Tensor-Networked Square-Root Kalman Filter for Online Video Completion

P. van Klaveren

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Tensor-Networked Square-Root Kalman Filter for Online Video Completion

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

P. van Klaveren

June 25, 2021

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

Online video completion aims to complete corrupted frames of a video in an online fashion. Consider a surveillance camera that suddenly outputs corrupted data, where up to 95% of the pixels per frame are corrupted. Real time video completion and correction is often desirable in such scenarios. Therefore, this thesis improves the Tensor-Networked Kalman Filter (TNKF) as presented in [12] by developing the Tensor-Networked Square-Root Kalman Filter (TNSRKF). The TNSRKF is a Square-Root Kalman Filter (SRKF) realized in a Tensor Networks (TN) structure. The square-root Kalman filter is an inherently stable algorithm, and indirectly allows for more information retention throughout the algorithm. Thereby, the filter aims to improve the performance of the TNKF. Furthermore, implementing the filter in TN results in an intuitive implementation of the filter while allowing for larger video dimensions than the widely used matrix format. This thesis concludes that the TNSRKF is too computationally burdensome to complete a video in an online fashion due to the implementation of the Modified Gram-Schmidt (MGS) algorithm in Tensor-Train (TT)-format. In addition, results demonstrate that a low-rank orthogonality matrix is not realizable, making it impractical to update the state covariance matrix via any method that uses an orthonormal matrix.

Table of Contents

List of Figures

List of Tables

Preface

This report is part of my Master of Science graduation thesis. This thesis continues the work of S. de Rooij's thesis [12] by expanding the research in a new direction. The code used for this thesis can found on Gitlab: https://gitlab.tudelft.nl/msc-projects/tn-square-root-kalman.

Furthmore, I want to express my gratitude to my supervisor dr. ir. Kim Batselier for his guidance and assistance during the last year. I have learned a lot from his academic insight and his approach towards challenges we faced. Secondly, I want to thank my parents, family, and friends for their continual support during my academic career.

Delft, University of Technology
June 25, 2021

P. van Klaveren

Chapter 1

Introduction

Online matrix completion has quickly become a hot topic, boasting applications in different industries and ranging over various fields of study. Online matrix completion techniques aim to fill in the missing entries of a matrix in an online way, implying that new measurements or data are immediately used to update the matrix. Examples of implementations are: the recommendation algorithm of Amazon [24], the movie recommendation system of Netflix [21] and video completion of a corrupted video [12].

The application of this thesis is in corrupted video completion. Specifically, the videos considered in this thesis are surveillance footage. Figure 1-1 is an example of such corrupted footage. As can be seen below, the video becomes corrupted at a certain moment in time, where frames of a Town Center's security video are shown [1]. The surveillance footage preceding the corrupted frame shown in Figure 1-1 is uncorrupted. The degree of corruption considered in this thesis is approximately 95% of each frame's pixels, and can be caused by a technical error or an obstruction of the camera's view. Moreover, direct access to the uncorrupted pixels and stationary camera positioning is assumed. The goal is to make an algorithm that performs online video completion on corrupted footage by recovering the corrupted pixels.

Figure 1-1: Example of corrupted frame in surveillance footage [12, 1].

A recently developed technique for performing online video completion is via the Tensor-Networked Kalman Filter (TNKF) [12]. The approach taken by TNKF aims to model any video as a state-space model wherein the consecutive frames of the video are modeled as a state vector of the state-space system. Moreover, based on this underlying state-space system the recursive Kalman filter is utilized to estimate and predict the missing entries of the state vector. Hence, the video is completed via matrix completion. Due to the large matrix dimensions the operations of the Kalman filter cannot be executed on a regular computer. Take for example a Full HD video, consisting of 1080 by 1920 pixels, which has a storage complexity of approximately $\mathcal{O}(10^6)$. The covariance matrix has a complexity of $\mathcal{O}(10^{12})$, requiring approximately 32 thousand GB of RAM. Hence, the operations are performed in a Tensor Networks (TN). Via a TN the storage complexity of the matrices and time complexity of the algorithm decreases, making it possible to execute the algorithm on a regular computer.

One of the assumptions of the Kalman filter is that the covariance matrix is (semi-)positive definite. This is to ensure a converging state. However, due to the nature of the Kalman filter itself it can lose this condition during the optimization. Hence, a major difficulty is that the covariance matrix loses its positive definiteness property, resulting in diverging states. In this thesis a solution for this problem is proposed: Tensor-Networked Square-Root Kalman Filter (TNSRKF). This technique uses a square-root factorization of the covariance matrix and updates this matrix instead of the covariance matrix itself.

The filter guarantees a numerically stable algorithm and converging state updates. Hence, the covariance matrix is positive definite by definition. The square-root Kalman filter is also implemented in TN and therefore is called Tensor-Networked Square-Root Kalman Filter (TNSRKF). Thus, the TNSRKF is an algorithm made to perform online matrix completion where the frame of the video is the matrix.

The research in this thesis addresses the following research question.

- *How can TNSRKF be implemented to recover a corrupted video?*

In addition, multiple other research questions are formulated, each deepening the research in a different way:

- *How can the square-root Kalman filter be implemented in Tensor-Train (TT)-format?*
- *How can the Modified Gram-Schmidt (MGS) orthogonalization procedure be implemented in TT-format and how can orthogonality be preserved?*
- *Is it possible to represent an orthogonal matrix as a low-rank Tensor-Train matrix (TTm)?*

An understanding of the problem at hand and insight into the proposed solution can lead to a deeper understanding of how the TNSRKF can be implemented. The hypothesis is that the TNSRKF can be used to estimate and complete the sparse matrix. However, the implementation of the MGS in TT-format can become very time consuming due to orthogonalization of column vectors.

This thesis is outlined as follows. First, online matrix completion or existing video completion techniques will be discussed. They will be referred to as the state of the art video completion

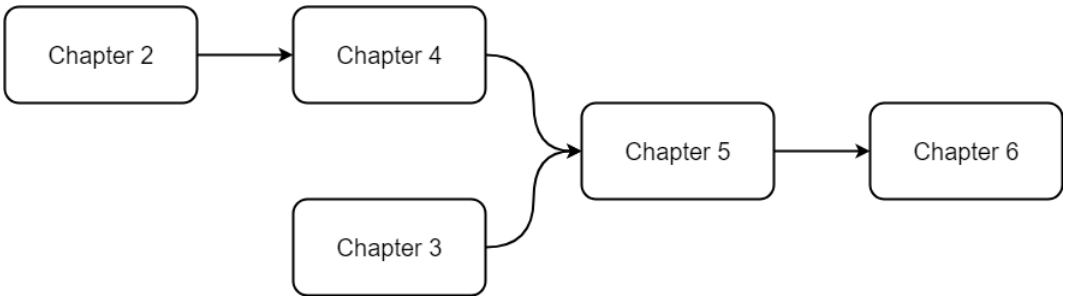

Figure 1-2: This diagram visualizes the structure of the report.

techniques and are presented in Chapter 2, which focuses on introducing the use of the Kalman filter in TN-format for online video completion. Chapter 3 begins by explaining what a tensor is and dives deeper into the notation of and operations with TTs and TTms. Chapter 4 begins with a introduction of a state-space representation, followed by a derivation and definition of the Kalman filter, and ends with a general implementation of a Square-Root Kalman Filter (SRKF). Chapter 5 combines insights from the previous two chapters. Its focal point is the implementation of a SRKF in TN for online video completion. Finally, Chapter 6 provides a reflection on the work and answers the research questions. In addition, it provides some thoughts concerning the improvement of the TNSRKF and future work. The structure can be summarized by means of Figure 1-2.

1-1 Notation

Throughout this thesis the following notation is used. Table 1-1 displays example notations using the generic placeholder A.

Notation	Definition
a	scalar
$\mathbf{a}$	vector
$\mathbf{a}(i) = a_i$	i^{th} entry of the vector $\mathbf{a}$
$\mathbf{A}$	Matrix
$\mathbf{A}(i,j) = a_{i,j}$	entry (i,j) of the matrix $\mathbf{A}$
$\mathcal{A}$	Tensor
$\mathcal{A}(i_1, i_2, \ldots, i_d) = a_{i_1,i_2,\ldots,i_d}$	entry $(i_1, i_2, \ldots, i_d)$ of the tensor $\mathcal{A}$
$\mathcal{A}^{(i)}$	i^{th} TT-core of a TT(m)
$\mathbf{A}^{-1}$	matrix inverse
$\mathbf{A}^T$	matrix transpose
$\|\cdot\|_F$	Frobenius norm
$\otimes$	Kronecker product

Table 1-1: Notation used throughout this thesis.

Chapter 2

State of the Art for Online Video Completion

This chapter introduces the notion of matrix completion in a general form in Section 2-1 by presenting the algorithm in [38] named Adaptive Online Low-Rank Matrix Completion. This section is not closely related to the problem at hand but sketches a general idea of matrix completion. The next section continues by introducing the Tensor-Networked Kalman Filter (TNKF) algorithm, which is specifically meant for streaming video completion but can be generalized to other applications. This chapter aims to identify the TNKF on which this thesis is built.

2-1 Adaptive Online Low-Rank Matrix Completion

Unless stated otherwise, this section is based on [38] .
The aim of adaptive low-rank matrix completion is to reconstruct an incomplete and noise corrupted sequence of measurement matrices online. The general idea of this method is based on stochastic gradient descent of a Least-Mean Squares (LMS) cost function. In their paper, Tripathi et al. describe the problem as follows. A low rank measurement matrix $\mathbf{M} \in \mathbb{R}^{I \times J}$ is obtained and observed over a subset Ω. A matrix $\mathbf{J}$ with its entries equal to 1 if $(i, j) \in \Omega$ is also obtained. The state matrix $\mathbf{X}$ can then be imputed using the rank minimization problem defined in (2-1)

$$\begin{aligned} &\min_{\mathbf{X}} \operatorname{rank}(\mathbf{X}) \\ &\text{s. t.} \quad \mathbf{M}(i,j) = \mathbf{X}(i,j) \quad (i,j) \in \Omega. \end{aligned} \tag{2-1}$$

However, as stated in the paper, this problem is non-convex and NP-hard to solve. Thus, the rank function is replaced by the nuclear norm function of $\mathbf{X}$ (2-2), which still gives exact results [35]. Since the application of the methods are in noise corrupted matrices, the

constraint of the minimization is $\|\mathbf{J} * (\mathbf{M} - \mathbf{X})\|_F^2 \leq \epsilon$

$$\begin{aligned} &\min_{\mathbf{X}} \ \|\mathbf{X}\|_* \\ &\text{s. t.} \quad \|\mathbf{J}(i,j) * (\mathbf{M}(i,j) - \mathbf{X}(i,j))\|_F^2 \leq \epsilon. \end{aligned} \tag{2-2}$$

The Least-Mean Squares algorithm makes use of stochastic gradient descent to compute the next state of the system according to a defined loss function $\ell_k(\hat{\mathbf{X}}_k)$. The next state $\hat{\mathbf{X}}_{k+1}$ can be estimated via minimization of a loss function and regularizer term as seen in (2-3), where μ is the learning rate. Further steps are taking the first order Taylor approximation of (2-3), differentiating with respect to $\mathbf{X}$ and setting this to zero to find the optimum. This yields the update equation for $\hat{\mathbf{X}}_{k+1}$ as presented in (2-4)

$$\hat{\mathbf{X}}_{k+1} = \underset{\mathbf{X}}{\operatorname{argmin}} \ \mu \sum_{\kappa=1}^{k} \ell_k(\mathbf{X}) + \frac{1}{2} \|\mathbf{X}\|_F^2 , \tag{2-3}$$

$$\hat{\mathbf{X}}_{k+1} = \hat{\mathbf{X}}_k - \mu \nabla \ell_k(\hat{\mathbf{X}}_k). \tag{2-4}$$

2-2 Tensor-Networked Kalman Filter

Online video completion can also done via the TNKF as presented by [22] and further developed in [12]. This method focuses on using a recursive Kalman filter to estimate corrupted pixels in a frame.

2-2-1 Discrete State-Space Model

For the TNKF to work the video has to be transformed into a discrete-time state-space system, where each frame can be seen as the current state of the state-space system. To model a frame into a state vector, each frame is transformed into a 8-bit integer valued matrix and transformed into a vector, $\mathbf{x}[k]$. Figure 2-1 illustrates this by considering a randomly chosen frame of the Town Center video. The state vector is created by reshaping the matrix into a vector by stacking its columns. Furthermore, the assumption is made that the difference between two successive frames is minimal. Hence, the state transition matrix is assumed to be equivalent to the identity matrix and the difference between each frame is caused by the dynamic driving noise or process noise. Moreover, uncorrupted access to the measurements is assumed. Thus, there is no measurement noise present in the state-space system. This results in the following state-space system:

$$\begin{aligned} \mathbf{x}[k+1] &= \mathbf{x}[k] + \mathbf{w}[k] \\ \mathbf{y}[k] &= C\mathbf{x}[k]. \end{aligned} \tag{2-5}$$

The state vector can be remodeled by subtracting the background from the state. This background can be estimated prior to the video corruption and is assumed to equal the mean of the prior frames. One benefit of subtracting the background is that the vector-rank is lower, which will decrease computational load when working with Tensor-Train (TT) because a lower rank approximation can be used.

Figure 2-1: This figure shows the transition from a frame of a video to an 8-bit integer valued matrix [1, 12].

2-2-2 Recursive Kalman Filter

The recursive Kalman filter that is applied for the video completion can be subdivided into two recursive updates, the time propagation update and the measurement update as presented in (2-6) and (2-7),

$$\begin{aligned} \hat{\mathbf{x}}[k]^- &= \hat{\mathbf{x}}[k-1]^+ \\ \mathbf{P}[k]^- &= \mathbf{P}[k-1]^+ + \mathbf{Q}[k], \end{aligned} \tag{2-6}$$

$$\begin{aligned} \mathbf{v}[k] &= \mathbf{y}[k] - \mathbf{C}\hat{x}[k] \\ \mathbf{S}[k] &= \mathbf{C}\mathbf{P}[k]^-\mathbf{C}^T \\ \mathbf{K}[k] &= \mathbf{P}[k]^-\mathbf{C}\mathbf{S}[k]^{-1} \\ \hat{\mathbf{x}}[k]^+ &= \hat{\mathbf{x}}[k]^- + \mathbf{K}[k]\mathbf{v}[k] \\ \mathbf{P}[k]^+ &= \mathbf{P}[k]^- - \mathbf{K}[k]\mathbf{S}[k]\mathbf{K}[k]^T. \end{aligned} \tag{2-7}$$

The time update gives a one-step ahead prediction state estimate and the measurement update gives a filtered state estimate [41]. Via these two equations the state $\mathbf{x}$ can be estimated and completed. Besides its format, the TNKF is in essence the same as the Kalman filter, since the matrices that are permuted into TT or Tensor-Train matrix (TTm), matrix-vector, and matrix-matrix products are done in TT-format. The Tensor Networks (TN)-format is introduced to compute the recursive Kalman steps in an efficient way and are used for all matrices such as the state vector, state-space matrices and covariance matrices.
As will be discussed in Chapter 3, matrix-vector and matrix-matrix product can easily be performed in TT-format. However, matrix inversions are more difficult. Hence, the implementation of the TNKF measurement update steps are performed via the so called Partitioned Update Kalman Filter (PUKF) to avoid matrix inversions:

$$\begin{aligned} &\text{For } j = 1 : J \\ &\quad v_j[k] = y_j[k] - \hat{x}_{j-1}(c_j)[k] \\ &\quad s_j[k] = \mathbf{P}[k^-](c_j, c_j) \\ &\quad \mathbf{k}_j[k] = \mathbf{P}[k^-](:, c_j)s_j[k]^{-1} \\ &\quad \hat{\mathbf{x}}_j[k^+] = \hat{\mathbf{x}}_{j-1}[k^-] + \mathbf{k}_j[k]v_j[k] \\ &\quad \mathbf{P}_j[k^+] = \mathbf{P}_{j-1}[k^-] - \mathbf{k}_j[k]\mathbf{s}_j[k]\mathbf{k}_j[k]^T. \end{aligned} \tag{2-8}$$

2-2-3 Initialization of the Kalman Filter

The Kalman filter needs initial information about the state, state covariance matrix and the process noise covariance matrix. The initial state is chosen as the last uncorrupted frame and is denoted by $\mathbf{x}[0]$. According to the assumption of the Kalman filter all the states are Gaussian distributions and can be denoted by (2-9)

$$\mathbf{x}[0] \simeq \mathcal{N}(\bar{\mathbf{x}}[0], \mathbf{P}[0]), \tag{2-9}$$

where $\bar{\mathbf{x}}[0]$ is the mean $\mathbf{x}[0]$ and $\mathbf{P}[0]$ is the initial state covariance matrix equaling $\mathbf{P}[0] = E\left[(\mathbf{x}[0] - \bar{\mathbf{x}}[0])(\mathbf{x}[0] - \bar{\mathbf{x}}[0])^T\right]$. Both these variables can be determined and updated in the background by using the final uncorrupted frame. This can lead to a very small covariance matrix. The state covariance matrix cannot equal zero since it must be symmetric positive definite. Therefore, another way of initializing the state covariance matrix is to define it by $\mathbf{P}[0] = \sigma_P \mathbf{I}$, where σ_P is equal to the variance of each pixels. This last method also ensures positive definiteness of the initial state covariance matrix.
The process noise covariance matrix is a prominent factor in the Kalman filter. Because the state-transition matrix is the identity matrix, all new information for the next state is captured in the process noise covariance matrix. Therefore, an accurate estimation of this matrix is important. The process noise covariance matrix can be estimated by (2-10)

$$\mathbf{Q}[k] = E\left[\mathbf{w}[k]\mathbf{w}[k]^T\right] = E\left[(\mathbf{x}[k+1] - \mathbf{x}[k])(\mathbf{x}[k+1] - \mathbf{x}[k])^T\right]. \tag{2-10}$$

2-3 Computational Load Comparison

In this section a comparison will be carried out. This comparison will be done over a specific adaptive matrix completion technique called Adaptive Singular Value Thresholding (ASVT) and TNKF. The main focus of this section is to identify the computational load of each algorithms and thereby identify the most computationally expensive calculation.
To begin, the ASVT algorithm is analyzed. The assumption is made that only a fraction α of the pixels are observed. This directly implies that $1 - \alpha$ of $\mathbf{J}$'s entries are equal to zero, which will reduce the amount of computations. The computation of $\mathbf{Y}_{k+1}$ has a complexity of $\mathcal{O}(2\alpha IJ)$ due to the Hadamard matrix multiplication and the matrix that is multiplied by a constant. The calculation of $\hat{\mathbf{X}}_{k+1}$ consists of a SVD of $\mathbf{Y}_{k+1}$ and subtractions. The SVD can be generated via various algorithms. Here, the complexity is obtained from the truncated-SVD. Li et al. determined that the complexity of the truncated-SVD equals $\mathcal{O}(2IJ^2 + J^3 + J + IJ)$ [23]. Hence, the total complexity of the ASVT algorithm is $\mathcal{O}(2IJ^2 + J^3)$.
The complexity of the Tensor-Networked Kalman Filter algorithm can be computed once the amount of TT-cores, the partitioning of the row and column of the matrix and the TT-ranks are known. Note that I in TNKF equals I multiplied with J and is denoted by (IJ), where I and J are the matrix dimensions of the frame. From this point onward, the remaining part of this chapter will use (IJ) instead of I to avoid confusion. The TT of the state vector has d cores, the partition of the state vector is $[(IJ)_1, (IJ)_2, \ldots, (IJ)_d]$, and $(IJ)_{\max}$ equals $\max\{(IJ)_1, (IJ)_2, \ldots, (IJ)_d\}$. Furthermore, the TT-ranks are $[R_0, R_1, \ldots, R_d]$ and $R_x^{\max}$ is the maximum TT-rank. The TTm of the covariance matrix can be denoted in the same way. The only difference is that the row and column indices of the matrix are defined by

Algorithm	Complexity
ASVT	$\mathcal{O}(2IJ^2 + J^3)$
TNKF	$\mathcal{O}(\alpha IJd(IJ)^2_{\max}R^4)$

Table 2-1: Complexity for one update of the ASVT and TNKF algorithm. I is the row dimension and J is the column dimension of the frame of the video. Furthermore, R is the maximum rank of the TT decomposition in TNKF, d equals the amount of TT-cores and α is the fraction of observed pixels. Furthermore, $(IJ)_{\max}$ equals $\max\{(IJ)_1, (IJ)_2, \ldots, (IJ)_d\}$.

$[(IJ)_1, (IJ)_2, \ldots, (IJ)_d]$. For simplicity, it is assumed that all TTs have the same amount of cores, partitions and ranks, also all the TTms have the same amount of cores, partitions and ranks. The most expensive calculation is the calculation of $\mathbf{P}_j$ because an outer product of two vectors is required. These vectors are composed of a row of the covariance matrix. The other computations require only additions in TT-format or scalar operations. Hence, the most complex computation has a complexity equal to $\mathcal{O}(d(IJ)^2_{\max}R^4)$ [12]. Since this operation is called L times, the total complexity of this function equals $\mathcal{O}(Ld(IJ)^2_{\max}R^4)$, where L equals αIJ. Thus, the complexity of TNKF equals $\mathcal{O}(\alpha IJd(IJ)^2_{\max}R^4)$.
An analysis of the complexities shows that the complexity of TNKF can only match the complexity of ASVT for $R < 1$. Since this is not a realistic TT-rank value, TNKF will most likely be more complex than ASVT considering a full HD resolution.

Chapter 3

Tensor Networks

As stated in the introduction of this thesis, this chapter covers the fundamental methods of tensor networks, which are important to be able to understand and improve the existing technique: Tensor-Networked Kalman Filter (TNKF). Therefore, the focus of this chapter is on Tensor Networks (TN). TN are efficient in data storage and are intuitive in their application for matrix-matrix and matrix-vector computations. First, the basic definition of a tensor is described and graphically explained, after which basic tensor operations are covered. The chapter continues with tensor decompositions in which mainly the Tensor-Train (TT) decomposition will be discussed. Furthermore, TT and Tensor-Train matrix (TTm) operations and functions are explained.

3-1 Tensor

Unless stated otherwise, this section is based on [20].
A tensor can be described as a type of array, where an array has d indices, a vector has 1 index, a matrix has 2 indices and a tensor can have $d \geq 3$ indices. Hence, a tensor is also known as a d-way or d^{th} order tensor. Tensors can be visualized by means of tensor diagrams, as Penrose presented in [34]. Figure 3-1 shows the tensor diagram of a scalar, a vector, a matrix and a 3-way tensor in that specific order. It makes use of closed loops nodes and branches. Each branch stands for a single index of the tensor. This definition will be used throughout this whole chapter to demonstrate the various algorithms that will be presented.
Mode-n fibers are defined by fixing all but one index and are equivalent to a single row or column of a matrix. Here, n stands for the n^{th} index that is not fixed. As portrayed for a 3-way tensor in Figure 3-2, the subfigures respectively show the mode-1 fibers, mode-2 fibers and the mode-3 fibers. Hence, if we consider a 3-way tensor $\mathcal{X}$, the mode-1 fibers are denoted by the vector $\mathbf{x}_{:jk}$.
Tensor slices are generated when two indices are free and all others are fixed. Again, an example is shown for a 3-way tensor in Figure 3-3. From left to right, the subfigures portray the horizontals slices, the lateral slices and the frontal slices. The horizontal slices of tensor

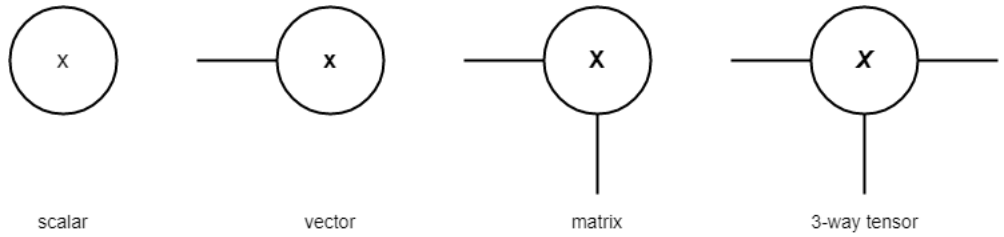

Figure 3-1: This figure shows the tensor diagrams of a scalar, a vector, a matrix and a 3-way tensor respectively.

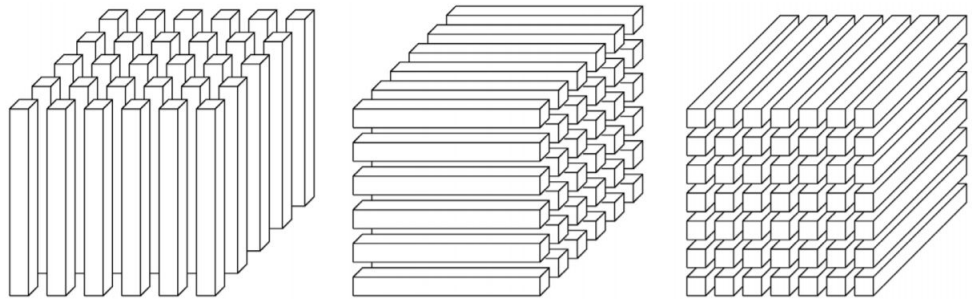

Figure 3-2: This figure shows the mode-n fibers of a 3-way tensor [20]. From left to right, the mode-1 or column fibers, mode-2 or row fibers and mode-3 or tube fibers can be seen.

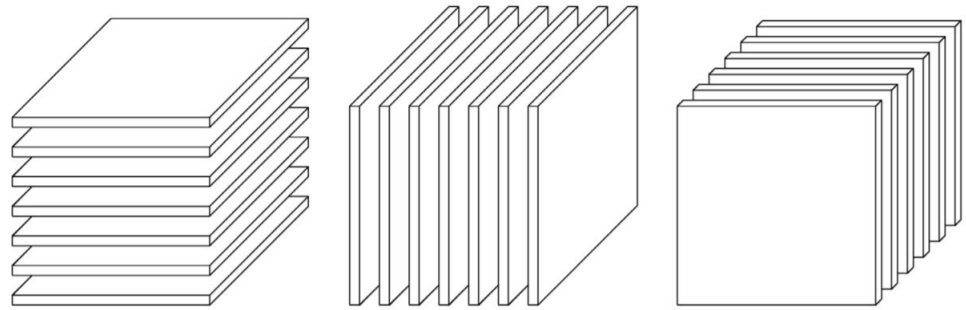

Figure 3-3: This figure shows the slices of a 3-way tensor [20]. From left to right, the horizontal slices, the lateral slices and the frontal slices can be seen.

$\mathcal{X}$ are matrices given by $\mathbf{X}_{i::}$.
The norm of a tensor $\mathcal{X}$ is equivalent to the square-root of the sum of the elements squared (3-1). This is equivalent to the Frobenius norm for a matrix

$$\|\mathcal{X}\| = \sqrt{\sum_{i_1}\sum_{i_2}\cdots\sum_{i_d} x^2_{i_1,i_2,\ldots,i_d}}. \tag{3-1}$$

3-2 Tensor Operations

This section is based on [20] unless stated otherwise.
This section covers the basic tensor operations, such as the inner product, matricization of a

tensor and the n-mode product.

3-2-1 Multi-index Ordering

The indices in this thesis are ordered according to the little-endian ordering, in order to be compatible with Matlab and de Rooij's thesis [12]. The ordering is given by (3-2),

$$\overline{i_1 i_2 \dots i_d} = i_1 + (i_2 - 1)I_1 + (i_3 - 1)I_1 I_2 + \dots + (i_d - 1)I_1 I_2 \cdots I_{d-1}. \tag{3-2}$$

3-2-2 Inner Product

The inner product of two tensors $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ and $\mathcal{Y} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_d}$ is the sum of the multiplication of each of the elements and only possible if $I_i = J_i \quad \forall i$ (3-3)

$$\langle \mathcal{X}, \mathcal{Y} \rangle = \sum_{i_1} \sum_{i_2} \cdots \sum_{i_d} x_{i_1, i_2, \dots, i_d} y_{i_1, i_2, \dots, i_d}. \tag{3-3}$$

3-2-3 Matricization

A given tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ can be unfolded into a matrix $\mathbf{X} \in \mathbb{R}^{I_n \times I_1 \cdots I_{n-1} I_{n+1} \cdots I_d}$. Hence, the n-mode matricization of tensor $\mathcal{X}$ is such that the n^{th} index are the rows of the matrix $\mathbf{X}_n$ and all other indices form one new index. The indices $(i_1, i_2, \dots, i_d)$ of tensor $\mathcal{X}$ transfer to a indices in the matrix $\mathbf{X}_n$ denoted by (i_n, j), where j is defined as in (3-4)

$$j = 1 + \sum_{k=1}^{d} (i_k - 1) \prod_{m=1}^{k-1} I_m. \tag{3-4}$$

In Figure 3-4 an example is given regarding the matricization of a 4-way tensor. In this figure it can be seen that the second index is isolated as a separate index, while the other indices are merged into $j = [i_1 i_3 i_4]$.

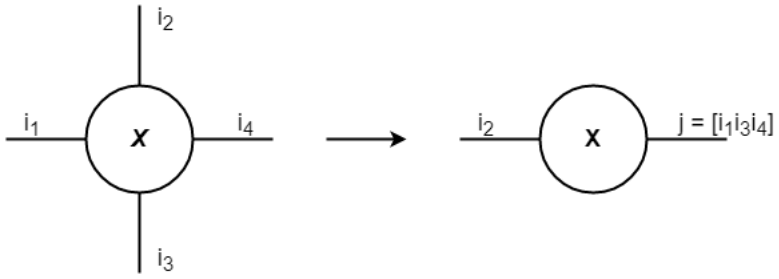

Figure 3-4: This figure illustrates the matricization process for a 4-way tensor.

3-2-4 n-mode Product

In this section the n-mode product is discussed. The n-mode product is the multiplication of a tensor by a matrix over a certain index n. To perform this tensor multiplication one first

needs to convert the tensor into matrices. Hence, the n-mode product uses the matricization procedure discussed in Section 3-2-3. The n-mode product is most clearly and very simply explained by use of tensor diagrams, such as the example of a 4-way tensor multiplied by a matrix displayed in Figure 3-5. As one can see, the indices i_4 and j_1 are connected. Hence, the tensor-matrix multiplication is performed over the fourth index of the tensor. Written in tensor format, the n-mode product is defined by $\mathcal{Y} = \mathcal{X} \times_n U$. In matrix format the formula becomes: $\mathbf{Y}_n = U \times \mathbf{X}_n$.

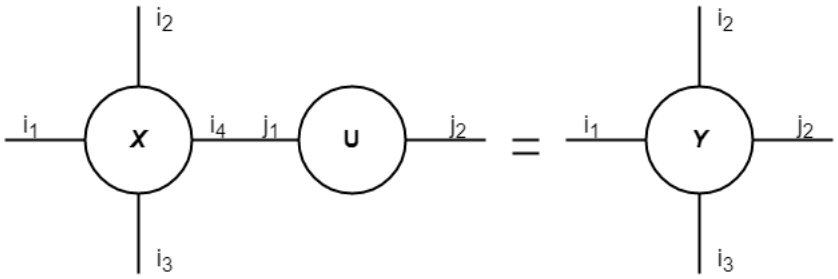

Figure 3-5: This figure shows the n-mode product in tensor diagram format.

3-2-5 Kronecker, Khatri-Rao and Hadamard Product

Unless stated otherwise, this section is based on [9].
The theory is presented for completeness but is assumed to be known. In this section some matrix/tensor products will be shortly discussed since they will be applied during the derivations of the tensor networks.
The tensor Kronecker product is denoted by $\otimes$ and computes the Kronecker product of two tensors. The general idea is that each element of tensor $\mathcal{X}$ is multiplied by tensor $\mathcal{Y}$. Consider $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ and $\mathcal{Y} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_d}$. Computing the Kronecker product of these tensors results in $\mathcal{Z} = \mathcal{X} \otimes \mathcal{Y} \in \mathbb{R}^{I_1 J_1 \times I_2 J_2 \times \cdots \times I_d J_d}$. The Khatri-Rao product [19] is a matrix product and is denoted by $\odot$. The general idea of the Khatri-Rao matrix product is to define a matrix whose columns equal the Kronecker product of the earlier two matrices' columns. Consider $\mathbf{X} \in \mathbb{R}^{I \times K}$ and $\mathbf{Y} \in \mathbb{R}^{J \times K}$, then the Katri-Rao product is defined as in (3-5)

$$\mathbf{X} \odot \mathbf{Y} = \begin{bmatrix} \mathbf{x}_1 \otimes \mathbf{y}_1 & \mathbf{x}_2 \otimes \mathbf{y}_2 & \dots & \mathbf{x}_d \otimes \mathbf{y}_d \end{bmatrix}, \tag{3-5}$$

where $\mathbf{Z} \in \mathbb{R}^{IJ \times K}$. The Hadamard product, also known as elementwise multiplication, is denoted by $*$. The product computes the product of single elements of two tensors. Consider $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ and $\mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$. The result of the Hadamard product, denoted by $\mathcal{Z}$, has the same dimensions as $\mathcal{X}$ and $\mathcal{Y}$ and its elements can be defined as, $z_{i_1,i_2,\dots,i_d} = x_{i_1,i_2,\dots,i_d} y_{i_1,i_2,\dots,i_d}$.

3-2-6 Rank-1 Tensors

As stated by Kolda [20], tensors can be composed via outer product of vectors. As a result, a d-way tensor can be formed out of the outer product of d vectors, $\mathcal{X} = \mathbf{x}_1 \circ \mathbf{x}_2 \circ \cdots \circ \mathbf{x}_d$. Furthermore, the idea is proposed that every tensor that can be exactly represented by the outer product of d vectors is a d^{th}-order rank-1 tensor.

3-3 Tensor Decompositions

This section will address tensor decompositions, mainly the TT and TTm decompositions. The theory presented in this section is based on [29] and [36]. The section will start off by obtaining a TT out of a tensor using the Tensor-Train Singular Value Decomposition (TT-SVD) algorithm. Moreover, the TTm and some other decompositions will be discussed.

3-3-1 Tensor-Train

As mentioned before, the Singular Value Decomposition (SVD) is used to obtain the cores of the TT. In short, the SVD is the decomposition of a matrix as seen in (3-6)

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathbf{T}}, \tag{3-6}$$

where $\mathbf{X} \in \mathbb{R}^{I \times J}$, $\mathbf{U} \in \mathbb{R}^{I \times I}$ and $\mathbf{V} \in \mathbb{R}^{J \times J}$ are orthogonal matrices containing the left- and right singular vectors respectively and $\mathbf{\Sigma} \in \mathbb{R}^{I \times J}$ is a diagonal matrix containing the singular values of the matrix $\mathbf{X}$ [37, 41].
Before describing the TT-SVD in detail, the definition of a TT will be discussed. Consider a d-dimensional tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, that is written in TT-format. The tensor can be decomposed into d cores, where each core is 3-way tensor. Hence, each core represents a single index of the original tensor as seen in (3-7)

$$\mathcal{X}(i_1, i_2, \ldots, i_d) = \mathcal{X}^{(1)}(i_1)\mathcal{X}^{(2)}(i_2) \cdots \mathcal{X}^{(d)}(i_d). \tag{3-7}$$

Each core has dimension $\mathcal{X}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$. Thus, when the index i_k is known and after permutation the core is a matrix with dimension $\mathbb{R}^{R_{k-1} \times R_k}$. The TT-ranks are indicated by R_k, and $R_0 = R_d = 1$. Because a specific element of a tensor is computed. In Figure 3-6 a graphical representation of a TT decomposition of a random 4-way tensor is given. Note that a 4-way tensor will result in 4 TT-cores. Thus, the information of the original d-way tensor is stored in d 3-way tensors and each element of the original tensor can be obtained by means of matrix-matrix multiplications.

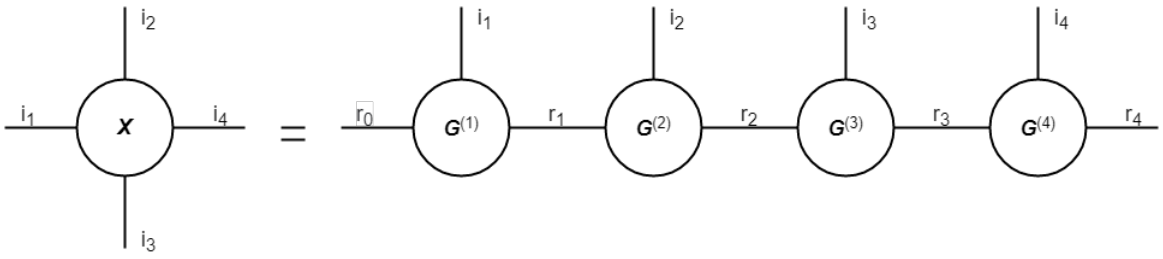

Figure 3-6: This figure shows the TT decomposition of any given 4-way tensor in a tensor diagram.

TT-SVD is an algorithm that computes these TT-cores via the SVD. The procedure of the algorithm will be given in tensor diagrams and will be further explained in words. Let $\mathcal{X}$ be the original tensor and $\mathcal{Y}$ be the TT approximation of $\mathcal{X}$. The goal of the algorithm is to approximate the tensor $\mathcal{X}$ by d TT-cores named $\mathcal{Y}$. The approximation $\mathcal{Y}$ must satisfy (3-8)

$$\|\mathcal{X} - \mathcal{Y}\|_F \leq \epsilon \|\mathcal{X}\|_F . \tag{3-8}$$

The full algorithm is presented in Figure 3-7. Initially, a new variable $\mathcal{C} = \mathcal{X}$ is created and used in the for-loop. In the first iteration $\mathcal{C}$ is reshaped into a matrix $\mathbf{C}$. Throughout the rest of the algorithm $\mathbf{C}$ is a matrix. Each iteration reshapes $\mathbf{C}$ into a matrix with dimensions $\mathbb{R}^{R_{k-1}I_k \times I_{k+2}\cdots I_d}$. Then, the SVD of this matrix is taken. The matrices generated by the SVD can be truncated. Truncation is an important procedure since it is a way of keeping the TT-ranks small. The TT-rounding algorithm will be discussed in Section 3-4-4. However, truncation also means that information is thrown away. Therefore, consider (3-8), since throwing away information results in a worse approximation of $\mathcal{X}$. A solution to this is to limit the Frobenius norm of $\mathbf{E}$, $\|\mathbf{E}\|_F$, by δ, which is a function of ϵ, see (3-9). Algorithm 1 gives the pseudo-code of the TT-SVD procedure.

$$\|\mathbf{E}\|_F \leq \delta, \quad \delta = \frac{\epsilon}{\sqrt{d-1}} \|\mathcal{X}\|_F . \tag{3-9}$$

Algorithm 1: TT-SVD [28, 29]

Data: d-dimensional tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, truncation parameter: ϵ

Result: A TT approximation $\mathcal{Y} = \langle\langle \mathcal{Y}^{(1)}, \mathcal{Y}^{(2)}, \ldots, \mathcal{Y}^{(d)} \rangle\rangle$ of $\mathcal{X}$. $\mathcal{Y}$ must have an approximation error smaller than ϵ as given by:

$$\|\mathcal{X} - \mathcal{Y}\|_F \leq \epsilon \|\mathcal{X}\|_F$$

Truncation parameter: $\delta = \frac{\epsilon}{\sqrt{d-1}} \|\mathcal{X}\|_F$
Temporary tensor: $\mathcal{C} = \mathcal{X}$
for *k = 1 to d - 1* **do**
  $\mathbf{C} \leftarrow \text{reshape}(\mathbf{C}, [r_{k-1}i_k, \frac{\text{numel}(\mathbf{C})}{r_{k-1}n_k}])$
  $\mathbf{U}_\delta(r_{k-1}i_k, r_k)\mathbf{\Sigma}_\delta(r_k, r_k)\mathbf{V}_\delta(\alpha_k, r_k)^T + \mathbf{E} \leftarrow \text{SVD}_{\text{trun}}(\mathbf{C}), \quad \|\mathbf{E}\|_F \leq \delta$
  $\mathcal{Y}^{(k)} \leftarrow \text{reshape}(\mathbf{U}_\delta, [r_{k-1}, i_k, r_k])$
  $\mathbf{C} \leftarrow \mathbf{\Sigma}_\delta \mathbf{V}_\delta^T$
end
$\mathcal{Y}^{(d)} \leftarrow \mathbf{C}$
Return $\mathcal{Y} = \langle\langle \mathcal{Y}^{(1)}, \mathcal{Y}^{(2)}, ..., \mathcal{Y}^{(d)} \rangle\rangle$.

3-3-2 Tensor-Train Matrix

The TTm decomposition can be obtained from a TT, as stated in [28]. Consider a very large matrix $\mathbf{X} \in \mathbb{R}^{I_1 \cdots I_d \times J_1 \cdots J_d}$. $\mathbf{X}$ can be reshaped and permuted into a tensor $\mathcal{X} \in \mathbb{R}^{I_1 J_1 \times \cdots \times I_d J_d}$. Now the TT-SVD algorithm can be used to obtain a TT approximation of $\mathbf{X}$. When doing this, the TT-cores are $\mathcal{X}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k J_k \times R_k}$. Then, the cores can be reshaped into $\mathcal{X}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times J_k \times R_k}$. Hence, the TTm is obtained.

3-3-3 Other Decompositions

This section is based on [9] unless stated otherwise.
Besides the TT(m) decomposition there are many other tensor decompositions. In this section, the general idea of some of these decompositions will be covered and a table presenting storage

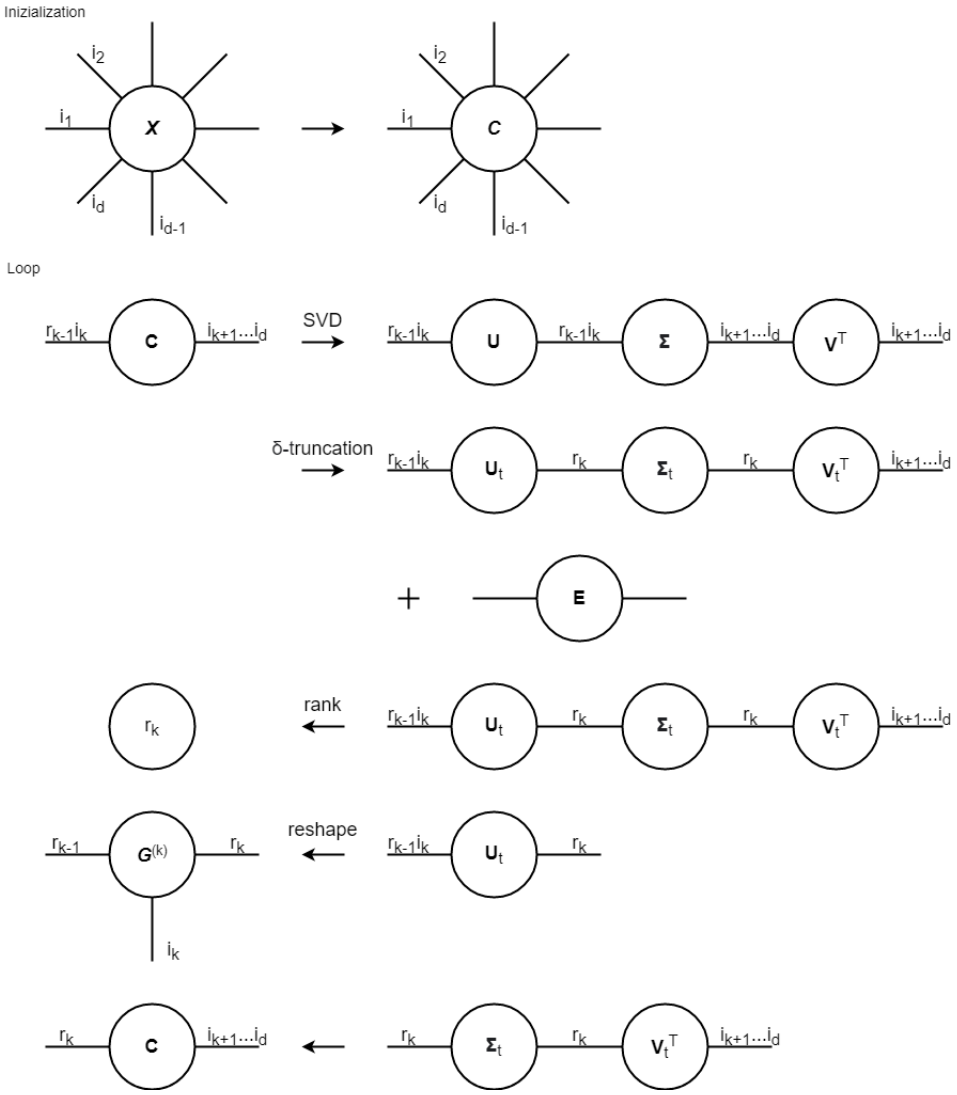

Figure 3-7: This figure shows the TT-SVD algorithm in tensor diagram format.

complexities is presented.
Two-way Component Analysis (CA) aims to decompose a matrix $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2}$ into two matrices $\mathbf{A} \in \mathbb{R}^{I_1 \times J}$ and $\mathbf{B} \in \mathbb{R}^{I_2 \times J}$. Constraints or a priori knowledge about the process can be imposed on the matrices $\mathbf{A}$ and $\mathbf{B}$. Two-way CA can be seen as a very general form of SVD, PCA, ICA and other forms of CA. The general formula for Two-way Component Analysis is given in (3-10) [8, 11]

$$\mathbf{X} = \mathbf{A}\mathbf{B}^T + \mathbf{E}, \tag{3-10}$$

where $\mathbf{E} \in \mathbb{R}^{I_1 \times I_2}$ is the residual matrix or noise. The Canonical Polyadic Decomposition (CPD), which is also known as PARAFRAC or CANDECOMP decomposes a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ into a linear combination of multiple d vector outer products multiplied by a constant λ_r, where the vectors are defined as $\mathbf{x}_r^n \in \mathbb{R}^{I_n \times 1}$. Considering Section 3-2-6, this

Decomposition	Storage Complexity
Tensor	$\mathcal{O}(I^d)$
TT	$\mathcal{O}(dIR^2)$
TTm	$\mathcal{O}(dI^2R^2)$
CPD	$\mathcal{O}(dIR)$
TD	$\mathcal{O}(dIR + R^N)$

Table 3-1: Storage complexity for tensor decompositions for tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$. Furthermore, $I = \max\{I_1, I_2, \ldots, I_d\}$, $R = \max\{R_1, R_2, \ldots, R_{d-1}\}$ and d is the amount of TT-cores [9].

means that tensor $\mathcal{X}$ can be decomposed into R rank-1 tensors, see (3-11) [4, 16, 17]

$$\mathcal{X} = \sum_{r=1}^{R} \lambda_r \mathbf{x}_r^1 \circ \mathbf{x}_r^2 \circ \ldots \circ \mathbf{x}_r^N. \tag{3-11}$$

The Tucker Decomposition (TD) method decomposes the tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ into a core tensor $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_d}$ and the mode-n unfoldings of $\mathcal{X}$ named $\mathbf{X}_n \in \mathbb{R}^{I_n \times R_n}$. The tensor can thus be expressed as in (3-12) [39, 42]. TD can be seen as a generalization of the Canonical Polyadic Decomposition (CPD), since the core tensor $\mathcal{G}$ is not diagonal as in CPD.

$$\mathcal{X} = \mathcal{G} \times_1 \mathbf{X}_1 \times_2 \mathbf{X}_2 \cdots \times_d \mathbf{X}_d \tag{3-12}$$

Besides the Two-way Component Analysis (CA), Canonical Polyadic Decomposition (CPD), and Tucker Decomposition (TD) there are other versions of these decompositions. More information is given by [9]. Furthermore, Cichocki compares the storage complexities of various tensor decompositions, which can be seen in Table 3-1. This table describes the storage complexities of a d-way tensor and four decompositions: TT, TTm, CPD and TD. This thesis will continue using the TT decomposition because it is simpler to implement low-rank matrix approximations [9], which allows for simple and intuitive implementation of tensor operations [10, 29] and efficient data storage.

3-4 TT(m) Operations

Unless stated otherwise, this section is based on [36, 29, 28].
This section will cover the most important operations in TT(m)-format, such as addition, multiplication, contraction and matrix and vector products. Furthermore, the section concludes with the TT-rounding function.

3-4-1 Basic TT Operations

TT addition

The addition of two d-dimensional TTs can be performed by addition of the single cores of each of the TTs. Consider two TTs, $\mathcal{A}$ and $\mathcal{B}$, as defined in (3-13)

$$\mathcal{A} = \langle\langle \mathcal{A}^{(1)}(i_1), \mathcal{A}^{(2)}(i_2), \ldots, \mathcal{A}^{(d)}(i_d) \rangle\rangle, \quad \mathcal{B} = \langle\langle \mathcal{B}^{(1)}(i_1), \mathcal{B}^{(2)}(i_2), \ldots, \mathcal{B}^{(d)}(i_d) \rangle\rangle. \tag{3-13}$$

The addition of $\mathcal{A}$ and $\mathcal{B}$ results in another TT named $\mathcal{C}$. Moreover, the TT-cores of $\mathcal{C} = \mathcal{A}+\mathcal{B}$ are given in (3-14),

$$\mathcal{C}^{(1)}(i_1) = \begin{bmatrix} \mathcal{A}^{(1)}(i_1) & \mathcal{B}^{(1)}(i_1) \end{bmatrix}, \quad \mathcal{C}^{(k)}(i_k) = \begin{bmatrix} \mathcal{A}^{(k)}(i_k) & 0 \\ 0 & \mathcal{B}^{(k)}(i_k) \end{bmatrix}, \quad \mathcal{C}^{(d)}(i_d) = \begin{bmatrix} \mathcal{A}^{(d)}(i_d) \\ \mathcal{B}^{(d)}(i_d) \end{bmatrix}. \tag{3-14}$$

The procedure can be checked in a very simple manner. When an addition of the same TT is done, the norm of the resulting TT must be equal to two times the norm of the original TT. In addition, the algorithm can be found in Appendix A-1-1.

Concatenation of two TTms

Concatenating two TT(m) is an operation that consists of multiple steps. Consider a TTm $\mathcal{A}$ and a TTm $\mathcal{B}$ that are being concatenated. It holds that both TTms must be of equal size and the partitioning of the row and column index must be identical. Both TTms must be extended with zero columns which can be realized by using the kronecker product of a vector with the TTms,

$$\begin{aligned} \text{kron}([1\ 0],\ \mathcal{A}) &= \begin{bmatrix} \mathcal{A}\ \mathbf{0} \end{bmatrix}, \\ \text{kron}([0\ 1],\ \mathcal{B}) &= \begin{bmatrix} \mathbf{0}\ \mathcal{B} \end{bmatrix}. \end{aligned} \tag{3-15}$$

The concatenation of the TTms can be concluded by addition of the extended TTms explained in Section 3-4-1.

TT multiplication with a constant

Multiplying a TT(m) with a constant is as trivial as multiplying a vector or a matrix with a constant. The TT(m) is scaled by the multiplication of a constant with one of its TT-cores.

Transpose

The transpose of a TTm is done by permuting the TTm-cores. Consider the following TT-cores of a random TTm: $\mathcal{G}^{(k)} \in \mathbb{R}^{R_{k-1}\times I_k \times J_k \times R_k}$. The transpose of the TTm is obtained by permuting the second and third index of each TT-core. The cores become $\mathcal{G}^{(k)} \in \mathbb{R}^{R_{k-1}\times J_k \times I_k \times R_k}$.

TT matrix inversion

As stated by Osedelets et al. in [30], an inverse can be approximated via a linear system since this is less computationally expensive. Different approaches have been found when the matrix has a typical structure, such as a Toeplitz matrix [32], sparse approximates [7], or low kronecker-rank matrices [27, 31]. For all these specific matrix inversions Newtons matrix inversion formula was used, where $\mathbf{X}$ is the inverse of $\mathbf{A}$.

$$\mathbf{X}_{k+1} = 2\mathbf{X}_k - \mathbf{X}_k\mathbf{A}\mathbf{X}_k \tag{3-16}$$

In their paper it is concluded that the various ways of finding the inverse of a TTm are preconditioners. Hence, more research must be done on the topic of TTm inversion.

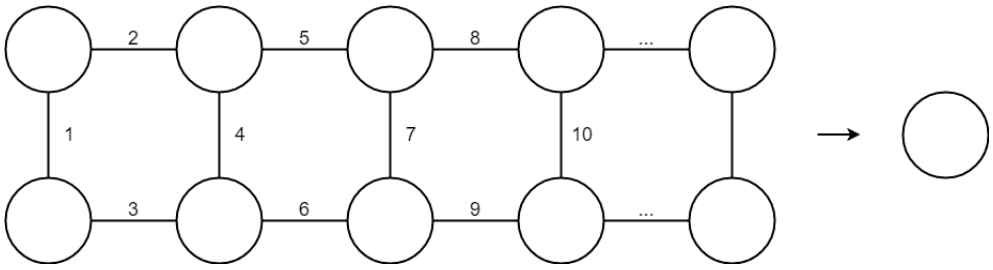

Figure 3-8: This figure shows the optimal order of contraction according to [36].

3-4-2 TT Contractions

Contraction is a very general procedure to combine connected TTs. This can be done in various manners, however [36] has presented an optimal way of contraction. In Figure 3-8 the order is presented, where the numbers at the connections indicate the right order of contraction.

Dot product

An application of TT contraction is the dot product. The dot product computes the inner or scalar product of two TTs, which can be done by contracting the connected TTs. The time complexity of the dot product equals $\mathcal{O}((d-2)(I(RS)^2)$ where the each of the d cores of the two TTs are defined by: $\mathcal{A}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$ and $\mathcal{B}^{(k)} \in \mathbb{R}^{S_{k-1} \times I_k \times S_k}$ and $R = \max(R_k)$, $S = \max(S_k)$ and $I = \max(I_k)$ [10, 12]. The algorithm of the inner product can be found in Appendix A-1-4.

3-4-3 Matrix and Vector Product

One of the most important functions in algebra is the matrix-vector and matrix-matrix product, since these operations are used in many applications. Formally, the matrix-vector product is computed as, $\mathbf{b} = \mathbf{A}\mathbf{x}$. When matrix $\mathbf{A}$ and vector $\mathbf{x}$ are defined in TT-format, it is possible to compute the product in TT-format and the structure of the TT(m)s is used. Suppose matrix $\mathbf{A}$ is represented by a TTm $\mathcal{A}$ with cores $\mathcal{A}^{(k)}$ and vector $\mathbf{x}$ is be represented by a TT $\mathcal{X}$ with cores $\mathcal{X}^{(k)}$. The product can be computed as a contraction per core, where the new cores of the TT of $\mathbf{b}$ are defined as in (3-17). The time complexity of the matrix vector product is $\mathcal{O}(dIJ(RS)^2)$ where the TTm and the TT both have d cores, and each of the cores are defined by: $\mathcal{A}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times J_k \times R_k}$ and $\mathcal{X}^{(k)} \in \mathbb{R}^{S_{k-1} \times J_k \times S_k}$. Furthermore, $R = \max(R_k)$, $S = \max(S_k)$, $I = \max(I_k)$ and $J = \max(J_k)$ [10, 12].

$$\mathcal{B}^{(k)}(i_k) = \sum_{j_k} \mathcal{A}^{(k)}(i_k, j_k) \otimes \mathcal{X}^{(k)}(j_k). \tag{3-17}$$

Hence, the product is shown in Figure 3-9. The TT-ranks of the matrix are denoted by R_k and the TT-ranks of the vector are denoted by S_k. Furthermore, the matrix-vector product displayed in this figure is $\mathbf{A}\mathbf{x}$, with the row indices of the matrix A denoted by I_k and the column indices denoted by J_k. Obviously, the column partitions of the matrix and the row

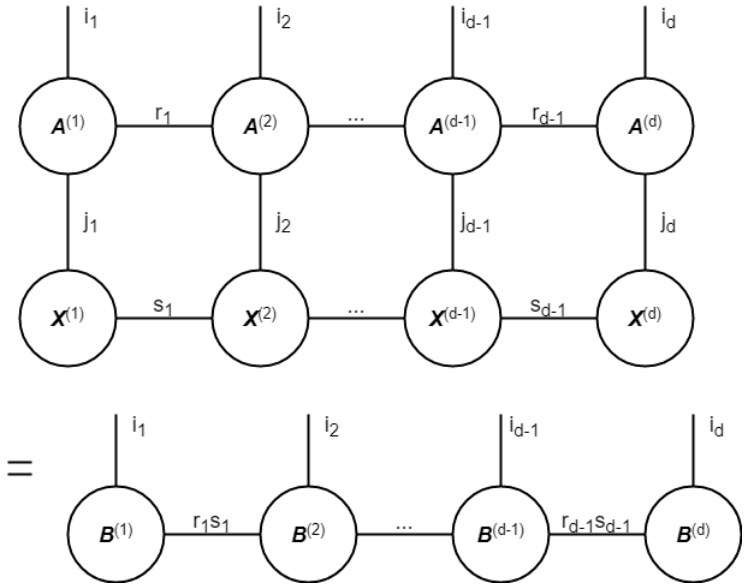

Figure 3-9: This figure illustrates the matrix-vector product in TT-format in tensor diagrams.

partitions of the vector must be identical. Therefore, both are denoted by J_k. Moreover, the algorithm of the matrix-vector is given in Appendix A-1-3.
The matrix-matrix product follows the same procedure as the matrix-vector product. Consider the matrix-matrix product, $\mathbf{C} = \mathbf{AB}$. Both the matrices $\mathbf{A}$ and $\mathbf{B}$ can be decomposed into TTms. Hence, the product of the TTm of $\mathbf{A}$ and $\mathbf{B}$ will yield the TTm of $\mathbf{C}$, which can be expressed as in (3-18). The time complexity of the matrix vector product is $\mathcal{O}(dIJL(RS)^2)$ where the TTms both have d cores, and each of the cores are defined by: $\mathcal{A}^{(k)} \in \mathbb{R}^{R_{k-1}\times I_k \times J_k \times R_k}$ and $\mathcal{X}^{(k)} \in \mathbb{R}^{S_{k-1}\times J_k \times L_k \times S_k}$. Furthermore, $R = \max(R_k)$, $S = \max(S_k)$, $I = \max(I_k)$, $J = \max(J_k)$ and $L = \max(L_k)$ [10, 12].

$$\mathcal{C}^{(k)}(i_k, l_k) = \sum_{j_k} \mathcal{A}^{(k)}(i_k, j_k) \otimes \mathcal{B}^{(k)}(j_k, l_k). \tag{3-18}$$

Consider the Algorithm 2 for matrix-matrix product in TT-format. It should be noticed

Algorithm 2: MatMatTT: Matrix-matrix product [29]

Data: $\mathcal{A} = \langle\langle \mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(d)} \rangle\rangle$ with $\mathcal{A}^{(k)} \in \mathbb{R}^{R_{k-1}, I_k, J_k, R_k}$,
$\mathcal{B} = \langle\langle \mathcal{B}^{(1)}, \mathcal{B}^{(2)}, \ldots, \mathcal{B}^{(d)} \rangle\rangle$ with $\mathcal{B}^{(k)} \in \mathbb{R}^{S_{k-1}, J_k, L_k, S_k}$.
Result: $\mathcal{C} = \langle\langle \mathcal{C}^{(1)}, \mathcal{C}^{(2)}, \ldots, \mathcal{C}^{(d)} \rangle\rangle$ with $\mathcal{C}^{(k)} \in \mathbb{R}^{R_{k-1}S_{k-1}, I_k, L_k, R_k S_k}$, as the product of $\mathcal{A}$ and $\mathcal{B}$.
for *k = 1 to d* **do**
  $\mathbf{A}^{(k)}(r_{k-1} i_k r_k, j_k) \leftarrow \mathcal{A}^{(k)}(r_{k-1}, i_k, j_k, r_k)$
  $\mathbf{B}^{(k)}(j_k, s_{k-1} l_k s_k) \leftarrow \mathcal{B}^{(k)}(s_{k-1}, j_k, l_k, s_k)$
  $\mathbf{C}^{(k)}(r_{k-1} i_k r_k, s_{k-1} l_k s_k) \leftarrow \mathbf{A}^{(k)}(r_{k-1} i_k r_k, j_k)\mathbf{B}^{(k)}(j_k, s_{k-1} l_k s_k)$
  $\mathcal{C}^{(k)}(r_{k-1} s_{k-1}, i_k, l_k, r_k s_k) \leftarrow \mathbf{C}^{(k)}(r_{k-1} i_k r_k, s_{k-1} l_k s_k)$
end
Return $\mathcal{C} = \langle\langle \mathcal{C}^{(1)}, \mathcal{C}^{(2)}, ..., \mathcal{C}^{(d)} \rangle\rangle$.

that the TT-ranks of the resulting matrix-vector and matrix-matrix product are equal to the product of the TT-ranks of the matrix and the vector. Hence, these ranks will grow for each product, which can cause computational problems. Therefore, the ranks have to be truncated to decrease the computational load. This can be done by using the TT-rounding algorithm which will be discussed in Section 3-4-4.

3-4-4 TT-rounding

As stated before the TT-rounding algorithm can be used to perform rounding. This procedure starts with a random existing TT with cores $\mathcal{G}^{(k)}$. Identical to the TT-SVD algorithm, the TT-rounding algorithm makes use of the SVD. However, when considering a random TT(m), one can not assume the norm is in one core. Hence, it is first required to bring the norm of the tensor into one single core. This is done via QR-factorization [37] of the core from left to right or from right to left meaning that the norm of the tensor is either in the leftmost or in the rightmost core. From this point onward, the truncated-SVD procedure can be used to truncate the TT-cores. This algorithm is presented in Algorithm 3. Performing the QR-

Algorithm 3: TT-rounding [29]

Data: $\mathcal{X} = \langle\langle \mathcal{X}^{(1)}, \mathcal{X}^{(2)}, \ldots, \mathcal{X}^{(d)} \rangle\rangle$ with $\mathcal{X}^{(k)} \in \mathbb{R}^{R_{k-1}, I_k, R_k}$, truncation parameter: ϵ

Result: $\mathcal{Y} = \langle\langle \mathcal{Y}^{(1)}, \mathcal{Y}^{(2)}, \ldots, \mathcal{Y}^{(d)} \rangle\rangle$ with $\mathcal{Y}^{(k)} \in \mathbb{R}^{S_{k-1}, I_k, S_k}$, where $\mathcal{Y}$ is a TT approximation of $\mathcal{X}$ with δ-truncated cores. $\mathcal{Y}$ must have an approximation error smaller than ϵ as given by:

$$\|\mathcal{X} - \mathcal{Y}\|_F \leq \epsilon \|\mathcal{X}\|_F$$

Truncation parameter: $\delta = \frac{\epsilon}{\sqrt{d-1}} \|\mathcal{X}\|_F$

for *k = d to 2 step -1* **do**

- $[\mathbf{X}^{(k)}(\beta_{k-1}, i_k\beta_k), \mathbf{R}(r_{k-1}, \beta_{k-1})] \leftarrow \text{QR}(\mathbf{X}^{(k)}(r_{k-1}, i_k\beta_k))$
- $\mathcal{X}^{(k)}(\beta_{k-1}, i_k, \beta_k) \leftarrow \text{reshape}(\mathbf{X}^{(k)}, [B_{k-1}, I_k, B_k])$
- $\mathcal{X}^{(k-1)} \leftarrow \mathcal{X}^{(k-1)} \times_3 \mathbf{R}$

end

for *k = 1 to d - 1* **do**

- $\mathbf{Y}^{(k)} \leftarrow \text{reshape}(\mathcal{X}^{(k)}, [S_{k-1}I_k, B_k])$
- $\mathbf{U}_\delta(s_{k-1}i_k, s_k)\mathbf{\Sigma}_\delta(s_k, s_k)\mathbf{V}_\delta(\beta_k, s_k)^T + \mathbf{E} \leftarrow \text{SVD}_{\text{trun}}(\mathbf{Y}^{(k)}), \quad \|\mathbf{E}\|_F \leq \delta$
- $\mathcal{Y}^{(k)} \leftarrow \text{reshape}(\mathbf{U}_\delta, [S_{k-1}, I_k, S_k])$
- $\mathcal{Y}^{(k+1)} \leftarrow \mathcal{Y}^{(k+1)} \times_1 \mathbf{\Sigma}_\delta \mathbf{V}_\delta^T$

end

Return $\mathcal{Y} = \langle\langle \mathcal{Y}^{(1)}, \mathcal{Y}^{(2)}, ..., \mathcal{Y}^{(d)} \rangle\rangle$.

factorization from right to left, as is done in the Algorithm 3, is a procedure that is used to bring the norm into one TT-core. This specific procedure is highlighted in tensor diagram format for one random TT-core $\mathcal{G}^{(k)}$ in Figure 3-10. The TT-core $\mathcal{G}^{(k)}$ is reshaped into a matrix $\mathbf{G}^{(k)}$. Furthermore, a QR-factorization of the matrix $\mathbf{G}^{(k)}$ is done, resulting in an orthogonal matrix $\mathbf{Q}$ and a triangular matrix $\mathbf{R}$. Note that the norm of the original TT-core is in $\mathbf{R}$. Hence, the orthogonalized TT-core $\mathcal{G}^{(k)}$ is, after reshaping, equal to $\mathbf{Q}$ and the next TT-core $\mathcal{G}_{k-1}$ is equal to $\mathcal{G}^{(k-1)} \times_3 \mathbf{R}$. Thus, the next TT-core contains its own norm and the

norm of the previous TT-core. When this procedure is initialized at $k = d$, the k^{th} TT-core will contain the norms of all TT-cores from $k = d$ to $k = k$. In addition, the computational complexity of the TT-rounding function equals $\mathcal{O}(dIR^3)$, where d equals the amount of cores, $I = \max\{I_1, I_2, \ldots, I_N\}$ and R is the maximum TT-rank [10].

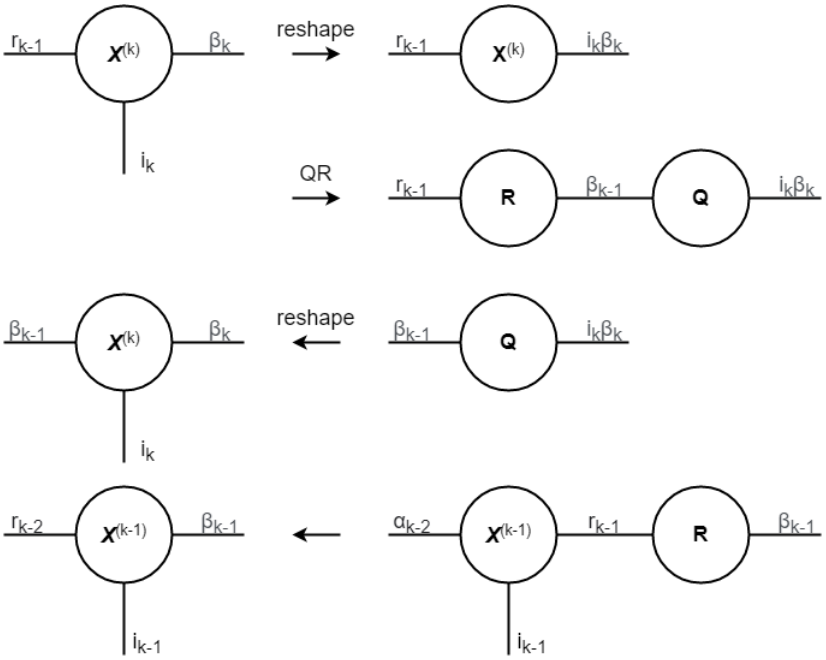

Figure 3-10: This figure illustrates the QR-orthogonalization part of the TT-algorithm.

Chapter 4

Square-Root Kalman Filter

This chapter is a continuation and generalization of Chapter 2, such that it takes a step back from the Tensor-Networked Kalman Filter (TNKF) to the conventional Kalman filter. Section 4-1 introduces a general state-space system that is used by Section 4-2 to derive the Kalman filter. Next, the proposed solution is introduced: a square-root Kalman filter.
A famous moment in history was defined by the space race of the 1960s, in which both the United States and the Soviet Union competed. The space race created a great need for research and development. One of those needs was the real-time positioning of spaceships when only discrete time information about the location of the ship was known. The Kalman filter proposed a solution to this problem. First introduced in 1960 by Rudolf Kalman [18], the Kalman filter proposed a new solution to linear filtering and prediction for discrete modeled systems with measurement and process noise. It is now a widely applicable theory. Among other uses, the Kalman filter can be applied to analytical chemistry [3], to economics, for example the estimation of assets pricing [33], to signal processing, for example when applied to solve robotics vision problems [6] and to navigation, such as in the first Global Positioning System (GPS) and inertial navigation [13].
In this chapter the Kalman filter will be discussed in Section 4-2 along with the Square-Root Kalman Filter (SRKF) in Section 4-4.

4-1 State-Space Representation

In this section the Kalman filter will be derived. The derivation is based on the Bayesian point of view, in which the mean and covariance of the state can be described as a function of Gaussian processes as is discussed in [26].
The linear time-variant state-space model of any given system is given by (4-1)

$$\begin{aligned} \mathbf{x}[k+1] &= \mathbf{A}[k]\mathbf{x}[k] + \mathbf{B}[k]\mathbf{u}[k] + \mathbf{w}[k] \\ \mathbf{y}[k] &= \mathbf{C}[k]\mathbf{x}[k] + \mathbf{D}[k]\mathbf{u}[k] + \mathbf{v}[k]. \end{aligned} \tag{4-1}$$

From now on, the following definitions will be used. $\mathbf{x}[k] \in \mathbb{R}^N$ is the state vector, $\mathbf{u}[k] \in \mathbb{R}^M$ is the input vector and $\mathbf{y}[k] \in \mathbb{R}^L$ is the output vector. Furthermore, $\mathbf{w}[k] \in \mathbb{R}^N$ is the

process noise and $\mathbf{v}[k] \in \mathbb{R}^L$ is the measurement noise. The system matrices are defined as: $\mathbf{A}[k] \in \mathbb{R}^{N\times N}$ is the state transition matrix, $\mathbf{B}[k] \in \mathbb{R}^{N\times M}$ is the input matrix, $\mathbf{C}[k] \in \mathbb{R}^{L\times N}$ is the output matrix and $\mathbf{D}[k] \in \mathbb{R}^{L\times M}$ is the feedforward matrix.
Furthermore, an assumption of the Kalman filter is that the process and measurement noise are white Gaussian noise. This implies statistical properties on $\mathbf{w}[k]$ and $\mathbf{v}[k]$ as defined in (4-2)

$$\mathbf{w}[k] \simeq \mathcal{N}(0, \mathbf{Q}[k]), \quad \mathbf{v}[k] \simeq \mathcal{N}(0, \mathbf{V}[0]), \tag{4-2}$$

where $\mathbf{Q} \in \mathbb{R}^{N\times N}$ and $\mathbf{V} \in \mathbb{R}^{L\times L}$. The initial condition of the state $\mathbf{x}[0]$ may not be known a priori and can therefore be modeled by a mean vector and a covariance matrix, in which the covariance matrix is semi-positive definite (4-3)

$$\mathbf{x}[0] \simeq \mathcal{N}(\bar{\mathbf{x}}[0], \mathbf{P}[0]), \tag{4-3}$$

where $\mathbf{P}[0] \in \mathbb{R}^{N\times N}$. Finally, a record of all the measurements for k equaling 1 to k are stored into a large vector $\mathbf{Y}[k] = [\mathbf{y}[1], \mathbf{y}[2], \ldots, \mathbf{y}[k]]^T \in \mathbb{R}^{Lk}$, which is realized by $\mathbf{Y}_k$.

4-2 Kalman Filter

In order to derive the Kalman update equations, proof by induction is used. The assumption that $\mathbf{x}[k]$ depending on $\mathbf{Y}_k$ $\forall k$ are Gaussian distributions is the basis for this derivation. For a proof by induction, two cases are needed: the base and the induction step. The base step is given by (4-3) which states that $\mathbf{x}[0]$ is Gaussian. In the derivation of the Kalman filter the notation of a probability density functions will be used. An example is the probability density function of $\mathbf{x}[k]$: $\mathbf{P}(\mathbf{x}[k])$. The distribution of $\mathbf{x}[0]$ depending on $\mathbf{Y}_0$ equals the distribution of $\mathbf{x}[0]$ because there are no measurements at time $k = 0$ or prior. Hence, $\mathbf{P}(\mathbf{x}[0]|\mathbf{Y}[0])$ equals $\mathbf{P}(\mathbf{x}[0])$ and is a Gaussian distribution. In the induction step, the distribution of $\mathbf{x}[k-1]$ depending on $\mathbf{Y}_{k-1}$ is assumed to be Gaussian. Furthermore, the propagation of the distribution is analyzed from time t_{k-1}^+ to t_k^+. This can be subdivided into two parts, from t_{k-1}^+ to t_k^-, which is the time propagation step and from t_k^- to t_k^+, which is the measurement update.

4-2-1 Time Propagation

The distribution $\mathbf{P}(\mathbf{x}[k]|\mathbf{Y}[k-1])$ is of interest after the time propagation. The state-space model in (4-1) shows that $\mathbf{x}[k]$ is a linear combination of $\mathbf{x}[k-1]$, $\mathbf{u}[k-1]$ and $\mathbf{w}[k-1]$. Thus, $\mathbf{P}(\mathbf{x}[k]|\mathbf{Y}[k-1])$ is a Gaussian distribution if $\mathbf{P}(\mathbf{x}[k-1], \mathbf{u}[k-1], \mathbf{w}[k-1]|\mathbf{Y}[k-1])$ is a Gaussian distribution. Note that $\mathbf{u}[k-1]$ is a known value, independent of $\mathbf{Y}[k-1]$ and is not a distribution. Hence, it can be ignored in determining the distribution $\mathbf{P}(\mathbf{x}[k]|\mathbf{Y}[k-1])$. Further derivations are shown in (4-4)

$$\begin{aligned}
\mathbf{P}(\mathbf{x}[k-1], \mathbf{w}[k-1]|\mathbf{Y}[k-1]) &= \frac{\mathbf{P}(\mathbf{x}[k-1], \mathbf{w}[k-1], \mathbf{Y}[k-1])}{\mathbf{P}(\mathbf{Y}[k-1])} \\
&= \frac{\mathbf{P}(\mathbf{x}[k-1], \mathbf{Y}[k-1])\mathbf{P}(\mathbf{w}[k-1])}{\mathbf{P}(\mathbf{Y}[k-1])} \\
&= \mathbf{P}(\mathbf{x}[k-1]|\mathbf{Y}[k-1])\mathbf{P}(\mathbf{w}[k-1]),
\end{aligned} \tag{4-4}$$

where $\mathbf{w}[k-1]$ is independent of $\mathbf{Y}[k-1]$ and can be separated. The final equation of (4-4) is a multiplication of two Gaussian distributions, which is still a Gaussian distribution. Thus, $\mathbf{P}(\mathbf{x}[k]|\mathbf{Y}[k-1])$ is a Gaussian distribution. Its mean and covariance are specified as follows, (4-5)

$$\begin{aligned} &E\left[\mathbf{x}[k]|\mathbf{Y}[k-1]=\mathbf{Y}_{k-1}\right]=\hat{\mathbf{x}}[k^-]=\mathbf{A}[k-1]\mathbf{x}[k-1^+]+\mathbf{B}[k-1]\mathbf{u}[k-1] \\ &E\left[(\mathbf{x}[k]-\hat{\mathbf{x}}[k^-])(\mathbf{x}[k]-\hat{\mathbf{x}}[k^-])^T|\mathbf{Y}[k-1]=\mathbf{Y}_{k-1}\right]=\mathbf{P}[k^-]= \\ &\qquad \mathbf{A}[k-1]\mathbf{P}[k-1^+]\mathbf{A}[k-1]+\mathbf{Q}[k-1]. \end{aligned} \tag{4-5}$$

4-2-2 Measurement Update

To complete the proof by induction, it must be shown that $\mathbf{P}(\mathbf{x}[k]|\mathbf{Y}[k])$ is a Gaussian distribution. Hence, the new measurement that is available at time k must be used to update the estimates of $\hat{\mathbf{x}}[k]^-$ and $\mathbf{P}[k]^-$. First, the distribution will be rewritten by means of the Bayes' rule $\mathbf{P}(\mathbf{x}[k]|\mathbf{Y}[k])$ with the aim of using the second part of (4-1) later on,

$$\begin{aligned} \mathbf{P}(\mathbf{x}[k]|\mathbf{Y}[k]) &= \frac{\mathbf{P}(\mathbf{x}[k],\mathbf{Y}[k])}{\mathbf{P}(\mathbf{Y}[k])} \\ &= \frac{\mathbf{P}(\mathbf{x}[k],\mathbf{y}[k],\mathbf{Y}[k-1])}{\mathbf{P}(\mathbf{y}[k],\mathbf{Y}[k-1])} \\ &= \frac{\mathbf{P}(\mathbf{y}[k]|\mathbf{x}[k],\mathbf{Y}[k-1])\mathbf{P}(\mathbf{x}[k]|\mathbf{Y}[k-1])}{\mathbf{P}(\mathbf{y}[k]|\mathbf{Y}[k-1])}. \end{aligned} \tag{4-6}$$

Note that $\mathbf{P}(\mathbf{x}[k]|\mathbf{Y}[k-1])$ is a Gaussian distribution as is discussed in Section 4-2-1. The distribution $\mathbf{P}(\mathbf{y}[k]|\mathbf{x}[k],\mathbf{Y}[k-1])$ is Gaussian because $\mathbf{y}[k]$ is a linear combination of a known vector $\mathbf{x}[k]$ with realization ξ and a white Gaussian noise vector $\mathbf{v}[k]$. Furthermore, the distribution in the denominator $\mathbf{P}(\mathbf{y}[k]|\mathbf{Y}[k-1])$ is also a Gaussian distribution because it can be rewritten into a multiplication of $\mathbf{P}(\mathbf{v}[k])$ and $\mathbf{P}(\mathbf{x}[k]|\mathbf{Y}[k-1])$, which are both Gaussian distributions. Thus, $\mathbf{P}(\mathbf{x}[k]|\mathbf{Y}[k])$ is a Gaussian distribution and the measurement updates can be written as in (4-7)

$$\begin{aligned} \hat{\mathbf{x}}[k^+] &= \hat{\mathbf{x}}[k^-]+\mathbf{P}[k^-]\mathbf{C}[k]^T\left[\mathbf{C}[k]\mathbf{P}[k^-]\mathbf{C}[k]^T+\mathbf{R}_c[k]\right]^{-1}(\mathbf{y}[k]-\mathbf{C}[k]\hat{\mathbf{x}}[k^-]) \\ \mathbf{P}[k^+] &= \mathbf{P}[k^-]-\mathbf{P}[k^-]\mathbf{C}[k]^T\left[\mathbf{C}[k]\mathbf{P}[k^-]\mathbf{C}[k]^T+\mathbf{R}_c[k]\right]^{-1}\mathbf{C}[k]\mathbf{P}[k^-]. \end{aligned} \tag{4-7}$$

Furthermore, the Kalman gain $K[k]$ is defined as $\mathbf{P}[k^-]\mathbf{C}[k]^T\left[\mathbf{C}[k]\mathbf{P}[k^-]\mathbf{C}[k]^T+\mathbf{R}_c[k]\right]^{-1}$.

4-2-3 Positive Definiteness of State Covariance Matrix

As explained in [14], computing the covariance matrix as presented in (4-7) requires a $L \times L$ matrix inversion. A large problem with calculating the covariance matrix $\mathbf{P}[k]$ on a computer is finite word length or computer round-off. This can cause the covariance matrix to lose its positive definiteness condition on which the time propagation (4-5) and measurement update (4-7) equations are based. In some scenarios this can result in a covariance matrix with negative values, which can further develop into a negative Kalman gain and diverging state

estimations. Based on [40], upper bounds on the error propagation for computation can be given. Assigning upper bounds to the conventional Kalman filter and the SRKF shows that the bound on the square-root covariance matrix $\mathbf{L}[k]$ is tighter, where $\mathbf{L}$ denotes the square-root covariance matrix. A square-root Kalman filter reformulates the Riccati update equations such that the square-root of the covariance matrix is updated. In the next section, various square-root Kalman filters will be discussed.

4-3 Cholesky Factorization

Unless stated otherwise, this section is based on [25].
As was stated before, square-root Kalman filters differ from conventional Kalman filters because the algorithm updates the square-root of the covariance matrix. This enforces $\mathbf{P}[k]$ to be positive definite and symmetric. The need for a square-root filter may seems to be a problem at the time that computers used a smaller word length. It can however still be a problem as is seen in [12] for example. Therefore, this section introduces the square-root or the Cholesky factorization of a matrix.
Let $\mathbf{P}$ be an $n \times n$ positive semidefinite, symmetric matrix. By definition $\mathbf{P}$ can be written as a multiplication of a matrix $\mathbf{L}$ times its transpose in which $\mathbf{L}$ is the square-root of $\mathbf{P}$ (4-8)

$$\mathbf{L}\mathbf{L}^T = \mathbf{P}. \tag{4-8}$$

There are many matrices $\mathbf{L}$ that satisfy this condition. A unique solution can be imposed by adding constraints to the factorization matrix. The Cholesky factorization does this by constraining $\mathbf{L}$ to be lower or upper triangular, in which $\mathbf{L}$ is defined as $\sqrt[c]{\mathbf{P}}$. Assume that the initial covariance matrix $\mathbf{P}[0]$ is given, then the Cholesky factorization must be computed such that it can be used for the recursive covariance matrix updates. To obtain a lower triangular square-root of a matrix, the procedure scans through the matrix, as can be seen in Figure 4-1. A lower triangular Cholesky square-root matrix can be computed in the following way, iterating for $i = 1, 2, \ldots, N$:

$$\sqrt[c]{\mathbf{P}}_{ij} = \begin{cases} (1/\sqrt[c]{\mathbf{P}}_{jj}) \left[\mathbf{P}_{ij} - \sum_{k=1}^{j-1} \sqrt[c]{\mathbf{P}}_{ik} \sqrt[c]{\mathbf{P}}_{jk} \right] & j = 1, 2, \ldots, i-1 \\ \left(\mathbf{P}_{ii} - \sum_{k=1}^{i-1} \sqrt[c]{\mathbf{P}}_{ik}^2 \right)^{1/2} & j = i \\ 0 & j > i, \end{cases} \tag{4-9}$$

where i and j are the row and column index respectively. If an upper triangular Cholesky factorization is desired, which is denoted by $\sqrt[c']{\mathbf{P}}$, the same algorithm can be used backwards. See (4-10) when iterating for $j = N, N-1, \ldots, 1$

$$\sqrt[c']{\mathbf{P}}_{ij} = \begin{cases} 0 & i > j \\ \left(\mathbf{P}_{jj} - \sum_{k=j+1}^{n} \sqrt[c']{\mathbf{P}}_{jk}^2 \right)^{1/2} & i = j \\ (1/\sqrt[c']{\mathbf{P}}_{jj}) \left[\mathbf{P}_{ij} - \sum_{k=j+1}^{n} \sqrt[c']{\mathbf{P}}_{ik} \sqrt[c']{\mathbf{P}}_{jk} \right] & i = j-1, j-2, \ldots, 1. \end{cases} \tag{4-10}$$

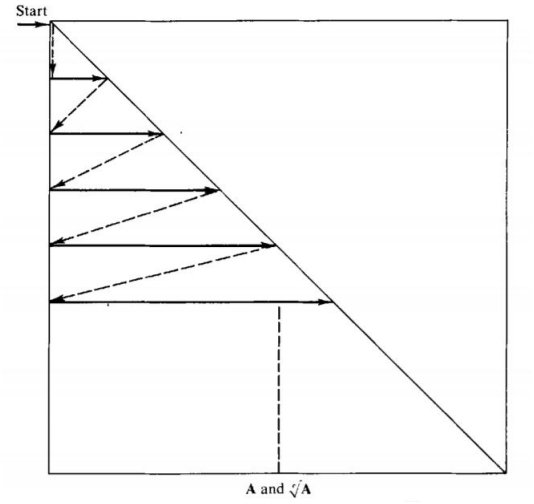

Figure 4-1: This figure shows the order of the scanning procedure of a random matrix to generate the Cholesky factorization [25].

4-4 General Approach for the Square-Root Kalman Filter

A general approach for a square-root implementation is presented by F. Gustafsson in his book *Adaptive Filtering and Change Detection* [15]. It is important to understand the move towards a square-root Kalman filter. From [12] it became very clear that no increase in performance was seen when more pixels are observed. This is caused by the forced rank-1 approximation of the covariance matrix. Due to this limitation a lot of information is thrown away. This is exactly where the square-root Kalman filter comes into play. The square-root version ensures that the covariance matrix $\mathbf{P}$ is symmetric, positive definite and ensures a numerically stable algorithm [25].

4-4-1 Square-Root Time Propagation

The time propagation as described in Chapter 7.5 of [25] will be used to compute the square-root prediction update of the covariance matrix. Recall equation (7-28) from that chapter, where a orthonormal transformation matrix $\mathbf{T}$ is found to make sure the update of $\mathbf{L}$ is square. This can be done via the QR factorization. The update equations for the time propagation are presented in (4-11),

$$\begin{aligned} \hat{\mathbf{x}}[k^-] &= \hat{\mathbf{x}}[k-1^+] \\ \begin{bmatrix} \mathbf{L}[k^-]^T \\ \mathbf{0} \end{bmatrix} &= \mathbf{T}^T \left[\mathbf{L}[k-1^+], \mathbf{W} \right]^T \end{aligned} \tag{4-11}$$

Consider the covariance matrix $\mathbf{P}$ that is equal to $\mathbf{L}\mathbf{L}^T$ as is described in (4-8). This is true because $\mathbf{L}$ is the Cholesky factor of $\mathbf{P}$. The time propagation formula for the update of the covariance matrix is given in (4-5). Note that the transition matrix is identity. Combining these two equation yields,

$$\mathbf{P}[k^-] = \mathbf{L}[k^-]\mathbf{L}[k^-]^T = \mathbf{P}[k-1^+] + \mathbf{Q}. \tag{4-12}$$

Furthermore, $\mathbf{P}[k-1^+]+\mathbf{Q}$ can be decomposed into $\begin{bmatrix}\mathbf{L}[k-1^+] & \mathbf{W}\end{bmatrix}\begin{bmatrix}\mathbf{L}[k-1^+], & \mathbf{W}\end{bmatrix}^T$. Integrating this into (4-12) concludes,

$$\mathbf{L}[k^-]=\begin{bmatrix}\mathbf{L}[k-1^+] & \mathbf{W}\end{bmatrix}. \tag{4-13}$$

Since this update of the covariance matrix yields a covariance matrix that is non-square, it will not be accepted. A QR decomposition of the matrix results in the following,

$$\begin{bmatrix}\mathbf{L}[k-1^+] & \mathbf{W}\end{bmatrix}^T=\mathbf{TR}. \tag{4-14}$$

Hence, it can be seen that the covariance matrix is equal to the product of the transposed upper-triangular matrix and the upper-triangular matrix,

$$\mathbf{P}[k^-]=\mathbf{R}^T\mathbf{T}^T\mathbf{TR}=\mathbf{R}^T\mathbf{R}. \tag{4-15}$$

The QR decomposition is used to compute the orthogonal transformation matrix $\mathbf{T}$ and the upper-triangular matrix $\mathbf{R}$ containing the norm. The dimension of $\mathbf{L}[k-1^+]$ is $\mathbb{R}^{N\times N}$ and of $\mathbf{W}$ is $\mathbb{R}^{N\times N}$. Hence, the dimension of $\begin{bmatrix}\mathbf{L}[k-1^+] & \mathbf{W}\end{bmatrix}^T$ equals $\mathbb{R}^{2N\times N}$. This implies that applying the QR decomposition of this matrix results in the orthogonal transformation matrix $\mathbf{T}\in\mathbb{R}^{2N\times 2N}$ and the upper triangular matrix $\mathbf{R}\in\mathbb{R}^{2N\times N}$. Since $\mathbf{R}$ is upper-triangular, it can be partitioned into two parts: $\mathbf{R}_1\in\mathbb{R}^{N\times N}$ and $\mathbf{R}_2\in\mathbb{R}^{N\times N}$, where $\mathbf{R}_2$ has only zero rows.

$$\mathbf{R}=\begin{bmatrix}\mathbf{R}_1\\ \mathbf{R}_2\end{bmatrix}=\begin{bmatrix}\mathbf{R}_1\\ \mathbf{0}\end{bmatrix}. \tag{4-16}$$

Filling this result into (4-15) yields:

$$\mathbf{P}[k^-]=\begin{bmatrix}\mathbf{R}_1^T & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{R}_1\\ \mathbf{0}\end{bmatrix}=\mathbf{R}_1^T\mathbf{R}_1 \tag{4-17}$$

The final step is to combine (4-12) and (4-17) to state that $\mathbf{L}[k^-]=\mathbf{R}_1^T$. This concludes the proof that the square-root update can be found via a QR decomposition of $\begin{bmatrix}\mathbf{L}[k-1^+] & \mathbf{W}\end{bmatrix}^T$.

4-4-2 Square-Root Measurement Update

An intuitive approach would be that of the Carlson filter. However, in this section the measurement update can be derived analogously to the time propagation. First, consider (4-7) where the update of the covariance matrix $\mathbf{P}[k-1^+]$ is presented. A matrix needs to be constructed such that the desired result is obtained after performing a QR decomposition. This specific matrix is given in (4-18)

$$\begin{bmatrix}\mathbf{R}_c^{1/2} & \mathbf{CL}[k^-]\\ \mathbf{0} & \mathbf{L}[k^-]\end{bmatrix}. \tag{4-18}$$

Although, there is no measurement uncertainty in this state-space model, it is used here to proof the approach. When the measurement covariance matrix is a zero matrix, this

approach is still valid. Taking the QR decomposition of this matrix results in (4-19), where $\mathbf{R}^T$ is lower-triangular and can be partitioned according to (4-20)

$$\begin{bmatrix} \mathbf{R}_c^{1/2} & \mathbf{CL}[k^-] \\ \mathbf{0} & \mathbf{L}[k^-] \end{bmatrix} = \mathbf{R}^T\mathbf{T}^T, \tag{4-19}$$

$$\mathbf{R}^T = \begin{bmatrix} \mathbf{R}_{11}^T & \mathbf{0} \\ \mathbf{R}_{12}^T & \mathbf{R}_{22}^T \end{bmatrix}. \tag{4-20}$$

Hence, $\mathbf{R}_{11}$, $\mathbf{R}_{21}$ and $\mathbf{R}_{22}$ can be computed. The matrix given in (4-18) can be multiplied by its transpose: $\mathbf{R}^T\mathbf{R}$,

$$\begin{aligned} \begin{bmatrix} \mathbf{R}_c^{1/2} & \mathbf{CL}[k^-] \\ \mathbf{0} & \mathbf{L}[k^-] \end{bmatrix} \begin{bmatrix} \mathbf{R}_c^{1/2} & \mathbf{CL}[k^-] \\ \mathbf{0} & \mathbf{L}[k^-] \end{bmatrix}^T &= \begin{bmatrix} \mathbf{CL}[k^-]\mathbf{L}[k^-]^T\mathbf{C}^T & \mathbf{CL}[k^-]\mathbf{L}[k^-]^T \\ \mathbf{L}[k^-]\mathbf{L}[k^-]^T\mathbf{C}^T & \mathbf{L}[k^-]\mathbf{L}[k^-]^T \end{bmatrix} \\ = \mathbf{R}^T\mathbf{T}^T\mathbf{T}\mathbf{R} = \mathbf{R}^T\mathbf{R} &= \begin{bmatrix} \mathbf{R}_{11}^T\mathbf{R}_{11} & \mathbf{R}_{11}^T\mathbf{R}_{12} \\ \mathbf{R}_{12}^T\mathbf{R}_{11} & \mathbf{R}_{12}^T\mathbf{R}_{12} + \mathbf{R}_{22}^T\mathbf{R}_{22} \end{bmatrix} \end{aligned} \tag{4-21}$$

From (4-21) it can be concluded that:

$$\begin{cases} \mathbf{R}_{11}^T\mathbf{R}_{11} = \mathbf{R}_c + \mathbf{CL}[k^-]\mathbf{L}[k^-]^T\mathbf{C}^T = \mathbf{S} \\ \mathbf{R}_{12}^T\mathbf{R}_{11} = \mathbf{L}[k^-]\mathbf{L}[k^-]^T\mathbf{C}^T \\ \mathbf{R}_{12}^T\mathbf{R}_{12} + \mathbf{R}_{22}^T\mathbf{R}_{22} = \mathbf{L}[k^-]\mathbf{L}[k^-]^T \end{cases} \rightarrow \begin{cases} \mathbf{R}_{11}^T = \mathbf{S}^{1/2} \\ \mathbf{R}_{21}^T = \mathbf{L}[k^-]\mathbf{L}[k^-]^T\mathbf{C}^T\mathbf{S}^{-1/2} \\ \mathbf{R}_{22}^T = \mathbf{L}[k^+] \end{cases} \tag{4-22}$$

The last case will be elaborated more. When $\mathbf{R}_{12}$ equals $\mathbf{L}[k^-]\mathbf{L}[k^-]^T\mathbf{C}^T\mathbf{S}^{-1/2}$ then $\mathbf{R}_{12}^T\mathbf{R}_{12}^T$ is equal to $\mathbf{L}[k^-]\mathbf{L}[k^-]^T\mathbf{C}^T\mathbf{S}^{-1}\mathbf{CL}[k^-]\mathbf{L}[k^-]^T$. Bringing this to the other side of the equal sign in the last case of (4-22) gives: $\mathbf{R}_{22}^T\mathbf{R}_{22} = \mathbf{L}[k^-]\mathbf{L}[k^-]^T - \mathbf{L}[k^-]\mathbf{L}[k^-]^T\mathbf{C}^T\mathbf{S}^{-1}\mathbf{CL}[k^-]\mathbf{L}[k^-]^T$, which is identical to the measurement covariance update equation presented in (4-7). Hence $\mathbf{R}_{22}^T\mathbf{R}_{22}$ equals $\mathbf{L}[k^+]\mathbf{L}[k^+]^T$. This means that the QR decomposition of the matrix presented in (4-18) results in the measurement updated state covariance matrix.
Hence, the final equations of the measurement update are:

$$\begin{aligned} \hat{\mathbf{x}}[k^+] &= \hat{\mathbf{x}}[k^-] + \mathbf{L}[k^-]\mathbf{L}[k^-]^T\mathbf{C}^T\mathbf{S}^{-1}(\mathbf{y}[k] - \mathbf{C}\hat{\mathbf{x}}[k^-]) \\ \mathbf{L}[k^+] &= \mathbf{R}_{22}^T \end{aligned} \tag{4-23}$$

4-4-3 Modified Gram-Schmidt Orthogonalization

As seen in Section 4-4-1 and Section 4-4-2, there is the need for a procedure that creates an orthonormal matrix $\mathbf{T}$ that reshapes the matrices into square-root covariance matrix update. There are several ways to implement this strategy: Modified Gram-Schmidt (MGS), Householder transformation, or Givens rotation.
The Householder transformation computes a reflection of the column of the matrix n times in order to make a triangular matrix. The creation of this matrix requires a column vector that determines the subspace over which the reflection is made. It is in the composition of the vector that certain entries of the original matrix are needed. This is not convenient when working with Tensor-Train matrices (TTm) since the TTms have to be fully contracted. The Givens rotation creates a triangular matrix by multiplying the original matrix with rotation

matrices. Hence, for a large matrix many rotational matrices have to be composed and many matrix-matrix products have to be executed. As was seen in Section 3-4-3, a matrix-matrix product causes the TT-ranks to grow exponentially, resulting in high computational cost and demanding the TT-rounding function to be called relatively often. Hence, this thesis will continue by using MGS to find the triangular matrix $\mathbf{R}$. In this procedure, no single entries of the matrix are needed, nor are matrix-matrix or matrix-vector products. The TT-ranks grow due to vector subtraction and hence, the TT-rounding is called fewer times than when using Givens rotation.
The MGS orthogonalization will be explained and its use for the Tensor-Networked Square-Root Kalman Filter (TNSRKF) will be elaborated upon. What is known now as the Classical Gram-Schmidt orthogonalization was originally discovered by Laplace and appeared later as a solution to a set of linear equations in Schmidt's work. However, the Classical Gram-Schmidt orthogonalization suffers from a loss of orthogonality due to its inability to correct for errors. Hence, MGS is used because it is numerically more stable than the classical Gram-Schmidt and is less sensitive to round-off errors [2].

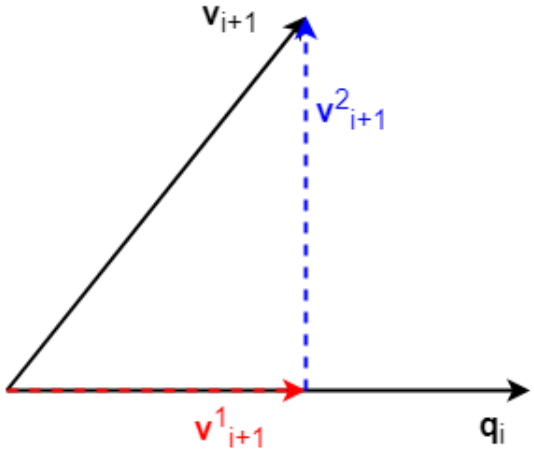

Figure 4-2: A graphical illustration of vector projection. The horizontal black arrow represents the vector $\mathbf{q}_i$ and the inclined black arrow represents the vector $\mathbf{v}_{i+1}$. The projection of $\mathbf{v}_{i+1}$ on $\mathbf{q}_i$ is indicated by the red striped arrow $\mathbf{v}^1_{i+1}$ and the blue, striped arrow indicates the orthogonal part of $\mathbf{v}_{i+1}$ on $\mathbf{q}_i$ and is denoted by $\mathbf{v}^2_{i+1}$. The updated $\mathcal{V}_{i+1}$ is equal to $\mathbf{v}^2_{i+1}$.

The MGS algorithm takes each of the columns of the matrix and creates an orthonormal subspace of these vectors by iteratively orthogonalizing the vector with respect to each other, making use of vector projection. Consider a vector $\mathbf{q}_i$ with $\|\mathbf{q}_i\|_2 = 1$ that is orthonormal to the set of unit vectors $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_{i-1}\}$. Consider also a vector $\mathbf{v}_{i+1}$ that is made orthogonal to $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_{i-1}\}$ but that is not yet orthogonal to $\mathbf{q}_i$. The next step is to make $\mathbf{v}_{i+1}$ orthogonal to $\mathbf{q}_i$. In Figure 4-2 one can see the vectors $\mathbf{q}_i$ and $\mathbf{v}_{i+1}$. The red, striped arrow is the projection of $\mathbf{v}_{i+1}$ onto $\mathbf{q}_i$ and is denoted by $\mathbf{v}^1_{i+1}$ and can be computed via (4-24). Note that $\mathbf{q}_i^T\mathbf{q}_i$ equals 1 and can be neglected, saving computational cost.

$$\mathbf{v}^1_{i+1} = \frac{\mathbf{q}_i^T\mathbf{v}_{i+1}}{\mathbf{q}_i^T\mathbf{q}_i}\mathbf{q}_i \tag{4-24}$$

Subtracting the projection $\mathbf{v}^1_{i+1}$ from $\mathbf{v}_{i+1}$ results in $\mathbf{v}^2_{i+1}$, which is orthogonal to $\mathbf{q}_i$. The MGS algorithm is presented by Algorithm 4. The algorithm is given for a general matrix

$\mathbf{A} \in \mathbb{R}^{M \times N}$ with arbitrary dimensions. For $M \leq N$, the algorithm gives M orthonormal vectors $\mathbf{q}_i \in \mathbb{R}^M$. For $M > N$ the algorithm could be stopped after the first N vectors, and will output N orthonormal vectors $\mathbf{q}_i \in \mathbb{R}^M$. The reason is that the other possible orthogonal vectors correspond to the zero part of $\mathbf{R}$, as is seen in (4-16). Therefore, it not needed to find this set of orthonormal vectors.

Algorithm 4: Modified Gram-Schmidt algorithm

Data: Matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$.
Result: Orthonormal set of vectors: $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_M\}$.
for *i = 1 to M* **do**
  $\mathbf{v}_i = \mathbf{A}(:, i)$
end
for *i = 1 to M* **do**
  $\mathbf{q}_i = \frac{\mathbf{v}_i}{\|\mathbf{v}_i\|_2}$
  for *j = i+1 to M* **do**
    $\mathbf{v}_j = \mathbf{v}_j - (\mathbf{q}_i^T \mathbf{v}_j)\mathbf{q}_i$
  end
end
Return $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_M\}$.

4-4-4 Partitioned Measurement Update

The method described in the previous section makes use of all the available measurements to update the state and the state covariance matrix at once. This method causes $\mathbf{S}$ to be a matrix with dimensions $\mathbb{R}^{L \times L}$, equal to the amount of observations. As is seen in (4-23), the inverse of $\mathbf{S}$ is computed. This is a relatively computationally expensive task, $\mathcal{O}(L^3)$, and can be avoided by using a Partitioned Update Kalman Filter (PUKF) as is done in [12] as well. The PUKF algorithm updates the state vector and state covariance matrix iteratively for each observation, implying that the measurement update is done L times and that the innovation covariance matrix is a scalar. Hence, taking the inverse of $\mathbf{S}$ is now a simple task. This change means that $\mathbf{C}$ is a row vector. The matrix in (4-18) has dimensions $\mathbb{R}^{1+N \times 1+N}$.

Chapter 5

Square-Root Kalman filter for Online Video Completion

This chapter merges the results of the previous two chapters by introducing the implementation of the Tensor-Networked Square-Root Kalman Filter (TNSRKF) for online video completion. According to [25], there are various ways of updating the Cholesky factor of the covariance matrix. As of now, the general thought that is proposed in Section 4-4 will be used to derive a TNSRKF. The outline of this chapter is the following. First, the state-space system that describes the video will be discussed. Second, the general approach for a Square-Root Kalman Filter (SRKF) as presented in Chapter 4 will be elaborated upon with the implementation of Tensor-Train (TT)-format as is introduced in Chapter 3. Finally, a detailed explanation of the implementation is given, concluding203. with a pseudo-algorithm.

5-1 State-Space System of a Video

This section presents the discrete time state-space system that is used to describe the video. The model and assumptions are mainly based on [12] and will be discussed here for comprehensiveness.
First of all, the video frames must be transformed into a vector. The pixels of the video are expressed as 8-bit integers on a gray-scale, see Figure 2-1. Where the integer value of 0 represents black pixels and 255 represents white pixels. Matlab works with double-precision arrays and thus the *double* function in Matlab is used to convert the matrix from a *uint8* class to a *double* class. The second step is to reshape the matrix into a vector by sequentially stacking the vectors as is seen in Figure 5-1. This gives a state vector where each of the pixels are represented by a state in the state-space system. Furthermore, the state transition matrix $\mathbf{A}$ is assumed to be identity by approximation. De Rooij's thesis [12] shows that difference between two consecutive frames throughout the whole video can be approximated by a zero mean Gaussian distribution. For this analysis the Grand Central Station video is used, out of which all the consecutive frames are analyzed against each other. By approximation it is

$$\mathbf{X}[k] = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,N} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{M,1} & x_{M,2} & \cdots & x_{M,N} \end{bmatrix} \quad \rightarrow \quad \mathbf{x}[k] = \left[\begin{array}{c} x_{1,1} \\ x_{2,1} \\ \vdots \\ x_{M,1} \\ \hline x_{1,2} \\ x_{2,2} \\ \vdots \\ x_{M,2} \\ \hline \vdots \\ \hline x_{1,N} \\ x_{2,N} \\ \vdots \\ x_{M,N} \end{array}\right]$$

Figure 5-1: This figure shows how to reshape a matrix into a vector by stacking its columns [12].

allowed to say that the mean of the data equals zero, implying that the next frame is equal to the next frame. Hence, the state-transition matrix is chosen to be the identity matrix. The difference between the pixels in consecutive frames is solely introduced by the process noise $\mathbf{w}$, which is statistically determined by (4-2). Hence, the process noise covariance matrix becomes an important factor on which the working principle of the Kalman filter relies. In Section 5-1-1 more information is given regarding the process noise covariance matrix.
Moreover, the output matrix $\mathbf{C}$ is a transformation matrix from the observations $\mathbf{y}$ to the state $\mathbf{x}$ and consists only of ones and zeros. No measurement noise is required, since direct and uncorrupted access to the batch of observed pixels is assumed. Finally, the input matrix $\mathbf{B}$ and the feedforward matrix $\mathbf{D}$ are zero, this is because there is no input in the video and hence, not in the state-space system.
All in all, the discrete state-space system that describes the dynamics of the video is:

$$\begin{aligned} \mathbf{x}[k+1] &= \mathbf{x}[k] + \mathbf{w}[k] \\ \mathbf{y}[k] &= \mathbf{C}[k]\mathbf{x}[k], \end{aligned} \tag{5-1}$$

where the process noise is statistically given by the initial part of (4-2).

5-1-1 Process Noise Covariance Matrix

Unless stated otherwise, this section is based on [12].
The process noise covariance matrix must represent the difference between two consecutive frames. This difference can be determined by investigating the correlation between the pixels in the frame. The correlation value between two pixels decreases when the distance between two pixels increases. In [12], a bandwidth equal to 15 is used for the Town Center video with a frame resolution of 480 by 720. This bandwidth is used to compose the process noise covariance matrix, and its value equals the distance between pixels when the correlation is less than 0.2. However, the bandwidth is dependent on the frame's resolution. As will be discussed in the next section, the resolution of the Town Center video is down scaled to 9 by 16. For this resolution the bandwidth equals 1, resulting in a process noise covariance matrix equal to the identity matrix. This also implies that the square-root process noise covariance

matrix is equal to the identity matrix. Because the columns of the original frame are stacked, neighboring pixels are not neighbors in the process noise covariance matrix. For this reason the process noise covariance matrix is created out of two smaller matrices: $\mathbf{Q}_1 \in \mathbb{R}^{J\times J}$ and $\mathbf{Q}_2 \in \mathbb{R}^{I\times I}$, see their structure in (5-2). The process noise covariance matrix must be positive definite, which can be ensured when $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are positive definite because the Kronecker product of two positive definite matrices results in a positive definite matrix [12, 5].

$$\mathbf{Q}_n = \begin{bmatrix} w(0) & w(1) & \dots & w(\alpha) & 0 & \dots & 0 \\ w(1) & w(0) & w(1) & \dots & w(\alpha) & \dots & 0 \\ & & \ddots & \ddots & \ddots & \ddots & \\ 0 & \dots & 0 & w(\alpha) & \dots & w(1) & w(0) \end{bmatrix}, \quad n = 1, 2 \tag{5-2}$$

where w is a function that denotes the correlation between the pixels and is defined in (5-3)

$$w(d) = \max\left(0, 1 - \frac{d}{\alpha + 1}\right), \tag{5-3}$$

where d is the distance and α is the bandwidth. The state space system and the noise covariance matrix are known and implemented into TT-format. The next sections implement the SRKF in TT-format.

5-1-2 Selection of Quantization

Before the TNSRKF algorithm is discussed, the quantization of the frame's row and column is defined. Due to the computational complexity of the algorithm, the video is down scaled from 1080 by 1920 to a resolution of 9 by 16. The partitioning of the row is chosen to be $[3, 3]$ and the partitioning of the column is chosen to be $[4, 2, 2]$. In general is it preferred to select low values for the partitioning in order to obtain the largest information compression rate in the TT, [12]. However, the downside of choosing low indices is that there are more cores, resulting in more calls of the Singular Value Decomposition (SVD) function. Therefore, the choice of the quantization is a trade-off between maximum compression rate and computational speed.

5-2 Modified Gram-Schmidt in TT-format

The application of Modified Gram-Schmidt (MGS) in TNSRKF follows the same structure as the MGS procedure in matrix format, which can be found in Section 4-4-3. In this section, the MGS algorithm will be applied on an arbitrary Tensor-Train matrix (TTm) $\mathcal{A} = \langle\langle \mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \dots, \mathcal{A}^{(d)} \rangle\rangle$ with $\mathcal{A}^{(k)} \in \mathbb{R}^{R_{k-1}, M_k, N_k, R_k}$. The procedure that is discussed is valid for the matrices stated in the time propagation (4-15) and in the measurement update (4-19). First, some functions that are needed for understanding the Tensor-Train Modified Gram-Schmidt (TT-MGS) algorithm are presented, after which Algorithm 5 is presented. Second, a detailed explanation of all the MGS steps in TT-format is given. Finally, a parameter choice analysis will be completed.

Multi-index

The *multi-index* function can be used to extract a specific TT out of a TTm or extend a TT into a TTm based on a given column partitioning: $[I_1, I_2, \ldots, I_d]$. The extraction process can be seen as taking a specific column out of a matrix. The extension process can be seen as extending a vector into a matrix by placing the vector in the correct column of the matrix. The set of unit vectors are given by: $\{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_d\}$. Consider an arbitrary TT: $\mathcal{A} = \langle\langle \mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(d)} \rangle\rangle$ with $\mathcal{A}^{(k)} \in \mathbb{R}^{R_{k-1}, I_k, R_k}$. Given the dimension of the TT, the dimensions of the unit vectors are given by: $\mathbf{e}_k \in \mathbb{R}^{I_k}$. The unit vectors all have a '1' placed at a specific location. This location is determined by the column dimension partitioning. Consider that the column dimension equals: $[I_1, I_2, I_3] = [2, 3, 2]$ and the 5^{th} column is selected. The indices can be determined via (3-2) in Section 3-2-1. Filling in (3-2) for the example results in:

$$5 = i_1 + (i_2 - 1)2 + (i_3 - 1)6.$$

Algorithm 13 in Appendix A-2-1 searches for the index in increments from the last index to the first index. For the last index i_3 the algorithm searched in increments of $I_1 I_2 = 6$. Hence, for the last index, the algorithm divides the column number 5 by $I_1 I_2 = 6$. It observes between which integer values the fraction is and selects i_3 to be equal to the upper bound. Hence, in this case i_3 is set to 1. The updated equation is:

$$5 = i_1 + (i_2 - 1)2 + (1 - 1)6 \rightarrow 5 = i_1 + (i_2 - 1)2.$$

The same procedure is followed for index i_2 and i_1. For i_2 the algorithm searches in increments of $I_1 = 2$, meaning that it checks between what integer value, the fraction of 5 and 2 is. Since this fraction is between 2 and 3, the second index i_2 equals 3, and the updated equation is:

$$5 = i_1 + (3 - 1)2 \rightarrow 1 = i_1.$$

This automatically gives the solution for the first index: $i_1 = 1$. Furthermore, the indices represent the 5^{th} vector and indicate the location of the '1' in the unit vectors. Hence, the unit vectors for this example are:

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \mathbf{e}_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \mathbf{e}_3 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Subtraction of TTs

The subtraction of two d-dimensional TTs can be performed by subtraction of the single cores of each of the TTs. Consider two TTs, $\mathcal{A}$ and $\mathcal{B}$, as defined in (5-4)

$$\mathcal{A} = \langle\langle \mathcal{A}^{(1)}(i_1), \mathcal{A}^{(2)}(i_2), \ldots, \mathcal{A}^{(d)}(i_d) \rangle\rangle, \quad \mathcal{B} = \langle\langle \mathcal{B}^{(1)}(i_1), \mathcal{B}^{(2)}(i_2), \ldots, \mathcal{B}^{(d)}(i_d) \rangle\rangle. \tag{5-4}$$

The subtraction of $\mathcal{A}$ and $\mathcal{B}$ result in another TT named $\mathcal{C}$. Moreover, the TT-cores of $\mathcal{C} = \mathcal{A} - \mathcal{B}$ are given in (5-5),

$$\mathcal{C}^{(1)}(i_1) = \begin{bmatrix} \mathcal{A}^{(1)}(i_1) & -\mathcal{B}^{(1)}(i_1) \end{bmatrix}, \quad \mathcal{C}^{(k)}(i_k) = \begin{bmatrix} \mathcal{A}^{(k)}(i_k) & 0 \\ 0 & \mathcal{B}^{(k)}(i_k) \end{bmatrix}, \quad \mathcal{C}^{(d)}(i_d) = \begin{bmatrix} \mathcal{A}^{(d)}(i_d) \\ \mathcal{B}^{(d)}(i_d) \end{bmatrix}. \tag{5-5}$$

Site-k orthogonal TT

This section quickly discusses the site-k orthogonalization procedure of a given TT. The function is given by using the SVD function to orthogonalize the TT-cores. This function aims to bring the norm of any TT to the specified k^{th} TT-cores. The *site-k* function also compresses the TT-ranks and can be used to compress the TT-ranks without performing truncation. See Algorithm 12 in Appendix A-1-5 for more information. The first nested

Algorithm 5: TT-MGS: Tensor-Train Modified Gram-Schmidt

Data: TTm $\mathcal{A} = \langle\langle \mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(d)} \rangle\rangle$ with $\mathcal{A}^{(k)} \in \mathbb{R}^{R_{k-1}, M_k, N_k, R_k}$. R_{thres} denotes the rank threshold, the maximum TT-rank that is allowed in a TT and ϵ is the truncation parameter.
Result: Orthonormal set of Tensor-Train vectors: $\{\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_m\}$
for *i = 1 to M* **do**
  Set of d unit vectors: $\mathbf{e} \leftarrow$ multi-index$(\mathbf{M}, i)$
  for *j = 1 to d* **do**
    $\mathcal{V}_i^{(j)} \leftarrow \text{reshape}(\text{permute}(\mathcal{A}^{(j)}, [1,2,4,3]), [R_{j-1} M_j R_j, N_j]) \times \mathbf{e}_j$
  end
end
for *i = 1 to M* **do**
  $\mathcal{Q}_i \leftarrow \mathcal{V}_i$
  $\mathcal{Q}_i \leftarrow \text{sitek}(\mathcal{Q}_i, 1)$
  $\text{norm}\mathcal{Q}_i \leftarrow \text{norm}(\text{reshape}(\mathcal{Q}_i^{(1)}, [R_0 M_1 R_1, 1]), \text{'fro'})$
  $\mathcal{Q}_i^{(1)} \leftarrow \mathcal{Q}_i^{(1)} / \text{norm}\mathcal{Q}_i$
  for *j = i+1 to M* **do**
    Temporal TT: $\mathcal{QQ} \leftarrow \mathcal{Q}_i$
    $\text{proj}\mathcal{V}_j\mathcal{Q}_i \leftarrow \text{innerprodTT}(\mathcal{V}_j, \mathcal{QQ})$
    $\mathcal{QQ}^{(1)} \leftarrow \text{proj}\mathcal{V}_j\mathcal{Q}_i \times \mathcal{QQ}^{(1)}$
    $\mathcal{V}_j \leftarrow \text{subTT}(\mathcal{V}_j, \mathcal{QQ})$
    if $\max(\mathbf{r}) \geq R_{thres}$ **then**
      $\mathcal{V}_j \leftarrow \text{roundTT}(\mathcal{V}_j, \epsilon)$
    end
  end
end
Return $\{\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_m\}$

for-loop in Algorithm 5 describes how all the columns of the TTm of $\mathcal{A}$ can be extracted iteratively. Given the partitioning of the columns of $\mathcal{A}$ and the desired column that is to be extracted, multiple unit vectors $\mathbf{e}_i$ can be formed. This is done via the *multi-index* function. These unit vectors are multiplied with the respective TT-cores of $\mathcal{A}$. Figure 5-2 displays how the contraction works. The unit vector $\mathbf{e}_i$ and $\mathcal{A}^{(i)}$ are contracted over the index, as is seen in (5-6). After contraction of the cores of $\mathcal{A}$ and $\mathbf{e}$, a TT will remain, which represents the selected vector.

$$\mathcal{A}^{(k)}(m_k) = \sum_{n_k} \mathcal{A}^{(k)}(m_k, n_k) \otimes \mathbf{e}_k(n_k) \quad \forall k. \tag{5-6}$$

The orthogonalization process starts, which is indicated by the second nested for-loop, as soon as all the columns are extracted. In order to normalize the vectors, the rank is brought to the first TT-core via the *site-k* function, as can be seen in Appendix A-1-5. By bringing the norm to the first TT-core, the *norm* function in Matlab can be used to compute the norm of the first norm. The complexity of this method equals $\mathcal{O}((d-1)R^3N) + \mathcal{O}(2RN)$ and is less computationally expensive than computing the norm via the *innerprodTT* function. The norm is used to normalize the TT by dividing the first TT-core with the norm.
Consider that the i^{th} vector was just normalized, and is indicated by $\mathcal{Q}_i$. The next step is to make $\mathcal{V}_{i+1}$ through $\mathcal{V}_m$ orthonormal to $\mathcal{Q}_i$. This is done by computing the inner product between $\mathcal{V}_{i+1}$ and $\mathcal{Q}_i$ and multiplying the result with the first TT-core of $\mathcal{QQ}$, which a substitute TTm for $\mathcal{Q}_i$. Furthermore, $\mathcal{QQ}$ is subtracted from $\mathcal{V}_{i+1}$ to make $\mathcal{V}_{i+1}$ orthogonal to $\mathcal{Q}_i$. This is realized by the *subTT* function, presented in Appendix A-1-2.

An important notion is the growth of the TT-ranks while performing MGS on a set of

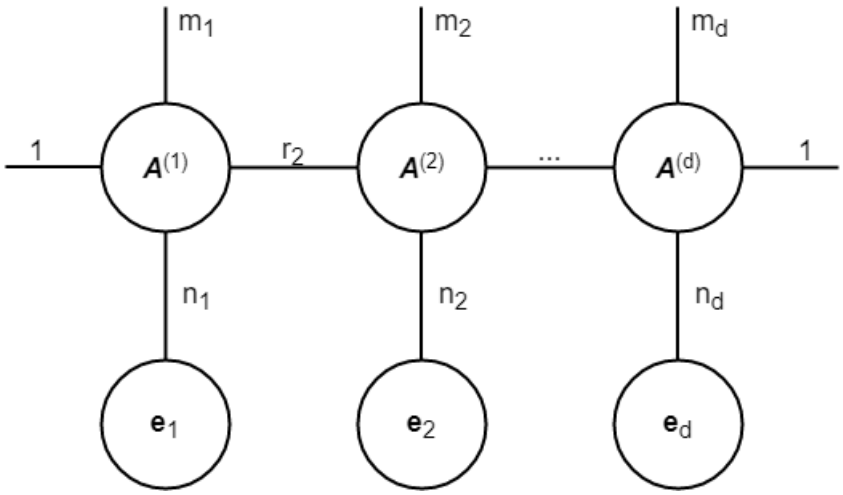

Figure 5-2: Multiplying each of the TT-cores with a unit vector to select a specific column from the TTm, resulting in a TT.

vectors. Each subtraction results in a summation of the TT-ranks of $\mathcal{V}_{i+1}$ and $\mathcal{Q}_i$. Hence, the TT-ranks of the updated TT vector $\mathcal{V}_{i+1}$ are equal to the sum of the TT-ranks of the current TT vector $\mathcal{V}_{i+1}$ and of the TT-ranks of the TT vector $\mathcal{Q}_i$. In order to prevent this growth from exceeding memory capabilities the *TT-rounding* function is utilized, as explained in Section 3-4-4. This process of activating and performing the truncation depends on two parameters: the rank threshold parameter R_{max}, and the truncation parameter ϵ which determines the maximum amount of information that is thrown away.
Now the influence of the two variables in the MGS algorithm on the orthogonality of the TTs will be analyzed. To do so, the orthogonality vectors are analyzed for a varying rank threshold parameter R_{max} and a varying truncation parameter ϵ. The aim of the MGS algorithm is to create a set of orthonormal TTs. Hence, the inner product between these vectors can be computed to analyze what error is introduced as a result of the truncation function. Note that the inner product between the same vectors is also taken into account, only the error is measured relative to 1. In Figure 5-3, Figure 5-4, Figure 5-5, and Figure 5-6, the orthogonality errors with respect to the identity matrix can be seen for a rank threshold of 5, 20, 50, and 90 respectively. The figures display the orthogonality error with respect to the truncation

parameter, which is located on the horizontal axis. Interestingly, the rank threshold does not affect the orthogonality error. This is because this parameter only affects the initialization of the truncation but has no direct or indirect influence on the truncation itself. However, it does influence the necessary time to complete the MGS algorithm. The running time of one orthogonalization procedure via Algorithm 5 took approximately 7.8 seconds for a rank threshold equal to 5, 8.9 seconds for a rank threshold of 20, 19.6 seconds for a rank threshold of 50 and 66.3 seconds for a rank threshold of 90. The difference in running time is significant. For a rank threshold of 5 and 20 seconds, the most burdensome part of the algorithm is the *TT-rounding* function. This is because of the SVD that is used to orthogonalize and truncate the TT. For a rank threshold of 50 the most computationally burdensome part is the computation of the inner product. The computational complexity of the inner product heavily depends on the TT-ranks, scaled with R^4. The same phenomena can be observed for a rank threshold of 90, where the computation of the inner product is approximately 90% of the running time. See Table 5-1 for a complete overview of the most burdensome parts of Algorithm 5's running time per child function. It can be seen that when the TT-ranks grow, the computational load of the inner product grows respectively to the other functions. Hence, it serves the algorithm well to choose a low rank threshold. Moreover, Figure 5-3 through

Rank threshold	5	20	50	90
subTT	20.1%	25%	16%	5.9%
roundTT	55.5%	39.5%	13%	4.7%
inner product	17.7%	29.4%	67.1%	87.9%

Table 5-1: This table provides insight in the running time of various child functions of Algorithm 5. Only the most computationally burdensome child functions are given.

Figure 5-6 show a direct relation between the truncation parameter and the orthogonality error, which is nearly identical for all the figures. Therefore, for this part of the algorithm, a rank threshold equal to 5 gives a satisfactory result. The truncation parameter is chosen such that the orthogonality error is minimal. Based on the figures, $\epsilon = 10^{-6}$ is a satisfactory choice. However, the orthogonalization procedure is not finished yet. The TTs still have to be concatenated to form an orthonormal TTm $\mathcal{Q}$. Therefore, this thesis will continue by diving deeper into the concatenation of the TTs in the next section.

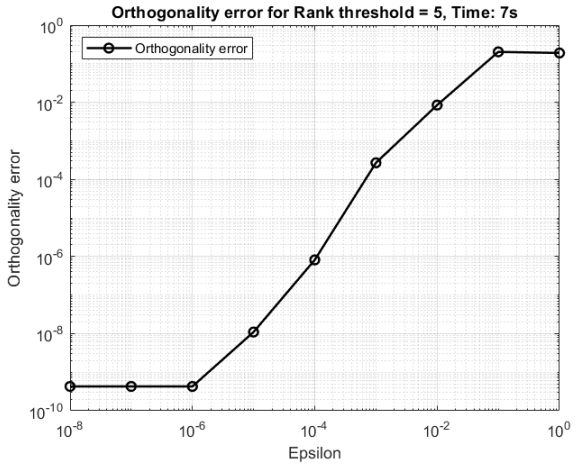

Figure 5-3: The orthogonality error of the MGS algorithm for a rank threshold value R_{max} equal to 5.

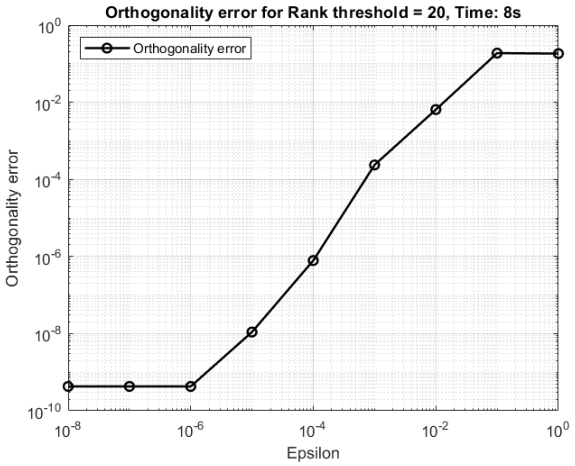

Figure 5-4: The orthogonality error of the MGS algorithm for a rank threshold value R_{max} equal to 20.

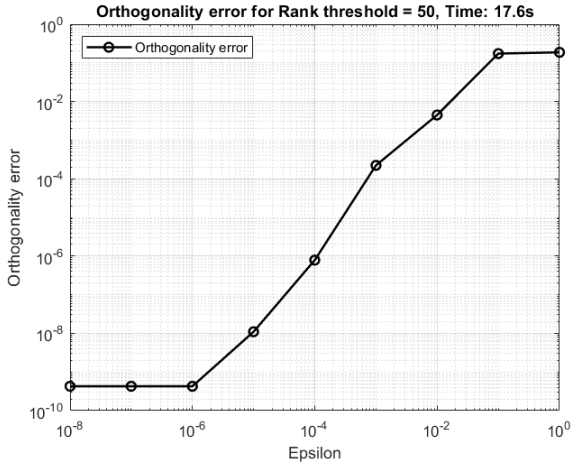

Figure 5-5: The orthogonality error of the MGS algorithm for a rank threshold value R_{max} equal to 50.

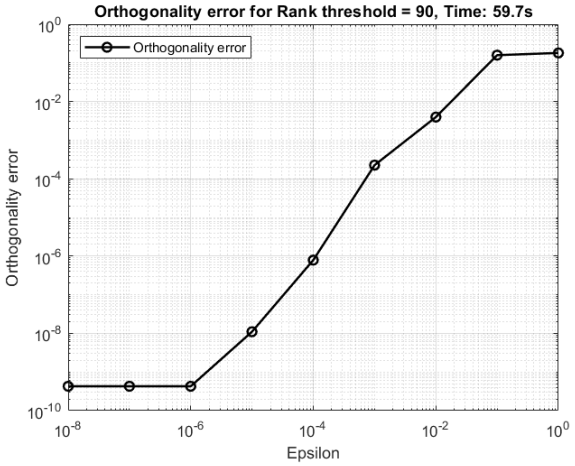

Figure 5-6: The orthogonality error of the MGS algorithm for a rank threshold value R_{max} equal to 90.

5-3 Combining TTs into one TTm

This section focuses on the concatenation of the set of TTs $\{\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_m\}$ into one TTm. For this part of the SRKF a specific algorithm, Algorithm 6, is made, which will be presented first, followed by a detailed explanation regarding the algorithm.

Algorithm 6: CombineTT: Combine Tensor-Trains into a Tensor-Train matrix.

Data: Given a set of orthonormal Tensor Trains: $\{\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_m\}$, where each of the TT is defined as: $\mathcal{Q}_i = \langle\langle \mathcal{Q}_i^{(1)}, \mathcal{Q}_i^{(2)}, \ldots, \mathcal{Q}_i^{(d)} \rangle\rangle$ with $\mathcal{Q}_i^{(k)} \in \mathbb{R}^{R_{k-1}, M_k, R_k}$. R_{thres} is the maximum allowed TT-rank. ϵ is the truncation parameter.
Result: Orthonormal Tensor-Train matrix: $\mathcal{Q}$
Extend the TTs into TTms:
for *i = 1 to prod(***M***)* **do**
  Set of d unit vectors: $\mathbf{e} \leftarrow$ multi-index$(\mathbf{m}, i)$
  for *j = 1 to d* **do**
    $\mathcal{Q}_i^{(j)} \leftarrow \text{reshape}(\mathcal{Q}_i^{(j)}, [R_{j-1} M_j R_j, 1]) \otimes \mathbf{e}_j$
    $\mathcal{Q}_i^{(j)} \leftarrow \text{reshape}(\mathcal{Q}_i^{(j)}, [R_{j-1}, M_j, R_j, M_j])$
    $\mathcal{Q}_i^{(j)} \leftarrow \text{permute}(\mathcal{Q}_i^{(j)}, [1, 2, 4, 3])$
  end
  $\mathcal{Q}_i \leftarrow \text{TTm2TT}(\mathcal{Q}_i)$
end
Addition of the TTms:
$\mathcal{QQ} \leftarrow \mathcal{Q}_1$
for *i = 2 to prod(***M***)* **do**
  $\mathcal{QQ} \leftarrow \text{addTT}(\mathcal{QQ}, \mathcal{Q}_i)$
  if $\max(\mathbf{r}) \geq R_{thres}$ **then**
    $\mathcal{QQ} \leftarrow \text{roundTT}(\mathcal{QQ}, \epsilon)$
  end
end
$\mathcal{Q} \leftarrow \text{TT2TTm}(\mathcal{QQ})$
Return: $\mathcal{Q}$

The algorithm starts by extending all the TTs to TTms. This is done in the nested for-loop via the *multi-index* function, which creates a set of d unit vectors: $\{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_d\}$. The unit vectors are multiplied with the respective TT-core by summing over the index 1, as seen in (5-7) and is visualized in Figure 5-7. Summation over an index that equals 1 is the Kronecker product between $\mathcal{Q}^{(i)}$ and $\mathbf{e}_i$. Hence, the unit vectors are used to place the information of the TT in the right column in the TTm. The next line introduces a new function, the *TTm2TT* function. The idea of this function is that the the rows and columns of the TTm are being brought together into one index and a TT is created. It is required to bring the extended TTms back to TTs because of the *addTT* function.

$$\mathcal{Q}^{(i)}(m_i, m_i) = \mathcal{Q}^{(i)}(m_i, 1) \otimes \mathbf{e}_i(1, m_i) \tag{5-7}$$

The final step of the algorithm is to add the TTms up in order to obtain one TTm. Again,

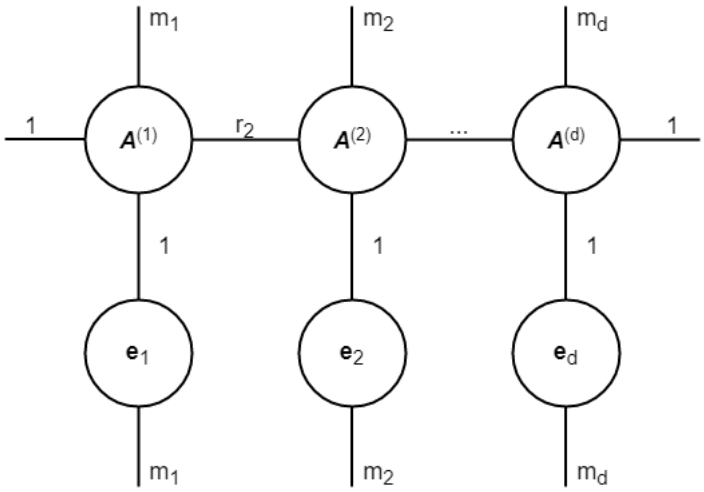

Figure 5-7: This figure shows how a TT can be extended to a TTm by multiplying each of the TT-cores with a unit vector.

when performing addition of the TT, the TT-ranks grow. Therefore, the *rounding* function is introduced to keep the TT-ranks small. This introduces two parameters that need to be tuned: the rank threshold R_{thres} and the truncation parameter ϵ. In Figure 5-8 the influence of the rank threshold and the truncation parameter is analyzed. Note that this analysis is a continuation of the previous section. Hence, the minimum orthogonality error that is possible is approximately 10^{-9}. What can be seen is that the error always reaches this minimum level when the epsilon is smaller than 10^{-4}. Furthermore, it can be stated that the rank threshold parameter has little influence on the orthogonality error. This does not only hold for the rank threshold shown in the plot but for all feasible rank threshold parameters. When allowing the ranks to grow up to approximately 150 the running time was empirically found to be the smallest. This is related to the slow TT-rank growth in the addition of TTs in combination with choosing a high rank threshold to decrease the amount of calls of the rounding function. However, selecting an over sized rank threshold will cause the rank to grow too large, resulting in a larger running time of the $addTT$ function. The orthogonality error is slightly larger for a rank threshold of 50 and slightly lower for a rank threshold of 250. A possible reason for this phenomena can be found in the rounding function. Initially, the rounding function orthogonalizes the TT via the *site-k* function. This function outputs a TT with the norm in the k^{th} core and with compressed ranks. Compressed ranks imply that if the ranks are larger than the full rank case, they are being brought back to the full rank state. Consider a TT with indices equal to $[9, 9, 16, 4, 4]$ with a current TT-rank equal to $[1, 54, 151, 64, 32, 1]$ and a full TT-rank state of $[1, 9, 81, 16, 4, 1]$. Employing the *site-k* function for k equal to d, meaning that the norm is brought to the last core, will result in the following TT-ranks: $[1, 9, 81, 64, 32, 1]$. This is because of the economical SVD, which causes the new right rank of the 1^{st} till the $d-1^{\text{th}}$ core to be equal to the minimum of current right TT-rank or the product of the current left TT-rank and the index: $R_k = \min\{R_k, R_{k-1}I_k\}$. Thereby, the TT is partially truncated without loss of information. The influence of the rank threshold is found in the fact that more information is stored in the TTm before the rounding function is called, which means that the compression is more effective and the information stored is more dense.

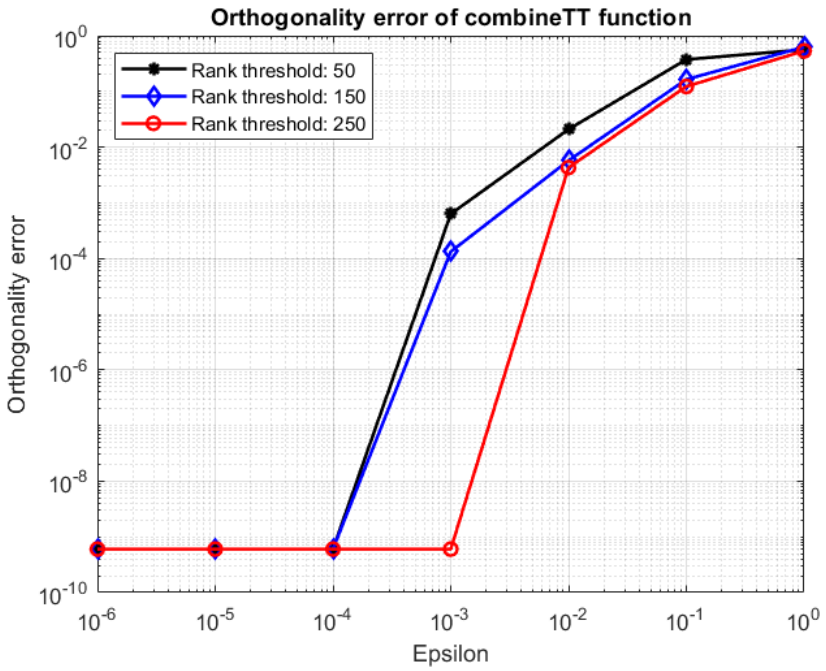

Figure 5-8: This figure shows the orthogonality error obtained after Algorithm 6 relative to the identity matrix. The rank threshold is set to: $\{50, 150, 250\}$, which are represented by the black, blue and red line respectively.

Finally, the rank threshold parameter equal to 150 is chosen because it was empirically found to result in the lowest running time. Furthermore, the truncation parameter is set to be 10^{-6}, because this results in the minimal orthogonality error. Even though the minimal orthogonality error is reached for $\epsilon = 10^{-4}$, due to a different structure of the square-root covariance the minimal orthogonality error may not be reached for this truncation parameter's value. Hence, a lower value is preferred to ensure orthogonality up to a higher degree. It must be noted that the TTm tends to grow to full TT-rank, which is determined by the index. Through investigation it is seen that even when the rank threshold is smaller than the individual maximum full TT-rank, the TT-rank is larger than the rank threshold after the addition of all the TTms. This is simply because the information that is stored cannot be thrown away. Therefore, the next section investigates whether or not is it possible to store an orthogonal matrix in a low-rank TTm.

5-4 Low-Rank Orthogonal TTm

This section discusses whether a low-rank orthogonality TTm is possible. This is investigated by performing simple experiments to check whether it is possible to truncate the ranks of an orthogonal TTm. One test will be done on the square-root covariance TTm of the previous section with full TT-ranks equal to: $[1, 9, 81, 16, 4, 1]$. Another test is done with a random matrix, which has the structure of a square-root covariance matrix but has larger dimensions than the dimensions used, namely 1024 by 1024. Truncation can be performed by specifying the maximum amount of information that can be thrown away or by selecting the desired

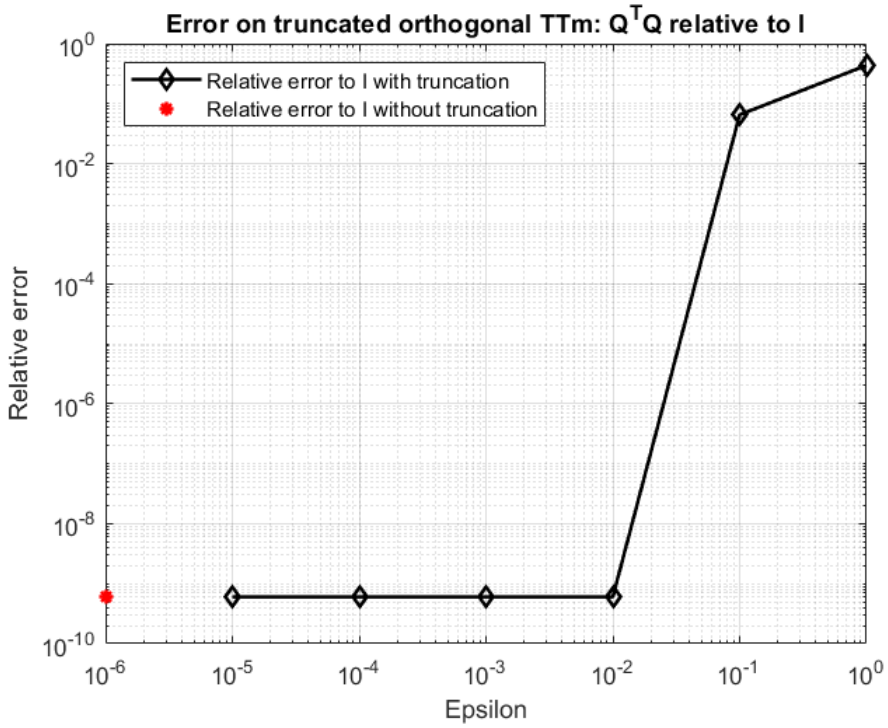

Figure 5-9: This figure shows the relative error of the product of the truncated TTm with its transpose $\mathbf{Q}^T\mathbf{Q}$ with respect to the identity matrix. The truncation parameter used in the rounding function, ϵ, is located on the horizontal axis. The black line shows the various errors obtained for truncating the TTm. The red asterisk indicates the relative error when no truncation is called. Hence, placing it at $\epsilon = 10^{-6}$ is not correct. It should be located at $\epsilon = 0$.

TT-ranks. Firstly, truncation based on the amount of information that is thrown away is analyzed. Secondly, truncation is analyzed by pre-selecting the desired TT-ranks.
In Figure 5-9 the relative error of the truncated orthonormal TTm multiplied by its transpose to the identity matrix is shown. As can be seen by the black diamonds, the relative error is measured for various increments of ϵ. The smallest relative error that can be obtained is equal to what is seen in Figure 5-8, which is approximately 10^{-9}. This relative error can be obtained for values of $\epsilon \leq 10^{-2}$. When ϵ equals 10^0 or 10^{-1} the relative error is significantly large and the orthonormal basis disappeared. The truncated TT-ranks corresponding to each of the values of ϵ are presented in Table 5-2. Notice that what is seen in Figure 5-9 is directly related to the TT-ranks presented in Table 5-2. For $\epsilon = 10^0$ and for $\epsilon = 10^{-1}$ truncation occurs, as can be seen by the lower TT-ranks in comparison to the TT-ranks of the untruncated TTm. For $\epsilon < 10^{-2}$ no truncation takes place, since the TT-ranks are identical to the TT-ranks of the untruncated TTm. This information concludes that a low-rank TTm representation of a orthonormal matrix is not possible. To strengthen this claim, observe Table 5-3. It is seen that when truncating any of the TT-ranks by reducing the dimension of each of the respective ranks with 1, a significant error arises. This means that all the information stored in the TT-cores is relevant to preserve the orthonormal basis. The magnitude of the relative errors, presented in the second column, are also worth mentioning: truncation of the largest TT-rank: 81 to 80, results in the smallest relative error and truncation of the smallest TT-rank: 4 to 3, results in the largest relative error. The other TT-ranks also correspond to this trend, with the

ϵ	TT-ranks
10^{0}	$[1, 1, 1, 1, 2, 1]$
10^{-1}	$[1, 9, 77, 16, 4, 1]$
10^{-2}	$[1, 9, 81, 16, 4, 1]$
10^{-3}	$[1, 9, 81, 16, 4, 1]$
10^{-4}	$[1, 9, 81, 16, 4, 1]$
10^{-5}	$[1, 9, 81, 16, 4, 1]$
0	$[1, 9, 81, 16, 4, 1]$

Table 5-2: This table shows the TT-ranks of the truncated TT for different values of the truncation parameter ϵ.

TT-ranks	Relative error
$[1, 8, 81, 16, 4, 1]$	0.1783
$[1, 9, 80, 16, 4, 1]$	0.0215
$[1, 9, 81, 15, 4, 1]$	0.0956
$[1, 9, 81, 12, 3, 1]$	0.2573

Table 5-3: This table shows the relative error of the truncated TT in comparison to the original TT.

truncation of the TT-rank equal to 16 resulting in the second smallest relative error and the truncation of the TT-rank of 9 resulting in the second largest relative error. This could be clarified by the fact that relatively more information is stored in smaller TT-ranks because of the higher compression rate than larger TT-ranks. To complement this section, it must be noted that this phenomena is valid for most orthogonal and orthonormal matrices. To further investigate this, multiple test are done on various matrices, such as random matrices with structures varying from completely random to lower triangular, to structures similar to the structure of the state square-root covariance matrix. These matrices are formed for different dimensions and in matrix format. Next, the orthogonal matrix of these matrices are computed in matrix format via the *orth* function in Matlab, which produces an orthogonal matrix with a tolerance based on the maximum singular value. The orthonormal matrix is converted into TT-format via the Tensor-Train Singular Value Decomposition (TT-SVD) function as presented in Algorithm 1. In this way one can already specify how much information the TT must contain, and thereby have an indirect influence on the TT-ranks. The truncation parameter is chosen such that all the information is kept, resulting in full TT-ranks.
An example will be provided, a matrix $\mathbf{A} \in \mathbb{R}^{1024\times1024}$ which has a structure similar to a lower triangular square-root covariance matrix and is equal to: $\mathbf{A} = \text{tril}(\text{rand}(1024)+\text{eye}(1024)*10)$. Moreover, let $\mathbf{Q} \in \mathbb{R}^{1024\times1024}$ be the orthonormal basis of $\mathbf{A}$. The matrix $\mathbf{Q}$ is converted to TT-format: $\mathcal{Q}$. The partitioning of the row and column of the orthogonal matrix are identical and their structure is such that the index of each core equals 2. Thus, there are 10 cores and the full TT-rank equals $[1, 4, 16, 64, 256, 1024, 256, 64, 16, 4, 1]$. The TTm of $\mathcal{Q}$ can be truncated for different truncation parameters $\mathcal{A}_{trun}(\epsilon)$ and brought back to matrix format $\mathbf{A}_{trun}(\epsilon)$ to compute: $\mathbf{A}_{trun}(\epsilon)^T\mathbf{A}_{trun}(\epsilon)$. Lastly, the error relative to the identity matrix is calculated and the TT-ranks of $\mathcal{A}_{trun}(\epsilon)$ are stored for later comparison. Figure 5-10 shows the relative error of $\mathbf{A}_{trun}(\epsilon)^T\mathbf{A}_{trun}(\epsilon)$ to the identity matrix. The black line shows the relative error of the truncated matrix, and the red star shows the relative error of the

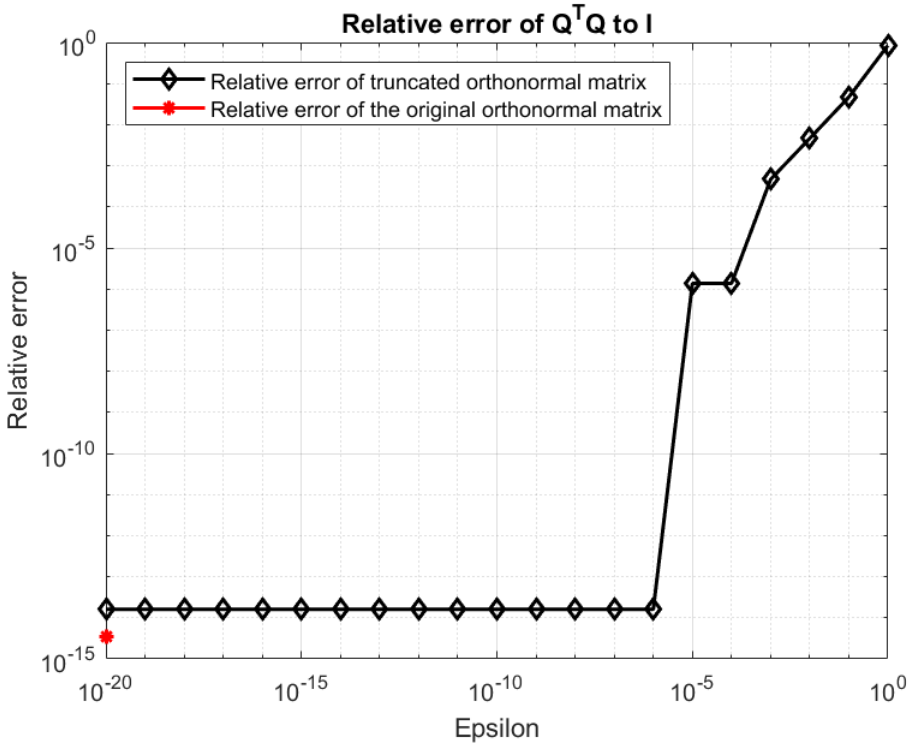

Figure 5-10: This figure shows the relative error of $\mathbf{Q}^T\mathbf{Q}$ to the identity matrix $\mathbf{I}$. The black line indicates the relative error of the truncated orthonormal matrix $\mathbf{Q}_{trun}$ and the red line indicates the relative error of the original orthonormal matrix $\mathbf{Q}$. The last one is placed at $\epsilon = 10^{-20}$ because $10^{-\inf}$ is not on the horizontal axis.

original orthonormal matrix. The figure clearly shows a large error when truncation takes place, as is seen for larger values of ϵ. To emphasize the general message from the earlier part of this section, see Table 5-4. Table 5-4 shows the truncated TT-ranks for the various truncation parameters used for the truncation. Next to the truncation parameter and the TT-ranks, the relative error of the respective truncation is also provided. The TT-ranks show that all information is kept for $\epsilon < 10^{-6}$. Furthermore, the TT-rank in the middle, r_5, is the only one truncated for values of ϵ between 10^{-5} and 10^{-1}. Only when ϵ equals 1, are almost all TT-ranks truncated. Thus, for $\epsilon < 10^{-1}$ the TT-ranks are always high, with the largest TT-rank above 908. This indicates that the TT-ranks are always close to full TT-rank and cannot be truncated while maintaining orthogonality. Thus, it can be concluded that a low-rank orthonormal TTm is not possible.

5-5 Implementation of the Tensor Network Square-Root Kalman Filter

This section combines and includes all the information presented in the previous sections. A complete overview of the TNSRKF will be given which will be finalized with the algorithm designed for this thesis work. The algorithm aims to complete a video with corrupted pixels

ϵ	TT-ranks	Relative error
10^{0}	$[1, 2, 11, 45, 170, 323, 150, 47, 14, 4, 1]$	0.8459
10^{-1}	$[1, 4, 16, 64, 256, 908, 256, 64, 14, 4, 1]$	0.0468
10^{-2}	$[1, 4, 16, 64, 256, 999, 256, 64, 14, 4, 1]$	0.0046
10^{-3}	$[1, 4, 16, 64, 256, 1018, 256, 64, 14, 4, 1]$	4.6538×10^{-4}
10^{-4}	$[1, 4, 16, 64, 256, 1023, 256, 64, 14, 4, 1]$	1.3706×10^{-6}
10^{-5}	$[1, 4, 16, 64, 256, 1023, 256, 64, 14, 4, 1]$	1.3706×10^{-6}
10^{-6}	$[1, 4, 16, 64, 256, 1024, 256, 64, 14, 4, 1]$	1.6058×10^{-14}

Table 5-4: This table shows the relative error of the truncated TT in comparison to the original TT. For values of ϵ smaller than 10^{-6} the TT-ranks and error are identical as for $\epsilon = 10^{-6}$, as is also seen in Figure 5-10.

in an online fashion using a combination of a SRKF and Tensor Networks (TN).
The general form of the SRKF in Section 4-4 is implemented into TT-format. In theory, the TNSRKF works the same as the general SRKF. However, the introduction of TTs have changed some approaches to solving specific steps in the algorithm. All the steps will be clearly explained in this section. To make this section clearer, it is divided into two subsections: the time propagation and the measurement update. In addition, the performance of the algorithm on the Town Center video is shown and a subsection is devoted to describing the computationally burdensome part of the algorithm.

5-5-1 Time Propagation in TT-format

The time propagation follows the update equations given in (4-11), where the next state is equal to the previous state and the state covariance matrix is updated by means of a transformation matrix $\mathcal{T}$. Such a transformation matrix must be found in order to make sure the update of the covariance matrix does not increase in dimension. In TT-format, the state update is simple. However, the update of the state covariance matrix requires more thought. First of all, the creation of the matrix $\begin{bmatrix}\mathcal{L}[k-1^{+}] & \mathcal{W}\end{bmatrix}$ can be done by concatenating the two matrices. The dimensions of $\mathcal{L}[k-1^{+}]$ and $\mathcal{W}$ are both $\mathbb{R}^{N_1 \times N_2 \times \cdots \times N_d \times N_1 \times N_2 \times \cdots \times N_d}$. This is important because when stacking the matrices next to each other, the row dimensions must be equal. However, in TT-format it is also required that the column dimensions are equal. Concatenation in TT-format is done by first extending the matrices separately with zero columns, as is done by the Kronecker product of the matrix and a vector $\mathbf{v} = [1\ 0]$. This process can be seen in Figure 5-11. The resulting TTm now represents $\begin{bmatrix}\mathcal{L}[k-1^{+}] & \mathbf{0}\end{bmatrix}$, following the same idea as the concatenation in Section 5-3. It must be pointed out that the same is done for $\mathcal{W}$, however $\mathbf{v}$ is equal to $[0\ 1]$. Thus $\mathcal{W}$ is extended to $\begin{bmatrix}\mathbf{0} & \mathcal{W}\end{bmatrix}$. To finalize the concatenation of $\mathcal{L}[k-1^{+}]$ and $\mathcal{W}$, the TTms must be added together, which is discussed in Section 3-4-1.
Following the concatenation, the TTm is transposed. This is done to constrict the procedure to computing the correct orthonormal TTm $\mathcal{Q}$ in the QR-decomposition. Because of the growth of the ranks and due to the addition of two TTm's, the *site-k* function is placed to truncate the ranks without throwing away information. After this process the concatenated matrix is created and the QR orthogonalization procedure can be initialized. The QR de-

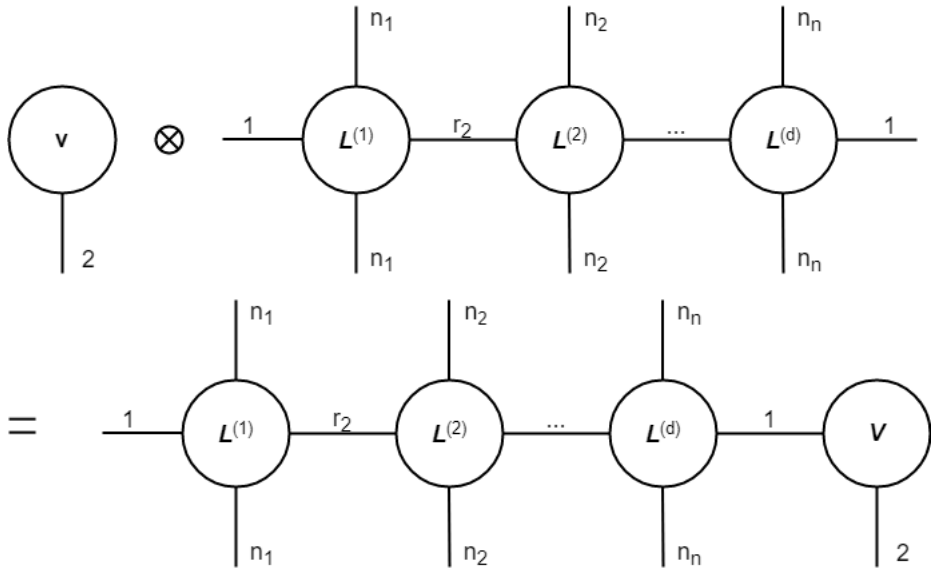

Figure 5-11: This figure shows the TT Kronecker product of a vector $\mathbf{v}$ and the TTm $\mathcal{L}$. As can be seen the vector is attached the last core of $\mathcal{L}$, this is because of little endian ordering of the indices.

composition of the concatenated matrix is discussed in Section 5-2. This procedure outputs n orthonormal TTs, $\mathcal{Q}_k \in \mathbb{R}^{N_1 \times N_2 \times \cdots \times 2N_d} \quad \forall k = 1, 2, \ldots, N$. To complete the time propagation of the TNSRKF, it is necessary to combine the orthonormal Tensor Train vectors: $\{\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_n\}$ into one Tensor Train matrix named $\mathcal{Q}$, as is explained in Section 5-3. When the TTm $\mathcal{Q}$ is finalized $\mathcal{R}$ can be computed, which is the final step of the time propagation. Because only the first N TTs are computed and placed in $\mathcal{Q}$, $\mathcal{R}$ is immediately the desired part of the matrix, denoted by $\mathbf{R}_1$ in Section 4-4-1. Hence, the time propagated update of the square-root covariance matrix equals $\mathcal{L}[k^-] = \mathcal{R}$.

5-5-2 Measurement Update in TT-format

The measurement update used in TNSRKF is the partitioned measurement update as described in Section 4-4-4. A result of using the partitioned measurement update is that the output matrix $\mathcal{C}$ becomes a row vector. Hence, the correct row of the TTm of $\mathcal{C}$ corresponding to the measurement must be selected. This is realized by generating d unit vectors based on the quantization of the rows of $\mathcal{C}$ and multiplying these with the corresponding cores of $\mathcal{C}$. The next step is the computation of the state update based on the measurement that is just observed, as is identified by the first equation of (4-23). This can be done in steps: computing the residual vector, the residual covariance matrix which is a scalar, the Kalman gain, and the state update. Furthermore, the update of the square-root covariance matrix is equal to $\mathbf{R}_{22}^T$ as seen in (4-23). Therefore, the matrix $\begin{bmatrix} \mathbf{R}_c^{1/2} & \mathbf{CL}[k^-] \\ \mathbf{0} & \mathbf{L}[k^-] \end{bmatrix}$ must be concatenated and transposed. However, direct access to the measurements is assumed, which results in $\mathbf{R}_c^{1/2}$ equaling 0. This matrix will thus be a matrix with one zero column. For the QR-decomposition of the matrix, this column does not need to be included to find the orthonormal basis of the matrix.

Hence, the matrix that is given to the TT-MGS algorithm is,

$$\begin{bmatrix} \mathbf{CL}[k^-] \\ \mathbf{L}[k^-] \end{bmatrix}^T. \tag{5-8}$$

A problem could arise with the concatenation of this matrix because the rows of $\mathbf{CL}[k^-]$ are not equal to the rows of $\mathbf{L}[k^-]$. However, one way to solve this is by not performing the concatenation. The TT of $\mathbf{CL}[k^-]$ and the TTm of $\mathbf{L}[k^-]$ can be individually presented to the TT-MGS algorithm and the first vector of the orthonormal basis of the matrix is simply the normalized TT of $\mathbf{CL}[k^-]$. Furthermore, the algorithm of TT-MGS extracts the columns out of $\mathbf{L}[k^-]$ to create a TT for each of the columns by using the *multi-index* function as was described before for identical processes. Thus, the matrix dimension of the matrix in (5-8) is $\mathbb{R}^{N+1\times N}$. Because this thesis computes $\mathbf{QR}$, and not $\mathbf{LQ}^T$, the matrix must transposed as was explained in Section 4-4-2. As the matrix is transposed, the orthonormal basis is $\mathcal{Q} \in \mathbb{R}^{N\times N}$. In order to compute $\mathcal{R}$, the TTm of $\mathcal{Q}$ and of the transpose of the matrix in 5-8 have to be composed. However, the dimension of $N+1$ cannot be partitioned over d cores. Hence, after having computed $\mathcal{Q}$, the columns of $\begin{bmatrix} \mathbf{CL}[k^-] \\ \mathbf{L}[k^-] \end{bmatrix}^T$ have to be complemented by $N-1$ zeros columns, see (5-9). The complemented square-root matrix now is $\mathbb{R}^{N_1\times N_2\times\cdots\times N_d\times N_1\times N_2\times\cdots\times N_d}$, which can be partitioned over d cores by a row partitioning of $[N_1, N_2, \ldots, N_d]$ and a column partitioning of $[N_1, N_2, \ldots, 2N_d]$.

$$\mathbf{L}[k^-] = \begin{bmatrix} \mathbf{CL}[k^-] \\ \mathbf{L}[k^-] \\ \mathbf{0} \end{bmatrix}^T \in \mathbb{R}^{N\times 2N} \tag{5-9}$$

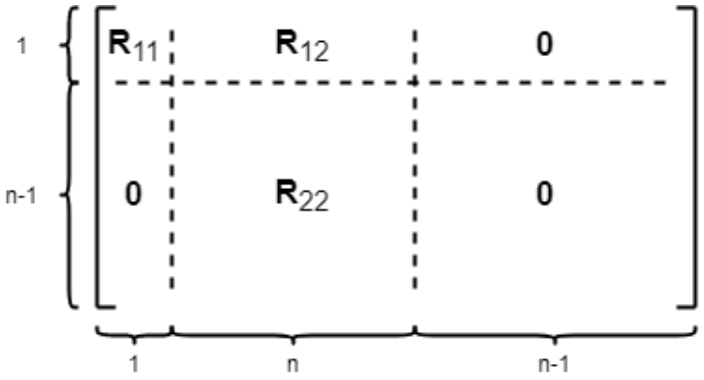

Figure 5-12: This figure presents the structure of the matrix $\mathbf{R}$.

The TTm of this matrix can be built up by first collecting all the columns as TTs and extending them into TTm's. The individual columns are already known, since $\mathbf{CL}[k^-]$ is known and the individual columns of $\mathbf{L}[k^-]$ are already extracted in the TT-MGS algorithm. Furthermore, the zero columns are TTs with TT-ranks equal to 1 and are filled with zeros. By extending these TTs into TTms and then adding to the correct square-root covariance matrix, $\mathcal{L}[k^-]$ according to (5-9) is created in TT-format. As a result of this, $\mathcal{R}$ can be computed out of the product of the transposed TTm of the orthonormal basis $\mathcal{Q}^T$ and $\mathcal{L}[k^-]$. The matrix

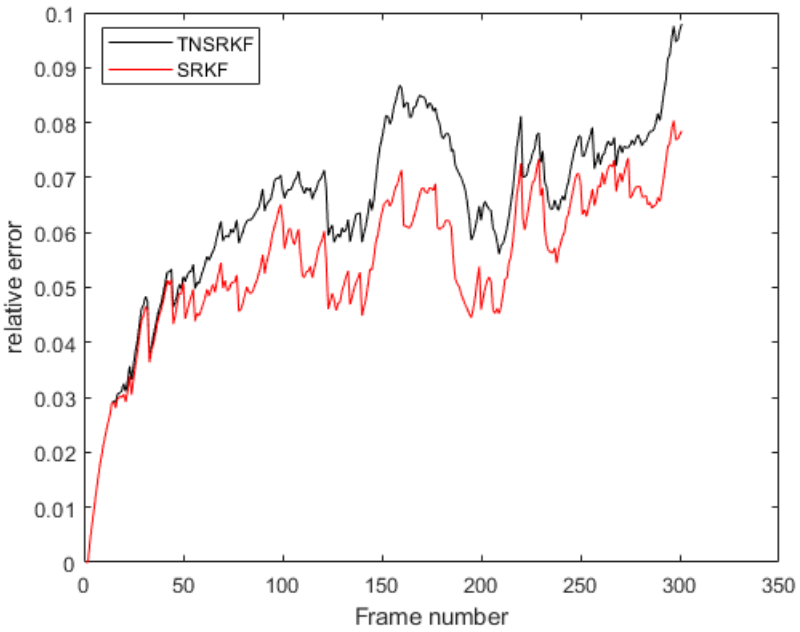

Figure 5-13: This figure shows the relative error per estimated frame. The figure consists of two error plots: the black line represents the relative error of the TNSRKF and the red line represents the relative error of the SRKF in matrix format. The video that is used for this simulation is the Town Center video [1]. Furthermore, 95% of the pixels are corrupted and the video is gray-scaled, and has a resolution of 9 by 16.

structure of $\mathbf{R}$ is presented in Figure 5-12.
The final step of the measurement update is to obtain $\mathbf{R}_{22}$ from the matrix structure. This is done by subtracting the 2^{nd} through the $n+1^{\text{th}}$ vector out of the $\mathcal{R}$, such that $\mathbf{R}_{12}$ and $\mathbf{R}_{22}$ are obtained. Furthermore, these vectors must be concatenated to a TTm and transposed. Out of the transposed TTm, the 2^{nd} through the n^{nd} columns are subtracted and one zero column is added to the set of columns. Hence, now $\begin{bmatrix}\mathbf{R}_{22} & \mathbf{0}\end{bmatrix}$ is obtained. The last column is equal to zero because the first column of the matrix in (4-19). However, because this zero column is left out before starting the computation, the zero column has to be added to $\mathbf{R}_{22}$. In addition, multiple other TTm operations such as, matrix-matrix multiplication in TT-format, are performed to compute the update of square-root covariance matrix in the partitioned measurement update. Because these operations increase the TT-ranks, the rounding function is called to truncate the TT-ranks. The truncation parameter is set to be 10^{-6}, equaling the truncation parameters for Algorithm 5 in Section 5-2 and Algorithm 6 in Section 5-3. In Figure 5-13 the relative error of the TNSRKF and its matrix form equivalent can be seen. The video on which this simulation is done is the Town Center Video from [1]. The video is originally in Full HD resolution, but is down scaled to a resolution of 9 by 16. Furthermore, the video is in gray scale. For this simulation, 95% of the pixels are corrupted. This is done randomly for each frame. As can be seen, the relative error is initially equal to zero, because the first frame is assumed to be known. Hence, from there on the relative error increases until it hovers within a certain margin of error, namely between 0.04 and 0.09. Only at the end of the simulation does the error make a sudden jump. This could be due to a item entering

the screen that is only observed in the measurements and accounted for in the state estimate at a later stage. A major downside of the algorithm is its high computational complexity, especially on computing the orthonormal basis TTm $\mathcal{Q}$. The total estimation of 300 frames, each having a resolution of 9 by 16, took approximately 8.3 hours, equaling an estimation time per frame of 100 seconds. Meanwhile, the complete estimation of the Town Center video took the SRKF only 100 seconds, resulting in a estimation time of 0.33 seconds per frame. Since, the frame dimensions of the Town Center video in [12] are not down-scaled as much as in this thesis, the running times cannot be compared directly. However, it must be stated that for a resolution equal to 480 by 720, the estimation of one frame via Adaptive Singular Value Thresholding (ASVT) and Tensor-Networked Kalman Filter (TNKF) takes 0.568 seconds and 38 seconds respectively. The difference in estimation time per frame is significant, especially when identical frame resolutions are considered. When larger dimensions are chosen, the storage complexity would quickly slow the SRKF down until it exceeds RAM capabilities. Note that these simulations are carried out on an Intel Core i7-6700HQ @2.60GHz with 8GB of RAM. In the next section, a deeper analysis and a critical view on the computational complexity of the algorithm will be presented.
All in all, 5-5 can be summarized by Algorithm 7, showing all the steps that are taken in order to complete one time propagation step and one measurement update of the TNSRKF.

5-5-3 Bottleneck of the Algorithm

This section discusses the bottleneck of the algorithm by analyzing and pointing out the computationally expensive steps.
One of the major drawbacks of the TNSRKF is the way in which the orthonormal basis TTm $\mathcal{Q}$ is computed. Consider a random matrix with dimensions: $\mathbb{R}^{N\times N}$. This requires $\sum_{i=1}^{N}(i-1)$ vector to vector orthogonalization procedures, which individually have to compute the inner product between $\mathcal{V}_j$ and $\mathcal{Q}\mathcal{Q}$, subtract $\mathcal{Q}\mathcal{Q}$ from $\mathcal{V}_j$ and perform rounding on $\mathcal{V}_j$. Consider that the cores of $\mathcal{V}_j$ are defined as: $\mathcal{V}_j^{(k)} \in \mathbb{R}^{R_{k-1}\times I_k \times R_k}$ and the cores of $\mathcal{Q}\mathcal{Q}$ are defined as: $\mathcal{Q}\mathcal{Q}^{(k)} \in \mathbb{R}^{S_{k-1}\times I_k \times S_k}$. The computational complexities of the inner product and the rounding function are $\mathcal{O}((d-2)I(RS)^2)$ and $\mathcal{O}(dIR^3)$ respectively, where $R = \max(R_k)$, $S = \max(S_k)$ and $I = \max(I_k)$. Over the course of one call of the TT-MGS function this function is called $\sum_{i=1}^{N}(i-1)$ times. For a video resolution of 9 by 16 with 5% pixel observation, this boils down to approximately 10296 calls.
Another disadvantage in terms of computational speed is the implementation of the partitioned measurement update. This requires L repetitions of the complete measurement update procedure, which includes one call of the TT-MGS function. Hence, expanding the amount of calls of the inner product to include the estimation of one frame, the inner product function is called on average 82368 times.
The most burdensome part of the algorithm is the rounding function. On average, this frame is called 86474 times for the estimation of one frame, costing approximately 61% of the estimation time.
Finally, this thesis only discussed the case in which 95% of the pixels are corrupted. In [12] a case in which 75% of the pixels are corrupted is also discussed, resulting is more measurement data and more iterations of the measurement update. As a result, TNSRKF's computational speed will decline drastically, therefore removing it from consideration.

Algorithm 7: TNSRKF

Data: State vector: $\mathcal{X}[k-1] = \langle\langle \mathcal{X}^{(1)}, \mathcal{X}^{(2)}, \ldots, \mathcal{X}^{(d)} \rangle\rangle$ with $\mathcal{X}^{(k)} \in \mathbb{R}^{R_{k-1}, N_k, R_k}$. Square-root covariance matrix: $\mathcal{L}[k-1] = \langle\langle \mathcal{L}^{(1)}, \mathcal{L}^{(2)}, \ldots, \mathcal{L}^{(d)} \rangle\rangle$ with $\mathcal{L}^{(k)} \in \mathbb{R}^{S_{k-1}, N_k, N_k, S_k}$. Square-root process noise matrix: $\mathcal{W}[k-1] = \langle\langle \mathcal{W}^{(1)}, \mathcal{W}^{(2)}, \ldots, \mathcal{W}^{(d)} \rangle\rangle$ with $\mathcal{W}^{(k)} \in \mathbb{R}^{Q_{k-1}, N_k, N_k, Q_k}$. Output matrix: $\mathcal{C}[k-1] = \langle\langle \mathcal{C}^{(1)}, \mathcal{C}^{(2)}, \ldots, \mathcal{C}^{(d)} \rangle\rangle$ with $\mathcal{W}^{(k)} \in \mathbb{R}^{P_{k-1}, N_k, N_k, P_k}$. Measurements $\mathbf{y} \in \mathbb{R}^L$ and location of measurements: $\mathbf{c} \in \mathbb{R}^L$, both are not in TT-format.

Result: $\mathcal{X}[k]$, $\mathcal{L}[k]$

Time propagation:

$\mathcal{L}[k] \leftarrow [\mathcal{L}[k-1], \mathcal{W}]$

$\mathcal{L}[k] \leftarrow \text{TransposeTTm}(\mathcal{L}[k])$

$\{\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_n\} \leftarrow \text{TT-MGS}(\mathcal{L}[k])$

$\mathcal{Q} \leftarrow \text{CombineTT}(\{\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_n\}, [N_1, N_2, \ldots, N_d])$

$\mathcal{L}[k] \leftarrow \text{MatMatTT}(\text{TransposeTTm}(\mathcal{Q}), \mathcal{L}[k])$

Measurement update:

for *i = 1 to l* **do**

  Set of d unit vectors: $\mathbf{e} \leftarrow \text{multi-index}(\mathbf{n}, c_j)$

  for *j = 1 to d* **do**

    $\mathcal{C}_i^{(j)} \leftarrow \text{reshape}(\text{permute}(\mathcal{C}^{(j)}, [1,2,4,3]), [P_{j-1} N_j P_j, N_j]) \times \mathbf{e}_j$

  end

  $\mathbf{v} \leftarrow \mathbf{y}_i - \text{innerprodTT}(\mathcal{C}_i, \mathcal{X}[k])$

  $s \leftarrow \text{innerprodTT}(\mathcal{C}_i, \text{MatVecTT}(\mathcal{L}[k], \text{MatVecTT}(\text{TransposeTTm}(\mathcal{L}[k], \mathcal{C}_i)))$

  $\mathcal{K} \leftarrow \text{MatVecTT}(\mathcal{L}[k], \text{MatVecTT}(\text{TransposeTTm}(\mathcal{L}[k]), \mathcal{C}_i))$

  $\mathcal{K}^{(1)} \leftarrow \frac{v}{s}\mathcal{K}^{(1)}$

  $\mathcal{X}[k] \leftarrow \text{addTT}(\mathcal{X}[k], \mathcal{K})$

  $\{\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_n\} \leftarrow \text{TT-MGS}(\begin{bmatrix} \text{MatVecTT}(\mathcal{L}[k], \mathcal{C}_i) \\ \mathcal{L}[k] \end{bmatrix}^T)$

  $\mathcal{Q} \leftarrow \text{CombineTT}(\{\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_n\}, [N_1, N_2, \ldots, N_d])$

  $\mathcal{L}[k] \leftarrow \text{CombineTT}(\{\text{MatVecTT}(\mathcal{L}[k], \mathcal{C}_i), \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_n, \}, [N_1, N_2, \ldots, 2N_d])$

  $\mathcal{R} \leftarrow \text{MatMatTT}(\text{TransposeTTm}(\mathcal{Q}), \mathcal{L}[k])$

  $\mathcal{L}[k] \leftarrow \begin{bmatrix} \mathcal{R}_{22} & \mathbf{0} \end{bmatrix}$

end

Return: $\mathcal{X}_{k+1}$, $\mathcal{L}_{k+1}$

Chapter 6

Conclusion and Future Work

This thesis investigated the application of the Tensor-Networked Square-Root Kalman Filter (TNSRKF) for online video completion. Starting with the definition of a Tensor-Train (TT), the thesis continued by applying a square-root Kalman filter to a state-space system. Chapter 5 presents the TNSRKF, a way to combine the TT decomposition from Chapter 3 and the Square-Root Kalman Filter (SRKF) from Chapter 4 into an online video completion algorithm. The following responses are this thesis' answers to the research questions posed in Chapter 1.

- *How can TNSRKF be implemented to recover a corrupted video?*
  All in all, this thesis shows that a general form of the SRKF can be implemented in TT-format and that the algorithm is able to complete a video with a relative error around 0.04 to 0.09. However, the implementation is slower than Adaptive Singular Value Thresholding (ASVT) and Tensor-Networked Kalman Filter (TNKF). Because of the high computational complexity of the Tensor-Train Modified Gram-Schmidt (TT-MGS) algorithm and the unobtainable low-rank orthonormal Tensor-Train matrix (TTm), it becomes clear that it is impossible to perform online video completion via the TNSRKF using a Modified Gram-Schmidt (MGS) orthogonality procedure.

- *How can the SRKF be implemented in TT-format?*
  The SRKF can be implemented in its general format, in which the update of the covariance matrix is found via the QR-decomposition. To obtain the orthogonal matrix, the MGS orthogonalization procedure is implemented. In TT-format, this can be implemented by using the TT operations presented in Chapter 3.

- *How can the MGS orthogonalization procedure be implemented in TT-format and how can orthogonality be preserved?*
  MGS can be implemented in TT-format, following the same steps as the MGS algorithm in matrix format. The TT operations are provided in Chapter 3 and provide an intuitive tool for the orthogonalization procedure of large vectors. Furthermore, TT-MGS

is proven to produce orthonormal vectors for certain rank threshold conditions and truncation parameters.

- *Is it possible to represent an orthogonal matrix as a low-rank TTm?*
  A low-rank orthonormal TTm is not realizable. Any truncation of the TT-ranks result in a large relative error, indicating that important orthogonality information is stored in the complete range of the TT-ranks.

Recommendations to speed up calculations

A significant improvement in the algorithm's calculation speed can be obtained by the parallelization of the orthogonalization procedure in the TT-MGS algorithm. The computing time can be reduced significantly, depending on the amount of threads the computer at hand has. Parallelization is possible because the orthogonalization of vector $\mathbf{a}$ to vector $\mathbf{c}$ is independent to the orthogonalization procedure of vector $\mathbf{b}$ to vector $\mathbf{c}$. Matlab's *Parallel Computing Toolbox* can be used for this.
Another computationally burdensome part of the algorithm is the implementation of the partitioned measurement update. By implementing the partitioned measurement update, the computation of the inverse can be bypassed. This is desired since no proven technique for computing the inverse in TT-format is available, implying that the inverse must be computed in matrix format, requires a conversion between matrix and TT-format. However, a trade-off could be found between the conventional measurement update and the partitioned measurement update could be found by considering a batch-based measurement update.

Recommendations on orthogonal matrix computation methods

In this thesis, the MGS procedure is taken to compute the orthonormal basis matrix in TT-format named $\mathcal{Q}$. However, the number of orthogonalization procedures between individual vectors increases drastically with the dimension of the state square-root covariance matrix. Therefore, scaling this procedure to higher resolution videos is not feasible. Other techniques, such as the Givens rotation and the Householder transformation will most likely give an identical problem. One option would be to use an alternating least-squares procedure in which the cores of the square-root covariance TTm are updated according to an objective function.

Appendix A

Algorithms

A-1 TT Operations

A-1-1 TT Addition

Algorithm 8: $addTT$: Addition of TTs [29]

Data: Given two Tensor-Train (TT)s: $\mathcal{A} = \langle\langle \mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(d)} \rangle\rangle$ with $\mathcal{A}^{(k)} \in \mathbb{R}^{R_{k-1}, I_k, R_k}$ and $\mathcal{B} = \langle\langle \mathcal{B}^{(1)}, \mathcal{B}^{(2)}, \ldots, \mathcal{B}^{(d)} \rangle\rangle$ with $\mathcal{B}^{(k)} \in \mathbb{R}^{S_{k-1}, I_k, S_k}$.

Result: $\mathcal{C} = \langle\langle \mathcal{C}^{(1)}, \mathcal{C}^{(2)}, \ldots, \mathcal{C}^{(d)} \rangle\rangle$ with $\mathcal{C}^{(k)} \in \mathbb{R}^{R_{k-1}S_{k-1}, I_k, R_k S_k}$ equaling the sum of $\mathcal{A}$ and $\mathcal{B}$.

$\mathbf{A}^{(1)} \leftarrow \text{reshape}(\mathcal{A}^{(1)}, [1, I_1 R_1])$

$\mathbf{B}^{(1)} \leftarrow \text{reshape}(\mathcal{B}^{(1)}, [1, I_1 S_1])$

$\mathcal{C}^{(1)} \leftarrow \text{reshape}\left(\begin{bmatrix} \mathbf{A}^{(1)} & \mathbf{B}^{(1)} \end{bmatrix}, [1, I_1, R_1 + S_1]\right)$

for *k = 2 to d-1* **do**

$\mathbf{A}^{(k)} \leftarrow \text{reshape}(\mathcal{A}^{(k)}, [R_{k-1}, I_k R_k])$

$\mathbf{B}^{(k)} \leftarrow \text{reshape}(\mathcal{B}^{(k)}, [S_{k-1}, I_k S_k])$

$\mathbf{C}^{(k)} \leftarrow \begin{bmatrix} \mathcal{A}^{(k)} & \mathbf{0} \\ \mathbf{0} & \mathcal{B}^{(k)} \end{bmatrix}$

$\mathcal{C}^{(k)} \leftarrow \text{reshape}\left(\mathbf{C}^{(k)}, [R_{k-1} + S_{k-1}, I_k, R_k + S_k]\right)$

end

$\mathbf{A}^{(d)} \leftarrow \text{reshape}(\mathcal{A}^{(d)}, [R_{d-1}, I_1])$

$\mathbf{B}^{(d)} \leftarrow \text{reshape}(\mathcal{B}^{(d)}, [S_{d-1}, I_1])$

$\mathcal{C}^{(d)} \leftarrow \text{reshape}\left(\begin{bmatrix} \mathbf{A}^{(d)} \\ \mathbf{B}^{(d)} \end{bmatrix}, [R_{d-1} + S_{d-1}, I_1, 1]\right)$

Return: $\mathcal{C}$

A-1-2 TT Subtraction

Algorithm 9: *subTT*: subtraction of TTs [29]

Data: Given two TTs: $\mathcal{A} = \langle\langle \mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(d)} \rangle\rangle$ with $\mathcal{A}^{(k)} \in \mathbb{R}^{R_{k-1}, I_k, R_k}$ and $\mathcal{B} = \langle\langle \mathcal{B}^{(1)}, \mathcal{B}^{(2)}, \ldots, \mathcal{B}^{(d)} \rangle\rangle$ with $\mathcal{B}^{(k)} \in \mathbb{R}^{S_{k-1}, I_k, S_k}$.

Result: $\mathcal{C} = \langle\langle \mathcal{C}^{(1)}, \mathcal{C}^{(2)}, \ldots, \mathcal{C}^{(d)} \rangle\rangle$ with $\mathcal{C}^{(k)} \in \mathbb{R}^{R_{k-1}S_{k-1}, I_k, R_k S_k}$ equaling the sum of $\mathcal{A}$ and $\mathcal{B}$..

$\mathbf{A}^{(1)} \leftarrow \text{reshape}(\mathcal{A}^{(1)}, [1, I_1 R_1])$

$\mathbf{B}^{(1)} \leftarrow \text{reshape}(\mathcal{B}^{(1)}, [1, I_1 S_1])$

$\mathcal{C}^{(1)} \leftarrow \text{reshape}\left(\begin{bmatrix} \mathbf{A}^{(1)} & -\mathbf{B}^{(1)} \end{bmatrix}, [1, I_1, R_1 + S_1]\right)$

for *k = 2 to d-1* **do**

$\mathbf{A}^{(k)} \leftarrow \text{reshape}(\mathcal{A}^{(k)}, [R_{k-1}, I_k R_k])$

$\mathbf{B}^{(k)} \leftarrow \text{reshape}(\mathcal{B}^{(k)}, [S_{k-1}, I_k S_k])$

$\mathbf{C}^{(k)} \leftarrow \begin{bmatrix} \mathcal{A}^{(k)} & \mathbf{0} \\ \mathbf{0} & \mathcal{B}^{(k)} \end{bmatrix}$

$\mathcal{C}^{(k)} \leftarrow \text{reshape}\left(\mathbf{C}^{(k)}, [R_{k-1} + S_{k-1}, I_k, R_k + S_k]\right)$

end

$\mathbf{A}^{(d)} \leftarrow \text{reshape}(\mathcal{A}^{(d)}, [R_{d-1}, I_1])$

$\mathbf{B}^{(d)} \leftarrow \text{reshape}(\mathcal{B}^{(d)}, [S_{d-1}, I_1])$

$\mathcal{C}^{(d)} \leftarrow \text{reshape}\left(\begin{bmatrix} \mathbf{A}^{(d)} \\ \mathbf{B}^{(d)} \end{bmatrix}, [R_{d-1} + S_{d-1}, I_1, 1]\right)$

Return: $\mathcal{C}$

A-1-3 TT Matrix-Vector Product

Algorithm 10: *MatVecTT*: matrix-vector product of TTm and TT [29]

Data: $\mathcal{A} = \langle\langle \mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(d)} \rangle\rangle$ with $\mathcal{A}^{(k)} \in \mathbb{R}^{R_{k-1}, I_k, J_k, R_k}$, $\mathcal{B} = \langle\langle \mathcal{B}^{(1)}, \mathcal{B}^{(2)}, \ldots, \mathcal{B}^{(d)} \rangle\rangle$ with $\mathcal{B}^{(k)} \in \mathbb{R}^{S_{k-1}, J_k, L_k, S_k}$.

Result: $\mathcal{C} = \langle\langle \mathcal{C}^{(1)}, \mathcal{C}^{(2)}, \ldots, \mathcal{C}^{(d)} \rangle\rangle$ with $\mathcal{C}^{(k)} \in \mathbb{R}^{R_{k-1}S_{k-1}, I_k, L_k, R_k S_k}$, as the product of $\mathcal{A}$ and $\mathcal{B}$.

for *k = 1 to d* **do**

$\mathbf{A}^{(k)}(r_{k-1} i_k r_k, j_k) \leftarrow \mathcal{A}^{(k)}(r_{k-1}, i_k, j_k, r_k)$

$\mathbf{B}^{(k)}(j_k, s_{k-1} l_k s_k) \leftarrow \mathcal{B}^{(k)}(s_{k-1}, j_k, l_k, s_k)$

$\mathbf{C}^{(k)}(r_{k-1} i_k r_k, s_{k-1} l_k s_k) \leftarrow \mathbf{A}^{(k)}(r_{k-1} i_k r_k, j_k) \mathbf{B}^{(k)}(j_k, s_{k-1} l_k s_k)$

$\mathcal{C}^{(k)}(r_{k-1} s_{k-1}, i_k, l_k, r_k s_k) \leftarrow \mathbf{C}^{(k)}(r_{k-1} i_k r_k, s_{k-1} l_k s_k)$

end

Return $\mathcal{C} = \langle\langle \mathcal{C}^{(1)}, \mathcal{C}^{(2)}, ..., \mathcal{C}^{(d)} \rangle\rangle$.

A-1-4 TT Inner Product

Algorithm 11: *innerprodTT*: Inner product two TTs [29]

Data: Given two TTs: $\mathcal{A} = \langle\langle \mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(d)} \rangle\rangle$ with $\mathcal{A}^{(k)} \in \mathbb{R}^{R_{k-1}, I_k, R_k}$ and $\mathcal{B} = \langle\langle \mathcal{B}^{(1)}, \mathcal{B}^{(2)}, \ldots, \mathcal{B}^{(d)} \rangle\rangle$ with $\mathcal{B}^{(k)} \in \mathbb{R}^{S_{k-1}, I_k, S_k}$.
Result: c, the inner product of $\mathcal{A}$ and $\mathcal{B}$
for *k = 1 to d* **do**
  $\mathbf{A}^{(k)} \leftarrow \text{reshape}(\text{permute}(\mathcal{A}^{(k)}, [1,3,2]), [R_{k-1}R_k, I_k])$
  $\mathbf{B}^{(k)} \leftarrow \text{reshape}(\text{permute}(\mathcal{B}^{(k)}, [2,1,3]), [I_k, S_{k-1}S_k])$
  $\mathcal{C}^{(k)} \leftarrow \text{reshape}(\mathbf{A}^{(k)}\mathbf{B}^{(k)}, [R_{k-1}R_k, S_{k-1}S_k])$
  $\mathbf{C}^{(k)} \leftarrow \text{reshape}(\text{permute}(\mathcal{C}^{(k)}, [1,3,2,4]), [R_{k-1}S_{k-1}, R_kS_k])$
end
$c = \mathbf{C}^{(1)}$
for *k = 2 to d* **do**
  $c \leftarrow c \times \mathbf{C}^{(j)}$
end
Return: c

A-1-5 Site-k Orthogonalization

Algorithm 12: *site-k*: k orthogonal TT [29]

Data: $\mathcal{A} = \langle\langle \mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(d)} \rangle\rangle$ with $\mathcal{A}^{(k)} \in \mathbb{R}^{R_{k-1}, I_k, R_k}$, k
Result: k-orthogonal TT $\mathcal{A}$.
for *j = 1 to k-1* **do**
  $\mathbf{A}^{(j)} \leftarrow \text{reshape}(\mathcal{A}^{(j)}, [S_{j-1}I_j, R_j])$
  $\mathbf{U}(s_{j-1}i_j, s_j)\mathbf{\Sigma}(s_j, s_j)\mathbf{V}(r_j, s_j)^T \leftarrow \text{SVD}(\mathbf{A}^{(j)})$ where $S_j = \min\{R_j, S_{j-1}I_j\}$
  $\mathcal{A}^{(j)} \leftarrow \text{reshape}(\mathbf{U}, [s_{j-1}, i_j, s_j])$
  $\mathcal{A}^{(j+1)} \leftarrow \mathcal{A}^{(j+1)} \times_1 \mathbf{\Sigma}\mathbf{V}^T$
end
for *j = d to k+1 step -1* **do**
  $\mathbf{A}^{(j)} \leftarrow \text{reshape}(\mathcal{A}^{(j)}, [R_{j-1}, I_jS_j])$
  $\mathbf{U}(r_{j-1}, s_{j-1})\mathbf{\Sigma}(s_{j-1}, s_{j-1})\mathbf{V}(i_js_j, s_{j-1})^T \leftarrow \text{SVD}(\mathbf{A}^{(j)})$ where
   $S_{j-1} = \min\{R_{j-1}, I_jS_j\}$
  $\mathcal{A}^{(j)} \leftarrow \text{reshape}(\mathbf{V}^T, [s_{j-1}, i_j, s_j])$
  $\mathcal{A}^{(j-1)} \leftarrow \mathcal{A}^{(j-1)} \times_3 \mathbf{U}\mathbf{\Sigma}$
end
Return $\mathcal{A} = \langle\langle \mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(d)} \rangle\rangle$.

A-2 Other Functions

A-2-1 Multi-index

Algorithm 13: *Multi-index*

Data: Given the column partitioning of a Tensor-Train matrix (TTm): $\mathbf{M} = [M_1, M_2, \ldots, M_d]$ with length equal to d and that the selected column is the i^{th} column. The algorithm is based on the little endian ordering, as seen in Section 3-2-1.

Result: Set of unit vectors: $\{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_d\}$

num $= i$
for *j = d to 1 step -1* **do**
  for *s = 1 to* M_j **do**
    if $s - 1 < \frac{num}{prod(\mathbf{M}(1:j-1))}$ ***and*** $\frac{num}{prod(\mathbf{M}(1:j-1))} \leq s$ **then**
      $\mathbf{e}_j = \text{zeros}(M_j, 1)$
      $\mathbf{e}_j(s) = 1$
      num $=$ num $- (s-1)\text{prod}(\mathbf{M}(1:j-1))$
    end
  end
end
Return: $\{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_d\}$

Bibliography

[1] Ben Benfold and Ian Reid. Stable multi-target tracking in real-time surveillance video. In *CVPR 2011*, pages 3457–3464. IEEE, 2011.

[2] Åke Björck. Numerics of gram-schmidt orthogonalization. *Linear Algebra and Its Applications*, 197:297–316, 1994.

[3] Steven D Brown. The kalman filter in analytical chemistry. *Analytica chimica acta*, 181:1–26, 1986.

[4] J Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrika*, 35(3):283–319, 1970.

[5] Pattrawut Chansangiam, Patcharin Hemchote, and Praiboon Pantaragphong. Inequalities for kronecker products and hadamard products of positive definite matrices. *Sci. Asia*, 35:106–110, 2009.

[6] SY Chen. Kalman filter for robot vision: a survey. *IEEE Transactions on Industrial Electronics*, 59(11):4409–4420, 2011.

[7] Edmond Chow and Yousef Saad. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM Journal on Scientific Computing*, 19(3):995–1023, 1998.

[8] Andrzej Cichocki. Generalized component analysis and blind source separation methods for analyzing mulitchannel brain signals. *Statistical and Process Models for Cognitive Neuroscience and Aging*, 1:201–272, 2007.

[9] Andrzej Cichocki. Era of big data processing: A new approach via tensor networks and tensor decompositions. *arXiv preprint arXiv:1403.2048*, 2014.

[10] Andrzej Cichocki, Namgil Lee, Ivan Oseledets, Anh-Huy Phan, Qibin Zhao, and Danilo P Mandic. Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions. *Foundations and Trends® in Machine Learning*, 9(4-5):249–429, 2016.

[11] Andrzej Cichocki, Rafal Zdunek, Anh Huy Phan, and Shun-ichi Amari. *Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation*. John Wiley & Sons, 2009.

[12] Seline de Rooij. Streaming video completion using a tensor-networked kalman filter, 2020.

[13] Mohinder S Grewal and Angus P Andrews. Applications of kalman filtering in aerospace 1960 to the present [historical perspectives]. *IEEE Control Systems Magazine*, 30(3):69–78, 2010.

[14] Mohinder S Grewal and Angus P Andrews. *Kalman filtering: Theory and Practice with MATLAB*. John Wiley & Sons, 2014.

[15] Fredrik Gustafsson and Fredrik Gustafsson. *Adaptive filtering and change detection*, volume 1. Citeseer, 2000.

[16] Richard A Harshman et al. Foundations of the parafac procedure: Models and conditions for an" explanatory" multimodal factor analysis. 1970.

[17] Frank L Hitchcock. Multiple invariants and generalized rank of a p-way matrix or tensor. *Journal of Mathematics and Physics*, 7(1-4):39–79, 1928.

[18] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

[19] CG Khatri and C Radhakrishna Rao. Solutions to some functional equations and their applications to characterization of probability distributions. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 167–180, 1968.

[20] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[21] Yehuda Koren. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 81(2009):1–10, 2009.

[22] Willem Laurenszoon van Doorn, Gijs Groote, Aniek Hiemstra, Thijs Veen, and Maarten ten Voorde. Reconstruction of faulty video data using the kalman filter and tensor networks. 2019.

[23] Xiaocan Li, Shuo Wang, and Yinghao Cai. Tutorial: Complexity analysis of singular value decomposition and its variants. *arXiv preprint arXiv:1906.12085*, 2019.

[24] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

[25] Peter S Maybeck. Square root filtering. *Stochastic models, estimation and control*, 1:368–409, 1979.

[26] PS Maybeck. Optimal filtering with linear system models. *Stochastic Models, Estimation, and Control*, 1:203–279, 1979.

[27] Vadim Olshevsky, Ivan Oseledets, and Eugene Tyrtyshnikov. Superfast inversion of two-level toeplitz matrices using newton iteration and tensor-displacement structure. In *Recent Advances in Matrix and Operator Theory*, pages 229–240. Springer, 2007.

[28] Ivan V Oseledets. Approximation of $2^d \times 2^d$ matrices using tensor decomposition. *SIAM Journal on Matrix Analysis and Applications*, 31(4):2130–2145, 2010.

[29] Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[30] Ivan V Oseledets and Sergey V Dolgov. Solution of linear systems and matrix inversion in the tt-format. *SIAM Journal on Scientific Computing*, 34(5):A2718–A2739, 2012.

[31] Ivan Valer'evich Oseledets and Eugene Evgen'evich Tyrtyshnikov. Approximate inversion of matrices in the process of solving a hypersingular integral equation. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 45(2):315–326, 2005.

[32] Victor Y Pan, Youssef Rami, and Xinmao Wang. Structured matrices and newton's iteration: unified approach. *Linear algebra and its applications*, 343:233–265, 2002.

[33] Gurnain Kaur Pasricha. Kalman filter and its economic applications. 2006.

[34] Roger Penrose. Applications of negative dimensional tensors. *Combinatorial mathematics and its applications*, 1:221–244, 1971.

[35] Benjamin Recht, Maryam Fazel, and Pablo A Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM review*, 52(3):471–501, 2010.

[36] Ulrich Schollwöck. The density-matrix renormalization group in the age of matrix product states. *Annals of physics*, 326(1):96–192, 2011.

[37] G Strang. Linear algebra and its application, 3rd. *Brooks Cole*, 1988.

[38] Ruchi Tripathi, Boda Mohan, and Ketan Rajawat. Adaptive low-rank matrix completion. *IEEE Transactions on Signal Processing*, 65(14):3603–3616, 2017.

[39] Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[40] Michel Verhaegen and Paul Van Dooren. Numerical aspects of different kalman filter implementations. *IEEE Transactions on Automatic Control*, 31(10):907–917, 1986.

[41] Michel Verhaegen and Vincent Verdult. *Filtering and system identification: a least squares approach*. Cambridge university press, 2007.

[42] Guoxu Zhou and Andrzej Cichocki. Fast and unique tucker decompositions via multiway blind source separation. *Bulletin of the Polish Academy of Sciences. Technical Sciences*, 60(3):389–405, 2012.

Glossary

List of Acronyms

LMS	Least-Mean Squares
ASVT	Adaptive Singular Value Thresholding
TNKF	Tensor-Networked Kalman Filter
TNSRKF	Tensor-Networked Square-Root Kalman Filter
PUKF	Partitioned Update Kalman Filter
TTm	Tensor-Train matrix
TT	Tensor-Train
TT-SVD	Tensor-Train Singular Value Decomposition
SVD	Singular Value Decomposition
CA	Component Analysis
CPD	Canonical Polyadic Decomposition
TD	Tucker Decomposition
PUKF	Partitioned Update Kalman Filter
MGS	Modified Gram-Schmidt
TT-MGS	Tensor-Train Modified Gram-Schmidt
SRKF	Square-Root Kalman Filter
TN	Tensor Networks