
Proceedings Paper:
Lyu, C., Liu, X. and Mihaylova, L. orcid.org/0000-0001-5856-2223 (2022) Efficient factorisation-based Gaussian process approaches for online tracking. In: Proceedings of the 2022 25th International Conference on Information Fusion (FUSION). 2022 25th International Conference on Information Fusion (FUSION), 04-07 Jul 2022, Linköping, Sweden. Institute of Electrical and Electronics Engineers . ISBN 9781665489416

https://doi.org/10.23919/FUSION49751.2022.9841257

Efficient Factorisation-based Gaussian Process Approaches for Online Tracking

Chenyi Lyu, Xingchi Liu and Lyudmila Mihaylova
Department of Automatic Control & Systems Engineering, University of Sheffield, S1 3JD, UK
Email: clyu5@sheffield.ac.uk, xingchi.liu@sheffield.ac.uk, l.s.mihaylova@sheffield.ac.uk

***Abstract*—Target tracking often relies on complex models with non-stationary parameters. Gaussian process (GP) is a model-free method that can achieve accurate performance. However, the inverse of the covariance matrix poses scalability challenges. Since the covariance matrix is typically dense, direct inversion and determinant evaluation methods suffer from cubic complexity to data size. This bottleneck limits the GP for long-term tracking or high-speed tracking. We present an efficient factorisation-based GP approach without any additional hyperparameters. The proposed approach reduces the computational complexity of the Cholesky decomposition by hierarchically factorising the covariance matrix into off-diagonal low-rank parts. Meanwhile, the resulting low-rank approximated Cholesky factor can also reduce the computation complexity of the inverse and the determinant operations. Numerical results based on offline and online tracking problems demonstrate the effectiveness of the proposed approach.**

***Index Terms*—Gaussian process, sensor networks, uncertainty quantification, factorisation, covariance matrix, hierarchical off-diagonal matrix, low-rank approximation, Cholesky factorisation, online tracking**

I. Introduction

Tracking an object's trajectory is a fundamental task for various applications including sea surveillance, autonomous vehicles, and traffic management. In order to improve tracking performance, a variety of model-based filtering approaches have been proposed such as [1]. These approaches, however, rely on well-defined motion models. In most applications, the actual target dynamic cannot be accurately captured by a model. Although the multiple-model method can capture complex behaviours, it suffers from high computational complexity which is not efficient when a large number of models are involved. In order to track multiple models, GP-based model-free methods can be used, which, as non-parametric methods, are capable of learning unknown functions directly from noisy data [2].

The use of Gaussian processes (GPs) is supported by efficient sampling algorithms, a rich methodological literature, and strong theoretical grounding [3], [4]. Unfortunately, due to the prohibitive computational and storage demands, exact GPs cannot be used in Bayesian models with more than a few thousand observations. It takes $\mathcal{O}(n^3)$ operations to calculate the inverse of the covariance matrix, where n is the number of input data sizes. Meanwhile, the storage of individual matrices scales at $\mathcal{O}(n^2)$, can quickly overwhelm the resources of most modern computers [5]. Since the covariance matrix is a symmetric matrix and the upper triangular matrix can easily solve the inverse and determinant. The Cholesky factorisation is widely used in the GP However, it still requires a cube computation cost for the factorisation itself.

This paper seeks to solve the online tracking problem. In the online tracking problem, the hyperparameters must be trained every step instead of only considering the prediction, since the input also changes. So there are more concrete challenges than the general $\mathcal{O}(n^3)$ complexity. In the optimization process of the GP, the inverting of the covariance matrix would be repeated until it finds the optimal value, thereby the computation complexity would be multiplied. GP would be unable to solve complex online tracking problems associated with big data.

There have been numerous approximation GP methods developed in the last decade to counteract these bottlenecks. There are two tactics to solve this bottleneck: global approximations, which distil the entire data, and local approximations, which divide the data for subspace learning [6]. In this paper, we focus on the global approximation. Subset-of-data is the most straightforward global approximation GP method. This method identifies the subset of training data to approximate the full kernel matrix [7]. However, the limited dataset produces an overconfident prediction variance due to the limited subset. Instead of selecting a subset of the training data, inducing points that are not limited to the existing dataset. The fully-independent training conditional [8] is the most popular inducing point method, which removes the dependence between the inducing data and the actual training data, allowing for a more stable result. After that, Titsias raised an elegant, sparse method based on variational inference and Kullback-Leibler divergence variational, which is called the free energy method [9], [10]. The computation cost of these methods is $\mathcal{O}(nm^2)$ where m is the number of the inducing points. Although inducing points solve the prediction quickly, this method adds the inducing points as extra parameters, increasing the workload when training the hyperparameters and making it easier to fall into the local optimum [8].

The other global approximation strategy is structure approximation. Wilson [11] raised a kernel interpolation for a scalable structured method to transform the matrix to the Kronecker or Toeplitz structure and reduce the computation cost on these structures. This method is a variant of the inducing point, which places the inducing point at the selected location.

Besides, several factorisation-based methods can directly work on the covariance matrix, such as Cholesky factorisation which can solve the dense symmetric matrix at $\mathcal{O}(1/3n^3)$

computation cost. Ambikasaran also raises a hierarchically factored into a product of block low-rank updates of the identity matrix [12].

In this paper, with the inspiration of hierarchical off-diagonal low-rank (HODLR) structure [12], [13] and Cholesky factorisation, we design a double factorisation method named hierarchical off-diagonal low-rank Cholesky factorisation Gaussian process (HCFGP). The HCFGP method offers a fast and reliable prediction of online tracking in the GP by boosting the Cholesky factorisation. For the HODLR matrix, all the dense parts are divided up into a few smaller components and remain at the diagonal of the matrix. Based on this characteristic, the HODLR structure can provide a very close approximation to the Cholesky factorisation method with only $\mathcal{O}(n\log^2 n)$ cost. The factor $\mathbf{L}$ can efficiently solve $\mathbf{A}^{-1}\mathbf{B}$ for $\mathbf{A}$ and $\mathbf{B}$ matrices,at $\mathcal{O}(n\log n)$ cost. The determinant $\det\left(\mathbf{K}+\sigma^2\mathbf{I}\right)$ can be calculated by the sum of the diagonal elements of Cholesky factor $\mathbf{L}$ matrix [12]. Here $\mathbf{I}$ denotes the identity matrix. The $\mathbf{K}$ matrix is a kernel matrix of the GP, σ - noise standard deviation and these will be introduced in detail in Section II.

The main contributions of this work are as follows: (i) The HODLR decomposition is applied to enhance the Cholesky decomposition step for the block diagonal matrices of the GP kernel. The HODLR factorisation enables the Cholesky decomposition only requires $\mathcal{O}(n\log^2 n)$, inverse in $\mathcal{O}(n\log n)$ operations and determinant, in $\mathcal{O}(n)$ operations; (ii) A comparison of computationally to other efficient factorisation based approaches for GP regression is presented; (iii) The performance of the considered algorithms is validated over target tracking in sensor networks. Results demonstrate reduction both the training and predicting time at a comparable level of accuracy with respect to the non-factorised GP algorithm.

The paper is organised as follows. Section II reviews fundamental knowledge about GP methods and related approximate GP efficiency acceleration methods. Section III describes the newly developed matrix factorisation for HODLR matrices. The following Section IV presents efficient GP solutions based on the HODLR method. Performance evaluation and validation results over tracking testing examples with data from sensor networks are presented in Section V. Section VI summarises the results and discusses future work.

II. Theoretical Background Knowledge

A. Overview of Gaussian Process

This section provides a brief overview of the GP regression method and the associated computational process for inference and learning process. In statistics, GP regression originally used in geostatistics is called Kriging and is a method of interpolation based on the GP governed by prior covariances [3]. Under suitable assumptions on the priors, GP regression provides the best linear unbiased prediction of the observations. From the algorithm perspective, a GP can also be assumed as a stochastic process used to map a nonlinear function from an input space to an output space. The problem of learning with GP is solved by learning the hyperparameters of the kernel function.

Assume that the training dataset $\mathcal{D}$ comprised of n input vectors $X=\{\mathbf{x}_1,\ldots,\mathbf{x}_n\}^\top$ and the observation vector $\mathbf{y}=(y(\mathbf{x}_1),\ldots,y(\mathbf{x}_n))^\top$ are given:

$$f\sim\mathcal{GP}(\bar{f}(\mathbf{x}),k(\mathbf{x},\mathbf{x}')). \tag{1}$$

$$y=f(\mathbf{x})+\varepsilon,\quad \varepsilon\sim\mathcal{N}\left(0,\sigma^2\right), \tag{2}$$

where $\bar{f}:\mathcal{X}\to\mathbb{R}$ is the mean function and $k:\mathcal{X}\times\mathcal{X}\to\mathbb{R}$ is the kernel function. The mean function is specified as $\mu(\mathbf{x})=\mathbb{E}[f(\mathbf{x})]$ where is often taken as 0. The kernel function controls the smoothness of GP specified as $k(\mathbf{x},\mathbf{x}')=cov\left(f(\mathbf{x})f(\mathbf{x}')\right)$. In (2), ε is the additive independent identically distributed Gaussian measurement noise and $\mathbb{E}[\cdot]$ is the mathematical expectation operation. The variance $\sigma^2\neq 0$ [3].

Then we define a GP model $\mathbf{f}_*$ which follows the Bayesian approach to predict the output $\mathbf{Y}_*$ for the new input X_* with the joint distribution $\mathbf{y}$ which can be written as

$$\begin{bmatrix}\mathbf{y}\\ \mathbf{f}_*\end{bmatrix}\sim\mathcal{N}\left(\mathbf{0},\begin{bmatrix}\mathbf{K}_{nn}+\sigma^2\mathbf{I} & \mathbf{K}_{nN}\\ \mathbf{K}_{Nn} & \mathbf{K}_{NN}\end{bmatrix}\right). \tag{3}$$

The prior mean is set up to be equal to zero, and the conditional distribution of (3) can be written as

$$\mathbf{f}_*\mid X,\mathbf{y},X_*\sim\mathcal{N}\left(\bar{\mathbf{f}}_*,\operatorname{cov}(\mathbf{f}_*)\right), \tag{4}$$

where

$$\bar{\mathbf{f}}_*=\boldsymbol{\mu}_{X_*}+\mathbf{K}_{X_*,X}(\mathbf{K}_{X,X}+\sigma^2\mathbf{I})^{-1}\mathbf{y} \tag{5}$$

$$\operatorname{cov}(\mathbf{f}_*)=\mathbf{K}_{X_*,X_*}-\mathbf{K}_{X_*,X}(\mathbf{K}_{X,X}+\sigma^2\mathbf{I})^{-1}\mathbf{K}_{X,X_*}. \tag{6}$$

From equations (5) and (6), the computational bottleneck to GP algorithms is to evaluate the hyperparameters and unknown function. The inverse of the covariance matrix $\mathbf{K}$ requiring $\mathcal{O}\left(n^3\right)$ operations and $\mathcal{O}\left(n^2\right)$ storage. So, the predictive mean and variance respectively cost $\mathcal{O}(n)$ and $\mathcal{O}\left(n^2\right)$ for each test point $\mathbf{X}_*$.

B. Adaptation of Hyperparameters

Maximum likelihood estimation for parameter-fitting given observations from a GP in space is a computationally-demanding task that restricts the use of such methods to moderately-sized data-sets. The hyperparameters θ of the kernel function $\mathbf{K}_{\boldsymbol{\theta}}$ are learned directly by maximizing the negative log marginal likelihood:

$$-\log p(\mathbf{y}\mid\boldsymbol{\theta})\propto\mathbf{y}^\top(\mathbf{K}_{\boldsymbol{\theta}}+\sigma^2\mathbf{I}^{-1})\mathbf{y}+\log\left|\mathbf{K}_{\boldsymbol{\theta}}+\sigma^2\mathbf{I}\right| \tag{7}$$

This paper proposes a factorisation-based framework for evaluating log-likelihood and its gradient (i.e., score equations). The approximation of the Cholesky factor gives a quickly and accurately computation for the maximum-likelihood estimation. The details are explained in Section IV

III. Hierarchical Matrices

This section would briefly introduce the HODLR matrices [14]. As we mentioned in Section II, the main reason for the cube computation cost in GP regression is the inverse of the dense covariance matrices. To handle the big dense covariance matrix, many mathematicians developed different

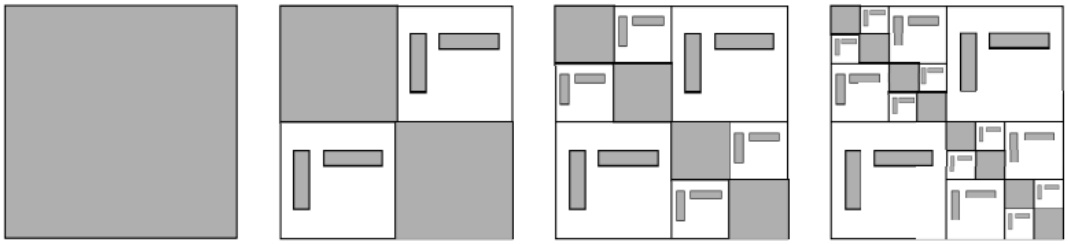

Fig. 1. Level 3 HODLR matrix, the grey blocks represent the matrices that need to be stored as the HODLR matrix.

strategies to rearrange the matrix. The HODLR matrix as a sparse representation of matrix is one of the shining structures to solve the covariance matrices.

A. The Structure of the HODLR Matrix

In accordance with the different low-rank approximation methods, the HODLR matrix has many versions or variants, but its main structure remains the same. In general, HODLR matrices are defined via a recursive block partition [15]. This method aims to do the low-rank approximation to the off-diagonal blocks and remain the diagonal parts. According to the k-dimensional tree, we sort the data points recursively [13]. At each level of the decomposition it would require at most $\mathcal{O}(n)$ time or often much faster. Here is an example of a two-level decomposition in HODLR matrix.

A real symmetric matrix $\mathbf{K} \in \mathbb{R}^{n\times n}$ can be decomposed to a two-level HODLR matrix (the notation are different to the last section):

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_1 & \mathbf{U}_1\mathbf{V}_1^T \\ \mathbf{V}_1\mathbf{U}_1^T & \mathbf{K}_2 \end{bmatrix}, \tag{8}$$

the diagonal blocks can be further decomposed to:

$$\begin{aligned} \mathbf{K}_1 &= \begin{bmatrix} \mathbf{K}_1^{(2)} & \mathbf{U}_1^{(2)}\mathbf{V}_1^{(2)^T} \\ \mathbf{V}_1^{(2)}\mathbf{U}_1^{(2)^T} & \mathbf{K}_2^{(2)} \end{bmatrix}, \\ \mathbf{K}_2 &= \begin{bmatrix} \mathbf{K}_3^{(2)} & \mathbf{U}_2^{(2)}\mathbf{V}_2^{(2)^T} \\ \mathbf{V}_2^{(2)}\mathbf{U}_2^{(2)^T} & \mathbf{K}_4^{(2)} \end{bmatrix}, \end{aligned} \tag{9}$$

where the $\mathbf{K}_1$ and $\mathbf{K}_2$ are the $n/2^j \times n/2^j$ diagonal block matrices from the original matrix K and $\mathbf{U}^{(j)}$, $\mathbf{V}^{(j)}$ matrices are $n/2^j \times r$ matrices with $r \ll n$. j is the level of decomposition which are 2 in this example and rank r is depends on the desired accuracy of the low-rank approximation. A higher rank results in less precision loss and a higher computation cost.

In Fig. 1, the matrix $\mathbf{K}_1$ represent the left top grey block in the second square, $\mathbf{K}_2$ are the left bottom block. $\mathbf{U}_1$, $\mathbf{V}_1$ are the tall and thin rectangle. Then the level 2 decomposition would result in the third square. The white parts in the square are the approximation part which help us to solve the dense matrix easily.

B. Fast Low-Rank Approximation

The most significant step in constructing a HODLR matrix is to compress the off-diagonal blocks into low-rank small matrices U, V. Moreover, these "tall" and "thin" matrices are the most time-consuming step in the factorisation of HODLR method. In this part, we will briefly discuss these methods.

The most important decomposing method: the singular value decomposition (SVD). For a matrix $A \in \mathbb{R}^{m\times n}$. Then there exists a factorisation of the form

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^* \tag{10}$$

where $\mathbf{U} \in \mathbb{R}^{m\times m}, \mathbf{V} \in \mathbb{R}^{n\times n}$ are unitary matrices and $\Sigma \in \mathbb{R}^{m\times n}$ is a diagonal matrix with non-negative entries on the diagonal, i.e., $\sigma_k = \Sigma_{kk} \geq 0$. And the eigenvalue are sort from big to small. The SVD method would offer us the optimal low-rank approximation according to the outcome of the SVD. Keep the first column or row of U and V that has the dominant eigenvalue (diagonal of Σ) [16]. However, SVD is a very expensive computation method. For the square matrix in $n \times n$ the SVD method costs $\mathcal{O}(n^3)$ since it compute the whole matrix. Although it leads to very reliable results, it is meaningless to apply this method to accelerate the GP.

Recently, in papers [14], more aggressive strategies have been proposed, such as partially pivoted adaptive cross approximation [17] or some analytical techniques such as Chebyshev interpolation [18], which can further reduce computation costs to $\mathcal{O}(rn)$. But it should be noted that in GP once the precision of the approximation is too high, the matrix cannot be invested. So the other popular method which called rank-revealing QR factorisation are used. Similar to the SVD method, rank-revealing QR are approximate the matrix by choosing a part of the matrix such as the 'best' rows or columns. The computation for these methods cost $\mathcal{O}(rn^2)$ [19]. The sub-sampling error estimate would be accurate to near machine precision if the underlying matrix (covariance kernel) is sufficiently smooth [20].

IV. Efficient Gaussian Process with Factorisation Based Solutions

As mentioned in Section Section II, the main bottleneck of the GP is the cube computation cost. In the practical, the standard GP is always solved by Cholesky decomposition [3] [21]. Since the covariance matrix is a symmetric positive

definite matrix, Cholesky decomposition can be used to reduce computation costs. The other classic decomposition method: lower-upper (LU) decomposition would take $\mathcal{O}(1/3n^3)$ time, and the Cholesky method only requires half of the computation cost for the covariance matrix $\mathcal{O}(1/6n^3)$, around half time cost. However, it still involves the cube computation cost since the entire matrix should be calculated.

Back to the HODLR matrix, there are different strategies to solve the inverse and determinant of the HODLR matrix such as continuous multiplication [12], [13]. In this paper, since the covariance matrix is a symmetric positive definite matrix, we apply the Cholesky decomposition to the HODLR matrix. The special structure of the HODLR matrix allows us to do it easily with only $\mathcal{O}\left(n\log^2(n)\right)$ computation cost.

A. HODLR Matrix with Cholesky factorisation

As the diagonal parts of the HODLR are still positive definite matrices, we only need to calculate the small diagonal matrices and a few low-rank matrices ($\mathbf{U}$ and $\mathbf{V}$), rather than the entire matrix. So for the HODLR matrix $\mathbf{K}$, an inexact Cholesky factorisation can be done as $\mathbf{K} \approx \mathbf{L}\mathbf{L}^T$ with the lower computation cost. This process can be done recursively, here is an 1 level example: the $\mathbf{K}$ matrix is a HODLR matrix, $\mathbf{U}_1\mathbf{V}_1^T$ are the low rank block, $\mathbf{K}_1$, $\mathbf{K}_2$ are the dense block in the form

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_1 & \mathbf{U}_1\mathbf{V}_1^T \\ \mathbf{V}_1\mathbf{U}_1^T & \mathbf{K}_2 \end{bmatrix}, \quad \mathbf{L} = \begin{bmatrix} \mathbf{L}_{11} & 0 \\ \mathbf{L}_{21} & \mathbf{L}_{22} \end{bmatrix} \tag{11}$$

Since $\mathbf{K} \approx \mathbf{L}\mathbf{L}^T$, it leads to the equations

$$\mathbf{K}_1 = \mathbf{L}_{11}\mathbf{L}_{11}, \tag{12}$$

$$\mathbf{U}_1\mathbf{V}_1^T = \mathbf{V}_1\mathbf{U}_1^T = \mathbf{L}_{11}\mathbf{L}_{21}, \tag{13}$$

$$\mathbf{K}_2 = \mathbf{L}_{21}\mathbf{L}_{21}^T + \mathbf{L}_{22}^2. \tag{14}$$

Therefore the Cholesky factorisation in HODLR structure can be proceed in the following step:

1) Compute Cholesky factors $\mathbf{L}_{11}$ of $\mathbf{K}_1$ from (12)
2) Compute $\mathbf{L}_{21} = \mathbf{L}_{11}^{-1}\mathbf{U}_1\mathbf{V}_1^T$ from (13)
3) Compute $\mathbf{L}_{22}$ of $\mathbf{K}_2 - \mathbf{L}_{12}\mathbf{L}_{21}^T$ from (14)

In this process only $\mathbf{L}_{11}, \mathbf{L}_{22}$ are the dense matrix. Each of these three steps can be addressed using Cholesky factorisation. In the level 1 case the matrices size is n/2, we can further reduce size of the matrices by increase the level of HODLR decomposition. The computation cost of these process is $\mathcal{O}\left(n\log^2(n)\right)$ [22].

B. Fast Solving Algorithm for Finding the Inverse Matrix and its Determinant

After we get the HODLR factorisation and the Cholesky factorisation. The inverse and the determent of the matrix can be solved in a very low computation cost. In this section, we will briefly review the algorithm used for the inverse and the determinant.

The inverse here can be solved as a linear system which is same to the use the Cholesky factorization to solve the matrix inversion. First set the $\mathbf{A} = \mathbf{K} + \sigma_n^2\mathbf{I}$. The matrix inversion can be regarded as get the $\mathbf{x}$ for $\mathbf{A}\mathbf{x} = b$. Then based on the Cholesky factorization

$$\mathbf{L}\mathbf{L}^\top\mathbf{x} = b \tag{15}$$

Assume for some y let $\mathbf{L}y = b$.

$$\mathbf{L}\mathbf{L}^\top\mathbf{x} = \mathbf{L}y \tag{16}$$

$$\mathbf{L}^\top\mathbf{x} = \mathbf{y} \tag{17}$$

Thus, if we can solve for $\mathbf{y}$ in $\mathbf{L}\mathbf{y} = \mathbf{b}$, and then solve for $\mathbf{x}$ in $\mathbf{L}^\top\mathbf{x} = \mathbf{y}$, we will have solved for the same $\mathbf{x}$ that solves $\mathbf{A}\mathbf{x} = \mathbf{b}$. Let the notation $\mathbf{A}\backslash\mathbf{b}$ denote the vector $\mathbf{x}$ that solves $\mathbf{A}\mathbf{x} = \mathbf{b}$. Then we have

$$\mathbf{x} = \mathbf{L}^\top\backslash(\mathbf{L}\backslash\mathbf{b}) \tag{18}$$

To gain the matrix $\mathbf{y}$, based on $\mathbf{L}\mathbf{y} = \mathbf{b}$. To be specific, with the same matrix in (11)

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_{11} & 0 \\ \mathbf{L}_{21} & \mathbf{L}_{22} \end{bmatrix}, \tag{19}$$

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_{11} & \mathbf{y}_{12} \\ \mathbf{y}_{21} & \mathbf{y}_{22} \end{bmatrix}, \tag{20}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix}, \tag{21}$$

where a low-rank matrix $\mathbf{L}_{21}$ and HODLR matrices $\mathbf{L}_{11}, \mathbf{L}_{22}$.To find the $\mathbf{Y}_{11}, \mathbf{Y}_{12}$:

$$\mathbf{L}_{11}\mathbf{y}_{11} = \mathbf{B}_{11}, \quad \mathbf{L}_{11}\mathbf{y}_{12} = \mathbf{B}_{12}. \tag{22}$$

To find $\mathbf{y}_{11}, \mathbf{y}_{12}$. Afterwards, we get $\mathbf{y}_{21}, \mathbf{y}_{22}$ from

$$\mathbf{L}_{22}\mathbf{Y}_{21} = \mathbf{B}_{21} - \mathbf{L}_{21}\mathbf{y}_{11}, \quad \mathbf{L}_{22}\mathbf{y}_{22} = B_{22} - \mathbf{L}_{21}\mathbf{y}_{12} \tag{23}$$

In this case the linear system can be solved with $\mathcal{O}\left(n\log(n)\right)$ [22]. Meanwhile, $\det\left(\mathbf{K} + \sigma_n^2\mathbf{I}\right)$ can be efficiently computed using the Cholesky factor $\mathbf{L}$ of $\mathbf{K} + \sigma_n^2\mathbf{I}$:

$$\det\left(\mathbf{K} + \sigma_n^2\mathbf{I}\right) = \det\left(\mathbf{L}\mathbf{L}^\top\right) = \det(\mathbf{L})\det\left(\mathbf{L}^\top\right)$$

The reason $\det(\mathbf{L})$ is efficient to compute is because $\mathbf{L}$ looks like the following, Since the determinant can be written as the sum of the products of the elements in the top row with their respective minors [23] the first step in the computation can be written as

$$\ell_{1,1} \times \begin{bmatrix} \ell_{2,2} & 0 & \cdots & 0 \\ \ell_{3,2} & \ell_{3,3} & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ \ell_{n,2} & \cdots & \cdots & \ell_{n,n} \end{bmatrix}$$

Continuing this logic, determinant can be write as the sum of the diagonal of the $\mathbf{L}$ matrix:

$$\det(\mathbf{L}) = \prod_{i=1}^{n} \ell_{i,i}. \tag{24}$$

The determinant is obtained by only $\mathcal{O}(n)$ computation cost, which can be ignored.

C. Computational Complexity

We propose a three-step factorisation method to solve the GP: (i) constructing a HODLR factorisation by computing the low-rank factors of all off-diagonal blocks, (ii) applying the Cholesky factorisation to the symmetric HODLR matrix, (iii) solving the inverse and determinant based on the factor of Cholesky factorisation.

In the first step, the low-rank approximation method is the primary factor influencing the computation cost. This paper applies the Householder QR decomposition with column pivoting [24] to the off-diagonal block. This algorithm is terminated when an upper bound for the spectral norm of the remainder is below ϵ times the maximum pivot element. If r denotes the HODLR rank of the output, this procedure has complexity $\mathcal{O}\left(rn^2\right)$.

For the Cholesky factorisation on HODLR matrix, Ballani [22] proposed a detailed calculation process which results in $\mathcal{O}\left(n\log^2(n)\right)$ costs. Then linear systems $Ax = b$ with the lower triangular HODLR matrix requires additional cost of $\mathcal{O}(n\log(n))$ and $\mathcal{O}(n)$ to sum the diagonal elements of the determinant.

V. Performance Validation

In this section, we will test the performance of the new factorisation GP on solve the synthetic data and the tracking in sensor network . All these experiments were run on a laptop with AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz and 8.00 GB RAM.

In this section, we will evaluate the computation time take and the accuracy of the HCFGP method. The standard GP which inverts the covariance matrix directly is used as the benchmark in this scheme. We also apply the LU and Cholesky factorisation method to compare with the HCFGP method since all these methods are directly factorisation-based methods. The accuracy are represent by the root mean squared error (RMSE) between the prediction to the true data. The accuracy of HCFGP method are rely on the tolerance of the low rank approximation. A smaller tolerance will leads a more accuracy approximation with higher computation cost. Here we select the tolerance as 0.00001 as a trade-off. This solution relies on maximum likelihood estimation (7) for learning of the hyperparameters.

A. Squared Exponential Kernel

This experiment uses the most common kernel function: the squared exponential kernel

$$K\left(x_i, x_j\right) = \sigma^2 \exp\left(-\frac{\left(x_i - x_j\right)^2}{2l^2}\right), \tag{25}$$

which is appropriate for modelling smooth functions. There are two hyperparameters in this kernel, the length scale l determines the length of the 'wiggles' in your function and the variance σ^2 determines the average distance of your function away from its mean as a scale factor.

B. Synthetic Data

The simultaneously modeling the following function:

$$y = 0.3x + 1.2 + \sin\left(x\right) + \epsilon, \tag{26}$$

where ϵ is the noise with standard deviation 0.5. Then randomly pick n initial x_i from $[-2, 2]$ and the real input x of the training data is $x = 1 + 4x_i + sign(x_i)$ which clusters the training data into two clusters. The 10000 inputs x_* are isometric take from -8 to 10 for the prediction. As a directly method, HCFGP and other factorisation methods can work on the training and prediction at same time. So here we record the time taken for training and prediction separately. Table I shows the time take for training of the hyperparameters and Table II shows the time needed for the prediction.

TABLE I
THE TIME FOR TRAINING THE HYPERPARAMETERS

	Time (second)			
Data size	**1000**	**2000**	**5000**	**10000**
Invert	2.41	22.19	299.73	1480.09
LU	2.27	28.67	277.12	1506.74
Chol	1.73	20.758	204.45	986.98
HCFGP	2.28	18.45	143.39	438.51

TABLE II
THE TIME FOR PREDICTION OF THE GP

	Time (second)			
Data size	**1000**	**2000**	**5000**	**10000**
Invert	0.64	2.19	19.63	97.22
LU	0.67	2.82	17.86	87.28
Chol	0.54	1.902	11.11	57.37
HCFGP	0.37	1.03	4.77	16.81

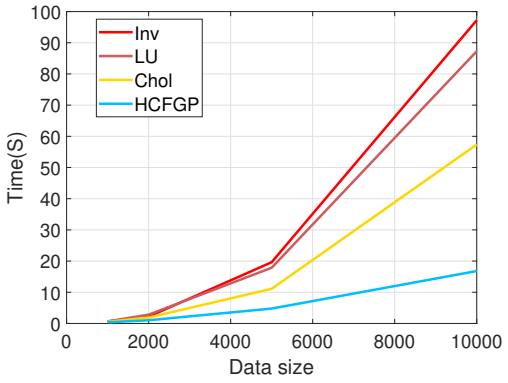

Fig. 2. The running time for different factorisation methods

In Table I, when the data size is more than 2000 the HCFGP method takes less time to learn the hyperparameter than other methods. In Table II, when the data size is more than 1000, the HCFGP method contributes to time reduction in the calculation of the prediction step. This result demonstrates that the HCFGP method reduces the computational costs in both training and prediction.

However, when the data size is less than 1000, the training process would take longer than the other methods. Meanwhile, the HCFGP has the shortest prediction time in prediction for 1000 data sizes. This phenomenon is caused by the uncertainty of the low-rank approximation method. Since the learning process is a repetition of the single HCFGP, the rank r of a low-rank approximation would be high to meet the tolerance under certain circumstances. The HCFGP based on QR decomposition, which requires $\mathcal{O}\left(rn^2\right)$ for the first step, would be no more competitive in this circumstance when the data size is small.

At last, in Table III, the RMSE of all these factorisation based method are very close the the benchmark (standard GP). It also demonstrate the HCFGP method achieves a high accuracy of the prediction.

TABLE III
THE RMSE FOR THE PREDICTION WITH DIFFERENT FACTORISATION BASED GP METHOD

	RMSE (meter)			
Data size	**1000**	**2000**	**5000**	**10000**
Invert	1.35	1.21	1.10	0.98
LU	1.35	1.21	1.05	0.98
Chol	1.37	1.21	1.05	0.98
HCFGP	1.36	1.21	1.05	0.99

C. Testing Scenarios

This section presents the evaluation of the HCFGP GP methods over a sensor network with 200 randomly distributed sensors in $1(km^2)$ two-dimensional square regions. The sensors' sensing range and the trajectory of one moving object of interest is represented in Fig. 3. There are two scenarios for this experiment. Trajectories of moving objects are only shown as long as at least one sensor contains the object in its range.

Performance of sparse approximation GP methods is evaluated in two manoeuvring examples and over 30 Monte Carlo independent runs. Here, the experiment consists of four scenarios. Testing scenarios in which the motion models used to generate the trajectory of the targets are the same models used in the tracking algorithms are called matched scenarios.

- **S1** the target trajectory is generated based on the nearly constant velocity model in the straight line, and the velocity changes at each pre-defined turning point.
- **S2** The target trajectory is generated by the Singer acceleration model. The maximum possible acceleration is 2 m/s^2, the probability of non-acceleration is 0.3.

The main objective of our factorisation-based method is to reduce the computation cost. Based on the trajectory the noise are adding to this scenarios. To evaluate HCFGP method in the big dataset, each trajectory would generate multiple data with Poisson distribution with the additional noise. Here is an example for S1 with 3000 observations size and S2 with 1500 observations size in Fig. 3.

D. Offline Tracking

The performance of the HCFGP method has been demonstrated in both the training and prediction processes. In each of the following tables, the timings are provided for the sum of training and prediction. Before testing online tracking, we first validate the performance of HCFGP on the offline tracking. Here we trade three-quarters of the data as training data, and the rest are the testing data. Since the 1D GP method is applied, we train the x and y, respectively. Here the data size is no more than 10000, so we set the minimum block size of the HCFGP matrix as 600. All offline results are collected from 30 Monte Carlo runs. The HCFGP The following is the result of the offline tracking for different factorisation methods:

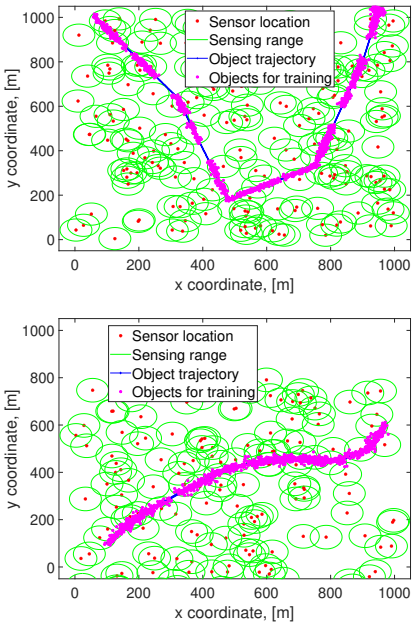

Fig. 3. S1 is a wide-angle turn matched scenarios, S2 is a random speed scenarios, where the green circles are the sensing range, the red point are the sensor location, the blue line is the object trajectory and the purple points are the sensor measurements.

TABLE IV
THE TIME TAKE FOR FACTORISATION METHODS IN S1 OFFLINE TRACKING

	Time (second)			
Data size	**600**	**1200**	**3000**	**8000**
Invert	1.12	9.88	39.53	508.82
LU	1.32	11.43	43.27	528.34
Chol	0.99	9.01	32.55	478.25
HCFGP	1.06	9.08	27.12	267.94

TABLE V
THE TIME TAKE FOR FACTORISATION METHODS IN S2 OFFLINE TRACKING

	Time (second)			
Data size	**1500**	**2500**	**5000**	**8000**
Invert	9.90	28.65	162.70	648.92
LU	11.12	32.03	173.00	668.97
Chol	8.19	22.71	114.08	416.42
HCFGP	8.18	21.84	86.22	276.95

Tables IV and V indicate that the HCFGP method reduces the time cost for the GP when the data size is greater than

3000. The LU decomposition has the longest take in almost all data sizes. The standard Cholesky decomposition is competitive when the data size is less than 3000. Obviously, in the large data size, our HCFGP method takes less than 300 seconds for the 8000 data size, which is significantly faster than another option.

During the same period, the tracking performance is represented by the RMSE. As illustrated in Table VI and VII, the HCFGP only has a very small loss of accuracy in different data sizes compared to other methods. This result demonstrates that the HCFGP method can meet our theoretical expectations to reduce the time taken of the GP to solve the tracking problems with reliable performance.

TABLE VI
THE PERFORMANCE OF FACTORISATION METHODS ON THE OFFLINE TRACKING IN S1

	Invert	LU	Chol	HCFGP
	Data size 600			
RMSE-X(m)	3.87	3.87	3.87	3.84
RMSE-Y(m)	5.03	5.03	5.03	5.05
	Data size 1200			
RMSE-X(m)	4.39	4.39	4.39	4.40
RMSE-Y(m)	5.17	5.17	5.17	5.17
	Data size 3000			
RMSE-X(m)	4.70	4.70	4.70	4.70
RMSE-Y(m)	6.47	6.47	6.47	6.47
	Data size 8000			
RMSE-X(m)	5.71	5.73	5.71	5.73
RMSE-Y(m)	9.25	9.24	9.24	9.25

TABLE VII
THE PERFORMANCE OF FACTORISATION METHODS ON THE OFFLINE TRACKING IN S2

	Invert	LU	Chol	HCFGP
	Data size 1500			
RMSE-X(m)	2.22	2.22	2.22	2.22
RMSE-Y(m)	3.72	3.72	3.72	3.72
	Data size 2500			
RMSE-X(m)	2.21	2.22	2.22	2.24
RMSE-Y(m)	1.77	1.77	1.77	1.77
	Data size 5000			
RMSE-X(m)	2.09	2.09	2.09	2.09
RMSE-Y(m)	1.58	1.58	1.58	1.58
	Data size 8000			
RMSE-X(m)	1.61	1.61	1.61	1.61
RMSE-Y(m)	1.75	1.75	1.75	1.75

E. Online Tracking

Online tracking is a repeat of offline tracking in every step of the tracking process with only a single prediction. In this scenario, we divide all the training data here into 120 time steps. For each time step, we learn the hyperparameter and predict the object state. The hyperparameters are learned recursively. The previous iteration of the optimize would be applied as the initial value of the next iteration to reduce the number of iterations required for convergence. Same as the offline tracking, we apply the SE kernel (25) to predict the x and y direction separately. Here are the performances of each method:

TABLE VIII
THE TIME TAKE FOR FACTORISATION METHODS IN S1 ONLINE TRACKING

	Time (second)			
Data size	**1200**	**2000**	**4000**	**6000**
Invert	99.99	367.34	2172.65	12202.07
LU	111.23	412.60	2358.82	13716.65
Chol	77.02	291.81	1811.69	8636.39
HCFGP	81.06	278.49	1223.03	5302.46

TABLE IX
THE TIME TAKE FOR FACTORISATION METHODS IN S1 ONLINE TRACKING

	Time (second)			
Data size	**1500**	**2500**	**5000**	**8000**
Invert	46.59	144.64	1132.32	3244.17
LU	49.91	157.14	1192.65	3458.21
Chol	36.26	125.35	830.32	2457.68
HCFGP	47.92	126.70	701.60	1453.30

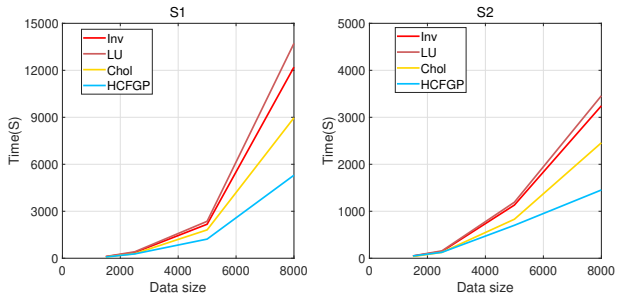

Fig. 4. The time take for different factorisation method for offline tracking

Tables VIII and IX show that the Cholesky decomposition method is faster than the standard GP and LU decomposition. However, when the data size is more than 3000, the time taken for the HCFGP has a clear advantage compared to the other methods. It proves the HCFGP method does improve the Cholesky decomposition in large data sizes. As shown in Figure 4, the HCFGP will significantly reduce the computation time for the larger data sizes. The RMSE result is shown in Table X and XI. The error loss of the HCFGP during the approximation can be ignored if we use the standard GP as the benchmark. These results demonstrate the efficiency of the HCFGP method in solving online tracking problems.

VI. CONCLUSIONS

In this paper, a factorisation-based model-free GP is raised to solve the bottleneck of the cubic computational cost for GP in the tracking problem. In the HCFGP factorisation, only the diagonal parts of the covariance matrix are considered for the Cholesky decomposition. This method only requires $\mathcal{O}(n \log^2 n)$ cost for Cholesky decomposition, inverse in $\mathcal{O}(n \log n)$ operations and determinant in $\mathcal{O}(n)$ operation. The performance of this method is proved in both online and offline tracking. Future work will focus on applying more aggressive low-rank approximations strategies and combine with sparse approximation GP methods.

TABLE X
THE PERFORMANCE OF FACTORISATION METHODS ON THE ONLINE TRACKING IN S1

	Invert	LU	Chol	HCFGP
	Data size 600			
RMSE-X(m)	14.70	14.70	14.70	14.70
RMSE-Y(m)	7.67	7.67	7.67	7.67
	Data size 1200			
RMSE-X(m)	15.46	15.46	15.46	15.46
RMSE-Y(m)	6.23	6.23	6.23	6.23
	Data size 3000			
RMSE-X(m)	13.95	13.95	13.95	13.94
RMSE-Y(m)	5.75	5.75	5.75	5.79
	Data size 8000			
RMSE-X(m)	18.61	18.61	17.45	17.45
RMSE-Y(m)	16.38	16.38	16.38	16.38

TABLE XI
THE PERFORMANCE OF FACTORISATION METHODS ON THE ONLINE TRACKING IN S2

	Invert	LU	Chol	HCFGP
	Data size 1500			
RMSE-X(m)	8.24	8.24	8.24	8.24
RMSE-Y(m)	7.32	7.32	7.32	7.32
	Data size 2500			
RMSE-X(m)	8.60	8.60	8.60	8.60
RMSE-Y(m)	6.74	6.74	6.74	6.74
	Data size 5000			
RMSE-X(m)	12.97	12.97	12.97	12.97
RMSE-Y(m)	10.61	10.61	10.61	10.61
	Data size 8000			
RMSE-X(m)	13.10	13.10	13.10	13.10
RMSE-Y(m)	10.89	10.89	10.89	10.89

ACKNOWLEDGMENT

This research is sponsored by the US Army Research Laboratory and the UK MOD University Defence Research Collaboration (UDRC) in Signal Processing under the SIGNeTS project. It is accomplished under Cooperative Agreement Number W911NF-20-2-0225. The views and conclusions contained in this document are of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory, the MOD, the U.S. Government or the U.K. Government. The U.S. Government and U.K. Government are authorised to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. This work was funded partly by the EPSRC EP/T013265/1 project NSFEPSRC: "ShiRAS. Towards Safe and Reliable Autonomy in Sensor Driven" and the National Science Foundation under Grant USA NSF ECCS 1903466. For the purpose of open access, the authors have applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising.

REFERENCES

[1] K. Granstrom, M. Baum, and S. Reuter, "Extended object tracking: Introduction, overview and applications," *arXiv preprint arXiv:1604.00970*, 2016.

[2] L. Mihaylova, A. Y. Carmi, F. Septier, A. Gning, S. K. Pang, and S. Godsill, "Overview of Bayesian sequential Monte Carlo methods for group and extended object tracking," *Digital Signal Processing*, vol. 25, pp. 1–16, 2014.

[3] C. E. Rasmussen, "Gaussian processes in machine learning," in *Advanced Lectures on Machine Learning. Lecture Notes in Computer Science*, vol. 3176, pp. 63–71, Springer, 2003.

[4] J. Q. Shi and T. Choi, *Gaussian Process Regression Analysis for Functional Data*. New York: Chapman and Hall CRC Press, 2011.

[5] J. Hensman, N. Fusi, and N. D. Lawrence, "Gaussian processes for big data," *arXiv preprint arXiv:1309.6835*, 2013.

[6] H. Liu, Y.-S. Ong, X. Shen, and J. Cai, "When Gaussian process meets big data: A review of scalable GPs," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 11, pp. 4405–4423, 2020.

[7] K. Hayashi, M. Imaizumi, and Y. Yoshida, "On random subsampling of Gaussian process regression: A graphon-based analysis," in *Proceeding of the International Conference on Artificial Intelligence and Statistics*, pp. 2055–2065, PMLR, 2020.

[8] E. Snelson and Z. Ghahramani, "Sparse Gaussian processes using pseudo-inputs," *Advances in Neural Information Processing Systems*, vol. 18, 2005.

[9] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.

[10] M. K. Titsias, "Variational model selection for sparse Gaussian process regression," *Report, University of Manchester, UK*, 2009.

[11] A. Wilson and H. Nickisch, "Kernel interpolation for scalable structured Gaussian processes (kiss-gp)," in *International Conference on Machine Learning*, pp. 1775–1784, PMLR, 2015.

[12] S. Ambikasaran, D. Foreman-Mackey, L. Greengard, D. W. Hogg, and M. O'Neil, "Fast direct methods for Gaussian processes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 2, pp. 252–265, 2015.

[13] S. Massei, L. Robol, and D. Kressner, "hm-toolbox: Matlab software for HODLR and HSS matrices," *SIAM Journal on Scientific Computing*, vol. 42, no. 2, pp. C43–C68, 2020.

[14] A. Aminfar, S. Ambikasaran, and E. Darve, "A fast block low-rank dense solver with applications to finite-element matrices," *Journal of Computational Physics*, vol. 304, pp. 170–188, 2016.

[15] S. Börm, L. Grasedyck, and W. Hackbusch, "Hierarchical matrices," *Lecture notes*, vol. 21, p. 2003, 2003.

[16] H. Andrews and C. Patterson, "Singular value decomposition (svd) image coding," *IEEE Transactions on Communications*, vol. 24, no. 4, pp. 425–432, 1976.

[17] Y. Liu, W. Sid-Lakhdar, E. Rebrova, P. Ghysels, and X. S. Li, "A parallel hierarchical blocked adaptive cross approximation algorithm," *The International Journal of High Performance Computing Applications*, vol. 34, no. 4, pp. 394–408, 2020.

[18] C. Effenberger and D. Kressner, "Chebyshev interpolation for nonlinear eigenvalue problems," *BIT Numerical Mathematics*, vol. 52, no. 4, pp. 933–951, 2012.

[19] T. F. Chan, "Rank revealing QR factorizations," *Linear Algebra and its Applications*, vol. 88, pp. 67–82, 1987.

[20] S. Ambikasaran, *Fast algorithms for dense numerical linear algebra and applications*. Stanford University, 2013.

[21] L. N. Trefethen and D. Bau III, *Numerical linear algebra*, vol. 50. SIAM, 1997.

[22] J. Ballani and D. Kressner, "Matrices with hierarchical low-rank structures," in *Exploiting Hidden Structure in Matrix Computations: Algorithms and Applications*, pp. 161–209, Springer, 2016.

[23] D. Dereniowski and M. Kubale, "Cholesky factorization of matrices in parallel and ranking of graphs," in *Proc. of the International Conference on Parallel Processing and Applied Mathematics*, pp. 985–992, Springer, 2003.

[24] P.-G. Martinsson, G. Quintana OrtÍ, N. Heavner, and R. van de Geijn, "Householder QR factorization with randomization for column pivoting (hqrrp)," *SIAM Journal on Scientific Computing*, vol. 39, no. 2, pp. C96–C115, 2017.