Universidad de Buenos Aires, Facultad de Ingeniería

Práctica 2 - Modelos Probabilísticos, Filtros Discretos y de Partículas

Robótica Móvil - Un enfoque probabilístico

Prof. Dr. Ignacio Mas

20 de septiembre de 2017

Fecha límite de entrega: 02/10/17, 10hs.
Modo de entrega: Enviar por correo electrónico a `imas@fi.uba.ar` el código (.m) comentado y los gráficos (.jpg ó .pdf) y/o animaciones.

1. Muestreo de distribuciones de probabilidad

Implementar tres funciones en Octave/MATLAB que generen muestras de una distribución normal $\mathcal{N}(\mu, \sigma^2)$. Los parámetros de entrada de dichas funciones son la media μ y la varianza σ^2 de dicha distribución. Como única fuente de aleatoriedad, utilizar muestras de una distribución uniforme.

1. En la primer función, generar las muestras de la distribución normal sumando muestras de 12 distribuciones uniformes.
2. En la segunda función, usar "muestreo con rechazo"
3. En la tercera función, usar el método de la transformación de Box-Muller. El método de Box-Muller permite generar muestras de una distribución normal usando dos muestras uniformemente distribuidas $u_1, u_2 \in [0, 1]$ según la ecuación:

$$x = cos(2\pi u_1)\sqrt{-2\log u_2} \tag{1.1}$$

Comparar los tiempos de ejecución de las tres funciones usando los comandos `tic` y `toc` de Octave/MATLAB.
Además, comparar el tiempo de ejecución de las tres funciones con el de la función `normrnd` de Octave/MATLAB.

2. Modelo de movimiento basado en odometría

Implementar un modelo de movimiento basado en odometría siguiendo estos pasos:

1. Crear una función en Octave/MATLAB que implemente un modelo de movimiento basado en odometría. los tres argumentos de entrada deberán ser:

$$\mathbf{x_t} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \qquad \mathbf{u_t} = \begin{pmatrix} \delta_{r1} \\ \delta_{r2} \\ \delta_t \end{pmatrix} \qquad \alpha = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix}, \tag{2.1}$$

   donde $\mathbf{x}_t$ es la pose actual del robot, $\mathbf{u}_t$ es la lectura de odometría obtenida por los sensores del robot y α son los parámetros de ruido del modelo de movimiento. La función deberá devolver la nueva pose del robot $\mathbf{x}_{t+1}$.

   La implementación de la función deberá incluir el error de obtención de la odometría. Usar los métodos de muestreo del ejercicio 1 para generar muestras aleatorias normalmente distribuidas para introducir ruido en el modelo de movimiento.

2. Evaluar el modelo de movimiento desarrollado generando 5000 muestras con los siguientes argumentos:

$$\mathbf{x_t} = \begin{pmatrix} 2{,}0 \\ 4{,}0 \\ 0{,}0 \end{pmatrix} \qquad \mathbf{u_t} = \begin{pmatrix} \frac{\pi}{2} \\ 0{,}0 \\ 1{,}0 \end{pmatrix} \qquad \alpha = \begin{pmatrix} 0{,}1 \\ 0{,}1 \\ 0{,}01 \\ 0{,}01 \end{pmatrix}, \tag{2.2}$$

   Graficar las posiciones $(\mathbf{x}_{t+1}, \mathbf{y}_{t+1})$ para las 5000 muestras en un solo gráfico.

3. Sensor de Distancia

Un vehículo autónomo no tripulado equipado con un módem 4G recorre las calles de Buenos Aires todos los dias desde la Facultad de Ingeniería Sede Paseo Colón hasta la Sede Las Heras transportando estudiantes. En un mapa de la ciudad, la Sede Paseo Colón se encuentra en la posición $m_0 = (10, 8)^T$ y la Sede Las Heras en la posición $m_1 = (6, 3)^T$. Se tiene acceso a los datos recolectados por dos torres de telefonía celular

T_0 y T_1, que se encuentran respectivamente en las posiciones $x_0 = (12, 4)^T$ y $x_1 = (5, 7)^T$. La distancia entre el vehículo y las torres puede calcularse según la intensidad de la señal de telefonía celular. Estas mediciones de distancia están perturbadas por ruido Gaussiano independiente de media cero y varianzas $\sigma_0^2 = 1$ para la torre T_0 y $\sigma_1^2 = 1{,}5$ para la torre T_1. Las lecturas de distancia de las torres en un momento dado son $d_0 = 3{,}9$ y $d_1 = 4{,}5$.

1. ¿En cúal de las dos sedes es más probable que se encuentre el vehículo? Justificar la respuesta.

2. Implementar una función en Octave/MATLAB que genere un gráfico en 3D de la función de likelihood usada en el punto 1. Indicar en el gráfico los puntos m_0, m_1, x_0 y x_1.

3. Suponer ahora que tenemos información sobre cuanto tiempo de espera tiene el vehículo en cada sede antes de ir hacia la otra. Por lo que se puede suponer que se encuentra en la sede Paseo Colon con una probabilidad P(Paseo Colón)=0.3 y en Las Heras con P(Las Heras)=0.7. Como el vehículo se mueve a gran velocidad por las calles de la ciudad ($v \approx c$), entonces P(otro lado)=0. Usar esta información *a priori* para calcular las probabilidades del punto 1.

4. Filtro Discreto

Queremos estimar la posición de un robot que se mueve es un mundo unidimensional acotado. El mundo tiene 20 celdas y el robot se encuentra inicialmente en la celda 10. El robot no puede salir del area especificada.
En cada paso, el robot puede ejecutar el comando 'avanzar un paso' o 'retroceder un paso'. Este movimiento puede tener cierto error, por lo que el resultado puede no siempre ser el esperado. De hecho, el robot se comporta de la siguiente manera: al ejecutar 'avanzar un paso' el resultado será el siguiente:

- el 25 % de las veces, el robot no se moverá
- el 50 % de las veces, el robot avanzará una celda
- el 25 % de las veces, el robot se moverá dos celdas
- el robot en ningún caso se moverá en el sentido opuesto o avanzará más de dos celdas.

El modelo de 'retroceder un paso' es similar, pero en el sentido opuesto. Ya que el mundo es acotado, el comportamiento al intentar 'avanzar un paso' en los bordes es el siguiente:

- si el robot está en la última celda, al tratar de avanzar se quedará en la misma celda el 100 % de las veces
- si el robot está en la penúltima celda, al tratar de avanzar se quedará en la misma celda el 25 % de las veces, y se moverá a la próxima celda el 75 % de las veces.

Lo mismo sucederá en sentido contrario en el otro extremo del mundo al intentar retroceder.
Implementar un filtro de Bayes discreto en Octave/MATLAB y estimar la posición final del robot (belief) despues de ejecutar 9 comandos consecutivos de 'avanzar un paso' seguidos de 3 comandos de 'retroceder un paso'. Graficar el belief resultante de la posición del robot. Comenzar con el belief inicial de `bel=[zeros(10; 1); 1; zeros(9; 1)];`.

5. Filtro de Partículas

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

5.1. 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
- **octave** 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 *octave* 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`.

5.2. 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* muestreando las partículas de manera proporcional a sus pesos con reemplazo.

El modelo de movimiento (1) está implementado en la estrucutra 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 independiantes 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 estandard de la medición es $\sigma_r = 0,2$. El código incluído 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.

5.3. 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 -b 500000 -i pf_%03d.png pf.mp4`