Solving the Hamilton-Jacobi-Bellman Equation for Route Planning Problems Using Tensor Decomposition

Albin Mosskull and Kaj Munhoz Arfvidsson

***Abstract*—Optimizing routes for multiple autonomous vehicles in complex traffic situations can lead to improved efficiency in traffic. Attempting to solve these optimization problems centrally, i.e. for all vehicles involved, often lead to algorithms that exhibit the curse of dimensionality: that is, the computation time and memory needed scale exponentially with the number of vehicles resulting in infeasible calculations for moderate number of vehicles. However, using a numerical framework called tensor decomposition one can calculate and store solutions for these problems in a more manageable way. In this project, we investigate different tensor decomposition methods and corresponding algorithms for solving optimal control problems, by evaluating their accuracy for a known solution. We also formulate complex traffic situations as optimal control problems and solve them. We do this by using the best tensor decomposition and carefully adjusting different cost parameters. From these results it can be concluded that the Sequential Alternating Least Squares algorithm used with canonical tensor decomposition performed the best. By asserting a smooth cost function one can solve certain scenarios and acquire satisfactory solutions, but it requires extensive testing to achieve such results, since numerical errors often can occur as a result of an ill-formed problem.**

***Sammanfattning*—Att optimera färdvägen för flertalet autonoma fordon i komplexa trafiksituationer kan leda till effektivare trafik. Om man försöker lösa dessa optimeringsproblem centralt, för alla fordon samtidigt, leder det ofta till algoritmer som uppvisar *The curse of dimensionality*, vilket är då beräkningstiden och minnes-användandet växer exponentiellt med antalet fordon. Detta gör många problem olösbara för endast en måttlig mängd fordon. Däremot kan sådana problem hanteras genom numeriska verktyg så som tensornedbrytning. I det här projektet undersöker vi olika metoder för tensornedbrytning och motsvarandes algoritmer för att lösa optimala styrproblem, genom att jämföra dessa för ett problem med en känd lösning. Dessutom formulerar vi komplexa trafiksituationer som optimala styrproblem för att sedan lösa dem. Detta gör vi genom att använda den bästa tensornedbrytningen och genom att noggrant anpassa kostnadsparametrar. Från dessa resultat framgår det att *Sequential Alternating Least Squares* algoritmen, tillsammans med kanonisk tensornedbrytning, överträffade de andra algoritmer som testades. De komplexa trafiksituationerna kan lösas genom att ansätta släta kostnadsfunktioner, men det kräver omfattande testning för att uppnå sådana resultat då numeriska fel lätt kan uppstå som ett resultat av dålig problemformulering.**

***Index Terms*—Autonomous vehicles, HJB equation, Tensor decomposition, Tensor Train decomposition**

***Supervisor:** Elis Stefansson*

TRITA number: TRITA-EECS-EX-2020:119

I. Introduction

A. Motivation

Around 3700 people die in traffic every day [1]. While there are a number of reasons behind these deaths, one can not neglect the human factor. Autonomous vehicles have the potential to reduce the number of deaths, by improved communication between vehicles and more responsible driving. Another great benefit with autonomous vehicles is that they free up time for commuters in an ever more urbanised world [2] [3]. However, in order for these autonomous vehicles to act safely and efficiently, one needs smart algorithms that calculate their optimal routes. Consider for example the scenario in Fig. 1.

Fig. 1. Two autonomous vehicles driving on a bilateral road approaches a roadblock (yellow and black).

Here, two autonomous vehicles are approaching a road block from each side of a bilateral road, potentially yielding a conflict between the two vehicles. One approach to this problem is to let each vehicle try to calculate their path individually. However, the decision of one vehicle affects the outcome of the other vehicle and vice versa: this coupling needs to be taken into account for accurate individual optimisation which in the general case is a highly non-trivial task [4] [5]. To avoid this coupling, it could be beneficial to instead solve the traffic conflict centrally. That is, include all vehicles in one single control problem and then solve it for all vehicles. Setting up the problem centrally would then avoid the coupling problem at the expense of a larger optimisation problem. In practise, this larger optimisation problem could be solved by some infrastructure with a greater computation capacity and the transmitted to the vehicles.

The single control problem can be formulated as an optimal control problem and solved via a partial differential equation known as the Hamilton-Jacobi-Bellman (HJB) equation [6]. Unfortunately, as the number of autonomous vehicles grow, the memory and number of computations required to solve the HJB equation naively increase exponentially. This phenomenon, generally called the curse of dimensionality, limits the ability to naively obtain optimal solutions. Finding efficient ways to approximate these problems is therefore crucial.

B. Contribution

The focus in this project is to investigate different ways to solve complex traffic situations as optimal control problems through numerical approximation and to evaluate different solvers performance. Common for all methods is that they use tensor decomposition in order to represent the problems with less amount of data than the naive form. The tensor decomposition methods that have been used are canonical tensor decomposition, tensor train decomposition and quantized tensor train decomposition. These will all be discussed in detail in sections below. For these decompositions, different solving algorithms have been analyzed. For the canonical form, a version of the Sequential Alternating Least Squares (SeALS) method was used [7], [8], while the algorithms Alternating Minimal Energy (AMEn) and Density Matrix Renormalization Group (DMRG) were used for the other two decomposition methods [9]. It was found that that the methods performed similarly on small-scale problems, but that SeALS method scaled best with the number of grid-points in the problem.

The project also found how a scenario with two autonomous vehicles sharing a road could be formulated as an optimal control problem. The particular key to formulating the problem in such a way that the generated simulation was successful required that the boundary conditions of the HJB equation had a smooth transition to the rest of the problem.

C. Related work

There has been several other works done with the purpose of approximating the HJB equation. In [10] the HJB equation is solved using tensor train decomposition and a policy iteration algorithm. Noise is however not considered. Other notable papers that largely influenced this work are [7], [8] which presents a toolbox for solving a linearised HJB equation using SeALS.

Apart from using optimal control theory, route planning has also been approached using game theory, as in [4] and [5]. The problem has also been approached using reinforcement learning, as in [11].

D. Outline

The remaining report is as follows. Section II introduces preliminaries, giving an overview of the HJB equation in section II-A and explaining how a linear version of it can be found in section II-B. This is followed by section II-C where the curse of dimensionality and tensor decomposition methods are explained. Section II-D then describes three algorithms for solving linear equations that are on tensor decomposed forms.

Section III depicts the evaluation of the different decomposition and solving algorithms. Section IV describe the procedure for specifying the dynamics and costs in the problem formulation for road sharing scenarios. Lastly, section V concludes the project with discussions and directions for future work.

II. Preliminaries

A. The Hamilton-Jacobi-Bellman Equation

1) Dynamics: As mentioned earlier, our approach to the route planning problem is by modelling the system as an optimal control problem. In every control system, there is some state variables that we want to control. These state variables can for example be the coordinates of an automated vehicle. The state variables change depending on which state you are in and what control you apply. This is called the dynamics of the system and can be written as

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

in which $x \in \Omega$ represents the state. This means x is some state within the state domain $\Omega \subset \mathbb{R}^d$. Further on, t represents time and u represents the control. x and u are most often dependant on time, but we denote them as simply x and u to reserve the notation $x(t)$ and $u(t)$ for discussion of trajectories. Generally we require the control to be admissible, i.e. an allowed control input $u(t) \in U$ for $t \in [t_i, t_f]$. $U \subset \mathbb{R}^m$ is a limited domain of acceptable control inputs while t_i and t_f are some start and end times. In practise this means that u is limited in what values it can take given a specific point in time. The limitation on u can in practise be that there is a limit to the amount of acceleration that can affect the vehicle at a given moment.

2) Cost: In order to find the optimal solution to a control problem, one needs to specify what is supposed to be optimal. Therefore, optimization problems are formulated as finding the maximum or minimum of a function, under some conditions. Most often this means specifying a cost function, which is supposed to be minimized. Consider a cost

$$c(t, x, u).$$

When speaking of vehicles and driving, this cost function could be set up to have high values if the vehicle deviates far from the intended path or if the speed is too high. Minimizing such a cost could assert that the vehicle follows traffic rules and expends a reasonable amount of fuel.

There is also often a desire to reach a final state x_f^*. To ensure that the solution reaches x_f^*, one can penalize for deviation from x_f^* through the terminal cost $\phi(x_f)$. When the solution reaches a boundary of the control problem, ϕ is applied. For example, this could be analogous with driving to the wrong destination.

3) Finite time HJB: Using the dynamics and cost models mentioned above, an optimal control problem starting in time t_i and ending at t_f can then typically [6] be formulated as

$$\begin{aligned} \min_{u \in U} \ \phi(x_f) + \int_{t_i}^{t_f} c(t, x, u)\, dt, \\ \text{subject to } \frac{dx}{dt} = f(t, x, u). \end{aligned} \tag{1}$$

As mentioned earlier, $c(t, x, u)$ accumulates cost during propagation in time and the terminal cost $\phi(x_f)$ penalizes for deviation from a desired final state.

Finding this u can in the example of the automated vehicle be that one knows the optimal acceleration and steering at

any given point, which is what we are trying to accomplish. This problem formulation is however too complicated to solve directly, which is why we will need to reformulate it in order to find an actual solution.

To simplify the problem formulation we define the *cost-to-go function* [6] as

$$J(t,x,u(t)) = \phi(x_f) + \int_t^{t_f} c(t,x,u)\,dt, \tag{2}$$

which is, like in (1), subject to the system's state dynamics and an admissible control trajectory. The *cost-to-go function* describes the total cost by taking a certain route $x \to x_f$.

Suppose that there is an optimal route by function of $f(t,x,u^*(t))$ with the optimal control trajectory $u^*(t)$. The cost for moving $x_0 \to x_f$ on the optimal route would then be

$$J(t_0,x_0,u^*(t)) = \min_{u\in U} J(t_0,x_0,u) \triangleq J^*(t_0,x_0).$$

with $t \in [t_i, t_f]$ and $x \in \Omega$.

By using dynamic programming and the principle of optimality [6], one can rewrite the optimal control problem as

$$\begin{aligned} &-\frac{\partial J^*}{\partial t}(t,x) = \\ &\min_{u\in U}\left\{c(t,x,u) + \frac{\partial J^*}{\partial x}(t,x)^T f(t,x,u)\right\}. \end{aligned} \tag{3}$$

This is called the Hamilton-Jacobi-Bellman (HJB) equation. Solving the HJB equation gives you the optimal cost-to-go function J^*. From J^* you can find the optimal control u^* which in our case can be used to find the optimal route.

4) First-exit HJB: Currently, (3) presumes that x_i, t_i and t_f are fixed [6]. However, in some control problems it is more beneficial to formulate the problem in regards to a set of fixed terminal states Γ instead of a fixed terminal time t_f. The terminal states could be the boundary of the state domain Ω, a desired goal region or a combination of the two. In the situation of an automated vehicle, this could mean that the problem is set to stop once it reaches the end-destination (a goal region) or if it drives off the road (a boundary of the state space).

Suppose there is an optimal cost-to-go function $V(t,x) = J^*(t,x)$, which we also will call the *value function*, for a given optimal control problem. The HJB equation (4) is by definition solved by the value function

$$\begin{aligned} &-\frac{\partial V}{\partial t}(t,x) = \\ &\min_{u\in U}\left\{c(t,x,u) + \frac{\partial V}{\partial x}(t,x)^T f(x,u)\right\}. \end{aligned} \tag{4}$$

Continuing, if the system dynamics, costs and boundary conditions are independent of time then (4) can be simplified to

$$0 = \min_{u\in U}\left\{c(x,u) + \frac{\partial V}{\partial x}(x)^T f(x,u)\right\}, \tag{5}$$

with boundary condition $V(x_f) = \phi(x_f)$ where $x_f \in \Gamma$.

5) Stochastic noise in HJB: In any practical application, there exists some kind of noise. In order to achieve accurate results it is useful to include this noise in the model of the system. This can be done by formulating the dynamics as

$$dx = f(x,u)dt + B(x)dw, \tag{6}$$

where $B : \mathbb{R}^d \to \mathbb{R}^{d\times p}$ is a noise function and $w \in \mathbb{R}^p$ is Gaussian noise with noise covariance matrix Σ_ϵ. The function B governs how the dynamics is affected by the noise w.

Using these dynamics, the HJB equation for a first exit problem, see (5), can be derived as

$$\begin{aligned} 0 = \min_{u\in U}\Bigg\{ & c(x,u) + \frac{\partial V}{\partial x}(x)^T f(x,u) + \\ & \frac{1}{2}Tr\Big(\frac{\partial^2 V}{\partial x^2}(x)\cdot B(x)\Sigma_\epsilon B(x)^T\Big)\Bigg\}, \end{aligned} \tag{7}$$

For further details of this derivation, see [12].

At this point, an equation has been derived which solves an optimal control problem that stops at a terminal state and takes noise into account. However, it is still complicated to solve in practise, since it is a non-linear partial differential equation. In the next section, we will see that the HJB equation can be made linear under some assumptions, which will make it much easier to solve.

B. The linear HJB equation

In order to make a linear version of the HJB equation (7), we will now make three assumptions. More precisely we assume that the cost $c(x,u)$ can be written as

$$c(x,u) = q(x) + \frac{1}{2}u^T R u, \tag{8}$$

where $q : \mathbb{R}^d \to \mathbb{R}$ is an arbitrary function and $R \in \mathbb{R}^{m\times m}$ is positive definite matrix. Here, $q(x)$ is a cost based on the current state, while $\frac{1}{2}u^T R u$ penalizes large control inputs. Our second assumption says that the state dynamics function f can be separated on the form

$$f(x,u) = h(x) + G(x)u, \tag{9}$$

where both $h : \mathbb{R}^d \to \mathbb{R}^d$ and $G : \mathbb{R}^d \to \mathbb{R}^{d\times m}$ are arbitrary functions. In this case, $h(x)$ is the self dynamics of the system and $G(x)$ specifies how the control input affects the dynamics.

Using these two assumptions we can now rewrite (7) as

$$\begin{aligned} 0 = \min_u \Big\{ & q + \frac{1}{2}u^T R u \;+\; (\nabla_x V)^T(h(x) + G(x)u) + \\ & \frac{1}{2}Tr\big((\nabla_{xx} V)B(x)\Sigma_\epsilon B(x)^T\big)\Big\}, \end{aligned} \tag{10}$$

yielding us the same expression as in [8]. The minimum of (10) with respect to u can now be found analytically to be

$$u^* = -R^{-1}G(x)^T(\nabla_x V), \tag{11}$$

which substituted in (10) gives

$$\begin{aligned} 0 = \; & q(x) \; + \; (\nabla_x V)^T h(x) \; - \\ & \frac{1}{2}(\nabla_x V)^T G(x) R^{-1} G(x)^T (\nabla_x V) \; + \\ & \frac{1}{2} Tr((\nabla_{xx} V) B(x) \Sigma_\epsilon B(x)^T). \end{aligned} \tag{12}$$

Observe that (12) has eliminated u, but remains a nonlinear PDE in V. Our third and final assumption to transform (12) into a linear PDE is then that there exists a $\lambda > 0$ such that

$$\lambda G(x) R^{-1} G(x)^T = B(x) \Sigma_\epsilon B(x)^T \triangleq \Sigma(x). \tag{13}$$

Introducing the *desirability function* Ψ given by

$$\Psi(x) = e^{-\frac{1}{\lambda} V(x)}, \tag{14}$$

we can rewrite (12) as

$$\begin{aligned} 0 = -\frac{1}{\lambda} q(x) \Psi \; + \; h(x)^T (\nabla_x \Psi) \; + \; \frac{1}{2} Tr((\nabla_{xx} \Psi) \Sigma) \\ \triangleq \mathcal{A}(\Psi), \end{aligned} \tag{15}$$

which is now a linear PDE in Ψ. This equation can then be seen as an operator $\mathcal{A}$ on Ψ. To sum up, we have the PDE $\mathcal{A}(\Psi)(x) = 0$ for $x \in \Omega \backslash \Gamma$, with the boundary conditions $\Psi(x) = e^{-\frac{\phi(x)}{\lambda}}$ for $x \in \Gamma$. More compactly put, consider

$$\tilde{\mathcal{A}}(\Psi)(x) = \mathcal{G}(x) \tag{16}$$

using a slightly different version of the operator in (15),

$$\tilde{\mathcal{A}}(\Psi)(x) = \begin{cases} \mathcal{A}(\Psi)(x) \text{ as in (15)}, & x \in \Omega \backslash \Gamma \\ \Psi(x), & x \in \Gamma, \end{cases}$$

and

$$\mathcal{G}(x) = \begin{cases} 0, & x \in \Omega \backslash \Gamma \\ e^{-\frac{\phi(x)}{\lambda}}, & x \in \Gamma. \end{cases}$$

The only difference between the operators is that $\tilde{\mathcal{A}}$ acts as the identity when on the boundary of the state domain. Solving (16) for Ψ then gives us the value function from (14) and lastly the optimal control u from (11), which is the desired end goal.

This problem is now much easier to solve than the nonlinear one we had earlier. This is because we can discretize (16) to $AF = G$, where A and G are discretized versions of $\mathcal{A}$ and $\mathcal{G}$ respectively. Lastly, F is the numerical solution to our problem, a discrete version of Ψ. However, one problem still remains. In order to discretize the state space naively, one typically constructs a grid which has grid points in the whole state space, which is complicated by the fact that the state space can be high-dimensional. To circumvent this, we will use tensor decomposition.

C. Tensor decomposition

A commonly used structure within this project is tensors. A tensor is a multidimensional array and by that a generalization of scalars, vectors and matrices. More formally, a d-dimensional tensor T is an element in the euclidean space $\mathbb{R}^{n_1 \times n_2 \times \ldots \times n_d}$ where $T(i_1, i_2, ..., i_d)$ is the element in T at multi-index $(i_1, i_2, ..., i_d)$, where each index is in the range $(n_1, n_2, ..., n_d)$. To stress the multi-index notation, we also write $T_{i_1, i_2, \ldots, i_d}$ to denote the full tensor T. Tensors are best understood in low dimensions. For example, if $d = 2$ and $n_1 = n_2 = 3$, then T_{i_1, i_2} is a 3×3 matrix. If instead $d = 3$ and $n_1 = n_2 = n_3 = 3$, then T_{i_1, i_2, i_3} is a $3 \times 3 \times 3$ cube. For higher-dimensions, tensors are harder to interpret visually, (but most of the intuition from low-dimensional tensors carries over to higher-dimensional ones).

Due to their structure, tensors are a natural choice for storing state spaces or operators that represent high dimensions in a discretized form. Consider two autonomous vehicles, with state variables x_1, y_1, x_2 and y_2. In order to represent each position these vehicles can have in within a state space, one would need a 4-dimensional tensor.

There will be different multiplications used within this project to combine tensors. Often when multiplying in higher-dimensions, it is the outer product

$$A_{i_1, i_2, i_3} \otimes B_{i_4, i_5, i_6} = A(i_1, i_2, i_3) B(i_4, i_5, i_6)$$

which is used. If $\cdot$ or no multiplication sign is used then it refers to the inner product

$$\begin{aligned} A_{i_1, i_2, i_3} \cdot B_{i_4, i_5, i_6} = A_{i_1, i_2, i_3} B_{i_4, i_5, i_6} = \\ \sum_i A(i_1, i_2, i) B(i, i_5, i_6), \end{aligned}$$

also known as dot product. If the factors are normal scalars, then the previous example is a scalar product.

Consider a tensor $T \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_d}$, where we for simplicity assume that $n_i = n$. Then, the number of elements in the tensor is n^d. In order words, the number of elements in a tensor scale exponentially with the dimension d prohibiting a high-dimensional tensor to be stored naively by just saving all its elements. For example, if $n = 100$ and $d = 12$ there is 10^{24} elements, which is too much for most computers to handle. This phenomenon, that the number of elements scale exponentially with increasing dimensions, is often referred to as the curse of dimensionality.

To circumvent the curse and by that make the problems manageable, tensors can be decomposed. The fundamental principle for decomposition is to approximate the tensor using less elements while still preserving an accurate representation. There are multiple formats or representations a tensor can be decomposed into, some of which will now be described.

1) Canonical decomposition: One of the first and most prominently used forms of tensor decomposition is called the canonical tensor decomposition (CANDECOMP/PARAFAC). The idea is that a tensor T can be expressed as an outer product of vectors K_{i_p} of length n_p,

$$\begin{aligned} & T(i_1, ..., i_d) = \\ & K_1(i_1) K_2(i_2) \; \ldots \; K_{d-1}(i_{d-1}) K_d(i_d) \end{aligned} \tag{17a}$$

$$\Leftrightarrow$$

$$T = K_1 \otimes K_2 \otimes \; \ldots \; \otimes K_{d-1} \otimes K_d. \tag{17b}$$

These vectors are called *canonical factors* and can be seen as cores to the tensor. The p-th core are, in this paper, written as

K_p or K_{i_p}, for $p \in \{1, 2, ..., d\}$, in comparison to a element in the core $K(i_p)$.

The canonical decomposition reduces the number of memory cells needed to store a tensor from n^d to nd. Now, instead of an exponential increase in required memory it is only linear. Still, decomposing a tensor like this is not always possible, because the representation will be too inaccurate. To solve this, one can partition the tensor into sums in which the summands individually are decomposable,

$$T = K_1^1 \otimes ... \otimes K_d^1 + ... + K_1^{\hat{r}} \otimes ... \otimes K_d^{\hat{r}}.$$

There is naturally an interest in how many terms are needed to decompose a tensor, this is said by the tensor rank $\hat{r}$. There is an incentive to have a low $\hat{r}$ and since

$$T = \sum_{r=1}^{\hat{r}} \bigotimes_{p=1}^{d} K_p^r, \tag{18}$$

and thus the required memory cells is $nd\hat{r}$.

There are many different variants of this classic canonical format. One such is Kruskal's tensor format [13], [14]. The idea is that one can factor out a constant λ^r from each element in a tensor term. Subsequently doing this for all terms results in

$$T = \sum_{r}^{\hat{r}} \lambda^r \, T^r. \tag{19}$$

The Kruskal tensor is especially useful when performing certain operations, e.g. multiplication by scalar constant. Then the operation can be done on λ, which is only $\hat{r}$ elements, instead of every element in the tensor.

2) Tensor train decomposition: The canonical decomposition is a common method to decompose a tensor into smaller entities (i.e., vectors), and thereby save memory and computational time. Another (slightly more complicated) method also common in the literature is the Tensor Train (TT) decomposition [10], [15]–[19]. The mathematical expression of a tensor in TT-format is

$$T(i_1, i_2, ..., i_d) = \sum_{\alpha_0, ..., \alpha_d} U_1(\alpha_0, i_1, \alpha_1) \; ... \; U_d(\alpha_{d-1}, i_d, \alpha_d) \tag{20a}$$

$$\Leftrightarrow$$

$$T = U_1 \cdot U_2 \cdot \; ... \; \cdot U_{d-1} \cdot U_d. \tag{20b}$$

Similar to the canonical decomposition we write either U_p, U_{i_p} or $U_{i_p}^{\alpha_{p-1}, \alpha_p}$ to denote a core, as well as, $U(\alpha_{p-1}, i_p, \alpha_p)$ for a specific element in U_p. The index order, which will be important, is $[\alpha_{p-1}, i_p, \alpha_p]$. The cores in the TT-format are notably 3-D tensors, i.e. "cubes", in comparison to the canonical cores which were vectors.

Equation (20b) is very similar to (17b) except for the difference in product operation. In the TT-format, we use inner product

$$U_p \cdot U_{p+1} = U_{i_p}^{\alpha_{p-1}, \alpha_p} \cdot U_{i_{p+1}}^{\alpha_p, \alpha_{p+1}}$$

with respect on the outer left/right indices, which would then contract the tensor and thus "connect" the cores via the index α_p. These α will be refered to as *summation indices*. The summation indices comparably replaces the tensor summations in the canonical decomposition. Consequently, the sum in (20a) would be roughly comparable to the sum in (18). Because of this structure there exists multiple ranks for a TT tensor, and they are given by the sizes of the summation indices. Consider the following core's dimension

$$\dim \; U_{i_p}^{\alpha_{p-1}, \alpha_p} = [r_{p-1} \times n_p \times r_p].$$

The rank of U_p is given by r_p, and similarly, there exists a rank for all other cores. Still, one could easily establish a bound $r_p < \hat{r}$, for $p \in \{1, ..., d\}$, to the TT tensor [15].

The structure of a TT tensor is best explained by introducing a graphical notation used by e.g. the quantum chemistry community for modern physics [19]–[21]. The notation focuses on graphically showing "connecting indices" and can simply be explained as follows.

To connect indices is to perform an inner product of two tensors. The example of an inner product between a matrix M and a vector v (2-D tensor and 1-D tensor) is shown in Fig. 2. The nodes in these diagrams are tensors of dimension equal to the number of outgoing edges. The M node has two outgoing edges and is thus a 2-D tensor (matrix). The product $M_{i,j} \, v_j$ becomes a vector and as can be seen in the figure, there is only one "loose" edge. The canonical representation is rather trivial. A tensor of rank 2 can be seen in Fig. 3. The cores are not connected through inner product but by outer product, therefore they are as one single node.

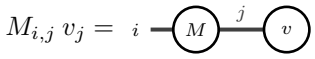

Fig. 2. Dot product of a matrix and a vector

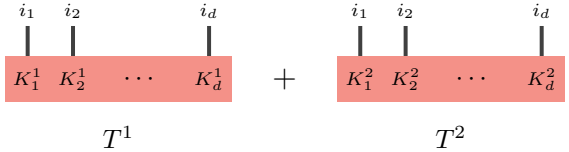

Fig. 3. The tensor network diagram for the canonical format.

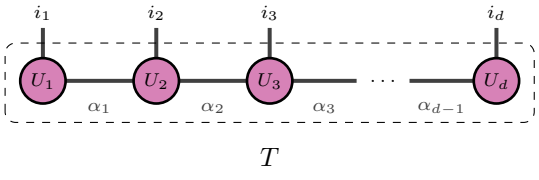

Fig. 4. The tensor network diagram for the TT format.

In the graphical notation for TT decomposition, the cores of a TT tensor are connected through the summation indices in a train-like manner (thereof the name), as shown in Fig. 4. Notably, the indices α_0 and α_d are not present in Fig. 4 yet included in (20). This highlights a "boundary condition" on the structure of the TT-format.

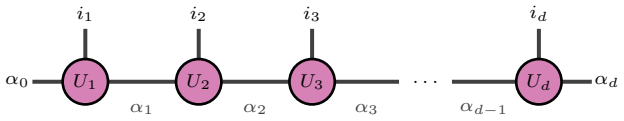

Fig. 5. Tensor in TT-format with non-trivial outer summation indices leaving the "train" with open ends.

The size of a tensor core is $r_{p-1} \times n_p \times r_p$. For the inner cores, indices connect like in the example of U_3 and U_4,

$$\begin{aligned}&\dim\ U_3 \cdot U_4 = \\ &[r_2 \times n_3 \times r_3]\,[r_3 \times n_4 \times r_4] = [r_2 \times n_3 \times n_4 \times r_4].\end{aligned}$$

But the outer cores, U_1 and U_d, must have their summation indices restricted to $r_0 = r_d = 1$. If this "boundary condition" would not be, the outer indices would have been non-trivial,

$$\dim\ T_{i_1,\ldots,i_d} = [r_0 \times n_1 \times n_2 \times \ldots \times n_{d-1} \times n_d \times r_d],$$

thus rendering the train incomplete by leaving the ends open, see Fig. 5. This is equivalent to the tensor being in wrong dimensions since the tensor T should only be dependent on the element indices i and not any summation indices α.

One of the main advantages of using the TT decomposition is its fast and relatively cheap operations [15], despite it's slightly higher memory requirement of dnr^2 [19].

3) Quantized-TT: In the creation of the TT format, a Quantized-TT (QTT) format emerged [19], [22]–[24]. This new variant seeks to give a tensor structure which can optimize algorithms, for instance, solving linear systems. The QTT structure is fundamentally not any different from the TT structure. What differentiates them is that QTT sets a condition on its mode sizes. Like the name, the QTT-format is the TT-format partitioned, i.e. *quantized*. This quantization is with respect on the element indices i, so that their mode sizes are significantly small. Most often the indices are quantized in a binary state such that $i \in \{1, 2\}$. However, the QTT-format are not restricted to only binary quantization (mode sizes $n = 2$), but could also be of another similarly small size, e.g. $n = 3$. Later on we will explain exactly how this quantization is done.

A problem formulation will most often set a condition on the resolution, i.e. the discretization grid or mode sizes, for the intended use-case. The problem may need a high resolution and thus n is large. In such cases, and if the problem is formulated in TT-format, it is easy to reshape the tensors into QTT. A requirement (if we consider binary quantization) is that the mode sizes must be a power of two, $n = 2^m$. The reshaping is done by partitioning each existing element index into m virtual dimensions.

$$\begin{aligned}&T(i_p),\ \ p \in \{1, ..., d\} \Rightarrow \text{Quantize}\ \ T \Rightarrow \\ &Q_{i_{p,\acute{p}}},\ \ p \in \{1, ..., d\},\ \ \acute{p} \in \{1, ..., m\},\end{aligned} \tag{21}$$

The new tensor Q contains more cores than T, but each individual core in Q has a smaller mode size than a core in T. Since a QTT tensor is essentially a TT tensor, under above stated condition, the mathematical expression for the QTT-format is almost the same as for the TT-format,

$$\begin{aligned}&Q(i_{1,1}, i_{1,2}, ..., i_{1,m}, i_{2,1}, ..., i_{2,m}, ..., i_{d,m}) = \\ &\sum_{\alpha_{p,\acute{p}}}^{r_{p,\acute{p}}} U_1(\alpha_0, i_{1,1}, \alpha_{1,1})\ \ldots\ U_d(\alpha_{d,m-1}, i_{d,m}, \alpha_{d,m})\end{aligned} \tag{22a}$$

$$\Leftrightarrow$$

$$T = U_{1,1} \cdot U_{1,2} \cdot\ \ldots\ \cdot U_{d,m-1} \cdot U_{d,m}. \tag{22b}$$

The tensor Q still has equally the same amount of elements as T, however, the quantization has reshaped the structure somewhat. The tensor is now of $d \cdot m$ dimensions but only of mode sizes $n = 2$,

$$\begin{aligned}&\dim\ Q = \\ &[n_{1,1} \times ... \times n_{1,m} \times ... \times n_{d,1} \times ... \times n_{d,m}] = \\ &[2 \times ... \times 2 \times ... \times 2 \times ... \times 2].\end{aligned} \tag{23}$$

As an example, consider a TT tensor of size $[4 \times 4 \times 4]$. The equivalent (binary quantized) QTT tensor would have the size $[(2 \times 2) \times (2 \times 2) \times (2 \times 2)]$.

In graphical notation this may be even more clear, see Fig. 6. In the figure, the columns represent full TT-cores U_p. Then, for each TT-core, the virtual sub-cores $U_{p,\acute{p}}$ are given as rows. Each QTT-core $U_{p,\acute{p}}$ will still have two summation indices through which they connect, one element index through which the elements are accessed, and the train-network is the same. Yet, the tensor contains over-all more cores (which individually are much smaller in size).

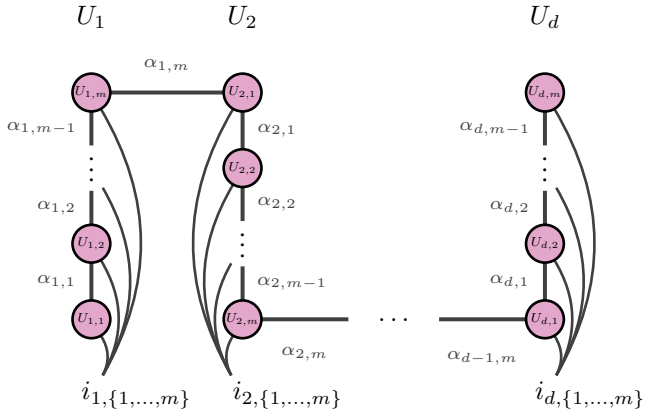

Fig. 6. The tensor network diagram for the TT format.

The QTT format results in a two-folded improvement from the ordinary TT format. The first one is the lower required memory of $d \log_2(n) r^2$ [19]. Secondly, the new format can utilize fast, efficient and accurate algorithms, by decreasing how much each element index can vary such that it is minimal but non-trivial, to solve e.g., linear equations for tensors.

D. Linear system solvers for tensor decomposition

Consider a linear system $AF = G$, where A, F and G are tensors. If F and G are of the same size, for example $R^{n_1 \times n_2 \times \ldots \times n_d}$. The operator A must be on the form $R^{(n_1 \times n_2 \times \ldots \times n_d) \times (n_1 \times n_2 \times \ldots \times n_d)}$, since that means the result

of AF has the same form as G. Since systems on this form have many off-the-shelf numerical solvers, the HJB will be rewritten like this, which is described in II-B.

A trivial example of a linear system is if $d = 1$, which means G is a vector and A is a matrix. The unknown solution F can easily be found as $A^{-1}G$, if A is non-singular.

However, in the case where A and G are in the form of decomposed versions of high-dimensional tensors as in II-C, one must use more sophisticated solver algorithms. The algorithms are based on the underlying decomposition method. Following is a brief overview of three algorithms for finding F.

1) Sequential Alternating Least Squares: In a high-dimensional linear system, using the canonical decomposition to represent tensors A, F and G, Sequential Alternating Least Squares (SeALS) can be used [7], [25]. The algorithm starts with guessing a solution F in the format of (17). It then fixates all vector products of F,

$$\text{fixate } \; K_p^r, \;\; \forall\, r \cap p \in \{1, ..., d\}/\tilde{p},$$

except for one dimension $\tilde{p}$,

$$\tilde{K} = K_{\tilde{p}}^r, \;\; \forall\, r,$$

which is used to minimize the solution error,

$$\min_{\tilde{K}} \|AF - G\|,$$

using the Frobenius norm $\|\cdot\|$. The process iterates through for all dimensions $\tilde{p} = \{1, ..., d\}$ until the *residual* $\epsilon = \|AF - G\|$ is satisfactory small.

2) Density Matrix Normalization Group: SeALS is a common method for the canonical format, but also for TT. This method is slightly different and utilize the same concept but for the TT (or MPS)-scheme. It is then called Density Matrix Normalization Group (DMRG) [9], [26], [27].

The method fixates all but one of the cores, U_p for $p \in \{1, ..., d\}/\tilde{p}$, and minimizes the solution residual with respect to $U_{\tilde{p}}$. The big difference is the network-like structure which sets some pre-conditions. For example in the DMRG algorithm, the cores' ranks must be fixed and must be guessed a priori. However, newer versions of this algorithm, like the *Two-site DMRG*, has overcome this to let the TT-ranks change adaptively [9].

3) Alternating Minimal Energy: Another algorithm similar to SeALS is the Alternating Minimal Energy algorithm (AMEn) [9]. This method combines the tensor product format of DMRG and the iterative methods of SeALS to provide an algorithm that have error convergence similar to that of two-site DMRG. For further information see [9].

III. Solver Evaluation

A goal within the project was to evaluate the different solvers and decomposition formats described in II-C and II-D. A two-dimensional problem called "Smooth 2D example" was chosen from [8] as a testing framework. The problem is not high-dimensional but easy to visually verify. The goal of this evaluation is only to see which of these solvers seem feasible to use in the context of solving (4). For the scope of these tests, two dimensions is enough.

The original problem was first set up in [28]. The goal of the problem is to reach the origin, following the dynamics

$$\begin{bmatrix} dx_1 \\ dx_2 \end{bmatrix} = \left(2 \begin{bmatrix} x_1^5 - x_1^3 - x_1 + x_1 x_2^4 \\ x_2^5 - x_2^3 - x_2 + x_2 x_1^4 \end{bmatrix} + \begin{bmatrix} x_1 & 0 \\ 0 & x_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \right) dt + \begin{bmatrix} x_1 & 0 \\ 0 & x_2 \end{bmatrix} \begin{bmatrix} dw_1 \\ dw_2 \end{bmatrix}. \quad (24)$$

The domain is set up as $\Omega = \{(x_1, x_2) | -1 \leq x_1 \leq 1, -1 \leq x_2 \leq 1\}$, with control penalty $R = 2I$ and state cost $q(x_1, x_2) = x_1^2 + x_2^2$. Also, the noise is modelled as Gaussian white noise, i.e. $\Sigma_\epsilon = I$. Finally, the terminal cost is set as $\phi(x_1, x_2) = 5$ at the boundary and $\phi(0, 0) = 0$ for the goal region. The problem is solved, and later used as reference, with the program from [8] which use canonical decomposition and the SeALS algorithm. The solution Ψ (desirability function) looks like a smooth cap, see Fig. 7.

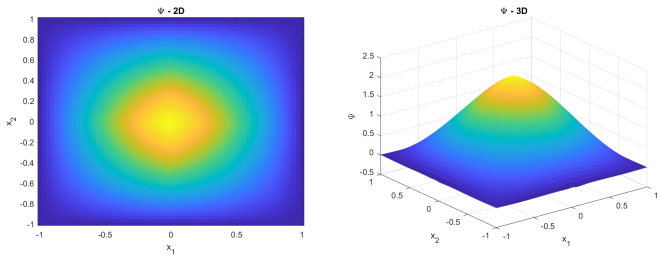

Fig. 7. Desirability function of "Smooth 2D Example" using CANDECOMP and SeALS, $n = 64$

Thereafter we utilized the TT-Toolbox [29] with pre-implemented solvers `dmrg_solve2` and `amen_solve2` to solve the problem. To ensure that the linear system components A and G were the same after transforming them to TT-representation, we let the SeALS program run its course and created the tensors as canonically decomposed up until the last step in which the system will be solved with `dmrg_solve2` and `amen_solve2`. We created complimentary MATLAB functions to make the change in decomposition and finally, using the TT-format the solvers is used. Fig. 8 and 9 show the solutions for mode size $n = 64$.

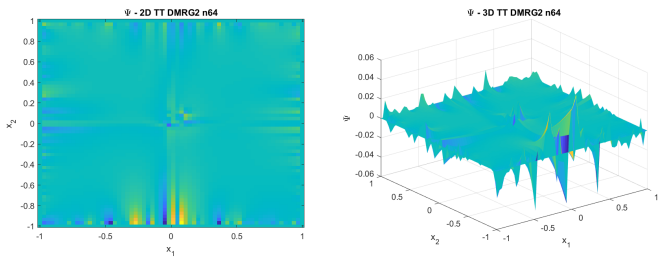

Fig. 8. Desirability function of "Smooth 2D Example" using TT and DMRG, $n = 64$

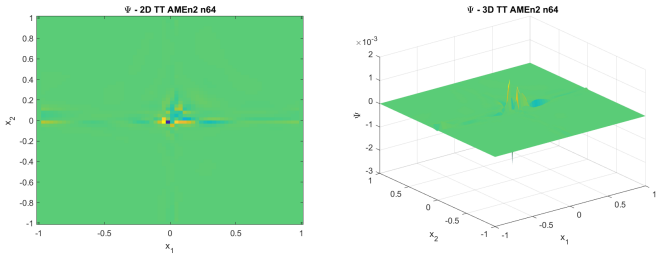

Fig. 9. Desirability function of "Smooth 2D Example" using TT and AMEn, $n = 64$

It is immediately clear that the SeALS algorithm outperforms both DMRG and AMEn with tensors in TT-format. However, according to the documentation of [29] the solvers work optimally if mode sizes are small, i.e. the tensors are quantized. Following the discovery of this, we reshaped the system components to QTT. This time, the results were much more in line with the reference solution, see Fig. 10 and 11.

The AMEn solver in Fig. 11 were still unusable, despite being in QTT representation. Not only is the shape rough (not smooth), but the amplitude is only in the order of 10^{-3}. On the other hand, DMRG seems to be identical to the reference solution.

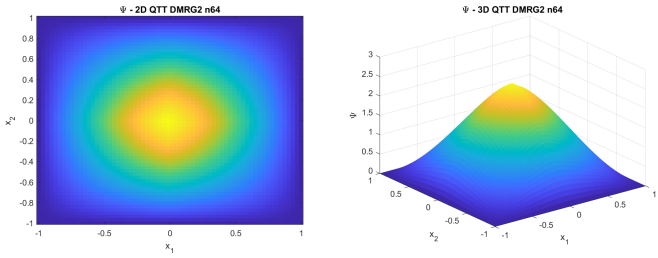

Fig. 10. Desirability function of "Smooth 2D Example" using QTT and DMRG, $n = 64$

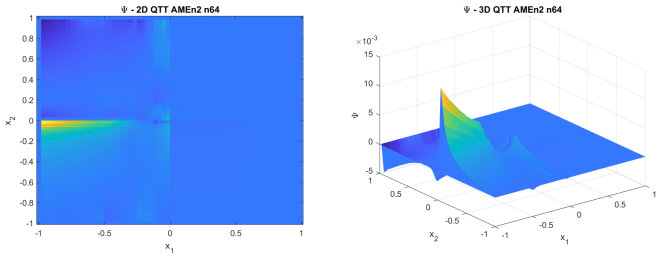

Fig. 11. Desirability function of "Smooth 2D Example" using QTT and AMEn, $n = 64$

Since the mode size has an impact on the quality of the solution, we continued the tests by increasing the mode size to $n = 128$. As can be seen in Figures 12 and 13, the DMRG algorithm show some stochastic noise but otherwise flat and AMEn seems completely flat except a small spike in the middle. However, even SeALS in Fig. 14 is not as smooth when $n = 128$. Admittedly much better than DMRG and AMEn, but still showcasing some discontinuity in the quadrant divisions.

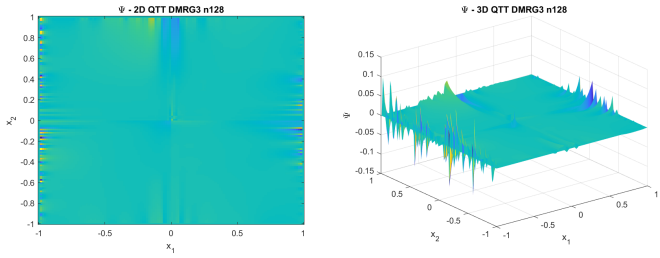

Fig. 12. Desirability function of "Smooth 2D Example" using QTT and DMRG, $n = 128$

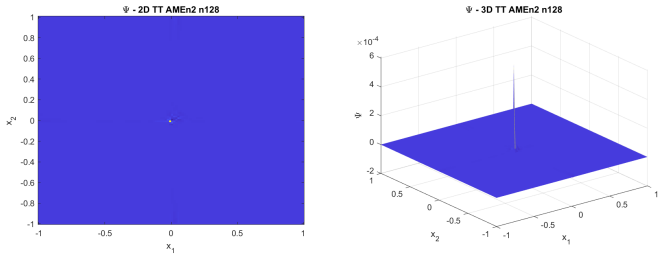

Fig. 13. Desirability function of "Smooth 2D Example" using QTT and AMEn, $n = 128$

A mode size of $n = 64$ results in relatively good resolution, so it is not necessary with any higher. Thus, for the purpose of solving the road sharing problem, either SeALS with CANDECOMP or DMRG with QTT should be used. Since a full numerical solver is already implemented in [8], using that program, with the intended SeALS algorithm, would be best.

IV. The Road Sharing Problem

In this section, we consider route planning problems for autonomous vehicles. Since it results in high dimensional structures, we use the methods for tensor decomposition and equation solving that were described in Section II-C and II-D. However, in order to apply the algorithms one must formulate the problem in a proper way. Every parameter needs to be specified in a correct way, or a solution might not exist. The different parameters must also weighted against the others to assert that the solution that is calculated is the desired one. In this section we present different route planning scenarios and describe how we formulated the dynamics and costs for them, ultimately enabling us to solve the example shown in Fig. 1 in section I.

A. Single autonomous vehicle scenario

Consider the traffic scenario in Fig. 15, where one autonomous vehicle approaches a roadblock. We want to formulate this traffic situation as an optimal control problem. One important part is to specify the dynamics of the problem. The dynamics specify how the states change depending on current state and on the control inputs. Using a higher level of abstraction, the autonomous vehicle in Fig. 15 can be seen to have two different state variables, that specify the position in the plane. The control inputs can then be set to directly impact each state, i.e. setting the velocity directly. The self

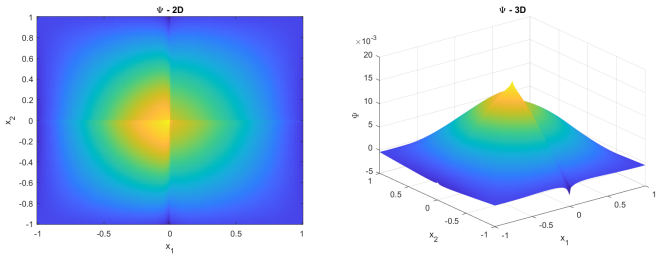

Fig. 14. Desirability function of "Smooth 2D Example" using CANDECOMP and SeALS, $n = 128$

dynamic can be set to not impact the states. This is equivalent to the vehicle having zero momentum and stopping as soon as no control is applied. In mathematical terms this can be described as

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$
$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}. \tag{25}$$

The benefit of this model is that the simplicity makes the problems easier to solve. The simplicity is not too far from reality either, since control signals for more realistic models can be deduced from results acquired with this model, by using motion planning or the like.

Fig. 15. Traffic scenario with one autonomous vehicle and one roadblock (yellow and black).

Another important aspect is specifying the cost. The cost will be a function of the states and regulates how the vehicle moves, since the solution that regulates the vehicles movement is based on minimizing the cost. Therefore, the desired end position needs to have a low cost, while there must be a high cost for driving too close to other vehicles, roadblocks or the end of the road. It is also important to use cost functions that change smoothly across different states for numerical reasons: If the cost magnitude changes too quickly it can result in numerical errors and unintended behavior.

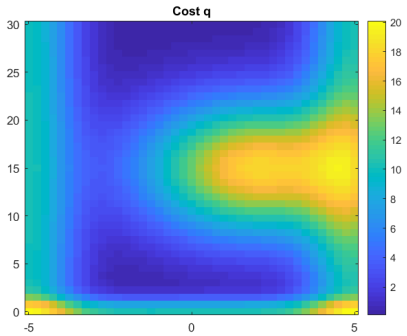

Fig. 16. The cost for different states for the autonomous vehicle as a function of x and y.

One of the measures we took to avoid too big changes in costs was that we created specialised functions for setting the costs efficiently. These functions can generate costs that are on the form of a Gaussian distribution, which changes smoothly.

For this scenario, we decided to set the state space as $x \in [-5, 5]$, $y \in [0, 30]$. Using the Gaussian cost functions on the state space we modelled the costs like in Fig. 16. There are costs for approaching every corner of the state space, except for upper limit of y, specified as one-dimensional gaussians. The upper limit of y is left without a cost since it is desired for the vehicle to approach the upper limit of y, it can be seen as proceeding forward on the road. There are also a Gaussian distribution that emulates the road block. We also set the cost for applying control as

$$R = \begin{bmatrix} 3 & 0 \\ 0 & 5 \end{bmatrix}.$$

Which means that it costs 3 to apply control to x and 5 to apply control to y. The magnitude of the costs is only relevant when compared to other costs and do not have a unit. Additionally, we set up the boundaries to have a low desirability(i.e. high terminal cost) in each dimension, but specified that there was a region with high desirability at $x \in [2, 4]$, $y \in [25, 27]$. This region corresponds to the desired position to reach, the goal region. This is not shown in the Fig 16, since it is a separate parameter choice from the cost. If you apply the toolbox in [7] that use the SeALS method on this problem, the following result is acquired.

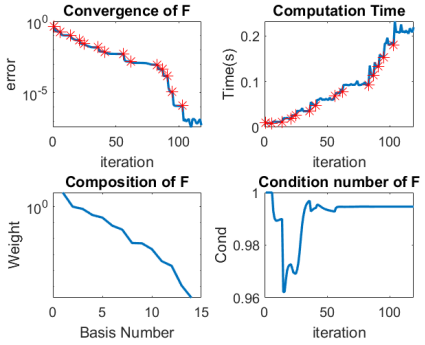

Fig. 17. Solution development of the single autonomous vehicle scenario.

The solver's progress is shown in Fig. 17. Most interesting is the upper-left corner, where the convergence of the solution is shown. The error rapidly decreases to a low value. In Fig. 18 the value function is shown. The value function is as mentioned earlier the inverse of the desirability function and indicates whether states are desirable or not. The car will try to reach states with low value (blue color). Finally, Fig. 19 shows the state trajectory for a simulation. This simulation is acquired by finding the optimal control using eq. (11) and the value function, and then creating a state trajectory based on the control. In the simulation we see that the car avoids the roadblock and reaches the desired region (the red block).

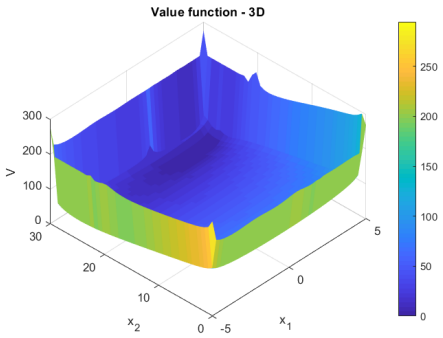

Fig. 18. Value function, inverse of the desirability function, as a function of x_1 and x_2. These variables correspond to x and y, respectively.

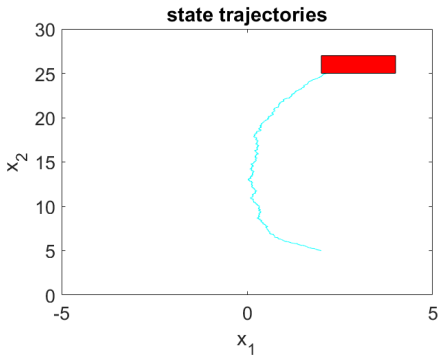

Fig. 19. A simulation of a car traversing the state space in accordance with the calculated value. The blue path is the trajectory and the red rectangle is the desired region.

For this run, the result is successful. The car reaches the other side of the roadblock without travelling into it, in a smooth matter. The solution converges to a small error fast, and the solution looks logical in terms of travelled path when compared to a real vehicle.

B. Two autonomous vehicles sharing the same lane

As mentioned in II-C, these problems do not scale easily with the number of states nor the number of grid-points. Consider a duplication of the scenario in Fig. 15, where two vehicles start in the exact same position in their respective state spaces. When imposing the exact same dynamics, costs and boundary conditions(as such, the vehicles to not take each other into account at all), we get the result shown in Fig. 20 and 21.

In Fig. 20 we can again see the solution procedure. In Fig. 21 we can see the value functions for the two cars. Important to note is that the value function is now a 4 dimensional structure, with both vehicles included. If we want to consider the value function for just one vehicle, we can fixate the other vehicles' position and then extract a two-dimensional "slice". We have chosen to only show two of these "slices", where the other car is in the middle of its state space. The value functions are obviously not smooth and as such the simulations are not going to generate any satisfactory results. Instead, the problem has to be reformulated, to avoid the numerical errors that likely caused these issues above.

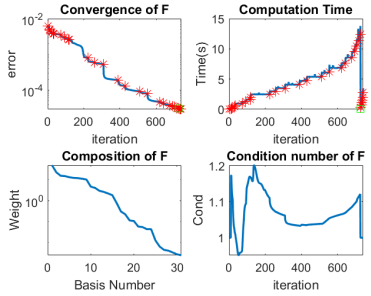

Fig. 20. Solution development of two autonomous vehicles starting in the same position. The solution requires significantly more computation time to converge at a larger error than with one car.

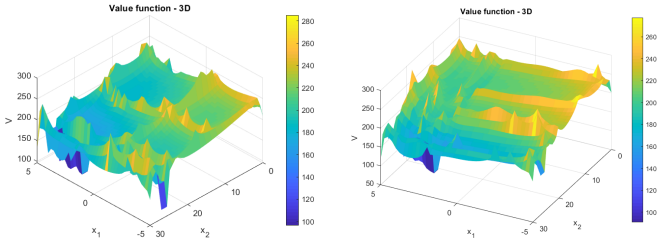

Fig. 21. Value functions for two autonomous vehicles starting in the same position seems more stochastic and is not smooth at all.

Fig. 22. Traffic scenario with two autonomous vehicles and one roadblock.

Since the single vehicle problem formulation did not translate well into two vehicles, we will now directly study two at once. The goal of this project was not only making a single autonomous vehicle avoid a roadblock, since the interaction between vehicles also were of interest. Consider the traffic scenario in Fig. 22, where two autonomous vehicles approach a roadblock on a unilateral road.

When approaching this traffic scenario, there are several more aspects to consider. With two cars, one must also construct a cost that makes them avoid running into each other. This cost function must have a greater value when the vehicles are close to each other. This can be addressed using a Gaussian function,

$$a \cdot e^{-(x_1-x_2)^2/2\cdot b^2}. \tag{26}$$

With a and b being arbitrary constants, x_1 and x_2 being the x-coordinates for each of the vehicles. There must of course be such a term for both the x and y coordinate. However, the toolbox that were used for these test required cost terms to be on separated form. Namely, it must be possible to accurately approximate the cost function as a summation of multiplied one-variable functions [30]. This means that this term is problematic. Therefore, the expression was first rewritten to

$$a \cdot e^{-x_1^2/2\cdot b^2} \cdot e^{-x_2^2/2\cdot b^2} \cdot e^{(x_1\cdot x_2)/b^2}. \tag{27}$$

In this case, only the third factor is an issue. This factor was therefore rewritten using a series expansion or grade 2,

yielding the expression

$$a \cdot e^{-x_1^2/2 \cdot b^2} \cdot e^{-x_2^2/2 \cdot b^2} \cdot (1 + \frac{x_1 \cdot x_2}{b^2} + \frac{x_2^2 \cdot x_2^2}{2 \cdot b^2}). \quad (28)$$

From this point a separation into a summation of multiplied one-variable functions is possible.

Another issue that arised with two vehicles is that whenever one of the vehicles exit the simulation, the simulation stops. This means that there can be situations were only one vehicle is inclined to move forward. The other vehicle needs to apply control, which costs, to move and it might result in a lower cost to simply wait. In order to prevent this, smart boundary conditions were created. In section IV-A, the boundaries were simply set to have desirability 1 for the side of the state space where the vehicle were supposed to go. This was not sufficient in the scenario with two vehicles. If the first vehicle has state variables x_1 and y_1 is going upwards in state space while the second vehicle has x_2 and y_2 is going downwards, smart boundary conditions can be created,

$$1 - (1 - \frac{y_1}{d})$$

$$1 - \frac{y_2}{d}.$$

For the lower and upper limit of the state space respectively, with d being the length of the state space in the y direction. This function makes it so that the desirability of the lower limit of y_2 only approaches 1 once the first car approaches $y_1 = 30$, and vice versa. This boundary formulation means that the cars will wait for each other to exit the simulation, thus preventing situations were only one car apply control.

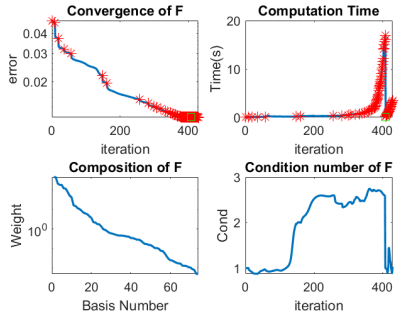

Fig. 23. Solution development using a larger state space and with vehicle avoidance functions.

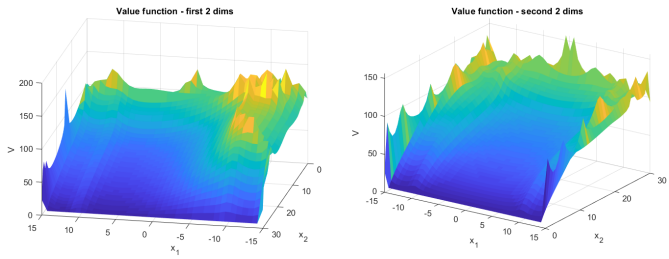

Fig. 24. Value function for two autonomous vehicles starting opposite to each other. The function are smooth and show only minor disturbances near the boundaries.

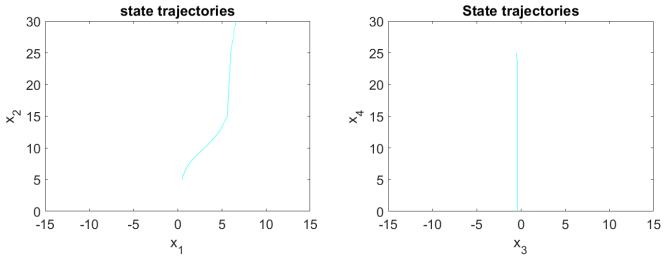

Fig. 25. State trajectories for two autonomous vehicles starting opposite to each other. One car avoids the other so that they do not collide.

Another numerical issue is the spikes you can see in Fig. 21. These can have different causes, but one cause can be that the magnitude difference in costs between two nearby points in the state space is too high, which can cause that unreasonable amounts of control is applied. This can be an issue if the costs within the state space have a very different value than the boundary conditions. In order to prevent this, the state space was made bigger, while the Gaussian distributed costs remained the same, which contributes to a smoother value function.

Using these new strategies and the dynamics in Fig. 15 for each car, we made a run without a roadblock and got the result shown in Fig. 23, 24 and 25. This result is successful since the cars avoid each other in the simulation and since the value function is much more smooth.

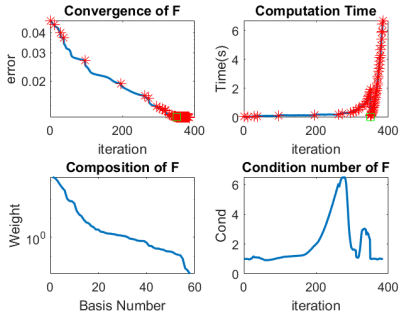

Fig. 26. Solution development using a larger state space, with vehicle avoidance functions and a roadblock.

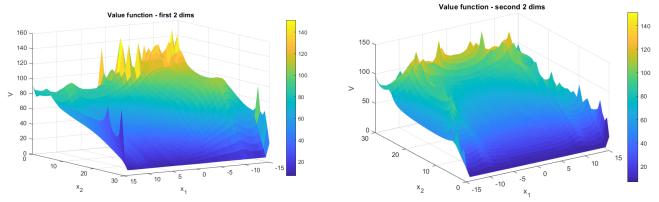

Fig. 27. Value function for two autonomous vehicles and a roadblock. The function seem smooth for both cars' state spaces.

The final challenge now is to add a roadblock to the two car situation. After several tries, the best run of the program generated was found by placing the roadblock in $(x, y) = (2.5, 15)$. From this, the results in Fig. 26, 27 and 28 were generated. It is clear from the upper left part of Fig. 26 that the solution has converged. One can also note that the solution is mainly smooth from Fig. 27. As we can see in Fig. 28, the

autonomous vehicles avoid each other and the roadblock. Due to the problem formulation, one of the vehicles pass on the narrow side of the roadblock, which is not intended. This result does however show that the vehicles can be manipulated to move as intended and that a smooth solution could be acquired.

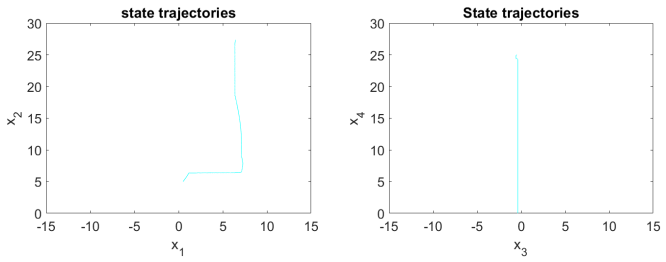

Fig. 28. State trajectories for two autonomous vehicles and a roadblock. One car moves to the side to avoid collision with the second car, and past the roadblock so that it can drive past it without crashing. Unexpectedly the car drives to the far-side of the road, squeezing in between of roadblock and roadside.

V. Conclusion

A. Summary and conclusion

In this project we have solved route planning problems for autonomous vehicles using optimal control. We compared a couple of different tensor decomposition methods briefly to find which one suited our needs the best. The major part of the project was then to formulate the costs and dynamics for a traffic scenario where two autonomous vehicles encounter a roadblock.

The results in this project concluded that the algorithms SeALS with canonical decomposition or DMRG with quantized tensor train decomposition performed equally well for $n = 64$. However, we concluded in III that the SeALS algorithm would be easier to use since we already had access to an existing solver using this algorithm.

The results also showed that formulating the costs for the traffic scenario was a complicated task. In order to achieve a desirable solution one must assert that the cost changes smoothly over the statespace and that it trivial solutions such as ending the simulation early are impossible. This could be done by using a large statespace and Gaussian cost functions, in conjunction with boundary conditions that required all vehicles to approach the end of the state space at the same time.

B. Discussion of the results

The evaluation of the solvers within this project gave a first glance of the difference in performance between the different variants, but there is much more that could be analyzed. Our quick test does not investigate how the solvers scale with dimensions, but rather with number of grid-points. We did not investigate error convergence or the conditioning of the operator in our comparison. Due to this, our evaluation was done as a way to select the solver for our specific needs and the result may therefore not be generalized to other applications.

Using optimal control for route planning problems like these have both advantages and disadvantages. One benefit with the approach is that for a given problem, the solution is known to be optimal. This means that a given solution is guaranteed to be the best in terms of minimizing the time or fuel spent, for all vehicles in the calculation. Since the calculation is done for all vehicles, in can be done by a remote device, which can potentially have a better computation power than the vehicles by themselves. The calculation also only needs to be done once, and can thereafter be used by all vehicles to obtain optimal controls. Calculating the value function is the time consuming part of the program, so once it is done one can simulate multiple situations with low computation times.

The problems with the approach is that the best problem formulation is difficult to find. One must carefully avoid numerical errors, which could have devastating effects on any real implementation. This means that long development times are needed to produce correct formulations, which is sub-optimal.

Another downside is that while the tensor decomposition makes the solvers more manageable, there is still significant computation time for relatively small problems. In this project we studied relatively low-dimensional cases, but in a real application it is of interest with many more autonomous vehicles in the calculation.

C. Future work

There are several interesting aspects of the project that could be expand on. As mentioned earlier, the different solvers were not analyzed thoroughly and this could be a topic for future work. Another interesting aspect would be to create an algorithm that can expand to multiple autonomous vehicles more easily, that can add costs automatically. This would make the real world implementation more feasible. One could also study another type of autonomous vehicle, such as drones. This would add another dimension to the positioning and could generate interesting results.

Acknowledgment

The authors would like to thank supervisor Elis Stefanson for his guidance, help and for the many interesting discussions within the subject of optimal control.

References

[1] T. V. Jan-Erik Berggren. (2019, May) 3 700 dör på vägarna varje dag – här är farligaste länderna. [Online]. Available: https://teknikensvarld.se/3-700-dor-pa-vagarna-varje-dag-har-ar-farligaste-landerna/

[2] J. Rios-Torres and A. Malikopoulos, "A survey on the coordination of connected and automated vehicles at intersections and merging at highway on-ramps," *IEEE Transactions on Intelligent Transportation Systems*, vol. PP, pp. 1–12, 09 2016.

[3] P. Varaiya, "Smart cars on smart roads: problems of control," *IEEE Transactions on Automatic Control*, vol. 38, no. 2, pp. 195–207, 1993.

[4] J. F. Fisac, E. Bronstein, E. Stefansson, D. Sadigh, S. S. Sastry, and A. D. Dragan, "Hierarchical game-theoretic planning for autonomous vehicles," 2018.

[5] E. Stefansson, J. F. Fisac, D. Sadigh, S. S. Sastry, and K. H. Johansson, "Human-robot interaction for truck platooning using hierarchical dynamic games," in *2019 18th European Control Conference (ECC)*, 2019, pp. 3165–3172.

[6] U. Jönsson, C. Trygger, and P. Ögren, *Optimal Control*. Stockholm: Optimization and Systems Theory, Royal Institute of Technology, 2010.

[7] E. Stefansson and Y. P. Leong, "Sequential alternating least squares for solving high dimensional linear hamilton-jacobi-bellman equation," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 3757–3764.

[8] E. Stefansson, Y. P. Leong, and J. C. Doyle, "Numerical solver for the linear hamilton-jacobi-bellman equation," Unpublished, Pasadena, 2015, research project at Caltech, Pasadena.

[9] S. V. Dolgov and D. V. Savostyanov, "Alternating minimal energy methods for linear systems in higher dimensions," *SIAM Journal on Scientific Computing*, vol. 36, no. 5, pp. 2248–2271, 2014.

[10] S. Dolgov, D. Kalise, and K. Kunisch, "Tensor decompositions for high-dimensional hamilton-jacobi-bellman equations," 2019. [Online]. Available: https://arxiv.org/abs/1908.01533

[11] M. Zolfpour-Arokhlo, A. Selamat, S. Z. M. Hashim], and H. Afkhami, "Modeling of route planning system based on q value-based dynamic programming with multi-agent reinforcement learning algorithms," *Engineering Applications of Artificial Intelligence*, vol. 29, pp. 163 – 177, 2014.

[12] W. H. Fleming and H. Soner, *Controlled Markov Processes and Viscosity Solutions*, second edition ed., ser. Stochastic Modelling and Applied Probability. New York, NY: Springer New York, 2006, vol. 25.

[13] B. W. Bader, T. G. Kolda *et al.* (2019, Dec., Jun.) Matlab tensor toolbox version 3.1. [Online]. Available: https://www.tensortoolbox.org

[14] B. W. Bader and T. G. Kolda, "Efficient MATLAB computations with sparse and factored tensors," *SIAM Journal on Scientific Computing*, vol. 30, no. 1, pp. 205–231, Dec. 2007.

[15] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.

[16] I. V. Oseledets and E. E. Tyrtyshnikov, "Breaking the curse of dimensionality, or how to use svd in many dimensions," *SIAM Journal on Scientific Computing*, vol. 31, no. 5, pp. 3744–3759, 2009.

[17] I. Oseledets and E. Tyrtyshnikov, "Tt-cross approximation for multidimensional arrays," *Linear Algebra and Its Applications*, vol. 432, no. 1, pp. 70–88, 2010.

[18] I. Oseledets, D. Savostyanov, and E. Tyrtyshnikov, "Linear algebra for tensor problems," *Computing*, vol. 85, no. 3, pp. 169–188, 2009.

[19] S. Dolgov and B. Khoromskij, "Two-level qtt-tucker format for optimized tensor calculus," *SIAM Journal on Matrix Analysis and Applications*, vol. 34, no. 2, pp. 593–623, 2013.

[20] C. F. V. Loan. (2008, Jan) Tensor network computations in quantum chemistry. [Online]. Available: http://www.cs.cornell.edu/cv/OtherPdf/ZeuthenCVL.pdf

[21] R. Hübener, V. Nebendahl, and W. Dür, "Concatenated tensor network states," *New Journal of Physics*, vol. 12, no. 2, p. 025004, 2010.

[22] I. V. Oseledets, "Approximation of $2^d \times 2^d$ matrices using tensor decomposition," *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 4, pp. 2130–2145, 2010.

[23] I. Oseledets, "Constructive representation of functions in low-rank tensor formats," *Constructive Approximation*, vol. 37, no. 1, pp. 1–18, 2013.

[24] B. Khoromskij and I. Oseledets, "Quantics-tt collocation approximation of parameter-dependent and stochastic elliptic pdes," *Computational Methods in Applied Mathematics*, vol. 10, no. 4, 2010.

[25] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[26] B. N. Khoromskij and I. V. Oseledets, "DMRG+QTT approach to computation of the ground state for the molecular schrödinger operator," 2010, MPI MIS Leipzig.

[27] S. White, "Density-matrix algorithms for quantum renormalization-groups," *Physical Review B*, vol. 48, no. 14, pp. 10 345–10 356, 1993.

[28] Y. P. Leong, M. B. Horowitz, and J. W. Burdick, "Linearly solvable stochastic control lyapunov functions," *SIAM Journal on Control and Optimization*, vol. 54, no. 6, pp. 3106–3125, 2016.

[29] I. V. Oseledets. (2020, Apr.) TT-toolbox version 2.2.2. [Online]. Available: https://github.com/oseledets/TT-Toolbox

[30] E. Stefansson and Y. Leong, "Sequential alternating least squares (SeALS) in MATLAB," Unpublished, Pasadena, 2015, research project at Caltech, Pasadena.