Implementation of Adaptive Kernel Kalman Filter in Stone Soup

James S. Wright, James R. Hopgood, Mike E. Davies, Ian K. Proudler, Mengwei Sun

***Abstract*—The recently proposed adaptive kernel Kalman filter (AKKF) is an efficient method for highly nonlinear and high-dimensional tracking or estimation problems. Compared to other nonlinear Kalman filters (KFs), the AKKF has significantly improved performance, reducing computational complexity and avoiding resampling. It has been applied in various tracking scenarios, such as multi-sensor fusion and multi-target tracking. By using existing Stone Soup components, along with newly established kernel-based prediction and update modules, we demonstrate that the AKKF can work in the Stone Soup platform by being applied to a bearing–only tracking (BOT) problem. We hope that the AKKF will enable more applications for tracking and estimation problems, and the development of a whole class of derived algorithms in sensor fusion systems.**

***Index Terms*—Adaptive kernel Kalman filter, Tracking, Stone Soup**

I. Introduction

Target tracking in sensor networks is a fundamental problem that arises in a variety of applications, including surveillance, environmental monitoring, and military operations. The objective of target tracking is to estimate the location, velocity, and other relevant parameters of a target based on noisy and incomplete measurements obtained from one or more sensors. Nonlinear and non-Gaussian system models and measurement noise pose challenges that have been addressed using Bayesian filters, such as the extended Kalman filter (EKF) [1], unscented Kalman filter (UKF) [2], and particle filter (PF) [3]. The adaptive kernel Kalman filter (AKKF) has been proposed [4], [5] and demonstrated to achieve better performance, reduced computational complexity, and avoidance of resampling, especially in high nonlinear and high-dimensional problems. It has been applied to various tracking scenarios, such as multi-target tracking [6], multi-sensor fusion [7], and magnetic anomaly detection-based metallic target tracking [8].

M. W. Sun, M. E. Davies and J. R. Hopgood are with Institute of Digital Communications, University of Edinburgh, Edinburgh, EH9 3FG, U.K. E-mail: (msun; mike.davies; james.hopgood)@ed.ac.uk.

J. S. Wright is with Dstl, Dstl Porton Down, Bldg 5, Rm 101, Salisbury, Wiltshire, SP4 0JQ, U.K. E-mail: jwright2@dstl.gov.uk.

I. Proudler is with the Centre for Signal & Image Processing (CeSIP), Department of Electronic & Electrical Engineering, University of Strathclyde, Glasgow, G1 1XW, U.K. E-mail: ian.proudler@strath.ac.uk.

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) Grant number EP/S000631/1; and the MOD University Defence Research Collaboration (UDRC) in Signal Processing.

Stone Soup [9], [10] is a software project that provides a framework for developing and testing algorithms for the target tracking and state estimation community. The project prioritises flexibility over optimisation to aid in selecting components and algorithms for real-world problems. Stone Soup has a number of components used to both build algorithms and enable an environment for testing and assessment. For example, the Kalman filter (KF), EKF, UKF, and PF have been implemented in the Stone Soup framework [11].

In this work, our goal is to demonstrate the efficacy of the AKKF, by using the flexibility and metrics provided by Stone Soup while showing how augmentation of the framework can be undertaken. Specifically, we aim to realise adaptive updates of the empirical kernel mean embeddings (KMEs) for posterior probability density functions (pdfs) using the AKKF, which is executed in the state space, measurement space, and reproducing kernel Hilbert space (RKHS). In the state space, we generate the proposal state particles and propagate them through the motion model to get the state particles. In the measurement space, the measurement particles are achieved by propagating the state particles through the measurement model. We map all these particles into RKHSs as feature mappings and linearly predict and update the corresponding kernel weight mean vector and covariance matrix to approximate the empirical KMEs of the posterior pdfs in the RKHS.

Our contributions include the first attempt to implement the AKKF in Stone Soup, particularly in their simplest reference forms. We designed the key new components required in Stone Soup for the AKKF to run, including defining different component types, such as the `KernelParticleState`, `Kernel`, `AdaptiveKernelKalmanPredictor` and `AdaptiveKernelKalmanUpdater`. Stone Soup is designed for easy inheritance, which provides the design choice of the `Kernel` class to enable the use of different kernels and will permit the AKKF to be used for a wide variety of dynamic and measurement models, as well as future extensions for joint tracking and parameter estimation problems.

The paper is organised as follows: Section II describes the AKKF's general operation mathematically. Section III provides additional analysis and explanation of the AKKF implementation in Stone Soup. Section IV presents simulation results for a bearing–only tracking (BOT) problem, highlighting the improved performance of AKKF compared to the PF with few particles. Lastly, Section V outlines future extensions to our Stone Soup contribution.

This is a peer-reviewed, accepted author manuscript of the following conference paper:
Wright, J. S., Hopgood, J. R., Davies, M. E., Proudler, I. K., & Sun, M. (2023). Implementation of adaptive kernel Kalman filter in stone soup. In 2023 Sensor Signal Processing for Defence Conference (SSPD) IEEE. https://doi.org/10.1109/sspd57945.2023.10256739

II. Adaptive kernel Kalman filter (AKKF)

The AKKF was inspired by the KME and KF and originally formulated in 2021 to address shortcomings of existing Bayesian filters for tracking problems in nonlinear systems [4], [5]. In the AKKF, the posterior pdf is approximated with particles and weights, but not in the state space as in the PF. Specifically, the pdf is embedded into an RKHS as an empirical KME,

$$p(\mathbf{x}_k \mid \mathbf{y}_{1:k}) \rightarrow \hat{\mu}^{+}_{\mathbf{x}_k} = \Phi_k \mathbf{w}^{+}_k, \tag{1}$$

where $\mathbf{x}_k$ represents the hidden state at the k-th time slot, and $\mathbf{y}_k$ is the corresponding observation. The feature mappings of particles $\mathbf{x}_k^{\{1:M_\mathrm{A}\}}$ are represented as Φ_k, i.e., $\Phi_k = \left[\phi_{\mathbf{x}}(\mathbf{x}_k^{\{1\}}), \ldots, \phi_{\mathbf{x}}(\mathbf{x}_k^{\{M_\mathrm{A}\}})\right]$, where M_A is the number of particles, and the weight vector $\mathbf{w}^{+}_k$ includes M_A non-uniform weights. The KME $\hat{\mu}^{+}_{\mathbf{x}_k}$ in (1) is an element in the RKHS that captures the feature value of the distribution.

A. How to choose different kernels in the AKKF

The feature mapping $\phi_{\mathbf{x}}(\mathbf{x})$ of different kernels can be chosen depending on the specific applications, such as linear kernels, polynomial kernels, including quadratic and quartic kernels, and Gaussian kernels. The following summarises kernel functions with the assumption that $\mathbf{x} = [x_1, \ldots, x_n]^\mathrm{T}$ and $\mathbf{x}' = \left[x'_1, \ldots, x'_n\right]^\mathrm{T}$.

- Linear kernel function

$$\mathrm{k}(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\mathrm{T}\mathbf{x}'. \tag{2}$$

  The linear kernel can capture the first-order moments of a distribution, such as the mean and variance. This kernel is often used in linear regression. It can be effective when the data is well-modelled by a linear relationship.
- Quadratic kernel function

$$\mathrm{k}(\mathbf{x}, \mathbf{x}') = \left(\alpha\langle \mathbf{x}, \mathbf{x}'\rangle + c\right)^2. \tag{3}$$

  Here, $c \geq 0$ is a free parameter that trades off the influence of higher-order versus lower-order terms in the polynomial. The parameter α represents the slope that scales the input vectors' dot product. These parameters allow the kernel to emphasise or de-emphasise the importance of different input features. The quadratic kernel can capture the second-order moments of a distribution, such as the covariance and correlations between pairs of variables. The quadratic kernel is appropriate when the data is nonlinear but relatively simple.
- Quartic kernel function

$$\mathrm{k}(\mathbf{x}, \mathbf{x}') = \left(\alpha\langle \mathbf{x}, \mathbf{x}'\rangle + c\right)^4. \tag{4}$$

  The quartic kernel can capture higher-order moments beyond the mean and covariance, such as skewness and kurtosis. The quartic kernel can be used when the data is highly nonlinear and complex.
- Gaussian kernel function

$$\mathrm{k}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right). \tag{5}$$

  Here, σ is a parameter that determines the width of the Gaussian kernel. The Gaussian kernel can capture the mean and covariance of the data, as well as the smoothness and correlation structure. The Gaussian kernel is appropriate when the data is highly nonlinear and complex, and the relationship between the variables is not well-defined.

B. How to implement the AKKF

The AKKF includes three modules, as shown in Fig. 1: a predictor that utilises both prior and proposal information, at time $k-1$, to update the prior state particles and predict the kernel weight mean and covariance at time k, an updater employs the predicted values to update the kernel weight and covariance, and an updater generates the proposal state particles.

1) Predictor takes prior and proposal: The predictor is executed in the state space and kernel space, i.e., RKHS. Suppose that the prior and proposal state particles at time $k-1$ are represented as $\mathbf{x}_{k-1}^{\{i=1:M_\mathrm{A}\}}$ and $\tilde{\mathbf{x}}_{k-1}^{\{i=1:M_\mathrm{A}\}}$, respectively. Their feature mappings in RKHSs are given by:

$$\begin{aligned}\Phi_{k-1} &= \left[\phi_{\mathbf{x}}(\mathbf{x}_{k-1}^{\{1\}}), \ldots, \phi_{\mathbf{x}}(\mathbf{x}_{k-1}^{\{M_\mathrm{A}\}})\right]\\ \Psi_{k-1} &= \left[\phi_{\mathbf{x}}(\tilde{\mathbf{x}}_{k-1}^{\{1\}}), \ldots, \phi_{\mathbf{x}}(\tilde{\mathbf{x}}_{k-1}^{\{M_\mathrm{A}\}})\right].\end{aligned}$$

■ At time k, the prior state particles in the state space are generated by passing the proposal particles at time $k-1$, i.e., $\tilde{\mathbf{x}}_{k-1}^{\{i=1:M_\mathrm{A}\}}$, through the motion model as

$$\mathbf{x}_k^{\{i\}} = \mathtt{f}\left(\tilde{\mathbf{x}}_{k-1}^{\{i\}}, \mathbf{u}_k^{\{i\}}\right), \tag{6}$$

where $i = 1 \ldots M_\mathrm{A}$, $\mathbf{u}_k^{\{i\}}$ represents process noise samples which are drawn from the process noise distributions.

■ In RKHS, $\mathbf{x}_k^{\{i=1:M_\mathrm{A}\}}$ are mapped to the RKHS as

$$\Phi_k = \left[\phi_{\mathbf{x}}(\mathbf{x}_k^{\{1\}}), \ldots, \phi_{\mathbf{x}}(\mathbf{x}_k^{\{M_\mathrm{A}\}})\right].$$

Based on [4], the transition matrix Γ_k, which represents the change of sample representation, is calculated as

$$\Gamma_k = \left(K_{\tilde{\mathbf{x}}\tilde{\mathbf{x}}} + \lambda_{\tilde{K}} I\right)^{-1} K_{\tilde{\mathbf{x}}\mathbf{x}}, \tag{7}$$

where the Gram matrices are $K_{\tilde{\mathbf{x}}\tilde{\mathbf{x}}} = \Psi_{k-1}^\mathrm{T}\Psi_{k-1}$ and $K_{\tilde{\mathbf{x}}\mathbf{x}} = \Psi_{k-1}^\mathrm{T}\Phi_{k-1}$. The regularisation parameter $\lambda_{\tilde{K}}$ is used to stabilise the inverse of $K_{\tilde{\mathbf{x}}\tilde{\mathbf{x}}}$, and I is the identity operator matrix. In practice, $K_{\tilde{\mathbf{x}}\tilde{\mathbf{x}}}$ can become ill-conditioned and challenging to invert, leading to numerical instability and poor performance. To address this, the regularisation parameter $\lambda_{\tilde{K}}$ is added to the diagonal of $K_{\tilde{\mathbf{x}}\tilde{\mathbf{x}}}$, which makes it better conditioned and easier to invert. The value of this regularisation parameter is usually chosen by cross-validation or other optimisation methods. Then, the predictive kernel weight vector, denoted as $\mathbf{w}^{-}_k$, and covariance matrix, denoted as S^{-}_k, are calculated as

$$\mathbf{w}^{-}_k = \Gamma_k \mathbf{w}^{+}_{k-1}, \tag{8}$$

$$S^{-}_k = \Gamma_k S^{+}_{k-1} \Gamma_k^\mathrm{T} + V_k. \tag{9}$$

Here, $\mathbf{w}^{+}_{k-1}$ and S^{-}_k are the posterior kernel weight mean vector and covariance matrix at time $k-1$, respectively, and V_k

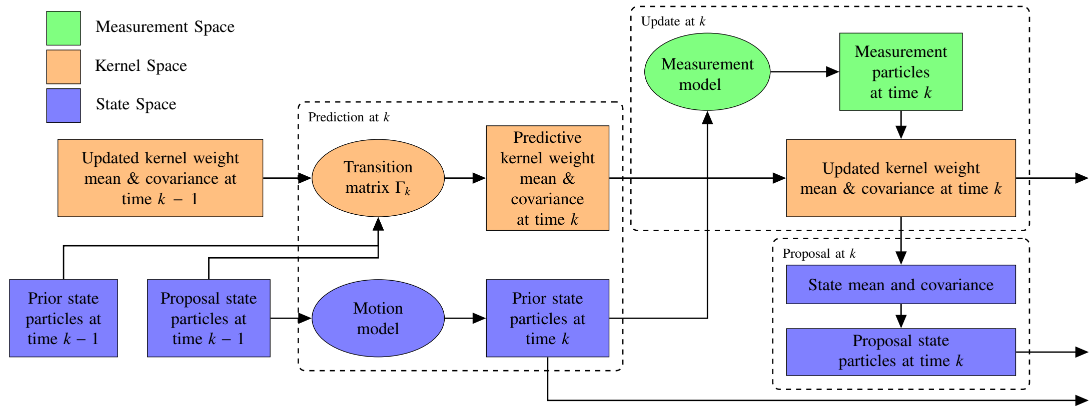


Fig. 1: Flow diagram of the AKKF.

represents the finite matrix representation of the transition residual matrix [4].

2) Updater uses prediction: The updater is executed in the measurement space and RKHS.

■ In the measurement space, the measurement particles are generated according to the measurement model as

$$\mathbf{y}_k^{\{i\}} = \mathrm{h}\left(\mathbf{x}_k^{\{i\}}, \mathbf{v}_k^{\{i\}}\right) \tag{10}$$

where $\mathbf{v}_k^{\{i\}}$ represent measurement noise samples which are drawn from the measurement noise distribution.

■ In RKHS, $\mathbf{y}_k^{\{i=1:M_\text{A}\}}$ are mapped in the RKHS as

$$\Upsilon_k = \left[\phi_{\mathbf{y}}(\mathbf{y}_k^{\{1\}}), \dots, \phi_{\mathbf{y}}(\mathbf{y}_k^{\{M_\text{A}\}})\right].$$

The posterior kernel weight vector and covariance matrix are updated as

$$\mathbf{w}_k^+ = \mathbf{w}_k^- + Q_k\left(\mathbf{g}_{\mathbf{y}\mathbf{y}_k} - G_{\mathbf{y}\mathbf{y}}\mathbf{w}_k^-\right) \tag{11}$$

$$S_k^+ = S_k^- - Q_k G_{\mathbf{y}\mathbf{y}} S_k^- \tag{12}$$

$$Q_k = S_k^-\left(G_{\mathbf{y}\mathbf{y}} S_k^- + \kappa I\right)^{-1}. \tag{13}$$

Here, $G_{\mathbf{y}\mathbf{y}} = \Upsilon_k^\mathrm{T}\Upsilon_k$, $\mathbf{g}_{\mathbf{y}\mathbf{y}_k} = \Upsilon_k^\mathrm{T}\phi_{\mathbf{y}}(\mathbf{y}_k)$ is the kernel vector of the measurement at time k, and κ is a regularisation parameter to ensure the inverse is well-defined. The kernel Kalman gain operator denoted as Q_k is derived by minimising the trace of the posterior covariance operator [4]. Then, the empirical KME of $p(\mathbf{x}_k|\mathbf{y}_{1:k})$ is calculated as (1).

3) Proposal generated in updater: The proposal is executed in the state space.

■ The AKKF replaces $\mathbf{x}_k^{\{i=1:M_\text{A}\}}$ by new weighted proposal particles $\tilde{\mathbf{x}}_k^{\{i=1:M_\text{A}\}}$ to approximate the KME that can be exactly propagated through the non-linearity. The proposal particles are generated according to the importance distribution as

$$\tilde{\mathbf{x}}_k^{\{i=1:M_\text{A}\}} \sim \mathcal{N}\left(\mathbb{E}\left(X_k\right), \mathrm{Cov}\left(X_k\right)\right). \tag{14}$$

In Stone Soup implementation, the state vector's mean and covariance for proposal are approximated using

$$\mathbb{E}\left(X_k\right) = X_k\mathbf{w}_k^+ \tag{15}$$

$$\mathrm{Cov}\left(X_k\right) = X_k S_k^+ X_k^\mathrm{T}, \tag{16}$$

where $X_k = \mathbf{x}_k^{\{i=1:M_\text{A}\}}$. We draw proposal particles from a Gaussian distribution for convenience, but other distributions with similar statistics could also be used. These particles are used to capture the key probability mass of the posterior pdf. It is not equivalent to approximating the posterior pdf with a Gaussian, but rather an adaptive change of basis within the feature space through a simple linear mapping.

III. Implementation in Stone Soup

The main goal behind the development of the Stone Soup framework is the ease of collaboration, consistent metrics and open standards. This enables fast prototyping and user-friendliness for researchers. To achieve this widespread adoption, it is crucial that the components are modular and the interfaces are consistent. Through this standardisation, users can utilise algorithms and components without requiring a full understanding the algorithms. Stone Soup follows an object-oriented modular approach with *AbstractClass* forming the base class with a `DerivedClass` inheriting its properties and methods from the abstract class. This inheritance is important to preserve the fundamental methods required of a class. For example, all *Predictor* classes must have a `predict()` method and all *Updater* classes must have an `update()` method.

The following subsections highlight the key new components required in the Stone Soup framework for AKKF to run. More details of the implementation and a tutorial can be found in [12].

A. The `KernelParticleState` Types

The `KernelParticleState` inherits the functionality of the *ParticleState* and adds the `kernel_covar` property as defined in (9) and (12).

B. The `Kernel` Types

The *Kernel* class provides a transformation from state space into the RKHS represented in (8) by $K_{\tilde{\mathbf{x}}\tilde{\mathbf{x}}}$ and $K_{\tilde{\mathbf{x}}\mathbf{x}}$ or a transformation from measurement space into the KME space represented in (11) by $G_{\mathbf{y}\mathbf{y}}$ and $\mathbf{g}_{\mathbf{y}\mathbf{y}_k}$. The kernel can be represented as either a polynomial or a Gaussian kernel. The polynomial kernels, `QuadraticKernel` and `QuarticKernel` have the following properties:

- `c`: is the parameter that trades off the influence of higher-order versus lower-order terms in the polynomial. c in (3) and (4),
- `ialpha`: is the inverse of α and is the slope parameter that controls the influence of the dot product on the kernel value. $1/\alpha$ in (3) and (4).

The Gaussian kernel (`GaussianKernel`) has the following property

- `variance`: The variance parameter of the Gaussian kernel. σ^2 in (5).

C. `Predictor` Types

As discussed previously, every `Predictor` class inherits from the base class *Predictor*. All *Predictors* accept a prior *State*, require a `predict()` method and return a *StatePrediction*. Since the AKKF is a derivative of the Kalman filter the base class to inherit from is the *KalmanPredictor*. This allows the framework to distinguish the different class components.

The `AdaptiveKernelKalmanPredictor` is a subclass of *KalmanPredictor* and inherits the methods and properties of the *KalmanPredictor*. The `AdaptiveKernelKalmanPredictor` includes the following new properties;

- `lambda_predictor`: $\lambda_{\tilde{K}}$ in (7), is a regularisation parameter to stabilise the inverse of the Gram matrix $K_{\tilde{\mathbf{x}}\tilde{\mathbf{x}}}$. According to the simulation results presented in [4], the tracking performance of the AKKF is relatively insensitive to the values of $\lambda_{\tilde{K}}$ when it falls within the range of $\left[10^{-4}, 10^{-2}\right]$ [4].
- `kernel`: The `Kernel` class which is chosen to be used to map the state space into the kernel space as described in section III-B.

D. `Updater` Types

In a similar way to the *KalmanPredictor* class represented the inheritance for all Kalman predictor subclasses, the *KalmanUpdater* provides the same for updaters. All *Updaters* accept a prediction-measurement pair, require an `update()` method and return a *StateUpdate*. The `AdaptiveKernelKalmanUpdater` is a subclass of *KalmanUpdater* and inherits the methods and properties of the *KalmanUpdater*. The `AdaptiveKernelKalmanUpdater` includes the following new properties;

- `lambda_updater`: κ in (13) is a regularisation parameter to ensure the inverse of $G_{\mathbf{yy}}S_k^-$ is well-defined. The tracking performance of the AKKF is relatively insensitive to κ when $\kappa \sim \left[10^{-4}, 10^{-2}\right]$ [4].
- `kernel`: The `Kernel` class which is chosen to be used to map the measurement space into the kernel space.

E. Implementation

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

Algorithm 1 Adaptive kernel Kalman filter

1: **Initialisation**: Initial particles $\mathbf{x}_0^{\{i=1:M_\text{A}\}}$, Φ_0, $\mathbf{w}_0 = 1/M_\text{A}\,[1,\ldots,1]^\text{T}_{M_\text{A}\times 1}$, $\Psi_0 = \Phi_0$. Kernel type and related parameters.
2: **for** $k = 1 : K$ **do**
3: The predictor
- Input: i.e., $\{\mathbf{x}_{k-1}^{\{i=1:M_\text{A}\}}, \tilde{\mathbf{x}}_{k-1}^{\{i=1:M_\text{A}\}}, \mathbf{w}_{k-1}^+, S_{k-1}^+\}$.
- Process: $\mathbf{x}_k^{\{i\}} = \mathtt{f}\left(\tilde{\mathbf{x}}_{k-1}^{\{i\}}, \mathbf{u}_k^{\{i\}}\right)$, $\mathbf{w}_k^- = \Gamma_k \mathbf{w}_{k-1}^+, \quad S_k^- = \Gamma_k S_{k-1}^+ \Gamma_k^\text{T} + V_k$.
- Output: $\{\mathbf{x}_k^{\{i=1:M_\text{A}\}}, \mathbf{w}_k^-, S_k^-\}$.

4: The updater
- Input: $\{\mathbf{x}_k^{\{i=1:M_\text{A}\}}, \mathbf{w}_k^-, S_k^-\}$.
- Process: $\mathbf{y}_k^{\{i\}} = \mathtt{h}\left(\mathbf{x}_k^{\{i\}}, \mathbf{v}_k^{\{i\}}\right)$, $\mathbf{w}_k^+ = \mathbf{w}_k^- + Q_k\left(\mathbf{g}_{\mathbf{y}y_k} - G_{\mathbf{yy}}\mathbf{w}_k^-\right),\ S_k^+ = S_k^- - Q_k G_{\mathbf{yy}} S_k^-$.
- Output: $\{\mathbf{x}_k^{\{i=1:M_\text{A}\}}, \mathbf{w}_k^+, S_k^+\}$.

5: The proposal generator
- Input: $\{\mathbf{x}_k^{\{i=1:M_\text{A}\}}, \mathbf{w}_k^+, S_k^+\}$.
- Process: $\tilde{\mathbf{x}}_k^{\{i=1:M_\text{A}\}} \sim \mathcal{N}\left(\mathbb{E}\left(X_k\right), \text{Cov}\left(X_k\right)\right)$.
- Output: $\{\mathbf{x}_k^{\{i=1:M_\text{A}\}}, \tilde{\mathbf{x}}_k^{\{i=1:M_\text{A}\}}, \mathbf{w}_k^+, S_k^+\}$.

6: **end for**

IV. Demonstration

In this section, we report the tracking performance of different filters in the Stone Soup platform. The corresponding dynamic state-space model (DSSM) is described as

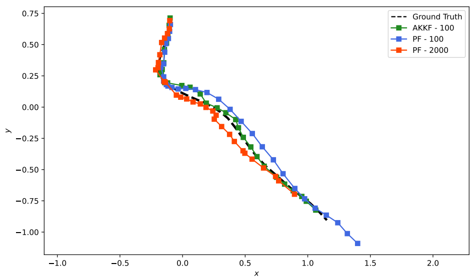
$$\mathbf{x}_k = \begin{bmatrix} 1 & \Delta T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta T \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{x}_{k-1} + \begin{bmatrix} 0.5 & 0 \\ 1 & 0 \\ 0 & 0.5 \\ 0 & 1 \end{bmatrix} \mathbf{u}_k, \tag{17}$$

$$y_k = \tan^{-1}\left(\frac{\eta_k - \eta_s}{\xi_k - \xi_s}\right) + v_k. \tag{18}$$

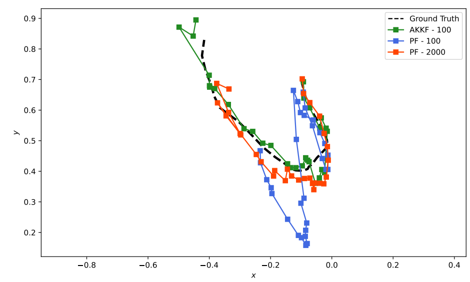
Here, ΔT represents the sampling period, k represents the time index and $k = 1, \ldots, K$. The hidden states are $\mathbf{x}_k = [\xi_k, \dot{\xi}_k, \eta_n, \dot{\eta}_k]^T$, where (ξ_n, η_k) and $(\dot{\xi}_k, \dot{\eta}_k)$ represent the target position and the corresponding velocity in X-axis and Y-axis, y_k is the corresponding observation. The sensor is located at $[\eta_s = 0, \xi_s = 0]$. The process noise $\mathbf{u}_k$ follows Gaussian distribution $\mathbf{u}_k \sim \mathcal{N}(\mathbf{0}, \sigma_u^2 I)$ and $\sigma_u = 0.01$. Following [14], the prior distribution for the initial state is specified as $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{u}_0, P_0)$ with $\mathbf{u}_0 = [-0.5, 0.001, 0.7, -0.05]^\text{T}$ and,

$$P_0 = \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0.005 & 0 & 0 \\ 0 & 0 & 0.1 & 1 \\ 0 & 0 & 0 & 0.01 \end{bmatrix}.$$

Fig. 2 and Fig. 3 displays two representative trajectories and the tracking performance obtained by three filters: the AKKF uses a quartic kernel with 100 particles, the PF with 100 particles, and benchmark performance achieved by the PF with

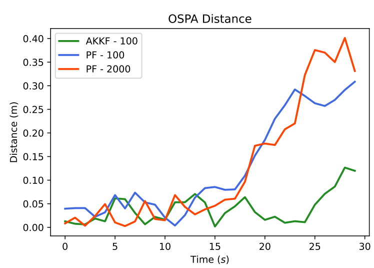


(a)

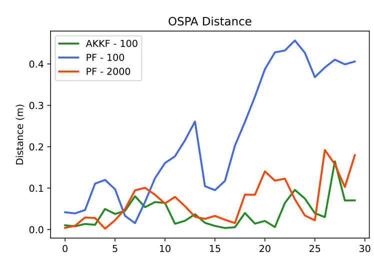


(b)

Fig. 2: BOT tracking of a moving target in two dimensions of Trajectory-1 (a) and Trajectory-2 (b) using the quartic kernel-based AKKF with 100 particles; the PF with 100 particles; and the benchmark performance of the PF with 2000 particles. The physical unit on the X-axis and Y-axis is the 'metre'.



(a)



(b)

Fig. 3: Performance of BOT tracking of a moving target in two dimensions of Trajectory-1 (a) and Trajectory-2 (b) using the OSPA [13] distance.

2000 particles. Trajectory-1 (Fig 3a) shows that the AKKF performs better than both the PF and the benchmark, while Trajectory-2 (Fig 3b) shows that the AKKF performs alongside the benchmark and outperforms the PF. From the results, we arrive at the following conclusions. The implementation of the AKKFs in Stone Soup works properly for the BOT problems, it shows improvement and robustness compared to the PF with the same number of particles.

V. Conclusions

This paper provides the complete implementation of the AKKF in the Stone Soup framework. We utilised and extended existing components to incorporate the AKKF. The new algorithm combines the PF's non-linearity with the KF's analytical properties via KMEs. This shows the compatibility of newer algorithms within the Stone Soup framework and establishes a workflow to support other algorithms being implemented in Stone Soup. The authors are interested in extending the AKKF further [6]–[8] by offering additional tutorials and demonstrations in the Stone Soup documentation [11].

References

[1] H. W. Sorenson, *Kalman Filtering: Theory and Application*. NJ: IEEE: Piscataway, 1985.

[2] S. J. Julier and J. K. Uhlmann, "New extension of the Kalman filter to nonlinear systems," in *Signal Processing, Sensor Fusion, and Target Recognition VI*, I. Kadar, Ed., vol. 3068, International Society for Optics and Photonics. SPIE, 1997, pp. 182 – 193. [Online]. Available: https://doi.org/10.1117/12.280797

[3] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *IEEE Trans. Signal Process.*, vol. 50, no. 2, pp. 174–188, 2002.

[4] M. Sun, M. E. Davies, I. K. Proudler, and J. R. Hopgood, "Adaptive kernel Kalman filter," *IEEE Trans. Signal Process.*, vol. 71, pp. 713–726, 2023.

[5] ——, "Adaptive kernel Kalman filter," in *2021 Sensor Signal Processing for Defence Conference (SSPD)*, 2021, pp. 1–5.

[6] ——, "Adaptive kernel Kalman filter based belief propagation algorithm for maneuvering multi-target tracking," *IEEE Signal Process. Lett.*, vol. 29, pp. 1452–1456, 2022.

[7] ——, "Adaptive kernel Kalman filter multi-sensor fusion," in *2021 IEEE 24th International Conference on Information Fusion (FUSION)*, 2021, pp. 1–8.

[8] M. Sun, T. Horton, R. Hodgskin-Brown, M. E. Davies, I. K. Proudler, and J. R. Hopgood, "Adaptive kernel Kalman filter for magnetic anomaly detection-based metallic target tracking," in *2023 Sensor Signal Processing for Defence Conference (SSPD)*, 2023, submitted, pp. 1–5.

[9] J. Barr, O. Harrald, S. Hiscocks, N. Perree, H. Pritchett, S. Vidal, J. Wright, P. Carniglia, E. Hunter, D. Kirkland, D. Raval, S. Zheng, A. Young, B. Balaji, S. Maskell, M. Hernandez, and L. Vladimirov, "Stone Soup open source framework for tracking and state estimation: enhancements and applications," in *Signal Processing, Sensor/Information Fusion, and Target Recognition XXXI*, I. Kadar, E. P. Blasch, and L. L. Grewe, Eds., vol. 12122, International Society for Optics and Photonics. SPIE, 2022, p. 1212205. [Online]. Available: https://doi.org/10.1117/12.2618495

[10] S. Hiscocks, O. Harrald, J. Barr, N. Perree, L. Vladimirov, R. Green, E. Rogers, I. Dorrington, E. Hunter, J. Wright, O. Rosoman, H. Pritchett, J. Osborne, P. Carniglia, C. England, L. Flaherty, D. Kirkland, R. Davies, S. Naylor, and M. Campbell, "Stone Soup." [Online]. Available: https://doi.org/10.5281/zenodo.4663993

[11] "Stone Soup's documentation," 2023. [Online]. Available: https://stonesoup.readthedocs.io/en/latest/

[12] J. Wright and M. Sun, "Stone Soup AKKF Tutorial," 2023. [Online]. Available: https://stonesoup.readthedocs.io/en/latest/auto_tutorials/filters/AKKF.html

[13] D. Schuhmacher, B.-T. Vo, and B.-N. Vo, "A consistent metric for performance evaluation of multi-object filters," *IEEE Transactions on Signal Processing*, vol. 56, no. 8, pp. 3447–3457, 2008.

[14] J. H. Kotecha and P. M. Djuric, "Gaussian particle filtering," *IEEE Trans. Signal Process.*, vol. 51, no. 10, pp. 2592–2601, 2003.