Computational Relaxation for Unscented Transformation based Kalman-Filters

May 18, 2021

1 Introduction

The Unscented Transformation (UT) was proposed to approximate the distribution of results of nonlinear mappings performed on statistic variables that is a crucial problem in Kalman-filters [1, 2]. The former linearization-based Extended Kalman-filter (EKF) assumes that the expected value is exactly mapped and takes into consideration only its local environment via linearization. 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 of the resulted stochastic variable can be estimated in a more accurate way than via local linearization of EKF.

The Kalman-filter based on UT (Unscented Kalman-filter, UKF) is between the EKF and the particle filter (PF) even in accuracy and computational cost [3].

The main difference of UKF (and EKF) compared to particle filter is the fact, that the former ones approximates the distribution with a Gaussian covariance ellipsoid described by the covariance matrix. In the original formulation of UT, the n dimensional ellipsoid is estimated by one sigma point as the expected value and $2n$ sigma points symmetrically around it. There are methods where more (e.g. $4n+1$) sigma points are used to achieve a more robust approximation of the uncertainty [4]. These sigma points are computed via Cholesky-factorization [5] of the covariance matrix.

But the principal way of decreasing the computational cost would be to omit the unnecessary operations. The first attempt to this – related to the UT – was proposed by [6]. It has showed, that a system with linear state update and nonlinear output update does not need two UT, but the classical uncertainty propagation of Kalman-filters can be used during the state update and sigma points based UT for the a'priori estimation and uncertainty of the output.

2 Notations

$a, b, \dots$	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 with size $a \times b$
$\mathbf{a}(\mathbf{i})$	reindexed vector with $\mathbf{i}$ index vector, as $\mathbf{a}(\mathbf{i}) = \begin{bmatrix} a_{i_1} & a_{i_2} & \ldots \end{bmatrix}^T$

3 Traditional Unscented Transformation

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

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

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

The header UT.h contains a function for this purpose with the following syntax:

```
template<typename Func>
void UT(const Eigen::VectorXd& x, const Eigen::MatrixXd& Sx,
        Func fin, Eigen::VectorXd& z, Eigen::MatrixXd& Sz,
        Eigen::MatrixXd& Sxz);
```

where Func must be a const Eigen::VectorXd&→Eigen::VectorXd function.

4 Relaxed Unscented Transformation

The relaxed UT considers the mapping in a partially linear form, like

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

where function f only depends on a subset of x_i inputs. This subset can be described as a list of indices $\mathbf{i}_{nl}$.

In some cases, the nonlinear function depends on a linear combination of a lot of variables. In these cases, a much better way can be used:

- by listing the standalone nonlinear dependencies into $\mathbf{i}_{nl}$
- and by giving the indices of the linear combination into $\mathbf{i}_1$ and the corresponding weights into $\mathbf{m}_1$,

via genQ function $\mathbf{Q}$ and $\mathbf{Q}_1$ matrices can be generated, and the implementation will consider the exact subspace of nonlinear dependencies.

To decrease the size of the nonlinear function, it is described in the following way:

$$\mathbf{z} = \mathbf{A}\mathbf{x}(\mathbf{i}_l) + \begin{bmatrix} \mathbf{I} \\ \mathbf{F} \end{bmatrix} f(\mathbf{x}),$$

where **F** can have rows number of zero, but the number of columns must be always the same as the size of $f(x)$.

Because sometimes the last elements cannot be given as a combination with **F** of the former ones so called reindexing can be used as

$$\mathbf{z} = \mathbf{A}\mathbf{x}(\mathbf{i}_l) + \mathbf{b}(\mathbf{g}),$$
$$\mathbf{b} = \begin{bmatrix} \mathbf{I} \\ \mathbf{F} \end{bmatrix} f(\mathbf{x})$$

where **g** is a vector of indices.

According to the parametrization of the set of nonlinear dependencies, and other options to decrease the computational demand, four implementations are given for the relaxed UT in header RelaxedUT.h.

4.1 Type I

In this case, reindexing and matrix **Q** is NOT applied, so the syntax is:

```
template <typename Func>
void RelaxedUT(const Eigen::MatrixXd& A, const Eigen::VectorXi& il,
     Func fin, const Eigen::MatrixXd& F, const Eigen::VectorXi& inl,
     const Eigen::VectorXd& x0, const Eigen::MatrixXd& S0,
     Eigen::VectorXd& y, Eigen::MatrixXd& Sy, Eigen::MatrixXd& Sxy);
```

where Func must be a const Eigen::VectorXd&→Eigen::VectorXd function.

4.2 Type II

In this case, reindexing is used but matrix **Q** is not, so the syntax is:

```
template <typename Func>
void RelaxedUT(const Eigen::MatrixXd& A, const Eigen::VectorXi& il,
     Func fin, const Eigen::MatrixXd& F, const Eigen::VectorXi& g,
     const Eigen::VectorXi& inl, const Eigen::VectorXd& x0,
     const Eigen::MatrixXd& S0, Eigen::VectorXd& y,
     Eigen::MatrixXd& Sy, Eigen::MatrixXd& Sxy);
```

where Func must be a const Eigen::VectorXd&→Eigen::VectorXd function.

4.3 Type III

In this case, **Q** is used but reindexing is not, so the syntax is:

```
template <typename Func>
void RelaxedUT(const Eigen::MatrixXd& A, const Eigen::VectorXi& il,
     Func fin, const Eigen::MatrixXd& F,
     const Eigen::SparseMatrix<double>& Q, const Eigen::SparseMatrix<double>& Q1,
     const Eigen::VectorXd& x0, const Eigen::MatrixXd& S0,
     Eigen::VectorXd& y, Eigen::MatrixXd& Sy, Eigen::MatrixXd& Sxy);
```

where Func must be a const Eigen::VectorXd&→Eigen::VectorXd function.

4.4 Type IV

In this case, reindexing and matrix **Q** both are used:

```
template <typename Func>
void RelaxedUT(const Eigen::MatrixXd& A, const Eigen::VectorXi& il,
Func fin, const Eigen::MatrixXd& F, const Eigen::VectorXi& g,
const Eigen::SparseMatrix<double>& Q, const Eigen::SparseMatrix<double>& Q1,
const Eigen::VectorXd& x0, const Eigen::MatrixXd& S0,
Eigen::VectorXd& y, Eigen::MatrixXd& Sy, Eigen::MatrixXd& Sxy);
```

where Func must be a const Eigen::VectorXd&$\rightarrow$Eigen::VectorXd function.

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*, vol. 3. IEEE, 1995, pp. 1628–1632.

[2] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004.

[3] S. Konatowski, P. Kaniewski, and J. Matuszewski, "Comparison of estimation accuracy of EKF, UKF and PF filters," *Annual of Navigation*, vol. 23, no. 1, pp. 69–87, 2016.

[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*, vol. 2, no. 3, pp. 519–524, 2018.

[5] N. J. Higham, "Cholesky factorization," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 1, no. 2, pp. 251–254, 2009.

[6] G. Hu, S. Gao, and Y. Zhong, "A derivative UKF for tightly coupled INS/GPS integrated navigation," *ISA transactions*, vol. 56, pp. 135–144, 2015.