Reconstruction of Faulty Video Data Using the Kalman Filter and Tensor Networks

Willem Laurenszoon van Doorn, Gijs Groote,
Aniek Hiemstra, Thijs Veen, Maarten ten Voorde

June 18, 2019

Abstract

With the current state-of-the-art video reconstruction algorithms, the result of the reconstruction of a faulty video, with only 1% of known pixels, is not sufficient enough to be recognisable compared to the original video.

To overcome this problem a new algorithm that makes use of the Kalman filter and tensor networks is proposed. By representing the video data as a dynamic state space system, it becomes possible to use the Kalman filter to estimate the unknown pixels. Tensor networks are introduced to overcome the problem that some input matrices grow sufficiently large that they cannot be stored.

The proposed algorithm performs better than the current state-of-the-art at reconstructing faulty videos which have a low number of known pixels. However, the downside is that the computation time is significantly longer than current reconstruction algorithms.

Contents

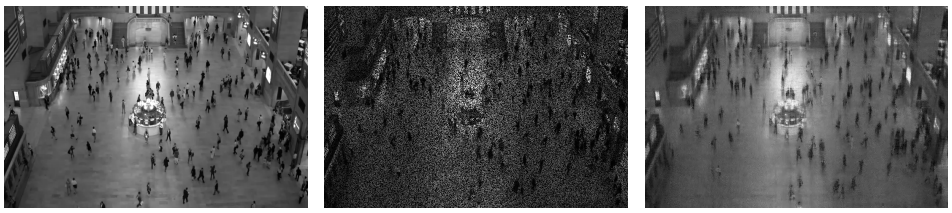
1 Introduction

Nowadays, cameras can be found everywhere and together they produce a lot of video data every moment. This data can be used for various purposes, for instance TV, surveillance or scientific experiments. Video data consists of frames that are in turn comprised of pixels. However, would the video data still be useful if it is faulty, i.e. some pixels have gone missing?
The answer is yes. Algorithms that are able to reconstruct video frames if a certain number of pixels are deleted, exist. Such a problem is called a 'low-rank matrix completion' problem. An example how faulty video may look like, can be found in figure 1.

An example of a state-of-the-art method for the reconstruction of faulty video data is the *Singular Values Regularised - Least Mean Squares* algorithm (SVR-LMS). [2] This algorithm involves the use of a singular value decomposition and is a least mean square method; a comparison of this algorithm with other algorithms can be found in section 2.1. However, these methods have certain drawbacks.

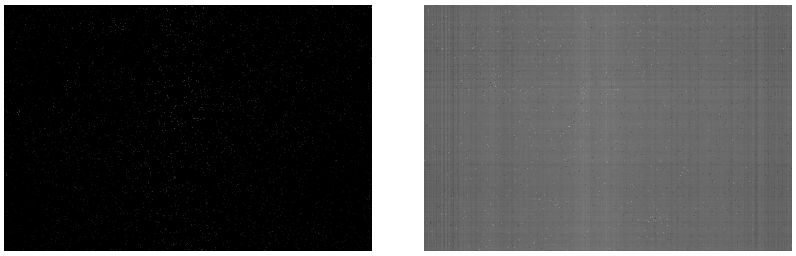
The most important drawback when using the SVR-LMS algorithm, is the quality of the reconstructed video, if a high percentage ($\geq 90\%$) of the pixels are faulty. If one deletes 50% of the pixels, the reconstruction gives a good impression of the original video, this is shown in figure 1. If one deletes more that 90%, one is likely to lose all details in a frame and obtain a uniformly dark reconstructed frame, as in figure 2. Another disadvantage is that the SVR-LMS size is a limiting factor. The SVD will become computationally too expensive if the frame size is too large. Thirdly, it can take up to 30 frames for the SVR-LMS algorithm to converge to an equilibrium value. Due to this, the first frames are poorly reconstructed and are thus less recognisable.

To overcome these disadvantages, a new method has been developed that will be discussed in this paper. It is based on the usage of state-space models and the Kalman filter. It is named the Kalman estimate algorithm. To test the new reconstruction algorithm, a video is chosen and destroyed deliberately instead



(a) Original frame (b) Faulty frame (c) SVR-LMS

Figure 1: Comparison between an unaltered frame [1], faulty frame and reconstructed frame where 50% of the pixels are deleted.



(a) Faulty frame (b) SVR-LMS

Figure 2: Reconstruction of frame 45 using SVR-LMS, with 99% of the pixels faulty.

of choosing an already faulty video. This is done in order to be able to compare the reconstructed video to its original version.

The usage of the state-space representation is a novelty in the reconstruction of video data. The motivation for its usage is that a video consists of a number of consecutive frames that differ slightly from one another. The state-space representation allows to model the change in pixels in a mathematical way and to take the observable pixels into account.[3] The exact representation is described in sections 2.2 and 3.2.

The state-space representation enables the deployment of a Kalman filter as well. The Kalman filter has been applied before in the areas of motion control and signal processing, among other areas. It was developed by the Hungarian-American scientist Rudolf E. Kálmán [4] and first implemented during the Apollo Space Program. However, its application in the field of video data reconstruction is new.

The Kalman filter will be implemented to make estimations of the missing pixels in a frame and will therefore act as the reconstructor of the video data. It is described in more detail in sections 2.3 and 3.3.

Ultimately, the performance of the Kalman estimate algorithm will be assessed on a number of criteria, the criteria being explained in section 2.1. The SVR-LMS algorithm is the reference algorithm used to compare the performance of the Kalman estimate algorithm with. The results can be found in section 4. The conclusions will follow in section 5.

2 Background

In this section, background information will be presented. This includes mathematical representations of dynamic systems, the Kalman filter and available algorithms for the reconstruction of video data.

2.1 Overview of methods available

Several algorithms already exist for the task of completing a low-rank matrix. One could choose between an adaptive method and a non-adaptive method. R. Tripathi *et al.* proposes a variety of adaptive methods and compares them to each other and to other non-adaptive algorithms. The adaptive methods that are proposed are the *Adaptive Singular Value Tresholding* (ASVT) method, the *Singular Value Regularised - Least Mean Square* (SVR-LMS) method and the *Proximal - Least Mean Square* (P-LMS) method. [2] An example of a non-adaptive method is the *Singular Value Tresholding* (SVT) method. [5] The performances of these methods will be briefly compared in this section.

Non-adaptive methods are methods that involve the same computation for every time step. This makes them computationally less expensive, but their performance does not change over time. Furthermore, their performance decreases if more entries are deleted.

Adaptive algorithms involve the usage of information of the previous time step in their computations. Their performance is characterised by large relative errors in the first time steps, but this improves fast over time and they eventually outperform the non-adaptive algorithms. The weak spot is that these algorithms will have large relative errors every time the underlying matrix changes abruptly, for example when the camera position changes, but these errors will again decrease fast with time.

A remarkable property of the adaptive algorithms is that their relative performance decreases slower than the non-adaptive algorithms when more entries are deleted. This makes them more suitable for applications where a large number of entries are deleted.

Among the adaptive algorithms, the performances differ slightly as well. The ASVT method eventually has the lowest relative error and the shortest time to reach convergence in the error, but is computationally expensive and is thus characterised by long run times. The SVR-LMS method has the worst performance, but has the shortest runtime. The P-LMS method has got a marginally better runtime, and its performance is between that of the ASVT and the SVR-LMS methods. It is noteworthy that the performance of these three methods converges as the number of deleted entries increases. In other words, the spread in the performance becomes smaller as the number of deleted entries becomes

larger. However, the absolute performance still worsens as the number of missing entries increases. [2]

2.2 State-space model

In the Kalman estimate algorithm, a state-space model is deployed for representing the dynamics of a system; this is a new feature compared to the algorithms mentioned earlier.

The model consists of two equations written down below, the first of which (2.1) represents the changes of the state and the second of which (2.2) represents the measured output of the system. [3] In this paper, the relations are linear, time-invariant (LTI) and in discrete time. The time index is denoted with a subscript k.

$$\boldsymbol{x_{k+1}} = A\boldsymbol{x_k} + B\boldsymbol{u_k} + \boldsymbol{w_k} \tag{2.1}$$

$$\boldsymbol{y_k} = C\boldsymbol{x_k} + D\boldsymbol{u_k} + \boldsymbol{e_k} \tag{2.2}$$

Symbol	*Dimension description*
n	Number of state variables
p	Number of control signals
m	Number of outputs
k	Discrete time index

Symbol	*Vector/matrix name*	*Dimension (rows × columns)*
$\boldsymbol{x}_k$	State vector	$n \times 1$.
$\boldsymbol{u}_k$	Input vector	$p \times 1$
$\boldsymbol{y}_k$	Measurement vector	$m \times 1$
$\boldsymbol{w}_k$	Process noise	$n \times 1$.
$\boldsymbol{e}_k$	Measurement noise	$m \times 1$
A	Dynamics matrix	$n \times n$
B	Control matrix	$n \times p$
C	Sensor matrix	$m \times n$
D	Feedforward matrix	$m \times p$

The variables $\boldsymbol{y}_k$ and $\boldsymbol{x}_{k+1}$ that occur in the state-space equations will be used later as input for the Kalman filter. The Kalman filter is a state observer that will make an estimations of $\boldsymbol{x}_{k+1}$. This will be discussed in the following section.

2.3 Kalman filter

The Kalman filter is named after Rudolf E. Kálmán [4] and first implemented by Stanley F. Schmidt in the use of nonlinear trajectory estimation for the Apollo program. [6] The Kalman filter has found many uses in the aerospace industries as well in military applications, mainly in the use of guidance, navigation and control of air/space vehicles and weapon systems. The Kalman filter is also used in signal processing, econometrics and used in robotics for motion planning and control.

The Kalman filter is a sort of state observer which can specifically be used for stochastic systems. A state observer can estimate the internal state of a system. One aspect that characterises the Kalman filter from other state observers is that it only requires data from the previous state. States at other time steps do not need to be stored, this is also why the Kalman filter works very quickly. [7] The data always contains some degree of noise. So all available measurement data from the previous state is combined with any knowledge about the system and its measuring devices, to estimate the next true value. With this information, the Kalman filter can make an educated guess about what the next step of the system is going to be. The Kalman filter works best for discrete-time measurements. [8] It can be used for linear systems, and the extended Kalman filter can also be used for nonlinear systems. Thus the Kalman filter can best be used if a prediction, on what the system is going to do next, is wanted and uncertain information about a system is available. [9]

There are many different forms of the Kalman filter. One which is most commonly used consists of 5 equations which can be divided into two categories: the time update equations and the measurement update equations. The time update equations calculate a prediction for a certain time step based on the previous time step. [8] While the measurement update equations use feedback to correct these values, these two sets of equations are put in an ongoing cycle. Some assumptions have been made for these equations which are that, w_k is normally distributed $w_k \sim \mathcal{N}(0, W_k)$, as well as $r_k \sim \mathcal{N}(0, R_k)$. It is also assumed that $x_0 \sim \mathcal{N}(x_0, P_0)$. The equations are shown below.

Time update equations:

$$x_{k+1}^{-} = Ax_k + Bu_k \tag{2.3}$$

$$P_{k+1}^{-} = AP_kA^T + W_k \tag{2.4}$$

Measurement update equations:

$$v = y_{k+1} - Cx_{k+1}^{-} \tag{2.5}$$

$$S = R_k + CP_{k+1}^{-}C^T \tag{2.6}$$

$$K = P_{k+1}^{-}C^T S^{-1} \tag{2.7}$$

$$x_{k+1} = x_{k+1}^{-} + Kv \tag{2.8}$$

$$P_{k+1} = P_{k+1}^{-} - KSK^T \tag{2.9}$$

The initial state consists of $\hat{\mathbf{x}}_k$ and an error covariance P_k. P is the measure of uncertainty in a particular state. Firstly, the time update equations are being used to calculate the a priori state $\hat{x}_{k+1}^{-}$ and its error covariance P_{k+1}^{-} where A_k $(n \times n)$ shows the correlation between the state at time step k to the state at time step $k+1$ in the absence of process noise u_k and a driving function B. B $(n \times p)$ shows the correlation between the control input u_k and the state x and W_k is the process noise covariance matrix.
Next, the measurement update equations are being used, where first the innovation matrix v $(m \times 1)$ and its innovation covariance matrix S $(m \times m)$ are calculated. R_k is the process noise covariance matrix. v is equal to the difference between the measurement of the state and its estimation. After, the Kalman gain K can be calculated. The Kalman gain helps to minimise the a posteriori error covariance. It determines how much the a priori estimate and the measurement contribute to the calculation of $\hat{\mathbf{x}}_k$ and its error covariance P_k. If the measurement noise R_k is small, the measurement is trusted more than the a priori state estimate x_{k+1}^{-}. Using the Kalman gain, the state vector $\hat{\mathbf{x}}_k$ and the according error covariance P_k is calculated. To calculate K, C and R_k are needed. C $(m \times n)$ shows the correlation between the state x and the measurement y_k, R_k is the measurement noise. [8] [9]

2.4 Tensor Network

The Kalman filter produces large matrices. Especially the covariance matrix P_k, the Kalman gain K and state vector X_k are too large to store on a modern day computer. With a 480×720 video the memory needed to store the covariance matrix P_k is around 20 GB. To be able to make computations, tensor networks that represent the matrices are used instead.[10]
In figure 3 from left to right the tensors v_i, m_{jk}, t_{lmn}, t_{opqr} are shown. The number of lines sticking out of the circles represent the number dimensions of the tensor. A tensor with one dimension is also called a vector, two dimensions is a matrix. The circle with a number of lines is not yet a tensor network.

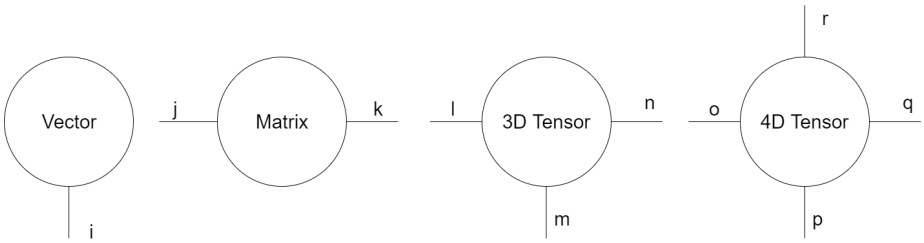


Figure 3: Visual representation of tensors.

A tensor that is connected to another tensor, like $A^{(1)}$ to $A^{(2)}$ in figure 4, creates a tensor network. Letters a, b, c, d, e and f denote the dimensions of the tensor network.

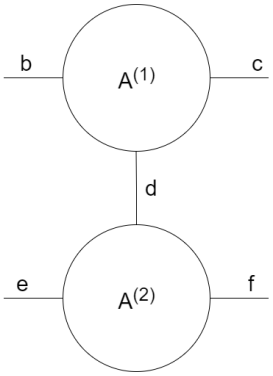


Figure 4: Visual representation of a tensor network.

Connecting two tensors and creating a tensor network minimises memory without loss of data. Matrix A can be written as tensor network A_{becf} as in figure 4.

$$A_{becf} = \sum_{d=1}^{G} A^{(1)}_{bdc} A^{(2)}_{edf}$$

A_{becf} is the contraction of index d, which amounts to the sum over its G possible values. [11]

For an example of a matrix converted to tensor network see 3.4.2.

3 Proposed method

In this section, the contribution of this paper towards the development of the Kalman estimate algorithm for video reconstruction is discussed. It consists of the following: state-space models, Kalman filter and the use of tensor networks for computation purposes. It builds forth upon the information described in the previous chapter.

3.1 Creation of faulty video data

Before the video can be reconstructed, a faulty video is needed. In this case, a video will first be made faulty so the results can be compared with the original video. This faulty video is formed mathematically. To do this, the video data needs to be represented mathematically first.

3.1.1 Mathematical representation of video data

A video is a series of images or frames that are being shown one after another. These frames are comprised of pixels, each pixel represents a small part of an image. These pixels are normally made up of three colours, namely red, green and blue; grayscale can be used alternatively. The values of these colours need to be translated to an 8-bit value to be used by a computer. A bit value of 0 corresponds to the smallest possible intensity of a colour, a bit value of 255 corresponds to the maximum possible intensity of a colour. In this way, the colour white is represented by a bit value of 255 for red, green and blue and the colour black is represented by a bit value of 0 for these colours. All these bit values can be placed in a matrix. The length and the width match the pixel dimensions of a frame which are the 1st and 2nd dimension of the matrix. The 3rd dimension of the matrix represents the number of colours and can thus be one (for grayscale) or three (for red, green and blue). For each frame, a new matrix is formed. So, if one shows these matrices one after each other, it will result in a video. In this paper, the changes between frames will be represented by a state-space model. Section 3.2 contains more information on this.

3.1.2 Deletion of pixels

To create a faulty video, certain pixels in a matrix have to be deleted. The first step to do this, is to determine which pixels will be deleted. For this job, a *selection matrix* J is constructed. J consists of only zeros and ones. The indices of the zeros represent the pixels that will be deleted; the indices of the ones correspond to the pixels that will remain. This selection matrix will be multiplied element-wise with the matrices that represent the video data to produce a faulty video matrix. The selection matrix will remain the same for

all frames on which it is applied, but it can be different for every colour of a frame.

So if one wants to delete a certain percentage of the entries, say 90%, the selection matrix has to be filled for 90% with zeros and for the remaining 10% with ones. The position of these zeros will be chosen randomly. The selection matrix is also involved in the state-space model, as described in the following section. An example is shown below, where a faulty frame F is calculated by multiplying a 3×4 greyscale frame A with the selection matrix J.

$$A = \begin{pmatrix} 84 & 211 & 45 & 11 \\ 122 & 90 & 100 & 239 \\ 6 & 250 & 29 & 156 \end{pmatrix}, J = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

$$F = A \odot J = \begin{pmatrix} 0 & 211 & 0 & 11 \\ 0 & 90 & 0 & 0 \\ 6 & 0 & 0 & 156 \end{pmatrix}$$

3.2 Usage of state-space models

As already mentioned in the introduction and section 2.2, a state-space model is used to represent the video data. The equations therefore are modified compared to the equations (2.1) and (2.2). The equations are:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{w}_k \tag{3.1}$$

$$\boldsymbol{y}_k = C\boldsymbol{x}_k \tag{3.2}$$

At the end of this section, an example on the calculation of the state and output vectors can be found. The vector $\boldsymbol{x}_k$ in equation (3.1) represents the state of frame k. The state vector of frame k is obtained by reshaping the frame matrix k into a column vector. The dynamics matrix A in equation (2.1) contains the relation between the states $\boldsymbol{x}_{k+1}$ and $\boldsymbol{x}_k$. In the case of a video as the dynamic system, it is assumed in this paper that every pixel is only related to itself and not to other pixels. Thus, the matrix A is the identity matrix and can therefore be left out of the system equations. Since there will not be any input to the dynamic system and perfect measurements of the data are available, $\boldsymbol{u}_k = 0$ and $\boldsymbol{e}_k = 0$ and are left out of equation (2.1).

The matrix C in equations (2.2) and (3.2) is used to map the state vector to the pixels that can be observed. This depends on the selection matrix described in section 3.1.2, because that selection matrix determines which pixels can be observed and which pixels cannot. The number of columns correspond to the total number of pixels in the state vector $\boldsymbol{x}_k$, the number of rows correspond to the number of pixels that can be observed; those pixels are captured in the vector $\boldsymbol{y}_k$. In every row there can be just one entry $'1'$, the position of which will correspond to the element of $\boldsymbol{x}_k$ that can be observed.

Furthermore, the noise term $\boldsymbol{w}_k$, that is assumed to be Gaussian white noise, remains. The noise is modelled to have a mean close to zero and a known covariance $\boldsymbol{\sigma^2}$. It is used to represent the changes between two consecutive frames. The calculation of the noise parameters is specified in section 4.3.

The following example serves to make clear how the $\boldsymbol{x}_k$ and $\boldsymbol{y}_k$ are derived from a matrix F with faulty video data. The same frame as in section 3.1.2 is used.

To arrive at $\boldsymbol{x}_k$, one should reshape the entries of the frame into a large column vector.

$$F = \begin{pmatrix} 0 & 211 & 0 & 11 \\ 0 & 90 & 0 & 0 \\ 6 & 0 & 0 & 156 \end{pmatrix} \rightarrow \boldsymbol{x}_k = \begin{pmatrix} 0 & 0 & 6 & 211 & 0 & 0 & 0 & 0 & 0 & 11 & 0 & 156 \end{pmatrix}^T$$

Similarly to $\boldsymbol{x}_k$, the entries of $\boldsymbol{y}_k$ are the values of the pixels that could be examined directly.

$$\boldsymbol{y}_k = C\boldsymbol{x}_k =$$

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} *$$

$$\begin{pmatrix} 0 & 0 & 6 & 211 & 0 & 0 & 0 & 0 & 0 & 11 & 0 & 156 \end{pmatrix}^T =$$

$$\begin{pmatrix} 6 & 211 & 11 & 156 \end{pmatrix}^T$$

The variables $\boldsymbol{y}_k$ and $\boldsymbol{x}_{k+1}$ will be used subsequently in the Kalman filter to estimate the pixels that cannot be observed. This is reviewed in the next section.

3.3 Kalman Filter

Following the assumptions made in section 3.2 and assuming the measurements are noise-free and that the process noise covariance is independent of time step k, equations (2.3) to (2.9) become:

$$x_{k+1}^- = x_k \tag{3.3}$$

$$P_{k+1}^- = P_k + W \tag{3.4}$$

Measurement update equations:

$$v = y_{k+1} - Cx_{k+1}^- \tag{3.5}$$

$$S = CP_{k+1}^{-}C^T \tag{3.6}$$

$$K = P_{k+1}^{-}C^T S^{-1} \tag{3.7}$$

$$x_{k+1} = x_{k+1}^{-} + Kv \tag{3.8}$$

$$P_{k+1} = P_{k+1}^{-} - KSK^T \tag{3.9}$$

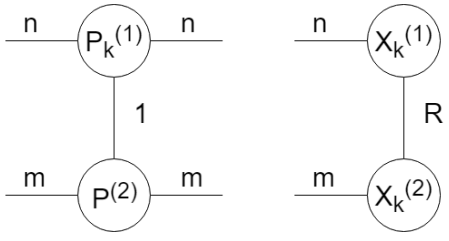
v and S are intermediate variables. The initial variables to put into the filter are P_0, x_0, W. W remains constant throughout the filter so it is independent of the time step k.

Because the measurement update matrices can get very big, tensor networks are used. This will be further explained in section 3.4.

3.4 Tensor Network

In order to work with large matrices, tensor networks are used.[12] An issue is storing these matrices with the available memory space. For a 720×480 video, storing the covariance matrix P_k, without the use of tensor networks, would take about 20 GB for one value of k.

Large matrices produced by the Kalman filter are the covariance matrix P_k and the state vector X_k, so these are represented as tensor networks. In the tensor network, the rank R is the number of linear independent vectors which are stored and used for computation, see figure 5. Lowering R gives faster computation, but a worse representation of the frame.
When the sum of two tensor networks is taken, the number of ranks are added. This results in a high number of ranks, which is unwanted. The rank should be kept low in order to keep the memory consumption manageable. For the



(a) P_k (b) X_k

Figure 5: P_k and X_k for an n x m video in tensor network form.

tensor network of P_k, the rank should always be reduced to one, otherwise it becomes too large to store. The implemented algorithm allows the rank of the state vector to grow, due to a summation that is called upon for every measured pixel. In the case of 1% of the total pixels measured of a 720×480 video and a chosen rank of 50, the rank will grow to $3456 + 50 = 3506$. When rank 3506 is reached, the rank will be reduced back to 50. For a more detailed explanation of how this rank grows, see section 3.4.3.

The tensor network are matrices $P^{(1)}$ of size 1 x n x n and $P^{(2)}$ (1 x m x m) instead of one matrix of size n^2 x m^2. The rank R in figure 5b should be chosen between 1 and the minimum of n and m. If $R = minimum(n, m)$, no data will be lost. If $R = 1$ is chosen, only the most important value will remain. Choosing an acceptable rank for X_k will discussed in section 4.2. Due to the large size of P_k and W, the rank $R = 1$ is chosen for these matrices.

3.4.1 Kalman filtering utilising tensor networks

A time consuming step could be (3.7), where the S^{-1} needs to be computed. S is an $m \times m$ matrix, so for a large number of m the computing time increases, while for $m = 1$ this step is very fast. So, to decrease computation time, (3.5) to (3.8), are computed for every measured pixel separately. (3.5) then becomes

$$v = y_{k+1}(\alpha) - X^{(1)}(i,:) * X^{(2)}(:,j) \tag{3.10}$$

Where i and j denote the row and column of the measured pixel. α denotes the index of the same pixel in vector form. Matlab notation is used for indexing. (3.6) and (3.7) become

$$S = P^{(1)}(i,:) * P^{(2)}(:,j) \tag{3.11}$$

$$K^{(1)} = P^{(1)}(i,:) * S^{-1} \tag{3.12}$$

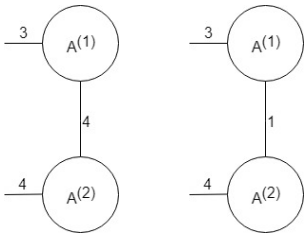
$$K^{(2)} = P^{(2)}(:,j) \tag{3.13}$$

3.4.2 Tensor network examples

Matrix A can be represented in tensor network form by taking the singular value decomposition of A, where $[U, S, V] = svd(A)$.

$$A = \begin{pmatrix} 84 & 211 & 45 & 11 \\ 122 & 90 & 100 & 299 \\ 6 & 259 & 29 & 156 \end{pmatrix}, U = \begin{pmatrix} -0.391 & 0.59 & -0.706 \\ -0.682 & -0.701 & -0.208 \\ -0.618 & 0.4 & 0.677 \end{pmatrix}$$

$$S = \begin{pmatrix} 458.58 & 0 & 0 & 0 \\ 0 & 221.8 & 0 & 0 \\ 0 & 0 & 94.412 & 0 \end{pmatrix}, V = \begin{pmatrix} -0.261 & -0.151 & -0.855 & -0.423 \\ -0.663 & 0.744 & 0.079 & -0.017 \\ -0.226 & -0.144 & -0.349 & 0.898 \\ -0.664 & -0.634 & 0.376 & -0.123 \end{pmatrix}$$



(a) A with full rank

(b) A with rank reduced to 1

Figure 6: Graphical representation of the tensor network of A.

Then $A = U * S * V^T$, $A^{(1)} = U * S$ and $A^{(2)} = V^T$, graphically shown in figure 6a. The dimensions of $A^{(1)}$ are $l \times m \times n$, where in this case $l = 3, m = 4$ and $n = 1$. The dimensions of $A^{(2)}$ are $o \times m \times p$, where $o = 3, m = 4$ and $p = 1$. The rank of this representation is full rank, which could be reduced to one to lower the required memory storage. Reducing the rank is done by deleting the lower singular values of S and the corresponding columns of U and V, so that

$$A^{(1)}_{RR} = U(:,1) * S(1,1) = \begin{pmatrix} -0.391 \\ -0.682 \\ -0.618 \end{pmatrix} * \begin{pmatrix} 458.58 \end{pmatrix} = \begin{pmatrix} -179.27 \\ -312.76 \\ -283.43 \end{pmatrix}$$

$$A^{(2)}_{RR} = (V(:,1))^T = \begin{pmatrix} -0.261 & -0.663 & -0.226 & -0.664 \end{pmatrix}$$

Note that the dimensions of $A^{(1)}_{RR}, 3 \times 1$, and $A^{(2)}_{RR}, 1 \times 4$, correspond to the dimensions in figure 6b. Contracting $A^{(1)}_{RR}$ and $A^{(2)}_{RR}$ over its shared dimension, 1 in the case above, would yield A_{RR}, where A_{RR} is matrix A after its rank has been reduced.

$$A_{RR} = \begin{pmatrix} 46.817 & 118.82 & 40.548 & 119.1 \\ 81.677 & 207.3 & 70.74 & 207.78 \\ 74.016 & 187.86 & 64.105 & 188.29 \end{pmatrix}$$

A_{RR} differs from A by a relatively large margin, this implies that too much information was lost reducing the rank to one. For a better representation, the value for the rank could be increased. Section 4.2 explains how the value, to which the rank is reduced, is determined.

3.4.3 Rank reduction of a tensor summation

For (3.4), (3.8) and (3.9), two tensor networks are summed. If $B = A + A$ is to be calculated as the sum of two tensor networks, it can be done by reshaping $A^{(1)}$ to size $(l * n) \times m$ and $A^{(2)}$ to size $m \times (o * p)$. $B^{(1)}_{col}$ is the concatenation of $A^{(1)}_{col}$ and $A^{(1)}_{col}$

$$A^{(1)}_{col} = \begin{pmatrix} -179.27 & 130.91 & -66.68 & 0 \\ -312.76 & -155.49 & -19.667 & 0 \\ -283.43 & 88.779 & 63.879 & 0 \end{pmatrix},$$

$$B^{(1)}_{col} = \begin{pmatrix} A^{(1)}_{col} & A^{(1)}_{col} \end{pmatrix}$$

$$= \begin{pmatrix} -179.27 & 130.91 & -66.68 & 0 & -179.27 & 130.91 & -66.68 & 0 \\ -312.76 & -155.49 & -19.667 & 0 & -312.76 & -155.49 & -19.667 & 0 \\ -283.43 & 88.779 & 63.879 & 0 & -283.43 & 88.779 & 63.879 & 0 \end{pmatrix},$$

$$A^{(2)}_{row} = \begin{pmatrix} -0.261 & -0.663 & -0.226 & -0.664 \\ -0.151 & 0.744 & -0.144 & -0.634 \\ -0.855 & 0.079 & -0.349 & 0.376 \\ -0.423 & -0.017 & 0.898 & -0.123 \end{pmatrix},$$

$$B^{(2)}_{row} = \begin{pmatrix} A^{(2)}_{row} \\ A^{(2)}_{row} \end{pmatrix} = \begin{pmatrix} -0.261 & -0.663 & -0.226 & -0.664 \\ -0.151 & 0.744 & -0.144 & -0.634 \\ -0.855 & 0.079 & -0.349 & 0.376 \\ -0.423 & -0.017 & 0.898 & -0.123 \\ -0.261 & -0.663 & -0.226 & -0.664 \\ -0.151 & 0.744 & -0.144 & -0.634 \\ -0.855 & 0.079 & -0.349 & 0.376 \\ -0.423 & -0.017 & 0.898 & -0.123 \end{pmatrix}$$

$B^{(1)}$ is $B^{(1)}_{col}$ reshaped back to it original size of $A^{(1)}$, $l \times (m+m) \times n$. In the case of this example $B^{(1)} = B^{(1)}_{col}$, because $n = 1$. The same can be done for $B^{(2)}$. If $B^{(1)}$ and $B^{(2)}$ were to be contracted, the resulting matrix is expected to be equal to $2A$

$$B = B^{(1)} * B^{(2)} = \begin{pmatrix} 168 & 422 & 90 & 22 \\ 244 & 180 & 200 & 598 \\ 12 & 518 & 58 & 312 \end{pmatrix} = 2A$$

Figure 8 shows the calculation of $B^{(1)}_{row}$ and $B^{(1)}$ graphically, while figure 7 graphically shows (2.4).

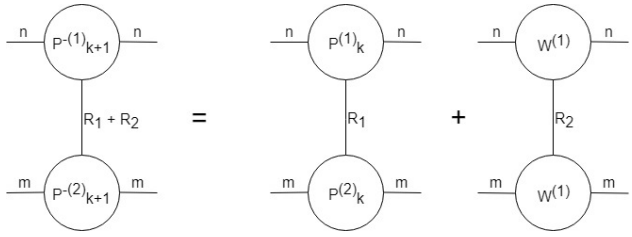


Figure 7: (2.4) in graphical form as tensor network.

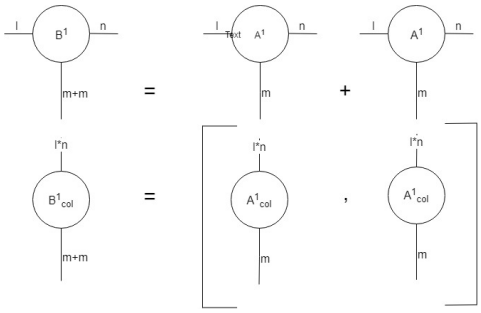


Figure 8: The sum of two tensors in graphical representation.

As apparent from the example, the rank of B has grown to $m + m = 2$. For (2.4) the summation is done for every frame k, so that the rank of this sum grows to $k + 1$, if m is one. This poses a problem, for it increases the required computational power while tensor networks were introduced to lower the required computational power. The rank of the sum can be reduced during the summation [13] by using the SVD of both $B^{(1)}_{col}$ and $B^{(2)}_{row}$ and reducing the rank of these matrices separately

$$[U^{(1)}, S^{(1)}, V^{(1)}] = SVD(B^{(1)}_{col})$$

$$[U^{(2)}, S^{(2)}, V^{(2)}] = SVD(U^{(1)} * B^{(2)}_{row})$$

The lower singular values can be deleted, in the same manner as the calculation of $A^{(1)}_{RR}$

$$B^{(1)}_{RR} = U^{(1)} * U^{(2)}(:, 1) * S^{(2)}(1, 1)^T$$

$$B^{(2)}_{RR} = (V^{(2)}(:, 1))^T$$

Here the rank is reduced from $m + m = 8$ to one.

4 Experimental results

In this chapter, the results that have been obtained with the Kalman estimate algorithm of chapter 3 are presented and discussed. The chapter first deals with the estimation of parameters necessary for the reconstruction and subsequently with the results and suggestions for future work.

4.1 Computer specifications

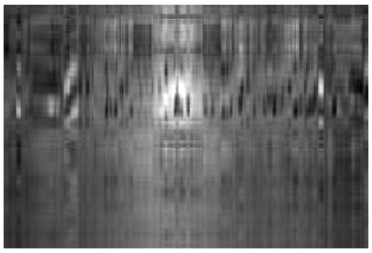
The experiments are done in Matlab R21018b - academic use 64bit, running on Intel Core i7 6700HQ at 2.60GHz, with 8,00GB RAM. The test video is footage from the Grand Central Station in New York.[1] The quality of the video is 480p, which has 480×720 pixels.

4.2 Rank determination

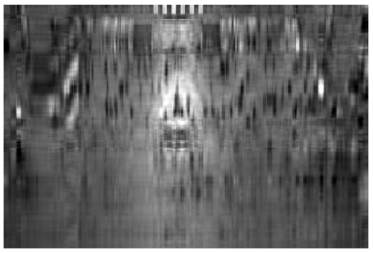
When reconstructing the video, tensor network representation of matrices are used for computing large matrices. The summation of two tensor networks is done for the a priori estimate of the covariance matrix (2.4), and for the measurement update equations (2.8) and (2.9), twice for every pixel in every frame. There is data in the tensor network which is of lesser importance, but is taking up as much memory space as more important data. An example of how to remove the data of lesser importance, see section 3.4.2 and 3.4.3.

The single value decomposition of a frame from the grand central station footage is taken. The rank is reduced and reconstructed to be visually inspected, see figure 9. The visual effect of the rank reduction has a large impact for a rank lower than 50, but barely improves when the rank is increased above 50. Figure 10 shows the computation time required to sum two tensor networks for a frame of 480×720, where the rank differs from 1 to the full rank 480. The computation time appears to rise linearly when the rank exceeds 100, although the increase is in the order of 10^{-3} seconds. The frame of the frames in the following sections is reduced to 50, for the visual quality does not increase substantially when the rank exceeds 50, while the computation time is kept relatively low.

(a) Original frame



(b) Rank reduced to 5



(c) Rank reduced to 10

(d) Rank reduced to 20

(e) Rank reduced to 50

(f) Rank reduced to 100

Figure 9: Visualisation of reducing the rank of one frame.

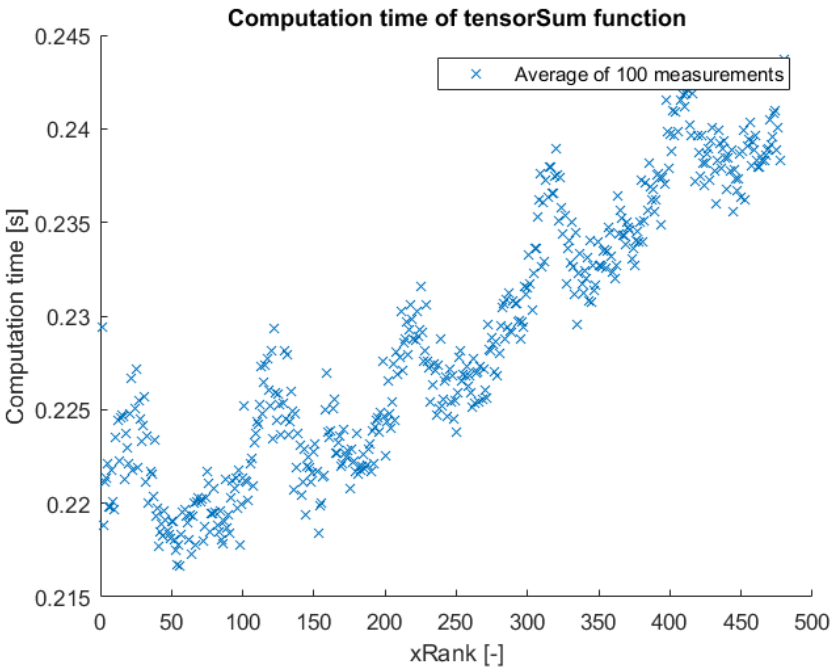


Figure 10: Computation time versus the sum of two tensor networks with different rank, with matrix size 480×720.

4.3 Determine the standard deviation of the process noise

In order to determine the standard deviation and show that the distribution of the process noise (w_k) is normal/Gaussian distributed, the difference between consecutive states have been calculated and plotted in a histogram. From figure 11 it becomes clear that the process noise w_k is indeed normally/Gaussian distributed around the zero mean, as previously assumed in 3.2, and has an σ_w of 5.008. The peak around the zero continues upwards to a probability of 0.7926. this spike is attributed to the used data set being a video with a fixed point of view and relatively constant lighting, which means that static objects often have pixel values that are constant between two consecutive frames. Because the value of σ_w changes for each new video, it has to be calibrated in order to get the optimal result.

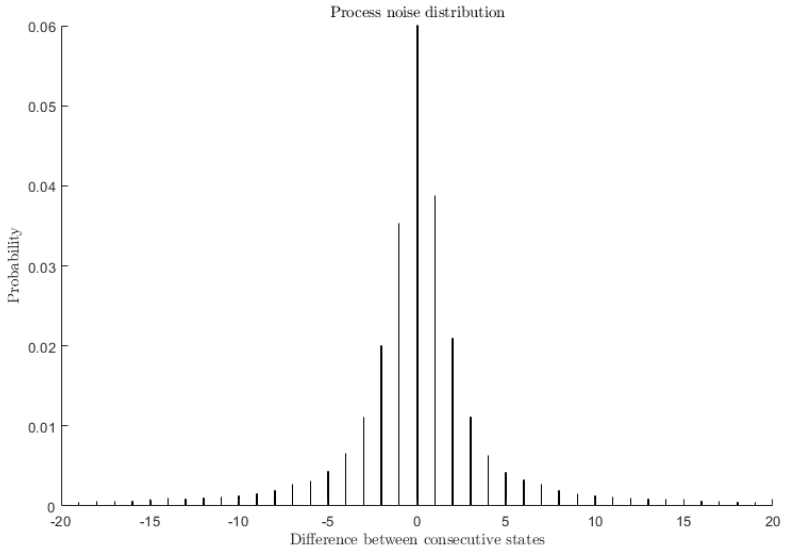
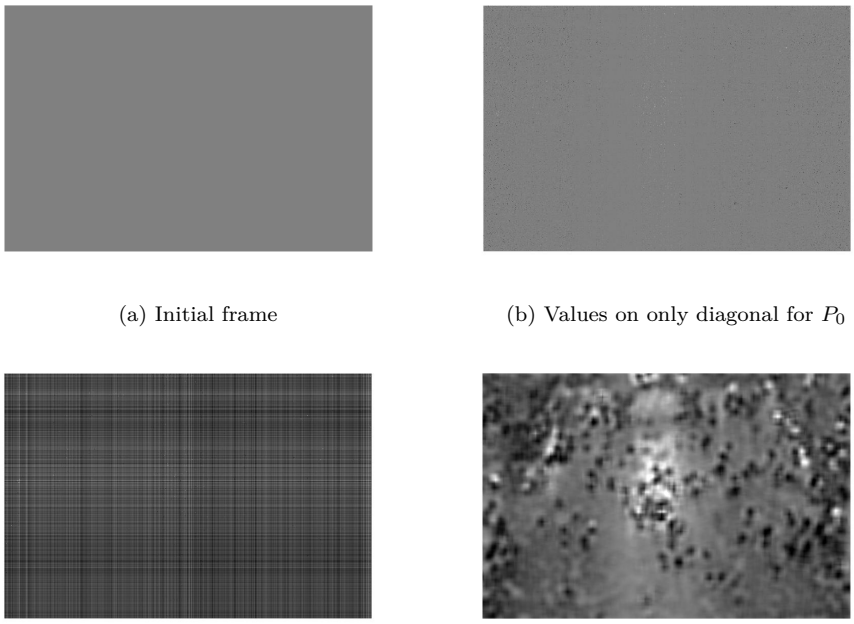


Figure 11: Normal distribution of the process noise (w_k), calculated from the grand central station data set, [1] with $\mu = 0.0015$ and $\sigma = 5.008$, the peak at zero continues up to a probability of approximately 0.8.

4.4 Initial value for P

The covariance P_k matrix indicates how much one pixel in a frame influences other pixels. Every pixels influences every other pixel. If two pixels are near one other they will influence each other than if they are further away from one other.
The value for P_0 for the time update equations (3.4) must be chosen. Several initial values for the covariance matrix P_0 have been tried, with different results. If P_0 is a matrix with non-zero values only on the diagonal, only the measured pixels change. Pixels on positions where no measurements is done do not change. This implies that the value for $\hat{\mathbf{x}}_{k+1}$ is the same for $\hat{\mathbf{x}}_k$ for all pixels that are not measured. If the covariance matrix of the noise W is also assumed a matrix with non-zero values on the diagonal, there will still be only non-zero values on the diagonal for P_{k+1}. This means that the state estimate $\hat{\mathbf{x}}_k$ will be static for all values of k, where no measurement is done.
If P_0 is a matrix filled with non-zero values, the change in a measured pixel will have influence on all other pixels, even if these pixels are separated by a large part of the frame. This would mean that the whole frame would change with the average difference of the measured pixel and the estimated pixel. The best result has been achieved when the pixels only influence the pixels around them. If, for example, top left pixel ($\alpha = 1$) is changed, its effect on change of the



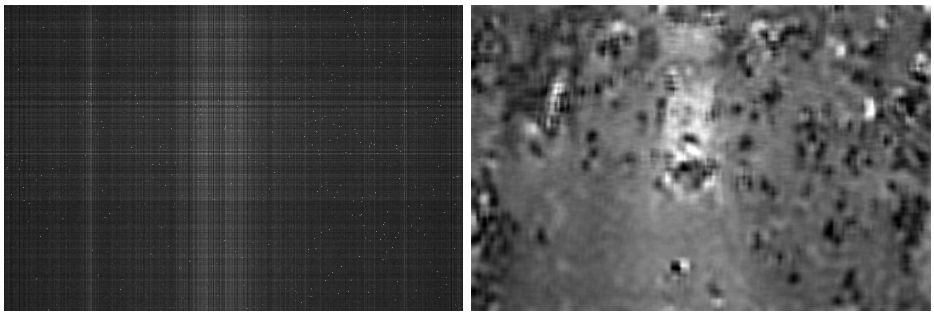
(a) Initial frame

(b) Values on only diagonal for P_0

(c) Values everywhere in P_0

(d) Higher values when closer to current pixel as P_0

Figure 12: Reconstructions for different P_0.



(a) SVR-LMS (b) Kalman estimate

Figure 13: Frame 198 reconstructed with SVR-LMS (13a) and the Kalman estimate algorithm (13b).

neighbouring pixels should have a magnitude similar to

$$\begin{pmatrix} 1 & 0.5 & 0.25 & 0.125 \\ 0.5 & 0.4 & 0.2 & 0.1 \\ 0.25 & 0.2 & 0.1 & 0 \end{pmatrix}$$

where pixel $\alpha = 1$, where α is the pixel index, only influences pixel ($\alpha = 5$) only 0.4 times as much as it influences itself. Then, $P_{\alpha=1}$ for the top left pixel looks like

$$P_{\alpha=1} = (1 \;\; 0.5 \;\; 0.25 \;\; 0.125 \;\; 0.5 \;\; 0.4 \;\; 0.2 \;\; 0.1 \;\; 0.25 \;\; 0.2 \;\; 0.1 \;\; 0)^T$$

Similarly to the pixel in the example above, P_α can be constructed for all pixels to complete the covariance matrix P.
Figure 12 shows that the estimate $\hat{X}_{k=10}$ is the best for the last of the explained P_0.

4.5 Comparison of Performance

To compare the algorithms, the performance has to be assessed. Aside from visual inspection, see figures 13a and 13b, this is done by a couple of methods that will be discussed in this section. For the comparison of the performance, 99% of the pixels has been deleted.

The results of the visual inspection show that the SVR-LMS algorithm eventually renders almost uniformly dark images, so the reconstruction is useless if too many pixels are deleted. The Kalman estimate algorithm on the contrary yields images where small details are not visible anymore, but it still provides a useful picture for recognising larger scale features.

The visual results match the outcomes of these assessment methods; this will be explained in the following sections.

4.5.1 Computation time per frame

The first criterion that is used, is the computation time (or runtime) of the algorithm. This is done by measuring the runtime of the calculations for a certain frame. The values in table 1 are averaged over 100 matrices for a 360×480 and a 480×720 video. The values are determined by Matlab.

Method	360 x 480	480 x 720
SVR-LMS	0.03301	0.07529
Kalman estimate	17.3658	90.85

Table 1: Avarage runtime in seconds over 100 frames, with 1% known entries.

As can be seen in the table, the runtimes for the Kalman estimate algorithm are 3 orders of magnitude larger than for the SVR-LMS algorithm. Furthermore, if the size of the video is doubled, the computation time for the SVR-LMS algorithm approximately doubles, whereas the computation time for the new algorithm is about 5 times higher. The computation time therefore probably does not scale linearly for the Kalman estimate algorithm, which might prove problematic for even larger video sizes.

For the Kalman estimate, the summation of two tensor networks is the most time consuming step. More than 90% of the time is spent on this step. This is largely due to reshaping large matrices and the number of times this function is called upon. This step is called upon for every measured pixel, additionally the tensor size grows with every iteration of pixel updates. This explains why the computation time does not scale linearly with frame size. So the more pixels are measured, the longer the runtime of the Kalman estimate algorithm will be. The SVR-LMS algorithm uses the SVD as the most expensive step, where it takes an SVD of the whol frame, but it is used just once for every frame. Furthermore, this step is independent of the number of measured pixels.

4.5.2 Peak signal-to-noise ratio (PSNR)

The *Peak Signal-to-Noise Ratio* (PSNR) [14] yields the ration of the power of the reconstruction vs. the power of the disturbance signal. The calculation involves two equations. Equation (4.1) is used to calculate the *Mean Squared Error* (MSE) between the original video and the reconstructed video. The $\boldsymbol{x}_k(i)$ in the equation denotes the entries of a frame of the original video and $\hat{\boldsymbol{x}}_k(i)$ represents a frame of the reconstructed video.

Equation (4.2) subsequently compares the power of the MSE to the maximum power that could be achieved. If the reconstructed video is identical to the original one, the MSE will be zero and the PSNR value will go to infinity.

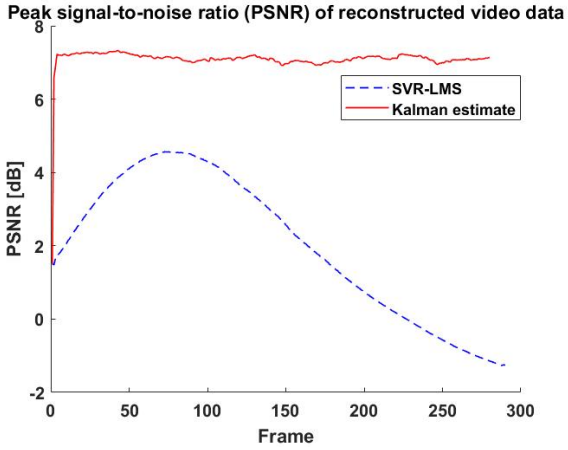


Figure 14: Performance comparison using PSNR, between SVR-LMS and the Kalman estimate, when 99% of the entries is deleted.

Therefore, the higher the value for the PSNR, the better the reconstruction.

$$MSE = \frac{1}{mn} \sum_{i=0}^{nm-1} \left[\boldsymbol{x}_k(i) - \hat{\boldsymbol{x}}_k(i) \right]^2 \tag{4.1}$$

$$PSNR = 20 \log(255) - 10 \log(MSE) \tag{4.2}$$

The value of 255 that appears in equation (4.2) is the maximum value a number can have when it is an 8-bit number, as in the case of this paper. The value should correspond to the maximum value that can be represented by the computer data type chosen. The calculation of the PSNR value is done for every time step and for every colour of the video separately.

The PSNR values are plotted in figure 14. The size of the video is 480×720. The adaptive behaviour of both algorithms is visible. The Kalman estimate algorithm converges to an equilibrium value rapidly, even within 10 frames. Although the general power of the reconstruction is low, it should be remarked that it is higher than the power of the SVR-LMS algorithm for every frame.

Concerning the PSNR value of the SVR-LMS algorithm, it can be noted that the value is low and after some time even negative. It does not reach an equilibrium value on this time interval; instead it first increases and then decreases. It is not clear whether an equilibrium will be reached after more frames. This corresponds to the results of the frames: the SVR-LMS will eventually yield almost uniformly dark frames.

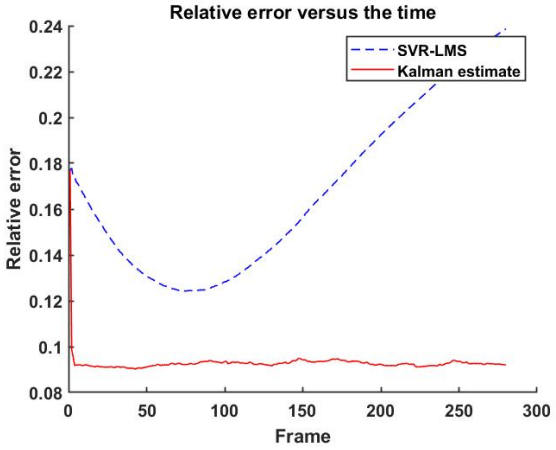


Figure 15: Performance comparison using relative error, between SVR-LMS and the Kalman estimate, when 99% of the entries is deleted.

4.5.3 Relative error

The calculation of the relative error involves the use of a matrix norm; the Frobenius norm is chosen and denoted with a subscript F. [2] The norm takes the sum of the squared elements. The other symbols are identical to the ones in the previous section. The calculation is again repeated for every frame and every colour.

$$\text{Relative error}(k) = \frac{\|\hat{\boldsymbol{x}}_k - \boldsymbol{x}_k\|_F}{\|\mathbf{x}_k\|_F} \tag{4.3}$$

In figure 15, the results are shown for the relative error of both algorithms. The size of the video is the same: 480×720. Again, it becomes clear that the Kalman estimate algorithm outperforms the SVR-LMS algorithm. The curves of the relative error behave similarly to the curves of the PSNR value, but this time the vertical movements are inverted.

The equilibrium error of the Kalman estimate algorithm is reached within a couple of frames (< 10) and has an equilibrium value between 0.08 and 0.10. There is no equilibrium value for the SVR-LMS algorithm; the relative error first decreases only to increase after about 80 frames. There is no indication if a possible equilibrium value exists.

Figure 16 shows what the effect of abruptly changing the underlying matrix has on the relative error. This is done in order to study what the effect is on the reconstruction of a video with transitions, when using the Kalman estimate algorithm. Two videos have been used to create the input where the transition between them happens at the 20th frame. The Kalman estimate algorithm shows a peak in the relative error at the transition but quickly adjusts to the

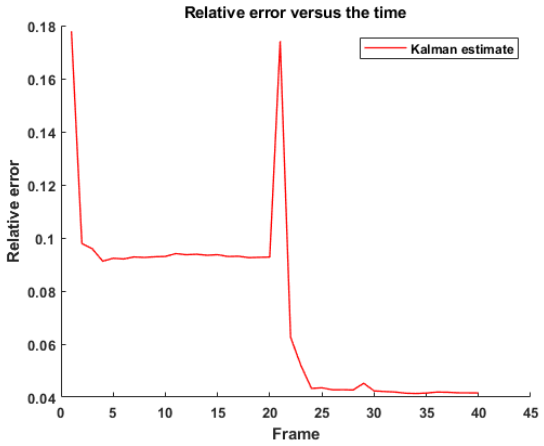


Figure 16: Relative error of the Kalman estimate and SVR-LMS with 99% removed pixels. The underlying matrix changes abruptly at frame 20.

new video and evens out at an even lower relative error than the video used in the beginning. The lower relative error is most likely caused by the video used and not as result of a transition.

5 Conclusion and future work

This paper introduced a new method for reconstructing faulty video data using the Kalman filter and Tensor-networks. Using these concepts for reconstructing video data had yet to be done. Other algorithms like the SVR-LMS already existed for this purpose but they only gave good results with video data that still had a relatively high number of known entries. The use of the Kalman estimate algorithm aimed to solve that problem.

The results are that the Kalman estimate algorithm, in the use of reconstructing faulty video data, performs better than the SVR-LMS filter when used on videos with a high percentage of removed pixels e.g. 99%. As observed by visual inspection and by the calculated lower relative error and higher PSNR. However, the run-time of the Kalman estimate is significantly higher than the that of the SVR-LMS filter. Also, the run-time of the Kalman estimate is relative to the number of removed pixels whereas for the SVR-LMS filter, it is not. When low run-time is a requirement and the number of removed pixels is low e.g. 70%, it is not advised to use the Kalman estimate purely based on the high run-time.

The relative error of the Kalman estimate algorithm converges very rapidly, often within 10 frames which for a 25fps video is as quick as 0.4 seconds. This is beneficial when the underlying matrix changes abruptly as the Kalman Estimate

algorithm quickly adjusts to this new change. This means that the filter is also effective on videos with lots of transitions and not only on continues fixed point-of-view videos as used in this paper.

A lot of work has been done to create and optimise the Kalman estimate algorithm described in this paper, but there are still some future work that can be done to improve it further. One of the main things would be to apply smoothing techniques to the reconstructed video in order to further improve the results. Also optimising or finding other ways to create the initial value of the covariance matrix (P_0) can result in a better reconstruction of the faulty video. Additionally the Kalman estimate in this paper is only used on one colour video data, testing the filter on RGB video data has yet to be tested. A point of interest for using RGB Video data would be to see if the respective colours of different objects still remain correct. Computation time could be decreased by creating tensor networks with smaller tensors than the tensors used in this paper.

6 References

References

[1] X. Wang B. Zhou and X. Tang. Understanding collective crowd behaviors: Learning a mixture model of dynamic pedestrian-agents., 2012. http://www.ee.cuhk.edu.hk/ xgwang/grandcentral.html.

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

[3] R. M. Murray K. J. Ångstrom. *Feedback system: an introduction for scientist and engineers*. Princeton University Press, Princetons, 1 edition, 2008.

[4] Rudolf Emil Kálmán. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[5] J. F. Cai. A singular value tresholding algorithm for matrix completion. *SIAM Journal on optimization*, 20(4):1956–1982, 2010. doi: 10.1137/080738970.

[6] S. F. Schmidt. The kalman filter - its recognition and development for aerospace applications. *Journal of Guidance and Control*, 4(1):4–7, 1981.

[7] Matlab. Understanding kalman filters, 2017. https://nl.mathworks.com/videos/series/understanding-kalman-filters.html.

[8] Gary Bishop Greg Welch. An introduction to the kalman filter. *Department of Computer Science, University of North Carolina at Chapel Hill*, page 16, sep 1997.

[9] Simo Särkkä. *Bayesian Filtering and Smoothing*, volume 3 of *IMS Textbooks*. Cambridge University Press, 2013.

[10] Ivan V. Oseledets. Approximation of 2^d\times2^d matrices using tensor decomposition. *SIAM Journal on Matrix Analysis and Applications*, 31(4): 2130–2145, 2010.

[11] Román Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349:117 – 158, 2014.

[12] Paul Springer. Welcome to the tensor network, 2019. https://tensornetwork.org/.

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

[14] Michael L. Hilton, Björn D. Jawerth, and Ayan Sengupta. Compressing still and moving images with wavelets. *Multimedia systems*, 2(5):218–227, 1994.

7 Appendix

Link to Github repository: https://github.com/GijsGroote/BEP

Contents

- Prediction of Kalman filter
- Update step of Kalman filter

```
function [X,P] = KalmanFilterTensorScriptFast(X,P,W,y,R)

%INPUT
%X is a struct with a the tensor network representation of the state vector
%P is a struct with a tensor network representation of the covariance matrix
%W is a struct with a tensor network representation of the process noise
%covariance matrix
%y is a vector with the value of the measured pixels
%R is a vector with the index of the measured pixels
```

Prediction of Kalman filter

```
%first kalman equation, but not needed in our case X[k+1] = X[k]
%X = X;

%A priori estimate of the covariance matrix P
[P.P1, P.P2] = tensorSum(P.P1, P.P2, W.W1, W.W2, 1);
```

Update step of Kalman filter

```
for n = 1: length(R)
%CX is the estimation of what should be measured according to the
%prediction, R1 and R2 are the indexes of the pixel that is currently measured
[CX, R1, R2] = CtimesX(R(n),X.X1,X.X2);

% v = y - C*X
v = y(n) - CX;         %v is a temporary variable (scalar)

% S = C*P*C'
S = P.P1(R1,R1)*P.P2(R2,R2);   %S is temporary variable (scalar)

%K = P*C'*1/S
K.K1 = P.P1(:,R1)*1/S; %K is a tensor network of two vectors
K.K2 = P.P2(:,R2);     %K is the Kalman Gain
```

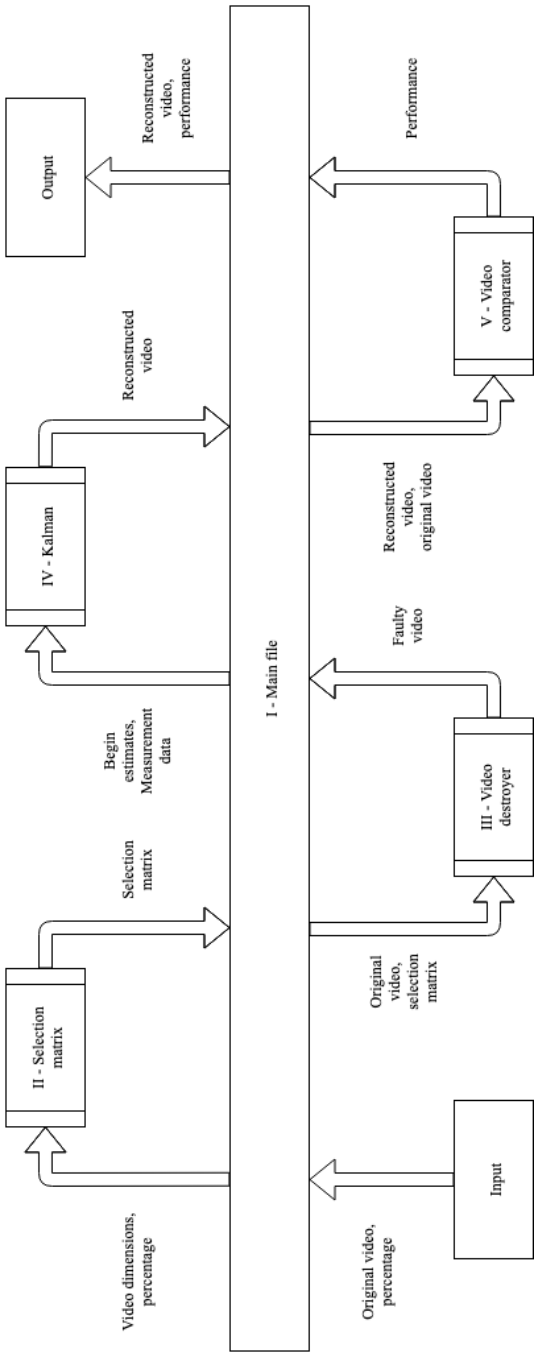


Figure 17: Flow of information in the Matlab code with the order of execution denoted with roman numerals

```
% X = X+ceK*v - The new state FUNCTION OUTPUT
[X.X1, X.X2] = tensorSum(X.X1, X.X2, K.K1*v, K.K2);
end
% -K*S*K'
KSKT.KSKT1 = -1*K.K1*S*(K.K1'); %KSK is a tensor network of two 2d matrixes
KSKT.KSKT2 = K.K2*(K.K2');

%P = P - K*S*K' - The new covariance matA=rix FUNCTION OUTPUT
[P.P1, P.P2] = tensorSum(P.P1, P.P2, KSKT.KSKT1, KSKT.KSKT2);

end

function [estimatedOutput,R1,R2] = CtimesX(R,X1,X2)
    %INPUT
    % R is the index of the measured pixel
    % X1 is a part of tensor network of X size n x 1 x d
    % X2 is a part of tensor network of X size m x 1 x d
    %OUTPUT
    % estimatedOutput is C * X-
    % R1 is the index for the rows
    % R2 is the index for the colums
    n = size(X1,1);
    m = size(X2,1);
    d = size(X1,3);

    % map R to the corresponding rows of the tensor network
    [R1, R2] = ind2sub([n, m], R);

    % estimate the value of the pixel from the tensor network of X
    estimatedOutput = reshape(X1(R1,:,:),1,d)*reshape(X2(R2,:,:),d,1);
    % The reshaping is needed to get the right dimensions for multipling
    % (X1 and X2 need to be 2d, while they are 1x1xd)
end
```

Contents

- Reshaping 3d to 2d
- Sum of the tensor networks
- Reshape into desired tensors
- Rank reduction using SVD

```
function [sumA, sumB] = tensorSum(P1, P2, W1, W2,desiredRank)

% Take the sum of 2 matrices, and reduce rank of the sum to the desired. If
% desiredRank is not specified, no rank reduction will be performed.
```

```
% rank.
% INPUT
% --(P1)--  --(W1)--
%    |         |
%    |PR       |PW
%    |         |
% --(P2)--  --(W2)--
% P1, P2 are tensor network representation of P (e.g. covariance)
% W1, W2 are tensor network representation of W (e.g. noise)
% P1 and W1 are of the same dimensions (n,n, ...)
% P2 and W2 are of the same dimensions (m,m, ...)
% x1 and x2 are numbers that can differ from eachother
% for covariance and noise PR=WR=1

% PR and WR (in drawing) are defined in the function, not in the input
% arguments.

% desiredRank is the rank of the outputted tensor network
% OUTPUT
% --(rdA)--
%     |
%     | desiredRank
%     |
% --(rdB)--
% reducedRankA, reducedRankB are tensor network represenation of the sum
%of input with the third rank reduced to desiredRank

% The size of the inputs is needed for reshaping
[n1, n2, PR] = size(P1); % The size of the 1st tensor network
[m1, m2, ~] = size(P2);  % The size of the 2nd tensor network

WR = size(W1,3);    %lenght of 3rd dimension of W1 (and W2)
```

Reshaping 3d to 2d

```
P1a = reshape(P1, n1*n2, PR); %reshape P1 to column of length n1*n2 (PR colums side by side)
W1a = reshape(W1, n1*n2, WR); %reshape W1 to column of length n1*n2 (WR colums side by side)

P2a = reshape(P2, m1*m2, PR)'; %reshape P1 to row of length m1*m2 (PR rows side by side)
W2a = reshape(W2, m1*m2, WR)'; %reshape P1 to row of length m1*m2 (WR rows side by side)

columnMatrix = [P1a W1a];   %merge the columns
rowMatrix = [P2a; W2a];     %merge the rows
```

Sum of the tensor networks

nargin is the number of inputs, so if desiredRank (nargin = 4) is not specified,
this function just sums, without reducing rank

```
if nargin == 4
```

Reshape into desired tensors

```
    sumA = reshape(columnMatrix, n1, n2, PR+WR);  %reshape into desired form
    sumB = reshape(rowMatrix', m1, m2, PR+WR);     %reshape into desired form
else % If rankReduced is specified (nargin = 5 =/= 4).
```

Rank reduction using SVD

SVD is taken and then the largest value of S is saved

```
    [U1, S1, V1] = svd(columnMatrix,'econ');
    interMatrix1 = S1*V1'*rowMatrix;

    [U2, S2, V2] = svd(interMatrix1, 'econ');
    % deleting all but the first 'desiredRank' rows (and colums) of U2,S2 and V2
    U2(:,(desiredRank+1):end) = [];
    S2 = S2(1:desiredRank,1:desiredRank);
    V2(:,(desiredRank+1):end) = [];
    %computing the tensor representation of the sum
    sumA = reshape(U1*U2*S2, n1, n2, desiredRank);
    sumB = reshape(V2', m1, m2, desiredRank);
end
end
```