Universidad de San Andrés, Depto. de Ingeniería

Práctica 4 - Filtros EKF

I-402 - Principios de la Robótica Autónoma

Prof. Dr. Ignacio Mas, Tadeo Casiraghi y Bautista Chasco

22 de septiembre de 2025

Fecha límite de entrega: 17/10/25, 23:59hs.
Modo de entrega: Enviar por el Aula Virtual del Campus el código comentado y los gráficos (.jpg ó .pdf) y/o animaciones.

1. Notas preliminares

Antes de poder comenzar con el trabajo práctico deberán haber completado las notas preliminares del tp anterior. También deberán agregar lo siguiente al paquete base provisto por la cátedra:

Paquete de custom_code

1. **setup.py**

   Al archivo de setup.py agregar dos entry points nuevos:

   a) `"ekf = custom_code.my_ekf:main"`

   b) `"features = custom_code.features:main"`

   c) `"feature_finder = custom_code.feature_finder:main"`

2. **rviz**

   Agregar el archivo ***ekf.rviz*** al directorio de rviz

3. **launch**

   Agregar el archivo `launch_my_ekf.launch.py` al directorio de launch

4. **package.xml**

   Agregar `<exec_depend>custom_msgs</exec_depend>` a las dependencias.

5. **código custom**

   Agregue los siguientes archivos junto con los otros códigos de python:

   - `my_ekf.py`
   - `features.py`
   - `feature_finder.py`

Paquete de custom_msgs
Agruegue el paquete de mensajes custom a su src.

2. Proceso de lanzamiento de nodos

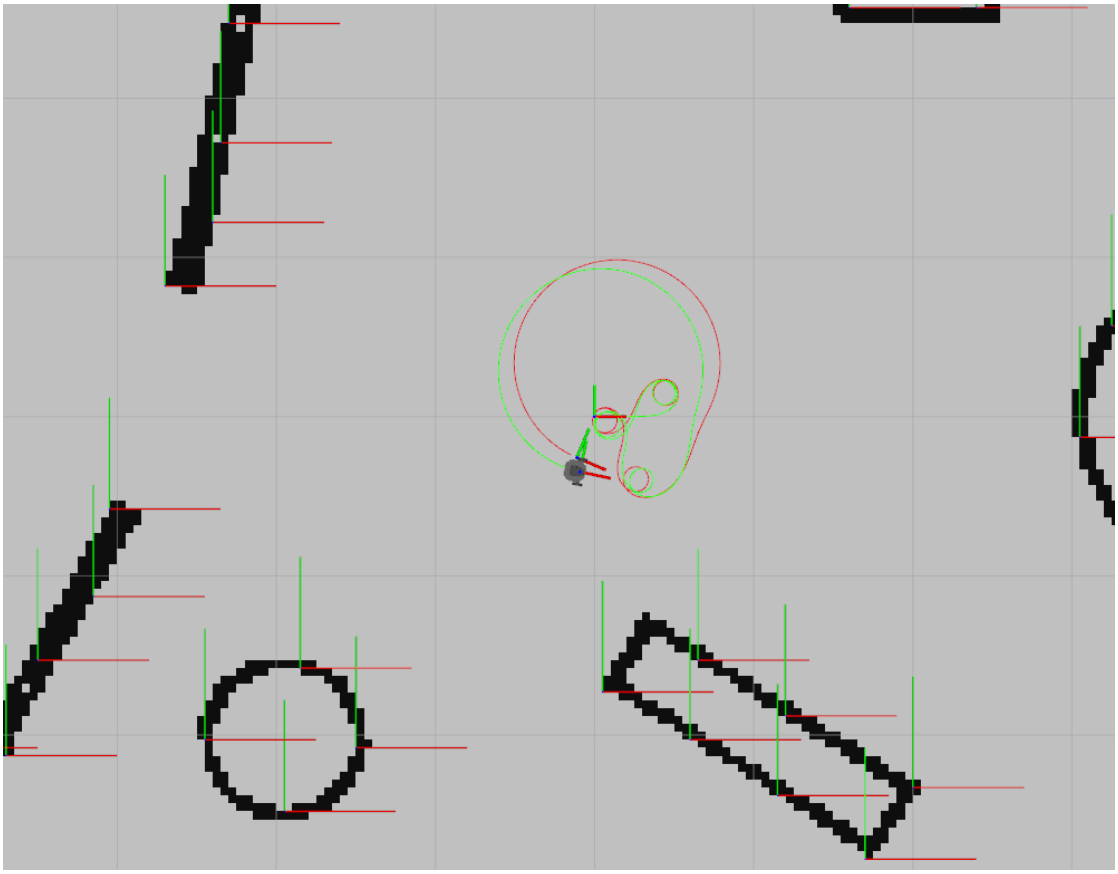
Para correr todo el código debemos tener 5 terminales (no se olviden del colcon build y el source install/setup.bash). En primer instancia, utilizaremos 3 (hay que respetar este orden de lanzamiento):

- Terminal 1: `ros2 launch custom_code launch_my_ekf.launch.py`
- Terminal 2: `ros2 launch turtlebot3_custom_simulation custom_room.launch.py`
- Terminal 3: `ros2 run turtlebot3_teleop teleop_keyboard`

Si siguieron todos los pasos correctamente verán lo siguiente en Rviz2:

Notarán que:

- Tenemos el mapa
- Estan marcados en el mapa las posiciones de los landmarks
- Si nos movemos con el keyboard se marcan los caminos del robor real, el robot calculado, pero falta el robot corregido con ekf.



3. Prediccion por EKF

Ahora tendrán que implementar un paso de predicción por EKF. Para eso tendrán que armar su propio nodo (puede ser un nuevo paquete incluso) que deberá hacer lo siguiente:

1. Importar el tipo de mensaje **Belief**. Este tipo está definido como:
   - `geometry_msgs/Pose2D mu`
     - `float64 x`
     - `float64 y`
     - `float64 theta`
   - `float64[9] covariance`

   Si hacen un nuevo paquete no se olviden de agregar `custom_msgs` a `package.xml`.
2. Importar el tipo de mensaje **DeltaOdom**. Este tipo está definido como:
   - `float64 dr1`
   - `float64 dr2`
   - `float64 dt`

   Si hacen un nuevo paquete no se olviden de agregar `custom_msgs` a `package.xml`.
3. Subscribirse al tópico `/belief` de tipo de dato **Belief**. Así recibirán actualizaciones del belief ya sean la inicial o las futuras correcciones del ekf.
4. Crear un publisher del tópico `/belief` de tipo de dato **Belief**. Con eso podrán publichar las predicciónes del belief.
5. Subscribirse al tópico `/delta` de tipo de dato **DeltaOdom**. Así recibirán actualizaciones de la odometria en formato deltas.

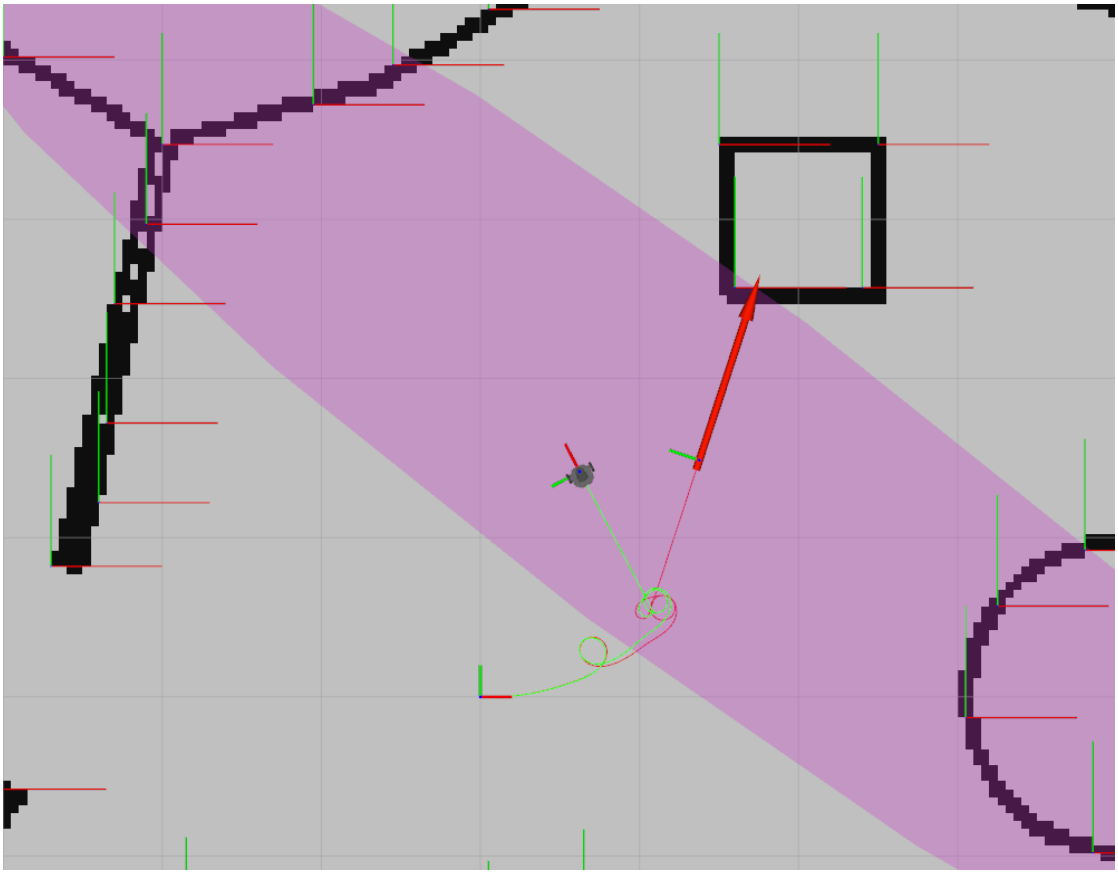
Este nodo no debe inicializar el belief. Cuando se corre el launch file de my_ekf se publica el belief en tiempo 0 durante la inicialización. Su nodo ya deberá estar corriendo para que pueda capturar este belief inicial. Esto quiere decir que no debe hacer ninguna predicción hasta no haber recibido el belief inicial.
El nodo deberá mantener siempre un valor de belief actualizado (a futuro la corrección cambiará el valor del belief).
Por último, este nodo deberá hacer el paso de predicción del belief en base a la teoría del EKF. Para eso tendrá que:

1. Calcular la matriz jacobiana G_t de la función de movimiento g sin ruido.
2. Implementar el paso de predicción usando el jacobiano G_t calculado en el punto anterior. Asumir que el ruido del modelo de movimiento esta definido por:

$$Q_t = \begin{pmatrix} 0{,}02 & 0 & 0 \\ 0 & 0{,}02 & 0 \\ 0 & 0 & 0{,}02 \end{pmatrix}$$

3. Publicar el belief actualizado

Si hizo todo correctamente y corre ahora el nodo de predicción en una cuarta terminal (recuerde correr primero este nodo para que pueda recibir el belief inicial), deberá ver lo siguiente:



La flecha roja indica la posición del belief calculado con la predicción, y el óvalo violeta representa la covarianza (se podría hacer un análogo a que es el área donde podría estar el robot real dado mis incertidumbres). Como pueden observar, el belief siguie a la posición calculada en base a odometría imperfecta, y la covarianza es muy grande.

4. Paso de corrección EKF

Ahora tendrán que implementar un paso de corrección por EKF. Para eso tendrán que armar su propio nodo (puede ser un nuevo paquete) que deberá hacer lo siguiente:

1. Importar el tipo de mensaje **Belief**. Este tipo está definido como: Si hacen un nuevo paquete no se olviden de agregar `custom_msgs` a `package.xml`.
2. Subscribirse al tópico `/belief` de tipo de dato **Belief**. Así recibirán actualizaciones del belief ya sean la inicial o las futuras predicciones.
3. Crear un publisher del tópico `/belief` de tipo de dato **Belief**. Con eso podrán publichar las correcciones del belief.
4. Subscribirse al tópico `/landmarks` de tipo de dato **PoseArray**. Así recibirán el dato de la posición de todos los landmarks. Esto se publica una única vez durante la inicialización de my_ekf, por lo que debe correr su nodo primero para que pueda capturarlo.
5. Subscribirse al tópico `/observed_landmaks` de tipo de dato **PoseArray**. Así recibirán el dato de rango y ángulo de los landmarks observados. Este array de `Pose2D` es de largo n, siendo n la cantidad total de landmarks. Cuando se detecta un landmark el valor tendrá el rango dentro del .x, y tendrá el ángulo en el .z. Si no se detecta un landmark todos los valores de esa pose serán 0. El órden de landmark detectado es el mismo que el del dato de lanmdarks.

Este nodo no debe inicializar el belief. Cuando se corre el launch file de my_ekf se publica el belief en tiempo 0 durante la inicialización. Su nodo ya deberá estar corriendo para que pueda capturar este belief inicial. Esto quiere decir que no debe hacer ninguna corrección hasta no haber recibido el belief inicial.
El nodo deberá mantener siempre un valor de belief actualizado, ya que la predicción modificará este valor.
Por último, este nodo deberá hacer el paso de corrección del belief en base a la teoría del EKF. Para eso tendrá que:

1. Calcular la matriz jacobiana H_t de la función de medición h en base a los landmarks.
2. Implementar el paso de corrección del filtro EKF en la función `correction_step` usando el jacobiano H_t. Asumir que el desvío estándar de la medición de rango es de 0.05 metros, y del ángulo 0.05 radianes.
3. Publicar el belief actualizado

Si hicieron todo correctamente, ahora puede lanzar primero el nodo de corrección, luego el de predicción, y finalmente las otras 3 terminales. Debería notar que el óvalo de covarianza es mucho menor o casi no visible, y que el belief ahora sigue a la posición real del robot.