Efficient Centralised and Decentralised Gaussian Process Approaches for Online Tracking within Stone Soup

Chenyi Lyu[1], Xingchi Liu[1,2], James Wright[3], Jordi Barr[3], Alasdair Hunter[3] and Lyudmila Mihaylova[1]
[1]Department of Automatic Control & Systems Engineering, University of Sheffield, S1 3JD, UK
[2]Scientific Computing Department, Rutherford Appleton Laboratory, Science and Technology Facilities Council, UK
[3]Defence Science and Technology Laboratory, UK
Email: clyu5@sheffield.ac.uk, xingchi.liu@stfc.ac.uk, jwright2@dstl.gov.uk,
jmbarr@dstl.gov.uk, Ahunter@dstl.gov.uk, l.s.mihaylova@sheffield.ac.uk

***Abstract*—This paper explores the application of centralised and distributed Gaussian process algorithms to real-time target tracking and compares their performance. By embedding the algorithms into the Stone Soup, the focus is on the innovative implementation of Gaussian process methods with learning hyperparameters and implementation with a factorised variance of the Gaussian kernel. The performance of the methods with different kernels was evaluated, not only with the Gaussian kernel. Extensive experiments with various kernel configurations demonstrate their importance in enhancing prediction accuracy and efficiency, especially in real-time tracking. The case studies with manoeuvring targets show significant advancements in tracking capabilities, particularly in wireless sensor networks, using optimised Gaussian process methods. This work advances Stone Soup's capabilities and lays the groundwork for future investigations into adaptive Gaussian Process applications in tracking and sensor data analysis.**

***Index Terms*—Learning Gaussian process methods, Distributed Gaussian process, sensor networks, covariance matrix, tracking, Stone Soup, online tracking**

I. Introduction

In this paper, we integrate Gaussian process (GP) and distributed Gaussian process (DGP) methods within the Stone Soup framework [1] to provide online target tracking in sensor networks. This area is crucial for many applications, including in transport systems, surveillance, environmental monitoring, and military operations, where the objective is to accurately estimate a target's location, velocity, and other states from noisy and incomplete sensor data. The GP methodology has proven its advantages and effectiveness and relies on solid theoretical foundations [2], [3].

Traditionally, Bayesian filters such as particle filters (PFs) [4] have addressed the challenges posed by nonlinear and non-Gaussian system models and measurement noises. However, our approach leverages the flexibility and robustness of GPs and DGPs, offering advantages over these traditional methods. The adaptability of GPs and DGP makes them particularly effective in handling high-dimensional and complex problem spaces characteristic of target tracking scenarios [5] and especially when implemented in a distributed manner.

The Stone Soup open-source framework represents a valuable tool for developing and testing state estimation algorithms [6]–[8]. It is designed to prioritise flexibility, enabling the effective selection and testing of various algorithmic components in real-world scenarios. This paper demonstrates how GPs and DGPs can be effectively implemented within this platform.

The main contributions of this work are: 1) it presents efficient centralised and distributed GP algorithms, implemented within the Stone Soup framework [9]–[11], 2) the Cholesky factorisation versions of these algorithms are also presented and evaluated over target tracking case studies. 3) Sensor network deployment: we can consider a variety of parameters, for example, the number of sensors and minimum distance. This implementation involves designing critical components compatible with Stone Soup, tailored for GP and DGP functionalities. We introduce specialised components like GP and GPKernel, GP-based predictors and updates. These additions to Stone Soup facilitate applying diverse kernel functions and initial hyperparameters, allowing for extensive experimentation and performance comparison across tracking trajectories. Our development paves the way for GP and DGP models to be adaptively used in various dynamic and measurement models. The versatility of Stone Soup's design also enables future enhancements in joint tracking and parameter estimation, leveraging the strengths of GP and DGP approaches.

The paper is organised as follows: Section II provides a mathematical background on GP and DGP. Section III describes the GP-based target tracking approaches and their detailed implementations within Stone Soup in Section IV. The simulation setup and results are presented in Section V. Finally; Section VI discusses future extensions and applications of our work within the Stone Soup framework.

II. Theoretical Background Knowledge

A. Overview of Gaussian Process

This section briefly overviews the GP regression method and the associated computational process for inference and learning.

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 [2]. 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:

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

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

where $\bar{f}(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})], \mathcal{X} \to \mathbb{R}$ is the mean function and $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is the kernel function. The training and the test input data are denoted by $\mathbf{x}$ and $\mathbf{x}_*$, respectively. The mean function is often assumed to be 0. The kernel function controls the smoothness of GP specified as $\mathbf{K} = k(\mathbf{x}, \mathbf{x}')$. In (2), the term ε represents an additive independent identically distributed Gaussian measurement noise and $\mathbb{E}[\cdot]$ is the mathematical expectation operation. The variance $\sigma^2 \neq 0$ [2].

Then we define a GP representation to $\mathbf{f}_*$, which follows the Bayesian approach to predict the output $\mathbf{Y}_*$ for the new input $X_* = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}^\top$ 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. Define the covariance between the training input $\mathbf{X}$ and test input $\mathbf{x}_*$ as $\mathbf{K}_{\mathbf{x},\mathbf{x}_*} = k(\mathbf{X}, \mathbf{x}_*)$, the predictive distribution of the target state at time t is given by a Gaussian distribution with the mean and covariance obtained from the GP representation $\mathcal{N}(\mu(\mathbf{x}_*), \sigma^2(\mathbf{x}_*))$ as

$$\mu(\mathbf{x}_*) = \bar{\boldsymbol{f}}(\mathbf{x}_*) + \mathbf{K}_{\mathbf{x},\mathbf{x}_*}^T \boldsymbol{\Sigma}^{-1}(\mathbf{y} - \bar{f}(\mathbf{x}_*)), \tag{4}$$

$$\sigma^2(\mathbf{x}_*) = \mathbf{K}_{\mathbf{x}_*,\mathbf{x}_*} - \mathbf{K}_{\mathbf{x},\mathbf{x}_*}^T \boldsymbol{\Sigma}^{-1} \mathbf{K}_{\mathbf{x},\mathbf{x}_*}, \tag{5}$$

where $\boldsymbol{\Sigma} = \mathbf{K}_{nn} + \sigma^2\mathbf{I}$ with $\mathbf{I}$ being the identity matrix and $\mathbf{K}_{nn}$ being the covariance matrix of the input training data.

B. Distributed Gaussian Process

The computational complexity and storage cost are major challenges for large-scale learning problems. Based on equations (4) and (5), the computational bottleneck to GP algorithms is to learn the hyperparameters and unknown functions. The computations require $\mathcal{O}(n^3)$ time with a standard GP implementation, where n represents the number of training instances. Besides, the standard GP also requires $\mathcal{O}(n^2 + nd)$ of memory, where d is the dimensionality of the data. Both facts limit the scalability of the standard GP regression. Moreover, according to (4) and (5), the standard GP can only make predictions based on all the available data, a centralised scheme requiring data to be shared among sensors.

In this section, inspired by the idea of divide-and-conquer, DGP methods are introduced to reduce not only the computational cost but also the memory cost of the standard GP by first training local GPs based on subsets of the whole training data set and then aggregating the knowledge of local GPs to achieve more accurate high-level predictions [12] [13]. The overall computational complexity and the memory cost can be reduced to $\mathcal{O}(n_{\textbf{local}}{}^2 n)$ and $\mathcal{O}(M n_{\textbf{local}}{}^2 + nd)$ ($n_{\textbf{local}} \ll n$), respectively, where M represents the number of local GPs, and $n_{\textbf{local}}$ represents the size of data used for training a local GP. The computational complexity and storage cost can be further reduced through parallel/distributed computing [14].

The first type of DGP method is the product of experts (PoEs) [15] approach. The idea is to multiply the local predictive probability distributions for overall predictions. Given the data $D^{(i)}$ collected by sensor i, the PoE predicts a function value $f(\mathbf{x}_*)$ at a corresponding test input $\mathbf{x}_*$ according to

$$p(f(\mathbf{x}_*) \mid \mathbf{x}_*, D) = \prod_{i=1}^{M} p_i(f(\mathbf{x}_*) \mid \mathbf{x}_*, D^{(i)}), \tag{6}$$

where M is the number of GP experts and represents the number of active sensors with measurements. Since the product of these Gaussian predictions is proportional to a Gaussian distribution, the aggregated predictive mean and variance can be calculated in closed form as

$$\mu_*^{\text{PoE}} = (\sigma_*^{\text{PoE}})^2 \sum_{i=1}^{M} \sigma_i^{-2}(\mathbf{x}_*)\mu_i(\mathbf{x}_*), \tag{7}$$

$$(\sigma_*^{\text{PoE}})^{-2} = \sum_{i=1}^{M} \sigma_i^{-2}(\mathbf{x}_*), \tag{8}$$

where $\mu_i(\mathbf{x}_*)$ and $\sigma_i^2(\mathbf{x}_*)$ represent the predictive mean and variance of GP expert i, respectively, which can be calculated based on (4) and (5)

The PoE model provides a straightforward way to aggregate local predictions and sidesteps the weight assignment issue in other DGP models, such as the mixture of experts model [16]. However, this model becomes overconfident when making predictions, especially in regions without any training data.

The generalised product of experts (GPoEs) model [17] improves PoE by adding weights representing different experts' contributions. For instance, the weight can be calculated as the difference in the differential entropy between the prior distribution $p(f(\mathbf{x}_*))$ and the posterior predictive distribution $p(f(\mathbf{x}_*) \mid \mathbf{x}_*, D)$, which can be written as

$$\beta_i = 0.5\left(\log \sigma_{**}^2 - \log \sigma_i^2(\mathbf{x}_*)\right), \tag{9}$$

where σ_{**}^2 represents the variance of the prior distribution $p(f(\mathbf{x}_*))$ and $\sigma_i^2(\mathbf{x}_*)$ denotes the predictive variance of GP expert i, which can be calculated based on (5).

Given the data $D^{(i)}$ collected by sensor i, the GPoE predicts a function value $f(\mathbf{x}_*)$ at a test input $\mathbf{x}_*$. The predictive distribution and the closed forms of the aggregated predictive mean and variance can be written as

$$p(f(\mathbf{x}_*) \mid \mathbf{x}_*, D) = \prod_{i=1}^{M} p_i^{\beta_i}(f(\mathbf{x}_*) \mid \mathbf{x}_*, D^{(i)}), \tag{10}$$

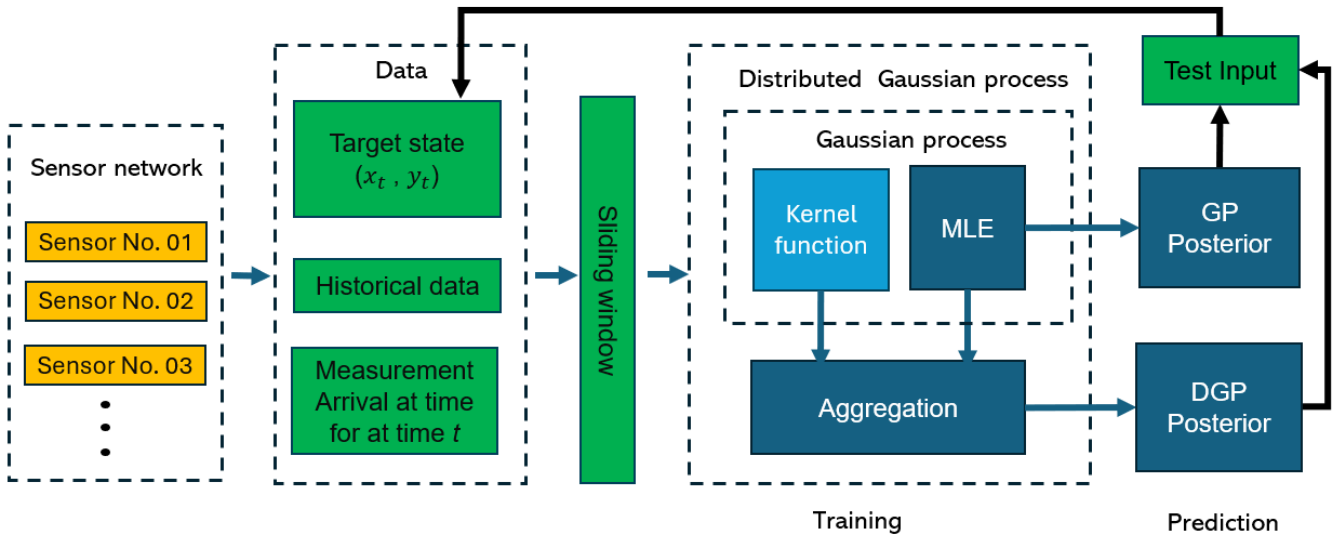

Fig. 1. Overview of the target tracking process using GP and DGP, highlighting the training and prediction phases. Green boxes represent sensor inputs. Light blue and dark blue boxes denote GP training components. Dashed lines delineate the flow of data through the system.

$$\mu_*^{\text{GPoE}} = (\sigma_*^{\text{GPoE}})^2 \sum_{i=1}^{M} \beta_i \sigma_i^{-2}(\mathbf{x}_*)\mu_i(\mathbf{x}_*), \tag{11}$$

$$(\sigma_*^{\text{GPoE}})^{-2} = \sum_{i=1}^{M} \beta_i \sigma_i^{-2}(\mathbf{x}_*). \tag{12}$$

All the models discussed in this section can be applied to infer the target states in a distributed way in the target tracking problem. The closed form of posterior predictions can be obtained, and the predictions are fully tractable.

C. Learning of Hyperparameters

Maximum likelihood estimation (MLE) 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 maximising the negative log marginal likelihood:

$$-\log p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}) \propto \mathbf{y}^\top (\mathbf{K}_{\boldsymbol{\theta}} + \sigma^2 \mathbf{I})^{-1}\mathbf{y} + \log \left|\mathbf{K}_{\boldsymbol{\theta}} + \sigma^2 \mathbf{I}\right| \tag{13}$$

For the DGP, assuming the local GPs are independent of each other, the log marginal likelihood can be factorised as

$$\begin{aligned} &\log p(\mathbf{y}|\mathbf{X}, \theta) \\ \approx\ &\sum_{i=1}^{M} \log p_i(\mathbf{y}^{(i)}|\mathbf{X}^{(i)}, \theta), \\ =\ &\sum_{i=1}^{M} \left(-\frac{1}{2}\mathbf{y}^{(i)\intercal}{\Sigma^{(i)}}^{-1}\mathbf{y}^{(i)} - \frac{1}{2}\log|\Sigma^{(i)}| - \frac{n_{\textbf{local}}}{2}\log 2\pi \right), \end{aligned} \tag{14}$$

where $\mathbf{X}^{(i)}$, $\mathbf{y}^{(i)}$, and $\Sigma^{(i)}$ represent the training input, training output, and the covariance matrix of local GP i, respectively.

D. A Gaussian Process with Cholesky Factorisation

Efficient GP regression implementations employ the Cholesky factorisation to compute the inverse of the covariance matrix during prediction efficiently, which only costs $\mathcal{O}\left(\frac{1}{6}n^3\right)$ for the GP. This method decomposes the positive-definite covariance matrix $\mathbf{K}_{nn}$ into the product of a lower triangular matrix $\mathbf{L}$ and its transpose, such that $\mathbf{K}_{nn} = \mathbf{L}\mathbf{L}^T$. The Cholesky factorisation facilitates stable and efficient solutions to systems of linear equations, which is essential in GP for calculating the posterior mean and variance.

With the Cholesky factorisation, the predictive mean $\mu(\mathbf{x}_*)$ and predictive variance $\sigma^2(\mathbf{x}_*)$ at a new test point $\mathbf{x}_*$ are given by

$$\mu(\mathbf{x}_*) = \bar{f}(\mathbf{x}_*) + \mathbf{K}_{\mathbf{x},\mathbf{x}_*}^T \mathbf{L}^{-T}\mathbf{L}^{-1}\mathbf{y}, \tag{15}$$

$$\sigma^2(\mathbf{x}_*) = k(\mathbf{x}, \mathbf{x}_*) - \mathbf{K}_{\mathbf{x},\mathbf{x}_*}^T \mathbf{L}^{-T}\mathbf{L}^{-1}\mathbf{K}_{\mathbf{x},\mathbf{x}_*}. \tag{16}$$

The lower triangular matrix $\mathbf{L}$ is used to solve for $\mathbf{L}^{-1}\mathbf{y}$ and $\mathbf{L}^{-1}\mathbf{k}_*$ through forward and backward substitution, which avoids direct matrix inversion and thus enhances numerical stability and computational efficiency, especially in the context of large datasets.

III. GP-based Target Tracking

A. Temporal and Spatial-Temporal GP

Recognising the temporal correlation in the target's motion in a target-tracking scenario is essential. This correlation suggests that recent movements are more closely related than distant ones, leading to adopting time as a primary variable for training GPs and making predictions. For this purpose, time is represented as t, and $\mathbf{x}$ represents the target state.

Given the temporal correlation in target motion, the target state is modelled as:

$$f(t) \sim GP(\bar{f}(t), k(t, t')), \tag{17}$$

where t and t' represent the times for training and testing. The observed value y is given by:

$$y = f(t) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2), \tag{18}$$

Extending to a spatial-temporal context, the model incorporates the previous state:

$$f(\mathbf{x}_{t-1}, t) \sim GP(\bar{f}(\mathbf{x}_{t-1}, t), k(\mathbf{x}_{t-1}, t; \mathbf{x}'_{t-1}, t')), \tag{19}$$

with observations:

$$y = f(\mathbf{x}_{t-1}, t) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2). \tag{20}$$

where $\mathbf{x}_{t-1}$ is the states of the target at $t-1$, the spatial-temporal GP enables incorporating the target's prior state into predictions, thus leveraging spatial and temporal information to enhance tracking accuracy.

B. Gaussian Process for Tracking

The schematic representation of the target tracking methodology using GP and DGP is depicted in Figure 1. The sensor network, composed of a multitude of sensors (e.g., Sensor No. 01, Sensor No. 02, Sensor No. 03, etc.), functions collaboratively to gather data about the state of the target, denoted by (x_t, y_t), as well as accumulating historical observations. This collective endeavour encompasses the integration of real-time measurements obtained at discrete time instances, constituting the foundational data for the GP models.

1) Training Phase: In the training phase, data procured from each sensor is individually processed via a designated GP. The GP model incorporates a carefully selected kernel function, articulating the covariance structure intrinsic to the dataset, alongside an optimisation process through MLE aimed at calibrating the model's hyperparameters to the observed data. It's also mentioned that in the tracking problem, single-dimensional output GP is still used, so the x and y coordinates are calculated separately. Following this, an aggregation mechanism amalgamates the distinct GPs to formulate a unified DGP model, synthesizing the discrete sensor data streams into a consolidated statistical model.

2) Prediction Phase: Upon transitioning to the prediction phase, the GP and DGP model leverages new test data, captured at time t, to refine the posterior distribution regarding the target's state. It computes the GP Posterior to encapsulate predictions based on individual sensor data and the DGP Posterior for an integrated estimation. The latter embodies a holistic approach, harnessing the collective intelligence of the entire sensor network, thus furnishing a more precise and resilient target-tracking mechanism.

C. Kernel Function Selection

The kernel function in GP models encapsulates prior assumptions about the behaviour of target motion [18]. In tracking applications, the choice of the kernel is crucial and must be aligned with the characteristics of the function representing the trajectory. [19] Below, we summarise the kernel choices with their respective mathematical representations:

Smoothness: For smooth trajectories, the Squared Exponential (SE) kernel is a standard choice due to its assumption of smoothness:

$$k_{\text{SE}}(\mathbf{x}, \mathbf{x}_*) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_*\|^2}{2l^2}\right), \tag{21}$$

where l is the length-scale parameter. the length-scale l controls how rapidly the correlation between the function values decreases with distance. A smaller l leads to a more complex model capable of capturing rapid changes, whereas a larger l results in a smoother model.

Periodicity: To model periodic movements, the Periodic kernel can be used:

$$k_{\text{Periodic}}(\mathbf{x}, \mathbf{x}_*) = \exp\left(-\frac{2\sin^2(\pi|\mathbf{x} - \mathbf{x}_*|/p)}{l^2}\right), \tag{22}$$

where p represents the period and l the length-scale.

Linearity: Linear kernels assume a linear relationship between inputs and are defined as:

$$k_{\text{Linear}}(\mathbf{x}, \mathbf{x}_*) = \sigma_b^2 + \sigma_v^2(\mathbf{x} - c)(\mathbf{x}_* - c), \tag{23}$$

with σ_b^2 as the bias term, σ_v^2 the variance, and c the offset.

D. Sliding Window-based Tracking

A sliding window-based tracking method is proposed to manage the voluminous data from local sensors, which might include irrelevant or misleading information. [1] This strategy prioritises recent data, deemed more pertinent for current predictions, by filtering out older, less relevant information. Employing a sliding window to refine the dataset minimises computational demands and enhances the precision of state predictions. This approach streamlines the process by excluding sensors that fail to provide current data from estimating the present state.

IV. Implementation of Gaussian Process Algorithms in Stone Soup

Incorporating GP algorithms into the Stone Soup framework [1] substantially augments sensor networks' data fusion and tracking capabilities. This section delves into the GP centralised and distributed implementation, offering a distinctive comparative analysis using the same datasets. Based on the Stone Soup [1], we can adapt a GP to track entities efficiently across various domains, such as space, air, land, and maritime. We have implemented centralised and distributed GP algorithms and validated their performance over Stone Soup case studies. This approach allows us to demonstrate the GP's versatility and effectiveness in a controlled setting without implying that the simulations or benchmarks are inherently tied to the Stone Soup framework. Stone Soup's state transition models, or state-space models, facilitate the representation of various entity dynamics, which can be customised according to the application domain.

By utilising GP and DGP to estimate and predict target states, we harness Stone Soup's focus on versatility and accessibility. This empowers researchers to easily adopt sophisticated tracking methodologies, even as they work with data simulated through PF techniques, thereby ensuring a comprehensive evaluation of GP's efficacy in practical applications.

A. Measurement Model

The Stone Soup framework employs several generic, low-fidelity measurement models applicable across various tracking and data fusion algorithms. These models do not rely on GP for their derivation but are designed to be compatible with a broad spectrum of state estimation methodologies. They allow us to generate experimental data and simulate sensor data reception.

B. GP Types

The GP code includes the *GaussianProcess* class, which features two types of GPs: GP and DGP. They extend the GP framework to suit various sensor network environments, providing versatile tools for state estimation. The *GaussianProcess* class is instrumental in realizing a comprehensive GP tracking framework. It enables the following critical functionalities:

- **Kernel Function Selection:** The class allows for choosing kernel functions, such as the SE, Matérn, and Rational Quadratic, each capturing different aspects of the data's covariance structure.
- **Hyperparameter Optimisation:** The class includes methods for fitting hyperparameters to enhance model performance. These methods employ gradient descent optimisation techniques to maximise the likelihood of the observed data under the GP model.
- **Posterior Probability Calculation:** For continuous learning and prediction updating, the class computes posterior probabilities. It uses Bayesian inference to update the model with new data, maintaining an evolving understanding of the state space.
- **Aggregation:** Aggregation in the context of DGP is a critical step in which individual GP predictions are combined, derived from different sensors' data. This process involves integrating the separate GP models, each representing a subset of the sensor network, into a unified model. The aggregation is crucial for synthesising a comprehensive understanding of the state space from diverse and distributed sensor data.

C. GP Tracking

The *GP tracking* class employs the *Gaussianprocess* class, combining offline training with online prediction. This dual capability is crucial for adapting to both historical and real-time data. The system includes a sliding window feature, enhancing its capability to handle dynamic data flows:

$$\mathbf{x}_{t+1|t} = f(\mathbf{x}_{t|t-1}) + \varepsilon \tag{24}$$

where f represents the state transition function and w_t the process noise, which is usually Gaussian. The following representation encapsulates the functionality of GP tracking:

$$p(\mathbf{x}_{t+1}|D_t) = \int p(\mathbf{x}_{t+1}|x_t, \boldsymbol{\theta}) p(\mathbf{x}_t|D_t) d\mathbf{x}_t \tag{25}$$

where D_t denotes the data available up to time t, $\boldsymbol{\theta}$ represents the hyperparameters of the kernel function, and $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$ are the states at times t and $t+1$, respectively. This integral reflects the core of the GP approach, updating the state estimate after observing new data.

D. Sensor Network Customisation

The *SensorNetwork* class in Stone Soup allows users to design and tailor sensor networks to their specific requirements, optimising the application of GP across different operational scenarios. This level of customisation is key to leveraging the full potential of GP for complex data fusion tasks and state estimation accuracy.

E. Implementation of the Gaussian Process

Based on the above descriptions, Algorithm 1 summarises the implementation of GP tracking in Stone Soup.

Algorithm 1 Gaussian Process Tracking Algorithm

1: **Initialisation:**
- Define the initial GP model with the chosen kernel function.
- Initialise sliding window with the initial data set.

2: **for** $t = 1$ to T **do**

3: **Model Training**
- Input: Historical data from t_0 to $t-1$.
- Process: Train GP or DGP.
- Output: Trained GP or DGP model.

4: **Prediction**
- Input: Trained GP model, current time t.
- Process: Predict state at time t using the GP model.
- Output: Predicted state at t.

5: **Update Sliding Window**

6: **end for**

V. Performance Validation and Evaluation

In this section, we evaluate the tracking performance of GP and DGP-based trackers using the Stone Soup framework. We use the standard SE kernel for our analyses and evaluate the trackers' performance in three cases of simulations. The first case follows a nearly constant velocity model. The second case is based on a cubic polynomial trajectory, The third one follows a unmanned aerial vehicle (UAV) trajectory. The code for the simulations can be accessed at https://github.com/Lyuchenyi/Gaussian-process-tracking.

A. Nearly-Constant Velocity Case

For the nearly constant velocity model, we utilise Stone Soup to generate the ground truth and the target measurements. The data generation process is formulated as follows:

$$\mathbf{x}_k = \mathbf{F}_k x_{k-1} + \mathbf{w}_k, \quad \mathbf{w}_k \sim \mathcal{N}(0, \mathbf{Q}_k), \tag{26}$$

where

$$\mathbf{F}_k = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix}, \quad \mathbf{Q}_k = q \begin{pmatrix} \frac{\Delta t^3}{3} & \frac{\Delta t^2}{2} \\ \frac{\Delta t^2}{2} & \Delta t \end{pmatrix}. \tag{27}$$

In this experiment, the ground truth path begins at (0,0). It progresses to the northeast, with each step representing one unit of distance in both the x and y dimensions, with Gaussian noise introduced to simulate realistic sensor data. The noise parameter q is set to 0.05 as the magnitude of the noise per Δt-sized timestep. The simulation of the measurement model is a Gaussian measurement model executed in a 2D setting by combining two 1D constant velocity models with $q_x = q_y = 0.05$ using the 'CombinedLinearGaussianTransitionModel'.

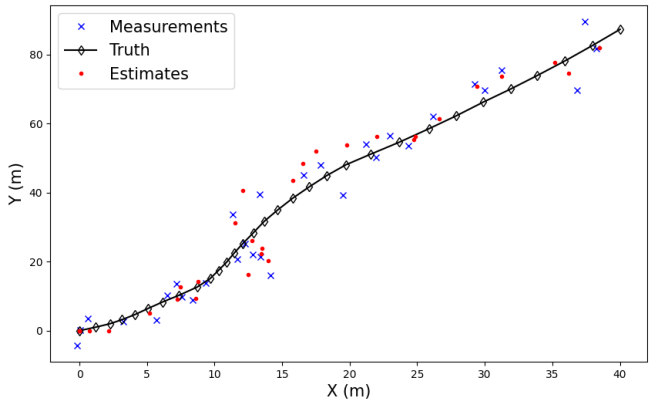

Fig. 2. GP tracking for nearly-constant velocity

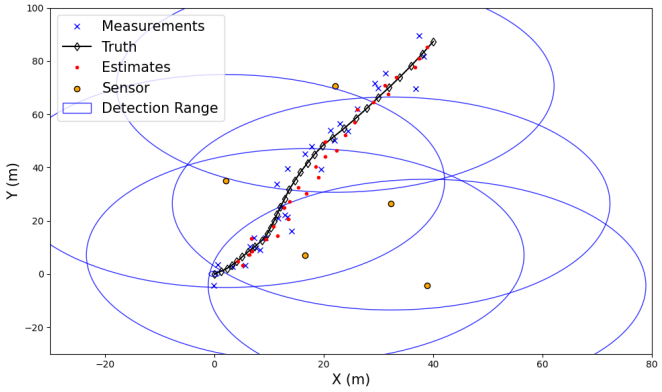

Fig. 3. DGP tracking for nearly-constant velocity

Figure 2 and 3 present the GP and DGP tracking estimates for a nearly constant velocity model. The blue dot represents the Gaussian noise measurement trajectory, while the red indicates the GP tracking estimate. The close alignment of the GP estimates with the true trajectory demonstrates the precision and effectiveness of the GP and DGP model in tracking the path, even with the presence of measurement noise and other operational uncertainties.

B. Cubic Trajectory Case Study

First, we plot the ground truth of one target moving on the Cartesian 2D plane. The target moves in a cubic function. The measurements are made of the ground truth. The Gaussian

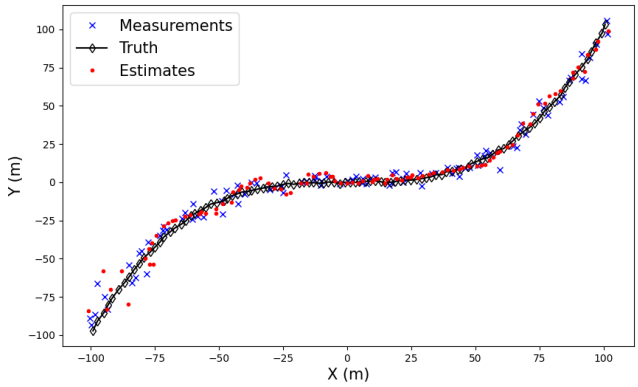

Fig. 4. GP tracking for cubic trajectory

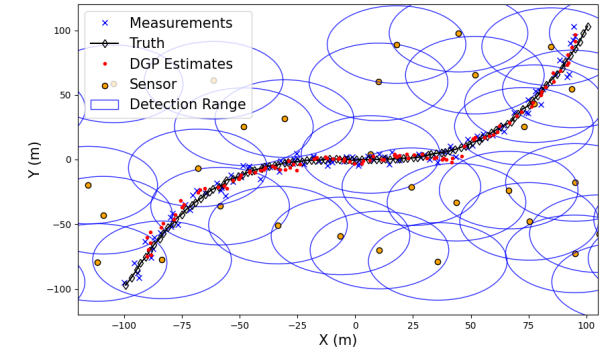

Fig. 5. DGP tracking for cubic trajectory

measurements model has an error matrix of variance 5 in both dimensions. [1]

Figures 4 and 5 present the cubic trajectory model's GP and DGP tracking estimates. The dashed blue dot represents the noisy measurement trajectory, while the dotted red dot indicates the GP tracking estimate. The close alignment of the GP estimates with the true trajectory demonstrates the precision and effectiveness of the GP model in tracking the path.

C. UAV Tracking Case Study

Our experiment employed GP and DGP for trajectory estimation tasks, utilising a dataset provided in the UAV demonstration from Stone-Soup documentation [20].

Figure 6 illustrates the measurements and ground truth data, with measurements indicated by blue crosses and the true trajectory by a black line with diamond markers. It also visualises the GP tracking estimates juxtaposed with the true trajectory. The estimates are represented with red dots, and the true trajectory is again shown with a black line with diamond markers.

Figure 7 presents the sensor positions and their ranges alongside the DGP estimated trajectory. The sensors are marked with yellow dots, and blue-shaded regions depict their sensing

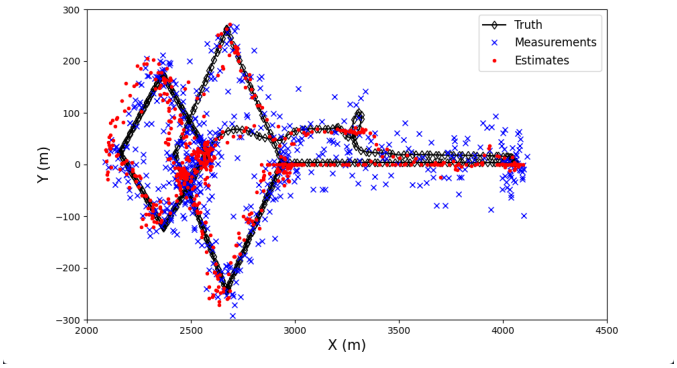

Fig. 6. UAV measurements and ground truth

ranges. The DGP estimates are plotted with red dots, while a black line with diamond markers indicates the true trajectory.

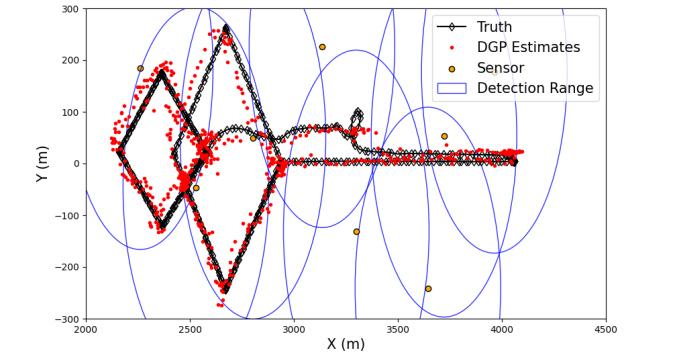

Fig. 7. Sensor positions and DGP estimates

Figure 8 presents the X-axis trajectory estimation by comparing the true and estimated trajectories from GP and DGP. A dashed blue line indicates the GP estimation, while the DGP estimates are marked with orange dots. A solid black line with diamond markers represents the true trajectory.

Figure 9 shows the Y-axis trajectory estimation, where the GP and DGP estimations are superimposed on the true trajectory. Similar to the Y-axis estimation, the GP estimation follows a dashed blue line, the DGP estimates are shown with orange dots, and the true trajectory is displayed with a solid black line.

D. Numerical Result

The results demonstrate the efficacy of GP and DGP in accurately estimating target trajectories, with the DGP algorithm providing a refined estimation by incorporating sensor network information. These figures underscore the GP model's potential to provide reliable estimates for UAV tracking in real-time applications. Our analysis indicates that despite the measurement noise, the DGP algorithm exhibits high accuracy and robustness in processing sensor data for tracking. The results are acquired from 100 independent Monte

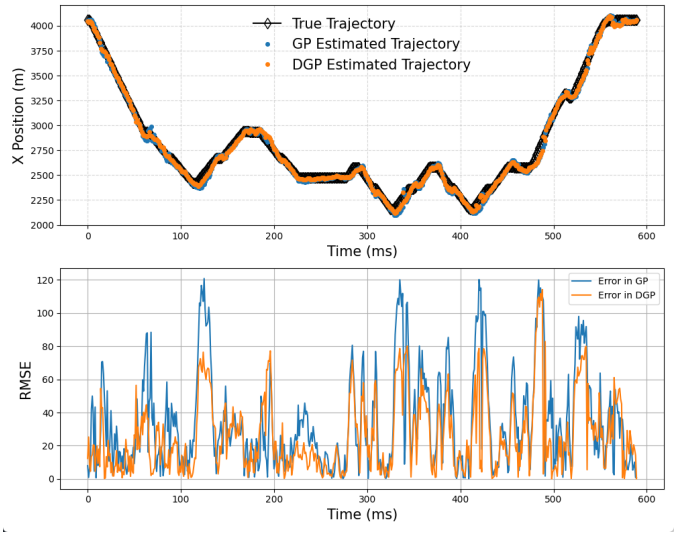

Fig. 8. Comparative analysis of UAV trajectory estimations in X-axis

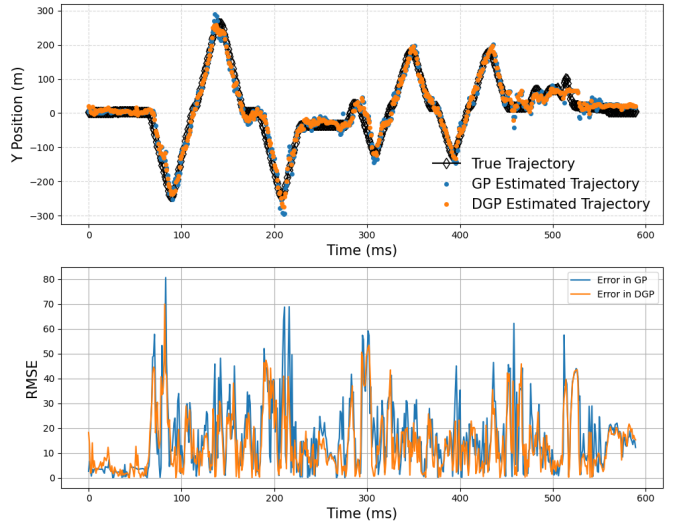

Fig. 9. Comparative analysis of UAV trajectory estimations in Y-axis

Carlo simulations, and the average values are presented. The trajectory is fixed, and the measurement noise variance is set to 5.

TABLE I
RMSE RESULTS OF GP AND DGP METHODS ACROSS DIFFERENT SCENARIOS

Scenario		Method	
		GP (m)	DGP (m)
Nearly Constant Velocity	X	2.10	4.43
	Y	2.08	6.56
Cubic trajectory	X	1.98	4.03
	Y	2.86	7.84
UAV	X	45.93	38.22
	Y	20.76	75.42

In scenarios with lower complexity, such as those involving nearly constant velocity and cubic trajectories, the GP method

outperforms its counterpart, demonstrating superior accuracy with lower RMSE values. Conversely, in the more intricate UAV scenario, both GP and DGP methods register elevated RMSE figures, albeit with distinct patterns of error distribution: the DGP method incurs higher errors along the Y-axis. At the same time, it fares better on the X-axis when compared to GP. Overall, the GP method showcases greater robustness, standing out as the more reliable approach in general terms.

Notwithstanding the broader trend, the DGP method's use of independent hyperparameters adjusted by multiple sensors does show promise under certain conditions, particularly noted in the UAV scenario's X-axis. This indicates that a method's efficacy depends on the application's specificities, including the scenario's complexity and the targeted dimension for prediction accuracy. Such nuanced performance underscores the necessity of method selection tailored to the precise demands of each unique tracking situation.

VI. Conclusions

This paper successfully integrates GP approaches into the Stone Soup framework, a significant advancement in sensor network-based tracking systems. By utilising and adapting existing components within Stone Soup, the paper demonstrates the effective incorporation of GP models, highlighting their ability to enhance data processing and analysis in complex environments. This integration proves Stone Soup's versatility in accommodating advanced algorithms and sets a benchmark for future implementations of similar technologies. The potential for further development and application of these approaches in Stone Soup is a promising avenue for continued research and innovation.

ACKNOWLEDGMENTS

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 those of the authors. They should not be interpreted as representing the official policies, expressed or implied, of the Army Research Laboratory, the MOD, the U.S. Government or the U.K. Government. The U.S. and U.K. governments 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 Systems' and the National Science Foundation under Grant USA NSF ECCS 1903466. For open access, the authors have applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising.

References

[1] S. Hiscocks, O. Harrald, J. Barr, N. Perree, L. Vladimirov, M. Harris, gawebb dstl, T. Glover, R. Green, O. Rosoman, etfrogers dstl, idorrington dstl, J. Wright, spike, E. Hunter, B. Fraser, H. Pritchett, jjosborne dstl, P. Carniglia, and C. Sherman, "dstl/stone-soup: v1.2," 2024.

[2] 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.

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

[4] J. Carpenter, P. Clifford, and P. Fearnhead, "Improved particle filter for nonlinear problems," *IEE Proceedings-Radar, Sonar and Navigation*, vol. 146, no. 1, pp. 2–7, 1999.

[5] B. Oakes, D. Richards, J. Barr, and J. Ralph, "Double deep Q networks for sensor management in space situational awareness," in *2022 25th International Conference on Information Fusion (FUSION)*, pp. 1–6, 2022.

[6] S. Hiscocks, J. Barr, N. Perree, J. Wright, H. Pritchett, O. Rosoman, M. Harris, R. Gorman, S. Pike, P. Carniglia, L. Vladimirov, and B. Oakes, "Stone soup: No longer just an appetiser," in *Proceedings of the 26th International Conference on Information Fusion, FUSION 2023, Charleston, SC, USA, June 27-30, 2023*, pp. 1–8, IEEE, 2023.

[7] J. Hiles, S. M. O'Rourke, R. Niu, and E. Blasch, "Implementation of ensemble kalman filters in stone-soup," in *Proceedings of the 24th IEEE International Conference on Information Fusion, FUSION 2021, Sun City, South Africa, November 1-4, 2021*, pp. 1–8, IEEE, 2021.

[8] X. Liu, C. Lyu, J. George, T. Pham, and L. Mihaylova, "A learning distributed Gaussian process approach for target tracking over sensor networks," in *Proceedings of the 25th International Conference on Information Fusion, FUSION 2022, Linköping, Sweden, July 4-7, 2022*, pp. 1–8, IEEE, 2022.

[9] C. Lyu, X. Liu, and L. Mihaylova, "Efficient factorisation-based Gaussian process approaches for online tracking," in *Proceedings of the 25th International Conference on Information Fusion, FUSION 2022, Linköping, Sweden, July 4-7, 2022*, pp. 1–8, IEEE, 2022.

[10] X. Liu, L. Mihaylova, J. George, and T. Pham, "Gaussian process upper confidence bounds in distributed point target tracking over wireless sensor networks," *IEEE J. Sel. Top. Signal Process.*, vol. 17, no. 1, pp. 295–310, 2023.

[11] J. S. Wright, J. R. Hopgood, M. E. Davies, I. K. Proudler, and M. Sun, "Implementation of adaptive kernel kalman filter in stone soup," in *Proceedings of the 2023 Sensor Signal Processing for Defence Conference (SSPD)*, pp. 1–5, 2023.

[12] H. Liu, Y. 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.

[13] M. Deisenroth and J. W. Ng, "Distributed gaussian processes," in *International Conference on Machine Learning*, pp. 1481–1490, PMLR, 2015.

[14] R. B. Gramacy, "laGP: large-scale spatial modeling via local approximate Gaussian processes in R," *Journal of Statistical Software*, vol. 72, pp. 1–46, 2016.

[15] G. E. Hinton, "Training Products of Experts by Minimizing Contrastive Divergence," *Neural Computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[16] S. E. Yuksel, J. N. Wilson, and P. D. Gader, "Twenty years of mixture of experts," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 8, pp. 1177–1193, 2012.

[17] Y. Cao and D. J. Fleet, "Generalized product of experts for automatic and principled fusion of Gaussian process predictions," *arXiv preprint arXiv:1410.7827*, 2014.

[18] A. Wilson and R. Adams, "Gaussian process kernels for pattern discovery and extrapolation," in *Proc. of the International Conference on Machine Learning*, pp. 1067–1075, PMLR, 2013.

[19] M. Debruyne, M. Hubert, and J. A. Suykens, "Model selection in kernel based regression using the influence function," *Journal of machine learning research.-Cambridge, Mass.*, vol. 9, pp. 2377–2400, 2008.

[20] "UAV tracking demonstation." https://stonesoup.readthedocs.io/en/v0.1b5/auto_demos/UAV_tutorial.html.