Modular C++ Library for Relaxed Unscented Kalman-Filtering

József Kuti[1] **and Péter Galambos**[1]

[1]Antal Bejczy Centre for Intelligent Robotics, Óbuda University, Budapest, Hungary; {jozsef.kuti, peter.galambos}@irob.uni-obuda.hu

Abstract: Filtering and sensor fusion are critical tasks in advanced engineering applications, especially in robotics and autonomous vehicles. It is a general problem that the high accuracy and low computational cost are mutually exclusive in such filtering algorithms. The Unscented Kalman-Filter (UKF) is a golden mean between the Extended Kalman-Filter (EKF) and the Particle Filter. Recently, the authors have proposed a generic computational relaxation for the EKF that provides options to decrease the computational cost by exploiting the partially linear nature of the mappings in the system model. This paper introduces an open-source C++ RelaxedUnscentedTransformation library that fully implements the proposed method. Since the technique offers several independent usage options, different components are implemented, and the corresponding use cases are illustrated through examples. Via numerical tests, the paper shows that the implementation can significantly decrease computational costs and even provide an opportunity to increase filtering accuracy.

Keywords: Filtering, Kalman filters, Sensor fusion, Unscented transformation, Unscented Kalman-filter, Computational relaxation

1 Introduction

Advanced engineering systems use complex mechatronics, where various sensors are responsible for measuring the changes in the system's state. Sensor fusion from the output of the different sensors and the prediction from the latest estimated state can lead to the optimal estimation of the system's state, on which the control algorithm relies.

Prediction is performed via the state and output update model. The traditional low computational approach (with limited accuracy) was the Extended Kalman-filter (EKF), while later, a much more accurate but computationally more demanding method, Particle Filtering, was introduced. Nowadays, the Unscented Transformation-based (UT) Unscented Kalman-filtering (UKF) offers a trade-off between accuracy and computational cost [1, 2].

The UT applies so-called sigma points around the expected value according to the distribution of the stochastic variable. By applying the nonlinear mapping on these sigma points, the expected value and the distribution can be estimated more accurately than via the local linearization of EKF [2].

In the standard formulation of UT, the n dimensional covariance ellipsoid is estimated by one sigma point at the expected value, and further $2n$ sigma points are selected symmetrically around it. These sigma points are computed via Cholesky-factorization [3] of the covariance matrix. There are other methods where more (e.g., $4n+1$) sigma points are used to approximate the uncertainty better [4]. Another approach decreases the number of sigma points to $(n+1)$ [5, 6] via an optimization step. However, it is computationally more expensive than the Cholesky-factorization, and the Cholesky-factorisation-based symmetric approach remained the most common method. (For a systematic overview, see [7].) Different scaling methods and Taylor-series-based derivations were proposed to improve the method using different weights, and sigma point distances [8–10].

The key idea of UKF was extended with multivariate adaptive methods (see [11–14]) to estimate the necessary parameters of filtering. However, there are robust extensions; see [15–18] to deal with varying parameters, sensor faults, and other phenomena. The method can be combined with the well-known approaches to account for the time delay of the measured signals, see [19].

The methodology is widely applied in distributed filtering [20], consensus filter [21], finite-horizon extended Kalman-filter [22] and the robustness of distributed filters is also analyzed [23]. There are use cases in remote state estimation with stochastic event-triggered sensor schedule [24] to test multi-agent communication schemes [25] and robust, adaptive, and consensus aims are also formulated in the framework, see [26].

Paper [27] of the authors proposed the Relaxed Unscented Transformation as a systematically structured set of methods to decrease the computational cost of UT in cases where the function does not depend on all of the variables in a nonlinear way. In these cases, fewer sigma points can be used, avoiding the unnecessary operations with computation seen in original, linear Kalman-filtering.

This paper aims to introduce a novel C++ library (available at [28]) developed to help the control community to exploit the advantages of Relaxed Unscented Transformation in Kalman-filtering. Because the library provides multiple options that can be fitted to the considered model, several components were defined according to the steps of the method. These components can be individually combined in a way that fits the given filtering problem.

To better illustrate the library's features, some numerical examples are prepared. The examples show that the computational cost can be reduced to almost 50%, and the accuracy of the filtering can also be increased.

2 Notations

$a, b, \ldots$	scalar values
$\mathbf{a}, \mathbf{b}, \ldots$	vectors
$\mathbf{A}, \mathbf{B}, \ldots$	matrices
$\mathbf{0}^{a \times b}$, $\mathbf{I}^{a \times b}$	zero matrix, identity matrix of size $a \times b$
$\mathbf{a}(\mathbf{i})$	reindexed vector with $\mathbf{i}$ index vector, as $\mathbf{a}(\mathbf{i}) = \begin{bmatrix} a_{i_1} & a_{i_2} & \ldots \end{bmatrix}^T$
$\mathbf{A}(\mathbf{i},:)$	matrix with reindexed rows using the $\mathbf{i}$ index vector, as $\mathbf{A}(\mathbf{i},:) = \begin{bmatrix} A_{i_1,1} & A_{i_1,2} & \cdots \\ A_{i_2,1} & A_{i_2,2} & \cdots \\ \vdots & \vdots & \end{bmatrix}$
$\mathbf{A}(:,\mathbf{i})$	matrix with reindexed columns using the $\mathbf{i}$ index vector, as $\mathbf{A}(\mathbf{i},:) = \begin{bmatrix} A_{1,i_1} & A_{1,i_2} & \cdots \\ A_{2,i_1} & A_{2,i_2} & \cdots \\ \vdots & \vdots & \end{bmatrix}$
$\mathbf{A}(\mathbf{i},\mathbf{j})$	reindexed matrix with $\mathbf{i}$, $\mathbf{j}$ index vectors, as $\mathbf{A}(\mathbf{i},\mathbf{j}) = \begin{bmatrix} A_{i_1,j_1} & A_{i_1,j_2} & \cdots \\ A_{i_2,j_1} & A_{i_2,j_2} & \cdots \\ \vdots & \vdots & \end{bmatrix}$
$\sqrt{\mathbf{A}}$	lower triangle Choleski-factorization of a matrix as $\mathbf{A} = \sqrt{\mathbf{A}}\sqrt{\mathbf{A}}^T$
$\hat{\mathbf{x}}$	estimated value of $\mathbf{x}$
$\tilde{\mathbf{x}}$	difference of $\mathbf{x}$ from the expected value
Σ_{xx}	estimated covariance matrix $\Sigma_{xx} = E(\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T)$
Σ_{xy}	estimated cross covariance matrix $\Sigma_{xy} = E(\tilde{\mathbf{x}}\tilde{\mathbf{y}}^T)$

3 Original Unscented Transformation (UT) and the related components

The Unscented Transformation considers a continuous nonlinear mapping in general, as

$$\mathbf{y} = f(\mathbf{x}), \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is an (approximately) Gaussian stochastic variable with covariance matrix Σ_{xx}. The method approximates the expected value of $\mathbf{y}$, its covariance matrix Σ_{yy} and the cross covariance matrix Σ_{xy}.

The σ points are chosen around the expected value of $\mathbf{x}$, such that the expected values and covariance matrices computed from the σ points with appropriate weighting returns exact values for second-order mappings.

In the Cholesky-factorization based method, the $(2m+1)$ sigma points are chosen as

$$\mathscr{X}_0 = \hat{\mathbf{x}}, \quad \mathscr{X}_i = \hat{\mathbf{x}} - \sqrt{\kappa}\delta_i, \quad \mathscr{X}_{i+m} = \hat{\mathbf{x}} + \sqrt{\kappa}\delta_i, \tag{2}$$

where m denotes the rank of Σ_{xx}, δ_i is for the non-zero columns of matrix $\sqrt{\Sigma_{xx}}$ ($i = 1,...,m$), and κ is a parameter that covers various parameters of different approaches, e.g., the so-called scaling parameter that provides an extra degree of freedom to tune the method.

Then by mapping the sigma points as $\mathscr{Y}_i = f(\mathscr{X}_i)$, the expected value and the covariance matrices can be approximated as

$$\begin{aligned}
\hat{\mathbf{y}} &= W_0\mathscr{Y}_0 + W_1\sum_{i=1}^{2m}\mathscr{Y}_i,\\
\Sigma_{yy} &= V_0(\mathscr{Y}_0-\hat{\mathbf{y}})(\mathscr{Y}_0-\hat{\mathbf{y}})^T + V_1\sum_{i=1}^{2m}(\mathscr{Y}_i-\hat{\mathbf{y}})(\mathscr{Y}_i-\hat{\mathbf{y}})^T,\\
\Sigma_{xy} &= V_1\sum_{i=1}^{2m}(\mathscr{X}_i-\hat{\mathbf{x}})(\mathscr{Y}_i-\hat{\mathbf{y}})^T,
\end{aligned}$$

where weights W_0, W_1, V_0, V_1 and distance κ depends on the method to be applied.

The systematic investigations in [29, 30] showed that the accuracy of UT can be highly improved by tuning the scaling parameter in each step. Many studies [30–33] proposed online optimization for this purpose that performs the UT much more times in each sampling step. In these methods, the computational demand of UT is crucial.

From a functional viewpoint, this method has two main parts:

1. The determination of vectors δ_i from the covariance matrix Σ_{xx}, their number will be equal to the rank of the matrix. This method is available in the repository as

   ```
   std::vector<Eigen::VectorXd>
     GenSigmaDifferences(
     const Eigen::MatrixXd& S);
   ```

   where the matrix `S` denotes the covariance matrix Σ_{xx} and the output is the `std::vector` of vectors δ_i ($i = 1,...,m$).

2. The second part is the computation of $\mathscr{X}_i$, then $\mathscr{Y}_i$ sigma points, and from the weighted summations the demanded values. It is available as

   ```
   template<typename Func>
   ValWithCov UTCore(
     const Eigen::VectorXd& x,
     const std::vector<Eigen::VectorXd>&
     xdiffs, Func f,
     const UTSettings& settings);
   ```

   where arbitrary weights and κ can be used (computed according to the value m) through the struct

```
struct UTSettings {
  double kappa, W0, W1, V0, V1;
  UTSettings(double kappa, double W0,
    double W1, double V0, double V1);
  UTSettings(double kappa, double W0,
    double W1); %Vi=Wi
};
```

that allows arbitrary UT parametrization like Scaled UT of [10, 34] but [INESKJ] proved relevance of original parameters for larger models.

The output of the method contains the expected value, the covariance matrix and the cross covariance matrix as

```
struct ValWithCov {
  Eigen::VectorXd y;
  Eigen::MatrixXd Sy, Sxy;
};
```

4 Relaxed Unscented Transformation

The key feature of the method is that it can consider functions as

$$\mathbf{y} = \mathbf{A} \cdot \mathbf{x}(\mathbf{i}_l) + f(\mathbf{x}),$$

where f has a larger dimensional null space and, this way, less sigma point is enough to generate and perform the UT on. Furthermore, the linear part can also depend on a subset of the variables denoted by $\mathbf{i}_l$ indices.

In this case, the computation can be decomposed into three main steps: first, the sigma points must be generated. Two different methods will be presented for this purpose. Then the properties of the output of the nonlinear part will be presented based on the determined sigma points. Then the properties of $\mathbf{y}$ will be computed by exploiting the properties of linear mappings.

4.1 Generation of sigma points

Denote the result of the nonlinear part as $\mathbf{b} = \mathbf{f}(\mathbf{x})$. The first challenge is to generate only as many sigma points as necessary. Paper [27] proposed two methods to obtain them.

4.1.1 Fewer sigma points by considering only a subset of variables

The nonlinear part depends on the variables with indices $\mathbf{i}_{nl}$. In this case, the Cholesky-factorization must be performed only in the given columns.

The result can also be described as performing Cholesky-factorization on the rearranged Σ_x as

$$\Delta\mathbf{A} = \sqrt{\Sigma_x\left(\begin{bmatrix}\mathbf{i}_{nl} & \bar{\mathbf{i}}_{nl}\end{bmatrix}, \begin{bmatrix}\mathbf{i}_{nl} & \bar{\mathbf{i}}_{nl}\end{bmatrix}\right)}, \tag{3}$$

and then $\Delta\mathbf{X}$ is constructed from its first m columns, by rearranging the rows as

$$\Delta\mathbf{X}(\begin{bmatrix}\mathbf{i}_{nl} & \bar{\mathbf{i}}_{nl}\end{bmatrix},:) = \Delta\mathbf{A}(:,\begin{bmatrix}1 & \ldots & m\end{bmatrix}), \tag{4}$$

where index vector $\bar{\mathbf{i}}_{nl}$ is constructed from the complementer set of indices in $\mathbf{i}_{nl}$ and the δ_i $(i = 1,...,m)$ vectors are its columns.

This method is available with function

```
std::vector<Eigen::VectorXd>
  GenSigmaDifferences(
  const Eigen::MatrixXd& S,
  const Eigen::VectorXi& inl);
```

4.1.2 Less sigma points by considering a subspace of the domain

Assume that the nonlinear part depends on the linear combinations of the variables. Describe these values as $(\mathbf{m}_1 \cdot \mathbf{x}(\mathbf{i}_1))$, $(\mathbf{m}_2 \cdot \mathbf{x}(\mathbf{i}_2))$, ..., $(\mathbf{m}_K \cdot \mathbf{x}(\mathbf{i}_K))$, where $\mathbf{m}_k$ is the vector of weights for values with indices $\mathbf{i}_k$ and K denotes the number of combinations.

In this case, the function depends on values of $\mathbf{M} \cdot \mathbf{x}$, where

$$\mathbf{M}(k,\mathbf{i}_k) = \mathbf{m}_k, \quad k = 1,...,K,$$

denote its rank by m.

Example 1. *Considering a mapping, e.g.,*

$$f(\mathbf{x}) = \sin(x_1 + 0.1x_3) - \cos(0.5x_2 + x_3),$$

two linear combinations of the variables can be seen in nonlinear functions. In the first case, the index vector is $\mathbf{i}_1 = \begin{bmatrix}1 & 3\end{bmatrix}$ *and the corresponding weight vector is* $\mathbf{m}_1 = \begin{bmatrix}1 & 0.1\end{bmatrix}$. *In the second linear combination, the index vector is* $\mathbf{i}_2 = \begin{bmatrix}2 & 3\end{bmatrix}$ *and the weight vector is* $\mathbf{m}_2 = \begin{bmatrix}0.5 & 1\end{bmatrix}$. *This way, the matrix* $\mathbf{M}$ *can be written in this case as*

$$\mathbf{M} = \begin{bmatrix}1 & 0 & 0.1\\ 0 & 0.5 & 1\end{bmatrix}.$$

Denote the matrix constructed from an orthonormal basis of the rowspace of $\mathbf{M}$ by $\mathbf{Q}_1 \in \mathbb{R}^{m\times n}$ and the matrix constructed from an orthonormal basis of nullspace of $\mathbf{M}$ by $\mathbf{Q}_2 \in \mathbb{R}^{(n-m)\times n}$. The matrix $\mathbf{Q}$ is constructed of them as

$$\mathbf{Q} = \begin{bmatrix}\mathbf{Q}_1\\ \mathbf{Q}_2\end{bmatrix}, \tag{5}$$

and can be easily computed as RQ factorization of $\mathbf{M}$.

The Choleski-factorization of rearranged Σ_x must be computed as

$$\Delta\mathbf{A} = \sqrt{\mathbf{Q}\Sigma_x\mathbf{Q}^T}, \tag{6}$$

and the $\Delta\mathbf{X}$ is constructed from its first m columns, by rearranging the rows as

$$\Delta\mathbf{X} = \mathbf{Q}^T \Delta\mathbf{A}(:, \begin{bmatrix} 1 & \dots & m \end{bmatrix}) \tag{7}$$

and the vectors δ_i are its columns.

The matrix $\mathbf{Q}$ and value m can be determined via the constructor of the following struct:

```
struct ExactSubspace {
  struct MixedNonlin {
    Eigen::VectorXi i;
    Eigen::VectorXd M;
  };
  typedef std::vector<MixedNonlin>
  MixedNonlinearityList;
  Eigen::SparseMatrix<double> Q;
  int m;
  ExactSubspace(int n, const
    Eigen::VectorXi& inl, const
    MixedNonlinearityList& mix);
};
```

This method must be used only at the initialization of the program.

From this data, the sigma points can be computed via the method:

```
std::vector<Eigen::VectorXd>
  GenSigmaDifferences(const
  Eigen::MatrixXd& S,
  const ExactSubspace& sp);
```

4.2 Computing Sigma points and approximations from them

The computation of the output of this nonlinear mapping $\mathbf{b} = f(\mathbf{x})$ is the same as before: the `UTCore(...)` function can be used to do it, but in this case it results in $\hat{\mathbf{b}}$, Σ_{bb}, Σ_{xb} values.

4.3 Merging linear and nonlinear results

It considers the problem of

$$\mathbf{y} = \mathbf{A} \cdot \mathbf{x}(\mathbf{i}_l) + \mathbf{b} \tag{8}$$

where $\hat{\mathbf{x}}$, Σ_{xx}, $\hat{\mathbf{b}}$, Σ_{bb}, Σ_{xb} are given from the previous computations. In order to solve it, the library implements optimized version of computation:

$$\begin{aligned} \hat{\mathbf{y}} &= \mathbf{A} \cdot \hat{\mathbf{x}}(\mathbf{i}_l) + \hat{\mathbf{b}}, \\ \Sigma_y &= \Sigma_b + \mathbf{A}\Sigma_x(\mathbf{i}_l, \mathbf{i}_l)\mathbf{A}^T + Sym(\mathbf{A}\Sigma_{xb}(\mathbf{i}_l, :)), \\ \Sigma_{xy} &= \Sigma_{xb} + \Sigma_x(:, \mathbf{i}_l)\mathbf{A}^T. \end{aligned} \tag{9}$$

That is available as:

```
ValWithCov MixedLinSources(
  const ValWithCov& x,
  const ValWithCov& b,
  const Eigen::VectorXi& il,
  const Eigen::MatrixXd& A);
```

4.4 Reduced size output of the nonlinear function

The previous components can be enough to perform the Relaxed Unscented Transformation with smaller computational cost than the original UT. The following component can decrease the computational cost further, if the values in the nonlinear function are not linearly independent.

The first one is that the function can be defined as

$$\mathbf{b} = \begin{bmatrix} \mathbf{b}_0 \\ \mathbf{F}\cdot\mathbf{b}_0 \end{bmatrix}, \quad \mathbf{b}_0 = f_0(\mathbf{x}). \tag{10}$$

In this case, the **b** related quantities can be computed as

$$\begin{aligned} \hat{\mathbf{b}} &= \begin{bmatrix} \hat{\mathbf{b}}_0 \\ \mathbf{F}\hat{\mathbf{b}}_0 \end{bmatrix}, \qquad \Sigma_{xb} = \begin{bmatrix} \Sigma_{xb_0} & \Sigma_{xb_0}\mathbf{F}^T \end{bmatrix}, \\ \Sigma_b &= \begin{bmatrix} \Sigma_{b_0} & \Sigma_{b_0}\mathbf{F}^T \\ \mathbf{F}\Sigma_{b_0} & \mathbf{F}\Sigma_{b_0}\mathbf{F}^T \end{bmatrix}. \end{aligned} \tag{11}$$

that is available in the component

```
ValWithCov LinearMappingOnb(
  const ValWithCov& b0,
  const Eigen::MatrixXd& F);
```

A similar case, where

$$\mathbf{b} = \begin{bmatrix} 0 \\ \mathbf{b}_0 \\ \mathbf{F}\cdot\mathbf{b}_0 \end{bmatrix}, \quad \mathbf{b}_0 = f_0(\mathbf{x}), \tag{12}$$

is also implemented and is available as

```
ValWithCov LinearMappingOnbWith0(
  const ValWithCov& b0,
  const Eigen::MatrixXd& F);
```

If there are multiple entries of this expression, or only the order must be changed reordering can be applied as

$$\mathbf{b} = \mathbf{b}_0(\mathbf{g}), \tag{13}$$

then

$$\hat{\mathbf{b}} = \hat{\mathbf{b}}_0(\mathbf{g}),\ \ \Sigma_b = \Sigma_{b_0b_0}(\mathbf{g},\mathbf{g}),\ \ \Sigma_{xb} = \Sigma_{xb_0}(:,\mathbf{g}). \tag{14}$$

where $\mathbf{g} \in \mathbb{N}^b$ is an index vector and $f_0 : \mathbb{R}^n \to \mathbb{R}^b$.

It is implemented as

```
ValWithCov Reordering(const ValWithCov&
  b0, const Eigen::VectorXi& g);
```

5 Multiscaled Unscented Transformation

In Multiscaled Unscented Transformation instead of $(2m+1)$ sigma points, $(2m \cdot N_{max}+1)$ sigma points are used, defined as

$$\mathscr{X}_0 = \hat{\mathbf{x}}, \quad \mathscr{X}_i^{(N)} = \hat{\mathbf{x}} - \sqrt{\kappa_N}\delta_i, \quad \mathscr{X}_{i+m}^{(N)} = \hat{\mathbf{x}} + \sqrt{\kappa_N}\delta_i, \tag{15}$$

where δ_i denotes the non-zero columns of matrix $\sqrt{\Sigma_x}$, m their numbers, $i = 1,...,m$, and κ_N is the parameter for $N = 1,...,N_{max}$ circle of sigma points.

Then by mapping the sigma points as $\mathscr{Y}_i^{(N)} = f(\mathscr{X}_i^{(N)})$, the expected value and the covariance matrices can be approximated as

$$\begin{aligned}
&\hat{\mathbf{y}} = W_0\mathscr{Y}_0 + \sum_{N=1}^{N_{max}} W_N \sum_{i=1}^{2n} \mathscr{Y}_i^{(N)}, \\
&\Sigma_{yy} = V_0(\mathscr{Y}_0 - \hat{\mathbf{y}})(\mathscr{Y}_0 - \hat{\mathbf{y}})^T + \\
&\quad + \sum_{N=1}^{N_{max}} V_N \sum_{i=1}^{2n} (\mathscr{Y}_i^{(N)} - \hat{\mathbf{y}})(\mathscr{Y}_i^{(N)} - \hat{\mathbf{y}})^T, \\
&\Sigma_{xy} = \sum_{N=1}^{N_{max}} V_N \sum_{i=1}^{2n} (\mathscr{X}_i^{(N)} - \hat{\mathbf{x}})(\mathscr{Y}_i^{(N)} - \hat{\mathbf{y}})^T,
\end{aligned}$$

where weights W_N, V_N and distances κ_N depend on the method to be applied.

This computation is available as

```
template<typename Func>
ValWithCov MultiScaledUTCore(const
  Eigen::VectorXd& x, const
  std::vector<Eigen::VectorXd>& xdiffs,
  Func f,
  const MultiScaledUTSettings& settings);
```

where the necessary constants are provided in the struct

```
struct MultiScaledUTSettings {
  std::vector<double> kappa, W, V;
  double W0, V0;
};
```

6 Kalman-filtering

For the sake of convenience, the well-known equations of the Kalman-filtering

$$\hat{\mathbf{x}} = \overline{\mathbf{x}} - \mathbf{K}(\overline{\mathbf{y}} - \mathbf{y}_{meas}),$$
$$\hat{\Sigma}_{xx} = \overline{\Sigma}_{xx} - \mathbf{K}\overline{\Sigma}_{yx},$$

where $\mathbf{K} = \overline{\Sigma}_{xy}\overline{\Sigma}_{yy}^{-1}(1-\varepsilon)$, are also available as

```
ValWithCov KalmanFilter(const ValWithCov&
  x, const ValWithCov& y, const
  Eigen::VectorXd& ymeas,
  double eps = 1e-5);
```

where ε is a small number to avoid negative eigen values, but also highly depends on the parity of weight V_0.

7 UKF from the given components

7.1 UT of a function

7.1.1 Original UT

If there is a nonlinear function, a struct can be built around it like

```
struct StateUpdate {
  static VectorXd f(const VectorXd& x) {
    VectorXd out = VectorXd::Zero(11);
    out(0) = ....;
    ...
    return out;
  }
  ValWithCov UT(const ValWithCov& x) {
    auto xdiffs =
      GenSigmaDifferences(x.Sy);
    int m = xdiffs.size();
    return UTCore(x.y, xdiffs, f,
      UTSettings(m, 0, 0.5 / double(m)));
  }
};
```

7.1.2 Relaxed UT

In this case, the necessary values can be initialized in the constructor and only used in the UT subroutine, as

```
struct StateUpdate {
  Eigen::MatrixXd A, F;
  Eigen::VectorXi il, inl;

  StateUpdate() {
    // Deriving matrix A
    A = Eigen::MatrixXd::Zero(11, 6);
    for (int i = 0; i < 6; i++)
    A(i, i) = 1;
    A(6, 2) = Ts;
    // il, inl
    il = Eigen::VectorXi(6);
    for (int n = 0; n < 6; n++)
      il(n) = n;
    inl = Eigen::VectorXi(5);
    for (int n = 0; n < 5; n++)
      inl(n) = n + 6;
    F = Eigen::MatrixXd::Zero(0, 11);
  }
  static VectorXd f(const VectorXd& x) {
    VectorXd out = VectorXd::Zero(11);
    out(0) = ....;
    ⋮
    return out;
  }
  ValWithCov UT(const ValWithCov& x) {
    auto xdiffs = GenSigmaDifferences(
      x.Sy, inl);
    auto b0 = UTCore(x.y, xdiffs, f,
      UTSettings(m, 0, 0.5 / double(m)));
    auto b = LinearMappingOnb(b0, F);
    return MixedLinSources(x, b, il, A);
  }
};
```

where $\kappa = m$, $W_0 = V_0 = 0$, $W_1 = V_1 = 1/2m$ weights were applied.

(Similarly, struct `ExactSubspace` can be also initialized and used in function `GenSigmaDifferences(...)` or if reordering is needed, the function `StateUpdate::UT` must return with

```
return MixedLinSources(x,
  Reordering(b, g), il, A);
```

where vector g is initialized in the constructor.)

7.2 Implementation of Unscented Kalman-Filter

If the standard UT is applied, the system model is traditionally defined as

$$\mathbf{x}_k = f_k(\mathbf{x}_{k-1}) + \mathbf{w}_k, \tag{16}$$

$$\mathbf{y}_k = g_k(\mathbf{x}_k) + \mathbf{v}_k, \tag{17}$$

or it can also depend on the disturbance and noise signals in a nonlinear way and the $\mathbf{x}_0$ value is considered an initial value. In this case, the output of the nonlinear function can be computed via UT, and the covariance must be increased with Σ_{ww} and Σ_{vv} (and the values with $\hat{\mathbf{w}}$ and $\hat{\mathbf{v}}$ if they are not zero).

To exploit the benefits of the Relaxed UT, one or both of them can be written in a partially linear form as

$$\mathbf{x}_k = \mathbf{A}_k \mathbf{x}_{k-1,l} + f_k(\mathbf{x}_{k-1}) + \mathbf{w}_k, \tag{18}$$

or in a more relaxed form

$$\mathbf{x}_k = \mathbf{A}_k \mathbf{x}_{k-1,l} + \mathbf{b}(\mathbf{g}) + \mathbf{w}_k, \tag{19}$$

$$\mathbf{b} = \begin{bmatrix} \mathbf{b}_0 \\ \mathbf{F}\mathbf{b}_0 \end{bmatrix}, \qquad \mathbf{b}_0 = f_k(\mathbf{x}_{k-1}). \tag{20}$$

If the function depends on the **w** disturbance or **v** noise in a more sophisticated way, the function can be considered on the variable

$$\mathbf{z} = \begin{bmatrix} \mathbf{x} \\ \mathbf{w} \end{bmatrix}$$

and the same formalisms can be used.

In all cases, a simple struct can wrap the specifics of the UT for either the state update or the output update. (Via so-called lambda functions, the varying sampling time T_s or other parameters can also be injected into the functions.)

Then the simple Kalman-filter can be implemented as

```
(Inputs: ValWithCov xold;
  Eigen::VectorXd ymeas, w, v;
  Eigen::MatrixXd Sw, Sv;
  StateUpdate stateUpdate;
  OutputUpdate outputUpdate;)
auto xnew = stateUpdate.UT(xold);
xnew.Sy += Sw;
xnew.y += w;
auto ynew = outputUpdate.UT(xnew);
ynew.Sy += Sv;
ynew.y += v;
auto xnewfiltered = KalmanFilter(
  xnew, ynew, ymeas);
```

It is easily extendable to achieve an adaptive UT methods see [30] for an example.

7.3 Implemented compact methods

The library provides compact methods for the original UT or some types of the relaxed UT as

```
template <typename Func>
ValWithCov FullUT(Func f,
  const ValWithCov& x);
```

```
template <typename Func>
ValWithCov RelaxedUT(
  const Eigen::MatrixXd& A,
  const Eigen::VectorXi& il, Func f,
  const Eigen::MatrixXd& F,
  const Eigen::VectorXi& g,
  const Eigen::VectorXi& inl,
  const ValWithCov& x);
```

However, they cover only a small subset of the functionality to give some examples as reference, and a large number of arguments should also be wrapped for the sake of compactness. For these reasons, the composition from the given components is recommended.

8 Comparison of functions and methods

8.1 Compared approaches

In this section, the following function

$$f(\mathbf{x}) = \begin{bmatrix} \sin(x_1 + 4x_2 - 0.5x_3) \\ \cos(x_1 + 4x_2 - 0.5x_3) \\ x_4 + x_5 \\ x_4 + x_6 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}.$$

Here the following implementations will be compared:

1. Original UT on the nonlinear function using $2 \cdot 6 + 1$ sigma points.
2. Relaxed UT considering the function as

   $$f(\mathbf{x}) = \mathbf{A} \cdot \mathbf{x}(\mathbf{i}_l) + \begin{bmatrix} \sin(x_1 + 4x_2 - 0.5x_3) \\ \cos(x_1 + 4x_2 - 0.5x_3) \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

   where $\mathbf{i}_l = [4, 5, 6]$ and

   $$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

   applying

   (a) using $2 \cdot 3 + 1$ sigma points,

   (b) using $2 \cdot 1 + 1$ sigma points applying exact subspace.

	1	2/a	2/b	3/a	3/b
Comp. time$[\mu s]$	9.436	7.512	5.075	7.772	6.027

Table 1
Comparison of average computational cost of different relaxed approaches with the original UT. The Relaxed UT (especially 2/b) highly overwhelm the Original UT (1).

3. Relaxed UT considering the function as

$$f(\mathbf{x}) = \mathbf{A}\cdot\mathbf{x}(\mathbf{i}_l) + \mathbf{b}([2,3,1,1,1,1,1]),$$

where

$$\mathbf{b} = \begin{bmatrix} 0 \\ f_0(\mathbf{x}) \end{bmatrix}, \quad f_0(\mathbf{x}) = \begin{bmatrix} \sin(x_1 + 4x_2 - 0.5x_3) \\ x_1 \cos(x_1 + 4x_2 - 0.5x_3) \end{bmatrix}.$$

applying

(a) using $2\cdot 3+1$ sigma points,

(b) using $2\cdot 1+1$ sigma points applying exact subspace.

8.2 Comparison of computational cost

First, the computational costs of the different implementations are compared. All of them were called 10^7 times on an i7-9750H (2.6GHz x64) processor. The measured average computational times are discussed in Table 1.

The first column in Table 1 refers to the original UT. It can be seen that reducing the sigma points from $2\cdot 6+1$ to $2\cdot 3+1$ in approach 2a does reduce the computational cost to 80% and by reducing it to $2\cdot 1+1$ in approach 2b reduces further it to 53%.

The following columns show that the reduced output size did not decrease the computational time because it is not crucial if there are only a few sigma points, as in these cases.

8.3 Comparison of accuracy

To compare the accuracy of the relaxed UT approach with the original UT method considering random inputs (number of 10^4 with given $Tr(\Sigma_{xx})$) with values resulting from Monte-Carlo simulations of number 10^5 samples.

The results of the method highly depends on the applied κ W_i and V_i values. In order to ensure positive definiteness of matrices

$$\Sigma_{yy}, \quad (\Sigma_{xx} - \Sigma_{xy}(\Sigma_{yy} + \Sigma_{vv})^{-1}\Sigma_{yx})$$

where $\Sigma_{vv} \to \mathbf{0}$ is assumed as a worst case scenario, all of the W_i weights must be positive. By applying

$$W_1 = V_1 = \frac{1}{2\kappa}, \quad W_0 = V_0 = 1 - \frac{m}{\kappa},$$

	$Tr(\Sigma_{xx}) = 0.1$		$Tr(\Sigma_{xx}) = 1$	
	$\varepsilon(\mathbf{y})$	$\varepsilon(\Sigma_{yy})$	$\varepsilon(\mathbf{y})$	$\varepsilon(\Sigma_{yy})$
$1(\kappa = 6)$	5.53e-5	3.26e-3	1.19e-1	0.283
$2/\text{a}(\kappa = 3)$	2.27e-5	8.48e-4	2.03e-2	0.153
$2/\text{b}(\kappa = 1)$	8.28e-5	4.55e-3	1.60e-1	0.472
$2/\text{b}(\kappa = 2)$	2.12e-5	6.51e-4	8.73e-3	0.197
$2/\text{b}(\kappa = 3)$	3.79e-6	1.55e-4	2.30e-2	0.336
$3/\text{a}(\kappa = 3)$	2.27e-5	8.48e-4	2.03e-2	0.153
$3/\text{b}(\kappa = 1)$	8.28e-5	4.55e-3	1.60e-1	0.472
$3/\text{b}(\kappa = 2)$	2.12e-5	6.51e-4	8.73e-3	0.197
$3/\text{b}(\kappa = 3)$	3.79e-6	1.55e-4	2.30e-2	0.336

Table 2
Comparison of accuracy of original UT with relaxed UT approaches with different weights and under different $Tr(\Sigma_{xx})$ circumstances. The Relaxed UT (especially 2/b $\kappa = 2$) highly overwhelm the Original UT (1 $\kappa = 6$).

where $2m+1$ is the number of sigma points, it can be seen that $\kappa \geq m$ must be used.

From this condition, and Taylor-series-based derivations that prefer $\kappa = 2,3$ values, the following situations will be considered in the next comparison:

- (1) $\kappa = m = 6$,
- (2/a) and (3/a) $\kappa = m = 3$,
- (2/b) and (3/b) where $m = 1$: $\kappa = 1,2,3$.

To characterize the results, the following values were computed

$$\varepsilon(\mathbf{y}) = mean(||E(\mathbf{y}) - \hat{\mathbf{y}}||),$$
$$\varepsilon(\Sigma_{yy}) = mean(||E(\Sigma_{yy}) - \hat{\Sigma}_{yy}||),$$

where $E(\mathbf{y})$ and $E(\Sigma_{yy})$ are computed from the Monte-Carlo simulations and mean is computed on the 10^4 random data.

The results can be seen in Table 2 for $Tr(\Sigma_{xx}) = 0.1$ and $Tr(\Sigma_{xx}) = 1$ cases. It can be seen that the relaxed UT approaches overwhelm the original UT by allowing a larger variety of κ values, and this increase in accuracy is significant.

Conclusion

The paper introduced the concepts and functionality of the open-source C++ library for Relaxed Unscented Transformation. It has shown the implemented components, their mathematical role, and practical usage. Finally, a numerical example has shown that the appropriate setup in the considered case reduced the computational cost to 53% and increased the filtering accuracy.

Acknowledgments

József Kuti and Péter Galambos thankfully acknowledge the financial support of this work by the project no. 2019-1.3.1-KK-2019-00007 implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2019-1.3.1-KK funding scheme. Péter Galambos is a Bolyai Fellow of the Hungarian Academy of Sciences. Péter Galambos is supported by the UNKP-23-5 (Bolyai+) New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development, and Innovation Fund.

References

[1] S. J. Julier, J. K. Uhlmann, and H. F. Durrant-Whyte. A new approach for filtering nonlinear systems. In *Proceedings of 1995 American Control Conference-ACC'95*, volume 3, pages 1628–1632. IEEE, 1995.

[2] S. J. Julier and J. K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[3] N. J. Higham. Cholesky factorization. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(2):251–254, 2009.

[4] R. Radhakrishnan, A. Yadav, P. Date, and S. Bhaumik. A new method for generating sigma points and weights for nonlinear filtering. *IEEE Control Systems Letters*, 2(3):519–524, 2018.

[5] S. J. Julier. The spherical simplex unscented transformation. In *Proceedings of the 2003 American Control Conference, 2003.*, volume 3, pages 2430–2434. IEEE, 2003.

[6] S. J. Julier and J. K. Uhlmann. Reduced sigma point filters for the propagation of means and covariances through nonlinear transformations. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, volume 2, pages 887–892. IEEE, 2002.

[7] H. M. T. Menegaz, J. Y. Ishihara, G. A. Borges, and A. N. Vargas. A Systematization of the Unscented Kalman Filter Theory. *IEEE Transactions on Automatic Control*, 60(10):2583–2598, 2015.

[8] E. A. Wan, R. Van Der Merwe, and S. Haykin. The unscented kalman filter. *Kalman filtering and neural networks*, 5(2007):221–280, 2001.

[9] L. Wang and Y. Zhao. Scaled unscented transformation of nonlinear error propagation: accuracy, sensitivity, and applications. *Journal of surveying engineering*, 144(1):04017022, 2018.

[10] S. J. Julier. The scaled unscented transformation. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, volume 6, pages 4555–4559. IEEE, 2002.

[11] W. Khalaf, I. Chouaib, and M. Wainakh. Novel adaptive UKF for tightly-coupled INS/GPS integration with experimental validation on an UAV. *Gyroscopy and Navigation*, 8(4):259–269, 2017.

[12] M. Bozorg, M. S. Bahraini, and A. B. Rad. New adaptive UKF algorithm to improve the accuracy of SLAM. *International Journal of Robotics, Theory and Applications*, 5(1):35–46, 2019.

[13] S. Gao, G. Hu, and Y. Zhong. Windowing and random weighting-based adaptive unscented Kalman filter. *International Journal of Adaptive Control and Signal Processing*, 29(2):201–223, 2015.

[14] G.-g. Hu, S.-s. Gao, and Y. Zhao. Novel adaptive UKF and its application in integrated navigation [j]. *Journal of Chinese Inertial Technology*, 22(3):357–361, 2014.

[15] S. Ishihara and M. Yamakita. Gain constrained robust UKF for nonlinear systems with parameter uncertainties. In *2016 European Control Conference (ECC)*, pages 1709–1714. IEEE, 2016.

[16] B. Gao, S. Gao, Y. Zhong, G. Hu, and C. Gu. Interacting multiple model estimation-based adaptive robust unscented Kalman filter. *International Journal of Control, Automation and Systems*, 15(5):2013–2025, 2017.

[17] J. Zhao, M. Netto, and L. Mili. A robust iterated extended Kalman filter for power system dynamic state estimation. *IEEE Transactions on Power Systems*, 32(4):3205–3216, 2016.

[18] W. Li, S. Sun, Y. Jia, and J. Du. Robust unscented Kalman filter with adaptation of process and measurement noise covariances. *Digital Signal Processing*, 48:93–103, 2016.

[19] A. Hermoso-Carazo and J. Linares-Pérez. Unscented filtering from delayed observations with correlated noises. *Mathematical Problems in Engineering*, 2009.

[20] Y. Zhang, B. Chen, L. Yu, and D. W. C. Ho. Distributed kalman filtering for interconnected dynamic systems. *IEEE Transactions on Cybernetics*, pages 1–10, 2021.

[21] B. Lian, Y. Wan, Y. Zhang, M. Liu, F. L. Lewis, and T. Chai. Distributed kalman consensus filter for estimation with moving targets. *IEEE Transactions on Cybernetics*, pages 1–13, 2020.

[22] P. Duan, Z. Duan, Y. Lv, and G. Chen. Distributed finite-horizon extended kalman filtering for uncertain nonlinear systems. *IEEE Transactions on Cybernetics*, 51(2):512–520, 2021.

[23] B. Lian, F. L. Lewis, G. A. Hewer, K. Estabridis, and T. Chai. Robustness analysis of distributed kalman filter for estimation in sensor networks. *IEEE Transactions on Cybernetics*, pages 1–12, 2021.

[24] L. Li, D. Yu, Y. Xia, and H. Yang. Remote nonlinear state estimation with stochastic event-triggered sensor schedule. *IEEE Transactions on Cybernetics*, 49(3):734–745, March 2019.

[25] M. C. Fowler, T. C. Clancy, and R. K. Williams. Intelligent knowledge distribution: Constrained-action pomdps for resource-aware multiagent communication. *IEEE Transactions on Cybernetics*, 2020.

[26] W. Li, G. Wei, F. Han, and Y. Liu. Weighted average consensus-based unscented kalman filtering. *IEEE Transactions on Cybernetics*, 46(2):558–567, 2016.

[27] J. Kuti, I. J. Rudas, H. Gao, and P. Galambos. Computationally Relaxed Unscented Kalman Filter. *IEEE Transactions on Cybernetics*, 2022. accepted.

[28] J. Kuti. C++ library for relaxed unscented transformation. `https://github.com/ABC-iRobotics/RelaxedUnscentedTransformation.git`, 2021.

[29] O. Straka, J. Dunik, and M. Simandl. Scaling parameter in unscented transform: Analysis and specification. In *2012 American Control Conference (ACC)*, pages 5550–5555. IEEE, 2012.

[30] O. Straka, J. Duník, and M. Šimandl. Unscented Kalman filter with advanced adaptation of scaling parameter. *Automatica*, 50(10):2657–2664, 2014.

[31] J. Dunik, M. Simandl, and O. Straka. Unscented Kalman filter: aspects and adaptive setting of scaling parameter. *IEEE Transactions on Automatic Control*, 57(9):2411–2416, 2012.

[32] L. A. Scardua and J. J. da Cruz. Adaptively tuning the scaling parameter of the unscented Kalman filter. In *CONTROLO'2014–Proceedings of the 11th Portuguese Conference on Automatic Control*, pages 429–438. Springer, 2015.

[33] M. S. Bahraini, M. Bozorg, and A. B. Rad. A new adaptive UKF algorithm to improve the accuracy of slam. *Int J Robot*, 2018.

[34] E. A. Wan and R. Van Der Merwe. The unscented Kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*, pages 153–158. Ieee, 2000.