UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI INGEGNERIA INFORMATICA, MODELLISTICA, ELETTRONICA E SISTEMISTICA

Robotics and Automation Engineering

Analysis and Application of UKF (Unscented Kalman Filter)

Prof.:

Giuseppe Fedele
Luigi D'Alfonso

Student:

Giuseppe Coppola
Mat. 263624

ACADEMIC YEAR 2024/25

Contents

Chapter 1

Introduction

The **Unscented Kalman Filter (UKF)**, like the **Extended Kalman Filter (EKF)**, is used for estimating the states of *nonlinear systems*. However, unlike the EKF, which relies on a first-order linearization of the nonlinear model, the EKF approach is often *sub-optimal* and can easily lead to *divergence* in the estimation.
The UKF outperforms the EKF in terms of both *prediction accuracy* and *estimation error*. In the EKF, the state distribution is propagated analytically through a linear approximation, which may distort the posterior **mean** and **covariance**.
The UKF, on the other hand, is a *derivative-free* alternative that overcomes this limitation by using a *deterministic sampling approach*. The state distribution is approximated using a minimal set of carefully chosen sample points, known as **sigma points**. These sigma points are selected to accurately capture the **mean** and **covariance** of the state without requiring linearization of the model.
To propagate these statistical characteristics (mean and covariance) through a nonlinear transformation, the UKF uses the **Unscented Transformation (UT)**.

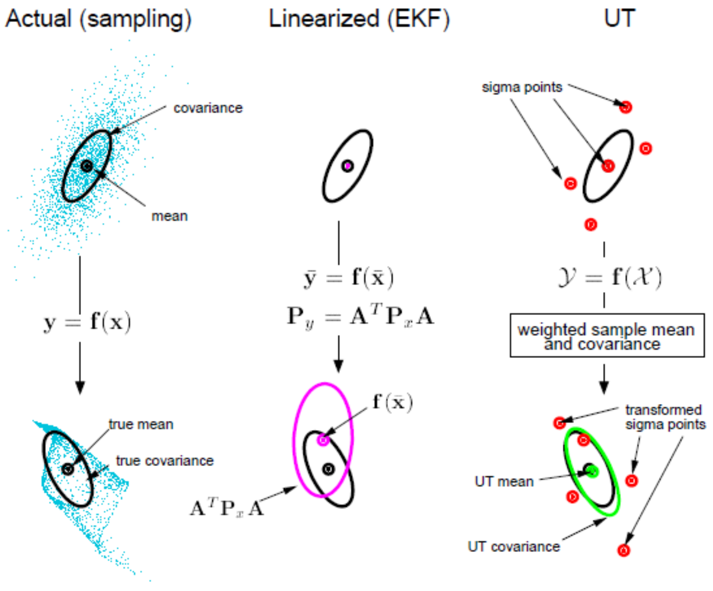
Actual (sampling)
Linearized (EKF)
UT
covariance
mean
sigma points
$\mathbf{y} = \mathbf{f}(\mathbf{x})$
$\bar{\mathbf{y}} = \mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{P}_y = \mathbf{A}^T\mathbf{P}_x\mathbf{A}$
$\mathcal{Y} = \mathbf{f}(\mathcal{X})$
weighted sample mean and covariance
true mean
true covariance
$\mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{A}^T\mathbf{P}_x\mathbf{A}$
transformed sigma points
UT mean
UT covariance

Figure 1.1: Example of the UT for mean and covariance propagation

Like the EKF, the UKF consists of two main steps: *model prediction* and *data assimilation*. However, in the UKF, these steps are *preceded* by an initial stage that involves the generation of sigma points, which is central to the Unscented approach.

Chapter 2

UKF Algorithm

Considering the system:

$$\begin{cases} x_k = f(x_{k-1}) + w_{k-1} \\ z_k = h(x_k) + v_k \end{cases} \tag{2.1}$$

where

- $x_k \in \mathbb{R}^n$
- **Process Noise**: $w_k \sim \mathcal{N}(0, Q)$
- **Measurement Error**: $v_k \sim \mathcal{N}(0, V)$

The *initial state* x_0 is a random vector with known mean $\mu_0 = E[x_0]$ and covariance $P_0 = E[(x_0 - \mu_0)(x_0 - \mu_0)^T]$.
For this type of algorithm there is a different management regarding the *propagation of noise*. We can introduce the UKF **augmented** and **non-augmented**.

2.1 Unscented Transformation

The **Unscented Transformation** (UT) is a method for calculating the statistics of a random variable which undergoes a nonlinear transformation.
If we consider one-step nonlinear transformation with additive noise:

$$y = f(x) + w \tag{2.2}$$

where x is an $n \times 1$ random vector with mean $\hat{x}$ and covariance P_x and w is an $m \times 1$ zero-mean noise vector with covariance Q.

Assumption 1: w and x are *uncorrelated.*
Assumption 2: Either x or w is not restricted to be Gaussian.

We can write (2.2) as:

$$y^a = f^a(x^a) \tag{2.3}$$

where it is the augmented representation; the augmented random vector is $x^a = \begin{bmatrix} x^T & w^T \end{bmatrix}^T$ and the new nonlinear transformation is defined as:

$$f^a(x^a) = f^a(\begin{bmatrix} x^T & w^T \end{bmatrix}^T) = f(x) + w$$

In both cases, the objective is to calculate the mean and the covariance of the transformations. Respectively, in the non-augmented case is $\hat{y}$ and P_y and, in the augmented case is $\hat{y^a}$ and P_{y^a}.
In general, let's analyze both cases.

2.1.1 Non-Augmented

In this case, Process Noise and Measurement Error are not included in the state. They are added after the propagation of the sigma points, as additive terms.
The random vector x is approximated by $2n+1$ symmetric sigma points and so, to calculate the statistics of y, we form a matrix χ according to the follow:

$$\begin{aligned} \chi_0 &= \hat{x} \\ \chi_i &= \hat{x} + \left(\sqrt{(n+\kappa)P_x}\right)_i \\ \chi_{i+n} &= \hat{x} - \left(\sqrt{(n+\kappa)P_x}\right)_i \\ i &= 1, ..., n \end{aligned} \tag{2.4}$$

where $(\sqrt{P})_i$ is the i^{th} column of the matrix square root of P.
We can define the weight associated to each χ:

$$\begin{aligned} W_0 &= \kappa/(n+\kappa) \\ W_i &= 1/2(n+\kappa) \\ W_{i+n} &= 1/2(n+\kappa) \\ i &= 1, ..., n \end{aligned} \tag{2.5}$$

The scaling parameter κ is a **scaling parameter** which is usually set to 0 or $3-n$.

- $\kappa = 0$: the sigma points and their weights will be related to n (dimension of x).
- $\kappa = 3-n$: it is selected so that the fourth-order moment information is mostly captured in the true Gaussian case

In general other choices of κ would lead to better or worse results depending on specific characteristics of the integrand.

Then, we can define the transformation:

$$\gamma_i = f(\chi_i) \tag{2.6}$$

The **mean** $\hat{y}$ is given by the weighted average of the trasformed points

$$\hat{y} = \sum_{i=0}^{2n} W_i \gamma_i \tag{2.7}$$

The **covariance** P_y is the weighted outer product of the transformed points plus the noise covariance

$$P_y = \sum_{i=0}^{2n} W_i(\gamma_i - \hat{y})(\gamma_i - \hat{y})^T + Q \tag{2.8}$$

2.1.2 Augmented

In this case, the state is extended to explicitly include noise.
The augmented random vector x^a is approximated by $2(n+m)+1$ symmetric sigma points

$$\begin{aligned} \chi_0^a &= \hat{x}^a \\ \chi_i^a &= \hat{x}^a + \left(\sqrt{(n+m+\kappa^a)P_{x^a}}\right)_i \\ \chi_{i+n+m} &= \hat{x}^a - \left(\sqrt{(n+m+\kappa^a)P_{x^a}}\right)_i \\ i &= 1, ..., n+m \end{aligned} \tag{2.9}$$

and their weights

$$\begin{aligned} W_0^a &= \kappa^a/(n+m+\kappa^a) \\ W_i^a &= 1/2(n+m+\kappa^a) \\ W_{i+n+m}^a &= 1/2(n+m+\kappa^a) \\ i &= 1, ..., n+m \end{aligned} \tag{2.10}$$

Note that $\hat{x}^a = \begin{bmatrix}\hat{x}^T & 0_{1\times m}\end{bmatrix}^T$ and

$$\begin{aligned} \left(\sqrt{P_{x^a}}\right)_i &= \left(\sqrt{\begin{bmatrix} P_x & 0_{n\times m} \\ 0_{m\times n} & Q \end{bmatrix}}\right)_i = \\ &= \begin{cases} \begin{bmatrix} \left(\sqrt{P_x}\right)_i \\ 0_{m\times 1} \end{bmatrix}, & i = 1, ..., n \\ \begin{bmatrix} 0_{m\times 1} \\ \left(\sqrt{Q}\right)_{i-n} \end{bmatrix}, & i = n, ..., n+m \end{cases} \end{aligned} \tag{2.11}$$

Substituting (2.11) in (2.9):

$$\begin{aligned} \chi_0^a &= \begin{bmatrix} \hat{x} \\ 0_{m\times 1} \end{bmatrix} \\ \chi_i^a &= \begin{bmatrix} \hat{x} + \left(\sqrt{(n+m+\kappa^a)P_x}\right)_i \\ 0_{m\times 1} \end{bmatrix} \\ \chi_{i+n}^a &= \begin{bmatrix} \hat{x} - \left(\sqrt{(n+m+\kappa^a)P_x}\right)_i \\ 0_{m\times 1} \end{bmatrix} \\ \chi_{j+2n}^a &= \begin{bmatrix} \hat{x} \\ \left(\sqrt{(n+m+\kappa^a)\,Q}\right)_j \end{bmatrix} \\ \chi_{j+2n+m}^a &= \begin{bmatrix} \hat{x} \\ -\left(\sqrt{(n+m+\kappa^a)\,Q}\right)_j \end{bmatrix} \\ i &= 1, ..., n \qquad j = 1, ..., m \end{aligned} \tag{2.12}$$

and their weights

$$\begin{aligned} W_0^a &= \kappa^a/(n+m+\kappa^a) \\ W_i^a &= 1/2(n+m+\kappa^a) \\ W_{i+n}^a &= 1/2(n+m+\kappa^a) \\ W_{j+2n}^a &= 1/2(n+m+\kappa^a) \\ W_{j+2n+m}^a &= 1/2(n+m+\kappa^a) \\ i = 1, ..., n &\qquad j = 1, ..., m \end{aligned} \tag{2.13}$$

Then, we can define the transformation:

$$\gamma_i^a = f^a(\chi_i^a) \tag{2.14}$$

The **mean** $\hat{y}$ is given by the weighted average of the trasformed points

$$\hat{y}^a = \sum_{i=0}^{2(n+m)} W_i^a \gamma_i^a \tag{2.15}$$

Substituting (2.12), (2.13) and (2.14) into (2.15)

$$\begin{aligned} \hat{y}^a = W_0^a f^a(\chi_0^a) + \sum_{i=1}^{n} W_i^a \left[f^a(\chi_i^a) + f^a(\chi_{i+n}^a) \right] + \sum_{i=1}^{m} W_i^a \left[f^a(\chi_{i+2n}^a) + f^a(\chi_{i+2n+m}^a) \right] = \\ = W_0^a f(\hat{x}) + W_i^a \sum_{i=1}^{n} \left[f\left(\hat{x} + \left(\sqrt{(n+m+\kappa^a)P_x} \right)_i \right) + f\left(\hat{x} - \left(\sqrt{(n+m+\kappa^a)P_x} \right)_i \right) \right] \\ +2mW_i^a f(\hat{x}) = \\ = \frac{m+\kappa^a}{n+m+\kappa^a} f(\hat{x}) \\ + \frac{1}{2\left(n+m+\kappa^a\right)} \sum_{i=1}^{n} \left[f\left(\hat{x} + \left(\sqrt{(n+m+\kappa^a)P_x} \right)_i \right) + f\left(\hat{x} - \left(\sqrt{(n+m+\kappa^a)P_x} \right)_i \right) \right] \end{aligned}$$

As compared with $\hat{y}$:

$$\sum_{i=0}^{2n} W_i \gamma_i = \frac{\kappa}{n+\kappa} f(\hat{x}) + \frac{1}{2(n+\kappa)} \sum_{i=1}^{n} \left[f\left(\hat{x} + \left(\sqrt{(n+\kappa)\, P_x} \right)_i \right) + f\left(\hat{x} - \left(\sqrt{(n+\kappa)\, P_x} \right)_i \right) \right]$$

Then, $\hat{y} = \hat{y}^a$ is satisfied if and only if

$$n + \kappa = n + m + \kappa^a \triangleq C \tag{2.16}$$

So, the sums $n+\kappa$ and $n+m+\kappa^a$ are identical and independent of the state dimension.

The **covariance** P_y^a is the weighted outer product of the transformed points

$$\begin{aligned} P_y^a = \sum_{i=0}^{2(n+m)} W_i^a \left(\gamma_i^a - \hat{y}^a\right) \left(\gamma_i^a - \hat{y}^a\right)^T = \\ = \sum_{i=0}^{2n} W_i^a \left(\gamma_i^a - \hat{y}^a\right) \left(\gamma_i^a - \hat{y}^a\right)^T + \sum_{i=2n+1}^{2(n+m)} W_i^a \left(\gamma_i^a - \hat{y}^a\right) \left(\gamma_i^a - \hat{y}^a\right)^T \end{aligned} \tag{2.17}$$

About the first term:

$$\begin{aligned}\sum_{i=2n+1}^{2(n+m)} W_i^a \left(\gamma_i^a - \hat{y}^a\right)\left(\gamma_i^a - \hat{y}^a\right)^T = \\ = \frac{C-m-n}{C}\left(f(\hat{x}) - \hat{y}\right)\left(f(\hat{x}) - \hat{y}\right)^T + \sum_{i=1}^{2n} W_i \left(f(\chi_i) - \hat{y}\right)\left(f(\chi_i) - \hat{y}\right)^T\end{aligned} \tag{2.18}$$

About the second term:

$$\begin{aligned}\sum_{i=2n+1}^{2(n+m)} W_i^a \left(\gamma_i^a - \hat{y}^a\right)\left(\gamma_i^a - \hat{y}^a\right)^T = \\ \frac{1}{2C}\sum_{i=1}^{m}\left\{\left[f(\hat{x}) - \hat{y} + \left(\sqrt{CQ}\right)_i\right]\left[f(\hat{x}) - \hat{y} + \left(\sqrt{CQ}\right)_i\right]^T + \right. \\ \left[f(\hat{x}) - \hat{y} - \left(\sqrt{CQ}\right)_i\right]\left[f(\hat{x}) - \hat{y} - \left(\sqrt{CQ}\right)_i\right]^T = \\ = \frac{1}{2C}\sum_{i=0}^{2n} 2\left\{\left[f(\hat{x}) - \hat{y}\right]\left[f(\hat{x}) - \hat{y}\right]^T + C\left.\sqrt{Q}\right)_i \left.\sqrt{Q}\right)_i^T\right\} = \\ = \frac{m}{C}\left(f(\hat{x}) - \hat{y}\right)\left(f(\hat{x}) - \hat{y}\right)^T + Q\end{aligned} \tag{2.19}$$

Therefore

$$P_y^a = \sum_{i=0}^{2n}\left(f(\chi_i) - \hat{y}\right)\left(f(\chi_i) - \hat{y}\right)^T + Q \tag{2.20}$$

and we have that

$$P_y^a \equiv P_y \tag{2.21}$$

In summary, the UT is configurable with the free parameters. If (2.16) is not satisfied the augmented and non-augmented version are different.
Then, the common version of the UT selects $C = 3$ so that the non-augmented and the augmented coincide.

2.2 Efficient Square Root Implementation

The most computationally expensive operation in UKF is to calculating the new set of sigma points; this because we use the square-root matrix of the state covariance $P \in \mathbb{R}^{L\times L}$ where $P = SS^T$.
An efficient implementation using Cholesky requires $O(L^3/6)$ computations.
There is a version of UKF that propagate directly the matrix S which is called the **Square-Root UKF**. In general, with this approach, the algorithm still $O(L^3)$ but with improoved numerical properties.
This algorithm uses three powerful linear algebra techniques:

- **QR Decomposition**: the QR decomposition of a matrix $A \in \mathbb{R}^{N\times L}$, with $N \geq L$,is given by $A^T = QR$ where $Q \in \mathbb{R}^{N\times N}$ is orthogonal and $R \in \mathbb{R}^{N\times L}$ is upper triangular.

  The upper triangular part of R, $\bar{R}$, is the transpose of the Cholesky factor $P = AA^T$ where $\bar{R} = S^T$ such that $\bar{R}^T\bar{R} = AA^T$.

  The computational complexity of QR decomposition is $O(NL^2)$. If we apply the Cholesky factorization directly on $P = AA^T$ the computational complexity is $O(L^3/6)$ plus $O(NL^2)$ to form AA^T.

- **Cholesky Factor Updating**: if S is the original Cholesky factor of P, then the Cholesky factor of the rank-1 update $P \pm \sqrt{\nu} uu^T$ is denoted as S = cholupdate$\{S$,u,$\pm\nu\}$.

  If u is a matrix, the result is M consecutive columns of u.

  This algorithm is only $O(L^2)$ per update.

- **Efficient Least Squares**: the solution of the equation $(AA^T)x = A^T b$ also corresponds to the solution of the overtermined least squares problem $Ax = b$.

Therefore, this makes everything more numerically stable and more efficient (no explicit inverses, less risk of non-positively defined matrices).

2.3 Algorithms

Algorithm 1 Unscented Kalman Filter (Non-Augmented Case)

```
*Initialize with:*
$\hat{x}_0 = E[x_0] = \mu_0$
$P_0 = E[(x_0 - \mu_0)(x_0 - \mu_0)^T]$

**for** $k = 1$ to $\infty$ **do**
    *Calculate **Sigma Points**:*
    $\chi_{k-1} = \left[\hat{x}_{k-1} \quad \hat{x}_{k-1} + \sqrt{(n+\lambda)P_{k-1}} \quad \hat{x}_{k-1} - \sqrt{(n+\lambda)P_{k-1}}\right]$

    ***Time Update:***
    $\chi_{k|k-1} = f(\chi_{k-1}, u_{k-1})$
    $\hat{x}_{\bar{k}} = \sum_{i=0}^{2n} W_i^{(m)} \chi_{i,k|k-1}$
    $P_{\bar{k}} = \sum_{i=0}^{2n)} = W_i^{(c)} \left(\chi_{i,k|k-1} - \hat{x}_k\right) \left(\chi_{i,k|k-1} - \hat{x}_k\right)^T + Q$
    $\gamma_{k|k-1} = h(\chi_{k|k-1})$
    $\hat{y}_{\bar{k}} = \sum_{i=0}^{2n} W_i^{(m)} \gamma_{i,k|k-1}$

    ***Measurement update equation:***
    $P_{y_{\bar{k}}\, y_{\bar{k}}} = \sum_{i=0}^{2n} W_i^{(c)} \left(\gamma_{i,k|k-1} - \hat{y}_{\bar{k}}\right) \left(\gamma_{i,k|k-1} - \hat{y}_{\bar{k}}\right)^T + V$
    $P_{x_k\, y_k} = \sum_{i=0}^{2n} W_i^{(c)} \left(\chi_{i,k|k-1} - \hat{x}_{\bar{k}}\right) \left(\gamma_{i,k|k-1} - \hat{y}_{\bar{k}}\right)^T$
    $K_k = P_{x_k\, y_k} P_{y_{\bar{k}}\, y_{\bar{k}}}^{-1}$
    $\hat{x}_k = \hat{x}_{\bar{k}} + K_k(y_k - \hat{y}_{\bar{k}})$
    $P_k = P_{\bar{k}} - K_k\, P_{y_{\bar{k}}\, y_{\bar{k}}}\, K_k^T$
**end for**
```

Algorithm 2 Unscented Kalman Filter (Augmented Case)

```
  *Initialize with:*
  $\hat{x}_0 = E[x_0] = \mu_0$
  $P_0 = E[(x_0 - \mu_0)(x_0 - \mu_0)^T]$
  $\hat{x}_0^a = [\hat{x}_0^T \;\; 0 \;\; 0]^T$
  $P_0^a = E[(x_0^a - \hat{x}_0^a)(x_0^a - \hat{x}_0^a)^T] = \begin{bmatrix} P_0 & 0 & 0 \\ 0 & Q & 0 \\ 0 & 0 & V \end{bmatrix}$
  **for** $k = 1$ to $\infty$ **do**
      *Calculate **Sigma Points**:*
      $\chi_{k-1}^a = \left[\hat{x}_{k-1}^a \quad \hat{x}_{k-1}^a + \sqrt{(n+m+\lambda)P_{k-1}^a} \quad \hat{x}_{k-1}^a - \sqrt{(n+m+\lambda)P_{k-1}^a}\right]$

      Define:
      $\chi^a = \left[(\chi^x)^T \;\; (\chi^w)^T \;\; (\chi^v)^T\right]$

      ***Time Update**:*
      $\chi_{k|k-1}^x = f(\chi_{k-1}^x, u_{k-1}, \chi_{k-1}^w)$
      $\hat{x}_{\bar{k}} = \sum_{i=0}^{2(n+m)} W_i^{(m)} \chi_{i,k|k-1}^x$
      $P_{\bar{k}} = \sum_{i=0}^{2(n+m)} = W_i^{(c)} \left(\chi_{i,k|k-1}^x - \hat{x}_k\right) \left(\chi_{i,k|k-1}^x - \hat{x}_k\right)^T$
      $\gamma_{k|k-1} = h(\chi_{k|k-1}^x, \chi_{k-1}^w)$
      $\hat{y}_{\bar{k}} = \sum_{i=0}^{2(n+m)} W_i^{(m)} \gamma_{i,k|k-1}$

      ***Measurement update equation**:*
      $P_{y_{\bar{k}}\, y_{\bar{k}}} = \sum_{i=0}^{2(n+m)} W_i^{(c)} \left(\gamma_{i,k|k-1} - \hat{y}_{\bar{k}}\right) \left(\gamma_{i,k|k-1} - \hat{y}_{\bar{k}}\right)^T$
      $P_{x_k\, y_k} = \sum_{i=0}^{2(n+m)} W_i^{(c)} \left(\chi_{i,k|k-1} - \hat{x}_{\bar{k}}\right) \left(\gamma_{i,k|k-1} - \hat{y}_{\bar{k}}\right)^T$
      $K_k = P_{x_k\, y_k} P_{y_{\bar{k}}\, y_{\bar{k}}}^{-1}$
      $\hat{x}_k = \hat{x}_{\bar{k}} + K_k(y_k - \hat{y}_{\bar{k}})$
      $P_k = P_{\bar{k}} - K_k P_{y_{\bar{k}}\, y_{\bar{k}}} K_k^T$
  **end for**
```

Algorithm 3 Square-Root Unscented Kalman Filter

```
Initialize with:
$\hat{x}_0 = E[x_0] = \mu_0$
$S_0 = chol\Big\{ E[(x_0 - \mu_0)(x_0 - \mu_0)^T] \Big\}$

for $k = 1$ to $\infty$ do
    Calculate Sigma Points:
    $\chi_{k-1} = \Big[ \hat{x}_{k-1} \quad \hat{x}_{k-1} + \sqrt{(n+\lambda)} S_k \quad \hat{x}_{k-1} - \sqrt{(n+\lambda)} S_k \Big]$

    Time Update:
    $\chi_{k|k-1} = f(\chi_{k-1}, u_{k-1})$
    $\hat{x}_{\bar{k}} = \sum_{i=0}^{2n} W_i^{(m)} \chi_{i,k|k-1}$
    $S_{\bar{k}} = qr\Big\{ \Big[ \sqrt{W_1^{(c)}} \left( \chi_{1:2n,k|k-1} - \hat{x}_{\bar{k}} \right) \quad \sqrt{Q} \Big] \Big\}$
    $S_{\bar{k}} = cholupdate\Big\{ S_{\bar{k}} \;,\; \chi_{0,k} - \hat{x}_{\bar{k}} \;,\; W_0^{(c)} \Big\}$
    $\gamma_{k|k-1} = h(\chi_{k|k-1})$
    $\hat{y}_{\bar{k}} = \sum_{i=0}^{2n} W_i^{(m)} \gamma_{i,k|k-1}$

    Measurement update equation:
    $S_{\bar{y}_k} = qr\Big\{ \Big[ \sqrt{W_1^{(c)}} (\gamma_{1:2n,k} - \hat{y}_k) \quad \sqrt{V} \Big] \Big\}$
    $S_{\bar{y}_k} = cholupdate\Big\{ S_{\bar{y}_k} \;,\; \gamma_{0,k} - \hat{y}_k \;,\; W_0^{(c)} \Big\}$
    $P_{x_k y_k} = \sum_{i=0}^{2n} W_i^{(c)} \left( \chi_{i,k|k-1} - \hat{x}_{\bar{k}} \right) \left( \gamma_{i,k|k-1} - \hat{y}_{\bar{k}} \right)^T$
    $K_k = \left( P_{x_k y_k} / S_{\bar{y}_k}^T \right) / S_{\bar{y}_k}$
    $\hat{x}_k = \hat{x}_{\bar{k}} + K_k (y_k - \hat{y}_{\bar{k}})$
    $U = K_k S_{\bar{y}_k}$
    $S_k = cholupdate\{ S_{\bar{k}} \;,\; U \;,\; -1 \}$
end for
```

In this cases we have that:

1. The **scaling parameter** is λ and it is given by:

   $$\lambda = \alpha^2 (L + \kappa) - L \tag{2.22}$$

   where α is a *constant* that determine the spread of sigma points around $\hat{x}$ and it's usually set to

   $$1e - 3 \leq \alpha \leq 1$$

   and L is the dimension of the state.

   The constant κ is a **secondary scaled pameter** (see [2.5] for details).

There is another constant β that is used to incorporate prior knowledge of the distribution of x.

2. The **weights** W_i are given by:

$$\begin{aligned} W_0^{(m)} &= \lambda/(L+\lambda) \\ W_0^{(c)} &= \lambda/(L+\lambda) + (1-\alpha^2+\beta) \\ W_i^{(m)} = W_i^{(c)} &= 1/[2(L+\lambda)] \quad , \quad i = 1, ..., 2L \end{aligned} \tag{2.23}$$

where $W_i^{(m)}$ is the *mean weight* and $W_i^{(m)}$ is the *covariance weight.*
We use the weights calculated using the parameters α, β e κ to have more precision and numerical stability.
Usually κ is set to 0 and β to 2.

Chapter 3

Simulations

This section is dedicated to simulations among the 3 types of algorithms.

3.1 Model (2D Motion)

Consider the following model:

$$\begin{bmatrix} x_{k+1}^{(1)} \\ x_{k+1}^{(2)} \\ x_{k+1}^{(3)} \\ x_{k+1}^{(4)} \end{bmatrix} = \begin{bmatrix} x_k^{(1)} + x_k^{(3)} \cdot \Delta t + 0.1 \cdot \sin(x_k^{(2)}) \\ x_k^{(2)} + x_k^{(4)} \cdot \Delta t \cdot \cos(x_k^{(1)}) \\ x_k^{(3)} \\ x_k^{(4)} \end{bmatrix} \tag{3.1}$$

which describes the 2-dimensional motion of an object in a plane.
The state is described by 4 components which they represent respectively **x-position**, **y-position**, **x-velocity** and **y-velocity**.
The dynamic of the system is described by the non-linear function $f(x)$, which expresses the evolution of the state over time, with a discrete step Δt.

$$f(x) = \begin{bmatrix} x_1 + x_3 \cdot \Delta t + 0.1 \cdot \sin(x_2) \\ x_2 + x_k 4 \cdot \Delta t \cdot \cos(x_1) \\ x_3 \\ x_4 \end{bmatrix} \tag{3.2}$$

This dynamic is affected by a **process Gaussian noise** with zero mean and covariance Q.
The observational model assumes that only the positions of the object are measured:

$$h(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{3.3}$$

And, they are affected by a **measurement Gaussian noise** with zero mean and covariance V.
The simulations are performed in $N = 150$ step by choosing a $\Delta t = 1$;
I want to simulate 3 situations:

- **Accurate sensors** (low measurement noise) and **well-modeled and low uncertainty dynamics** (low process noise):

$$Q = \begin{bmatrix} 0.01 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0.005 & 0 \\ 0 & 0 & 0 & 0.005 \end{bmatrix} \qquad V = \begin{bmatrix} 0.05 & 0 \\ 0 & 0.05 \end{bmatrix} \tag{3.4}$$

  Due to "low" Q, both positions and velocities are expected to evolve fairly predictably and with little perturbation.

  Due to "low" V, the sensors that measure position are precise, with little random deviation in observations.

- **Not accurate sensors**: the measurements are noisy. This scenario is typical of situations where sensors are cheap, noisy, or located in bad environments

$$Q = \begin{bmatrix} 0.01 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0.005 & 0 \\ 0 & 0 & 0 & 0.005 \end{bmatrix} \qquad V = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \tag{3.5}$$

  Due to "low" Q, as before, both positions and velocities are expected to evolve fairly predictably.

  Due to "high" V, observations (measured positions) are much noisier, meaning the sensors are less reliable and introduce significant uncertainty into the measurements.

- **Stress Test** (loud noises):

$$Q = \begin{bmatrix} 0.2 & 0 & 0 & 0 \\ 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix} \qquad V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.6}$$

  Due to "high" Q, the dynamic model is highly uncertain, meaning reality can deviate significantly from the prediction.

  Due to "high" V, as before, the observations are extremely noisy and unreliable.

  This scenario is only used in test scopes.

3.1.1 Functions

In this section will be explained the Matlab functions that describe the algorithms.

Standard UKF (Non-Augmented)

```
function [x_est, P_est] = UKF(f,h,y,x_prev,P_prev,Q,V,alpha,beta,kappa)
    n = length(x_prev);
    m = length(y);

    lambda = alpha^2*(n+kappa)-n;

    W_m = [lambda/(n+lambda), 1/(2*(n+lambda))*ones(1,2*n)];
    W_c = W_m;
    W_c(1) = lambda/(n+lambda) + (1-alpha^2+beta);

    S = sqrtm(P_prev);
    X = [x_prev, x_prev+sqrt(n+lambda)*S, x_prev-sqrt(n+lambda)*S];

    X_k = zeros(n,2*n+1);
    for k = 1:2*n+1
        X_k(:,k) = f(X(:,k));
    end

    x_k = X_k*W_m';
    P_k = Q;
    for k = 1:2*n+1
        P_k = P_k + W_c(k)*(X_k(:,k)-x_k)*(X_k(:,k)-x_k)';
    end

    Y_k = zeros(m,2*n+1);
    for k = 1:2*n+1
        Y_k(:,k) = h(X_k(:,k));
    end

    y_k = Y_k*W_m';
    P_yy = V;
    for k = 1:2*n+1
        P_yy = P_yy + W_c(k)*(Y_k(:,k)-y_k)*(Y_k(:,k)-y_k)';
    end

    P_xy = zeros(n,m);
    for k = 1:2*n+1
        P_xy = P_xy + W_c(k)*(X_k(:,k)-x_k)*(Y_k(:,k)-y_k)';
    end

    K = P_xy / P_yy;

    x_est = x_k + K*(y-y_k);
    P_est = P_k - K*P_yy*K';
end
```

As described in **Algorithm 1**, the function is divided into various phases:

- **Calculation of Parameters** λ, $W^{(m)}$ and $W^{(c)}$.
- **Generation of Sigma Points** by calculating the Cholesky covariance matrix
- **Prediction of the State** propagating the sigma points through the function f. Then the computation of mean and covariance.
- **Prediction of the Observation**, we act as in the case of the prediction of the state
- The computation of the cross covariance and of the Kallman Gain.

UKF Augmented

```
function [x_est, P_est] = UKF_augmented(f,h,y,x_prev,P_prev,Q,V,alpha,beta,kappa)
    n = length(x_prev);
    nw = size(Q,1);
    nv = size(V,1);
    L = n+nw+nv;

    x_a = [x_prev; zeros(nw,1); zeros(nv,1)];
    P_a = blkdiag(P_prev,Q,V);

    lambda = alpha^2*(L+kappa)-L;

    W_m = [lambda/(L+lambda), 1/(2*(L+lambda))*ones(1,2*L)];
    W_c = W_m;
    W_c(1) = lambda/(L+lambda) + (1-alpha^2+beta);

    % Sigma Points
    S = sqrtm(P_a);
    X = zeros(L,2*L+1);
    X(:,1) = x_a;

    for k = 1:L
        X(:,k+1) = x_a + sqrt(L+lambda)*S(:,k);
        X(:,k+L+1) = x_a - sqrt(L+lambda)*S(:,k);
    end

    % -- State Prediction --
    X_k = zeros(n,2*L+1);

    for k = 1:2*L+1
        w_k = X(n+1:n+nw,k);
        X_k(:,k) = f(X(1:n,k))+w_k;
    end

    % Mean (State)
    x_k = X_k*W_m';

    % Covariance (State)
    P_k = zeros(n);

    for k = 1:2*L+1
        P_k = P_k + W_c(k)*(X_k(:,k)-x_k)*(X_k(:,k)-x_k)';
    end

    % -- Measurement Prediction --
    Y_k = zeros(nv,2*L+1);

    for k = 1:2*L+1
        v_k = X(n+nw+1:end,k);
        Y_k(:,k) = h(X_k(:,k)) + v_k;
    end

    % Mean (Output)
    y_k = Y_k*W_m';

    % Covariance (Output)
    P_yy = zeros(nv);
    for k = 1:2*L+1
        P_yy = P_yy + W_c(k)*(Y_k(:,k)-y_k)*(Y_k(:,k)-y_k)';
    end

    % Cross Covariance
    P_xy = zeros(n,nv);
    for k = 1:2*L+1
        P_xy = P_xy + W_c(k)*(X_k(:,k)-x_k)*(Y_k(:,k)-y_k)';
    end

    K = P_xy / P_yy;

    x_est = x_k + K*(y-y_k);
    P_est = P_k - K*P_yy*K';
end
```

As described in **Algorithm 2**, the function is divided into various phases which are identical to those described for the UKF standard.
The only difference is in the calculation of the augmented state at the biginning of the function.

Square-Root UKF

```matlab
function [x_est,S_est] = UKF_squareroot(f,h,y,x_prev,S_prev,Q,V,alpha,beta,kappa)
    n = length(x_prev);
    m = length(y);

    lambda = alpha^2*(n+kappa)-n;

    W_m = [lambda/(n+lambda), 1/(2*(n+lambda))*ones(1,2*n)];
    W_c = W_m;
    W_c(1) = lambda/(n+lambda) + (1-alpha^2+beta);

    % Sigma Points
    X = [x_prev, x_prev+sqrt(n+lambda)*S_prev, x_prev-sqrt(n+lambda)*S_prev];

    % -- State Prediction --
    X_k = zeros(n,2*n+1);

    for k = 1:2*n+1
        X_k(:,k) = f(X(:,k));
    end

    % Mean (State)
    x_k = X_k*W_m';

    % qr update for predicted covariance square root
    x_dev =  X_k - x_k;
    [~,S_k] = qr([sqrt(W_c(2:end)).*x_dev(:,2:end), sqrtm(Q)']',0);
    U = x_dev(:,1)*sqrt(W_c(1));

    if W_c(1) < 0
        S_pred = cholupdate(S_k,U,'-');
    else
        S_pred = cholupdate(S_k,U,'+');
    end

    % -- Measurement Prediction --
    Y_k = zeros(m,2*n+1);

    for k = 1:2*n+1
        Y_k(:,k) = h(X_k(:,k));
    end

    % Mean (Output)
    y_k = Y_k*W_m';

    % qr update for measurement covariance square root
    y_dev = Y_k - y_k;
    [~,S_y] = qr([sqrt(W_c(2:end)).*y_dev(:,2:end), sqrtm(V)']',0);
    U_y = y_dev(:,1)*sqrt(W_c(1));

    if W_c(1) < 0
        Sy_pred = cholupdate(S_y,U_y,'-');
    else
        Sy_pred = cholupdate(S_y,U_y,'+');
    end

    % Cross Covariance
    P_xy = zeros(n,m);
    for k = 1:2*n+1
        P_xy = P_xy + W_c(k)*(X_k(:,k)-x_k)*(Y_k(:,k)-y_k)';
    end

    K = (P_xy / Sy_pred') / Sy_pred;

    x_est = x_k + K*(y-y_k);
    U_k = K*Sy_pred;
    S_est = S_pred;

    for i = 1:size(U_k,2)
        S_est = cholupdate(S_est, U_k(:,i),'-');
    end
end
```

As described in **Algorithm 3**, the function is divided into various phases which are identical to those described for the UKF standard and UKF augmented.
In this case, the algorithm works with the square-root matrix of P, S.
Instead of $P = \sum_i W_i^{(c)}(\chi_i - \hat{x})(\chi_i - \hat{x})^T$, it is used **QR decomposition** ($qr\{.....\}$) to compute an upper triangular matrix R equivalent to the square-root of P avoiding the computation of matrix-matrix products.
After qr function, S is updated with the matlab function **cholupdate** to efficiently updates a Cholesky factorization and to include the remaining part of the covariance. The update is perfoming in this way:

$$if \ \left(W_1^{(c)} < 0\right) \quad \rightarrow \quad S = cholupdate(S, U,' -')$$

$$else \qquad\qquad\qquad \rightarrow \quad S = cholupdate(S, U,' +')$$

It is related to the correct handling of the square root of the covariance update when you add the contribution of the first sigma point.
Since we want to maintain the semidefinite positivity of the covariance matrix, this check is performed on the weight of the first sigma point. This because we want to mantain numeric stability and mathematical correctness in the algoritm.
So, if we add a term UU^T to S the variance increases, otherwise the variance decreases.

3.2 Simulation scenarios

In this section, the various simulations will be presented comparing the noise management between Non-Augmented UKF and Augmented UKF.

3.2.1 Accurate Sensors

We are in the case where we have good sensors available. In this **Figure 3.1**, the

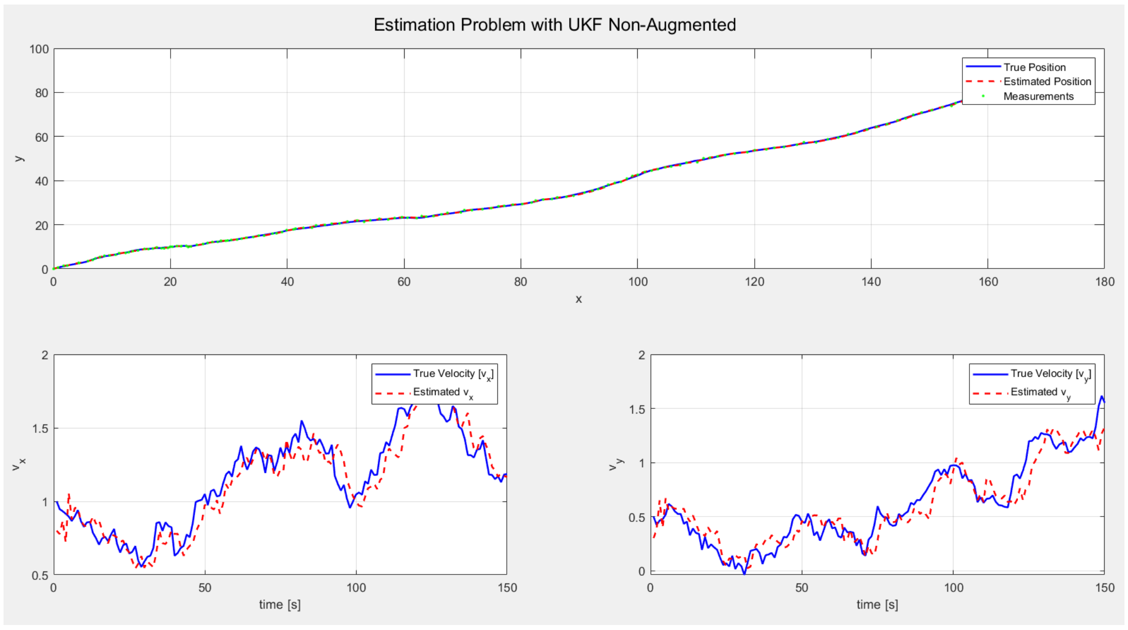

Figure 3.1: Estimation problem with UKF Non-Augmented

estimated trajectory aligns well with the ground truth, indicating accurate position tracking. However, velocity estimates show small oscillations, especially at points of curvature.
This is expected, as the UKF Non-Augmented does not explicitly model noise in the sigma points, making it slightly less stable in velocity estimation.

In the **Figure 3.2**, which describes the estimation using the Augmented UKF, the position estimation is nearly identical to the true trajectory.
The velocity estimates are visibly smoother than those from the UKF Non-Augmented. By incorporating process noise directly into the state vector, the augmented UKF enhances the estimation of dynamic states. This leads to better consistency and reduced sensitivity to small variations in the measurements.
Furthermore, estimates are smoother and less noisy and they are more consistent and stable over time.

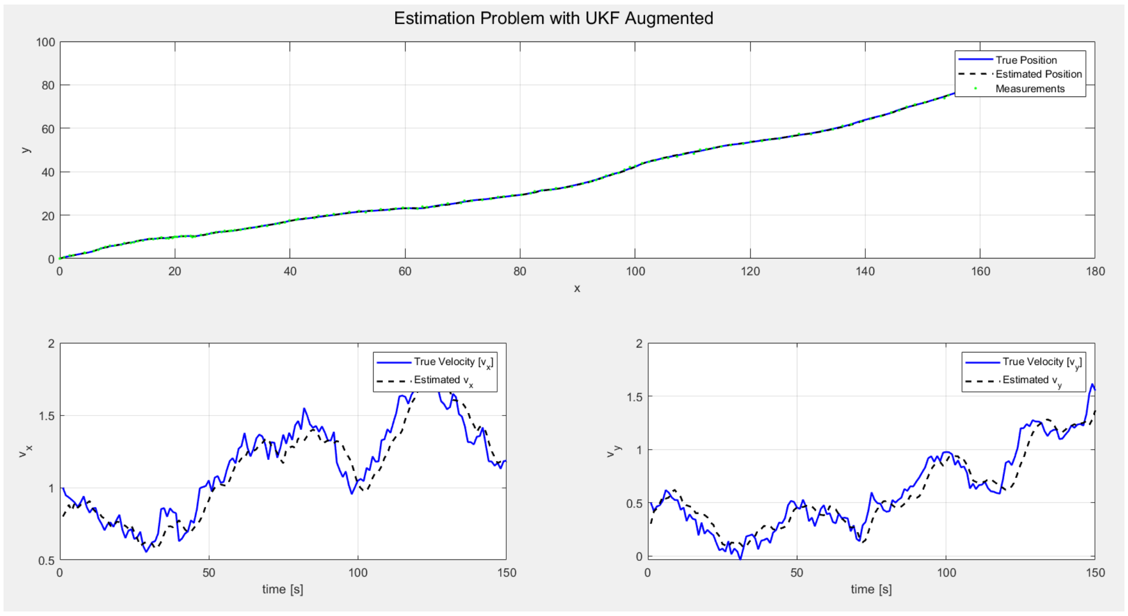

Figure 3.2: Estimation problem with UKF Augmented

About the Square Root UKF, we can make a comparison with the standard UKF.
As we can see in the **Figure 3.3**, both filters track the true velocity well, including changes and inflections.
In some places, the SR-UKF shows slightly smoother estimates, with fewer sharp deviations than the standard UKF. In fact, the differences are subtle.
A little difference is that the Square-Root UKF is slightly better, especially in its ability to absorb noise but the main difference is in the computational complexity.
The Square-Root UKF is more numerically stable due to Cholesky decomposition, the computational cost is lower, more efficient matrix square root operations and, therefore, it offers slight improvements in estimation smoothness, which may be beneficial in systems requiring high robustness or running on limited-precision hardware.

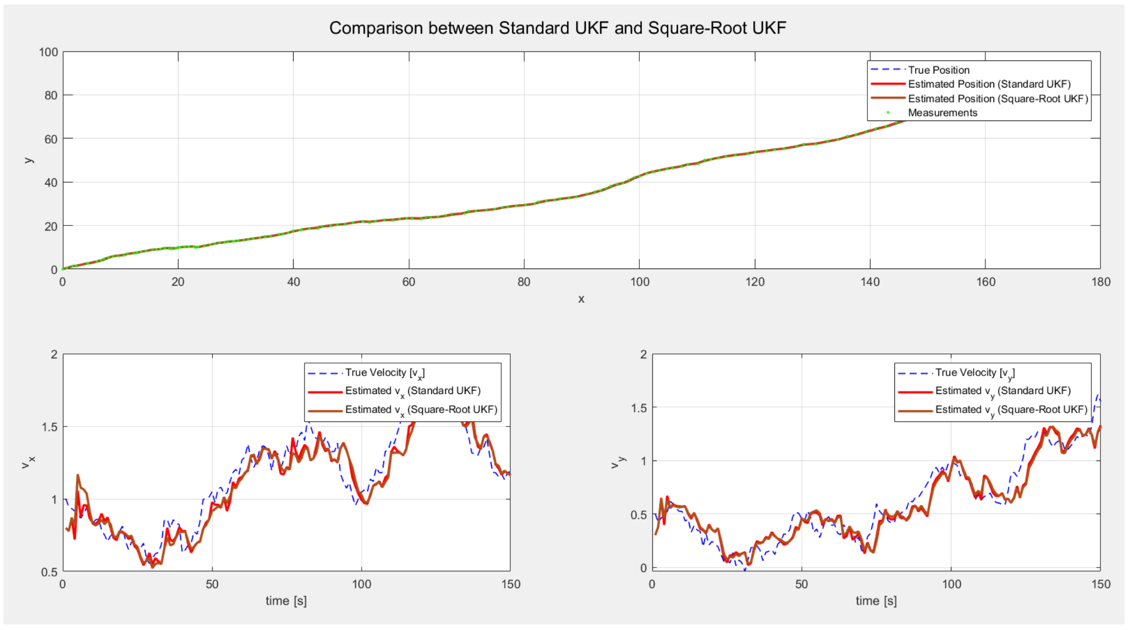

Figure 3.3: Comparison between UKF and SR-UKF

3.2.2 Not Accurate Sensors

In this case the trend of the estimates will be analyzed, especially by comparing the non-augmented UKF and the augmented UKF.
This section is very important because we will see how the filters act with the presence of significant noise due to ”low quality” sensors.
In the **Figure 3.4** we can see the comparison between UKF Non-Augmented and UKF Augmented.

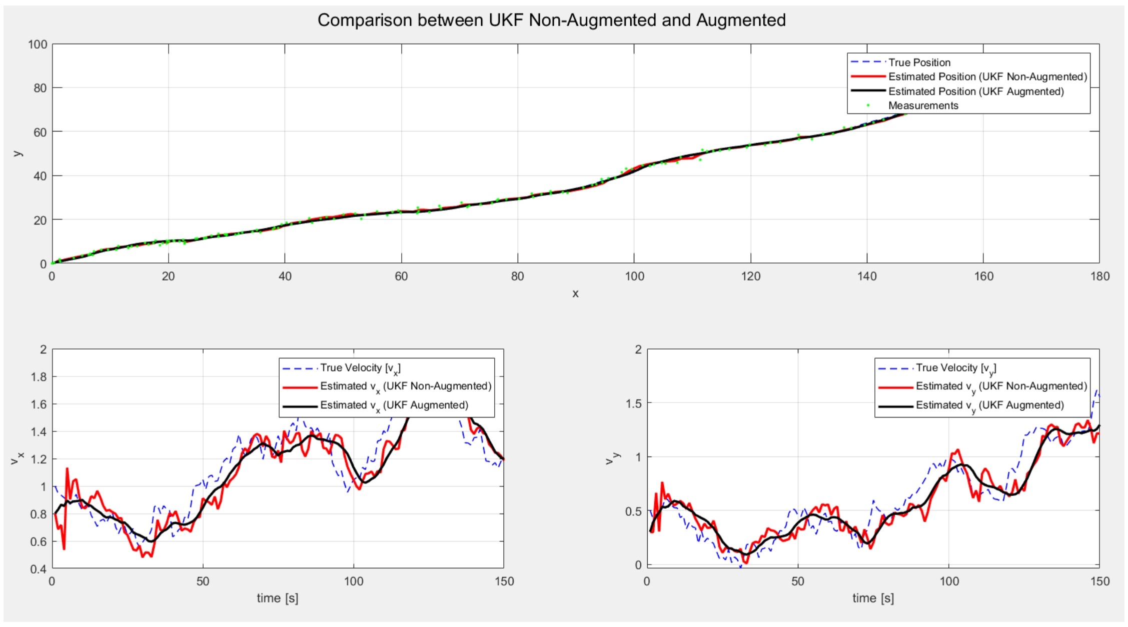

Figure 3.4: Comparison between UKF Non-Augmented and UKF Augmented

The UKF (Non-Augmented and Augmented) works very well in terms of accuracy of position but, about the velocity, UKF Augmented is better than Non-Augmented UKF because the estimated velocity is smoother.

The numerical stability is better in UKF Augmented due to noise modeling which is direct (augmented with noise).
Finally, about computational cost, the UKF Augmented one is higher than the UKF Non-Augmented one because it calculates more sigma points.

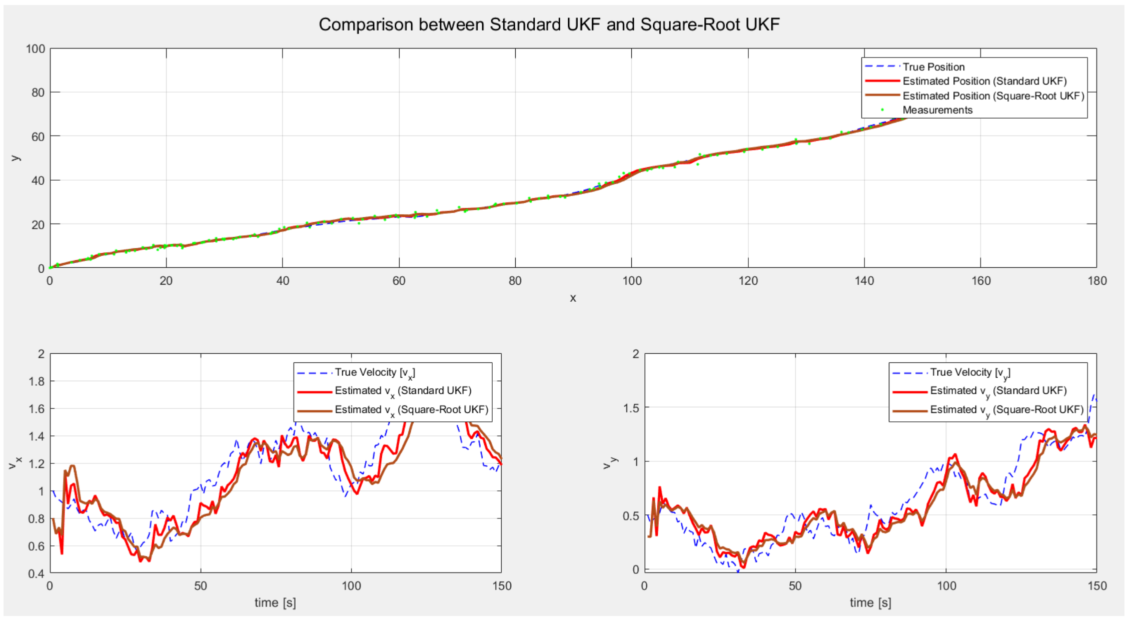

Figure 3.5: Comparison between Standard UKF and Square-Root UKF

As before, SR-UKF estimates appear slightly smoother because it's potentially better performance in noisy scenarios.
Slight deviations from the ground truth occur, especially at peaks and valleys, which are common in velocity estimation due to measurement noise and system nonlinearity.
As we can see in **Figure 3.5** SR-UKF shows slightly better noise suppression, particularly during sharp changes.

3.2.3 Stress Test

A stress test was designed (significantly increasing Q and V) to evaluate the robustness of the filters under critical conditions.
About Non-Augmented UKF, **Figure 3.6**, it shows lightly larger error in the early parts of the trajectory; in fact, velocities deviate more noticeably.
Noise handling is worse because the non-augmented UKF does not account for process noise directly in the sigma points.

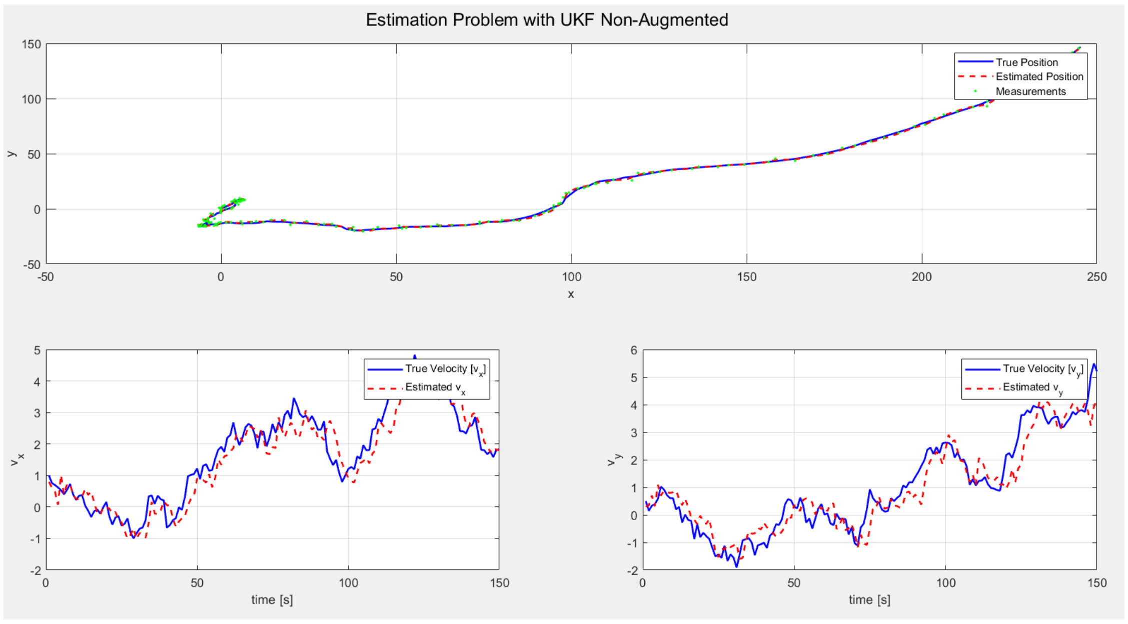

Figure 3.6: Estimation Problem with Non-Augmented UKF

About Augmented UKF, **Figure 3.7**, it tracks the position quite well even under stress. Also in this estimation, there are some visible deviation in velocity estimates, especially in rapid-change zones, but, the Augmented UKF is likely compensating for noise better due to inclusion of process noise in state.

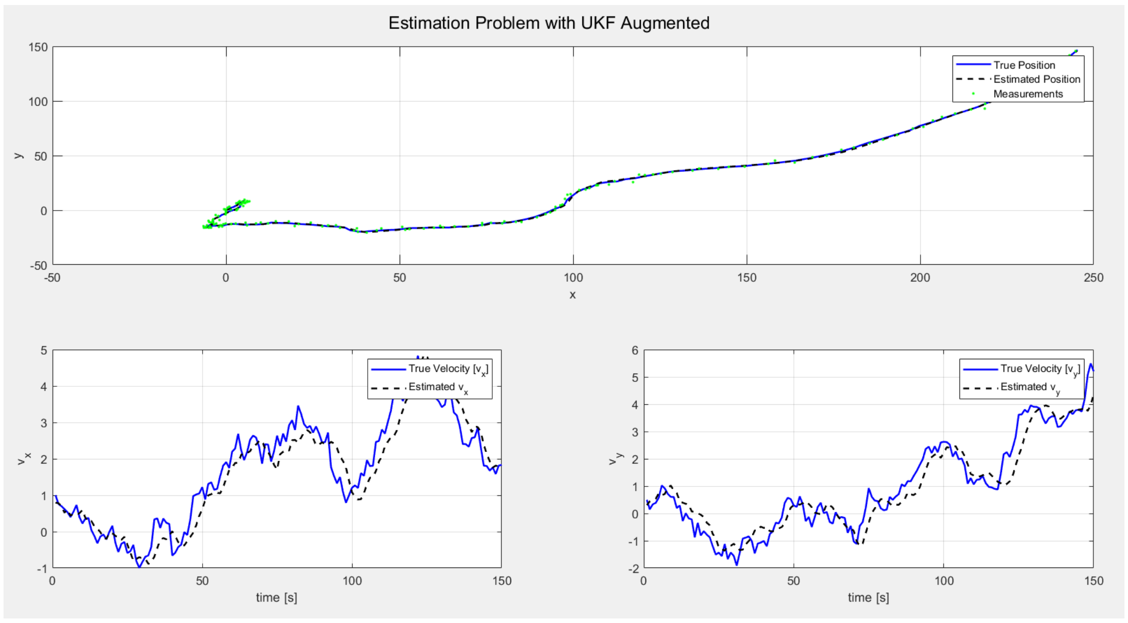

Figure 3.7: Estimation Problem with Augmented UKF

About the comparison between Standard UKF and SR-UKF under a stress test, both filters show almost identical position tracking, **Figure 3.8**.
But, SR-UKF offers slightly outperforms in velocity estimation, showing smoother tracking curves. Since SR-UKF is numerically more stable and better conditioned, we can conclude that it is advantageous to use this filter in stress test or high noise scenarios.

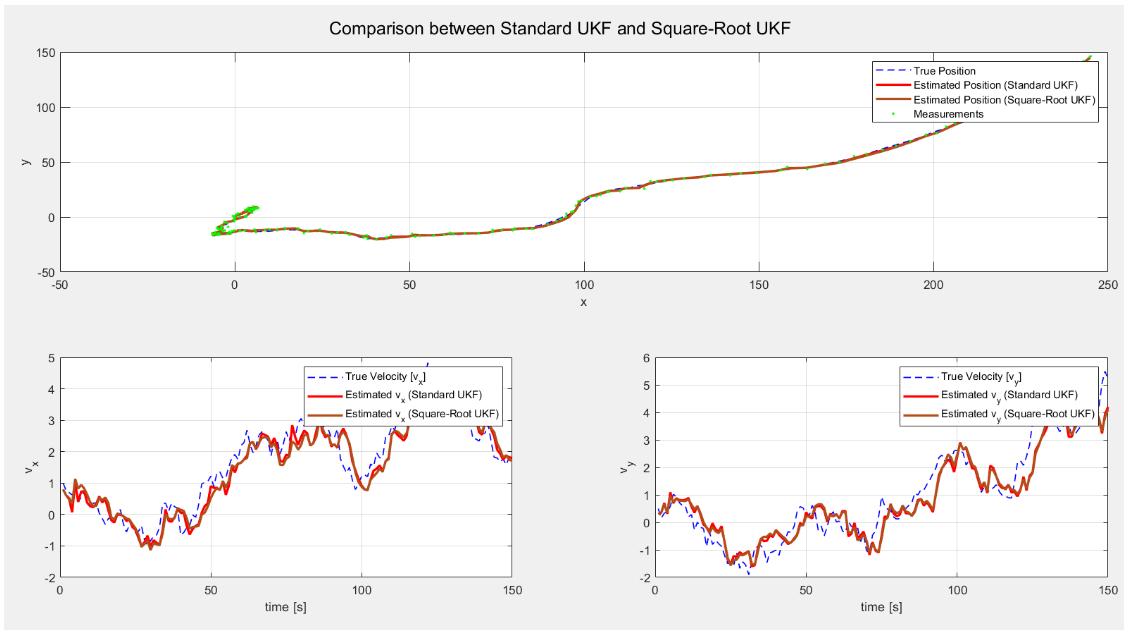

Figure 3.8: Comparison between Standard UKF and Square-Root UKF

Chapter 4

Conclusions

In this work, the Unscented Kalman Filter (UKF) has been thoroughly analyzed and applied in three different configurations: standard (non-augmented), augmented, and square-root.
The study has shown how the UKF, by using a deterministic sampling technique via sigma points, effectively addresses the limitations of the Extended Kalman Filter (EKF), particularly in dealing with nonlinear systems.
The theoretical analysis demonstrated how the non-augmented and augmented versions differ in their treatment of process and measurement noise. The augmented UKF includes the noise terms in the state vector, allowing for better modeling accuracy but at the cost of increased computational complexity.
About the simulations, results confirmed that:

- In scenarios with accurate sensors and low process noise, all versions of UKF provided excellent tracking, with the augmented and square-root versions offering slightly smoother and more stable estimates.

- With noisy sensors, the Augmented and Square-Root UKF outperformed the standard UKF, particularly in velocity estimation, thanks to better noise modeling and numerical stability.

- Under stress test conditions (high noise), the Square-Root UKF showed the best performance in terms of robustness and noise suppression, validating its improved numerical properties and efficient matrix handling.

Overall, this study highlights the trade-offs between computational complexity and estimation accuracy. While the standard UKF is lighter and simpler, the augmented and square-root variants offer superior performance in challenging environments. The choice of filter should therefore consider the specific system constraints, noise characteristics, and computational resources available.

Bibliography

[1] Erik A- Wan and Rudolph van der Merwe, *The Unscendent Kalman Filter for Nonlinear Estimation*

[2] Erik A- Wan and Rudolph van der Merwe, *The Unscendent Kalman Filter, Chapter 7*

[3] Yuanxin Wu, Dewen Hu, Member, IEEE, Meiping Wu and Xiaoping Hu, *Unscented Kalman Filtering for Additive Noise Case: Augmented vs. Non-augmented*

[4] Gabriel A. Terejanu, *Unscented Kalman Filter Tutorial*

[5] Erik A- Wan and Rudolph van der Merwe, *The Square-Root Unscendent Kalman Filter for state and parameter-estimation*