations [s]	60 iterations [s]	80 iterations [s]
64	0.058 66	0.114 06	0.168 43	0.233 81
128	0.101 81	0.190 55	0.286 08	0.375 84
256	0.193 69	0.387 96	0.562 87	0.794 13
512	0.471 21	0.804 33	1.219 23	1.590 47
1024	0.691 23	1.350 74	2.852 09	2.767 81

(b) Standard deviation across different population sizes and iteration numbers

Population size	20 iterations [s]	40 iterations [s]	60 iterations [s]	80 iterations [s]
64	0.016 78	0.036 27	0.003 60	0.091 52
128	0.003 22	0.023 17	0.050 15	0.043 12
256	0.006 19	0.022 07	0.111 01	0.008 22
512	0.047 19	0.024 64	0.018 63	0.060 45
1024	0.013 33	0.032 84	0.012 38	0.007 52

similar simulation as in section 4.1. The scenario was set out to start its lane change with a temporal headway of 0.5 s and the initial velocities of $80\,\mathrm{km\,h^{-1}}$, and the final velocity of the leading vehicle of $30\,\mathrm{km\,h^{-1}}$. The best outcome was given by the GA at a population size of 512 individuals. Lane changes that were successfully completed or aborted were counted as a success, whereas a lane change that entered the emergency brake state at any time was discarded as a failure.

Fig. 4.4 shows the characteristics of a successful abort from a initialized lane change. From $t = 0$ s the lane change is initialized with a temporal headway of $t_h = 0.5$ s. At $t = 4.35$ s the first axle of the LCV has passed into the target lane and the vehicle in front starts to deceellerate constantly with $-6.9\,\mathrm{m\,s^{-2}}$. The LCV continues to try to complete the lane change until $t = 5.0$ s where the framework no longer can find a solution to complete the lane change. From that point the LCV starts an lane change abort plan. Fig. 4.5 shows the active driver model parameters for the same simulation as above. Throughout the simulation it was required to update the parameters five times in total.

Adding to the simulation that the acceleration of the vehicle ahead is known, and not approximating each closed loop to have a constant velocity, the optimization is able to pass even the toughest scenario of having an

Table 4.3: GPU run times for the PSO.

(a) Average times across different swarm sizes and iteration numbers

Swarm size	20 iterations [s]	40 iterations [s]	60 iterations [s]	80 iterations [s]
64	0.164 29	0.313 08	0.441 68	0.538 21
128	0.173 41	0.326 97	0.451 97	0.564 51
256	0.222 26	0.396 18	0.551 41	0.685 95
512	0.295 68	0.578 97	0.828 88	1.055 59
1024	0.405 44	0.807 77	1.215 41	1.572 85

(b) Standard deviation across different swarm sizes and iteration numbers

Swarm size	20 iterations [s]	40 iterations [s]	60 iterations [s]	80 iterations [s]
64	0.005 25	0.007 66	0.020 35	0.010 62
128	0.007 11	0.005 66	0.006 82	0.006 78
256	0.055 82	0.011 12	0.012 10	0.008 79
512	0.009 87	0.011 91	0.017 63	0.017 73
1024	0.007 30	0.038 38	0.049 88	0.065 91

Table 4.4: CPU run times for the PSO.

(a) Average times across different swarm sizes and iteration numbers

Swarm size	20 iterations	40 iterations	60 iterations	80 iterations
64	0.053 26	0.110 31	0.155 28	0.200 39
128	0.092 81	0.179 54	0.285 61	0.366 63
256	0.164 59	0.326 48	0.479 85	0.638 66
512	0.317 39	0.645 23	0.886 68	1.196 43
1024	0.597 58	1.174 18	1.771 13	2.384 26

(b) Standard deviation across different swarm sizes and iteration numbers

Swarm size	20 iterations	40 iterations	60 iterations	80 iterations
64	0.007 79	0.019 47	0.018 69	0.020 26
128	0.006 76	0.021 88	0.051 68	0.050 59
256	0.012 77	0.037 21	0.042 22	0.053 35
512	0.027 81	0.098 74	0.024 96	0.053 22
1024	0.026 02	0.008 37	0.042 31	0.100 32

initial velocity of 80 km h^{-1} and final velocity of 20 km h^{-1} with 100 %, given by a total of 20 simulation runs. For the generated results the PSO algorithm was used having a swarm size of 512 particles and 20 iterations of the algorithm was used.

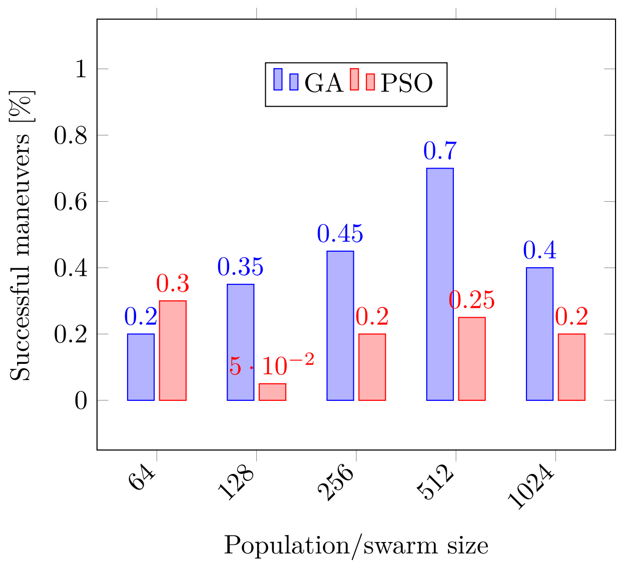
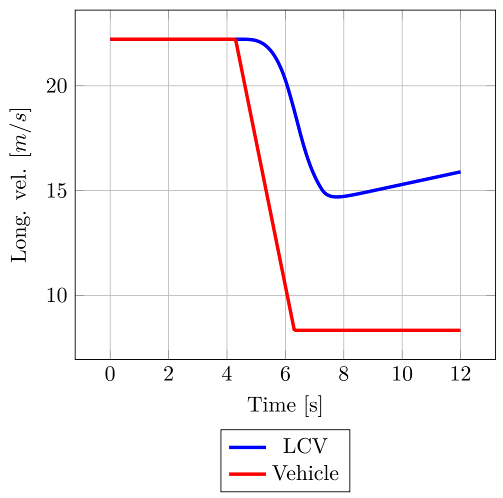
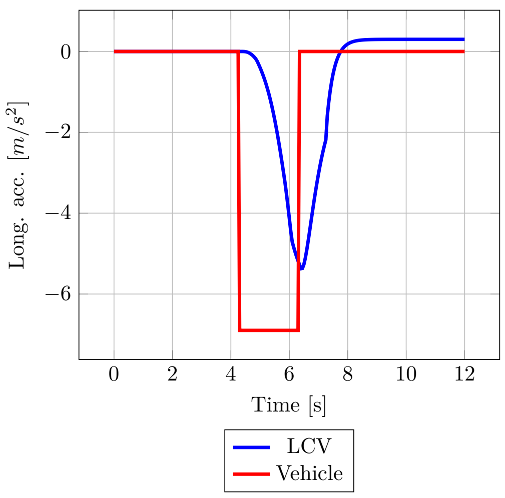


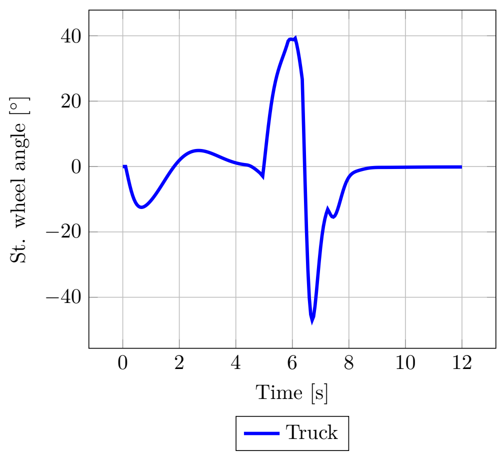
Figure 4.3: *Benchmark of the two optimization algorithms on how well they perform on a given specific scenario with only* 20 *iterations left for the optimization. The population and swarm sizes were varied in the range from* 64 *to* 1024 *with a power of two in between. A statistical average was taken of* 20 *independant simulations for each populaiton and swarm size.*



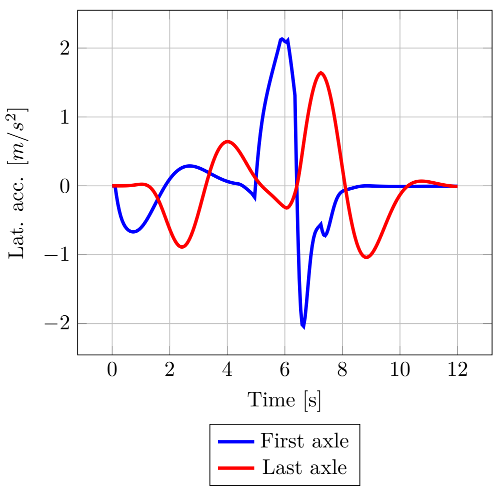
(a) *Longitudinal velocity profile*



(b) *Longitudinal acceleration profile*



(c) *Steering wheel angle profile*



(d) *Lateral acceleration profile*

Figure 4.4: *Characteristics of a succesfully aborted lane change where the vehicle starts breaking when the target lane is entered. The initial velocities of both vehicle and LCV was set to* $80\,\mathrm{km\,h^{-1}}$*, and the final velocity of the vehicle was decellerated down to* $30\,\mathrm{km\,h^{-1}}$*. The lane change was initiated with a temporal headway of* 0.5 *s. The figure displays the longitudinal velocity (top left), longitudinal acceleration (top right), steering wheel angle (bottom left) and lateral accelerations on the first and last axle (right).*

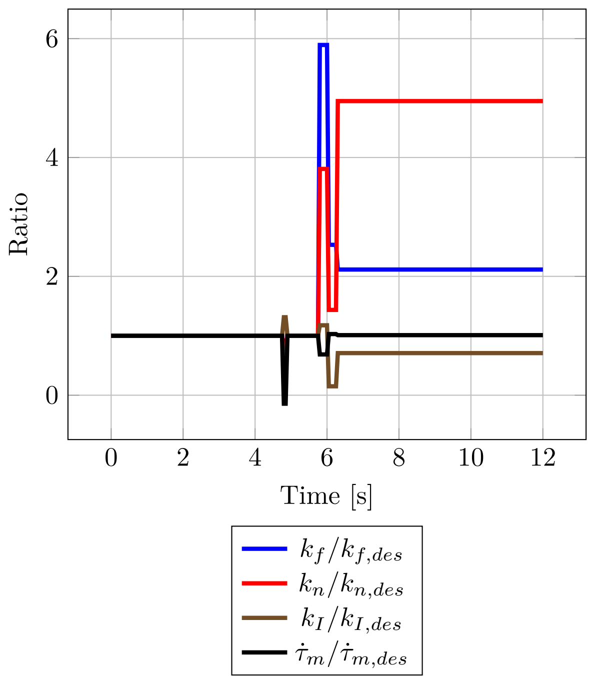


Figure 4.5: *The ratio of the actual and desired driver model parameters for the simulation presented in Fig. 4.4.*

5 Discussion

The initial process of the thesis was to implement functionality to an existing framework in form of parallel processing. OpenCL was used and allowed for the opportunity to run the simulations both on the CPU and GPU hardware without any difficulty or necessity to rewrite any of the existing OpenCL code. The open standard is well documented and easy to use since it is very similar to the C/C++ language.

The lane change scenario with optimized parameters managed to solve up to nine cases more than previously possible when having static driver model parameters. An interesting observation was that the GA and PSO had different success rates at different cases. Most of the cases had 100%, or very close to 100%, success rates. Thus the stability of the solutions seem to be working well enough, however in order to implement this in a real production vehicle it would need to be evaluated more extensively with more simulations and different scenarios. Since the predictions assumed a constant velocity at each time step, there is an introduced mismatch. With the lack of information about the acceleration, the framework is thus tricked into trying to complete a lane change, even if it is dangerous and it should abort earlier. Each time step the vehicle in front will be closer than previously predicted and thus most likely another optimization will be required for that case. Because of the problem being continuous the PSO in this case shows small tendencies to being advantageous between the iterations since all but one scenario where the solution was found a success rate of 100% was achieved. The GA on the other hand managed to find solutions in more cases, but less frequent than the PSO.

The reason behind this difference might be seen in the way how both algorithms work. The PSO in essence follows a swarm that is occupying the search space and thus more likely to follow in the direction towards the best particle. With the problem being continuous, and a mismatch is introduced, it would be possible that the best solution is therefore shifted only a little bit between the iterations and the PSO needs only to make a little change to account for the mismatch. The GA on the other hand works in a bit more of a stochastic way, since crossovers and mutations of the chromosomes can have drastic effects in to the decoded search space. Thus, it cannot utilize the search direction effectiveness of the PSO and perhaps that is why it is not always able to find solutions and fails more frequently. The whole process could be seen as walking on a narrow beam. The solution is only found along the thin path, and everywhere else it is invalid. Thus the PSO is advantageous given its information about the search direction while the GA is jumping around in a more random manner.

The run times indicate that the PSO implementation was marginally faster than the GA. The reason behind this is that the PSO has the positions encoded already and acted upon during the updating process, whereas the GA needs to decoded each chromosome into the search space, and then later on update the population by looping through each allele of the chromosome. Thus by lowering the number of genes for each variable the speed for the GA should increase, but the resolution of the search space is thus lowered in return.

The run times per se showed that they only favored the GPU at high numbers of population and swarm sizes. This is to be expected since the GPU was designed to have a parallel architecture. What it does lack though is the flexibility of the CPU and is therefore not as fast in the lower numbers. What limits the GPU in this case is the amounts of data transfer needed for the algorithms. Sending the data to the GPU memory buffer takes time and is a bottle neck in this case. The difference of changing from 64 to 128 particles or individuals is barely visible in the timing results for the GPU. The run time for the CPU is affected almost twice by the same change.

With the given problem at hand, it was from the start hinting that the GPU would have a hard time keeping up with the CPU if one would compare the threads with each other on an individual level, since a single CPU thread is clearly more powerful than a single GPU thread. The part of the framework that was ported to OpenCL was the entire time horizon prediction, which included multiple integrations of the complex non-linear vehicle model, but also a number of non-linear constraint checks. In essence, the code that ran through OpenCL was not broken down in smaller pieces, because of it being impractical, but actually ran in whole as it was running previously in the non-modified framework. What this means for the GPU, is that each thread was given a large complex task to complete, compared to what GPUs usually are meant to do. In general each thread should handle an easy and small task instead of the given rather large and complex task.

Ideally it would be best to break down the problem into multiple pieces that exhibited linear properties, so that it would be possible to perform parallel computation in a form of matrix multiplication at the lower levels, instead of the higher levels as it currently has been implemented. However, as the problem is formulated now, it is not possible to do so because of the non-linear properties. Therefore the most straight forward solution was to implement the parallel computation on a higher level, with the down side of having a rather complex task.

If it is envisioned to have a real time system with an update frequence of $10 - 20$ Hz, and provided with the current hardware of the system, the CPU shows tendencies to be the better choice for this problem. Even though its parallel capabilities are less than of the GPU, the CPU has the strength and speed to overtake the power of the GPU. What should be noted in this specific case is that the implemented OpenCL code is a rewritten version of normal code that usually runs on the CPU. This means that the GPU is not being used in the conventional way of performing a large number of repeated tasks, but instead is solving a larger task that is often suitable to be run on a CPU, with conditionals statements and integrations etc. Currently the optimization is performed on either CPU or GPU, meaning that while the CPU or GPU is running, the other is waiting. To increase effectivity a proposed improvement is to interface both CPU and GPU in order to utilize most of the resources at hand.

It should be noted that the computational system used for the evaluation of the optimization did not use state-of-the-art hardware. Therefore, by upgrading both CPU and GPU hardware, it should be possible to decrease the run times even further. It may be argued that the advancement of CPU technology has stagnated a bit, but a new generation of GPUs are on the rise. With the coming years bringing newer and better GPUs, the importance of GPU programming will probably become of higher importance and by then this problem might be more suitable for the GPU.

GAs and PSOs are generally known to work well with a limited population and swarm size, and instead using a higher number of generation or iterations. From a real time perspective this is not suitable since each iteration step depends on the other, the population and swarm size on the other hand benefits from the parallel properties. From the run time table it is visible that real time possibilities are achievable with at least 20 iterations. From the specific lane change scenario the effect of lowering the iteration steps but varying the population and swarm sizes. The PSO performed better with 64 particles at 20 iterations with 30% success rate than it did for 128 particles at 80 iterations. Whereas the GA reached 70% success rate with 512 individuals, which is a slight decrease in performance from the previous 80% with 128 individuals and 80 generations. This shows tendencies that it is possible to reach valid solutions with lower iteration steps and thus achieve a real time feasible implementation.

As it was stated in the results, including also surrounding vehicle accelerations, the introduced mismatch to the framework is removed and the system is given the best case scenario to work with. With this information the framework immediately predicts that the vehicle in front is decelerating, and assuming it will come to a full stop, it realizes that it cannot complete the lane change. Therefore it initiates the lane change abort maneuver. However, estimating acceleration of other moving vehicles is not always an easy task and it gives rise to very noisy data. Using at least any information regarding the acceleration should give rise to better performance of the framework in general.

6 Conclusions

The effect of optimizing driver model parameters has been studied in this thesis using an existing framework for automated driving. The framework uses a driver model for the navigation of a LCV in order to achieve high driver acceptance behaviour. In order to research of the effect of changing driver model parameters, an optimization scheme has been proposed and implemented in the existing framework. The underlying methods for the optimization consisted of biologically inspired algorithms such as the well-known GA and PSO. The algorithms were implemented and used due to their simple form, but also highly parallel nature. It was envisioned that exploiting the parallel nature of the algorithms, the optimization would be suitable for real-time applications.

In addition, the programming framework OpenCL was used in order to efficiently implement the parallel optimization. Using OpenCL, it was possible to execute the code on either GPU or CPU hardware, without any modification of the code. This property made it easy to perform a comparison of performance for both hardwares. However, in order for this to work a large part of the existing automated driving framework needed to be rewritten and implemented in the OpenCL standard. This included implementing the integration for the LCV and surrounding vehicle in the closed-loop prediction horizon.

The parameters for the longitudinal and lateral driver models were used as variables for the two algorithms. The GA encoded its chromosomes with values for each parameter, whereas the PSO used the spatial position of its particles to encode the variables. Each particle, or individual, was evaluated under a prediction horizon, where a performance value was obtained. Implementing the update rules for each algorithm and iterating a fixed number of steps, convergence to a set of feasible parameters could be achieved for both algorithms.

Evaluation of the proposed optimization scheme was done by comparing results from a previous study where the driver model parameters were fixed. The scenario consisted of a lane change with a breaking lead vehicle. By altering the initial temporal headway and initial and final velocities between the LCV and the leading vehicle, dangerous situations could arise which the previous framework could not handle.

Optimization of driver model parameters showed that for the same given dangerous scenario it successfully aborted or completed the maneuvers within the safety limits. The framework managed to safely perform most of the scenarios except when the initial velocites were $60\,\mathrm{km\,h^{-1}}$ to $80\,\mathrm{km\,h^{-1}}$ and the final velocity was $20\,\mathrm{km\,h^{-1}}$ but also when the initial velocities were $70\,\mathrm{km\,h^{-1}}$ to $80\,\mathrm{km\,h^{-1}}$ and the final velocity was $20\,\mathrm{km\,h^{-1}}$. Comparing to the previous framework with static driver model parameters, this was clearly an improvement.

Even though using a stochastic optimization approach, the cases where the algorithms did succeed to find solutions was very seldom far away from 100%. However, in order for this to be used in practice, the convergence assurance needs to be investigated further. If the automated driving framework were to include any form of estimated acceleration of surrounding traffic, should improve the performance of the framework and allow it to react to certain events just in time before it is too late.

With the parallel optimization approach it was visible that the number of iterations was the key factor affecting the run-time since it could not be done in parallel. The results hinted that by lowering the number of iteration steps for the algorithm, a real-time implementation lies within the limits given the current implementation. However, it would be preferred to use upgraded hardware and optimization of the code in order to bring the run-times to a possibility of a real time feasible implementation.

Bibliography

[1] John Aurell, Thomas Wadman, and Volvo Trucks. Vehicle combinations based on the modular concept. *NVF-reports, Report*, 1, 2007.

[2] Peter Nilsson, Leo Laine, Niels Van Duijkeren, and Bengt Jacobson. Automated highway lane changes of long vehicle combinations: A specific comparison between driver model based control and non-linear model predictive control. In *Innovations in Intelligent SysTems and Applications (INISTA), 2015 International Symposium on*, pages 1–8. IEEE, 2015.

[3] Kirill Van Heerden, Yasutaka Fujimoto, and Atsuo Kawamura. A combination of particle swarm optimization and model predictive control on graphics hardware for real-time trajectory planning of the under-actuated nonlinear acrobot. In *Advanced Motion Control (AMC), 2014 IEEE 13th International Workshop on*, pages 464–469. IEEE, 2014.

[4] Luis Miguel Rios and Nikolaos V Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293, 2013.

[5] Aaftab Munshi. The opencl specification. In *2009 IEEE Hot Chips 21 Symposium (HCS)*, pages 1–314. IEEE, 2009.

[6] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, 2008.

[7] Brian Paden, Michal Cap, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *arXiv preprint arXiv:1604.07446*, 2016.

[8] Torvald Ersson and Xiaoming Hu. Path planning and navigation of mobile robots in unknown environments. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 2, pages 858–864. IEEE, 2001.

[9] Reinhardt M Rosenberg. Introduction. In *Analytical Dynamics of Discrete Systems*, pages 1–5. Springer, 1977.

[10] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, 29(5):485–501, 2010.

[11] Jose Luis Blanco, Mauro Bellone, and Antonio Gimenez-Fernandez. Tp-space rrt-kinematic path planning of non-holonomic any-shape vehicles. *International Journal of Advanced Robotic Systems*, 12, 2015.

[12] Julius Ziegler, Philipp Bender, Thao Dang, and Christoph Stiller. Trajectory planning for bertha—a local, continuous method. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 450–457. IEEE, 2014.

[13] Janick V Frasch, Andrew Gray, Mario Zanon, Hans Joachim Ferreau, Sebastian Sager, Francesco Borrelli, and Moritz Diehl. An auto-generated nonlinear mpc algorithm for real-time obstacle avoidance of ground vehicles. In *Control Conference (ECC), 2013 European*, pages 4136–4141. IEEE, 2013.

[14] Steven M LaValle and James J Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[15] M. W. Disse. Real-time path planning for long heavy vehicle combinations. Master's thesis, TU Delft, Delft University of Technology, Delft University of Technology, 2012.

[16] NJ Van Duijkeren. Real-time receding horizon trajectory generation for long heavy vehicle combinations on highways. Master's thesis, TU Delft, Delft University of Technology, 2014.

[17] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. Acado toolkit—an open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.

[18] Peter Nilsson, Leo Laine, Ola Benderius, and Bengt Jacobson. A driver model using optic information for longitudinal and lateral control of a long vehicle combination. In *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*, pages 1456–1461. IEEE, 2014.

[19] Leonardo Dagum and Ramesh Menon. Openmp: an industry standard api for shared-memory programming. *IEEE computational science and engineering*, 5(1):46–55, 1998.

[20] Vincent Roberge, Mohammed Tarbouchi, and Gilles Labonté. Comparison of parallel genetic algorithm and particle swarm optimization for real-time uav path planning. *Industrial Informatics, IEEE Transactions on*, 9(1):132–141, 2013.

[21] Dario D Salvucci and Rob Gray. A two-point visual control model of steering. *Perception*, 33(10):1233–1248, 2004.

[22] Gustav Markkula, Ola Benderius, and Mattias Wahde. Comparing and validating models of driver steering behaviour in collision avoidance and vehicle stabilisation. *Vehicle System Dynamics*, 52(12):1658–1680, 2014.

[23] David N Lee. A theory of visual control of braking based on information about time-to-collision. *Perception*, 5(4):437–459, 1976.

[24] Peter Nilsson. *On Traffic Situation Predictions for Automated Driving of Long Vehicle Combinations*. Licentiate thesis, Chalmers University of Technology, 2015.

[25] Peter Nilsson and Kristoffer Tagesson. Single-track models of an a-double heavy vehicle combination. Technical report, Chalmers University of Technology, 2014.

[26] Hans Pacejka. *Tire and vehicle dynamics*. Elsevier, 2005.

[27] Pedro F Lima, Marco Trincavelli, Jonas Mårtensson, and Bo Wahlberg. Clothoid-based model predictive control for autonomous driving. In *Control Conference (ECC), 2015 European*, pages 2983–2990. IEEE, 2015.

[28] Peter Nilsson, Leo Laine, Bengt Jacobson, and Niels Van Duijkeren. Driver model based automated driving of long vehicle combinations in emulated highway traffic. In *Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on*, pages 361–368. IEEE, 2015.

[29] Mattias Wahde. *Biologically inspired optimization methods: an introduction*. WIT press, 2008.

[30] John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–72, 1992.

[31] Stephen K. Park and Keith W. Miller. Random number generators: good ones are hard to find. *Communications of the ACM*, 31(10):1192–1201, 1988.

[32] William B Langdon. A fast high quality pseudo random number generator for nvidia cuda. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2511–2514. ACM, 2009.