# Filtro de Kalman en ROS 2
# Practica 2


## Manuel Escobar Neupavert


### Modificaciones de código

1. motion_models.py:

Se han completado las matrices A y B de cada filtro.

Para el primero:

$$A = I_{3\times 3}, \quad B = \begin{bmatrix} \cos(\theta)\Delta t & 0 \\ \sin(\theta)\Delta t & 0 \\ 0 & \Delta t \end{bmatrix}$$

En el código:

```
A = np.eye(3)

B = np.array([
[np.cos(theta)*delta_t, 0],
[np.sin(theta)*delta_t, 0],
[0, delta_t]
])
```

Para el segundo:

$$A = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad B = 0$$

En el código:

```
A = return np.array([
[1, 0, 0, 1, 0, 0],
[0, 1, 0, 0, 1, 0],
[0, 0, 1, 0, 0, 1],
[0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 1]
])


B = np.zeros(6,2)
```

Se observa que Δt vale 1. En estos ejercicios siempre tomará ese valor.

## 2. observationmodels.py

Las matrices de observación serán las siguientes:

```
C1 = np.eye(3)

C2 = np.eye(6)
```

En el primer caso se considera que x, y, θ se pueden observar. En el segundo caso también se pueden observar $v_x$, $v_y$ y ω;

## 3. kf_estimation.py

Parte del código se utiliza para generar las gráficas del siguiente apartado. Dado que no está directamente relacionado con los filtros de Kalman, no se explicará aquí.

Primero se han inicializado el estado y covarianza iniciales:

```
initial_state = np.zeros((3,1))
initial_covariance = np.eye(3) * 0.1
```

Posteriormente, en *odom_callback* se ha obtenido la posición y orientación del robot de los mensajes correspondientes. La orientación venía dada en cuaterniones, se ha obtenido la guiñada del robot usando la función *euler_from_quaternion*.

Una vez obtenidas posición y orientación se ejecutan las etapas de predicción y actualización del friltro de Kalman.

Por último, se publican los resultados en el tópico correspondiente. Nótese que la guiñada del robot se ha expresado en cuaterniones usando la función *quaternion_from_euler*.

4. kf_estimation_vel.py

El código de este archivo es similar al del anterior. En este caso, en *odom_callback* también se han obtenido la velocidad lineal del robot, de la que posteriormente se han obtenido las componentes x e y.

5. kalman_filter.py

Haciendo uso de las expresiones correspondientes, se han implementado las etapas de predicción y actualización de ambos filtros de kalman. En general, lo único que se ha hecho es traducir esas expresiones a Python haciendo uso de la librería Numpy.

## Cómo ejecutar los nodos:

Para ejecutar los nodos bastará con usar uno de los siguientes comandos:

```
ros2 run p2_kf_adr kf_estimation_
ros2 run p2_kf_adr kf_estimation_vel
```

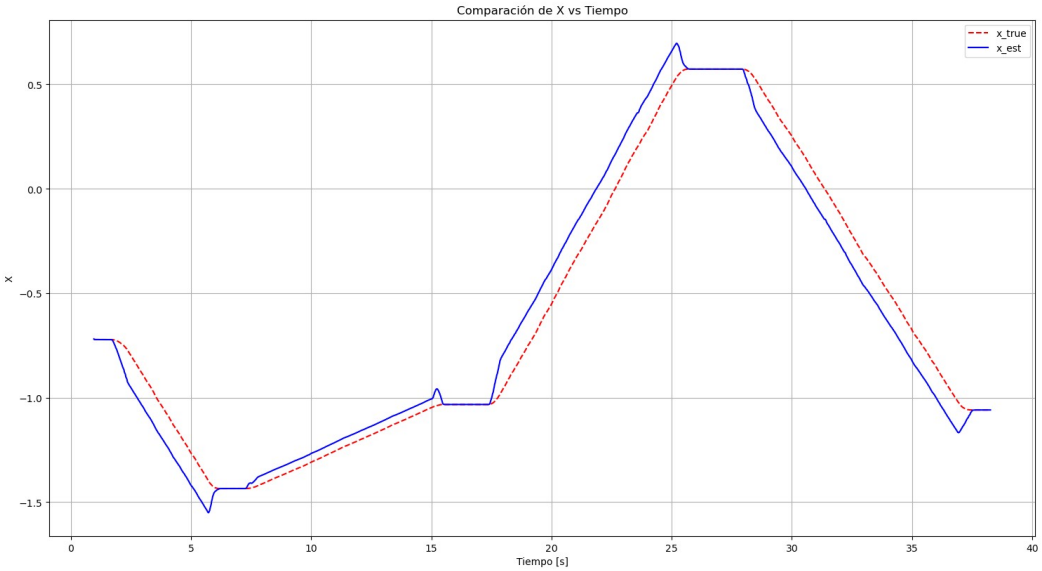
Para ver las gráficas realice una simulación y, tras cerrarla, ejecute *graphs.py.* Se mostrarán los datos que se han ido recopilando durante la simulación.

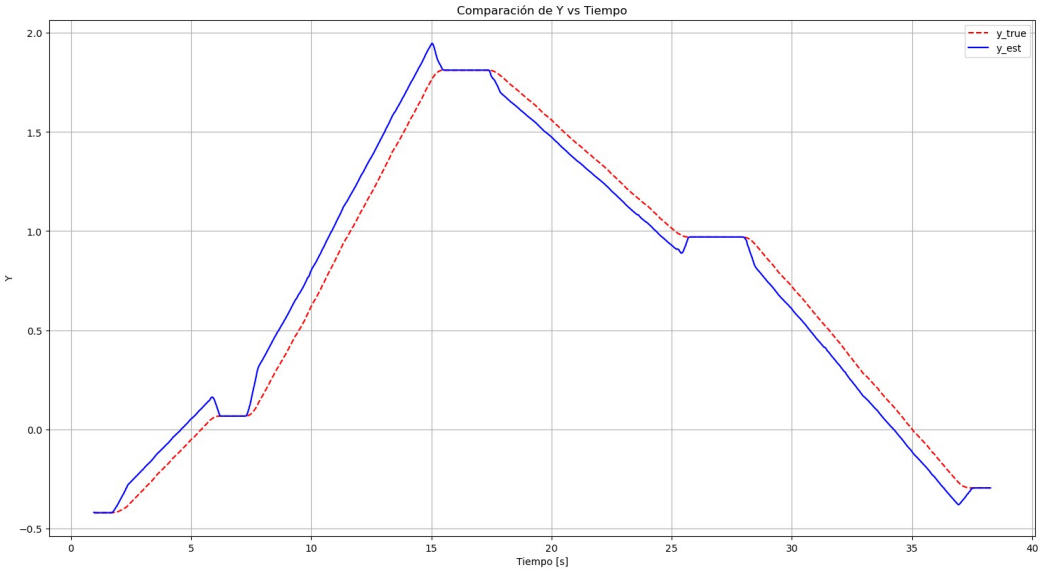
## Gráficas obtenidas

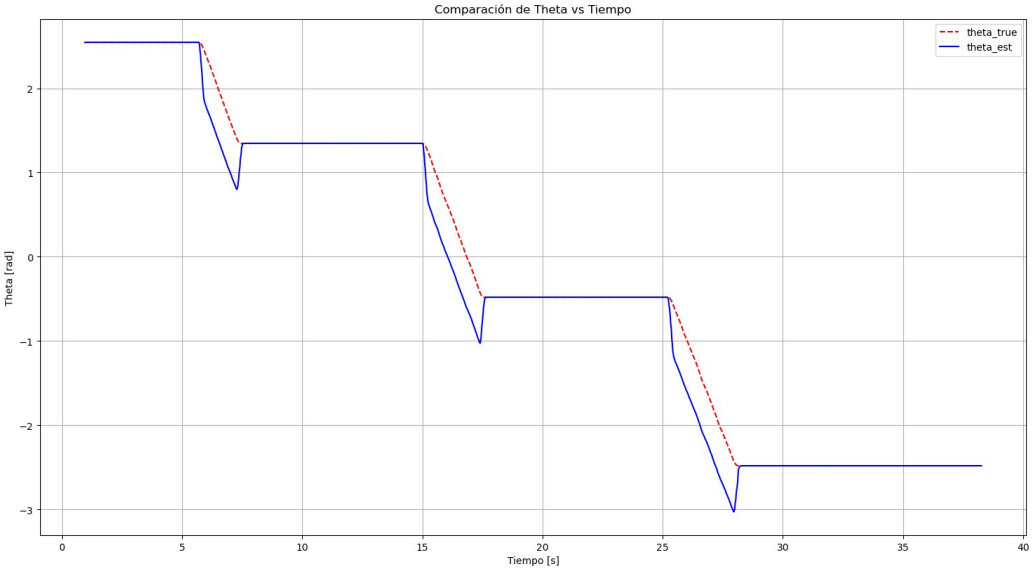
### Modelo simplificado

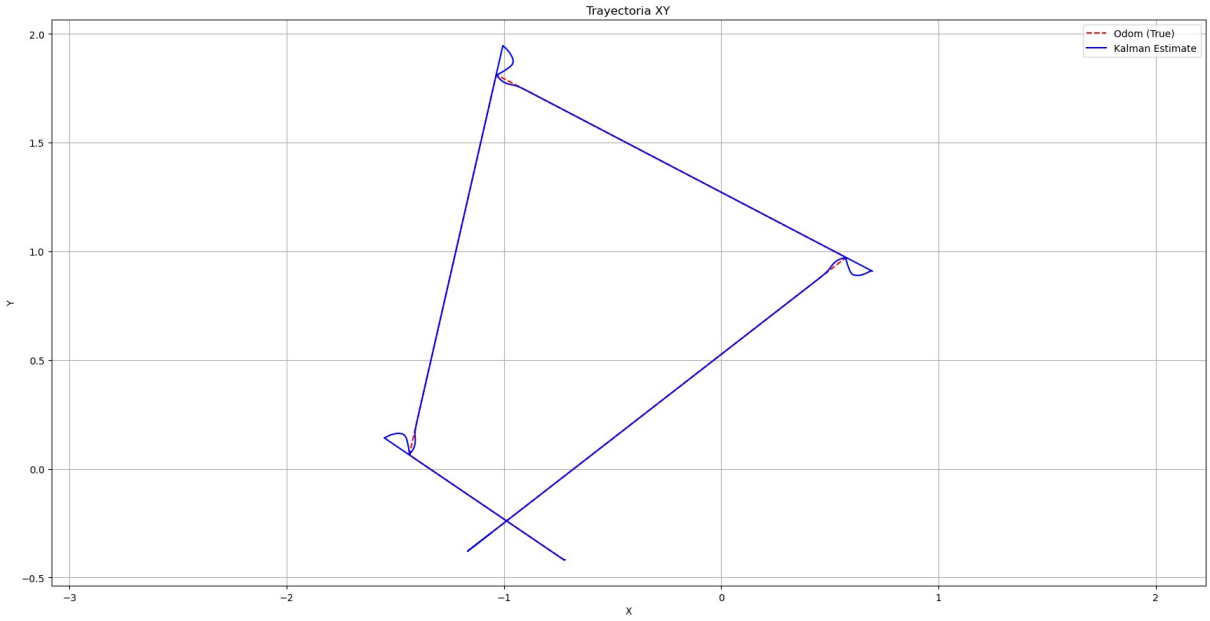
Para este primer modelo se muestran las gráficas de x, y, theta y la trayectoria.

1. Ruido bajo:

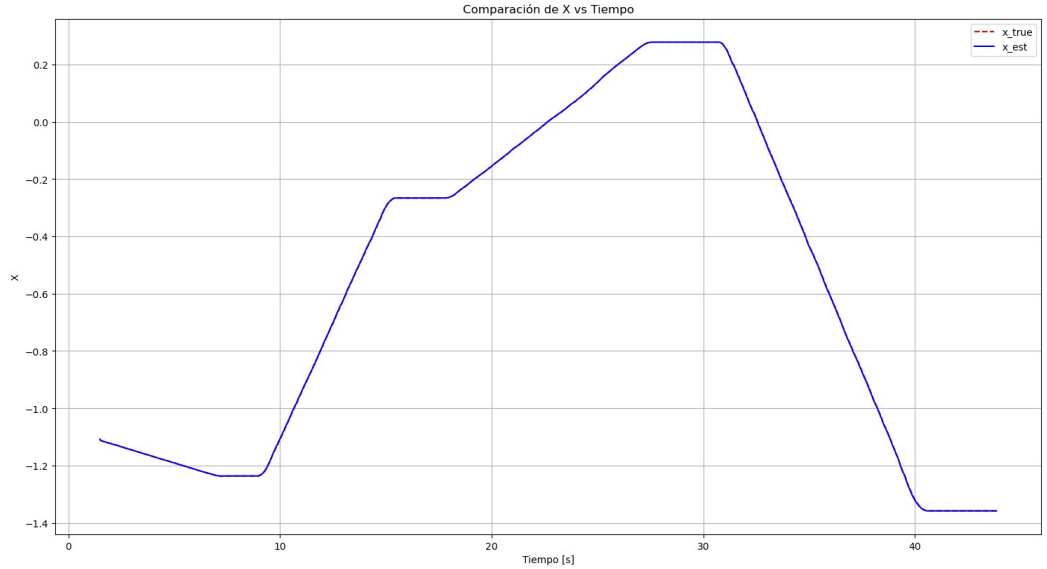


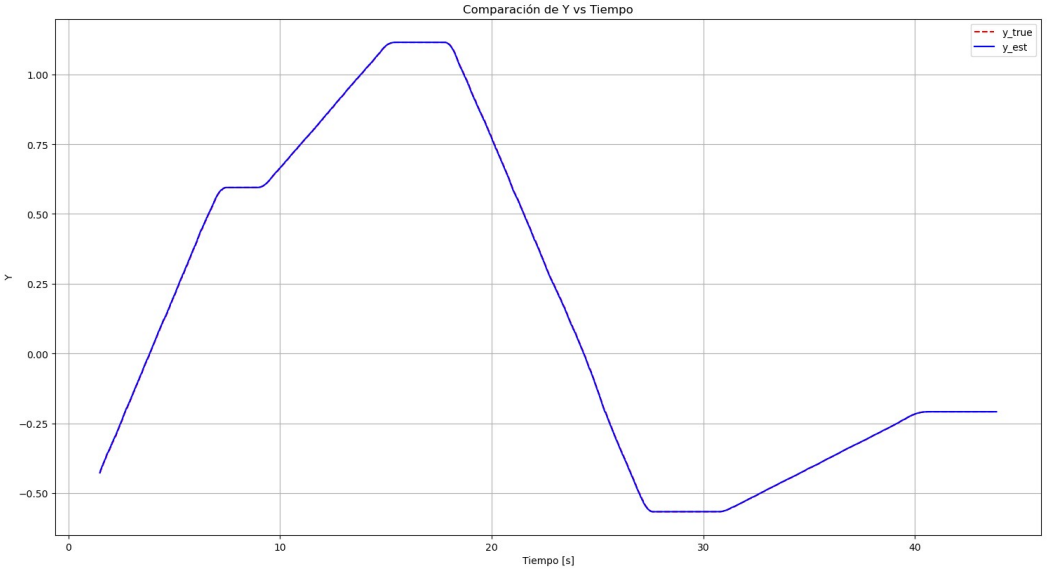



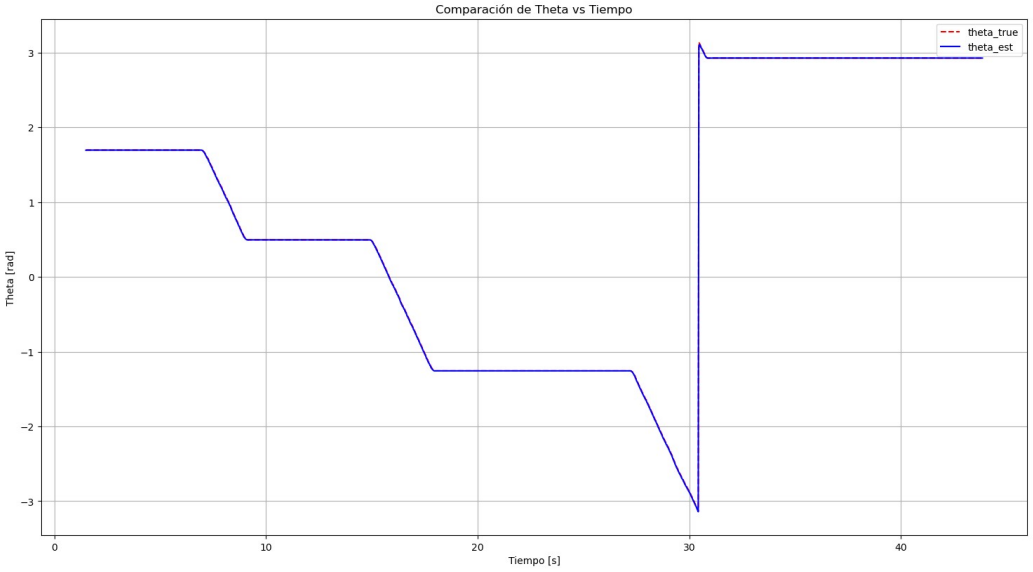

Comparación de Theta vs Tiempo
theta_true
theta_est
Theta [rad]
Tiempo [s]

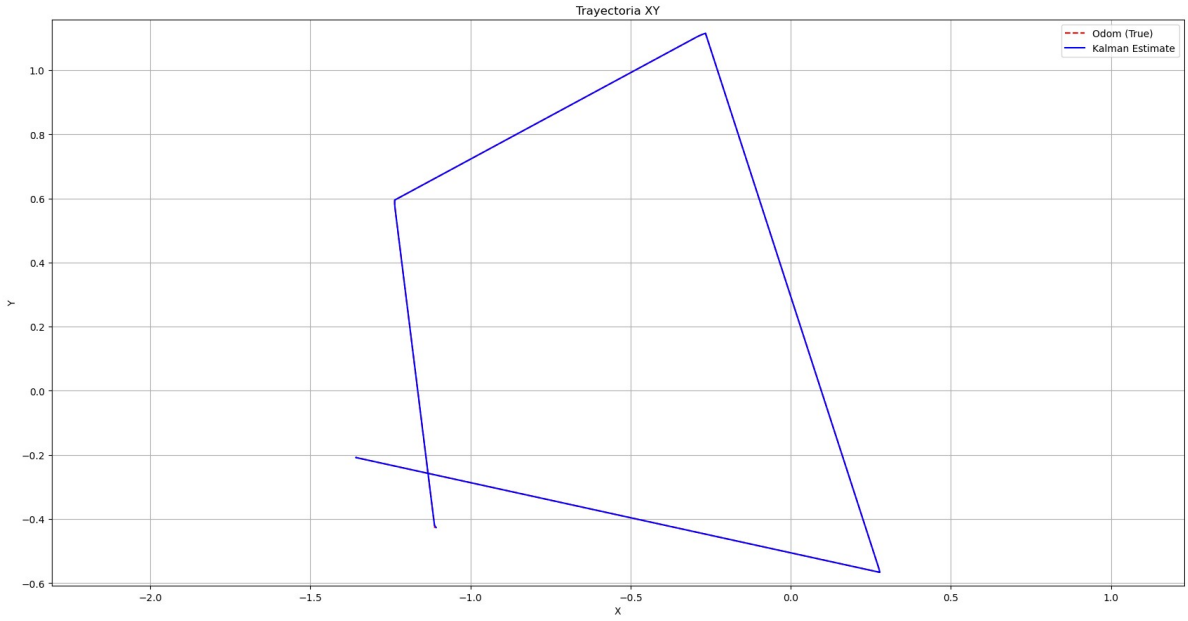

Trayectoria XY
Odom (True)
Kalman Estimate
X
Y

2. Ruido alto en el proceso:





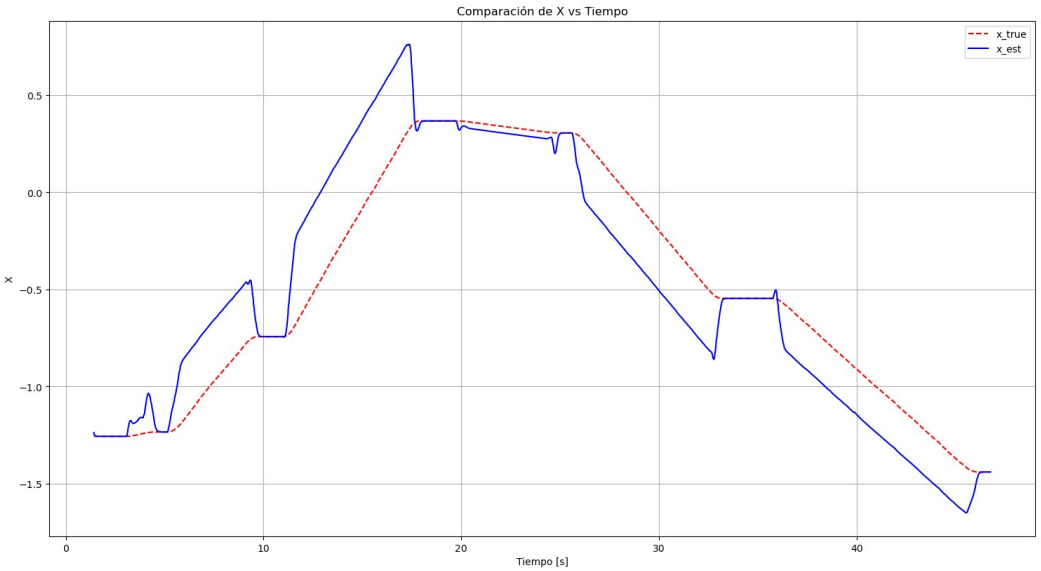



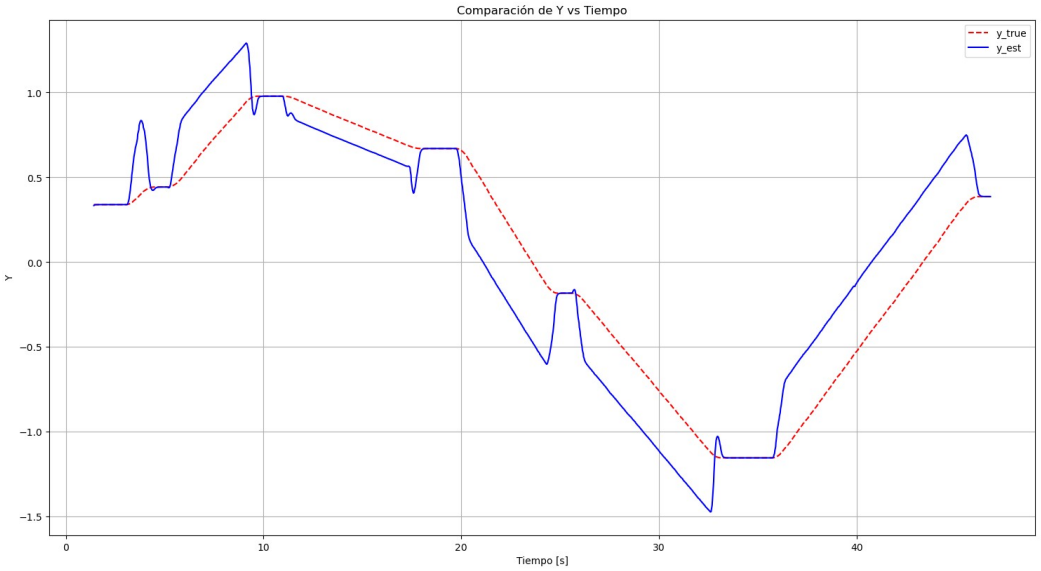

Se ha usado este ruido:

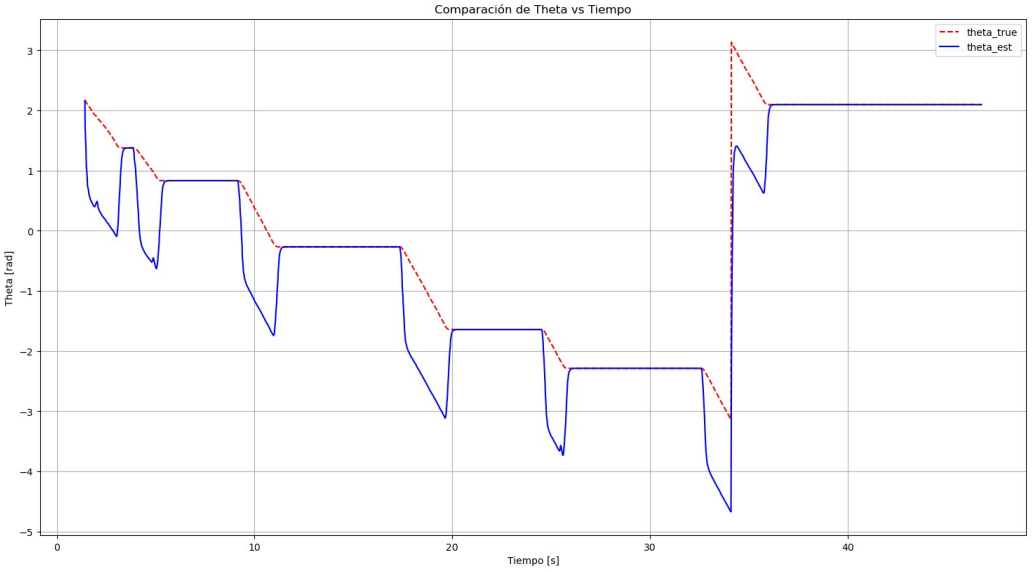
```
proc_noise_std = [0.2, 0.2, 0.1]
```

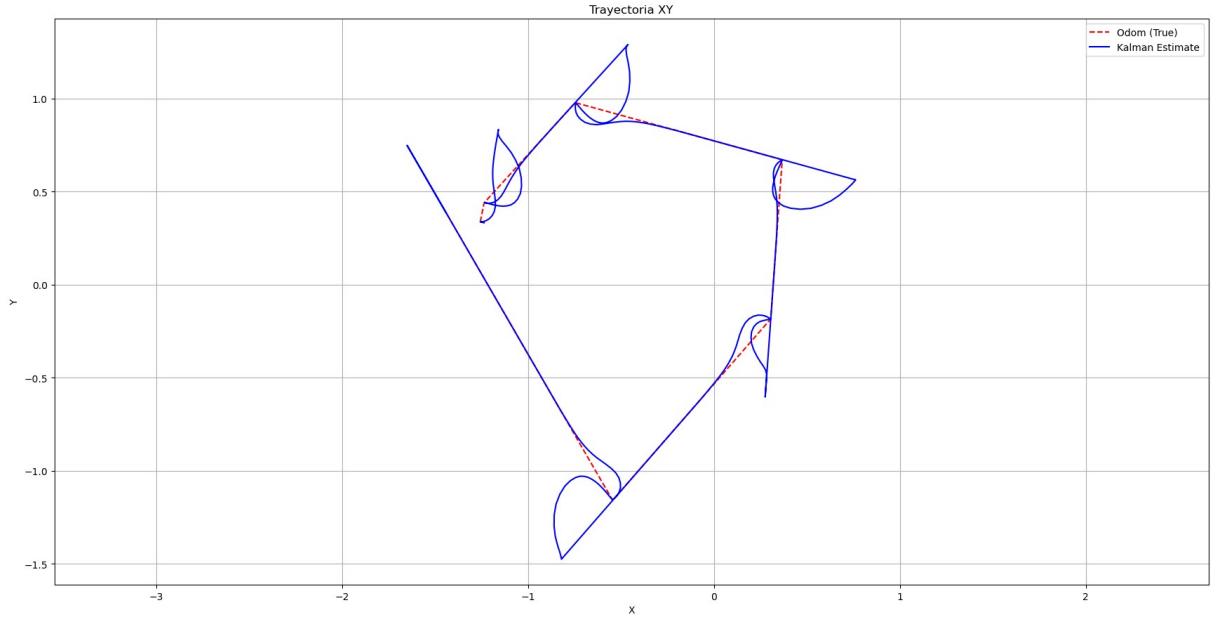
En este caso se da más peso a las observaciones, por lo que la estimación las seguirá más de cerca.

3. Ruido alto en la medición:






Se ha usado este ruido:

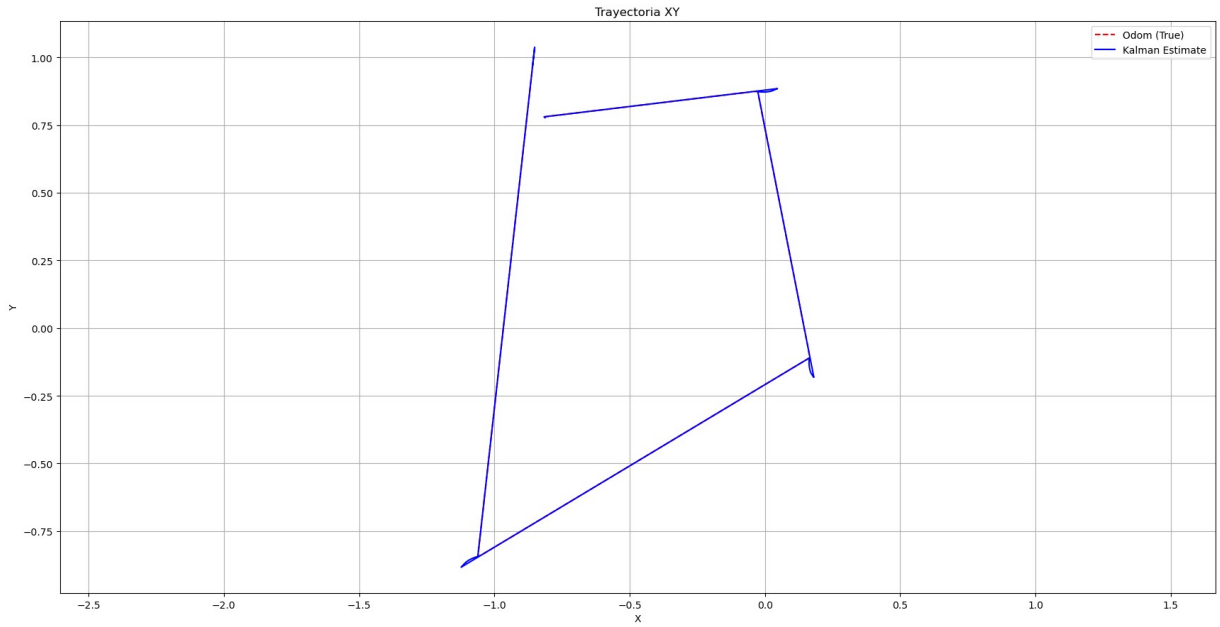
```
proc_noise_std = [0.04, 0.04, 0.02]
```

En este caso las mediciones son menos fiables, por lo que la predicción no las seguirá tan de cerca.

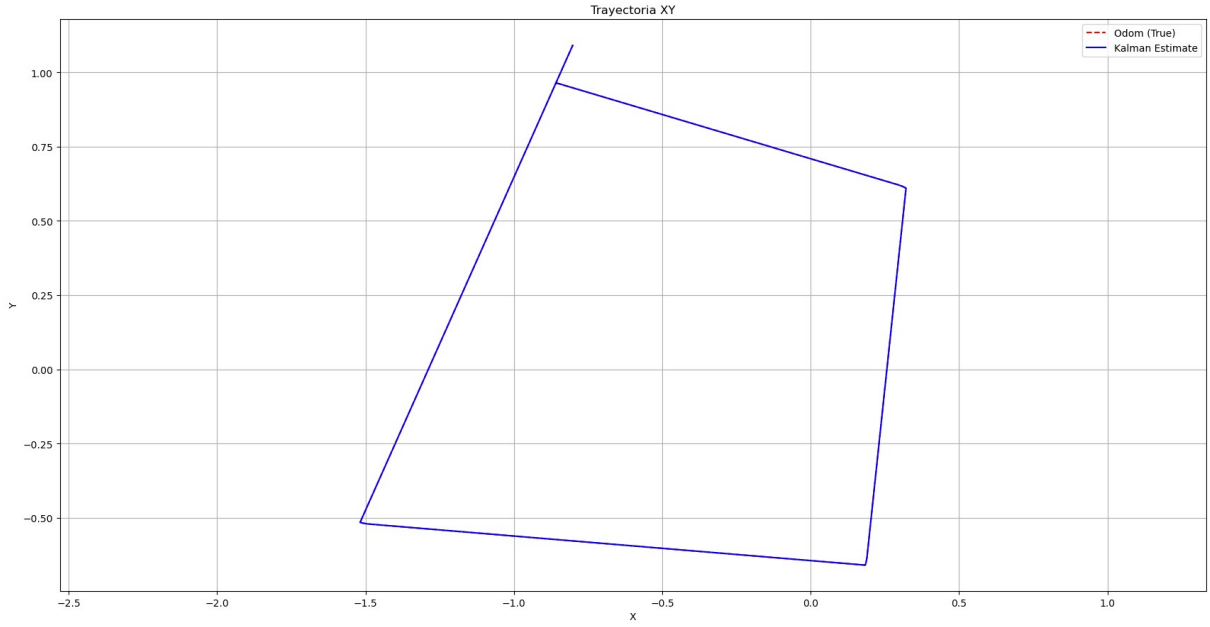
## Modelo simplificado

Para este segundo modelo solo se mostrarán las trayectorias, todas las gráficas serán análogas a las del apartado anterior.

1. Ruido bajo:




2. Ruido alto en el proceso:

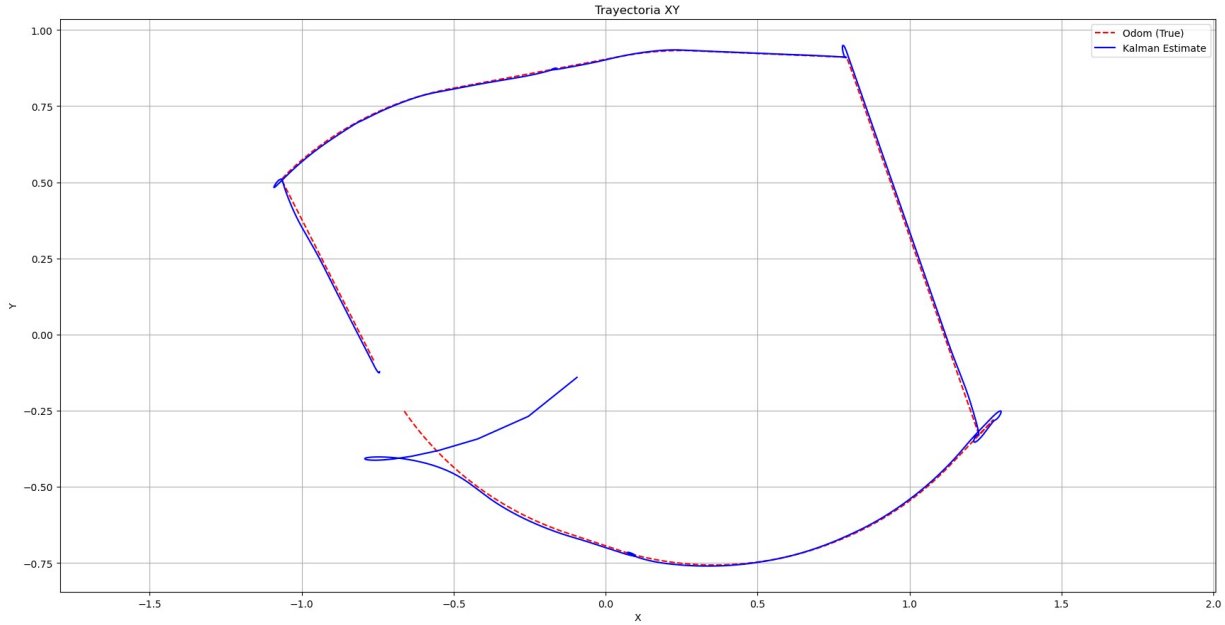



Las predicciones seguirán más de cerca a las mediciones. Véanse las esquinas del cuadrilátero.

Se ha usado este ruido:

```
proc_noise_std = [0.2]*6
```

3. Ruido alto en la medición:




Al ser las observaciones menos fiables, las predicciones no las seguirán tan de cerca.

Se ha usado este ruido:

```
proc_noise_std=[0.01]*6
obs_noise_std=[1, 0.5, 0.5, 1, 0.5, 1]
```