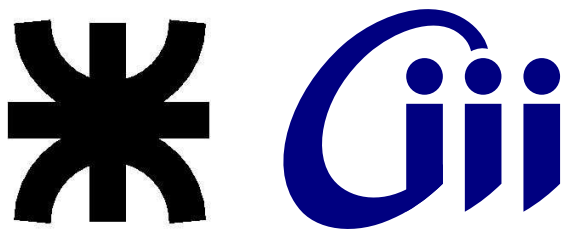
Filtro de partículas para localización de robots móviles implementado en arquitecturas multi-núcleo

Claudio J. Paz Gonzalo F. Perez Paina Luis R. Canali Julio H. Toloza

Universidad Tecnológica Nacional - Facultad Regional Córdoba
http://ciii.frc.utn.edu.ar
Córdoba, Argentina

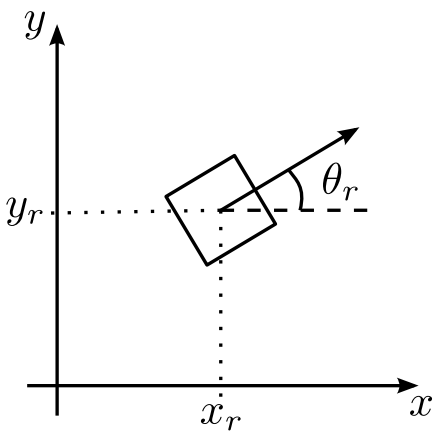


VII Jornadas Argentinas de Robótica 2012
21, 22 y 23 de Noviembre de 2012
Facultad de Ingeniería - Olavarría - UNICEN

Contenido

Motivación

La estimación de la posición y la orientación de un vehículo son fundamentales en robótica para implementar algoritmos de navegación y evasión de obstáculos.



- Las aplicaciones industriales o de servicio se desarrollan en ambientes techados imposibilitando el uso de GPS
- Los métodos de *dead reckoning* tienen errores de integración que crecen sin límites con el tiempo

Objetivos

- Implementar un filtro de partículas para determinar la posición y orientación del robot utilizando información provista por sensores de rango.
- Implementar los algoritmos en distintos tipos de arquitecturas y realizar comparaciones de performance con el objetivo de determinar la viabilidad del sistema trabajando en tiempo real.

Estimación Bayesiana

Métodos recursivos que permiten estimar el estado de un sistema.

- En base al estado previo del sistema.
- La información sensorial más reciente.
- La dinámica del sistema posee componentes modeladas por una *pdf*
- Las mediciones también se modelan probabilisticamente mediante una *pdf*.

Proceso estocástico

$$\mathbf{x}_k = \mathbf{f}_k(\mathbf{x}_{k-1}, \mathbf{w}_{k-1})$$
$$\mathbf{z}_k = \mathbf{h}_k(\mathbf{x}_k, \mathbf{v}_k)$$

Estimación Bayesiana

La estimación de la *pdf* a posteriori $p(\mathbf{x}_k|\mathbf{z}_{1:k})$ se realiza en dos pasos:

- Predecir el estado en el instante k a partir del estado estimado en el instante previo por medio de la ecuación Chapman-Kolmogorov

$$p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})d\mathbf{x}_{k-1}$$

- Actualizar la *pdf* a priori mediante la identidad de Bayes

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})}$$

donde

$$p(\mathbf{z}_k|\mathbf{z}_{1:k-1}) = \int p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})d\mathbf{x}_k$$

Filtro de partículas

El filtro aproxima la *pdf* a posteriori por medio de un conjunto de muestras aleatorias con pesos asociados en función de esta distribución y estima el estado del sistema en base a estos pesos.

La estimación del estado se realiza en un proceso iterativo que consta de las tres etapas:

- *Predicción*
- *Actualización*
- *Remuestreo*

Filtro de partículas
Método

En la etapa de *predicción* se utiliza la ecuación de proceso como modelo para hacer evolucionar cada partícula $\mathbf{x}_k^i$ y así obtener una hipótesis del estado a partir de cada una.

La *actualización* consiste en evaluar cada partícula $\mathbf{x}_k^i$, con el modelo de medición para obtener $\mathbf{z}_k^i$. Esta medición se compara con la $\mathbf{z}_k$ para obtener la verosimilitud de esa partícula. A mayor verosimilitud, mayor el peso q_k^i asociado a la partícula $\mathbf{x}_k^i$

$$q_k^i = q_{k-1}^i \, p(\mathbf{z}_k | \mathbf{x}_k^i)$$

El *remuestreo* es necesario debido a la degeneración de las muestras. Luego de pocas iteraciones una partícula tiene un gran peso y las demás tienen pesos muy próximos a cero. La solución consiste en sustituir las partículas de menor peso por copias de aquellas de mayor peso.

Filtro de partículas
Predicción

Con la ecuación de proceso se obtiene la evolución de cada partícula dado su estado anterior

$$\mathbf{x}_k^i = \mathbf{f}_k(\mathbf{x}_{k-1}^i, \mathbf{w}_{k-1})$$

usando las lecturas de movimiento relativo obtenidos de la odometría del robot para realizar una transformación, la cual está comprendida por una primera rotación, una traslación y luego una segunda rotación

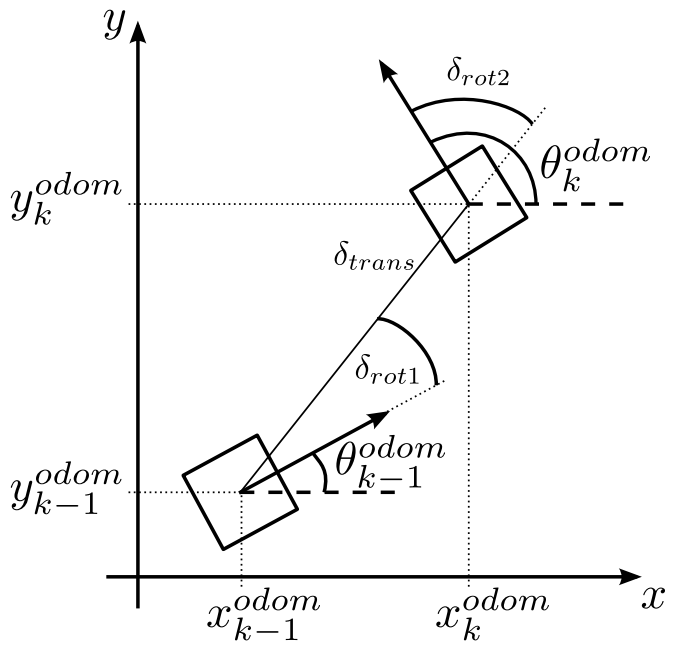
$$[\delta_{rot1}, \delta_{trans}, \delta_{rot2}]^T$$

dadas las lecturas de odometría al tiempo discreto k y $k-1$

$$\mathbf{x}_k^{odom} = \left[x_k^{odom}, y_k^{odom}, \theta_k^{odom}\right]^T$$

$$\mathbf{x}_{k-1}^{odom} = \left[x_{k-1}^{odom}, y_{k-1}^{odom}, \theta_{k-1}^{odom}\right]^T$$

Filtro de partículas
Predicción



$$\delta_{rot1} = atan2(y_k^{odom} - y_{k-1}^{odom}, x_k^{odom} - x_{k-1}^{odom}) - \theta_{k-1}^{odom}$$

$$\delta_{trans} = \sqrt{(x_{k-1}^{odom} - x_k^{odom})^2 + (y_{k-1}^{odom} - y_k^{odom})^2}$$

$$\delta_{rot2} = \theta_k^{odom} - \theta_{k-1}^{odom} - \delta_{rot1}$$

Filtro de partículas
Predicción

Asumiendo que estas variables son afectadas por ruido gaussiano de media cero

$$\begin{aligned}
\delta_{rot1} &= \hat{\delta}_{rot1} + \varepsilon_{rot1}, \qquad \varepsilon_{rot1} \sim \mathcal{N}(0, \sigma_{rot1}) \\
\delta_{trans} &= \hat{\delta}_{trans} + \varepsilon_{trans}, \qquad \varepsilon_{trans} \sim \mathcal{N}(0, \sigma_{trans}) \\
\delta_{rot2} &= \hat{\delta}_{rot2} + \varepsilon_{trans}, \qquad \varepsilon_{rot2} \sim \mathcal{N}(0, \sigma_{rot2})
\end{aligned}$$

con

$$\begin{aligned}
\sigma_{rot1} &= \alpha_1 |\delta_{rot1}| + \alpha_2 |\delta_{trans}| \\
\sigma_{trans} &= \alpha_3 |\delta_{trans}| + \alpha_4 (|\delta_{rot1}| + |\delta_{rot2}|) \\
\sigma_{rot2} &= \alpha_1 |\delta_{rot2}| + \alpha_2 |\delta_{trans}|
\end{aligned}$$

donde $\alpha_i, i = 1, \ldots, 4$ son los parámetros de movimiento específicos del robot usado.

Filtro de partículas
Predicción

Entonces, el estado actual del robot dependiendo del estado anterior usando el modelo de movimiento queda

$$\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} + \begin{bmatrix} \delta_{trans} \cos(\theta_{k-1} + \delta_{rot1}) \\ \delta_{trans} \sin(\theta_{k-1} + \delta_{rot1}) \\ \delta_{rot1} + \delta_{rot2} \end{bmatrix}$$

En el caso del filtro de partículas, en cada paso, debe hacerse evolucionar cada partícula con este modelo y con

$$\hat{\delta}_{rot1} = \delta_{rot1} + \mathcal{N}(0, \sigma_{rot1})$$
$$\hat{\delta}_{trans} = \delta_{trans} + \mathcal{N}(0, \sigma_{trans})$$
$$\hat{\delta}_{rot2} = \delta_{rot2} + \mathcal{N}(0, \sigma_{rot2})$$

donde $\hat{\delta}_{rot1}$, $\hat{\delta}_{trans}$ y $\hat{\delta}_{rot2}$ son las entradas de control que se usan para cada partícula y σ_{rot1}, σ_{trans} y σ_{rot2} las desviaciones estándar.

Filtro de partículas
Predicción

Entonces, para cada partícula, el cálculo de la nueva posición de la misma, dada la lectura de odometría es

$$x_k^i = x_{k-1}^i + \hat{\delta}_{trans} cos(\theta_{k-1}^i + \hat{\delta}_{rot1})$$
$$y_k^i = y_{k-1}^i + \hat{\delta}_{trans} sin(\theta_{k-1}^i + \hat{\delta}_{rot1})$$
$$\theta_k^i = \theta_{k-1}^i + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$$

El algoritmo es completamente paralelizable, ya que no hay dependencia entre las partículas

for all $old\_particles$ **do**
  $new\_particle(i) = \mathbf{f}(old\_particle(i), robot\_odom, \mathbf{w})$
end for
return $new\_particles$

Filtro de partículas
Actualización

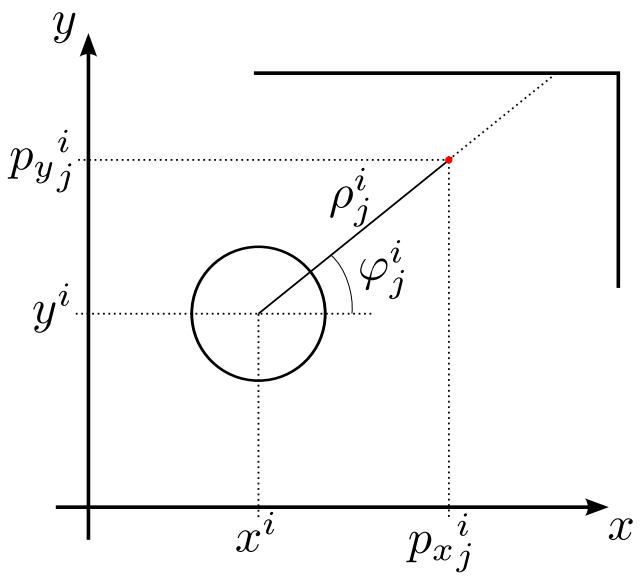
En la actualización se genera la medición con la que se evaluará cada partícula.

$$\mathbf{z}_k^i = \mathbf{h}(\mathbf{x}_k^i, \mathcal{M})$$

donde $\mathbf{h}(\cdot)$ es una función no lineal que relaciona una posición en el mapa $\mathcal{M}$ con la lectura que indica el escáner LASER y donde

$$\mathbf{z}_k^i = \left[z_k^1, \ldots, z_k^M\right]^T$$

es el conjunto de M mediciones realizadas por el LASER correspondientes al tiempo discreto k para la partícula i. Esta lectura es generada para cada partícula mediante un algoritmo trazado de rayos



Filtro de partículas
Actualización

Nuevamente, el algoritmo muestra que esta parte del código es paralelizable ya que no hay relación entre las partículas y los LASER

for all $particles$ **do**
  for all $beams$ **do**
    $dot \leftarrow particle(i).pos$
    $beam(j).range \leftarrow 0$
    while $beam(j).range < max\_range$ **do**
      $dot \leftarrow fwd(dot, beam(j).ang)$
      if $map(dot)\ is\ occupied$ **then**
        $beam(j).range \leftarrow dist(particle(i).pos, dot)$
        $break$
      end if
    end while
  end for
  $particle(i).ranges \leftarrow beams.ranges$
end for
return $particles.ranges$

Filtro de partículas
Actualización

Entonces, debido a que el ruido de cada haz del sensor no esta correlacionado con los otros, se puede escribir la verosimilitud de cada medición como un producto

$$p(\mathbf{z}_k^i|\mathbf{x}_k^i, \mathcal{M}) = \prod_{m=1}^{M} p(z_k^m|\mathbf{x}_k^i, \mathcal{M})$$

Este algoritmo no es completamente paralelizable ya que hay dependencia entre los productos que forman el peso de la partícula

$z \leftarrow real\_beams$
for all $particles$ **do**
  $weight \leftarrow 1$
  $z\_hat \leftarrow particle(i).ranges$
  for all $beams$ **do**
    $weight \leftarrow weight \times \frac{1}{\sqrt{2\pi} \times sigma\_laser} e^{-\frac{1}{2} \frac{(z(j) - z\_hat(j))^2}{sigma\_laser^2}}$
  end for
  $particle(i).weight \leftarrow weight$
end for
return $particles.weights$

Filtro de partículas
Actualización

Finalmente, para obtener el estado filtrado se utiliza un promedio ponderado de los valores de pose de cada partícula con su respectivo peso. Este algoritmo tampoco es paralelizable

for all $particles$ **do**
$x\_hat \leftarrow x\_hat + particle(i).weight \times particle(i).x$
$y\_hat \leftarrow y\_hat + particle(i).weight \times particle(i).y$
$theta\_hat \leftarrow theta\_hat + particle(i).weight \times particle(i).theta$
end for

Filtro de partículas
Remuestreo

Antes de proceder con el remuestreo, se debe adecuar el conjunto de pesos, normalizándolo y realizando la suma acumulada de cada peso normalizado

$total\_weight \leftarrow 0$
for all $particles$ **do**
    $total\_weight \leftarrow total\_weight + particle(i).weight$
end for
$cumsum \leftarrow 0$
for all $particles$ **do**
    $particle(i).weight \leftarrow particle(i).weight/total\_weight$
    $cumsum \leftarrow cumsum + particle(i).weight$
    $particle(i).cumsum \leftarrow cumsum$
end for

El algoritmo muestra que no se puede paralelizar esta etapa, ya que se hace una suma de todos los pesos para normalizar y luego, partícula por partícula se calcula la suma acumulada, siendo este el paso de más consumo computacional.

Filtro de partículas
Remuestreo

Luego de pocos pasos de actualización algunas partículas tienen pesos muy grandes y la mayoría pesos despreciables. Para evitar este fenómeno se realiza el remuestreo.

for all $particles$ **do**
  $u \leftarrow \mathcal{U}(0, 1]$
  $l \leftarrow 0$
  while $u > particle(l).cumsum$ **do**
    $l++$
  end while
  $new\_particle(i) \leftarrow particle(l)$
end for
return $new\_particles$;

Resultados
Plataformas de prueba

El mismo filtro de partículas fue implementado en tres arquitecturas diferentes

- Rack Server PowerEdge R715
  - 16 Núcleos ADM Opteron
  - 2GHz Clock
  - 16GB RAM
- NVIDIA GTX560
  - 336 CUDA Cores
  - 1.9GHz Clock
  - 1GB RAM
- Intel Dual Core
  - 2.13GHz Clock
  - 2GB RAM

Resultados
Consideraciones

- Para evaluar el desempeño del filtro en cada arquitectura, se propuso un mapa de $20m \times 10m$ y una trayectoria de $17m$ donde el error de localización debía ser menor a $10cm$ una vez localizado el robot.
- Se consideró que el tiempo que debía demorar el filtro en terminar un ciclo debía ser menor al período de muestreo de un sensor de rango comercial para poder operar en tiempo real. Se fijó este tiempo en $40ms$.
- Se utilizó C/C++ para implementar el filtro de partículas. En el Dual Core se utilizó un solo núcleo.
- En el servidor de 16 núcleos se utilizaron las bibliotecas *OpenMP* para distribuir la carga en todos los núcleos.
- Para el GPU se codificaron las funciones en lenguaje C para la arquitectura CUDA.

Resultados
Los números

Partículas	Dual Core		
	50	100	200
500	**0.024116s**	0.047149s	0.09326s
1000	0.048838s	0.094874s	0.186954s
2000	0.100629s	0.192564s	0.377385s
3000	0.156249s	0.294173s	0.570679s
4000	0.21409s	0.398343s	0.767899s

Partículas	GTX560		
	50	100	200
500	**0.008371s**	**0.016783s**	**0.035316s**
1000	**0.016369s**	**0.032414s**	0.067133s
2000	**0.025286s**	0.049757s	0.10148s
3000	**0.033914s**	0.067074s	0.13055s
4000	0.042073s	0.083439s	0.16423s

Partículas	Opteron 16 Núcleos		
	50	100	200
500	**0.001564s**	**0.005621s**	**0.010434s**
1000	**0.006930s**	**0.012090s**	**0.021029s**
2000	**0.015966s**	**0.025951s**	0.042716s
3000	**0.028121s**	0.042018s	0.069182s
4000	0.040781s	0.059155s	0.097497s

- El servidor fue más veloz en el procesamiento en todos los casos.
- Ninguna implementación logró cumplir con los requisitos temporales para 4000 partículas con ninguna cantidad de LASER.
- La GPU tuvo velocidades de procesamiento del mismo orden de magnitud que el servidor.
- El Dual Core solo pudo cumplir los requisitos para la menor cantidad de partículas y lecturas del sensor LASER.

Conclusiones y trabajos futuros

Conclusiones

- Es posible utilizar filtros de partículas para localización en ambientes estructurados para aplicaciones en tiempo real.
- Si bien el servidor fue mas rápido que la GPU, la implementación corriendo en el GPU fue la más trivial posible obteniendo valores del mismo orden de magnitud.
- Montar un servidor multi-núcleo en un robot móvil puede ser tarea complicada o impracticable en robots más pequeños, mientras que se comercializan por una fracción del costo laptops con GPU de características semejantes que la evaluada.

Trabajos futuros

- Evaluar métodos de optimización para realizar las etapas paralelizables.
- Implementar los algoritmos conocidos para tratar problemas seriales con arquitecturas multi-núcleos.
- Codificar una versión de *ray tracing* donde se trate a los sensores de rango y a los muros como vectores.

Filtro de partículas para localización de robots móviles implementado en arquitecturas multi-núcleo

Muchas Gracias

Claudio Paz
cpaz@scdt.frc.utn.edu.ar