Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ciencias Exactas y Naturales
Departamento de Ciencias de la Computación
R-521 — Robótica Móvil

Trabajo Práctico

EKF y PF para Localización de un Robot Móvil

1. Introducción

El objetivo del Trabajo Práctico es comprender cómo funciona el Filtro de Kalman Extendido (EKF) y el Filtro de Partículas (PF) para la localización de un robot móvil y desarrollar una implementación de cada uno.

Para este trabajo se utilizará el framework de Python provisto por la cátedra. [1]

Material de lectura útil: slides de las clases, Capítulos 3, 4, 5, 7 y 8 del libro Probabilistic Robotics, Thrun, Burgard and Fox.

2. Entrega

- Se debe proveer un repositorio git que contenga el código desarrollado y un archivo `README.md` con las instrucciones de compilación y ejecución.
- Se debe entregar un informe en LaTeX explicando el trabajo realizado y analizando los resultados obtenidos.

3. Ejercicio Teórico

Ejercicio 1. Jacobiano del Modelo de Movimiento

La Figura 1 describe un modelo de movimiento simple. El estado del robot es su posición y orientación 2D: $s = [x, y, \theta]$. El control del robot es $u = [\delta_{rot1}, \delta_{trans}, \delta_{rot2}]$, es decir el robot rota δ_{rot1}, se desplaza δ_{trans} y luego rota δ_{rot2}.

Las ecuaciones para el modelo de movimiento g son las siguientes:

$$\begin{aligned} x_{t+1} &= x_t + \delta_{trans} \cos(\theta_t + \delta_{rot1}) \\ y_{t+1} &= y_t + \delta_{trans} \sin(\theta_t + \delta_{rot1}) \\ \theta_{t+1} &= \theta_t + \delta_{rot1} + \delta_{rot2} \end{aligned}$$

donde $s_{t+1} = [x_{t+1}, y_{t+1}, \theta_{t+1}]$ es la predicción del modelo de movimiento. Derivar los Jacobianos de g respecto del estado $G = \dfrac{\partial g}{\partial s}$ y el control $V = \dfrac{\partial g}{\partial u}$

$$G = \begin{bmatrix} \dfrac{\partial x'}{\partial x} & \dfrac{\partial x'}{\partial y} & \dfrac{\partial x'}{\partial \theta} \\ \dfrac{\partial y'}{\partial x} & \dfrac{\partial y'}{\partial y} & \dfrac{\partial y'}{\partial \theta} \\ \dfrac{\partial \theta'}{\partial x} & \dfrac{\partial \theta'}{\partial y} & \dfrac{\partial \theta'}{\partial \theta} \end{bmatrix} \qquad V = \begin{bmatrix} \dfrac{\partial x'}{\partial \delta_{rot1}} & \dfrac{\partial x'}{\partial \delta_{trans}} & \dfrac{\partial x'}{\partial \delta_{rot2}} \\ \dfrac{\partial y'}{\partial \delta_{rot1}} & \dfrac{\partial y'}{\partial \delta_{trans}} & \dfrac{\partial y'}{\partial \delta_{rot2}} \\ \dfrac{\partial \theta'}{\partial \delta_{rot1}} & \dfrac{\partial \theta'}{\partial \delta_{trans}} & \dfrac{\partial \theta'}{\partial \delta_{rot2}} \end{bmatrix}$$

[1] Este trabajo práctico está basado en el Homework 2 del curso CSE571: Probabilistic Robotics of University of Washington - Paul G. Allen School of Computer Science & Engineering `https://courses.cs.washington.edu/courses/cse571/20sp/homeworks/HW2.pdf`.

4. Ejercicios de Programación

Implementar un Filtro de Kalman Extendido (EKF) y un Filtro de Partículas (PF) para localizar un robot utilizando *landmarks*. Usaremos el modelo de movimiento basado en la odometría que se derivó previamente. Asumimos que hay landmarks presentes en el entorno del robot. El robot recibe los ángulos (*bearing*) a los landmarks y los ID de los landmarks como observaciones: (orientación, ID de landmark).

Asumimos un modelo de ruido para el modelo de movimiento de odometría con parámetro α (ver Libro Probabilisitc Robotics Tabla 5.6) y un modelo de ruido separado para las observaciones de ángulo con parámetro β (ver Libro Probabilisitc Robotics Sección 6.6). El ID del landmark en las observaciones es sin ruido. Consulte el código provisto para obtener detalles de implementación.

En cada paso de tiempo, el robot comienza desde el estado actual y se mueve de acuerdo con la entrada de control. El robot luego recibe una observación del mundo. Utilizar esta información para localizar el robot sobre la secuencia de tiempo completa con un EKF y PF.

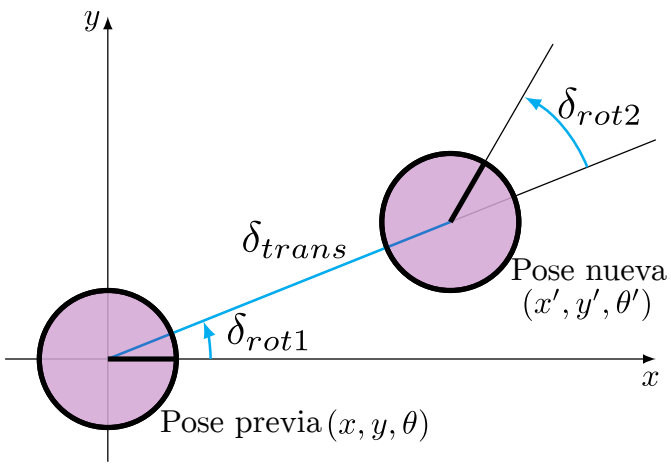

Figura 1: Modelo de movimiento.

4.1. Descripción del código

El código provisto está escrito en Python y depende de las librerías `NumPy` y `Matplotlib`.

- `localization.py` — principal punto de entrada para ejecutar experimentos.
- `soccer_field.py` — implementa las funciones del modelo dinámico y de observación, así como los modelos de ruido para ambos. ¡Implementar los jacobianos acá!
- `utils.py` — contiene varias funciones de ploteo, así como una función útil para normalizar un ángulo entre $[-\pi, \pi]$.
- `policy.py` — contiene una política simple que puede ignorar sin problemas.
- `ekf.py` — ¡Implementar acá el Filtro de Kalman Extendido!
- `pf.py` — ¡Implementar acá el Filtro Partículas!

4.2. Interfaz de comando

Para visualizar el robot en el entorno del campo de fútbol, ejecute

```
python localization.py --plot none
```

La línea azul traza la posición del robot (ground-truth), que es el resultado de acciones ruidosas. La línea verde traza la posición del robot asumiendo que las acciones no son ruidosas (trayectoria deseada pero no la real). Después de implementar un filtro, la posición del robot estimada por el filtro se dibujará en rojo.

```
python localization.py --plot ekf
python localization.py --plot pf
```

Para ver los flags de la línea de comandos disponibles, ejecute

```
python localization.py -h
```

4.3. Formato de los datos

- estado: $[x, y, \theta]$
- control: $[\delta_{rot1}, \delta_{trans}, \delta_{rot2}]$
- observación: $[\theta_{bearing}]$

4.4. Notas

- Llamar a `utils.minimized_angle` cada vez que un ángulo o una diferencia de ángulo pueda exceder $[-\pi, \pi]$.
- Utilizar el muestreador sistemático de baja varianza visto en clase. Da una distribución más suave de partículas y también requiere un único número aleatorio por paso de remuestreo.
- Desactivar la visualización para acelerar la ejecución.

Ejercicio 2. Implementar EKF

Implementar el algoritmo de Filtro de Kalman Extendido en `ekf.py`. Deberá completar `ExtendedKalmanFilter.update` y los métodos G, V y H. Los resultados de una implementación exitosa de EKF deben ser comparables a los siguientes resultados.

```
python localization.py ekf --seed 0
...
Mean position error: 8.9983675360847
Mean Mahalanobis error: 4.416418248584298
ANEES: 1.472139416194766
```

a) Graficar el camino del robot real y el camino estimado por el filtro con los parámetros predeterminados (proporcionados).

b) Graficar el error de posición medio a medida que los factores α y β varían sobre $r = [1/64, 1/16, 1/4, 4, 16, 64]$[2] y analizar los resultados obtenidos.

Ejecutar 10 ensayos por valor de r. Una ejecución podría ser algo como:

```
python localization.py ekf --data-factor 4 --filter-factor 4
```

[2]Dado que los factores se multiplican con varianzas, esto es entre $1/8$ y 8 veces los valores de ruido predeterminados.

c) Graficar el error de posición medio y ANEES (*average normalized estimation error squared*) a medida que los factores α, β del filtro varían sobre r (como arriba) mientras los datos son generados con los valores por defecto. Analizar los resultados obtenidos.

Ejercicio 3. Implementar PF

Implementar el algoritmo de Filtro de Partículas en `pf.py`. Se debe completar `ParticleFilter.update` y `ParticleFilter.resample`.

```
python localization.py pf --seed 0
...
Mean position error: 8.567264372950905
Mean Mahalanobis error: 14.742252771106532
ANEES: 4.914084257035511
```

a) Graficar el camino del robot real y el camino estimado por el filtro con los parámetros predeterminados.

b) Graficar el error de posición medio a medida que los factores α, β varían sobre r y discuta los resultados.

c) Graficar el error de posición medio y ANEES a medida que los factores α, β del filtro varían sobre r mientras los datos son generados con los valores por defecto.

d) Graficar el error de posición medio y ANEES a medida que los factores α, β varían sobre r y el número de partículas varía sobre $[20, 50, 500]$.