Taller 10: Localización basada en EKF

Robótica Móvil

2^{do} cuatrimestre 2025

Introducción

En este taller se estudiará un método de localización basado en el Filtro de Kalman Extendido (EKF), tanto en la etapa de predicción como de actualización del filtro, para un robot móvil dotado de una unidad inercial (IMU), odómetros (*encoders*) y un LiDAR.

Bajo el esquema de EKF, para la fase de predicción, el robot utilizará la información de velocidad lineal v provista por los encoders y la de velocidad angular ω provista por el giróscopo incluido en la unidad inercial. El modelo de proceso a utilizar es el del vehículo diferencial visto hasta ahora en la materia. La fase de actualización será resuelta mediante la detección de una serie de postes ubicados en distintos lugares del ambiente, que serán los *landmarks* del sistema.

En este taller se deberá completar el esqueleto de código provisto, correspondiente al nodo principal de localización. A su vez, este nodo ya resuelve:

- Lectura de los sensados de odometría, de la unidad inercial y detección de *landmarks*.
- Invocación del módulo de EKF para ejecutar el modelo de movimiento (proceso) y modelo de sensado (actualización).
- Publicación de la pose estimada en términos de media y covarianza.

El módulo EKF mencionado encapsula la implementación del filtro en su totalidad, realizando las operaciones necesarias para resolver la fase de predicción y actualización del filtro. Lo que deberá ser completado por el alumno son los modelos correspondientes que definen al Filtro de Kalman Extendido en el problema particular de la localización de un robot diferencial como fue descrita.

Para ello, se deberán implementar una serie de métodos que deben resolver el cálculo de los distintos modelos, jacobianos y matrices de covarianza asociados. Tener en cuenta que, debido a que el Filtro de Kalman Extendido está implementado en forma matricial, todos estos modelos serán siempre matrices o vectores.

Modelo de movimiento para la predicción y actualización

Consideramos el modelo de estados de un robot diferencial para la predicción del movimiento:

$$\mathrm{x_t} = f(\mathrm{x_{t-1}}, \mathrm{u_t}, \mathrm{w_t}) = g(\mathrm{x_{t-1}}, \mathrm{u_t}) + \mathcal{N}(0, Q_t) =$$
$$\begin{pmatrix} x_{t-1} + v_t \Delta t \cos(\theta_t) \\ y_{t-1} + v_t \Delta t \sin(\theta_t) \\ \theta_{t-1} + \omega_t \Delta t \end{pmatrix} + \begin{pmatrix} w_{t,x} \\ w_{t,y} \\ w_{t,\theta} \end{pmatrix} = \begin{pmatrix} x_{t-1} + v_t \Delta t \cos(\theta_t) + w_{t,x} \\ y_{t-1} + v_t \Delta t \sin(\theta_t) + w_{t,y} \\ \theta_{t-1} + \omega_t \Delta t + w_{t,\theta} \end{pmatrix}$$

donde $\mathrm{x_t}$ es el vector de estados en el tiempo t, $\mathrm{u_t}$ es la entrada de control en el tiempo t, y Q_t covarianza del error asociado a las incertidumbres del modelo:

$$\mathrm{x_t} = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} \qquad \mathrm{u_t} = \begin{pmatrix} v_t \\ \omega_t \end{pmatrix} \qquad Q_t = \begin{pmatrix} \sigma^2_{w_{t,x}} & 0 & 0 \\ 0 & \sigma^2_{w_{t,y}} & 0 \\ 0 & 0 & \sigma^2_{w_{t,\theta}} \end{pmatrix}$$

Se desea obtener un primer método de localización puramente predictivo, es decir, sin ejecutar la fase de actualización del EKF. Para ésto utilizaremos los jacobianos correspondientes:

$$A_t = \frac{\partial f(\mathrm{x_t}_{-1}, \mathrm{u_t}, \mathrm{w_t})}{\partial \mathrm{x_t}_{-1}} \qquad W_t = \frac{\partial f(\mathrm{x_t}_{-1}, \mathrm{u_t}, \mathrm{w_t})}{\partial \mathrm{w_t}}$$

Consideramos ahora el modelo de medición utilizado para actualizar los estados del robot diferencial. La actualización que consideraremos estará dada por cada estimación de la posición de cada landmark sensado:

$$\mathrm{z_{i,t}} = h(\mathrm{x_t}, m_{i,t}, \mathrm{v_{t,i}}) = \begin{pmatrix} r_{i,t} \\ \phi_{i,t} \end{pmatrix} + \mathcal{N}(0, R_t) = \begin{pmatrix} \sqrt{\Delta x^2 + \Delta y^2} \\ \arctan(\Delta x, \Delta y) - \theta_t \end{pmatrix} + \begin{pmatrix} v_{t,r,i} \\ v_{t,\phi,i} \end{pmatrix}$$

donde

$$\Delta x = m_{i,t,x} - x_t \quad ; \qquad \Delta y = m_{i,t,y} - y_t \quad ; \qquad m_{i,t} = (m_{i,t,x}, m_{i,t,y})$$

y la covarianza del ruido de medición está dada por:

$$R_t = \begin{pmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\phi^2 \end{pmatrix}$$

Los jacobianos correspondientes serán:

$$H_t = \frac{\partial h(\mathrm{x_t}, m_{i,t}, \mathrm{v_{t,i}})}{\partial \mathrm{x_t}} \qquad V_t = \frac{\partial h(\mathrm{x_t}, m_{i,t}, \mathrm{v_{t,i}})}{\partial \mathrm{v_{t,i}}}$$

En `/robmovil_ekf/src/localizer_ekf.cpp` se encuentran los diferentes métodos que deberán completar con información del modelo. Ésto será utilizado por el módulo de EKF para hacer los cálculos de predicción y actualización de estado correspondientes.

Nota: El primer elemento de los vectores y matrices del módulo EKF comienzan en la posición 1.

Para este taller deberán utilizar la escena `/robmovil_ekf/coppeliaSim/laser_landmarks.ttt`. Realizar un *launch* mediante el comando:

```
$ ros2 launch robmovil_ekf robmovil_ekf.launch.py
```

Cargar además en Rviz2 la configuración provista en `/robmovil_ekf/launch/ekf.rviz`., donde se podrá ver una elipsoide indicativa de la covariancia de la posición estimada. Específicamente, se muestran los puntos alejados hasta un desvío estándar del valor medio estimado de la posición.

Ejercicio 1: Modelo de movimiento para la predicción

Para activar únicamente la etapa de predicción del filtro, sin considerar la integración de la información provista por el LiDAR, existe un parámetro de configuración en el archivo `.launch.py`, correspondiente al nodo `robmovil_ekf`:

```
parameters=[{'only_prediction': True}
```

Se pide, utilizando únicamente la etapa de predicción:

a) Implementar el modelo de movimiento calculando los jacobianos correspondientes. Para esto deberán completar (donde indican los comentarios) las funciones:

```
void makeBaseA()
void makeA()
void makeBaseW()
void makeProcess()
```

donde `makeProcess` refiere al modelo de proceso $g(\mathrm{x_t}_{-1}, \mathrm{u_t})$ (es decir, la corrección de la posición sin considerar el ruido, que es desconocido para el robot); `makeA` y `makeBaseA` al Jacobiano de f respecto de $\mathrm{x_t}$ y `makeBaseW` al Jacobiano de f respecto de $\mathrm{w_t}$.

Notar que el ruido del modelo de movimiento se encuentra ya definido en:

```
void makeBaseQ()
```

donde `makeBaseQ` refiere a la covarianza de w_t.

b) Evaluar mediante una simulación en CoppeliaSim el sistema de localización obtenido, comparando la pose estimada con la real (*ground-truth*) en RViz2.

c) ¿Qué pasa con la covarianza? ¿Crece, decrece o se mantiene oscilando?

d) ¿Qué sucede si se cambia la fricción del piso? ¿Qué tan buena es la estimación obtenida?

Ejercicio 2: Estimar la posición de un poste en el mapa

Para este ejercicio y el siguiente, utilizar el parámetro `only_prediction` como `False`. De esta forma, incorporaremos la etapa de actualización a partir de las mediciones obtenidas con el LiDAR.

Toda medición recibida nos da información de algún poste que se encuentra cercano al robot.

Las nuevas mediciones ingresan al modelo por el método `localizer_ekf.cpp`:

```
bool set_measure(const Vector& new_measure_z)
```

Primero, al obtener una nueva medición, calcula la posición más probable en que se encuentra el poste dentro del mapa, mediante el método:

```
tf2::Vector3 measure2landmark(const Vector& measure)
```

Éste recibe una medición en coordenadas polares en referencia al robot y devuelve la posición estimada en coordenadas cartesianas del poste en referencia al mapa. Habiendo estimado la posición del poste en el mapa, lo siguiente que se requiere es definir cual es su verdadera posición. Para esto se requiere corresponder la posición estimada con alguno de los postes pertenecientes al mapa.

Nota: La correspondencia será exitosa siempre y cuando el sistema mantenga una estimación de localización "suficientemente" buena.

Completar el método:

```
bool find_corresponding_landmark(
          const tf2::Vector3& measured_landmark,
          tf2::Vector3& corresponding_landmark,
          float delta_radio)
```

Éste debe recibir las coordenadas cartesianas estimadas de un poste (en referencia al mapa) y devolver la posición *real* de dicho poste en el mapa, asignando la referencia `corresponding_landmark`. Para esto deberán "buscar" el poste del mapa que se encuentre dentro de un determinado radio tomando como centro la posición del poste estimada.

Pueden suponer que los postes se encuentran suficientemente separados.

Ejercicio 3: Modelo de sensado

Por último, habiendo establecido la posición donde se encuentra el poste en el mapa, se debe utilizar el modelo de sensado para predecir "la manera en que se debería haber observado" dicho poste. De esta manera el filtro de Kalman extendido será capaz de contrastar la medición recibida originalmente con la medición predicha y refinar los diferentes estimadores.

El modelo de sensado $z_{i,t} = h(x_t, m_{i,t}, v_{t,i})$ actualiza el filtro por medio del método:

```
void makeMeasure(void)
```

asignando a la variable global z la medición predicha. Para ello,

```
Vector landmark2measure(const tf2::Vector3& landmark)
```

recibe la posición de un poste en el mapa y devuelve la medición predicha en referencia al robot y en coordenadas polares.

a) Completar los jacobianos correspondientes para que la actualización por medición sea correctamente interpretada por el Filtro de Kalman Extendido:

```
void makeH(void)
void makeBaseV(void)
```

Donde `makeH` refiere al Jacobiano de h respecto de x_t y `makeV` al Jacobiano de f respecto de $v_{t,i}$.

b) Recorrer con el robot el entorno simulado en CoppeliaSim y analizar el funcionamiento del filtro.

Ejercicio 4: Análisis del modelo

El nodo `robmovil_ekf` tiene un parámetro de configuración llamado `min_landmark_size`, inicializado por defecto en 2 en el archivo `.launch.py`. Este parámetro se encarga de habilitar la etapa de corrección por sensado sólo si la cantidad de *landmarks* encontrados es mayor o igual al número indicado, utilizando sólo la etapa de predicción en caso contrario.

Modificar este valor (considerando que el número máximo de *landmarks* en la escena es 4) y recorrer con el robot el entorno simulado en CoppeliaSim. ¿Qué sucede al modificar este valor? ¿Qué ocurre con las estimaciones de la posición del robot si se deja de percibir este número mínimo de *landmarks*?

Por último, argumentar qué ocurriría si, por algún motivo, aumentaran los errores en el modelo del proceso y/o de las mediciones. ¿Cómo sería el comportamiento del robot en tal caso? ¿Qué modificaría para tener esto en consideración? ¿Cómo esperaría que se comporte el robot luego de esta modificación?