In [ ]:

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import sympy
from tools.task import wrap_angle
```

Task A:

Write the value for the covariance Q of the noise added to the observation function, knowing that the parameter *bearing_std* is its standard deviation. To find out which is the value of *bearing_std* you should look at the default parameters passed to *run.py* lines 44 - 121.Write the equation for the covariance R_t of the noise added to the transition function, as explained in class and their corresponding numeric values for the initial robot command $u = [\delta_{rot1}, \delta_{trans}, \delta_{rot2}]^\top = [0, 10, 0]^\top$. Find out the default values of α in *run.py* line 152. Then derive the equations for the Jacobians G_t , V_t and H_t , and evaluate them at the initial mean state $\mu_1 = [x, y, \theta]^\top = [180, 50, 0]^\top$ as it is considered in *run.py*. (It is not requested to evaluate observation Jacobians.)

$Q = \begin{pmatrix} \sigma^2_{bearing} & 0 \\ 0 & 0 \end{pmatrix}$ - from *tools.py* [102 line]

$Q = \begin{pmatrix} 0.35^2 & 0 \\ 0 & 0 \end{pmatrix}$ - from *run.py* [80-85 line]

$\sqrt{\Alpha} = [0.05, 0.001, 0.05, 0.01]$ - from *run.py* [79 line]

$\Alpha = [0.05^2, 0.001^2, 0.05^2, 0.01^2]$ -from *run.py* [152 line]

$M_t = \begin{pmatrix} \alpha_1\delta^2_{rot_1} + \alpha_2\delta^2_{trans} & 0 & 0 \\ 0 & \alpha_3\delta^2_{trans} + \alpha_4(\delta^2_{rot_1} + \delta^2_{rot_2}) & 0 \\ 0 & 0 & \alpha_1\delta^2_{rot_2} + \alpha_2\delta^2_{trans} \end{pmatrix}$ - from *task.py* [136 line]

$$u = [\delta_{rot_1}, \delta_{trans}, \delta_{rot_2}]^\top = [0, 10, 0]^\top$$

$$M_t = \begin{pmatrix} \alpha_2\delta^2_{trans} & 0 & 0 \\ 0 & \alpha_3\delta^2_{trans} & 0 \\ 0 & 0 & \alpha_2\delta^2_{trans} \end{pmatrix}$$

$$M_t = \begin{pmatrix} 0.001^2 * 100 & 0 & 0 \\ 0 & 0.05^2 * 100 & 0 \\ 0 & 0 & 0.001^2 * 100 \end{pmatrix}$$

$$M_t = \begin{pmatrix} 0.0001 & 0 & 0 \\ 0 & 0.25 & 0 \\ 0 & 0 & 0.0001 \end{pmatrix}$$

For calculating *R_t* we need find *V_t*:

In [2]:
```
V_t = np.array([[0, 1, 0],
                [10, 0, 0],
                [1, 0, 1]])
M_t = np.array([[0.0001, 0, 0],
                [0, 0.25, 0],
                [0, 0, 0.0001]])
R_t = V_t @ M_t @ V_t.T
print('R_t =', R_t)
```

```
R_t = [[2.5e-01 0.0e+00 0.0e+00]
 [0.0e+00 1.0e-02 1.0e-03]
 [0.0e+00 1.0e-03 2.0e-04]]
```

$$G_t = \frac{\partial g(x_{t-1}, u_t, \varepsilon_t)}{\partial x_{t-1}}|_{\mu_{t-1}, \varepsilon_t = 0} = \begin{bmatrix} 1 & 0 & -\delta_{trans} \sin(\theta + \delta_{rot1}) \\ 0 & 1 & \delta_{trans} \cos(\theta + \delta_{rot1}) \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -10 * sin \\ 0 & 1 & 10 * cos \\ 0 & 0 & 1 \end{bmatrix}$$

```
V_t = \frac{\partial g(x_{t-1}, u_t, \varepsilon_t)}{\partial u_{t}} |_{\mu_{t-1}, \varepsilon_t =0} = \begin{bmatrix}
   - \delta_{trans} \sin (\theta + δ_{rot1}) & \cos (\theta + δ_{rot1}) & 0 \\
   \delta_{trans} \cos (\theta + δ_{rot1}) & \sin (\theta + δ_{rot1}) & 0 \\
   1 & 0 & 1 \\
\end{bmatrix} $= ({\mu_{0} = \begin{bmatrix} x & y & z\\ \end{bmatrix} = \begin{bmatrix} 180 & 50 & 0\\
\end{bmatrix} )$ = \begin{bmatrix}
   -10 * sin(0 + 0) & cos(0 + 0) & 0 \\
   10 * cos( 0 + 0)& sin(0 + 0) & 0 \\
   1 & 0 & 1 \\
\end{bmatrix} $ = \begin{bmatrix}
   0 & 1 & 0 \\
   10 & 0 & 0 \\
   1 & 0 & 1 \\
\end{bmatrix}
```

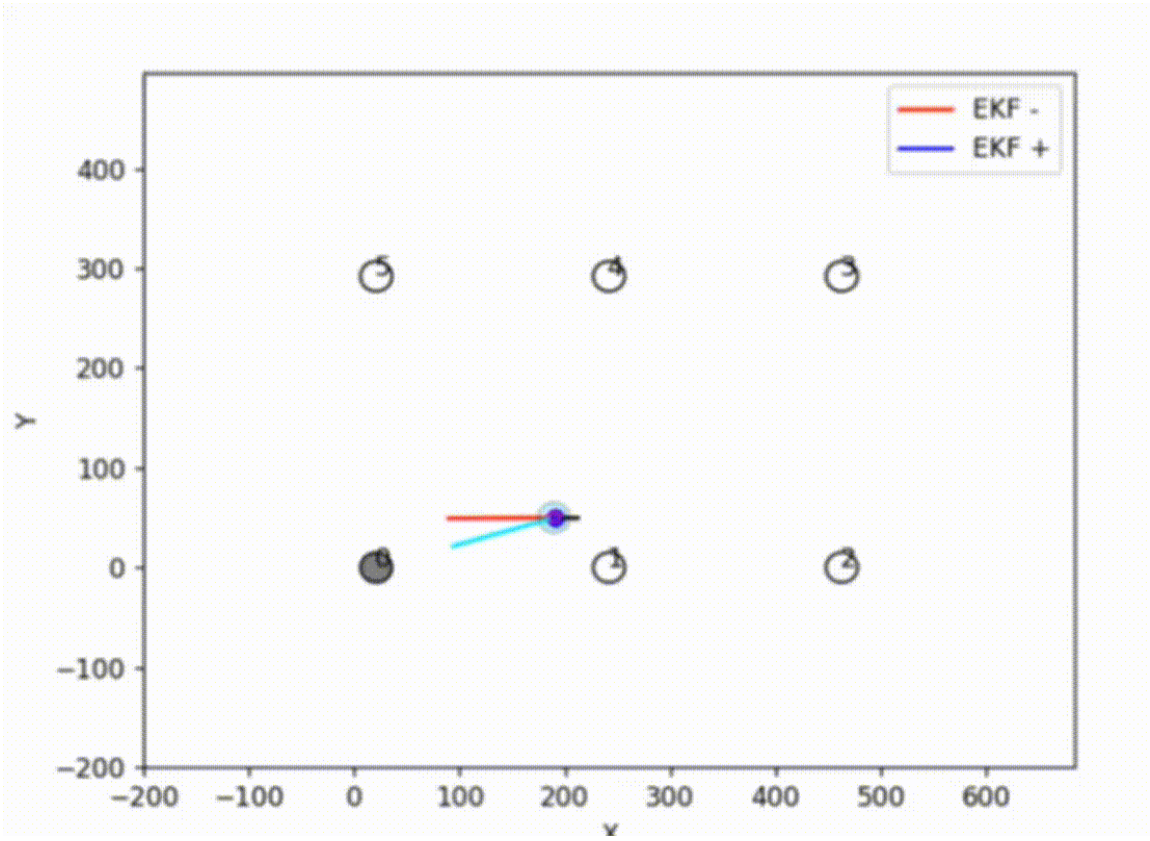
$$H_t = \frac{\partial h(x_t)}{\partial x_t}|_{\bar{\mu}_t} = \begin{bmatrix} \frac{m_{i,y} - \bar{\mu}_{t,y}}{(m_{i,x} - \bar{\mu}_{t,x})^2 + (m_{i,y} - \bar{\mu}_{t,y})^2} & \frac{-(m_{i,x} - \bar{\mu}_{t,x})}{(m_{i,x} - \bar{\mu}_{t,x})^2 + (m_{i,y} - \bar{\mu}_{t,y})^2} & -1 \end{bmatrix}$$

Task B: Implement EKF and PF-based robot localization using odometry and bearing-only observations to features in a landmark map.

1. EKF is presented in file `filters/ekf.py`

In [3]:
```
from IPython.display import Image
Image(url='video_ekf.gif')
```

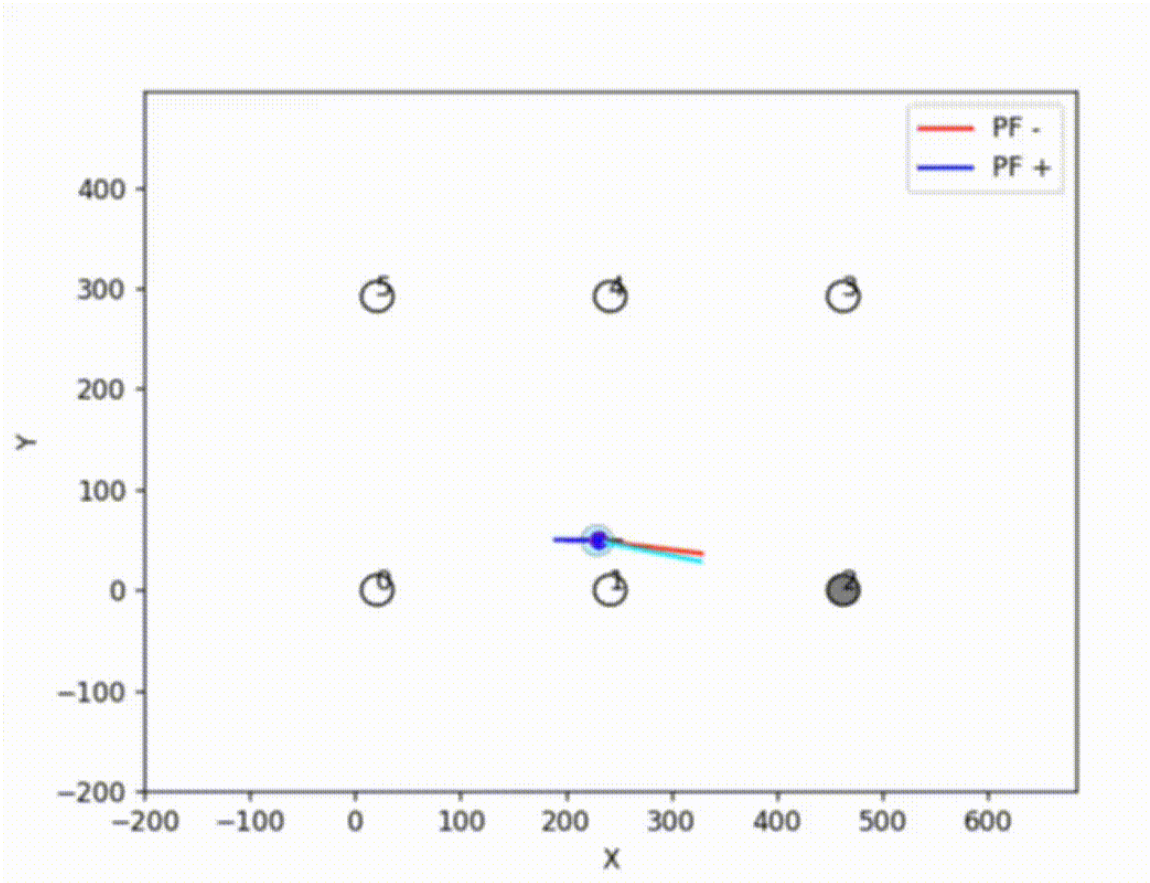
Out[3]:



PF is presented in file `filters/pf.py`

In [5]:
```
Image(url='video_pf.gif')
```

Out[5]:

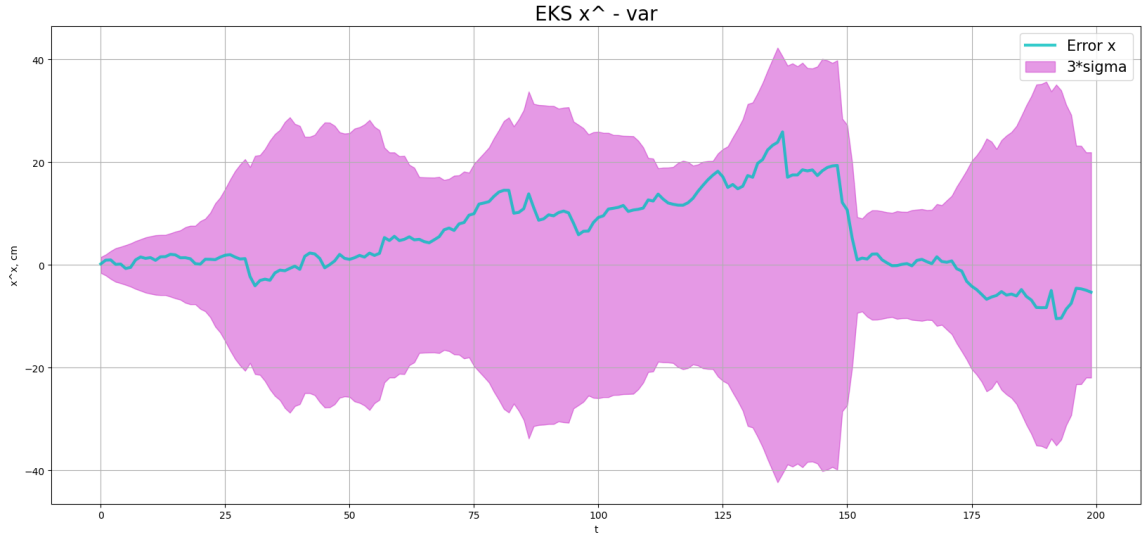


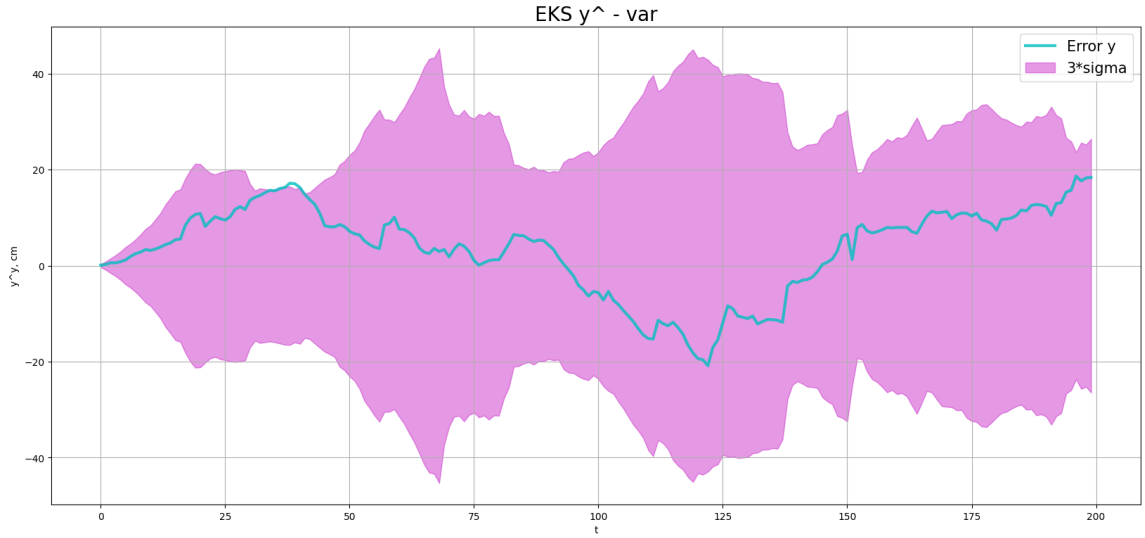
Task C: Create plots of pose error versus time i.e., a plot of $\hat{x} - x$ vs. t, $\hat{y} - y$ vs t, and $\theta - \theta$ vs. t where $(\hat{x}, \hat{y}, \theta)$ is the filter estimated pose and (x, y, θ) is the ground-truth actual pose known only to the simulator. Plot the error in blue and in red plot the $\pm 3\sigma$ uncertainty bounds. Your state error should lie within these bounds approximately 99.73% of the time (assuming Gaussian statistics). For the PF, use the sample mean and variance.

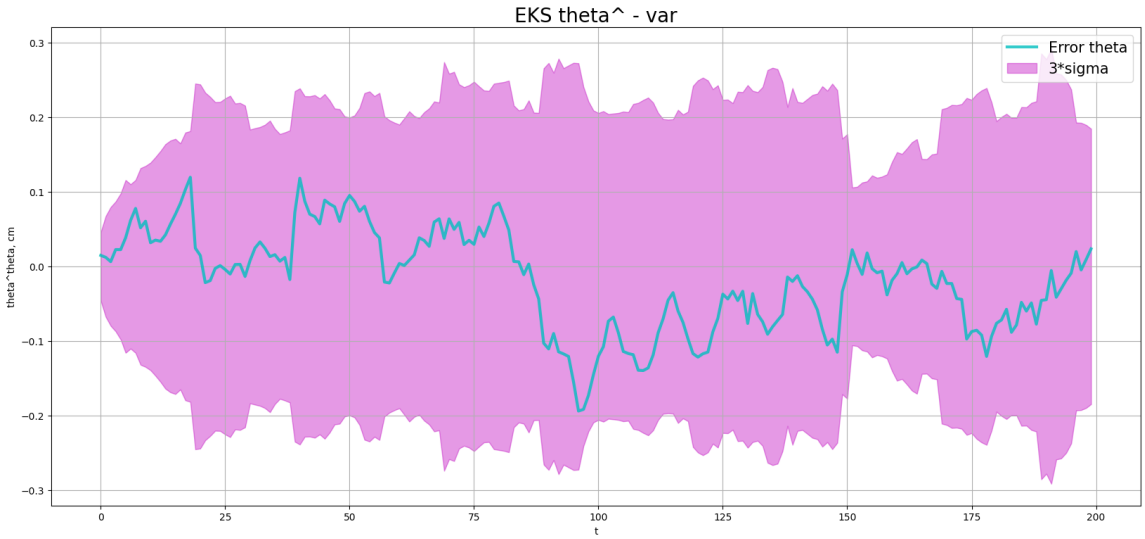
In [9]:
```
def angle(angle):
    pi_2 = 2 * np.pi
    while angle < -np.pi:
        angle += pi_2

    while angle >= np.pi:
        angle -= pi_2

    return angle
```

In [10]:
```
def plot(inp, outp, var):
    est = {'x': 1, 'y': 2, 'theta': 3}
    v_hat = outp.f.mean_trajectory[:, est[var] - 1]
    v = inp.f.real_robot_path[:, est[var] - 1]
    v_std = np.sqrt(outp.f.covariance_trajectory[est[var] - 1, est[var] - 1
    if var == 'theta':
        v = np.array([angle(v_hat[i] - v[i]) for i in range(len(v))])
        v_hat = 2 * v
    t = np.linspace(0, len(v) - 1, len(v))

    plt.subplot(3, 1, est[var])
    plt.plot(t, v_hat - v, linewidth=3, color='c', label='Error ' + var, al
    plt.fill_between(t, 3 * v_std, -3 * v_std, color='m', alpha=0.4, label=
    plt.title('EKS ' + var +'^ - ' + 'var', fontsize=20)
    plt.legend(fontsize=15)
    plt.xlabel('t')
    plt.grid(True)
    plt.ylabel(var + '^'  + var + ', cm')
```

In [11]:
```
plt.figure(figsize=[20,30])
plot(np.load('/Users/daria/Desktop/PS2/out_pf/input_data.npy'),np.load('/Us
plot(np.load('/Users/daria/Desktop/PS2/out_pf/input_data.npy'),np.load('/Us
plot(np.load('/Users/daria/Desktop/PS2/out_pf/input_data.npy'),np.load('/Us
```







Task D: Once your filters are implemented, please investigate some properties of them. How do they behave

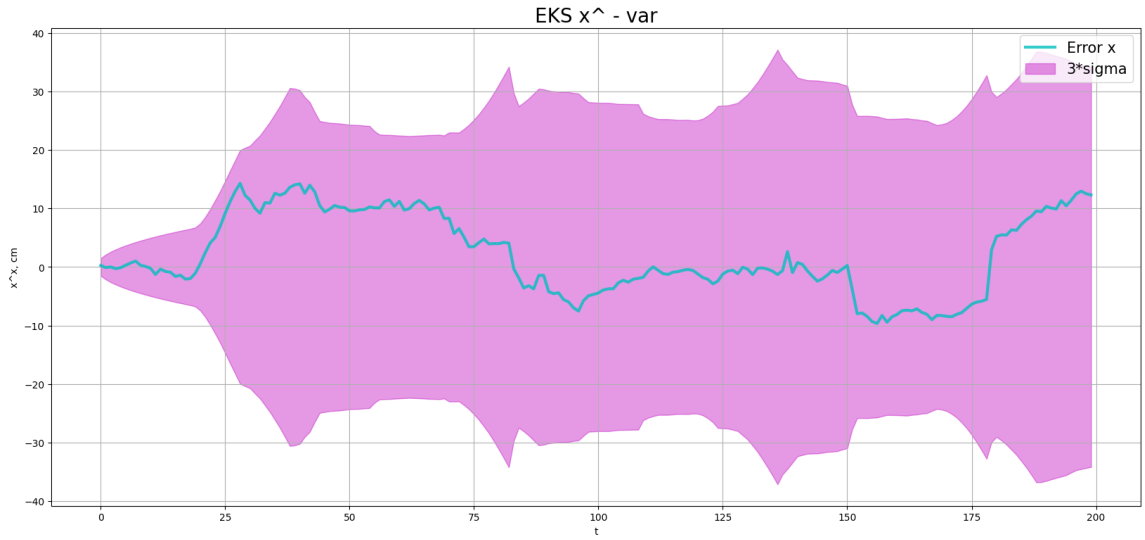
- as the sensor or motion noise go toward zero? (Please provide plots and explanation)
- as the number of particles decrease?
- if the filter noise parameters underestimate or overestimate the true noise parameters? Please clarify what underestimation and overestimation of noise is?

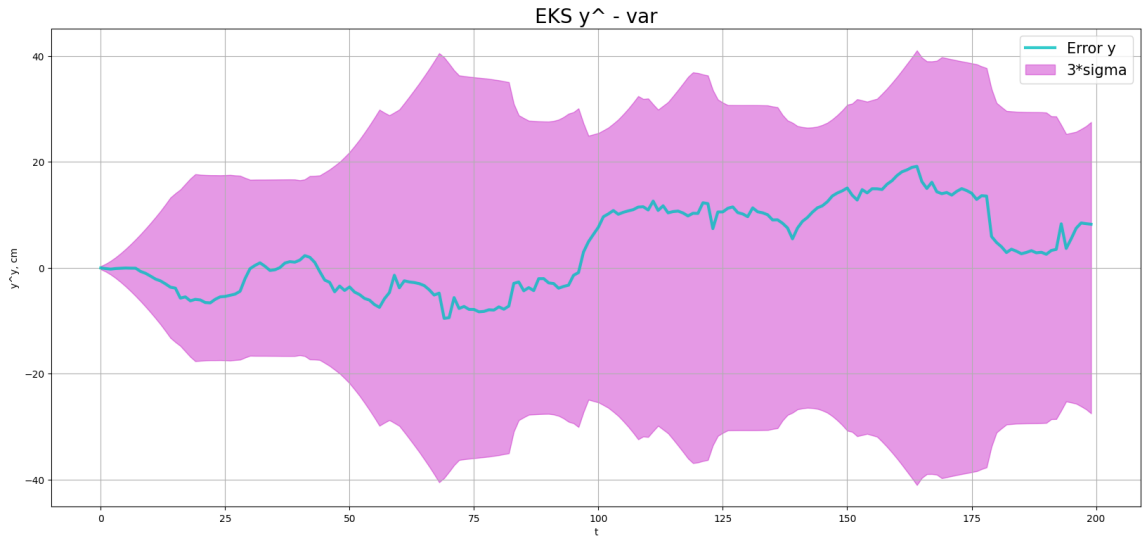
(Items 1 and 3 are for EKF and "the noise" refers to both observation and control noises)

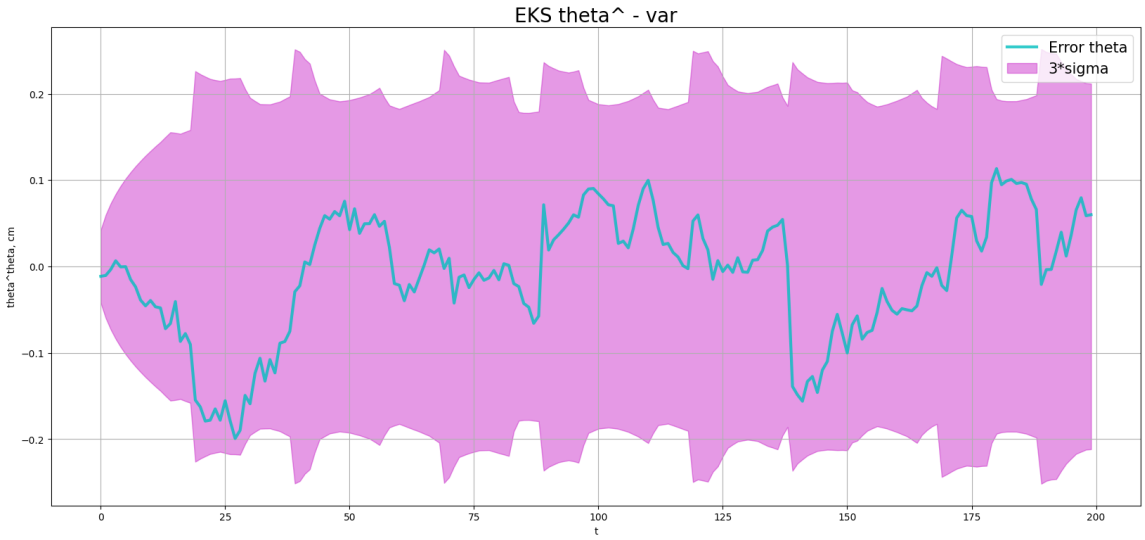
Type *Markdown* and LaTeX: α^2

1. Q goes to zero

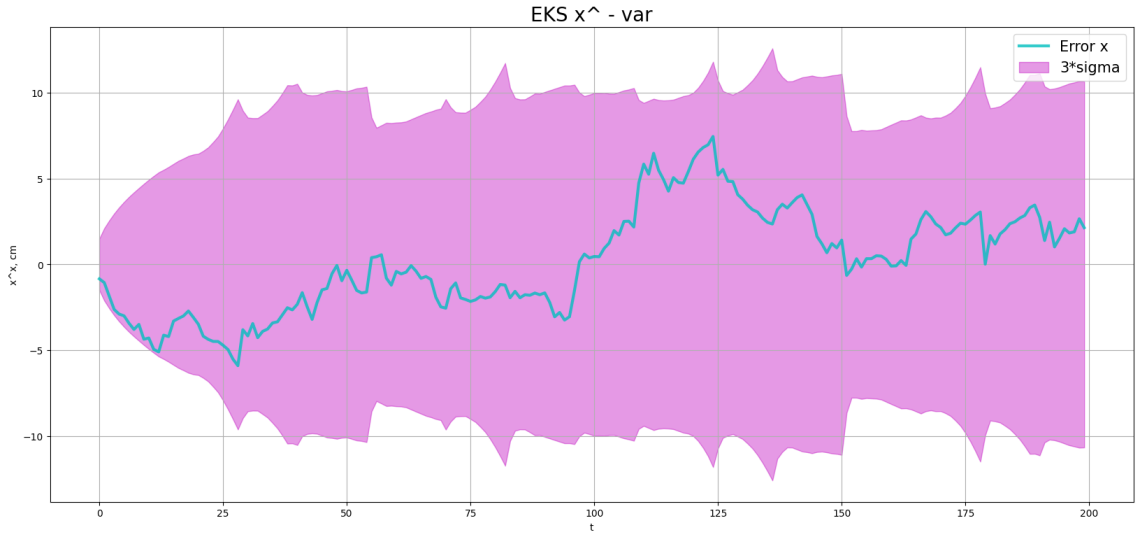
In [12]:
```
#Q = 0.2^2
plt.figure(figsize=[20,30])
plot(np.load('Task_D/set_q_1/input_data.npy'),np.load('Task_D/set_q_1/outpu
plot(np.load('Task_D/set_q_1/input_data.npy'),np.load('Task_D/set_q_1/outpu
plot(np.load('Task_D/set_q_1/input_data.npy'),np.load('Task_D/set_q_1/outpu
```

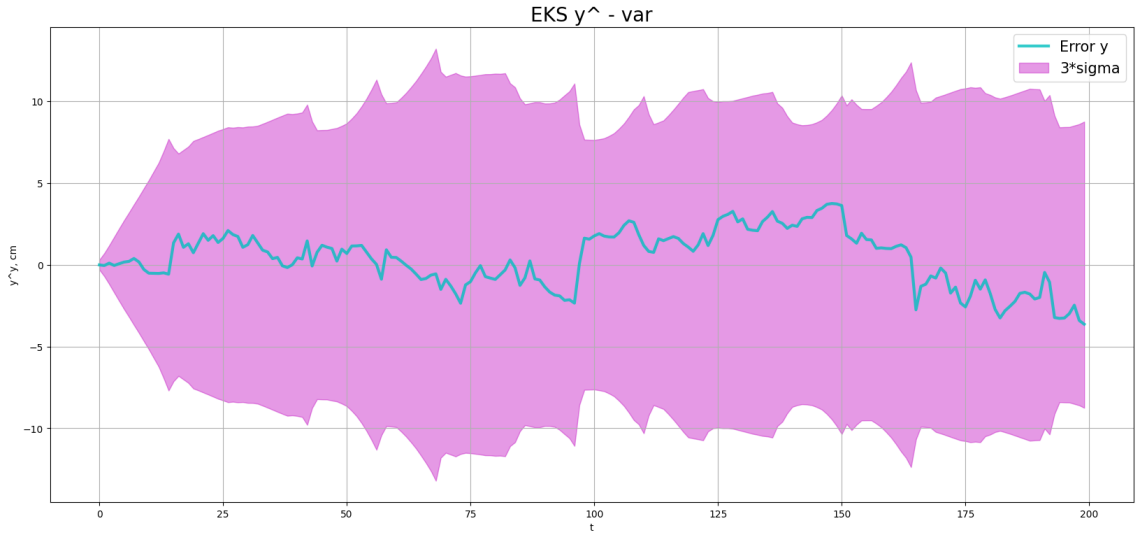


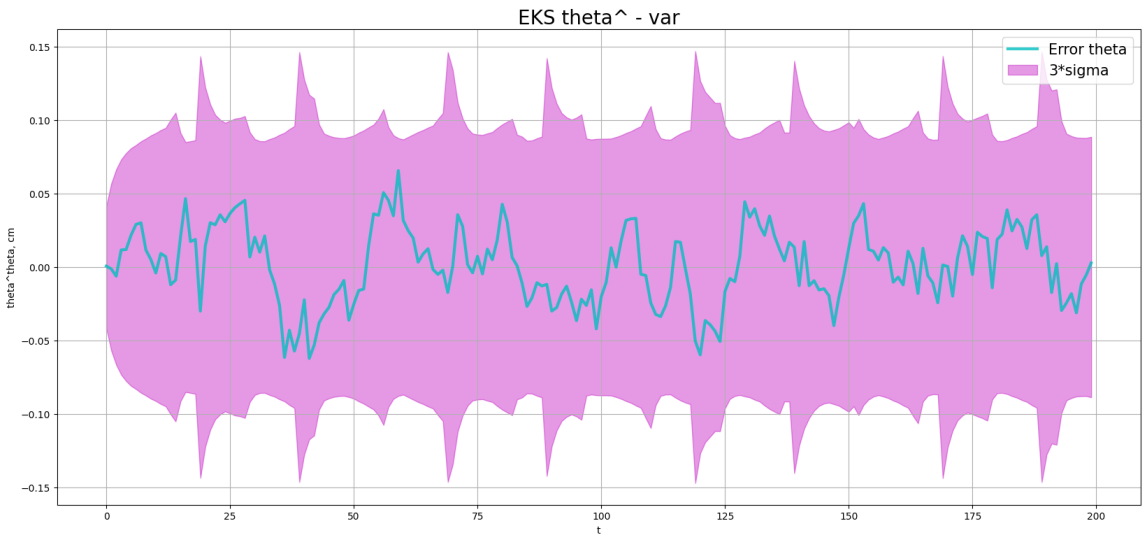




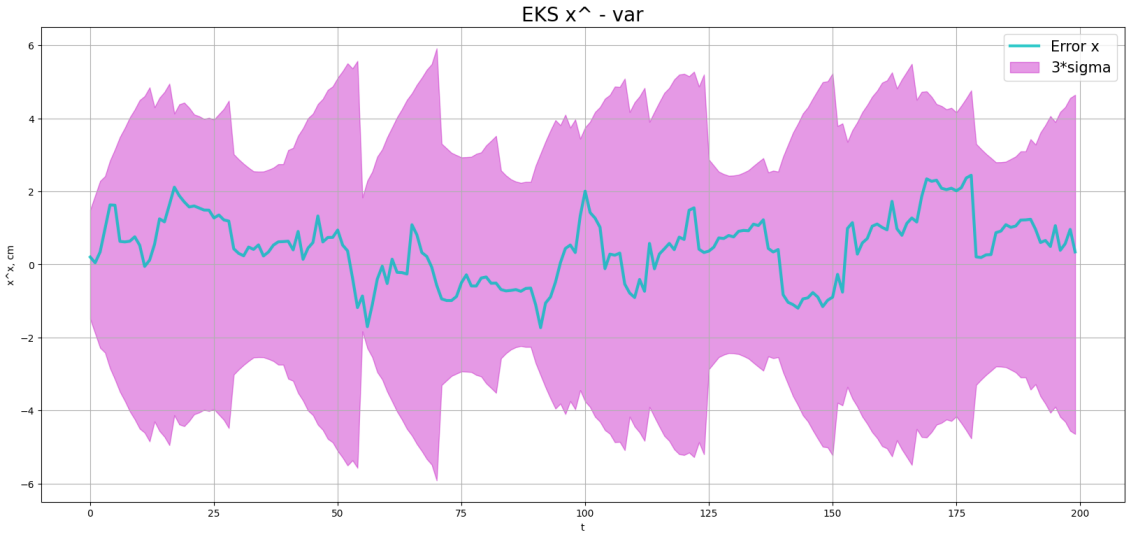
In [13]:
```
#Q = 0.05^2
plt.figure(figsize=[20,30])
plot(np.load('Task_D/set_q_2/input_data.npy'),np.load('Task_D/set_q_2/outpu
plot(np.load('Task_D/set_q_2/input_data.npy'),np.load('Task_D/set_q_2/outpu
plot(np.load('Task_D/set_q_2/input_data.npy'),np.load('Task_D/set_q_2/outpu
```

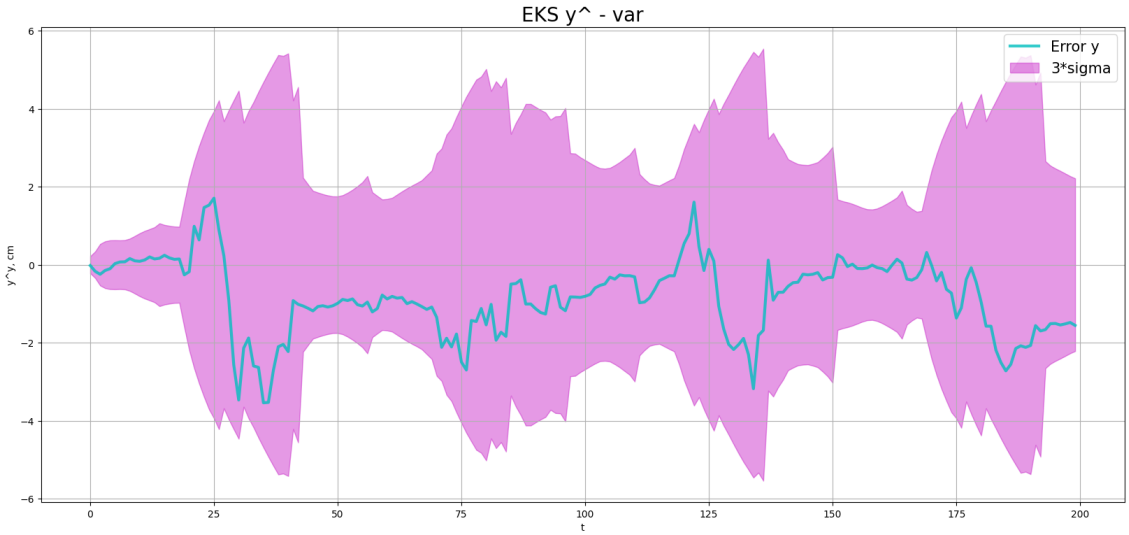






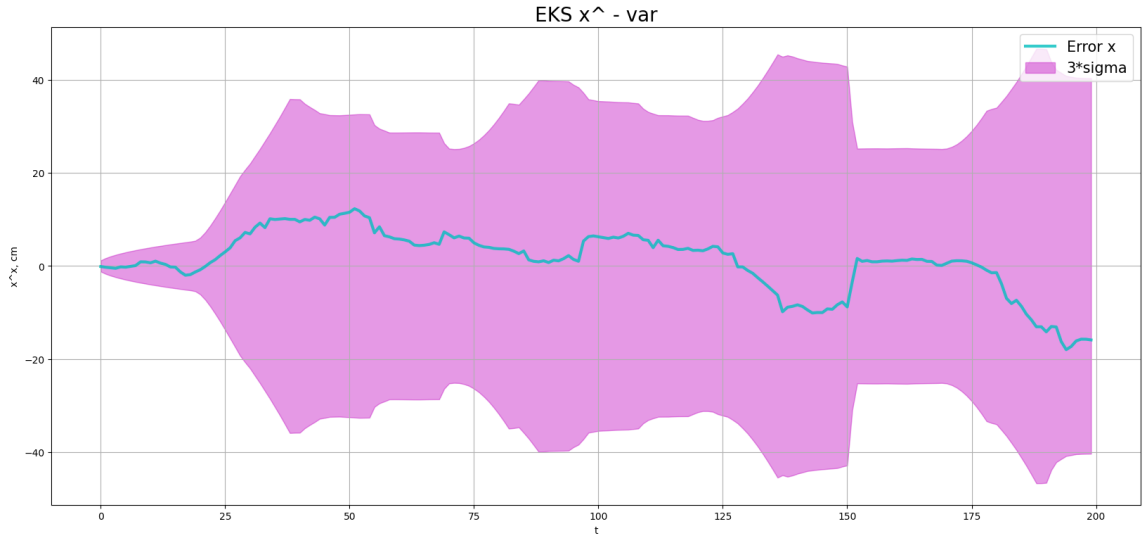
In [14]:
```
#Q = 0
plt.figure(figsize=[20,30])
plot(np.load('Task_D/set_q_3/input_data.npy'),np.load('Task_D/set_q_3/outpu
plot(np.load('Task_D/set_q_3/input_data.npy'),np.load('Task_D/set_q_3/outpu
plot(np.load('Task_D/set_q_3/input_data.npy'),np.load('Task_D/set_q_3/outpu
```

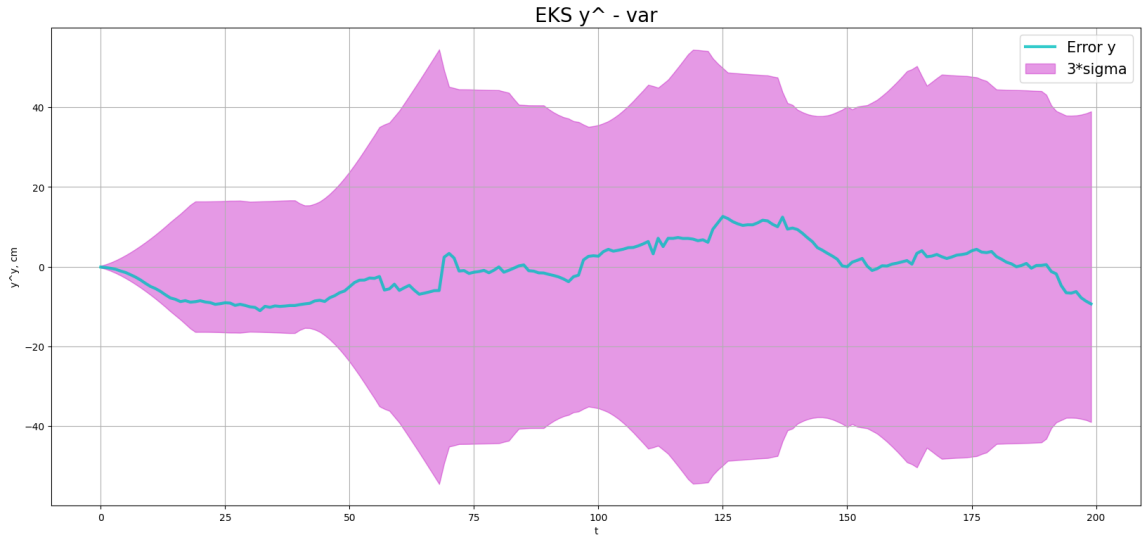


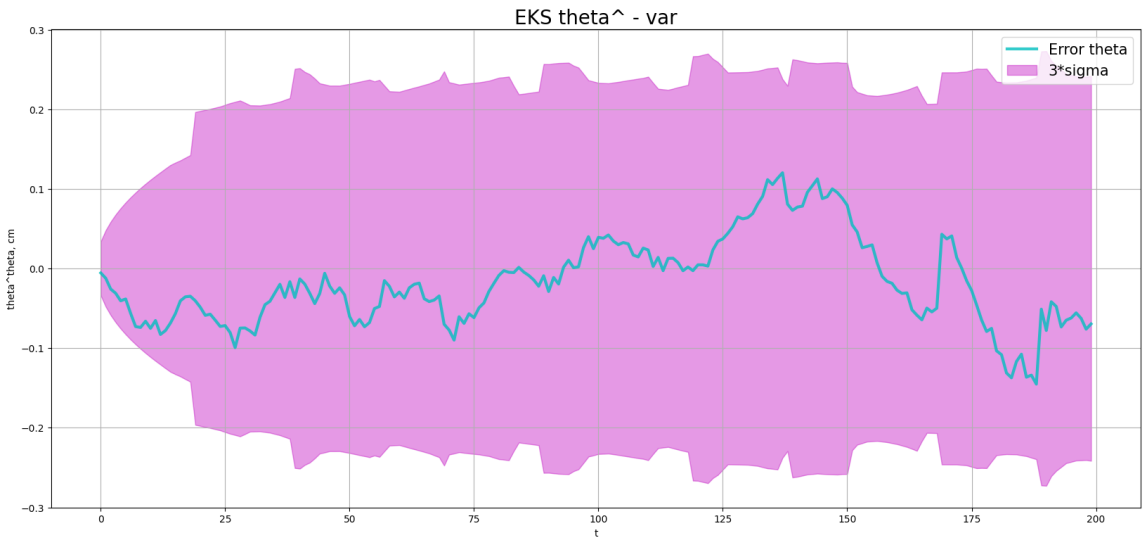


2. Alpa goes to zero

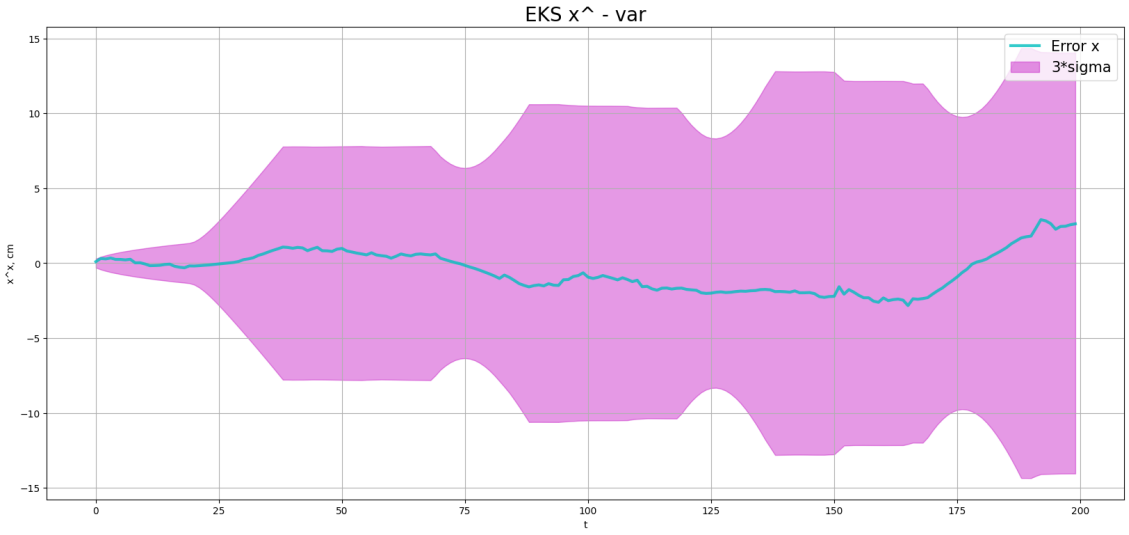
In [15]:
```
plt.figure(figsize=[20,30])
plot(np.load('Task_D/Exp_1/M_n/set_1/input_data.npy'),np.load('Task_D/Exp_1/
plot(np.load('Task_D/Exp_1/M_n/set_1/input_data.npy'),np.load('Task_D/Exp_1/
plot(np.load('Task_D/Exp_1/M_n/set_1/input_data.npy'),np.load('Task_D/Exp_1/
```

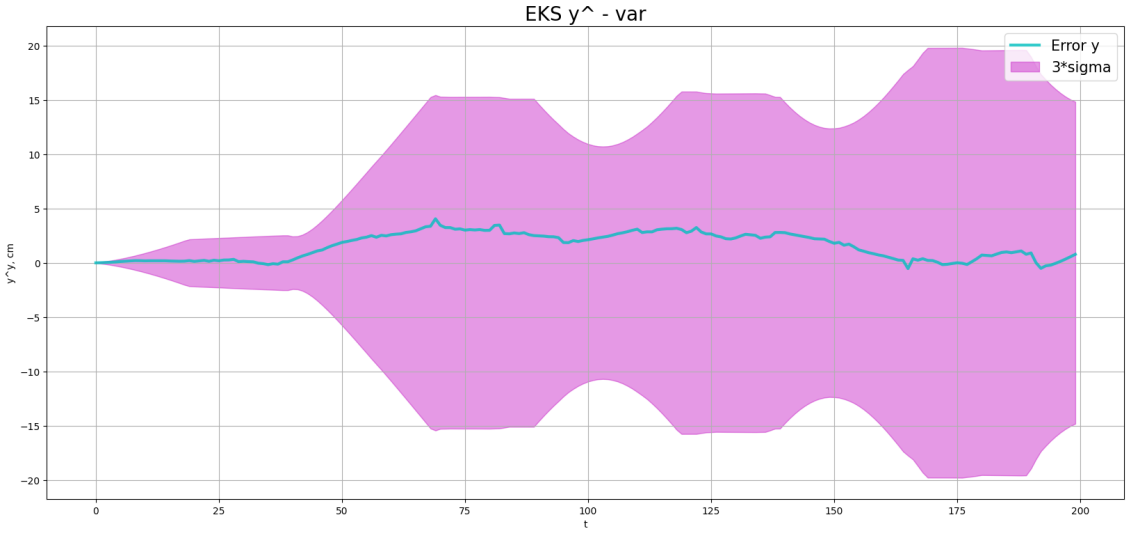


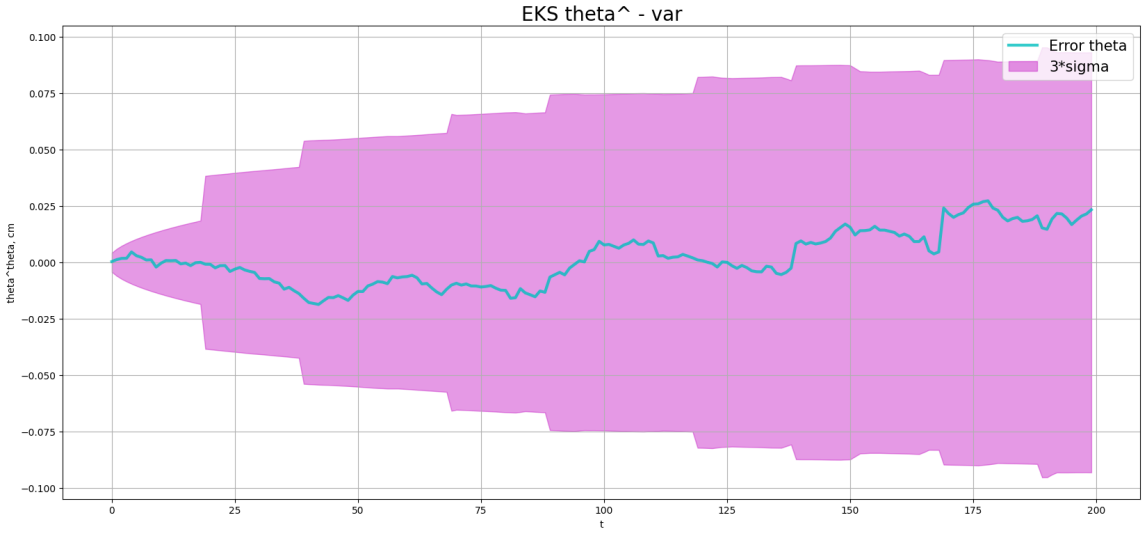




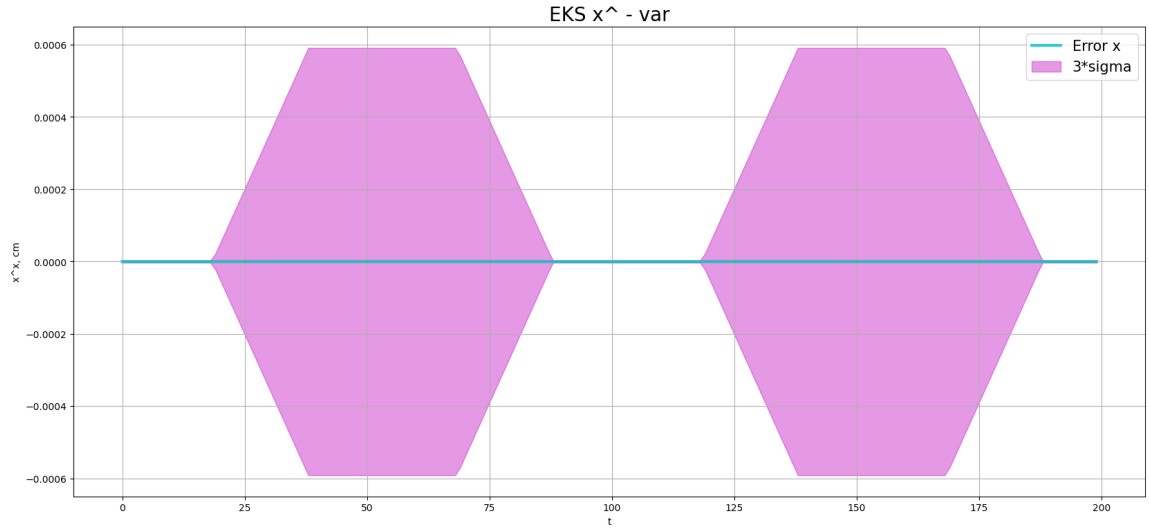
In [16]:
```
plt.figure(figsize=[20,30])
plot(np.load('Task_D/Exp_1/M_n/set_2/input_data.npy'),np.load('Task_D/Exp_1/
plot(np.load('Task_D/Exp_1/M_n/set_2/input_data.npy'),np.load('Task_D/Exp_1/
plot(np.load('Task_D/Exp_1/M_n/set_2/input_data.npy'),np.load('Task_D/Exp_1/
```

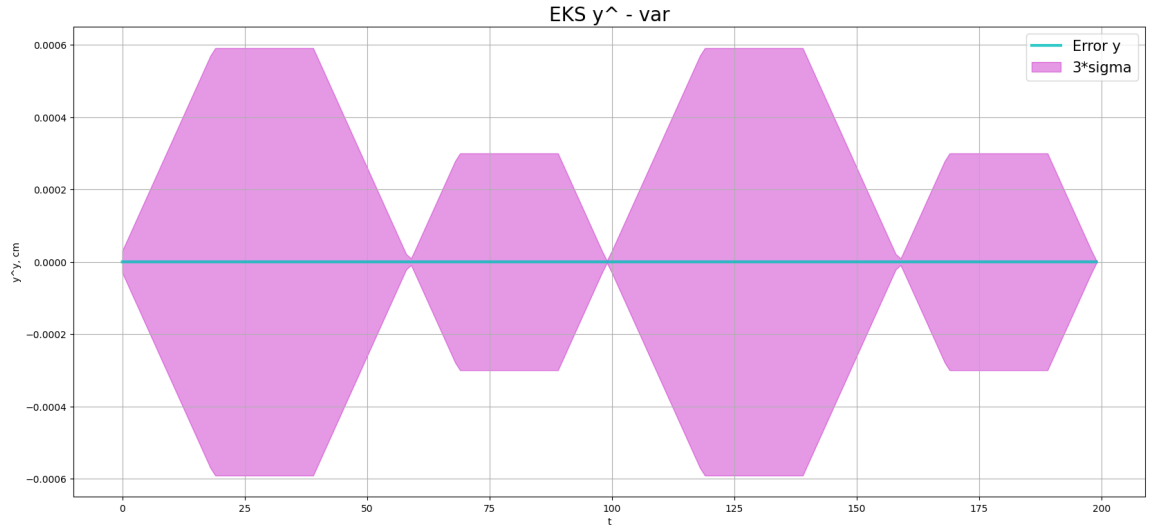


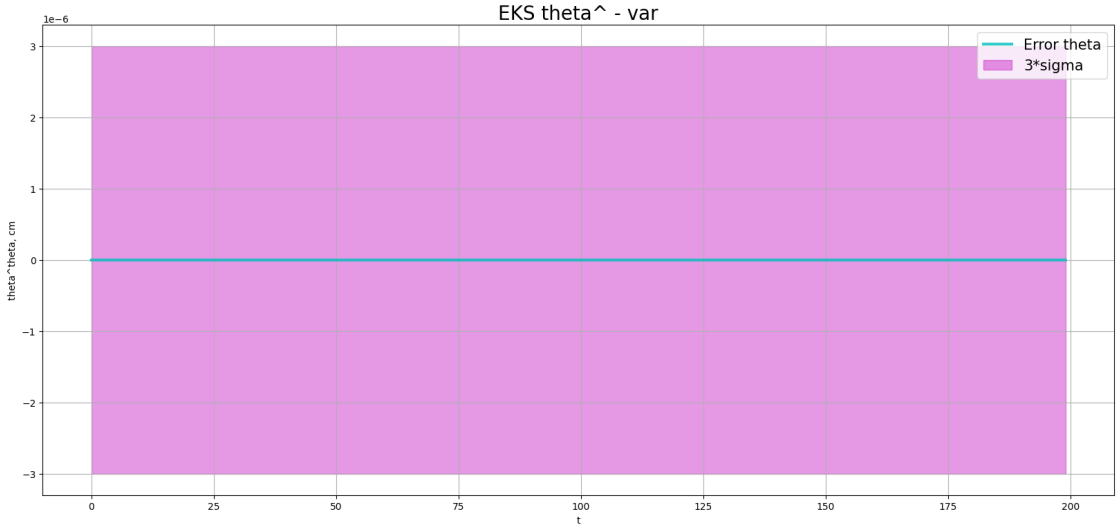




In [17]:
```
plt.figure(figsize=[20,30])
plot(np.load('Task_D/Exp_1/M_n/set_3/input_data.npy'),np.load('Task_D/Exp_1/
plot(np.load('Task_D/Exp_1/M_n/set_3/input_data.npy'),np.load('Task_D/Exp_1/
plot(np.load('Task_D/Exp_1/M_n/set_3/input_data.npy'),np.load('Task_D/Exp_1/
```



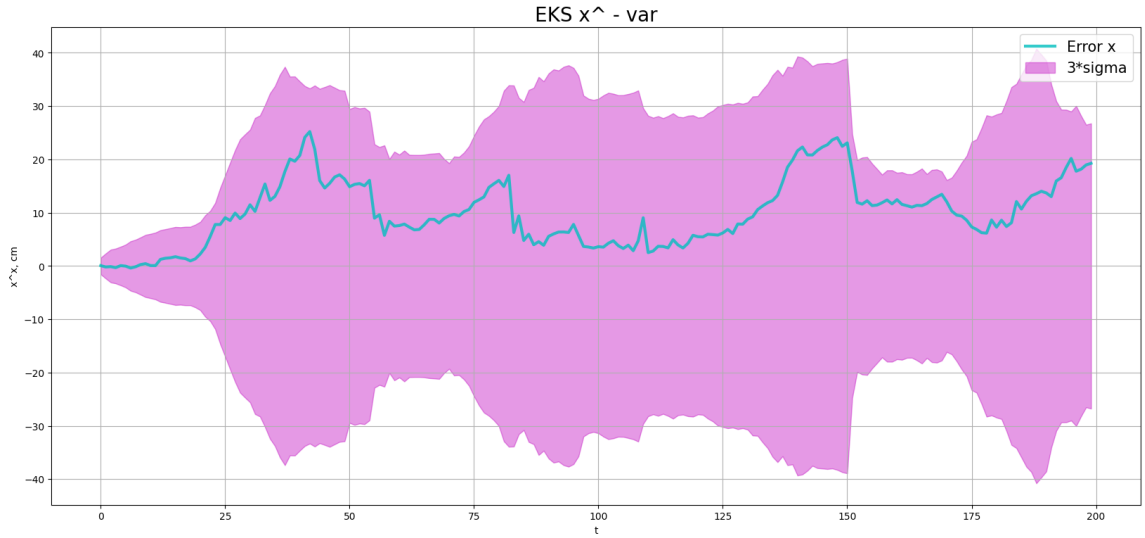


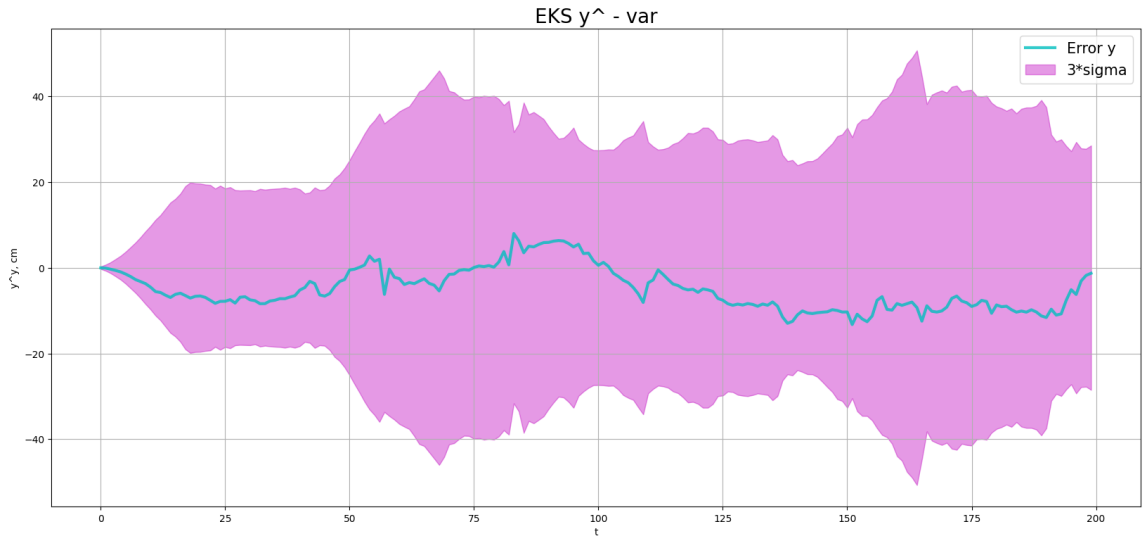


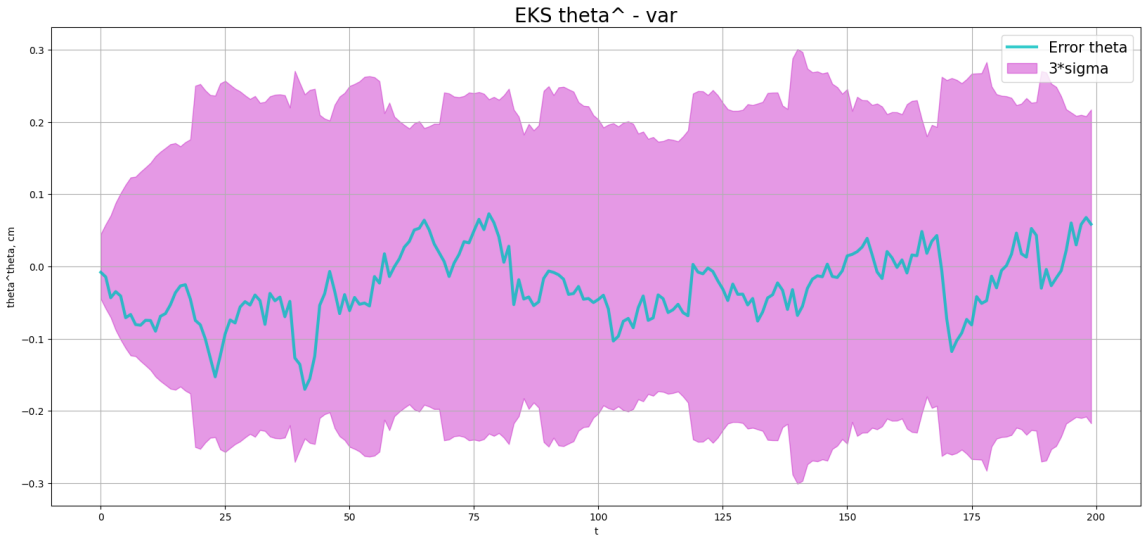
Conclusion: With low motion noise, a robot cannot account for noise that affects the transition function and therefore thinks that it is following the commanded odometry sequence. But it is not true. That means that uncertainty decreases, but since we still have motion noise than cannot absolutely rely on measurements and eleminate estimation errors.

3. Let's check how does PF behaves when amount of particles is decreased.

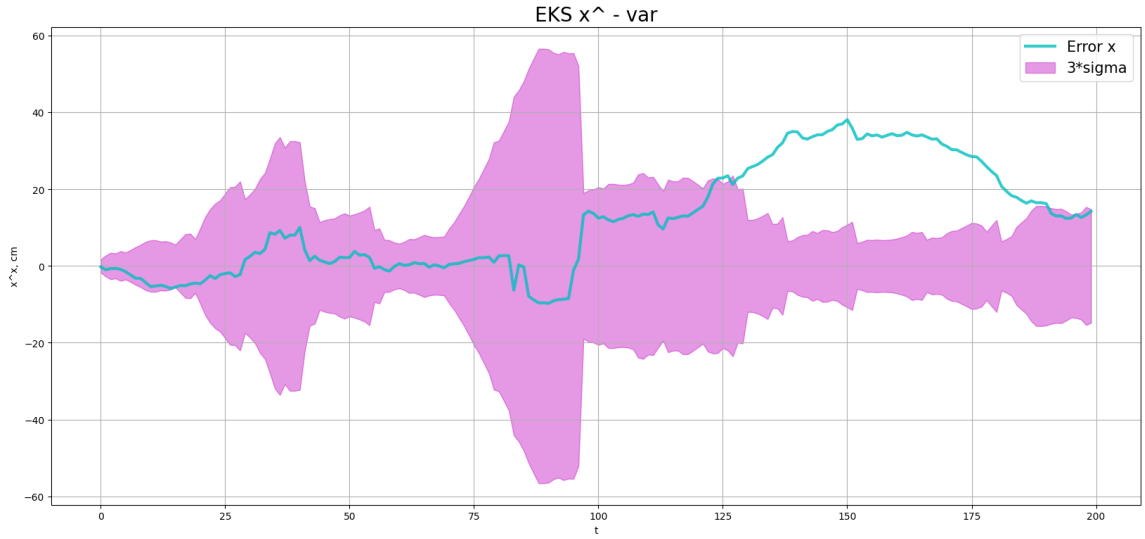
In [18]:
```
#numb = 150
plt.figure(figsize=[20,30])
plot(np.load('Task_D/Exp_2/set_1/input_data.npy'),np.load('Task_D/Exp_2/set_
plot(np.load('Task_D/Exp_2/set_1/input_data.npy'),np.load('Task_D/Exp_2/set_
plot(np.load('Task_D/Exp_2/set_1/input_data.npy'),np.load('Task_D/Exp_2/set_
```

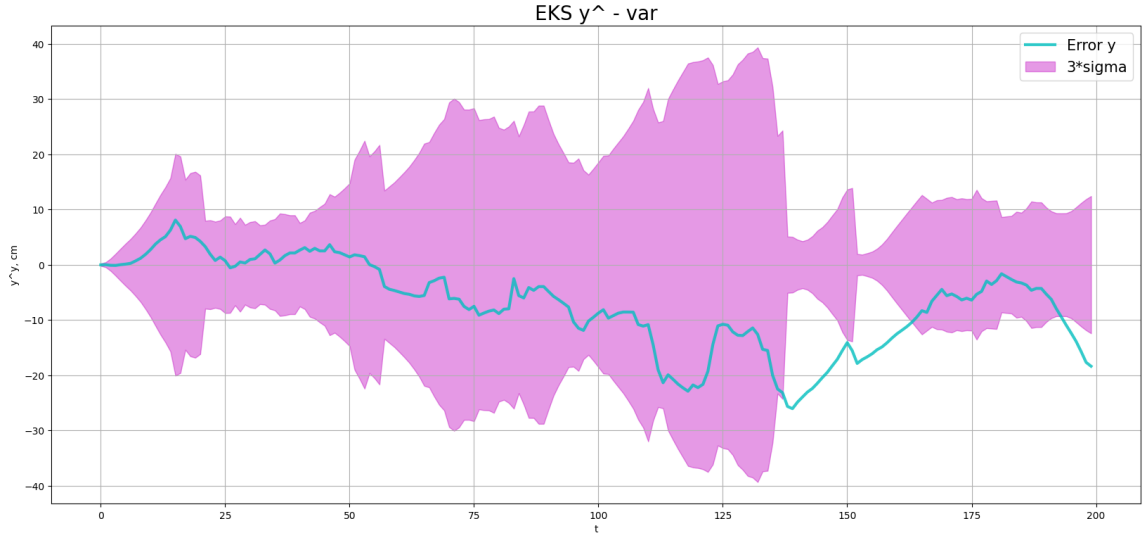


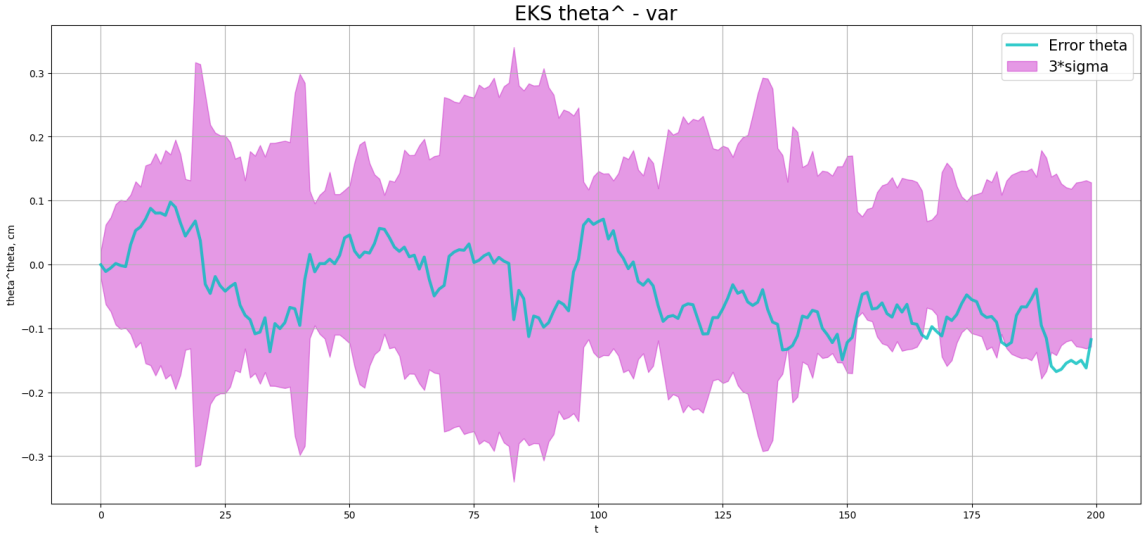




In [19]:
```
#numb = 10
plt.figure(figsize=[20,30])
plot(np.load('Task_D/Exp_2/set_2/input_data.npy'),np.load('Task_D/Exp_2/set_
plot(np.load('Task_D/Exp_2/set_2/input_data.npy'),np.load('Task_D/Exp_2/set_
plot(np.load('Task_D/Exp_2/set_2/input_data.npy'),np.load('Task_D/Exp_2/set_
```



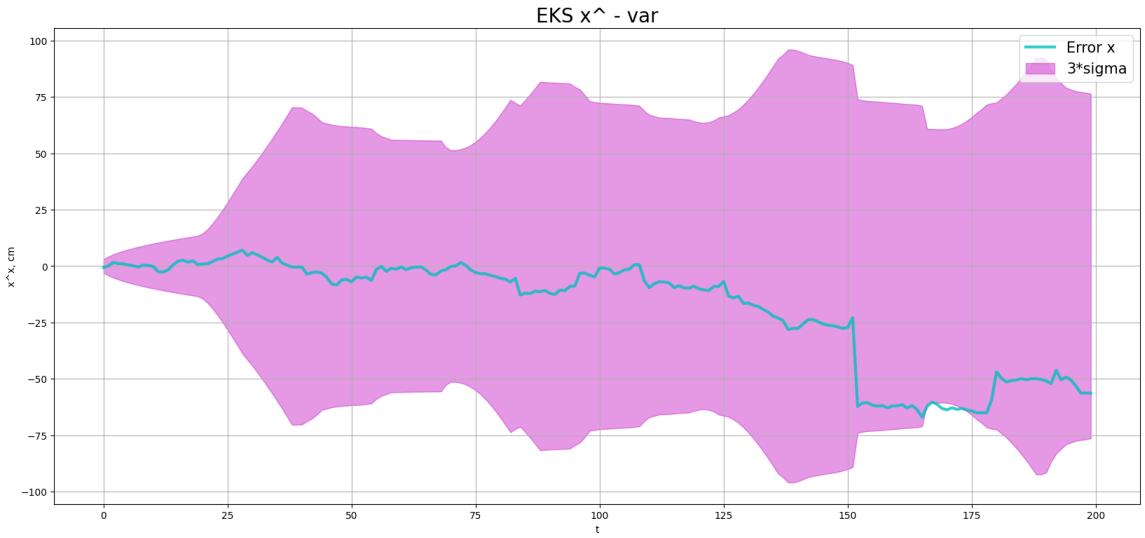


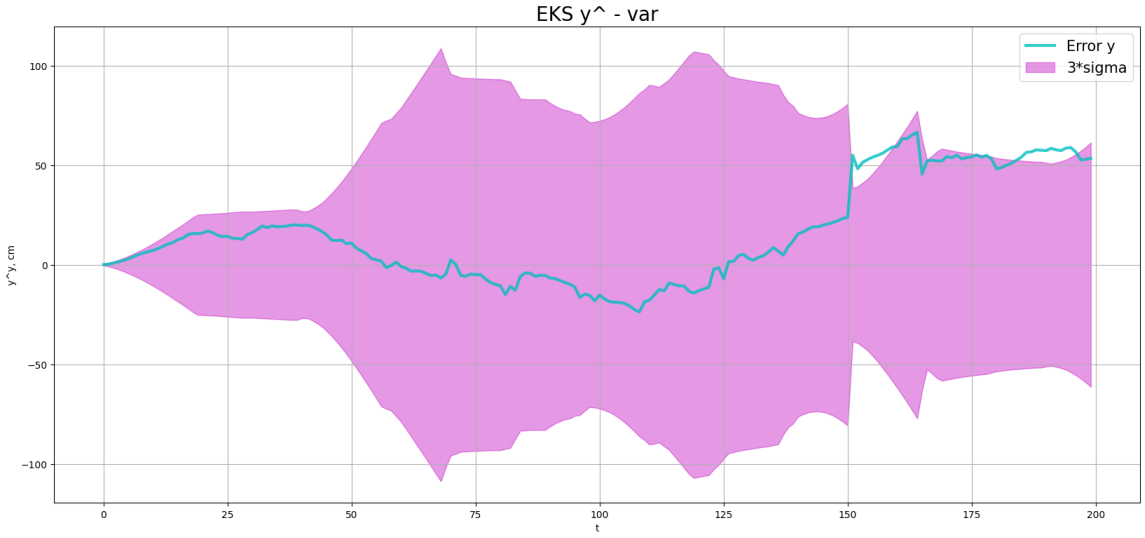


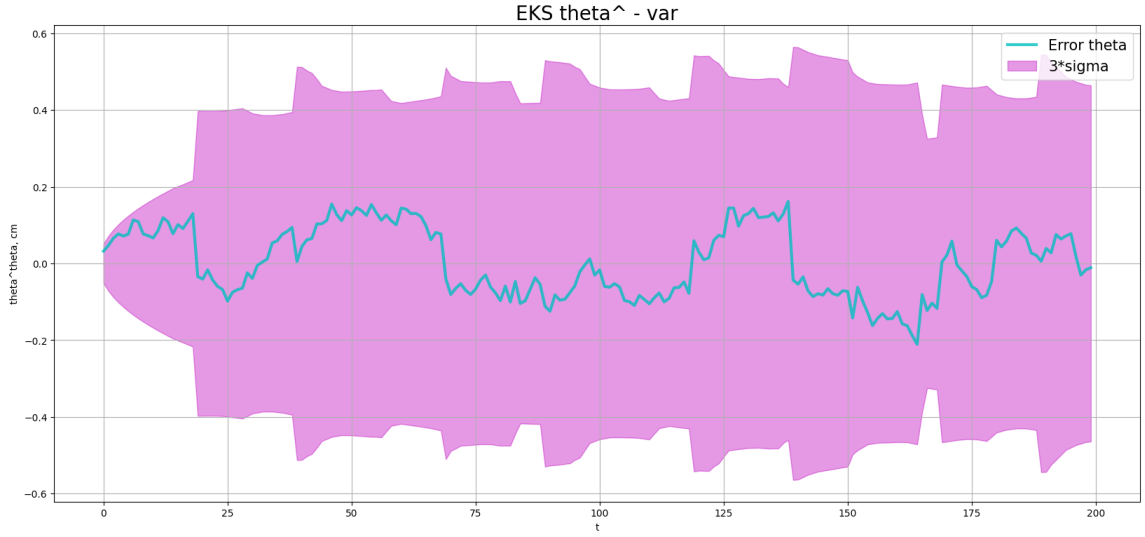
Conclution: when the number of particles is extremely low, we cannot provide a reliable estimation, as the distribution of particles provides us with incorrect localisation information.

How does EKF behaves when we underestimate or overestimate filter parameters Q and R?

In [20]:
```
#Overestimated
plt.figure(figsize=[20,30])
plot(np.load('Task_D/Exp_3/M_N/input_data.npy'),np.load('Task_D/Exp_3/M_N/ou
plot(np.load('Task_D/Exp_3/M_N/input_data.npy'),np.load('Task_D/Exp_3/M_N/ou
plot(np.load('Task_D/Exp_3/M_N/input_data.npy'),np.load('Task_D/Exp_3/M_N/ou
```

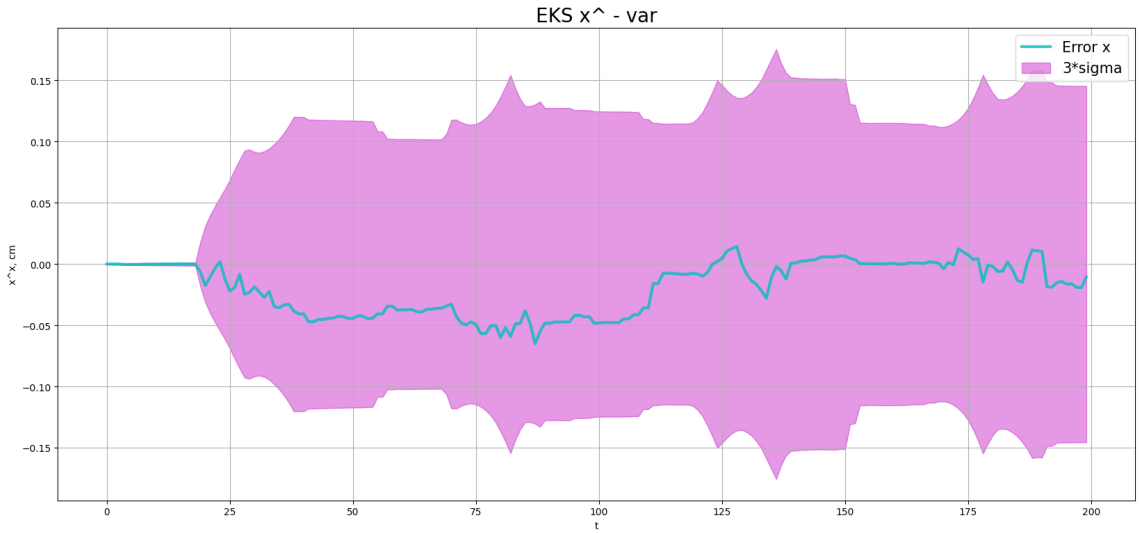


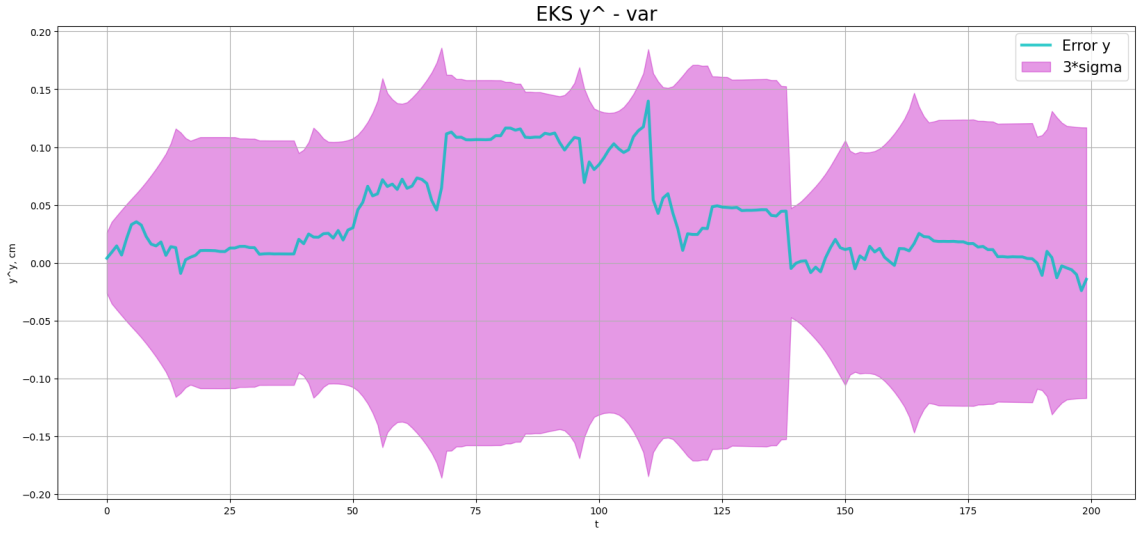


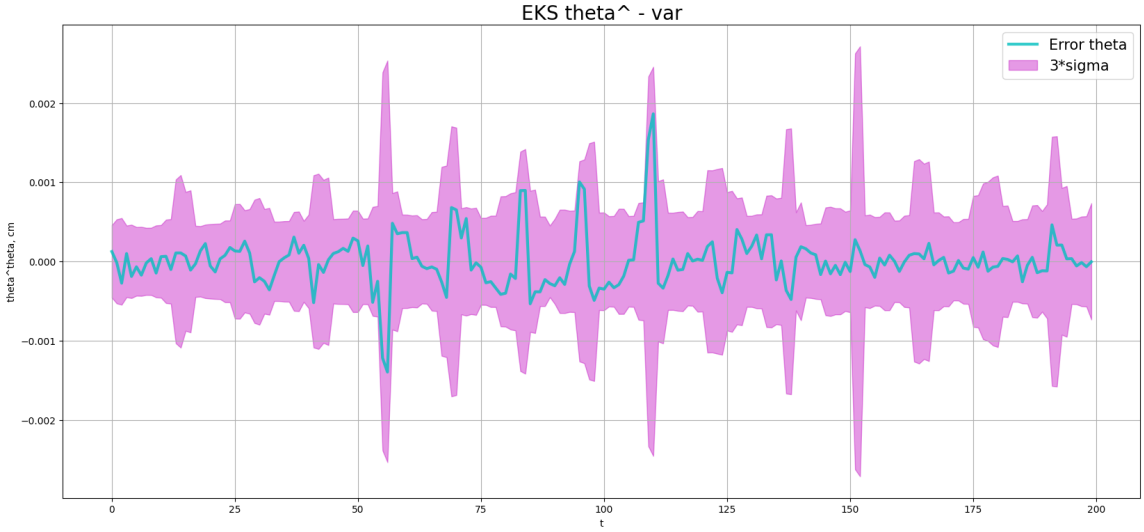


Type *Markdown* and LaTeX: α^2

In [21]:
```
#Underestimate
plt.figure(figsize=[20,30])
plot(np.load('Task_D/Exp_3/O_N/input_data.npy'),np.load('Task_D/Exp_3/O_N/ou
plot(np.load('Task_D/Exp_3/O_N/input_data.npy'),np.load('Task_D/Exp_3/O_N/ou
plot(np.load('Task_D/Exp_3/O_N/input_data.npy'),np.load('Task_D/Exp_3/O_N/ou
```







Conclutions: Underestimating or overestimating noise affects the performance of the extended Kalman filter. In the case of overestimation, we get a slightly higher convergence rate of the filter characteristics and a wider range of possible estimated values. It can be said that in practice, overestimated values are better than underestimated values.

In [ ]: