Dynamic texture modeling and synthesis using Multi-kernel Gaussian process Dynamic model

University of Jean Monnet

Master 1 MLDM

Maksim KOPTELOV

August 2016

Academic Supervisors:

Assoc. prof. Marianne CLAUSEL
Assoc. prof. Rémi EMONET
Prof. Olivier ALATA

Introduction

Our world is full of examples of dynamic textures. Any visual scene can be considered as dynamic texture. What is a definition of texture? For a single image a texture is a realization from stationary stochastic process. For a sequence of images this texture becomes dynamic by defining the stochastic process of interest over space and time.

Why is it important to study dynamic textures? There are many applications, which can be found in computer vision and video processing fields. Some of them are: video indexing, video surveillance, object tracking and video animation [2].

Existing researches [1] shows that dynamic texture can be captured from a video and can be used then to synthesize new videos with necessary length of time. Existing dynamic textures modeling and synthesis methods can be summarized into three categories: physics based, sampling based and learning based methods [1].

Physics based methods synthesize dynamic texture by modeling the physical mechanism of some specific phenomena. The advantage is the quality of the result, which can be obtained, however these methods are not universal and the computation cost for a single texture is extremely expensive [1].

Sampling based methods reassembles the sequence of frames taken from original video to form a longer sequence with unnoticeable transitions. They require to storage large amounts of original frames and cannot generate a texture frame unseen [1].

Learning based methods assume specific model linking the different frames of the video and learn its parameters. Once all parameters are learnt the information from the model can be used to synthesize new videos. Compared with other two types of methods, they are universal. Moreover, these methods only require few memory space for dynamic texture synthesize and can achieve high data compression.

Thus, in this work learning based method is used for dynamic texture synthesis. The method proposed consists of three imperative steps:

1. Dimensionality reduction,
2. Dynamic texture learning
3. Dynamic texture synthesis.

Any video sequence has high dimensionality, thus curse of dimensionality problem will be faced (data becomes too sparse and distance functions become not accurate). To prevent it dimensionality reduction process must be applied. As an algorithm, it can be linear (like PCA) or nonlinear. However, linear algorithms cannot capture complex dynamic textures and some nonlinear algorithms produce irreversible mapping or different coordinate systems [1]. Thus, it is necessary to find an algorithm, which is free of these weak points. In this work, the reduction function is inferring by using Gaussian process.

The algorithm for learning of dynamic texture cannot be linear due to the fact that most of dynamic textures are not linear. It can be switching or piecewise linear, but in this case it would not be universal, because it does not suite for all dynamic textures. Thus, more flexible model must be found. In this work, dynamic texture is modeling using first-order Markov model based on Gaussian process [1].

Dynamical texture synthesis step generates new video data using learned dynamic texture. It can be done by estimate necessary parameters (latent variable vector, observed dynamic texture vector, kernel matrix mapping hyperparameters, weights for kernel functions and different kernel parameters), then by predicting new sequence of dynamic textures. It is necessary to have a good performance at this step due to the big computational cost of optimization. In this work, mean-prediction method based on first-order Markov model using Gaussian prediction is adopted for dynamical texture synthesis [1].

Context of the project

The grant for the project is provided by Laboratoire Jean Kuntzmann (LJK). Located in Grenoble the LJK is a laboratory of Applied Mathematics and Computer Science. It brings together teams of different branches: mathematicians, numerical analysts, computer graphics, image processing and computer vision specialists. Three main departments represent the organization of the laboratory:

- The Probability / Statistics department
- The Geometry-Image department
- The department of Models and Deterministic Algorithms.

My chief supervisor, associate professor Marianne CLAUSEL, is a member of the Optimization and Learning for Data Science team (DAO) of the LJK. The research topics of the DAO team are focused around mathematical methods for data science capitalizing on the interplay between mathematical optimization and machine learning. Main research themes are:

- Convex and stochastic optimization
- Distributed, incremental, and randomized numerical algorithms
- Robust, scalable pattern recognition and machine learning
- Applications to academic and real-life problems in computer vision, electricity generation, signal/image processing, and finance.

I did my training period in Laboratoire Hubert Curien (HC). Located in Saint-Etienne the HC is the laboratory of Physics and Computer Science. Two main departments represents the organization:

- Optics, Photonics & Hyper-frequencies
- Computer Science, Telecom & Image

Objectives of the research teams are:

- Machine Learning
- Data Mining & Information Retrieval
- Knowledge Representation
- Multi-agents systems
- Virtual Communities and Social Networks
- Optical Design and Image Reconstruction
- Macroscopic Modelisation of Images
- Image analysis and understanding

There is a direct connection between my master program, Machine Learning and Data Mining, and the subject of my internship. Dynamic texture modeling and synthesis stages of the project imply Machine Learning processes for capturing and synthesis the texture and application of the approach could be interesting in Computer Vision field. Moreover, both the laboratories, the LJK and HC, are widely involved in doing researches in Machine Learning and Computer Vision areas.

Survey of the state of the art

1. Gaussian process

There are two possible definitions for Gaussian processes. A stochastic process is a sequence of random variables indexed by time $(X_t)_t$. It is said to be Gaussian if and only if for every finite set of indices $t_1,\dots,t_k$ in the index set T: $X_{t1,\dots,tk} = (X_{t1},\dots, X_{tk})$ is a multivariate Gaussian random variable [10]. It can be seen as infinite-dimensional generalization of multivariate Gaussian distributions.

In multivariate Gaussian distribution initially we have d random variables $(X_{t1},\dots, X_{td})$, which are not necessary equally spaced in timeline t_i. For example, for d = 100 it is necessary to have 100 time indexes t_i (Fig. 1, i_0 – index when space is increased).

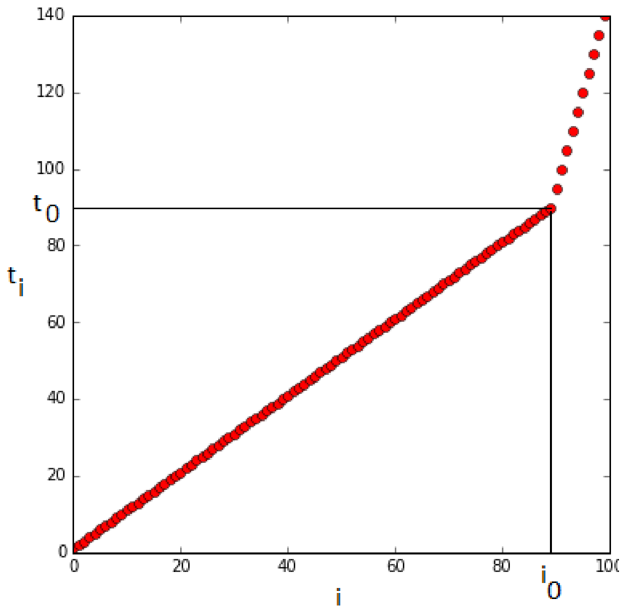


Fig. 1. Time indexes t_i

Any Gaussian multivariate distribution can be characterized by its mean and covariance. In what follows, we shall considered that all the vector of interest are centered. We then have zero mean initially.

In our case, we shall assume a specific form for the covariance of any subsample $(X_{t1},\dots, X_{td})$ of our Gaussian process. More precisely, we assume that it can be expressed by (1).

$$k(t_i, t_j) = cov(X_{t_i}, X_{t_j}) = \alpha \exp\left(-\frac{\|t_i - t_j\|^2}{2l^2}\right) \quad (1)$$

Parameter *α* in kernel function (1) is a scale parameter, whereas the parameter *l* is a dispersion parameter. List of most popular kernels along with its covariance matrices for the given timeset T is presented on Fig. 4.

From one covariance matrix and one sequence of time $(t_1,\dots,t_k)$ possible Gaussian distribution $(X_{t1},\dots, X_{tk})$ with zero mean and covariance C (see Fig. 2) is shown on Fig. 3 (x_{ti} represents values of random variables, t_i represents time indexes).

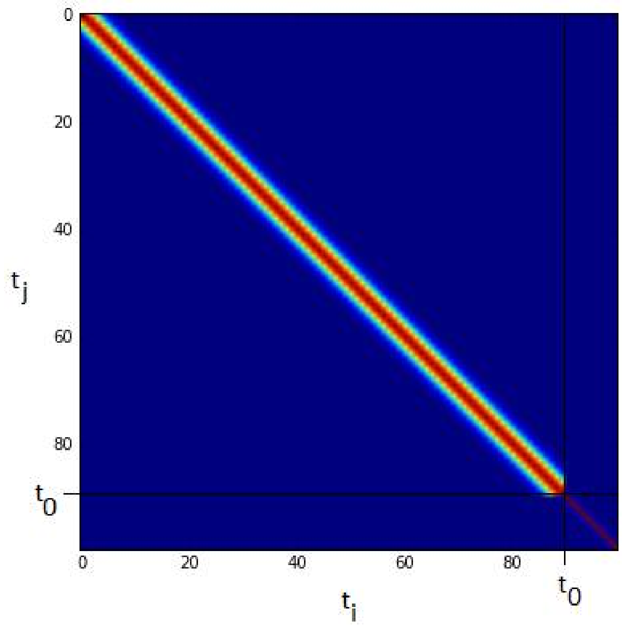


Fig. 2. Covariance matrix C

As we can see, the trend of generated samples is not spoiled by not equally spaced time indexes and can be easily seen.

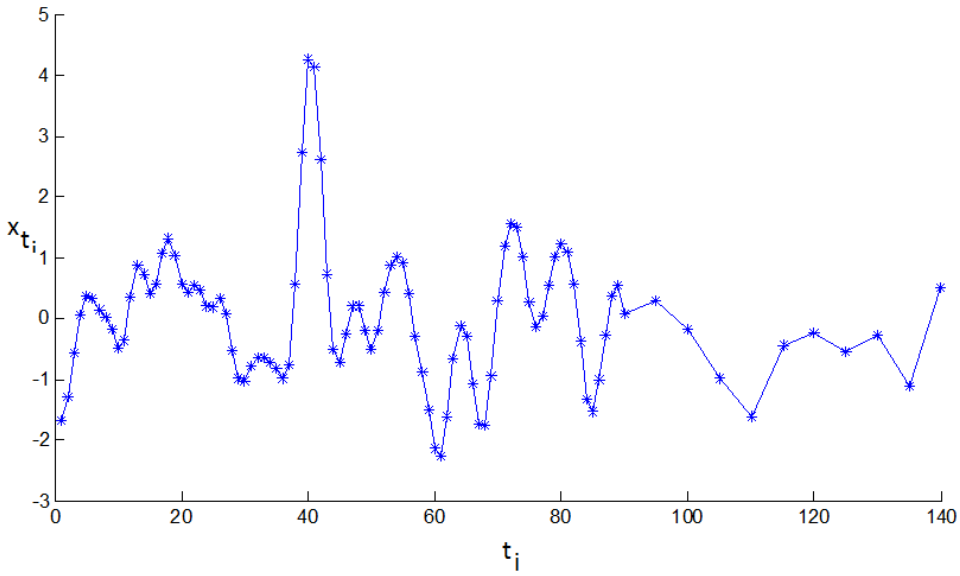


Fig. 3. One possible realization of multivariate Gaussian distribution

Now assume that we are conditioning our stochastic process for example setting $x_{t1} = 1$, $x_{t2} = 1$ with $t_1 = 40$, $t_2 = 80$. Computed the new mean vector and covariance matrix using equations (2) from [3] for multivariate conditional density, a new set of realizations of multivariate conditional distribution will be obtained (where f_* – denotes the Gaussian vector $x_* = x_{\{t \in T \setminus T'\}}$, f – denotes the Gaussian vector $x_f = x_{\{t \in T'\}}$ and K – covariance matrices associated to the two random variable sets $X_* = X_{\{t \in T \setminus T'\}}$, $X_f = X_{\{t \in T'\}}$ computed as $K_{*,f} = cov(X_*,X_f)$, $K_{f,f} = cov(X_f,X_f)$, $K_{f,*} = cov(X_f,X_*)$, $K_{*,*} = cov(X_*,X_*)$).

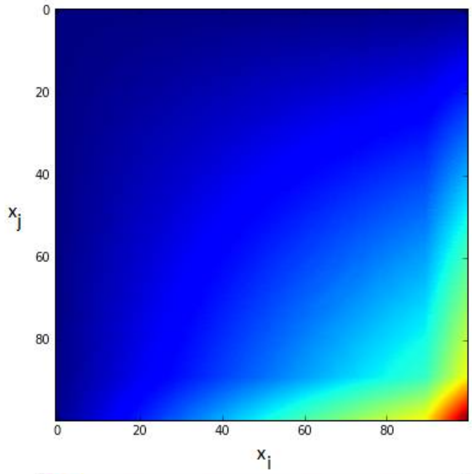
$$p(\mathbf{f}_*|\mathbf{f}) = \mathcal{N}(\mathbf{f}_*|\mu, \Sigma), \quad \mu = \mathbf{K}_{*,\mathbf{f}}\mathbf{K}_{\mathbf{f},\mathbf{f}}^{-1}\mathbf{f}, \quad \Sigma = \mathbf{K}_{*,*} - \mathbf{K}_{*,\mathbf{f}}\mathbf{K}_{\mathbf{f},\mathbf{f}}^{-1}\mathbf{K}_{\mathbf{f},*} \qquad (2)$$

1. Linear kernel:

$$k(x, y) = \sum_{i=1}^{\text{input\_dim}} \sigma_i^2 x_i y_i$$

where *k(x,y)* = *k*(x_i,x_j), σ^2 = 1 – variance,

dimensionality of input vector x is (100 x 1)

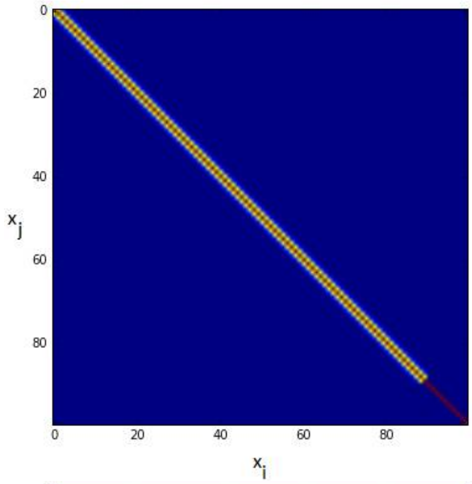


2. RBF - Radial Basis Function kernel:

$$k(r) = \sigma^2 \exp\left(-\frac{1}{2}r^2\right)$$

where *k(r)* = *k*(x_i,x_j), $r^2 = x_i^2 + x_j^2$, σ^2 = 1 – variance

dimensionality of input vector x is (100 x 1)



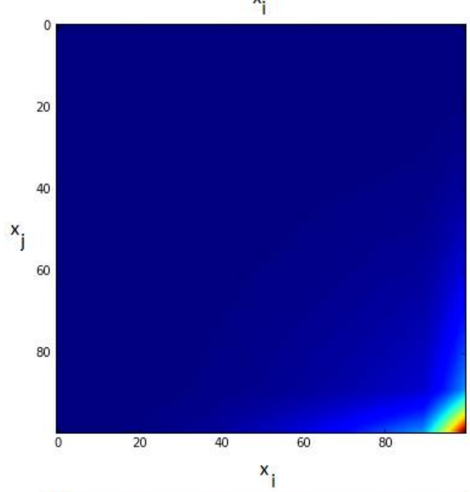
3. Poly - Polynomial kernel:

$$K(x, y) = (x^\top y + c)^d$$

where *K(x,y)* = *k*(x_i,x_j),

c = 0 – trading off parameter, d = 2 – power,

dimensionality of input vector x is (100 x 1)



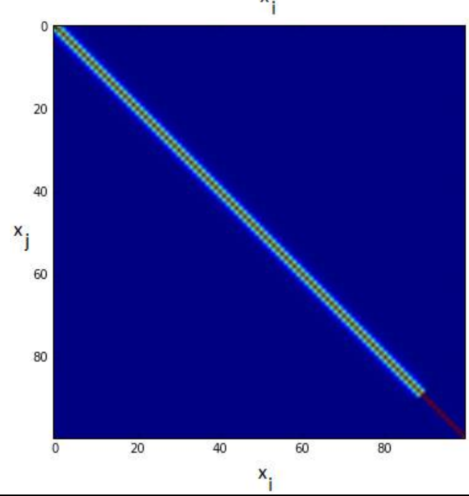
4. RatQuad – Rational Quadratic kernel:

$$k(r) = \sigma^2\left(1 + \frac{r^2}{2}\right)^{-\alpha}$$

where *k(r)* = *k*(x_i,x_j), $r^2 = x_i^2 + x_j^2$,

σ^2 = 1 – variance, α = 2 – power,

dimensionality of input vector x is (100 x 1)



5. MLP - Multi layer perceptron kernel:

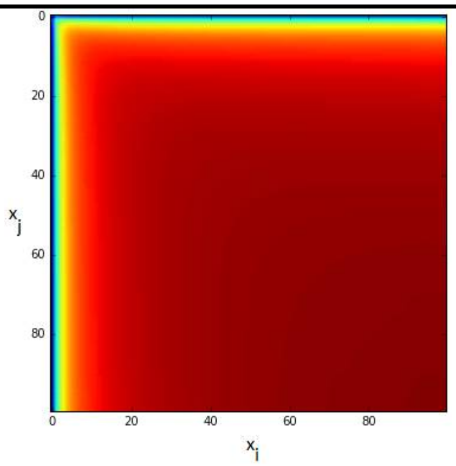
$$k(x,y)=\sigma^2\frac{2}{\pi}\text{asin}\left(\frac{\sigma_w^2 x^\top y+\sigma_b^2}{\sqrt{\sigma_w^2 x^\top x+\sigma_b^2+1}\sqrt{\sigma_w^2 y^\top+y\sigma_b^2+1}}\right)$$

where $k(x,y) = k(x_i,x_j)$, $x^T y$ corresponds to x_i x_j,

$\sigma^2 = 1$ – variance, $\sigma_w^2 = 1$ – weight variance,

$\sigma_w^2 = 1$ – bias variance,

dimensionality of input vector x is (100 x 1)



6. Matern32 - Matern 3/2 kernel:

$$k(r)=\sigma^2(1+\sqrt{3}r)\exp(-\sqrt{3}r)\quad \text{where } r=\sqrt{\sum_{i=1}^{\text{input\_dim}}\frac{(x_i-y_i)^2}{\ell_i^2}}$$

where $k(r) = k(x_i,x_j)$, $r = \text{sqrt}(x_i^2+x_j^2)$,

$\sigma^2 = 1$ – variance

dimensionality of input vector x is (100 x 1)

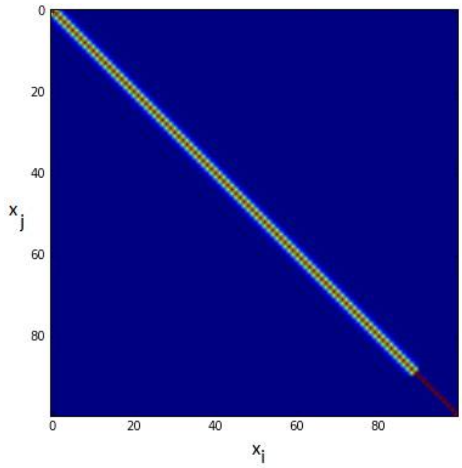


Fig. 4. Classical kernels and theirs covariance matrices

The example of the result of such process can be seen on the Fig. 5. As we can see from the plot all the random lines intersect at two points (40,1) and (80,1) which are our conditions. It means that we can fix some values of a Gaussian process, and that the resulting process is still Gaussian. This property is widely used in Gaussian process.

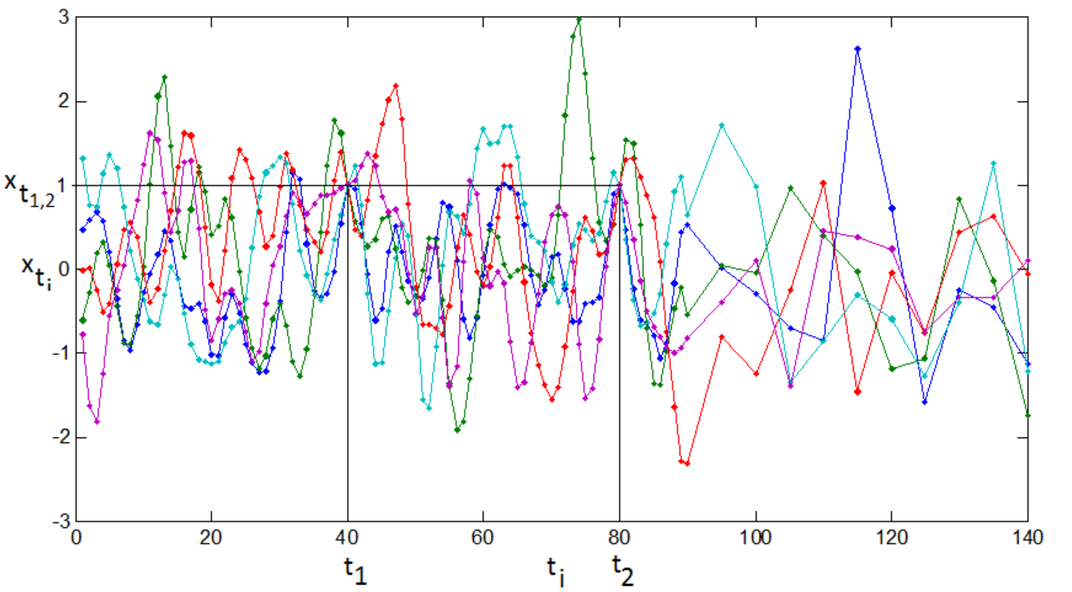


Fig. 5. 5 realizations of multivariate conditional Gaussian distribution

2. Gaussian process latent variables model

Until now, we considered classical Gaussian processes, where X is a multivariate Gaussian process indexed by time t. Gaussian process latent variable model (GPLVM) is a new class of models which consists of Gaussian process mappings from a latent space $X \in R^Q$ to an observed data-space Y $\in R^D$, where Q << D, X, Y – multivariate Gaussian processes, through a set of parameters [6].

In a simplest case the output dimensions of Gaussian process latent variable model are a priori assumed to be linear, independent and identically distributed. In this case it can be interpreted as a probabilistic version of PCA.

Probabilistic PCA (PPCA) is a latent variable model in which the maximum likelihood solution for the parameters is found through solving an eigenvalue problem on the data's covariance matrix. The relationship between the latent variable and the data point is linear with noise added. This relationship can be expressed by (3), where the matrix $W \in R^{DxQ}$ specifies the linear relationship between the latent-space and the data space and the noise values, $\eta_n \in R^{Dx1}$, are taken to be an independent sample from a spherical Gaussian distribution with mean zero and covariance β^{-1} I (4) [6].

$$\mathbf{y}_n = \mathbf{W}\mathbf{x}_n + \eta_n \quad (3)$$

$$p(\eta_n) = N(\eta_n | \mathbf{0}, \beta^{-1}\mathbf{I}) \quad (4)$$

The likelihood for a data point can then be written as (5). To obtain the marginal likelihood we integrate over the latent variables (6), which requires us to specify a prior distribution over x_n. For probabilistic PCA the appropriate prior is a unit covariance, zero mean Gaussian distribution (7) [6].

$$p(\mathbf{y}_n | \mathbf{x}_n, \mathbf{W}, \beta) = N(\mathbf{y}_n | \mathbf{W}\mathbf{x}_n, \beta^{-1}\mathbf{I}) \quad (5)$$

$$p(\mathbf{y}_n | \mathbf{W}, \beta) = \int p(\mathbf{y}_n | \mathbf{x}_n, \mathbf{W}, \beta)\, p(\mathbf{x}_n)\, d\mathbf{x}_n \quad (6)$$

$$p(\mathbf{x}_n) = N(\mathbf{x}_n | \mathbf{0}, \mathbf{I}) \quad (7)$$

The marginal likelihood for each data point can then be found analytically (through the marginalization in (6)) as (8). Taking advantage of the independence of the data points, the likelihood of the full data set is given by (9) [6].

$$p(\mathbf{y}_n | \mathbf{W}, \beta) = N(\mathbf{y}_n | \mathbf{0}, \mathbf{W}\mathbf{W}^T + \beta^{-1}\mathbf{I}) \quad (8)$$

$$p(\mathbf{Y} | \mathbf{W}, \beta) = \prod_{n=1}^{N} p(\mathbf{y}_n | \mathbf{W}, \beta) \quad (9)$$

The parameters W can then be found through maximizing (9). Marginalizing the latent variables and optimizing the parameters via maximum likelihood is a standard approach for fitting latent variable models [6].

3. Dynamical systems

Consider a system of autonomous, first-order, linear difference equations (10) where the state variable x_t is an n-dimensional vector, $x_t \in R^n$, i,j = 1, 2, ..., n, and B is a n dimensional column vector of constant parameters, $B \in R^n$. The initial value of the state variable x0 is given.

$$x_{t+1} = Ax_t + B, \qquad t = 0, 1, 2, \cdots, \infty \quad (10)$$

A solution to the multi-dimensional linear system (10) is a trajectory of the vector x that satisfies this equation at any point in time and relates the value of the state variable at time t, x_t to the initial condition x_0 and the set of parameters embodied in the vector B and the matrix A. Given the value of the state variable at time 0, x_0, the method of iterations generates a pattern that constitutes a general solution (11) [11].

$$\begin{aligned} x_1 &= Ax_0 + B; \\ x_2 &= Ax_1 + B = A(Ax_0 + B) + B = A^2x_0 + AB + B; \\ x_t &= A^t x_0 + A^{t-1}B + A^{t-2}B + ... + AB + B = A^t x_0 + \sum_{i=0}^{t-1} A^i B \end{aligned} \quad (11)$$

In this work dynamical texture modeling consists of two steps: dimensionality reduction and dynamic texture learning. They can be expressed as (12).

$$\begin{aligned} x_{t+1} &= f(x_t, A) + n_{x,t} \\ y_t &= g(x_t, B) + n_{y,t} \end{aligned} \quad (12)$$

The first line of (12) is dynamical system such as presented below, where x_t - latent variable which affects dynamic behavior, $x_t \in R^Q$, and instead of B in (10) $n_{x,t}$ in (12) - represent the noise, f() - dynamic modelling function R^Q–>R^Q , A - input parameters for function f(). Second line of (12) represents dimensionality reduction stage, where y_t - column vector unfolded from the frame at time t, $y_t \in R^D$, D – large, Q<<D , g() - dimensionality reduction function R^D–>R^Q, B - input parameters for function g(), and $n_{y,t}$ - represents the noise [8]. If f() is linear such as in equation (10) dynamical system (12) is called linear, but in this work it is not the case, because most of dynamic textures are not linear.

4. Gaussian process dynamic model

The Gaussian process dynamic model (GPDM) is a latent variable model. It comprises a generative mapping from a latent space X to the observation space Y and a dynamical model in the latent space (Fig. 6). These mappings are, in general, nonlinear.

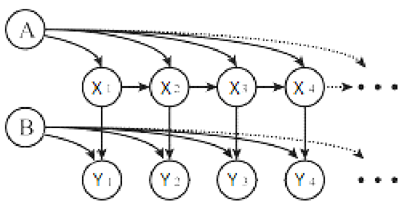


Fig. 6. Dynamical texture model

The GPDM is obtained by marginalizing out the parameters of the two mappings and optimizing the latent coordinates of training data [8].

In this work it is assumed that the dynamic texture sequence Y_i is a multivariate Gaussian process indexed by X_i expressed by likelihood function (13), where Y – observed dynamic texture sequence vector, X – latent variable vector, K_Y – kernel matrix of latent mapping, D – dimensionality of original sample, N – number of frames in original sample [1].

$$p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\theta}) = \prod_{t=1}^{N} p(y_t|x_t,\boldsymbol{\theta}) = \frac{1}{(2\pi)^{DN/2}|\boldsymbol{K}_Y|^{D/2}}\exp\left(-\frac{1}{2}\mathrm{tr}\left(\boldsymbol{K}_Y^{-1}\boldsymbol{Y}\boldsymbol{Y}^T\right)\right) \quad (13)$$

To achieve nonlinear mapping special squared exponential covariance function (14) is used, where θ_i – hyperparameters, $\delta_{xi,xj}$ – Kroneker delta function.

$$K_Y = k_Y(x_i,x_j) = \theta_1 \exp(-\frac{\theta_2}{2}(x_i - x_j)(x_i - x_j)^T) + \theta_3\delta_{x_i,x_j} \quad (14)$$

At the same time, the latent dynamic behaviour changes among different types of dynamic textures. The second kernel function K_X used in Gaussian process detects the dynamic behaviour of the latent variables. As for the kernel function itself, one of the functions from Fig. 4 can be taken. This second kernel comprises a generative mapping of dynamical model in the latent space, which defines this model dynamic.

5. Multi-kernel Gaussian process dynamic model

For dynamic texture modeling, the latent dynamic behavior varies greatly among different types of dynamic texture. Thus, it is very difficult to design the most suitable kernel for a dynamic texture empirically. To overcome this problem, a multi-kernel Gaussian process dynamic model (MK-GPDM) for dynamic texture modeling is proposed [1].

In this model for the mapping of dynamic texture in the latent space kernel function (15) is proposed.

$$K_X = k_X(x_i,x_j) = \sum_{l=1}^{M} w_l k_l(x_i,x_j) + w_\delta \delta_{x_i,x_j} \quad (15)$$

K_X denotes the constructed kernel matrix generated by the multi-kernel function, where W – vector of weights of kernel functions K = k_l, l ∈ [1,M], M – number of different kernel functions used.

In this case the likelihood function extends to (16), where Q – latent dimensionality, λ – set of parameters of kernels used in K_X.

$$p(\boldsymbol{X}|\lambda,\boldsymbol{W}) = p(x_1)\prod_{t=2}^{N} p(x_t|x_{t-1},\lambda,\boldsymbol{W}) = p(x_1)\frac{1}{(2\pi)^{Q(N-1)/2}|\boldsymbol{K}_X|^{Q/2}}\exp\left(-\frac{1}{2}\mathrm{tr}\left(\boldsymbol{K}_X^{-1}\boldsymbol{X}_{2:N}\boldsymbol{X}_{2:N}^T\right)\right) \quad (16)$$

6. Mean-prediction method for state space models

The goal of the model proposed in this work is to generate new video data using learned dynamic texture. It can be done by estimating necessary parameters (latent variable vector, observed dynamic texture vector, kernel matrix mapping hyperparameters, weights for kernel functions and different kernel parameters) and then by predicting new sequence of dynamic texture.

State space models is a class of models where we do not have temporal latent function, however, we focus on transition function between latent states instead. MK-GPDM is an example of such a model. As an approach to synthesize new data the adopted mean-prediction method based on first-order Markov model using Gaussian prediction is used in it. In the prediction process, the latent variable x_t is set to be the mean point given by the previous time index as (17) [1].

$$x_t = \mu_X(x_{t-1}) \quad (17)$$

The mean point itself computes using equation (18) [1]. According to our model x_{t-1} must be used instead of X in (18), while $X_{2:N}$ and K_X remain the same.

$$\mu_X(x) = \mathbf{X}_{2:N}^T \mathbf{K}_X^{-1} \mathrm{k}_X(x) \quad (18)$$

The new video sequence is then generated by (19), where mean and covariance for y_t can be found throw (20), vector containing $k_Y(x,x_i)$ in the i-th entry [8].

$$y_t \sim \mathcal{N}\left(\mu_Y(x_t), \sigma_Y^2(x_t) I\right) \quad (19)$$

$$\begin{aligned} \mu_Y(\mathbf{x}) &= \mathbf{Y}^T \mathbf{K}_Y^{-1} \mathbf{k}_Y(\mathbf{x}), \\ \sigma_Y^2(\mathbf{x}) &= k_Y(\mathbf{x}, \mathbf{x}) - \mathbf{k}_Y(\mathbf{x})^T \mathbf{K}_Y^{-1} \mathbf{k}_Y(\mathbf{x}) \end{aligned} \quad (20)$$

7. Variational Gaussian process dynamical system

Variational Gaussian process dynamical systems (VGPDS) can be seen as a supervised version of the Bayesian GP-LVM. Bayesian GP-LVM is an extension of the traditional GP-LVM where the latent space is approximately marginalized out in a variational fashion [12].

In this model we do not make a strong assumptions about the functional form of the latent functions. Instead we infer them in a fully Bayesian non-parametric way using Gaussian processes. We assume that X is a multivariate Gaussian process indexed by time T and f is a different Gaussian process indexed by X [12]. They can be expressed as (21)

$$\begin{aligned} &X(t) \sim GP(0, K_x(t_i,t_j)) \\ &f(x) \sim GP(0, K_f(x_i,x_j)) \end{aligned} \quad (21)$$

The first line in (21) describes dynamic behavior by assuming temporal latent function $X(t) \in R^Q$, where $Q << D$, while the second represents mapping from latent space X to observed space Y. Covariance functions are represented by $K_x(t_i,t_j)$ and $K_f(x_i,x_j)$ with own parameters.

We assume independence over the observed space Y, matrix F denoted latent variables mapping and latent dimensions X given time t. The joint probability model is described by (22)

$$P(Y, F, X|t) = P(Y|F)P(F|X)P(X|t) = \prod_{d=1}^{D} P(y_d|f_d)P(f_d|X) \prod_{q=1}^{Q} P(x_q|t) \quad (22)$$

In order to simplify calculations and minimize the time for optimization we include M extra samples of the GP latent mapping f, known as inducing points, so that $u_m \in R^D$ is such a sample. The inducing points are evaluated at a set of pseudo-inputs $X \in R^{M \times Q}$.

$$P(Y, F, U, X'|t) = \prod_{d=1}^{D} P(y_d|f_d)P(f_d|u_d, X)P(u_d|X')P(X|t) \quad (23)$$

The augmented joint probability density takes the form (23), where $P(u_d|X')$ is a zero-mean Gaussian with a covariance matrix K_{MM} constructed using the same function as for the GP prior $K_{NN} = K_f(X,X)$:

$$P(f_d|u_d, X) = N(f_d|K_{NN}K_{MM}^{-1}u_d, K_{NN} - K_{NN}K_{MM}^{-1}K_{MN}) \quad (24)$$

By dropping X' from our expression, we write the augmented GP prior analytically as (24) [12].

8. Mean-prediction method for temporal latent models with inducing points

Prediction of a new dynamic texture sequence for temporal latent models with inducing points such as VGPDS is different from one described below. We can not use mean prediction method for state space models here, simply because using temporal latent function we do not capture relationship between frames and, therefore, we can not use this information to synthesize new data.

The prediction of new observed texture consists of two main parts: prediction of new latent texture and prediction of new observed texture itself. For VGPDS first part is done by using the equation (25), which is derived in [12] in the appendix section, where Xs_{ynt} is new latent texture obtained using given time series t and new series t_{synt} with covariance function K and known latent space X.

$$X_{synt} = K(t_{synt}, t)K(t, t)^{-1}X \quad (25)$$

The second part is derived from the information provided by [12] using quantities Ψ_i in the appendix part.

$$\begin{aligned}
Y_{synt} &= E(Y^*) = E(F^*) = B^T\psi_1^* \\
B &= \beta(K_{MM} + \beta\psi_2)^{-1}\psi_1^T Y \\
\psi_1^* &= K_{M*} = K(X_u, X_{synt}) \qquad (26) \\
\psi_1 &= K_{NM} \\
\psi_2 &= K_{MN}K_{NM}
\end{aligned}$$

In (26) the derivation is shown, where β represents the error of the model. The final equation for prediction the new observed texture using inducing points is presented in (27).

$$Y_{synt} = ((K(X_u) + K(X_u, X)K(X, X_u))^{-1}K(X, X_u)^T Y)^T K(X_u, X_{synt}) \quad (27)$$

The results of using such a method is shown in the results part of this paper.

Methodological contribution

My contribution to existing research [1] consists of several parts:

1. GP linear model implementation in Matlab

First of all, the implementation of Gaussian process linear model described in first part of Survey of the state of the art was done in Matlab. It helps to get better understanding and visualize of what is Gaussian process.

2. Reimplementation of MK-GPDM in Python

The reimplementation of the method in Jupyter notebook in Python was chosen as the way to get better understanding of the whole model. This approach implies to understand many things used in the method. They are sequence of steps of learning algorithm, functions which must be optimized, their gradients along with dimensionality analysis.

Learning Multi-kernel Gaussian process dynamic model algorithm is presented in [1]. It can be generalized in order to get easiest understanding into two main steps. First, we fix weights vector W and perform optimization of function (28) with respect to latent variable X, vector of hyperparameters θ of kernel K_Y and vector λ of kernels parameters of kernel K_X with using Scaled conjugate gradient optimization method.

$$F(X,\theta,\lambda) = -lnP(X,\theta,\lambda|Y) = \frac{D}{2}ln|K_Y| + \frac{1}{2}tr(K_Y^{-1}YY^{-1}) + \frac{Q}{2}ln|K_X| + \frac{1}{2}tr(K_X^{-1}X_{2:N}X_{2:N}^T) + \sum_i \theta_i + \sum_{i,j}(\lambda_i)_j + C \quad (28)$$

This function represents posterior distribution of our model. It is obtained from (13) by using Bayesian inference rules.

In second step, we fix obtained X, θ and λ and optimize function (29) with respect to W using gradient descent method. We do not need to use function (28) any more, we can simplify it to function (29), because once we fix X, θ and λ they become constant vectors and we can drop them (the minimum of the function does not depend on constants).

$$F(W) = \frac{Q}{2}ln|K_X^{-1}| + \frac{1}{2}tr(K_X^{-1}X_{2:N}X_{2:N}^T) + \alpha\|W\|_2 \quad (29)$$

Due to the fact that functions (28) and (29) are not convex we must repeat these two main steps for I times, where I is fixed.

First kernel K_Y used for latent mapping (14) has dimensionality N by N. It can be easily derived as shown in (30).

x_i, x_j = 1 x Q, thus $(x_i - x_j)(x_i - x_j)^T$ = 1 x 1, while $\theta_1 exp(-\frac{\theta_2}{2}(x_i - x_j)(x_i - x_j)^T)$ = 1 x 1 and δ_{x_i,x_j} = 1 x 1

To sum up: $(K_Y)_{i,j}$ = 1 x 1, thus K_Y = N x N (30)

Kernel K_X used for dynamic modeling (15) has dimensionality N-1 by N-1. It can be easily checked as shown in (31).

$$(K_X)_{i,j} = \sum_{l=1..M} w_l k_l(x_i,x_j) + w_\delta \delta_{x_i,x_j},$$

$i,j = 1,\dots,N-1$, $k_l(x_i,x_j)$ = 1 x 1, δ_{x_i,x_j} = 1 x 1 (31)

To sum up: $(K_X)_{i,j}$ = 1 x 1, thus K_X = N-1 x N-1

To perform optimization all necessary gradients must be properly derived. General derivation is shown in Appendix A of [1].

To obtain gradient of function (28) with respect to X library functions for computing gradients for K_Y with respect to $X_{1:N}$ and K_X with respect to $X_{1:N-1}$ must be used. The dimensionality of resulting gradient is N by Q since K_X shows correlation between N-1 transitions of N latent frames.

To obtain gradient of function (28) with respect to θ library function for computing gradient for K_Y with respect to θ must be used. The dimensionality of resulting gradient depends on dimensionality of hyperparameters vector θ.

To obtain gradient of function (28) with respect to λ library function for computing gradient for K_X with respect to λ must be used. The dimensionality of resulting gradient depends on dimensionality of parameters vector λ.

$$\begin{aligned}
\frac{\partial F}{\partial W} &= \frac{\partial F}{\partial K_X}\frac{\partial K_X}{\partial W} + \frac{\partial F}{\partial W}, \text{ where} \\
\frac{\partial F}{\partial K_X} &= \frac{Q}{2}K_X^{-1} - \frac{1}{2}K_X^{-1}X_{2:N}X_{2:N}^T K_X^{-1} \\
\frac{\partial K_X}{\partial W} &= \sum_{l=1..M} k_l(x_i, x_j) \\
\frac{\partial F}{\partial W} &= \frac{\alpha W}{\|W\|_2}
\end{aligned} \qquad (32)$$

Gradient of function (29) with respect to W is derived in (32). The dimensionality of resulting gradient depends on dimensionality of parameters vector W.

3. Implementation of GPLVM in Python

Gaussian process latent variable model (GPLVM) has been implemented in Python as simpler version of MK-GPDM during the implementation of the full model. This model was used later to perform comparison of different models.

GPLVM models the joint distribution of the observed data and their corresponding representation in a low-dimensional latent space. Dimensionality of latent space was fixed to 20 as in Matlab implementation. For KX kernel construction combination of six kernel functions presented in Fig. 4 was chosen. As for optimization function Scaled conjugate gradient method was chosen as well. For synthesizing new sequence of latent variables mean prediction method for state space models is used along with multivariate Gaussian prediction for new dynamic texture synthesis.

4. Development and implementation of evaluation approaches

The research described in [1] does not provide any numerical criterion for quality measuring; the visual evaluation approach is used only. That is why several evaluation methods are tried in this work. They are mean square error, absolute mean value and missing frame estimation.

The mean square error (33) is a classical approach used in such kind of models. It provides rough numerical estimation of the error between two frames input Y and predicted one Y_{synt} with dimensionality n.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y - Y_{synt})^2 \qquad (33)$$

Absolute mean value (34) is a new criterion for such models. It is easier to interpret comparing to MSE. It shows average difference of intensity between between two frames input Y and predicted one Y_{synt}.

$$ABSM = mean(abs(Y - Y_{synt})) \quad (34)$$

Missing frames estimation method can be used as well. However, it is not possible to perform estimation throw conditions on Gaussian process. Conditions are used to bound what we want to expect from the result. If we want to get one exact frame at time t we can input it as a condition. This model does not work in the opposite way. Therefore, two different approaches for missing frame estimation was found.

First approach is simple. We split input texture Y with N frames into two parts: training set N-m frames and test set of m frames. We use training set to learn the model, then we synthesise N new frames. After that we can compare last m frames of the new texture Y_{synt} with last m frames with original texture Y.

Second, the optimization function (28) was changed to be able to work with N-1 observed frames of texture Y. The basic idea is to skip missing frame everywhere it is used. Moreover, the gradients for (28) were changed to make it possible to match dimensionalities and perform optimization. It gives opportunity to predict any frame in the middle of the input sequence Y and then to compare it with original one.

5. Test of VGPDS and different models comparison

Mean prediction method for temporal latent variables with inducing points has been implemented for VGPDS implementation given in Matlab. Moreover, three evaluation approaches described below (mean square error, absolute mean and missing last frames evaluation approach) were implemented too.

Series of experiments were performed to compare MK-GPDM model implemented in Matlab, GPLVM implementation in Python and VGPDS implementation given in Matlab. Test results can be found in the Results section of this paper.

Results

1. Matlab implementation properties

The implementation of MK-GPDM given in Matlab had several problems, which were fixed.

First is inexplicable instability. The program crashes quite often with SVD computation error, as a result you need to run it again using the same input and setting. It seems that one of the kernel matrices becomes not positive definite which bring to this error. The problem was fixed by performing normalization procedure using mean and standard deviation before the initialization step. It allows to work with less scales of numbers which prevents from crashes of matrix inverse operation.

Second, they did not use random initialization in the code, as a result all synthesized textures were the same. It was easily fixed in the beginning of the code, but it brings to another feature. It can be explained by the following setup. In this experiment I used sample video **straw.avi** from Dyntex database [9] pre-processed to grayscale 120x90 pixels 10 seconds length (250 frames). Index of original pre-processed video are given on Fig. 7.

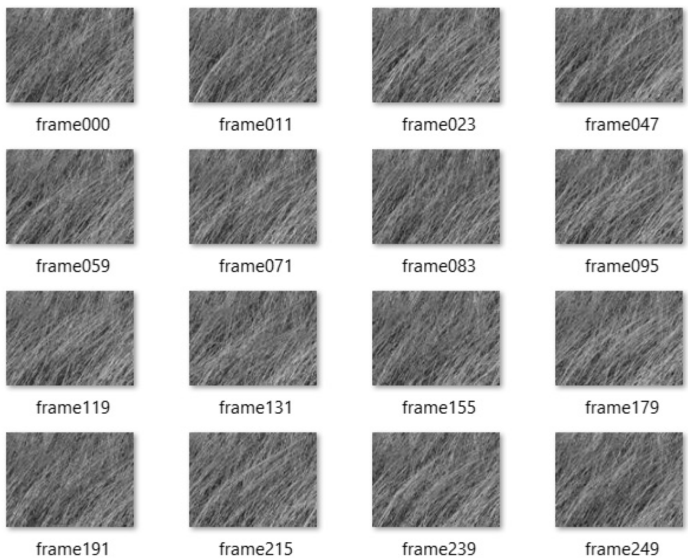


Fig. 7. Index of straw.avi sample video

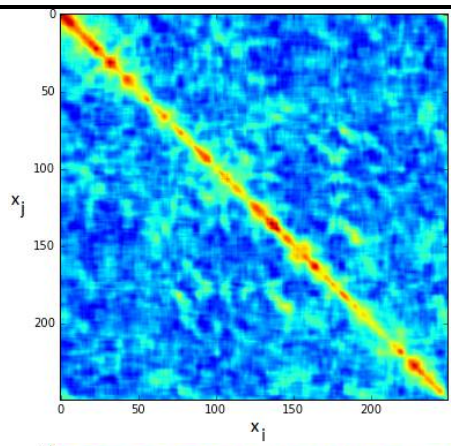
In this setting program was run several times. Weights obtained for kernels used for computing K_X are presented in Tab. 1, kernel functions used in the implementation with respect to input sample are presented in Fig. 8.

Attempt	**Linear**	**RBF**	**Poly**	**RatQuad**	**MLP**	**Matern32**
1	0.040278	0.682626	0	0	0	0.277096
3	0.067891	0.720069	0	0	0.212038	0
7	0.003638	0.368365	0	0.345235	0.069727	0.213032

Tab. 1. Weights of different kernels after optimization with respect to different attempts

As we can see from Tab. 1, weights learned for the texture were different for every attempt. It can be explained by the fact that random weights initialization is used in the code.

1. Linear kernel:

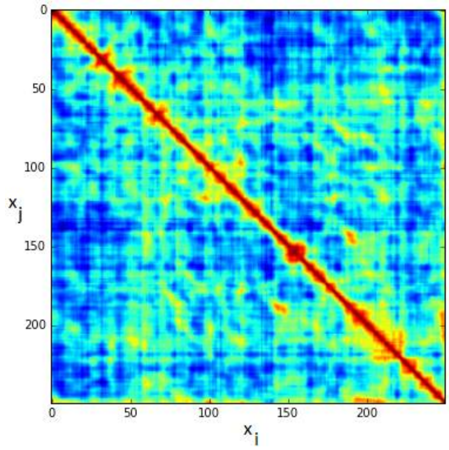


$$k(x,y) = \sum_{i=1}^{\text{input\_dim}} \sigma_i^2 x_i y_i$$

where $k(x,y) = k(x_i,x_j)$, $\sigma^2 = 1$ – variance,

dimensionality of input vector x is (250 x 20)

2. RBF - Radial Basis Function kernel:

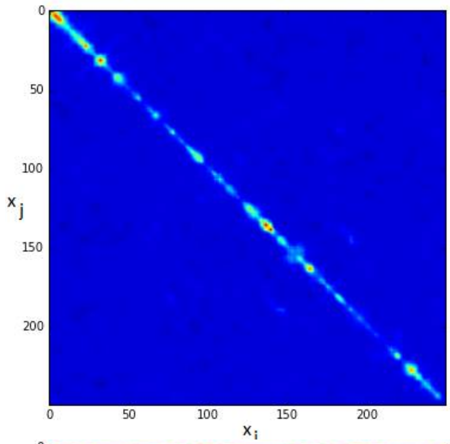


$$k(r) = \sigma^2 \exp\left(-\frac{1}{2}r^2\right)$$

where $k(r) = k(x_i,x_j)$, $r^2 = x_i^2+x_j^2$, $\sigma^2 = 1$ – variance

dimensionality of input vector x is (250 x 20)

3. Poly - Polynomial kernel:



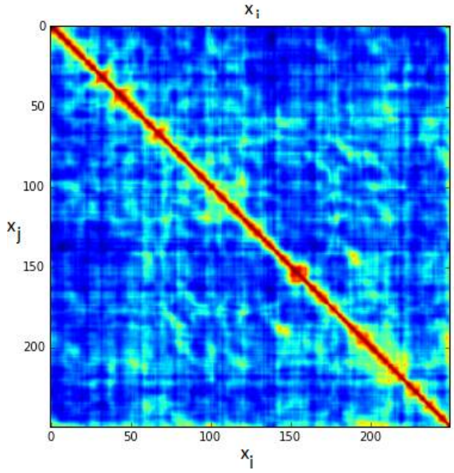
$$K(x,y) = (x^\top y + c)^d$$

where $K(x,y) = k(x_i,x_j)$,

c = 0 – trading off parameter, d = 2 – power,

dimensionality of input vector x is (250 x 20)

4. RatQuad – Rational Quadratic kernel:



$$k(r) = \sigma^2 \left(1 + \frac{r^2}{2}\right)^{-\alpha}$$

where $k(r) = k(x_i,x_j)$, $r^2 = x_i^2+x_j^2$,

$\sigma^2 = 1$ – variance, $\alpha = 2$ – power,

dimensionality of input vector x is (250 x 20)

5. MLP - Multi layer perceptron kernel:

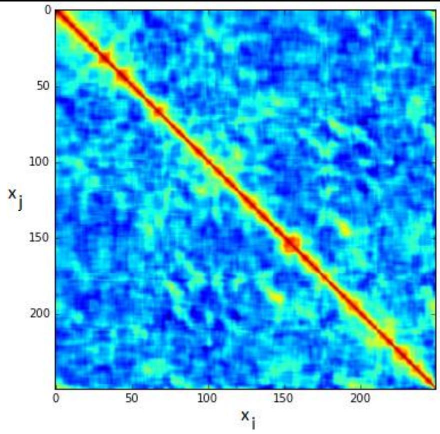
$$k(x,y) = \sigma^2 \frac{2}{\pi} \operatorname{asin}\left(\frac{\sigma_w^2 x^\top y + \sigma_b^2}{\sqrt{\sigma_w^2 x^\top x + \sigma_b^2 + 1}\sqrt{\sigma_w^2 y^\top + y\sigma_b^2 + 1}} \right)$$

where $k(x,y) = k(x_i,x_j)$, $x^T y$ corresponds to x_i x_j,

$\sigma^2 = 1$ – variance, $\sigma_w^2 = 1$ – weight variance,

$\sigma_w^2 = 1$ – bias variance,

dimensionality of input vector x is (250 x 20)



6. Matern32 - Matern 3/2 kernel:

$$k(r) = \sigma^2(1 + \sqrt{3}r)\exp(-\sqrt{3}r) \quad \text{where } r = \sqrt{\sum_{i=1}^{\text{input\_dim}} \frac{(x_i - y_i)^2}{\ell_i^2}}$$

where $k(r) = k(x_i,x_j)$, $r = \mathrm{sqrt}(x_i^2+x_j^2)$,

$\sigma^2 = 1$ – variance
dimensionality of input vector x is (250 x 20)

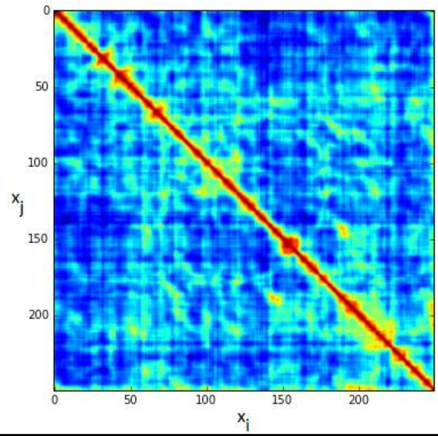


Fig. 8. Kernels and theirs covariance matrices with respect to latent variable X

Moreover, as we can see from Fig. 8, most of the kernels are generalizations of others and we do not really need some of them. That is why some kernels such as Linear, RBF and Matern32 can be safely removed from the implementation.

2. Python implementation properties

The reimplementation of full MK-GPDM in Python is not finished due to the problems with optimization. The main problem is to find a library which can provide kernel functions along with theirs gradients and perform optimization at the same time. It could be interesting to finish this implementation and to compare the results with Matlab implementation.

GPLVM implementation is done. To define a model, to build kernel function for latent mapping and to perform optimization GPy Gaussian process library [4] is used.

GPLVM is not a dynamical model. In practice, it means that there is no separate kernel function K_Y for dynamic modeling. Moreover, it assumes that data is generated independently without taking into account its temporal structure. However, the results are quite surprising. This method takes much more time for optimization comparing to Matlab implementation, but it is able to capture and generate new sequences of periodic things in dynamic textures.

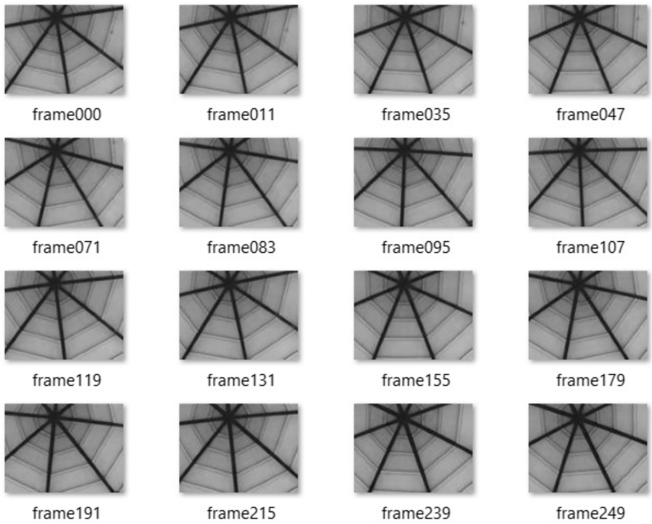


Fig. 9. Index of sunshade.avi sample video

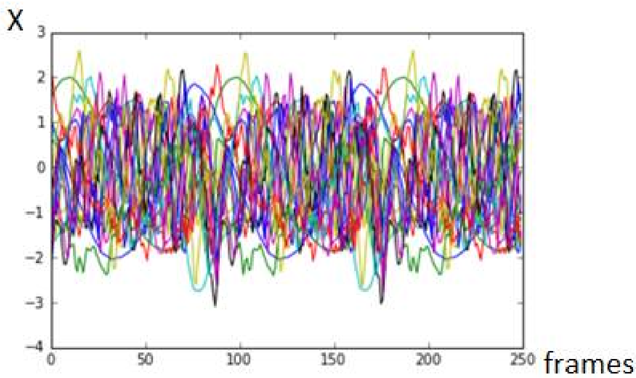


Fig. 10. New latent texture generated from the given input

For the experiments sample video **sunshade.avi** from Dyntex database [9] pre-processed to grayscale 120x90 pixels 10 seconds length (250 frames) was used. Index of original pre-processed video are given on Fig. 9. The result is presented in Fig. 10. Some periodicity can be seen in the figure.

The main drawback of this approach is that it takes a lot of time to perform optimization (around 1 hour for 10000 iterations for better result using laptop with i7 CPU). Moreover, in spite of the fact that the visible result is good in general, the transitions between periods are still recognizable.

3. Different models comparison

The mean square error, absolute mean and missing frame evaluation approaches described in Methodological contribution section were implemented for MK-GPDM, GPLVM and VGPDS models which allowed to perform comparison of these models in terms of accuracy. First, last missing frames evaluation method was tested on two different inputs: real and artificial one. They are **straw.avi** and **sunshade.avi** from Dyntex database [9] pre-processed to grayscale 120x90 pixels 10 seconds length (250 frames). Index of original pre-processed videos are given on Fig. 7 and 9. The results are presented in Tab. 2 and Fig. 11.

Video/ model	VGPDS				MK-GPDM				GPLVM			
	MSE			abs	MSE			abs	MSE			abs
	min	*mean*	*max*	*mean*	*min*	*mean*	*max*	*mean*	*min*	*mean*	*max*	*mean*
straw	296.9	310.8	333.8	14	393.4	439.2	502.05	16	849.7	926.9	991.6	24
sunshade	417.1	660.8	875.5	17	2165	2208.7	2265.3	36	2010	2050	2099	29

Tab. 2. Last 10 frames estimation in different models with two different samples

As we can see from Tab. 2 VGPDS provides better accuracy for both samples artificial and real one, while GPLVM is able to synthesise real texture better than MK-GPDM, but MK-GPDM provides better result than GPLVM for artificial texture conversely.

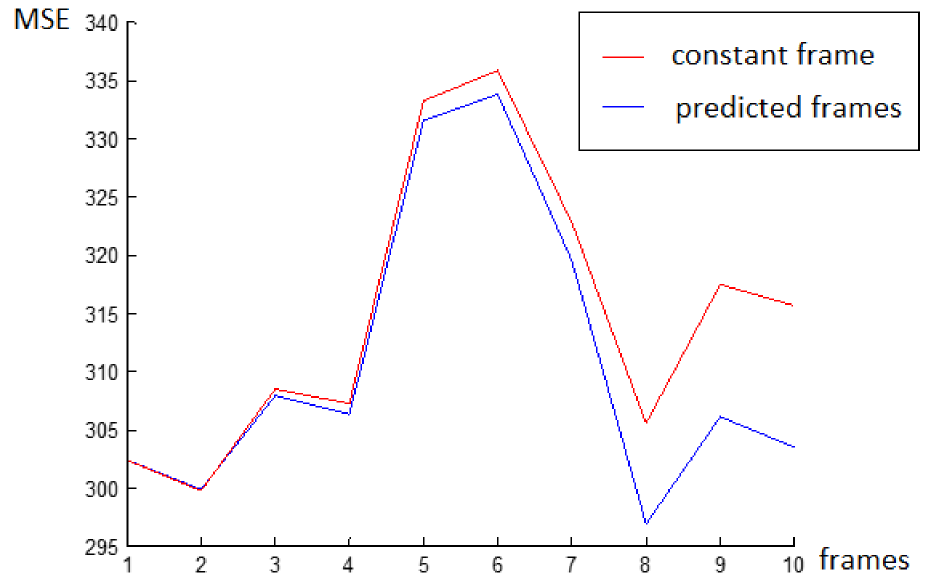
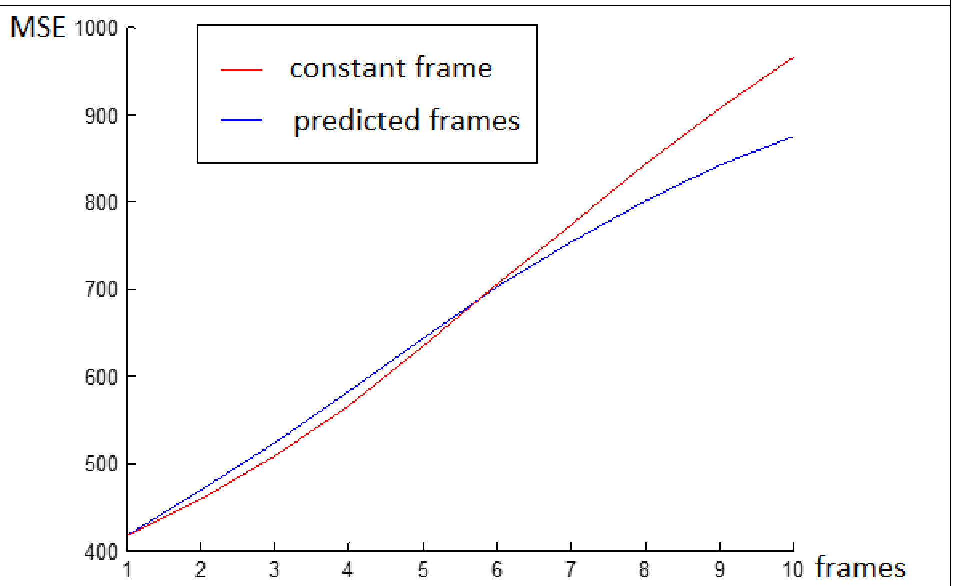
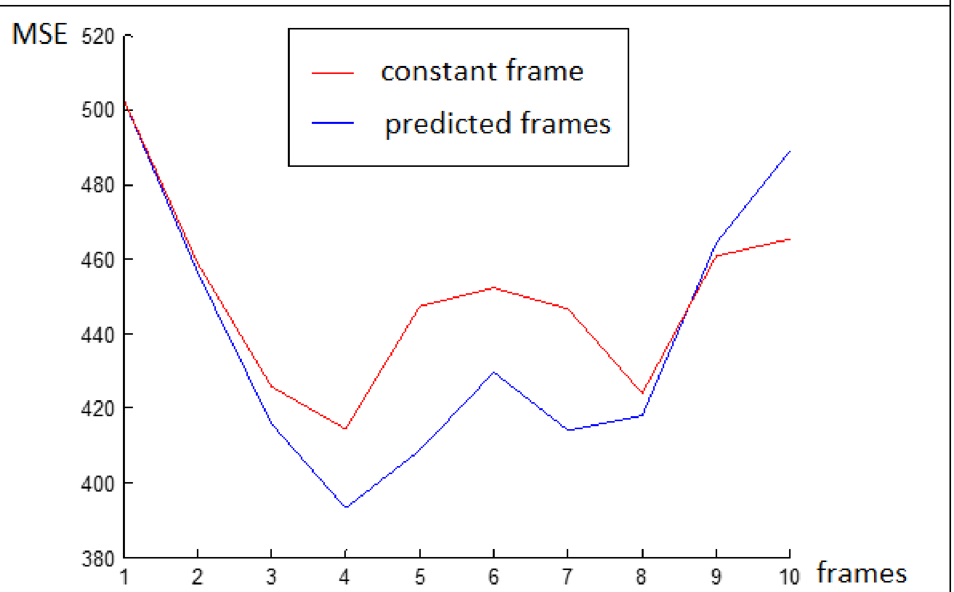
As we can see from Fig. 11 VGPDS and GPLVM at some frame start to provide less error comparing with last constant frame. It tells that it has some sense in performing such an estimation method. However, MK-GPDM provides quite contradictory result – the error is more than for the constant frame. It does not tell anything, because the visual result is little better than just static texture.

Another estimation approach described in Methodological contribution section was implemented for MK-GPDM. It is based on ignoring a missing frame in the middle of dynamic texture and allows to reconstruct it during the optimization. An experiment was performed to compare both of the methods: middle frame estimation and last one. The same two test videos were used: **straw.avi** and **sunshade.avi** from Dyntex database [9] pre-processed to grayscale 120x90 pixels 10 seconds length (250 frames). Index of original pre-processed videos are given on Fig. 7 and 9. The results of the experiment are presented in Tab. 3.

video/model	MK-GPDM			
	i=100 frame		*last frame*	
	MSE	abs mean	MSE	abs mean
straw	275.06	13	355.198	15
sunshade	180.32	11	1449.63	29

Tab. 3. Two different approaches of missing frame estimation in MK-GPDM

As we can see from Tab. 3 middle frame estimation approach is more accurate for both of input samples, but sometimes it does not tell anything, because the visual quality can be quite similar in spite of the value of error. For example, there are two frames on Fig. 12 original and predicted for both of the samples. They are different, but still are quite similar to each other. The MSE and abs mean is huge for both frames, especially for artificial one.

<table>
<tr><td rowspan="2">VGPDS</td><td>straw</td><td>
</td></tr>
<tr><td>sunshade</td><td>
</td></tr>
<tr><td>MK-GPDM</td><td>straw</td><td>
</td></tr>
</table>

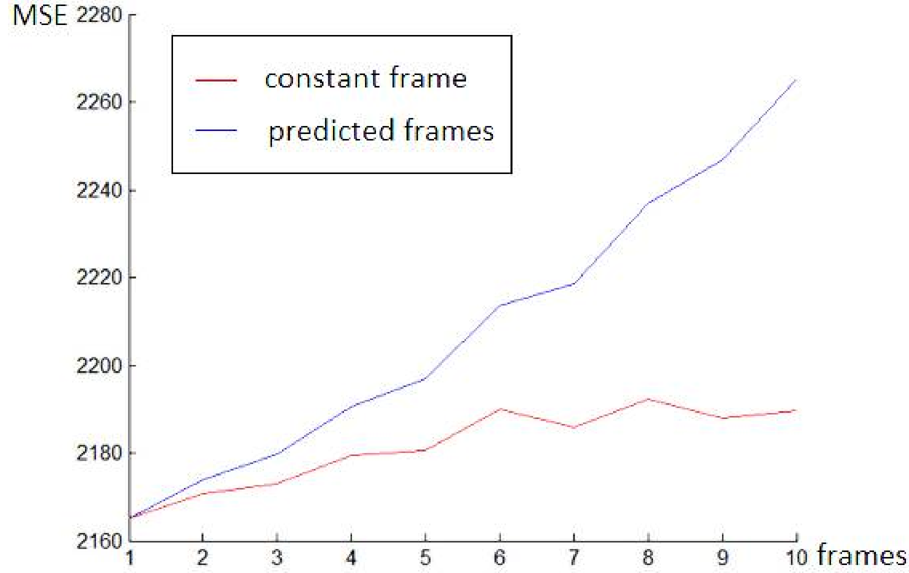
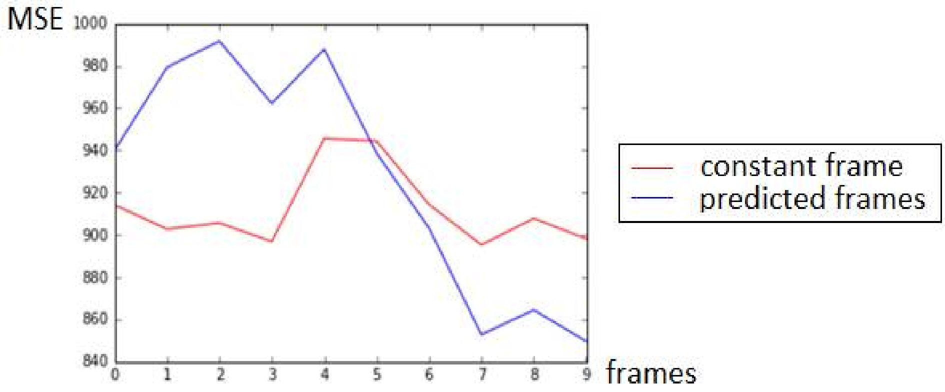
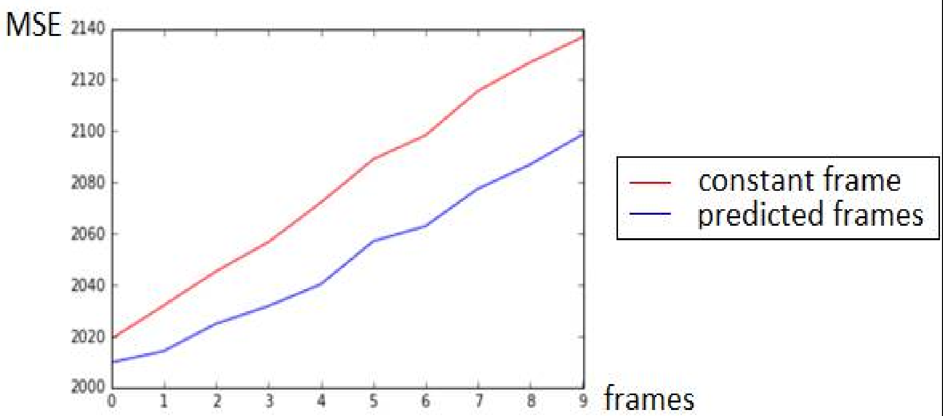
<table>
<tr><td></td><td>sunshade</td><td>
</td></tr>
<tr><td rowspan="2">GPLVM</td><td>straw</td><td>
</td></tr>
<tr><td>sunshade</td><td>
</td></tr>
</table>

Fig. 11. Last 10 frames estimation in different models with two different sample. Comparison with last constant frame

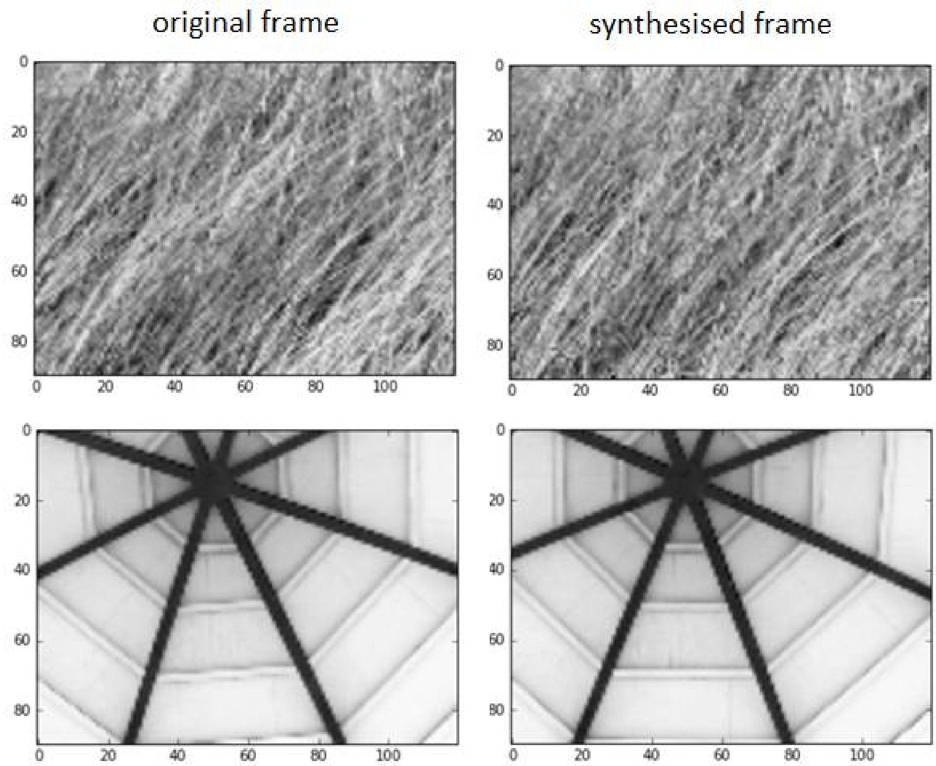


Fig. 12. Comparison of original frame and predicted one for 2 different samples

To sum up, there is no universal approach for comparison and quality measuring. However, it is good to use several of them for getting a better picture of the whole problem. And in case that some of them are not possible to use, for example visual method, others can be used as a good heuristic specially for Computer vision applications.

Conclusion and perspectives

Multi-kernel Gaussian process dynamic model is studied quite well and its implementation in Matlab is tested in different settings and angles. The implementation had some problems, which were fixed. The same model in Python was not finished due to some problems with implementation, however the simpler version, GPLVM, was implemented in Python and provides quite interesting results. It could be interesting to finish this reimplementation and to compare the results.

The given model did not provide any evaluation criteria for numerical quality measuring. Several evaluation methods were implemented: mean square error, absolute mean and missing frame evaluation. The results are not surprising. There is no universal method which can show the same result as a visual approach. It could be interesting to continue the research in this direction.

At last, three different methods were tested and compared using mentioned approaches: MK-GPDM, GPLVM and VGPDS. GPLVM has quite good results for its simple model, while VGPDS seems the best. MK-GPDM is less predicted comparing to others and too complex. It might be interesting to implement another model, which is better than all mentioned in this work.

Bibliography

1. Z. Zhu, et al., Dynamic texture modeling and synthesis using multi-kernel Gaussian process dynamic model, Signal Processing (2015), http://dx.doi.org/10.1016/j.sigpro.2015.10.025
2. R. Mourya, Modeling and Analyzing Dynamic Color Texture, Master Thesis Report, Université Jean Monnet, Saint-Etienne, France, 2012
3. N. Lawrence, Introduction to Bayesian Methods and Gaussian Process Regression, Conference on Computer Vision and Pattern Recognition CVPR 2012 - Providence, Rhode Island, USA, http://www.cs.toronto.edu/~urtasun/tutorials/GP_tutorial.html
4. N. Lawrence, Gaussian Process models with GPy, GP Summer School – Sheffield, 10-13th of June 2013, http://gpss.cc/gpss13/
5. N. Lawrence, Dimensionality Reduction using Gaussian Processes, GP Summer School – Sheffield, 10-13th of June 2013, http://gpss.cc/gpss13/
6. N. Lawrence, Probabilistic Non-linear Principal Component Analysis with Gaussian Process Latent Variable Models, Journal of Machine Learning Research 6 (2005) 1783–1816
7. G. Doretto, A. Chiuso, Y. N. Wu, S. Soatto, Dynamic textures, Int. J. Comput. Vis. 51 (2) (2003) 91–109, http://refhub.elsevier.com/S0165-1684(15)00368-0/sbref15
8. J.M. Wang, D.J.Fleet, A.Hertzmann, Gaussian process dynamical models for human motion, IEEE Trans. Pattern Anal. Mach. Intell. 30 (2) (2008) 283–298 http://refhub.elsevier.com/S0165-1684(15)00368-0/sbref33
9. R. Péteri, S. Fazekas, M. J. Huiskes, Dyntex: a comprehensive database of dynamic textures, PatternRecognit.Lett. 31 (12) (2010) 1627–1632, http://refhub.elsevier.com/S0165-1684(15)00368-0/sbref34
10. https://en.wikipedia.org/wiki/Gaussian_process
11. O. Galor, Discrete Dynamical Systems, Brown University, April 1, 2005
12. A. Damianou, M. Titsias, N. Lawrence, Variational Gaussian Process Dynamical Systems, Machine Learning, 2011, https://arxiv.org/pdf/1107.4985.pdf.