Universidad de Buenos Aires, Facultad de Ingeniería

# Práctica 3 - Localización: Filtro de Partículas y EKF

Robótica Móvil - Un enfoque probabilístico

Prof. Dr. Ignacio Mas

2 de noviembre de 2020

**Fecha límite de entrega:** 13/11/20, 13:00hs.
**Modo de entrega:** Enviar por el Aula Virtual del Campus el código (.m) comentado y los gráficos (.jpg ó .pdf) y/o animaciones.

## 1. Filtro de Partículas

En este ejercicio se implementará un filtro de partículas basándose en una estructura de código provista por la cátedra.

### 1.1. Notas preliminares

La estructura provista contiene los modelos de movimiento y de medición, para que el esfuerzo del desarrollo sea en el filtro propiamente dicho. El archivo comprimido que se provee incluye las carpetas:

- **data** contiene la descripción del mundo y las lecturas del sensor
- **matlab** contiene la estructura del filtro de partículas con secciones para ser completadas
- **plots** guarda los gráficos generados como resultado

Para ejecutar el filtro, desde la carpeta *matlab* correr el archivo `particle_filter.m`. Mientras que corre el filtro, los gráficos se van guardando en la carpeta *plots*. El algoritmo inicialmente está incompleto y no correrá correctamente. La estructura hace uso de la

librería *librobotics*, escrita por Kai Arras (ASL-EPFL) para la visualización. Todas las funciones están definidas en la estructura de código.
Algunos consejos adicionales:

- Desactivar la visualización comentando la línea `plot state(...)` en el script `particle_filter.m` para acelerar la ejecución.
- Para probar el filtro sólo con algunos pasos, cambiar el tiempo final en el bucle *for* principal con, por ejemplo, `for t = 1:50`.

## 1.2. Ejercicio

Un filtro de partículas consiste principalmente de tres pasos:

1. Muestrear nuevas partículas usando el modelo de movimiento.
2. Calcular el peso de las nuevas partículas usando el modelo de medición.
3. Calcular el nuevo *belief* remuestreando las partículas de manera proporcional a sus pesos, con reemplazo.

El modelo de movimiento (1) está implementado en la estructura de código provista. En este ejercicio se deben implementar (2) y (3).

- Completar la función `measurement_model.m`. Esta función actualiza el filtro usando un sensor de distancia. La función toma como entrada un conjunto $l$ de marcadores (landmarks), un conjunto $z$ de observaciones independientes de marcadores y un conjunto de partículas $x$, con:
  - $l$: una estructura representando un mapa de marcadores en el ambiente, donde cada marcador *l(i)* tiene un identificador *l(i).id* y una posición *l(i).x, l(i).y*.
  - $z$: una estructura que contiene un número de observaciones de marcadores, donde cada observación *z(i)* tiene un identificador *z(i).id* y una distancia medida *z(i).range*.
  - $x$: una matriz de tamaño $N \times 3$ donde $N$ es el número de partículas, *x(:,1)* representa la coordenada $x$ y *x(:,2)* representa la coordenada $y$ de la partícula. La orientación *x(:,3)* no es usada en este ejercicio.

  La función debe devolver un vector de pesos $w$ con el mismo tamaño que el número de partículas. En vez de calcular la probabilidad, es suficiente calcular el *likelihood* *p(z |x,l)*. El desvío estándar de la medición es $\sigma_r = 0,2$. El código incluido muestra como obtener un marcador con un identificador dado.
- Completar la función `resample.m` implementando el Muestreo Estocástico Universal. La función toma como entrada un conjunto de partículas $x$ con la estructura ya mencionada y un vector de pesos $w$ con los pesos de cada partícula. La función debe devolver otro conjunto de partículas manteniendo la misma estructura.

### 1.3. Visualización

Se desea elegir una sola hipótesis del conjunto de partículas del filtro para representar la posición del robot. Implementar la función `mean_position.m` que se usa para graficar la pose del robot.

Nota: Con el comando *ffmpeg* (si está instalado en tu sistema operativo) se puede generar una animación de los gráficos de la carpeta *plots* de la siguiente manera:
`ffmpeg -r 10 -i pf_%03d.png pf.mp4`

## 2. Filtro de Kalman Extendido

En este ejercicio se implementará un filtro de Kalman Extendido (EKF) basándose en una estructura de código provista por la cátedra.

### 2.1. Notas preliminares

La estructura provista contiene los distintos componentes del filtro EKF, para que el esfuerzo del desarrollo sea en los detalles de la implementación del filtro propiamente dicho. El archivo comprimido que se provee incluye las carpetas:

- **data** contiene la descripción del mundo y las lecturas del sensor
- **matlab** contiene la estructura del filtro EKF con secciones para ser completadas
- **plots** guarda los gráficos generados como resultado

Para ejecutar el filtro, desde la carpeta *matlab* correr el archivo `extended_kalman_filter.m`. Mientras que corre el filtro, los gráficos se van guardando en la carpeta *plots*. El algoritmo inicialmente está incompleto y no correrá correctamente. La estructura hace uso de la librería *librobotics*, escrita por Kai Arras (ASL-EPFL) para la visualización. Todas las funciones están definidas en la estructura de código.
Algunos consejos adicionales:

- Desactivar la visualización comentando la línea `plot_state(...)` en el script `extended_kalman_filter.m` para acelerar la ejecución.
- Para probar el filtro sólo con algunos pasos, cambiar el tiempo final en el bucle *for* principal con, por ejemplo, `for t = 1:50`.

### 2.2. Paso de predicción EKF

Supongamos que tenemos un robot diferencial operando en el plano, por lo que su estado esta definido por $\langle x, y, \theta \rangle$. Su modelo de movimiento esta definido como el Modelo de Odometría visto en clase en el capítulo de *modelos de movimiento*.

1. Calcular la matriz jacobiana $G_t$ de la función de movimiento $g$ sin ruido.

2. Implementar el paso de predicción del filtro EKF en el archivo `prediction_step.m` usando el jacobiano $G_t$ calculado en el punto anterior. Asumir que el ruido del modelo de movimiento esta definido por:

$$Q_t = \begin{pmatrix} 0{,}2 & 0 & 0 \\ 0 & 0{,}2 & 0 \\ 0 & 0 & 0{,}02 \end{pmatrix}.$$

## 2.3. Paso de corrección EKF

1. Calcular la matriz jacobiana $H_t$ de la función de medición $h$ sin ruido de un sensor que sólo mide distancia (range).

2. Implementar el paso de corrección del filtro EKF en el archivo `correction_step.m` usando el jacobiano $H_t$. Asumir que el ruido de medición está caracterizado por la matriz diagonal cuadrada $R_t$:

$$R_t = \begin{pmatrix} 0{,}5 & 0 & 0 & \dots \\ 0 & 0{,}5 & 0 & \dots \\ 0 & 0 & 0{,}5 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \in \mathbb{R}^{size(z_t) \times size(z_t)}.$$

Luego de implementar los pasos de predicción y corrección, puede generarse una animación con las figuras guardadas en la carpeta *plots*.

Nota: Con el comando *ffmpeg* (si está instalado en tu sistema operativo) se puede generar una animación de los gráficos de la carpeta *plots* de la siguiente manera:
`ffmpeg -r 10 -i kf_%03d.png ekf.mp4`