

AUTO-LOCALIZACIÓN DE UN ROBOT MÓVIL AIBO MEDIANTE EL MÉTODO DE MONTE CARLO

Pablo Guerrero, Javier Ruiz-del-Solar

Dept. Ing. Eléctrica, Casilla 412-3, Universidad de Chile

Email: pguerrer@ing.uchile.cl, jruizd@ing.uchile.cl

Web: <http://www.robocup.cl>

RESUMEN

La auto-localización de un robot móvil es una importante habilidad a ser desarrollada, pues permite a éste conocer en cada momento su posición y su orientación, es decir su pose, respecto a una representación global del espacio. Los métodos existentes de auto-localización resuelven este problema como uno de estimación de parámetros. Entre los métodos existentes destaca el de Monte Carlo por su generalidad, elegancia y rapidez de convergencia. El objetivo de este artículo es presentar la resolución del problema de localización de robots cuadrúpedos AIBO, en el contexto del fútbol robótico, utilizando el método de localización de Monte Carlo. Localizarse correctamente es esencial para un futbolista y por ende para un robot futbolista. Por lo mismo el rendimiento de un equipo de fútbol robótico dependerá fuertemente de la capacidad de sus integrantes para auto-localizarse adecuadamente en la cancha.

1. Introducción

La robótica móvil es un área de intensa investigación cuyo objetivo último es el desarrollo de robots autónomos que puedan desenvolverse en ambientes dinámicos (cambiantes). En robótica móvil se presentan tres problemas fundamentales a resolver: la navegación, la localización, y el reconocimiento del entorno o construcción de mapas de entorno.

La navegación requiere tener un conocimiento del entorno donde se desenvuelve el robot para ser eficiente. Este conocimiento del entorno requiere la existencia de un mapa del entorno y requiere que el robot esté localizado en dicho mapa. Si la navegación es de tipo deliberativa, se utiliza un mapa del entorno, construido en forma previa por el robot o entregada al robot por su diseñador, para navegar. Si la navegación es de tipo reactiva se navega utilizando únicamente la información percibida por los sensores, para por ejemplo evitar chocar. Los mapas del entorno permiten que el robot tenga una representación del medio en que se está desarrollando. Para que el robot pueda construir un mapa del entorno por si mismo,

requiere poder navegar reactivamente por su entorno y además estar posicionado o localizado adecuadamente en el mapa que está construyendo. Idealmente mientras el robot está construyendo el mapa de su entorno debiera estar alerta ante cambios en éste.

La tarea de localización implica que el robot debe conocer en cada momento su posición y su orientación, es decir su pose, respecto a una representación global del espacio. Es decir se trata de una auto-localización. Por motivos de economía de lenguaje en el contexto de este trabajo utilizaremos sencillamente el término localización, aunque debe entenderse que nos referimos a la auto-localización del robot. Por otra parte y como ha sido mencionado anteriormente, la localización de un robot es imprescindible para las tareas de navegación y de construcción automática de mapas de entorno antes descritas. Es decir, resolver el problema de localización es un requisito esencial para poder desarrollar robots autónomos.

Existen diversos métodos de localización. La existencia de faros, por ejemplo infrarrojos o

satelitales (e.g. GPS) permite que los robots se localicen fácilmente. Sin embargo, su uso presenta inconvenientes. La utilización de faros de infrarrojo o similares requiere que estos estén presentes en el entorno en el cual el robot se desenvuelve, lo que le resta generalidad a este tipo de soluciones. Por otra parte, la utilización de GPS también presenta inconvenientes: su resolución espacial se encuentra limitada y existen problemas de cobertura al trabajar en ambientes interiores. Por estos motivos la solución más ampliamente adoptada consiste en la utilización conjunta de la información odométrica y de las percepciones del robot. La información odométrica determina cuanto se ha desplazado el robot en base a los comandos de movimiento que éste ha recibido y a un *modelo de movimiento* (similar situación acontece cuando los ciegos estiman su desplazamiento en base a los pasos que han dado). La medición de distancia hacia objetos conocidos (*landmarks*) que el robot ha detectado en el medio ambiente y cuyas posiciones conoce previamente, usada en conjunto con un *modelo observacional*, permite también determinar la localización del robot. El problema es que tanto la información odométrica, como las mediciones realizadas por el robot, presentan errores, los cuales deben ser tomados en cuenta al resolver el problema de localización.

En base a lo anteriormente expuesto puede apreciarse que el problema de localización corresponde a un problema de estimación de parámetros, donde los parámetros a estimar son los que definen la pose del robot, es decir su posición y su orientación. De la teoría del control automático sabemos que el problema de estimación de parámetros se puede resolver en forma óptima utilizando el conocido *Filtro de Kalman* [2]. Sin embargo, en el contexto de la robótica móvil los supuestos de optimabilidad del Filtro de Kalman no se cumplen [6]. El modelo de movimiento y el modelo observacional son normalmente no lineales, por lo cual el supuesto de linealidad no es tal. Por otra parte el filtro de Kalman asume que las densidades de probabilidad correspondientes a los errores de ambos modelos son Gaussianas, lo que tampoco se cumple normalmente. El *Filtro de Kalman Extendido* [2] permite tomar en cuenta las no-linealidades de los procesos al linealizarlos en torno a su punto de

operación, en cada instante. Pese a esto el filtro de Kalman extendido aún presenta problemas: mantiene la hipótesis de Gaussianidad y su cálculo es computacionalmente pesado.

Los filtros de partículas permiten resolver los problemas asociados a los filtros de Kalman, no asumen linealidad ni Gaussianidad en los procesos, y su cálculo es computacionalmente eficiente. La idea básica es representar las distribuciones de probabilidad asociadas a los parámetros a estimar mediante partículas, no uniformemente distribuidas. En otras palabras, las distribuciones de probabilidad se discretizan en forma no uniforme. El comportamiento de cada partícula se modifica a través del tiempo en forma adaptiva, utilizando el modelo de movimiento y el modelo observacional. La forma de operar con estas partículas de alguna forma recuerda la forma de operar de los algoritmos genéticos, pero es conceptualmente diferente. *Condensation* [6] y el método de localización de Monte Carlo [3], son dos ejemplos de implementación de la idea básica de los filtros de partículas.

El objetivo de este artículo es presentar la resolución del problema de localización de robots cuadrúpedos AIBO, en el contexto del fútbol robótico, utilizando el método de localización de Monte Carlo. Localizarse correctamente es esencial para un futbolista y por ende para un robot futbolista. El arquero debe conocer exactamente su posición respecto al arco para poder desempeñarse adecuadamente; un defensa debe saber cuán cerca de su área y de la mitad de la cancha se encuentra; finalmente, un atacante debe conocer su posición respecto al arco rival de tal forma de saber en qué dirección enviar el balón y hacia dónde atacar. Como vemos, si no podemos resolver el problema de localización no podemos jugar fútbol. De hecho, todos sabemos que un jugador humano ciego no se puede desenvolver adecuadamente en un cancha de fútbol estándar.

El presente artículo se estructura de la siguiente manera. En la sección 2 se describe en detalle el método de localización de Monte Carlo (MCL) y tres de sus variantes MCL Dual, Mixture MCL y Adaptive MCL. En la sección 3 se presenta la resolución del problema de localización para los robots del Equipo de Fútbol Robótico UChile-1

[9], mediante tres variantes de MCL. En la sección 4 se describe la implementación realizada, así como las pruebas de rendimiento efectuadas sobre las tres variantes de MCL. Finalmente, en la sección 5 se presentan algunas conclusiones y proyecciones de este trabajo.

2. Localización de Monte Carlo

2.1. Filtros de Bayes

El Método de Localización de Monte Carlo (MCL) corresponde a un filtro de Bayes recursivo que estima la distribución *a posteriori* del estado de un sistema condicionada en función de los datos. Los filtros de Bayes apuntan al problema de estimar el estado x de un sistema dinámico, entendido como un sistema variante en el tiempo. En el caso de la localización de un robot móvil, el sistema dinámico es el robot móvil y su entorno, mientras que el estado es su pose, que corresponde a la posición del robot y su ángulo de orientación (x, y, θ) dentro de un sistema de coordenadas cartesianas fijo a su entorno, que en el caso del fútbol robótico corresponde a la cancha. La idea central de los filtros de Bayes es estimar una densidad de probabilidad, sobre el espacio de los estados, condicionada a los datos. Esta densidad de probabilidad, en adelante, la *Creencia* (en inglés *belief*), se denota como:

$$Bel(x_t) = p(x_t / o_t, a_{t-1}, o_{t-1}, a_{t-2}, \dots, o_0)$$

Donde x_t es el estado en el tiempo t , o_t corresponde a los datos perceptuales obtenidos en el instante t , y a_t corresponde a los comandos de movimiento enviados al robot en el instante t , los cuales se usan como datos odométricos. Usando la regla de Bayes y asumiendo que el proceso es de tipo Markoviano se puede derivar la siguiente ecuación recursiva [10]:

$$Bel(x_t) = \eta \cdot p(o_t / x_t) \int p(x_t / x_{t-1}, a_{t-1}) Bel(x_{t-1}) dx_{t-1} \quad (1)$$

donde η corresponde a una constante de normalización. Esta es la ecuación de actualización recursiva para filtros de Bayes. Para implementarla, se necesita conocer dos densidades de probabilidad: $p(x_t / x_{t-1}, a_{t-1})$, a la cual nos

referiremos como *modelo de movimiento*, y $p(o_t / x_t)$, la cual llamaremos *modelo observacional*. Ambos modelos son estacionarios por lo que pueden ser escritos como $p(x' / x, a)$ y $p(o / x)$, respectivamente. El método MCL no requiere una definición analítica del modelo de movimiento; basta con tener un modelo de muestreo que sea compatible con $p(x' / x, a)$. Un modelo de muestreo es una función que recibe x y a como parámetros y entrega posiciones aleatorias x' distribuidas de acuerdo con $p(x' / x, a)$.

2.2. Filtros de Partículas

En espacios de estados continuos, como es el caso de la localización de robots móviles, implementar la ecuación (1) no es trivial, particularmente si son de importancia los costos computacionales. De esta forma, el cálculo de $Bel(x_t)$ es muy costoso para una resolución aceptable, ya que para el cálculo de la integral se debe evaluar $p(x_t / x_{t-1}, a_{t-1})$ y $Bel(x_t)$ en todo el espacio de estados. La idea de los métodos de filtros de partículas, entre los que se encuentra MCL, es representar la creencia $Bel(x_t)$ como un conjunto de N muestras ponderadas, distribuidas de acuerdo con $Bel(x_t)$. Las muestras $X_t^{(i)}$ son también llamadas *partículas* o *individuos*, sus ponderaciones $\omega_t^{(i)}$ son también llamadas *pesos* o *puntajes normalizados*, mientras que el conjunto Φ_t de muestras es llamado *población*.

$$\Phi_t = \{X_t^{(i)}, \omega_t^{(i)} / i = 1, \dots, n\}$$

donde n corresponde al tamaño de la población.

La correspondencia entre los filtros de Bayes y la aproximación usando partículas está dada por:

$$Bel(x_t) \approx \Phi_t$$

Luego, de (1) obtenemos:

$$Bel(X_t^{(i)}) = \frac{p(o_t / X_t^{(i)})}{\sum_{\Phi_t} p(o_t / X_t^{(j)})} \left[\sum_{\Phi_{t-1}} p(X_t^{(i)} / X_{t-1}^{(j)}, a_{t-1}) Bel(X_{t-1}^{(j)}) \right] \quad (2)$$

Definiéndose:

$$\omega_t^{(i)} = \frac{p(o_t/X_t^{(i)})}{\sum_{\Phi_t} p(o_t/X_t^{(j)})} \quad (3)$$

De esta forma, el cálculo de $Bel(x_t)$ se reduce a encontrar Φ_t en cada momento, por lo cual se deben encontrar los valores de todos los $X_t^{(i)}$ y $\omega_t^{(i)}$. Para llevar a cabo este cálculo, MCL utiliza el algoritmo mostrado en la Figura 1.

El término

$$\sum_{\Phi_{t-1}} p(X_t^{(i)}/X_{t-1}^{(j)}, a_{t-1}) Bel(X_{t-1}^{(j)})$$

es calculado en los dos primeros pasos del algoritmo, mientras que la ecuación (3) se refleja en el último paso del algoritmo. Al final de cada iteración se cumple que:

$$Bel(x_t) \approx \begin{cases} \omega_t^{(i)} & x_t = X_t^{(i)} \\ 0 & x_t \neq X_t^{(i)} \end{cases}$$

Notar que en este caso los únicos estados con probabilidad distinta de cero son aquellos iguales a alguna muestra con puntaje positivo, por lo que $Bel(x_t)$ se transforma en una función discreta.

1. Elegir N nuevos $\tilde{X}_t^{(i)}$ a partir de los $X_{t-1}^{(j)}$, en base a su $\omega_{t-1}^{(j)}$. (*Resampling*). Algunos $X_{t-1}^{(j)}$ pueden desaparecer y otros multiplicarse (ver Sección 3.1.1.)
2. $X_t^{(i)} = \tilde{X}_t^{(i)} + \Delta X_{t-1}^{(i)}$ (*sampling*).

Donde $\Delta X_{t-1}^{(i)}$ depende de a_{t-1} y sigue una distribución de probabilidades normal que se deriva de $p(X_t^{(i)}/X_{t-1}^{(j)}, a_{t-1})$ (ver sección 3.1.2.).
3. $\omega_t^{(i)} = \frac{p(o_t/X_t^{(i)})}{\sum_{\Phi_t} p(o_t/X_t^{(j)})}$ (*importance sampling*). Donde o_t corresponde a las observaciones (ver Sección 3.1.3.)

Figura 1. Algoritmo Recursivo para MCL.

2.3. Monte Carlo Dual

Ahora se presentará una versión alternativa de MCL, llamada Localización Dual de Monte Carlo (MCL Dual). La idea central de MCL Dual es invertir el proceso de muestreo. MCL genera las nuevas partículas según la observación más reciente y luego ajusta los factores de importancia según la creencia anterior $Bel(x_{t-1})$. Por esta razón, el algoritmo dual tiene fortalezas y debilidades complementarias a las de MCL: es ideal para sensores altamente precisos pero es muy sensible al ruido en las mediciones. En esta aproximación, la ecuación recursiva de los filtros de Bayes (1), es interpretada como:

$$p(X_t^{(i)}/\Phi_{t-1}, a_{t-1}, o_t) = \frac{p(o_t/X_t^{(i)})}{\sum_{\Phi_t} p(o_t/X_t^{(j)})} \quad (4)$$

$$\omega_t^{(i)} = \sum_{\Phi_{t-1}} p(X_t^{(i)}/X_{t-1}^{(j)}, a_{t-1}) \quad (5)$$

En este método, $p(X_t^{(i)}/X_{t-1}^{(j)}, a_{t-1})$ necesita ser calculado explícitamente, a diferencia de MCL donde sólo se necesitaba un método de muestreo que siguiera $p(X_t^{(i)}/X_{t-1}^{(j)}, a_{t-1})$. Este cálculo es realizado proyectando $X_t^{(i)}$ a un posible predecesor $\tilde{X}_{t-1}^{(i)}$ usando la información de odometría a_{t-1} . Luego, se calcula $\omega_t^{(i)} = Bel(\tilde{X}_{t-1}^{(i)})$ usando Φ_{t-1} . Para llevar la recursión planteada en las ecuaciones (4) y (5), Monte Carlo Dual aplica a cada individuo $X_{t-1}^{(j)}$ el algoritmo mostrado en la Figura 2. La ecuación (4) es abordada en el primer paso del algoritmo, mientras que la ecuación (5) se refleja en los dos últimos pasos del algoritmo.

Este método no es de utilidad por sí solo porque es muy ruidoso y no se puede aplicar cuando no hay información observacional. Usado en conjunto con el método de Monte Carlo da como resultado un método más poderoso, llamado Mixture - Monte Carlo.

2.4. Mixture - Monte Carlo

La idea original de este algoritmo es una mejora a MCL llamada *sensor resetting* [7] y consiste en

insertar, en cada iteración, una pequeña cantidad de individuos arbitrariamente en lugares donde la información de los sensores lo indique. Esto, con el objetivo de acelerar la convergencia del método. Posteriormente, fue planteado el método de Localización Mixture-Monte Carlo (Mix-MCL) formalmente. Este algoritmo pretende heredar las fortalezas de MCL y MCL Dual “mezclando” ambos algoritmos. La idea central es que cada muestra sea generada usando uno de los dos algoritmos, siendo aleatoria la decisión acerca de cuál de ellos usar para cada muestra. Se usa una tasa de mezcla ϕ ($0 \leq \phi \leq 1$) constante, de manera que cada muestra es generada con probabilidad $1 - \phi$ usando MCL y con probabilidad ϕ usando MCL Dual.

1. $X_t^{(i)}$ es generado aleatoriamente usando la distribución $\frac{p(o_t/X_t^{(i)})}{\sum_{\Phi_t} p(o_t/X_t^{(j)})}$.
2. $\tilde{X}_{t-1}^{(i)} = X_t^{(i)} - \Delta X_{t-1}^{(i)}$. (cálculo de antecesor probable).

Donde $\Delta X_{t-1}^{(i)}$ sigue una distribución de probabilidades que se deriva de $p(X_t^{(i)}/X_{t-1}^{(j)}, a_{t-1})$ (Ver sección 3.2.2.).
3. $\omega_t^{(i)} = \sum_{\Phi_{t-1}} p(X_t^{(i)}/X_{t-1}^{(j)}, a_{t-1})$.

Figura 2. Algoritmo Recursivo para MCL Dual

2.5. Adaptive - Monte Carlo

Este método fue desarrollado por el equipo de fútbol robótico de la Universidad de Washington en RoboCup 2002 [1]. La idea central de esta aproximación es usar una combinación de dos estimadores de la *probabilidad observacional* \tilde{p} . La probabilidad observacional, definida como la probabilidad promedio de los individuos de la población según el modelo observacional, dice referencia a qué tan bien se ajustan los individuos a los datos obtenidos desde la cámara:

$$\tilde{p} = \frac{\sum p(o_t/x_t^{(i)})}{N} \quad (6)$$

Se definen dos estimadores de la probabilidad observacional. El primer estimador, \bar{p}_l , es llamado *promedio a largo plazo* de la probabilidad observacional, y el segundo, \bar{p}_s , es llamado *promedio a corto plazo* de ella. Mientras \bar{p}_l estima el nivel de ruido en el ambiente y los sensores (de cambio lento), \bar{p}_s es usado para estimar cambios rápidos en la probabilidad debidos a fallas en la estimación de la posición. El cálculo de los estimadores \bar{p}_l y \bar{p}_s es recursivo y se actualiza en cada iteración de la siguiente manera:

$$\bar{p}_s' = \bar{p}_s + \eta_s (\tilde{p} - \bar{p}_s) \quad (7)$$

$$\bar{p}_l' = \bar{p}_l + \eta_l (\tilde{p} - \bar{p}_l) \quad (8)$$

La única diferencia entre \bar{p}_l y \bar{p}_s descansa en los factores η_l y η_s , $0 \leq \eta_l \ll \eta_s \leq 1$. La diferencia de éste método con Mix-MCL es que la probabilidad ϕ de usar el algoritmo de MCL Dual es ahora variable y se calcula como:

$$\phi = \max\left(1 - v \frac{\bar{p}_s}{\bar{p}_l}, 0\right) \quad (9)$$

Notar que para que ϕ sea positivo, se debe cumplir que $\bar{p}_l > v\bar{p}_s$. En otras palabras, el parámetro v permite ajustar el umbral para $\frac{\bar{p}_s}{\bar{p}_l}$ sobre el cual se empieza a usar MCL Dual.

3. Estimación de Pose para Robots AIBO de Equipo de Fútbol Robótico UChile-1

El equipo de fútbol robótico de la Universidad de Chile UChile-1 fue creado en el año 2003 para participar en la competencia de fútbol robótico *RoboCup*, específicamente en la categoría *Four Legged*. En esta categoría compiten robots cuadrúpedos modelo SONY AIBO ERS210A, los cuales usan una cámara en su cabeza para adquirir los datos que les permiten tomar decisiones en forma autónoma. La arquitectura de software desarrollada para el equipo (ver figura 3) consta de 5 módulos principales: Visión, Localización, Estrategia de Alto Nivel, Estrategia de Bajo Nivel,

y Actuación. Una descripción detallada de cada uno de estos módulos pueden encontrarse en [9].

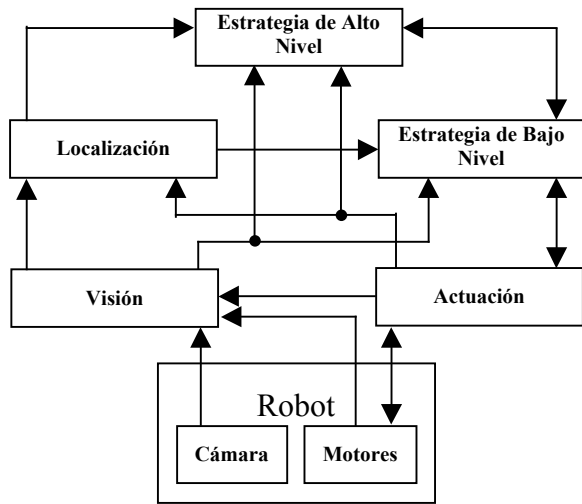


Figura 3. Diagrama de Bloques de la Arquitectura de Software de UChile-1.

El módulo de localización se implementó con distintas variantes del método de Monte Carlo. Posteriormente se eligió la de mayor rendimiento (ver sección 4 y 5). En la plataforma de desarrollo de software usada, OPEN-R, las conexiones entre los módulos se traducen en mensajes que éstos se envían entre sí. Un mensaje enviado de un módulo a otro permite ejecutar una rutina en el módulo de destino. Como se puede apreciar en la figura 3, el módulo de localización recibe información desde los módulos de visión (información observacional) y de actuación (información odométrica), por lo que el módulo debe implementarse a través de dos rutinas: una que se ejecute cada vez que se reciban datos desde visión, llamada *ciclo de visión*, y otra que se ejecute cada vez que se reciban datos de actuación, llamada *ciclo de odometría*.

3.1. Implementación de Monte Carlo

El algoritmo implementado, en forma iterativa, tiene tres pasos fundamentales, los cuales son conocidos en la literatura [5] como *resampling*, *sampling* e *importance sampling* (ver sección 2). Estos pasos permiten respectivamente:

1. Resampling: Generar una nueva población usando la población anterior.

2. Sampling: Actualizar la población usando la información de odometría; y
3. Importance Sampling: Calcular los puntajes de los individuos usando la información de visión.

3.1.1. Generación de una nueva población

En este paso, se usa el puntaje normalizado (cuya obtención será detallada más adelante) de cada individuo de la población anterior para asignarle una “probabilidad de reproducción”. A cada individuo de la población anterior se le asigna un subconjunto del intervalo $[0,1]$ de ancho igual a su puntaje normalizado. Finalmente, se sacan N números aleatorios con una distribución uniforme en $[0,1]$ y se copian los individuos correspondientes al subintervalo en que quedó cada número aleatorio, en la población nueva. Notar que como la distribución de los números aleatorios que se obtienen es uniforme, no importa el orden en que se asignen los subintervalos a los individuos. De esta forma tenemos una población nueva a partir de la población anterior (tomando en cuenta sus puntajes). La población mantiene el número de individuos constante e igual a N . Este procedimiento es conocido como *Ruleta de Monte Carlo*.

3.1.2. Uso de la información de odometría

Cada individuo de la población será desplazado siguiendo los comandos enviados al robot entre los instantes $t - 1$ y t . En general, los mensajes provenientes del módulo de actuación llegan con mayor frecuencia que aquellos provenientes del módulo de visión. Por ello, en términos de odometría, existen varios instantes de tiempo a considerar entre $t - 1$ y t :

$$a_{t-1} = \begin{bmatrix} \Delta X_{i_1} \\ \vdots \\ \Delta X_{i_t} \end{bmatrix}$$

A continuación, explicaremos la actualización de la odometría considerando que $l = 1$, es decir, que sólo existe un dato de odometría entre los instantes t y $t - 1$, la cual corresponde a ΔX_t (ver figura 4). En el caso general, el método utilizado es la repetición, en forma secuencial, para cada ΔX en

a_{t-1} , del mismo mecanismo. En una primera aproximación podemos simplificar el modelo de movimiento, de manera de hacerlo independiente del estado X_{t-1} . La simplificación consistiría en considerar:

$$p(X_t/X_{t-1}, a_{t-1}) = p(\Delta X_t/a_{t-1})$$

Donde ΔX_t correspondería al desplazamiento $X_t - X_{t-1}$. Notar que en rigor este modelo no es exacto porque el robot, dependiendo de su posición, puede encontrar mayor dificultad para hacer un determinado movimiento o puede resbalar más o menos. Por ejemplo, si está junto a una muralla. En nuestro caso, modelamos $p(\Delta X_t/a_{t-1})$ como una Gaussiana cuya media y varianza están determinadas por a_{t-1} . Cada individuo es actualizado, sumándole a su pose $X_t^{(i)}$ un desplazamiento aleatorio $\Delta X_t^{(i)}$ que sigue una distribución Gaussiana tridimensional de componentes independientes de media ΔX_t y varianza $\lambda \cdot \Delta X_t$, donde λ es el parámetro escalar que relaciona linealmente la media y la varianza de esta Gaussiana. Se supone esta relación lineal, aunque no ha sido obtenida estadísticamente, por el hecho de que mientras más se desplace o rote el robot, probablemente mayores errores cometerá en el desplazamiento y rotación efectuado.

$$X_t^{(i)} = X_{t-1}^{(i)} \oplus \Delta X_t^{(i)} \quad (10)$$

Donde $\Delta X_t^{(i)} \sim Normal(\Delta X_t, \lambda \cdot \Delta X_t)$. El operador \oplus de suma de odometrías está definido como:

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \oplus \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{bmatrix} = \begin{bmatrix} x + \Delta x \cdot \cos \theta + \Delta y \cdot \text{sen} \theta \\ x + \Delta x \cdot \text{sen} \theta + \Delta y \cdot \cos \theta \\ \theta + \Delta \theta \end{bmatrix} \quad (11)$$

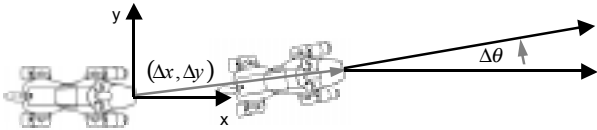


Figura 4. Interpretación de un Vector de Odometría:
 $\Delta X_t = (\Delta x, \Delta y, \Delta \theta)$.

Para comprender este operador, se recomienda revisar [4], donde se explica la actualización según odometría de la posición de salida, operación completamente análoga a ésta.

3.1.3. Uso de la información de visión

La información de visión es usada para calcular el puntaje de los individuos. El puntaje de cada individuo con pose $X_t^{(i)}$ es calculado como:

$$\omega_t^{(i)} = p(o_t/X_t^{(i)})$$

Donde o_t corresponde a un conjunto de objetos detectados por visión dentro de la imagen capturada por la cámara y sus respectivas posiciones relativas al robot en coordenadas polares. Dentro de todos los posibles objetos detectados –pelota, otros robots, faros, y arcos– (ver explicación en [9]), sólo se consideran de interés para la localización aquellos cuya posición absoluta con respecto a la cancha es conocida y fija, es decir, los faros y los arcos. Estos objetos son llamados *landmarks* (marcas de terreno). En cada instante t , el número de *landmarks* recibidos desde visión es variable y depende, entre otras cosas, de cuántos *landmarks* se encuentran dentro del área de visión del robot. En particular, es muy frecuente que o_t sea vacío (visión no identificó ningún *landmark*). $o_t = \{o_t^{(j)}\}$ corresponde al conjunto de *landmarks* recibidos desde visión. La información $o_t^{(j)}$ recibida desde visión para cada *landmark* observado, incluye una estimación de la posición relativa de dicho *landmark* (su distancia d con respecto al robot y el ángulo θ al cual se encuentra con respecto al eje del robot). Esta posición relativa es, en realidad, tratada como una variable aleatoria de distribución Gaussiana, por lo que:

$$o_t^{(j)} = (Id^j, \mu_d^{(j)}, \sigma_d^{(j)}, \mu_\theta^{(j)}, \sigma_\theta^{(j)}, \dots)$$

Por otro lado, para cada individuo $X_t^{(i)}$ es posible calcular, usando trigonometría, la posición $R_{i,j} = (d_{i,j}, \theta_{i,j})$ de cualquier *landmark* j , relativa a un sistema de coordenadas solidario a $X_t^{(i)}$. Esta posición coincide con la posición en la que el robot vería al *landmark* j si tuviera justo la pose $X_t^{(i)}$ y visión no tuviera ruido. Luego, para calcular la probabilidad $p(o_t^{(j)}/X_t^{(i)})$, se evalúa la Gaussiana bidimensional de parámetros $\mu_d^{(j)}$, $\sigma_d^{(j)}$, $\mu_\theta^{(j)}$, $\sigma_\theta^{(j)}$ en el punto $R_{i,j}$. Es decir:

$$p(o_t^{(j)}/X_t^{(i)}) = \frac{1}{\sigma_d^{(j)}\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{d_{i,j}-\mu_d^{(j)}}{\sigma_d^{(j)}}\right)^2} \cdot \frac{1}{\sigma_\theta^{(j)}\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\theta_{i,j}-\mu_\theta^{(j)}}{\sigma_\theta^{(j)}}\right)^2} \quad (12)$$

Esta probabilidad la podemos entender como el puntaje del individuo i asociado con la observación del *landmark* j , y será mayor mientras más parecidos sean los cálculos de la posición del *landmark* j , calculada con respecto a dicho individuo ($R_{i,j}$) y calculada usando la observación del robot ($\mu_d^{(j)}, \mu_\theta^{(j)}$). En nuestra implementación, la probabilidad $p(o_t/X_t^{(i)})$ del conjunto de observaciones o_t , corresponde a la probabilidad conjunta $p(\{o_t^{(j)}\}/X_t^{(i)})$ de las observaciones $o_t^{(j)}$ de cada uno de los *landmarks*. Notar que esto también es una aproximación pues no se está considerando la probabilidad de no ver los *landmarks* que no están en el conjunto o_t . Por lo tanto, la información que aporta el hecho de no ver un *landmark* se está perdiendo. Si consideramos que las observaciones $o_t^{(j)}$ son independientes entre sí, se puede calcular el puntaje del individuo como:

$$\tilde{\omega}_t^{(i)} = p(o_t/X_t^{(i)}) = \prod_{o_t} p(o_t^{(j)}/X_t^{(i)}) \quad (13)$$

3.2. Implementación de Monte Carlo Dual

En el mecanismo dual, se invierte el proceso de generación y cálculo de puntaje de los individuos. Los individuos son generados directamente usando la información visual y su puntaje es posteriormente calculado a partir de la población anterior y el modelo de odometría. De esta forma, tendrán un mayor puntaje los individuos que se ajusten más a posibles individuos de la población anterior, actualizados según odometría. Al igual que Monte Carlo, este algoritmo tiene, en cada iteración, tres pasos fundamentales:

1. Generar una nueva población usando los datos de visión.
2. Retroceder a los individuos al instante anterior, usando odometría; y
3. Calcular los puntajes de los individuos usando la población anterior.

3.2.1. Uso de los datos de visión

Los individuos son generados directamente de manera que satisfagan las observaciones de visión. Se puede desprender de lo anterior que no se puede usar MCL Dual cuando o_t es vacío, es decir cuando no hay observaciones. Cada vez que un individuo $X_t^{(i)}$ es generado, se escoge aleatoriamente una o dos observaciones del conjunto o_t , dependiendo del tamaño de dicho conjunto (cuando hay más de una observación se escogen dos de ellas). De esta manera, el individuo queda totalmente determinado en el caso de usar dos observaciones, quedando un grado de libertad libre cuando existe sólo una observación. Se podría restringir arbitrariamente el grado de libertad que queda en este caso asignándole un valor aleatorio a la variable libre que satisfaga el hecho de que el robot esté dentro de la cancha. En nuestra implementación es posible elegir entre usar o no MCL Dual cuando hay sólo un *landmark* en la información de visión. El uso de la información de visión para generar individuos cuando se recibe sólo un *landmark* desde visión es discutible ya que se generan individuos en puntos muy dispersos, sobre todo si el objeto es muy distante. Actualmente, en nuestra implementación se ha optado por no usar MCL Dual cuando se reconoce sólo un *landmark* en la imagen.

3.2.2. Uso de la información de odometría

Este paso es muy similar al paso 2 del método de Monte Carlo. Se calcula un desplazamiento $\Delta X_t^{(i)}$ de la misma forma que en 3.1.2, pero ahora se calcula un hipotético predecesor $\bar{X}_{t-1}^{(i)}$ como $X_t^{(i)} - \Delta X_t^{(i)}$. Este hipotético predecesor $\bar{X}_{t-1}^{(i)}$, no formará parte de la nueva población, sino que se usará en el paso siguiente para estimar la correspondencia entre el individuo $X_t^{(i)}$ y la población anterior.

3.2.3. Cálculo de los puntajes de los individuos

La idea es calcular el puntaje del individuo usando el modelo de movimiento. Para ello, se encuentra

el individuo $X_{t-1}^{(k)}$ de la población anterior más cercano al hipotético predecesor $\overline{X}_{t-1}^{(i)}$ y se calcula el puntaje en función de la distancia entre ambos. En nuestra implementación se usa una Gausiana centrada en $X_{t-1}^{(k)}$ y con varianza $\lambda \cdot \Delta X_t + \sigma_0^2$. La varianza mínima σ_0^2 permite que los individuos generados por MCL Dual tengan un puntaje mayor que cero, y en consecuencia, puedan sobrevivir, aún cuando el robot no se haya movido.

3.3. Implementación de Mixture Monte Carlo

La selección del método usado para cada individuo se hace en forma aleatoria, existiendo una probabilidad ϕ de usar Monte Carlo Dual y $1-\phi$ de usar Monte Carlo. Esta probabilidad ϕ es variable y depende básicamente del puntaje del mejor individuo de la población anterior. Mientras mayor sea el puntaje del mejor individuo de la población anterior, menor será ϕ . Esto de tal forma de darle mayor estabilidad al método cuando está bien localizado y rapidez a la convergencia en caso contrario. Además existe un umbral para el puntaje del mejor individuo sobre el cual no se usa MCL Dual. Por otro lado, si no hay *landmarks* observados, no se usa MCL-Dual por las razones antes explicadas.

3.4. Implementación de Adaptive Monte Carlo

La implementación de este método es muy similar a la de Mix-MCL. La única diferencia con respecto a éste se manifiesta en el parámetro ϕ . Ahora, se usa la ecuación (9) para calcular dicho parámetro. De igual forma que en Mix-MCL, no se usa MCL Dual si el puntaje del mejor individuo supera un umbral o si no hay *landmarks* observados.

4. Resultados

4.1. Implementación y Pruebas

Se midió y comparó el rendimiento de los métodos de Monte Carlo, Mixture Monte Carlo y Adaptive

Monte Carlo. Dual Monte Carlo solo se utilizó dentro de los métodos Mixture y Adaptive.

Para medir el rendimiento de estos métodos se utilizaron los parámetros planteados en [5], es decir, el error medio en la estimación de la posición para el problema de seguimiento de la posición, y la rapidez con que el método recupera su localización en el problema del raptó. Se entiende por raptó o secuestro una situación en que el robot es arbitrariamente movido de su posición. Este experimento estándar permite medir la rapidez con la que el método de localización usado por el robot puede volver a localizarse adecuadamente (converger). La imagen captada por la cámara del robot fue transmitida a un PC para poder ejecutar en él los tres algoritmos de localización simultáneamente. Para las pruebas realizadas se usaron los parámetros mostrados en la tabla 1.

Tabla 1. Parámetros usados en las pruebas.

Parámetro	MCL	Mix-MCL	A-MCL
Número de Individuos (N)	100	100	100
Probabilidad de usar MCL Dual (ϕ)	-	0,1	-
Factor de Varianza (λ)	0,1	0,1	0,1
Factor de Suavidad de Corto Plazo (η_s)	-	-	0,1
Factor de Suavidad de Largo Plazo (η_l)	-	-	0,001
Umbral para usar MCL Dual (v)	-	-	2

No se realizó un afinamiento de los parámetros anteriormente mostrados, los valores de varios de ellos son los mismos usados en [5].

4.1.1. Prueba de Seguimiento de la Posición

La idea de esta prueba es medir qué tan preciso es cada uno de los métodos para resolver el problema de seguimiento (*tracking*) de la posición, que es el

que hay que resolver la mayor parte del tiempo durante un juego. En ella, el robot se dejó navegar libremente por la cancha durante 10 minutos, en una situación de juego normal, sin raptos. Para estimar la precisión de cada método, se consideraron dos parámetros: error medio en la estimación de la posición y error medio en la estimación del ángulo de orientación. Estos errores medios se calculan como:

$$\overline{e_{pos}} = \frac{e_{pos,t}}{T} \quad (14)$$

$$\overline{e_{\theta}} = \frac{e_{\theta,t}}{T} \quad (15)$$

Donde $e_{pos,t}$ corresponde a la distancia euclidiana entre la posición de referencia y la posición entregada por el método, y $e_{\theta,t}$ corresponde a la diferencia entre el ángulo de orientación de la referencia y el entregado por el método. Es decir:

$$e_{pos,t} = \left\| (x_{ref}, y_{ref}) - (x_{met}, y_{met}) \right\| \quad (16)$$

$$= \sqrt{(x_{ref} - x_{met})^2 + (y_{ref} - y_{met})^2}$$

$$e_{\theta,t} = \left| \theta_{ref} - \theta_{met} \right| \quad (17)$$

Los resultados obtenidos se presentan en las tablas 2 y 3.

Tabla 2. Error en la posición de cada método con respecto a la referencia.

	MCL	Mix-MCL	A-MCL
Error Medio (cm)	32,7	36,1	43,2
Desviación Estándar del Error (cm)	17,6	21,0	24,5
Error Máximo (cm)	87,0	87,5	190,8
Error Mínimo (cm)	2,3	0,9	3,9

4.1.2. Prueba de Rapto

La idea de esta prueba es medir qué tan rápido es cada uno de los métodos para recuperarse de un rapto, problema que se debe resolver pocas veces en el juego (en el fútbol robótico cuando un robot comete una falta es sacado de la cancha por 1 minuto por el juez), pero que es vital hacerlo bien

porque mientras más rápido se localice el robot más rápido podrá empezar a actuar correctamente dentro de la cancha. En esta prueba, se realizaron repetidas realizaciones del siguiente experimento: Se dejó al robot localizarse correctamente en una posición arbitraria de la cancha, y luego, éste fue transportado rápidamente por un operador hacia otro lugar de la cancha, sin que el robot fuera informado de ello.

Tabla 3. Error en el ángulo de cada método con respecto a la referencia.

	MCL	Mix-MCL	A-MCL
Error Medio	32,7°	36,1°	43,2°
Desviación Estándar del Error	24,0°	23,4°	26,0°
Error Máximo	87,0°	87,5°	190,8°
Error Mínimo	2,3°	0,9°	3,9°

Tabla 4. Número de iteraciones necesarias para converger a una posición aceptable después de un rapto.

Radio de la Vecindad Aceptable (cm)	MCL	Mix-MCL	A-MCL
100	22	1	1
75	26	2	1
50	30	4	5
25	35	6	Más de 35
10	Más de 35	13	Más de 35

Luego, para cada método, se midió el número de iteraciones que el método demoró en converger a una pose aceptable. Una pose aceptable está definida como una pose que pertenezca a una vecindad, en el espacio de estados, de la pose de referencia. Esta vecindad es llamada *vecindad aceptable* y en nuestra prueba es un círculo de radio variable. Los resultados obtenidos se muestran en la tabla 4.

4.2. Análisis de Resultados

Tal como se esperaba, el método de Monte Carlo (MCL) demostró ser eficaz resolviendo el problema del seguimiento de la posición (tablas 2 y 3) pero su lenta respuesta frente a los raptos lo hacen insuficiente para resolver el problema del robot secuestrado. Por otro lado, los métodos Mix-MCL y A-MCL tuvieron un desempeño un poco inferior a MCL en la prueba de seguimiento de la posición. Sin embargo, su desempeño fue muy superior en la prueba de raptos. Una posible explicación del hecho de que MCL haya tenido un desempeño un poco superior a los otros dos métodos en las pruebas realizadas es el hecho de que la posición de referencia fue calculada usando este mismo método. Por esta razón, es posible que la posición de referencia se haya movido más lentamente que el robot en algunos instantes, lo cual haya hecho que ella se mantuviera cerca de la posición arrojada por MCL y lejos de la posición real del robot. Otra explicación para el mejor desempeño de MCL en la prueba de seguimiento de la posición está en el nivel de ruido del sensor de visión, el cual es alto para las condiciones de iluminación y la implementación del módulo de visión existentes al momento de la prueba. Mix-MCL y A-MCL funcionan mejor con sensores más precisos, a diferencia de MCL que tiene mejor desempeño con sensores con un nivel de ruido mediano. Por otro lado, de la tabla 2 vemos que el error medio en la prueba realizada se encuentra entre 30 y 40 centímetros aproximadamente para los métodos implementados. Como una posible explicación para este nivel de error, debemos mencionar el hecho de que la información recibida desde el módulo de visión tiene un alto nivel de ruido y muchas veces tiene errores que alcanzan hasta 100 centímetros. El desempeño de los métodos puede, sin embargo, ser mejorado para funcionar con niveles arbitrarios de ruido de los sensores, ajustando para ello los parámetros de cada método. Finalmente, en la tabla 4 vemos que MCL demora una gran cantidad de iteraciones (y por lo tanto, mucho tiempo) para converger a posiciones aceptables incluso con vecindades aceptables de radios muy grandes. Por ejemplo, MCL se demoró 22 iteraciones, aproximadamente 7 segundos, en converger a una vecindad radio igual a 1 metro. Este desempeño es totalmente insatisfactorio,

sobre todo si se compara con el mostrado por Mix-MCL, el cual demora muy pocas iteraciones en converger a vecindades aceptables, incluso aquellas de menor radio. Por ejemplo, Mix-MCL demoró sólo 6 iteraciones, aproximadamente 2 segundos, en converger a una vecindad de radio igual a 25 centímetros. A-MCL por su parte demoró muy poco en converger a vecindades de radio grande (hasta 50 centímetros), pero no convergió, en el número de iteraciones consideradas en la prueba, a vecindades más pequeñas. Este último hecho junto con el error medio calculado para A-MCL en la prueba de seguimiento de la posición permiten intuir que los valores de los parámetros η_s , η_s y ν hacen que el método sea muy reactivo frente a los datos de visión, y por lo tanto no funcione bien con los niveles de ruido que estos presentan. Si esta presunción fuese cierta, un ajuste en dichos parámetros de este método podría permitir una mejora en ambas pruebas.

5. Conclusiones

En base a las pruebas realizadas sobre los tres métodos implementados podemos decir que el desempeño de ellos fue similar en la prueba de seguimiento de la posición. MCL mostró una leve superioridad en dicha prueba sobre Mix-MCL y A-MCL, la cual fue atribuida al ruido en los sensores y a la metodología usada para la comparación. En la prueba de raptos, el desempeño de MCL fue muy inferior al de Mix-MCL para todos los radios de las vecindades aceptables. Por su parte, A-MCL funcionó muy bien en las vecindades de radios grandes, pero no convergió nunca a las vecindades pequeñas. A pesar de esto este último método fue seleccionado para ser utilizado en el módulo de localización del equipo de fútbol robótico UChile-1.

Como perspectivas de mejoramiento de los métodos de localización implementados, debemos destacar, el afinamiento de los numerosos parámetros usados por cada uno de los métodos implementados. En particular, parecen ser muy susceptibles de mejorar los parámetros de A-MCL. Una posibilidad para realizar esta tarea es la evolución mediante algoritmos genéticos de los parámetros de los métodos de localización, similar al trabajo ya realizado para evolucionar los

parámetros de reconocimiento visual [11]. Es posible también mejorar la forma en que está implementado el algoritmo MCL Dual. A modo de ejemplo, se puede mejorar la forma en que se calculan los puntajes de los individuos generados usando éste método, lo cual tiene un efecto de gran importancia en el desempeño de Mix-MCL y A-MCL, ya que este puntaje es el que determinará la probabilidad de sobrevivencia de los individuos agregados según los datos de visión. Una posible expansión de las funciones del módulo de localización es el cálculo de las velocidades de los objetos, lo cual tiene gran importancia en el caso de la pelota, ya que muchas de las decisiones que toma el robot tienen relación con la posición de la pelota, pero estas decisiones podrían ser más acertadas si se contara con un estimador de la posición futura de ella. Un ejemplo extremo es la decisión que debe tomar el arquero cuando ve que la pelota se acerca en dirección hacia su arco. Es natural que este cálculo sea realizado en el módulo de localización ya que es él quien tiene más información con respecto a las posiciones de los objetos, porque comparte información con los otros robots y es el que maneja las posiciones absolutas de los objetos con respecto a la cancha. Finalmente, sería de gran contribución para el desempeño de los algoritmos implementados el mejoramiento de la cantidad y calidad de la información recibida desde visión. Por ejemplo, mejorar el cálculo de distancias de *landmarks* y arcos a distancias grandes y poder detectar las líneas blancas existentes sobre la cancha. Existen en la literatura varias aproximaciones a las mejoras anteriormente mencionadas. Por ejemplo, el *German Team* ha desarrollado un sistema de localización basado en la detección de las líneas de la cancha [8].

Agradecimientos

Esta investigación ha sido financiada por el proyecto FONDECYT 1030500.

Referencias

[1] Z. Crisman, E. Curre, C. Kwok, N. Ratliff, L. Tsybert, D. Fox, "Team description: UWHuskies-02", in G. Kaminka, P. Lima, and R. Rojas, editors, *RoboCup-2002: Robot Soccer*, Springer, 2003.

[2] G. Dudek, and M. Jenkin, *Computational Principles of Mobile Robotics*, Cambridge University Press, 2000.

[3] D. Fox, S. Thrun, W. Burgard, and F. Dellaert, "Particle filters for mobile robot localization", in A. Doucet, N. de Freitas, and Gordon. N., Editors, *Sequential Monte Carlo Methods in Practice*, pages 499–516. Springer Verlag, 2001.

[4] P. Guerrero. "Localización de un robot móvil usando información visual obtenida desde una cámara móvil", Memoria de Ingeniero Civil Electricista, Universidad de Chile, 2003.

[5] J.-S. Gutmann and D. Fox, "in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems - IROS'02*, 2002.

[6] M. Isard, A. Blake, "Condensation: Conditional density propagation for visual tracking", *Int. J. Comput. Vision* 29 (1), pages 5–28, 1998.

[7] S. Lenser, M. Veloso, "Sensor resetting localization for poorly modeled mobile robots", *Proc. IEEE International Conference on Robotics and Automation - ICRA-2000*, San Francisco, CA, 2000.

[8] T. Röfer and M. Jünger, "Fast and robust edge-based localization in the Sony four-legged robot league", *RoboCup 2003 International Symposium*. To be published in *Lecture Notes in Artificial Intelligence*, Springer, 2004.

[9] J. Ruiz-del-Solar, J. Zagal, P. Guerrero, P. Vallejos, C. Middleton, X. Olivares, "UChile1 Team Description Paper", *Proc. of the RoboCup 2003 International Symposium*.

[10] S. Thrun, D. Fox, W. Burgard, F. Dellaert, "Robust Monte Carlo localization for mobile robots", *Artificial Intelligence* 128, pages 99–141, 2001.

[11] J. Zagal, J. Ruiz-del-Solar, P. Guerrero, and R. Palma, "Evolving Visual Object Recognition for Legged Robots", *RoboCup 2003 International Symposium*. To be published in *Lecture Notes in Artificial Intelligence*, Springer, 2004.