VNIVERSIDAD NACIONAL
AVTNOMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

Localización de Viterbi basada en Odometría, Redes Neuronales Artificiales y Sensores Láser para Robots Móviles

T E S I S

QUE PARA OBTENER EL GRADO DE:

DOCTOR EN INGENIERÍA

(COMPUTACIÓN)

P R E S E N T A:

ADALBERTO HERNÁNDEZ LLARENA

DIRECTOR DE TESIS:

DR. JESÚS SAVAGE CARMONA

México, D.F. **2012**

Localización de Viterbi basada en Odometría, Redes Neuronales Artificiales y Sensores Láser para Robots Móviles

Adalberto Hernández Llarena

APROBADO:

Dr. Angel Kuri Morales	Dr. Boris Escalante Ramírez
Dr. Fernando Arámbula Cosio	Dr. Carlos Rivera Rivera

DIRECTOR DE TESIS:

Dr. Jesús Savage Carmona

A ese nuevo pequeño ser que acaba de llegar.

Agradecimientos

Al Dr. Jesús Savage Carmona por dirigir mi trabajo, por compartir su experiencia en el campo de la robótica y por su mente abierta a las nuevas ideas.

A los doctores Angel Kuri y Boris Escalante por sus invaluables recomendaciones durante el desarrollo del presente trabajo. También al Dr. Fernando Arámbula, coordinador del Posgrado en Ciencia e Ingeniería de la Computación en la UNAM, por apoyar al equipo de futbol robótico humanoide pUNAMoids desde el 2008 y mi proyecto doctoral.

A las autoridades del Posgrado en Ciencia e Ingeniería de la Computación, por el excelente nivel alcanzado por el programa, tanto en lo académico como en lo administrativo.

Al Consejo Nacional de Ciencia y Tecnología CONACyT por solventar mis estudios doctorales.

A Springer-Verlag por publicar mi trabajo.

A Lulú, Diana y Amalia, por su apoyo y entusiasmo.

A las autoridades de la Facultad de Ingeniería por su apoyo y facilidades para desarrollar este proyecto.

A Alejandra Sánchez, Mauricio Matamoros, Francisco Dorantes y Marco Negrete por su apoyo al desarrollar el robot ARTURito. También a Israel Figueroa por sus consejos y sugerencias.

A mi esposa Betty por su apoyo incondicional y a mi hija Naomi por sus cariños.

A mis padres por fomentar mi sed de conocimiento.

A mi mismo, por aceptar este reto y

A Dios por permitirme concluirlo.

Adalberto Hernández Llarena
México, D.F.
Diciembre 2011

Índice

Lista de Tablas

Lista de Figuras

Glosario

El presente trabajo aborda diversos conceptos descritos y planteados por sus autores inicialmente en idioma inglés, cuya traducción literal al castellano puede resultar incorrecta o vaga en términos del sentido original que posee cada concepto. A continuación se especifican los términos o conceptos cuya traducción literal pudiera resultar confusa y que se incluyen dentro del trabajo sin realizar traducción alguna.

Lógica Crisp : Lógica que sólo acepta valores verdaderos o falsos.

Fuzzy Logic : Lógica "borrosa" o "difusa". El valor de las proposiciones o variables lógicas puede ir desde un valor mínimo hasta un valor máximo.

Fuzzyfier : Método utilizado para convertir valores *crisp* en valores borrosos.

Defuzzyfier: Método utilizado para convertir valores borrosos en valores *crisp*.

Feed-forward: Referente a las redes neuronales progresivas, donde las unidades o neuronas actualizan su valor desde las capas de entrada, hacia las capas de salida. Ninguna unidad posee realimentación consigo misma ni con las demás unidades de su capa. Tampoco conecta sus salidas con alguna(s) unidad de una capa inferior (más próxima a las entradas).

Feedback: También se le llama retroalimentación, se presenta cuando la salida de alguna neurona se conecta a la entrada de sí misma, o a la entrada de alguna(s) neurona de una capa inferior (más próxima a las entradas).

Cluster: Conjunto de puntos o vectores en torno a un núcleo o centroide, que para efectos de cálculo son tomados como una unidad de dimensiones fijas.

Clustering: Proceso mediante el cual se encuentran los grupos o *clusters* correspondientes a un conjunto de vectores o puntos en el plano.

Adaline: Proviene de Adaptive Linear Element, uno de los primeros modelos de neurona artificial.

Dead Reckoning: Estimación de la posición utilizando exclusivamente la odometría interna del robot, sin tomar en cuenta información del mundo exterior.

SLAM: Siglas de Simultaneous Localization And Mapping, problema que implica para un robot móvil moverse dentro de su entorno y al mismo tiempo estimar su posición y elaborar un mapa de su entorno.

VQ: Siglas de *Vector Quantization*. Proceso mediante el cual se representa un conjunto de patrones o vectores con cierta proximidad espacial, mediante un solo vector o prototipo de clase.

Capítulo 1

Introducción

1.1 Motivación

Una de las tareas fundamentales en la robótica móvil es sin duda la de localizarse dentro del entorno de operación del robot. Diversos métodos se han desarrollado para tales fines como el método de localización de *Monte Carlo* [Dellaert 99] y el llamado *Filtro de Kalman* [Kalman 60]. En este tipo de métodos la posición y orientación del robot se representa en forma de distribuciones de probabilidad que se actualizan conforme el robot se mueve y toma lecturas. Otros métodos [Lu 97] utilizan las propias lecturas de los sensores del robot para estimar el diferencial en el desplazamiento del robot, cotejando cada nueva lectura contra una transformación de la lectura anterior (siempre que dicha transformación pueda ser establecida).

El problema de la autolocalización [Thrun 05, p. 195] puede ser visto en 3 diferentes niveles: a) Localización Global (*Global Localization*): Implica que un robot pueda ubicarse dentro de un ambiente previamente conocido (sin información previa sobre su posible ubicación), b) Rastreo de Posición (*Position Tracking*): Consiste en calcular la posición en la que se encuentra el robot en función de su(s) posición(es) previa(s), los comandos de control y las mediciones (observaciones) realizadas durante su recorrido y c) Secuestro del Robot (*Robot Kidnapping*): El robot es repentinamente sustraído del ambiente y colocado en una nueva posición, sin haber realizado lectura alguna durante el desplazamiento, entones su tarea será localizarse de nuevo.

Históricamente, dos tipo de métodos han sido desarrollados para resolver el problema de la localización, las asociaciones *datos-datos* o *datos-modelo*.

Las asociaciones *datos-datos* estiman la ubicación del robot buscando la observación más parecida a la observación actual percibida con los sensores del robot dentro de una extensa base de datos de observaciones previas realizadas en ubicaciones conocidas. Aquella observación dentro del banco de datos que resulte más similar a la observación actual determina la posición del robot.

Las asociaciones *datos-modelo* calculan la ubicación del robot mediante una combinación de hipótesis de localización que incorporan un modelo o "mapa" del ambiente de operación del robot, un modelo de movimiento que determina las posibles secuencias de observaciones de acuerdo a las capacidades motoras del robot y un modelo de observación que relaciona ubicaciones físicas con observaciones supuestas para dichas posiciones. A menudo se utiliza el mapa para calcular en tiempo real la observación supuesta evitando el uso de extensas bases de datos o una toma exhaustiva de muestras en el entorno. Dentro de este rubro se encuentran la mayoría de los métodos que han proporcionado los mejores resultados, gozando de gran popularidad los métodos probabilísticos.

Ambos tipo de asociaciones requieren de un amplio conocimiento previo del entorno (bajo la forma de alguna representación o mapa métrico o bien mediante un conjunto extenso de toma de lecturas por parte del robot), sin embargo debemos recordar que la localización sólo es parte de un problema mucho más complejo denominado SLAM o Simultaneous Localization and Mapping, el cual plantea el problema de que un robot móvil sea capaz de construir una representación y localizarse dentro de la misma, conforme se mueve por el ambiente y recolecta información con sus sensores.

Existen diversos métodos para resolver el problema de la localización, gozando de gran popularidad los métodos probabilísticos como el método de localización de Monte Carlo [Dellaert 99]. En este tipo de métodos la posición y orientación del robot se

representa en forma de distribuciones de probabilidad que se actualizan conforme el robot se mueve y toma lecturas. En general los métodos probabilísticos relacionan la posición del robot con los datos obtenidos con los sensores del mismo, para cada ubicación posible dentro del ambiente. Una vez establecida dicha relación, se trata de inferir la posición del robot con base en las lecturas de los sensores cuando se desconoce la ubicación exacta. Lo anterior se conoce como Filtros Bayesianos [Thrun 03, p. 56]. Dentro de este tipo de filtros es posible mencionar el llamado Filtro de Kalman [Kalman 60] que permite estimar la posición del robot con base en las mediciones de sus sensores, sin embargo este método impone serias restricciones en cuanto a la distribución probabilística del error en las mediciones de los sensores, principalmente de tipo sonar o láser. Cabe destacar que, no obstante el problema SLAM se ha resuelto con los métodos antes citados (utilizando modelos probabilísticos de los sensores para generar una lectura estimada que luego es comparada con la medición real). Por otro lado, el uso de técnicas visuales como único medio de percepción está aún lejos de ser resuelto [Thrun 01]. SLAM con visión o vSLAM como se le conoce, permite a los robots móviles prescindir de costosos sensores como los láseres o sensores imprecisos como los sonares, permitiendo su uso en equipos de tamaño reducido [Folkesson 05].

En años recientes, con el auge de las Redes Neuronales Artificiales [Haykin 96] una nueva gama de métodos han sido implementados utilizando las capacidades de aprendizaje y generalización de las mismas, sin embargo no se ha explotado su capacidad de aprendizaje al máximo y su desempeño y complejidad no ha alcanzado la de los métodos probabilísticos tradicionales.

Es bien sabido que las Redes Neuronales Artificiales son aproximadores universales de Funciones y mientras más complejos se vuelven los sistemas de locomoción y percepción en los robots móviles más difícil resulta su correcto modelado, control y desde luego predicción (en el caso de observaciones de tipo visual).

A diferencia de los métodos de localización probabilísticos, que poseen la ventaja de proporcionar estimaciones de tipo continuo para la posición y orientación, existe otra variedad de implementaciones que utilizan una discretización del entorno de operación. Es posible utilizar la teoría de los Filtros Bayesianos para estimar la ubicación del robot, dentro de dicho espacio discreto de posiciones o estados, donde cada estado corresponde a una posición representativa (por lo regular el centroide) de un conjunto de posiciones aledañas a dicho nodo o estado (como en las Regiones de Voronoi [Du 99] o en los Mapas de Ocupación Espacial [Elfes 89]). Utilizar una representación discreta para representar la ubicación del robot permite simplificar el proceso de búsqueda de la posición, sin embargo requiere asociar un conjunto de lecturas en posiciones similares con cierta posición exclusiva (el centroide o nodo), por lo que métodos como la Cuantificación Vectorial [Brío 02, p. 106] resultan de utilidad tanto para encontrar los centroides o nodos como para encontrar una representación similar entre lecturas en forma de vectores tipo o característicos, que permiten reducir la dimensionalidad de los vectores de entrada (lecturas de los sensores) a una sola etiqueta o símbolo (identificador o índice asociado a cada vector tipo).

1.2 Descripción de la Problemática

Si se posee una serie de nodos o estados discretos y una serie de clases o símbolos (observaciones) y existe una asociación directa entre cada estado con los símbolos observados, entonces es posible analizar el problema de la localización como un *Modelo de Markov de Variable Oculta* [Rabiner 89] donde la variable oculta es la posición o nodo que se desea determinar, siendo posible utilizar métodos como el de *Viterbi* [Viterbi 67] para encontrar la variable oculta dentro del modelo, mientras que la *Cuantificación Vectorial* [Brío 02, p. 106] resulta de utilidad para obtener una representación similar entre lecturas en forma de vectores tipo o característicos, que permiten reducir la dimensionalidad de los vectores de entrada (lecturas de los sensores) a una sola etiqueta o símbolo (identificador o índice asociado a cada vector tipo).

El algoritmo de Viterbi hace uso de programación dinámica para reducir notablemente la cantidad de cálculos requeridos para encontrar la variable oculta en un Modelo de Markov. De manera general se aplica para encontrar la secuencia de estados que haya generado con mayor probabilidad una secuencia de símbolos observados. En cada instante de tiempo discreto el algoritmo calcula de forma progresiva la probabilidad de que la secuencia de observaciones que se tienen hasta ese momento, correspondiente a una serie de transiciones entre estados, tenga como destino final a cada uno de los estados del modelo y se observe el último símbolo. El estado con la mayor probabilidad corresponderá con la estimación de la posición del robot.

La diferencia del algoritmo de Viterbi radica en que, en cada iteración, se conserva exclusivamente para cada uno de los nodos o estados la secuencia más probable que lleve hacia ese estado, descartando por completo las transiciones con menor probabilidad, lo anterior supone un grave problema si no se establece de forma adecuada la probabilidad de observar cada uno de los símbolos para cada uno de los estados y la probabilidad de transición a cada uno de los mismos. En caso contrario y en un ambiente real de operación, si se observa un símbolo (debido a errores en los sensores) en cierta ubicación donde en teoría existe una probabilidad casi nula de observar dicho símbolo, la secuencia real de transición será descartada por el método y se buscará otra con mayor probabilidad. Lo mismo ocurre con las transiciones entre estados si el robot llega por errores de odometría a un estado (o *nodo* físico dentro de un mapa discreto del ambiente) con probabilidad casi nula de transición.

El problema anterior implica el establecimiento de probabilidades de observación y transición entre estados lo más precisas posible, algo realmente difícil en ambientes reales de operación sujetos a variaciones en el entorno y a errores en los sensores y en la odometría difíciles de caracterizar. También requiere realizar diversos recorridos por el ambiente real de operación para tomar muestras, generar los símbolos y observar la transición entre estados para poder calcular dichas probabilidades, proceso que realiza generalmente “fuera de línea”, es decir, posterior a la toma de muestras y mientras el robot no se encuentra en operación.

1.3 Objetivo de la Tesis

El objetivo del presente trabajo será proponer un método que permita a un robot móvil resolver tanto el problema de la localización global como el rastreo de posición y el secuestro del robot utilizando el método de Viterbi para Modelos de Markov de Variable Oculta. Dicha metodología deberá obtener las matrices de probabilidad A, B y Π del modelo a partir de una representación métrica (mapa) previamente generada. Para poder obtener la matriz B de probabilidad de observación de símbolos, el método deberá relacionar una observación supuesta para cada ubicación navegable dentro del mapa (mediante un simulador operando fuera de línea), generando previamente una representación simbólica de las observaciones realizadas utilizando alguna forma de Cuantificación Vectorial o clasificador. Se propone utilizar como sensor principal un sensor láser comercial, sin embargo el método al utilizar Cuantificación Vectorial será susceptible de utilizar información proveniente de otro tipo de sensores como cámaras web o sonares.

El algoritmo de Viterbi se utilizará para calcular la ubicación actual del robot, resolviendo el modelo de Markov de Variable Oculta , utilizando para ello la información de odometría del robot (desplazamiento estimado) y las observaciones realizadas con el sensor láser a medida que se mueve en el ambiente de trabajo.

El método se apoyará en herramientas tales como Redes Neuronales Artificiales [Haykin 96] dadas sus capacidades de entrenamiento y generalización, métodos de Cuantificación Vectorial [Brío 02, p. 106] o Mapas Autoorganizados [Kohonen 95] para la representación simbólica de las observaciones y Redes de Agrupamiento o Clustering [Brío 02, p. 125] para encontrar los nodos (estados) en el modelo.

1.4 Contribución y Relevancia

En la actualidad uno de los métodos de localización que gozan de amplia popularidad es el método de localización probabilística de Monte Carlo [Fox 99]

utilizado ampliamente en las competencias mundiales de futbol de robots RoboCup (www.robocup.org) o FIRA (www.fira.org). Su utilización se debe en gran medida a la sencillez del método, sin embargo, dado que esta basado en distribuciones de probabilidad de posición para un elevado número de posiciones aleatorias (filtro de partículas), la probabilidad de cada partícula debe ser actualizada con cada observación, lo que requiere de un elevado procesamiento cuando la cantidad de partículas es elevada (alrededor de 100,000 para ambientes de 8 × 10 m). Por otro lado requiere una estimación precisa de la observación para cada ubicación representada por las partículas, lo que implica una exhaustiva toma de muestras o una costosa simulación en tiempo real de lecturas en el caso de sensores láser o cámaras web. El uso de marcas o landmarks en el ambiente para su correcta operación si bien puede ayudar simplificar un poco el tiempo de proceso, supone un problema adicional en ambientes desconocidos o para aplicaciones sin visión a bordo.

Por otro lado, el uso de localización discreta (localización de Markov) posee la desventaja de requerir con exactitud probabilidades de observación de símbolos y de transición entre estados, además de definir con anterioridad el tipo de discretización del ambiente o estados. El error en la localización de Markov es proporcional a la distancia Euclidiana entre estados (ubicaciones fijas del robot). Mientras más cercanos se encuentran los nodos (o estados que representan una ubicación del robot) más pequeño será también el error de localización. Cuando se define una distancia entre nodos pequeña con respecto a las dimensiones del ambiente de navegación, se incrementa notablemente el número de estados del modelo y con ello los cálculos requeridos (de forma análoga a lo que sucede en el filtro de partículas al aumentar su número).

La principal ventaja que supondría el utilizar un método discreto para calcular una ubicación continua del robot radica desde luego en la rapidez de operación. De esta forma se evitaría calcular con precisión la observación para cada ubicación supuesta por el método de localización. Al tratarse de nodos (estados) con ubicación fija en el ambiente, dicha observación es susceptible de ser generada artificialmente (o "simulada") con anterioridad o inclusive realizar una toma directa con el sensor en dichas ubicaciones.

Por otro lado, dado que el discretizar una información continua como lo es la ubicación del robot supone una pérdida de precisión, el presente trabajo tendrá como objetivo analizar las relaciones entre aquellos estados discretos (ubicaciones) con alta probabilidad, para proporcionar una estimación de posición continua que no se centre exclusivamente en el estado con mayor probabilidad luego de la incorporación de la observación más reciente, a diferencia del Método clásico de Viterbi.

En cuanto a la implementación, es bien sabido que una de las formas menos costosas en términos computacionales y más eficientes de las Redes Neuronales Artificiales es en forma de operaciones matriciales, involucrando el vector de entradas, las matrices de pesos de las distintas capas y funciones que se aplican sobre las matrices resultado (como la función escalón o lógica sigmoidal para obtener el nivel de activación). El método propuesto propone el uso de dichas redes, lo que facilita y posibilita su implementación en microprocesadores vectoriales o DSP (Digital Signal Processors) para obtener una máxima capacidad de cómputo, siendo susceptible de ser integrado a los robots existentes en forma de un dispositivo externo que les proporcione una estimación de la posición sin requerir de costosos cálculos en el procesador central del robot.

Finalmente, la localización utilizando exclusivamente visión computacional es un tema aún no resuelto por la dificultad de reconocer y clasificar observaciones realizadas con una cámara montada sobre el robot, principalmente por cuestiones de escala, rotación, traslación y obstrucción. Dado que el método propuesto resulta independiente del tipo de sensor utilizado, dicho método es susceptible de ser implementado directamente sobre una plataforma DSP o FPGA y facilitar enormemente las tareas de localización cuando se utiliza exclusivamente información visual.

El rango de aplicaciones prácticas para un método o un dispositivo como el mencionado anteriormente van desde el campo de la robótica móvil, hasta aplicaciones automotrices, industriales y militares.

1.5 Vista General de la Tesis

El capítulo 2 nos da una breve introducción al problema de la localización robótica, indicando los métodos más utilizados en la actualidad y mostrando brevemente el Estado del Arte en localización robótica.

El capítulo 3 proporciona las bases necesarias para entender el problema de la localización robótica desde el punto de vista probabilístico, que es el enfoque que tradicionalmente ha dado los mejores resultados y en el que se basa el presente trabajo.

El capítulo 4 muestra los principales métodos actuales de localización robótica que utilizan Redes Neuronales Artificiales y sus inconvenientes.

El capítulo 5 muestra un análisis comparativo de los diferentes métodos de localización robótica presentados en los capítulos introductorios, realizado mediante la implementación de algunos de ellos y probados en un simulador. Dicho análisis sirvió de base para establecer las características "deseables" de todo método de localización robótica.

El capítulo 6 presenta el método de Localización de Viterbi basado en Odometría y Redes Neuronales Artificiales, incluyendo fundamentos de la Localización de Markov, que constituye el punto de partida para entender la operación del Algoritmo de Viterbi.

En el capítulo 7 se discute el método propuesto en el presente trabajo: las pruebas realizadas, los resultados obtenidos, sus principales ventajas y desventajas, mientras que el capítulo 8 muestra las conclusiones sobre el método y los trabajos futuros.

Finalmente, en el Apéndice A se incluye el artículo in extenso tema de esta investigación ya publicada, con el amable permiso de Springer Science+Business Media: Journal of Intelligent & Robotic Systems, DOI: 10.1007/s10846-011-9627-8, mientras que el Apéndice B muestra el uso del software desarrollado rSLAM.

Capítulo 2

Localización Robótica

2.1 Introducción al Problema de la Localización Robótica

SLAM (Simultaneous Localization and Mapping) [Thrun 03] es un término que se aplica a las diferentes acciones que un robot móvil debe realizar para poder explorar su entorno de manera autónoma con sus sensores, obtener conocimiento acerca de éste, interpretar la escena, construir un modelo apropiado y localizarse a sí mismo en relación a dicho modelo o mapa de manera simultánea.

El problema de la autolocalización [Thrun 05, p. 195] puede ser visto en 3 diferentes niveles:

a) Localización Global.- Implica que un robot pueda ubicarse dentro de un ambiente previamente conocido, sin información previa sobre su posible ubicación.

b) Rastreo de Posición (Position Tracking).- Consiste en calcular la posición en la que se encuentra el robot en función de su(s) posición(es) previa(s), los comandos de control y las mediciones (observaciones) realizadas durante su recorrido.

c) Secuestro del Robot (Robot Kidnapping).- El robot es repentinamente sustraído del ambiente y colocado en una nueva posición, sin haber realizado lectura alguna durante el desplazamiento. Su tarea será localizarse de nuevo.

Para que un robot pueda localizar su posición y orientación dentro de un ambiente, es indispensable contar con un modelo que integre tanto información sobre

dicho ambiente, como sobre los movimientos y lecturas que el robot puede realizar. Dicho modelo servirá como base para poder inferir la posición del robot utilizando la información que el robot recopila a medida que se desplaza por su entorno.

2.1.1 Modelo de un Fenómeno

En general podemos definir un modelo como una abstracción simplificada que pretende describir los atributos principales de un fenómeno o proceso, ya sea físico, económico, social o puramente matemático.

Hoy en día el desarrollo de modelos constituye una parte fundamental en el desarrollo científico. El uso de un modelo apropiado permite experimentar con una simplificación que, si bien no es el fenómeno en sí, posee muchas de las principales características del fenómeno original sobre las que se desea experimentar, con el consiguiente ahorro de tiempo y recursos.

2.1.2 Modelos Espacio-Estado

En este tipo de modelos se introduce el concepto de *estado*, por estado se entiende un conjunto de valores que representan, en un instante de tiempo discreto *t*, información relacionada ya sea con el propio robot (como en el caso de los valores angulares de cada uno de sus motores, el contenido en su memoria o un punto en particular en su proceso de decisión) o con su entorno o *espacio de configuración* [Latombe 91] por ejemplo su posición y orientación con respecto a algún punto tomado como *origen* del marco de referencia, o las lecturas realizadas con sus sensores.

2.1.3 El Problema de la Localización Robótica

En el problema de la localización de un robot móvil, el estado del robot (concebido como su ubicación dentro del ambiente de referencia) se representa como una variable multidimensional $\overline{X}$, que representa la posición y orientación actual del robot dentro del marco de referencia de su entorno o mapa, es decir

$$\bar{X} = (x, y, z, \theta, \phi)$$

donde *x*, *y* y *z* corresponden con la posición espacial, θ con la dirección hacia la cual se encuentra orientado el frente del robot (conocida en inglés como *panning*) y ϕ como el ángulo de elevación vertical sobre el horizonte (o *azimut*) hacia el cual se encuentra dirigido el sensor (por ejemplo una cámara montada sobre una unidad giratoria sobre el robot), conocido como inclinación o *tilt* en idioma inglés.

En el caso de robots móviles que se desplazan sobre una superficie plana como el piso, es posible eliminar el término *z*, ya que se asume que su valor se mantendrá siempre constante. De igual forma para robots móviles que no cuentan con cámaras de video o unidades *pan-tilt,* es posible asumir el término ϕ como constante.

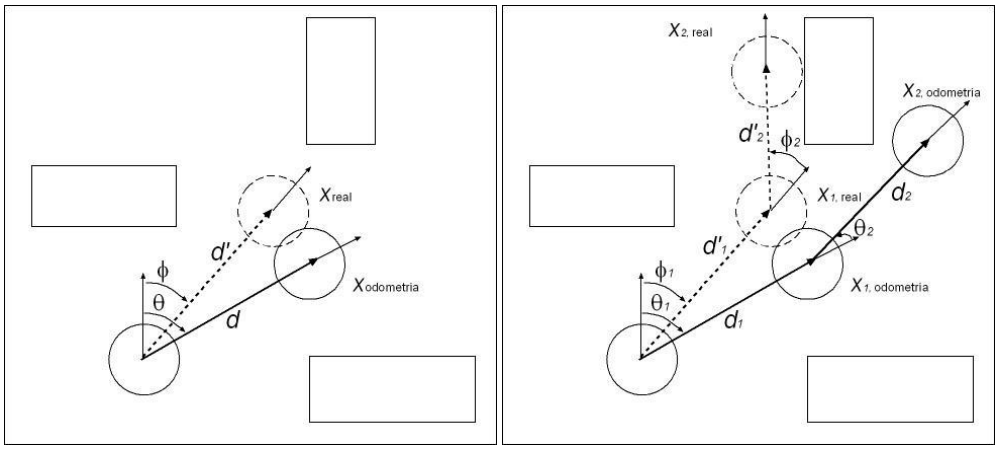


Figura 2.1. Acumulación de errores sucesivos debido a la odometría del robot

En la determinación de la localización de un robot móvil, usualmente intervienen 3 tipos de factores (Fig. 2.1):

a) El *estado* del robot: *x*(*t*)*, denotado por* x_{real} *y* $x_{odometría}$.
b) Las lecturas de sus sensores: *s*(*x*)
c) Los comandos de control: *u*(*t*) o la estimación de desplazamiento (d_t, θ_t, ϕ_t)

Como puede observarse, tanto *x* como *u* se representan en función del tiempo, lo cual indica que, tanto la posición como la información de control dependerán del instante de tiempo determinado en que deseen evaluar. A diferencia de estos, *s* se asume como una función de *x*, es decir, las lecturas de los sensores del robot dependen exclusivamente de la ubicación *x* del robot dentro del ambiente y no del paso del tiempo.

2.2 Principales Métodos de Localización Robótica

2.2.1 Métodos de Cotejamiento de Observaciones (Datos-Datos)

En este tipo de métodos las sucesivas lecturas obtenidas por el robot son utilizadas para determinar el movimiento diferencial que el robot ha realizado, con base en la transformación de dichas lecturas debido a tales desplazamientos del robot. Podemos citar [Lu 97] como un claro ejemplo de un método de ajuste de la posición (rastreo de posición) utilizando la proyección de una lectura de referencia con un sensor láser (previo al movimiento) a la ubicación indicada por la odometría. Mediante minimización por mínimos cuadrados el método obtiene un estimado de la nueva ubicación que maximiza el emparejamiento entre las dos lecturas sucesivas. Este método funciona eficientemente para el caso de sensores con poco ruido como los sensores láser (ver sección de análisis de métodos).

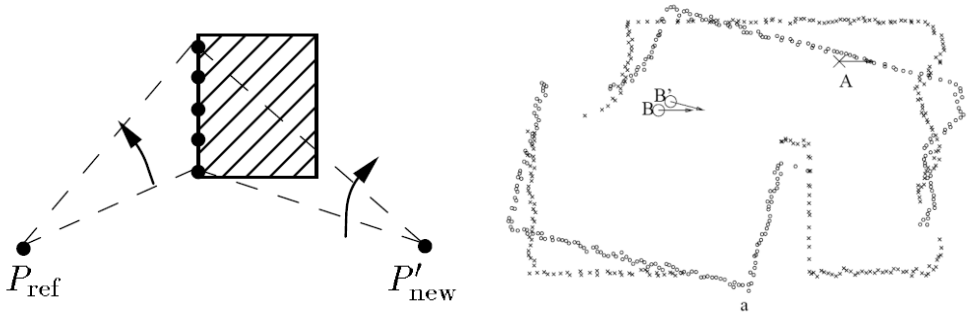


Figura 2.2. Método de alineamiento de lecturas láser por mínimos cuadrados. Tomado de [LU 97]

Por otro lado quizás el método más sencillo de obtener un estimado de la posición con respecto a la lectura más reciente consiste en realizar una toma de muestras sucesivas (utilizando algún método de cotejamiento como el anteriormente citado) y estableciendo

un conjunto de muestras que es almacenado en forma de un mapa perceptual. Una vez que se obtiene una lectura en modo de ejecución, es posible encontrar el vector más similar almacenado en el mapa, mediante distancia Euclidiana. Si junto con cada lectura se almacena la posición desde donde se originó la misma, al encontrar el valor más próximo, también se obtiene un estimado de la posición; sin embargo este método depende de que el sensor proporcione un bajo error de medición (inferior al 5%, como lo es el caso de los sensores láser) y es extremadamente sensible a oclusiones o a errores tales como los que se obtienen con sensores ultrasónicos (sonares).

2.2.2 Enfoque Probabilístico (Datos-Modelo)

Desde el punto de vista probabilístico, el problema de la localización robótica se reduce al cálculo de una función de densidad de probabilidad o *PDF* (que puede se unimodal o multimodal) que represente la ubicación X actual del robot, donde es una variable aleatoria que representa la posición y orientación del robot.

Diversas variantes se han desarrollado utilizando distribuciones paramétricas continuas (Filtro de Kalman Extendido [Joseph 04]), distribuciones discretas (Localización de Markov [Thrun 99]) y aproximaciones de distribución mediante filtros de partículas (Localización de Monte Carlo o MCL [Thrun 00]).

En dichos métodos probabilísticos, como la posición o variable X no puede ser estimada directamente (al menos completamente ya que, si bien en principio es posible utilizar un compás digital para conocer la orientación, la resolución de los Dispositivos de Posicionamiento Global o *GPS* aún no alcanzan la precisión de algunos centímetros que se requiere) se utiliza el principio de los Modelos de Markov de Variable Oculta (donde la variable oculta o estado es la posición del robot) para poder inferir el estado en función de las observaciones $Z^k = \{z_k, i = 1..k\}$ realizadas por el robot y las transiciones o comandos de control $u(t)$ dados al robot entre cada movimiento sucesivo.

Bajo este esquema la principal suposición que se hace es la llamada *Markov Assumption* o Suposición de Markov, según la cual las observaciones realizadas Z^k en un cierto estado X^k dependen exclusivamente de dicho estado, es decir

$$P(Z^k \mid X^k, X^{k-1}, ..., Z^{k-1}, ..., u(k), u(k-1), ...) = P(Z^k \mid X^k) \tag{2.1}$$

Lo cual claramente presupone que el ambiente sea estático, es decir, que lo único que se mueva dentro de él sea el propio robot. Dicho de otra forma lo anterior significa que lo que el robot percibe en cierta posición con cierta orientación depende exclusivamente de dicha ubicación en el ambiente y que además será constante y además *predecible*. La anterior presunción resulta inexacta para la gran mayoría de los ambientes en donde se desea que interactúen los robots; por esa razón los autores han propuesto diferentes alternativas para permitir su uso en ambientes dinámicos [Thrun 99b].

En la estimación de la posición del robot bajo el enfoque probabilístico se utilizan tres tipos de modelos:

1) Modelo de Sensor.- Permite predecir la lectura que proporcionará el sensor. Integra de igual forma un modelo de ruido.

2) Modelo de Observación.- Asocia una o un conjunto de lecturas para cierta ubicación *X* utilizando para ello el Modelo de Sensor.

3) Modelo de Movimiento.- Permite estimar el movimiento efectuado por el robot en función de los comandos dados. Comúnmente este modelo se sustituye por el valor proporcionado por la odometría del robot o el comando de control $u(t)$ dado al robot.

En el Filtro de Kalman se asume un modelo lineal de movimiento (Modelo de Planta), un Modelo lineal de Observación y un modelo de ruido con distribución Gaussiana y media cero. Lo anterior resulta claramente inapropiado para sensores tipo láser o cámaras digitales y robots con varios grados de libertad como los robots

humanoides bípedos donde los modelos de observación y movimiento no se comportan de manera lineal.

En el caso de la localización de Markov y el filtro de partículas MCL, el Modelo de Sensor es de tipo probabilístico e integra la probabilidad de mediciones erróneas debidas tanto a errores propios del sensor como a obstrucciones del mismo por parte de objetos móviles. Su Modelo de Observación prácticamente implica la necesidad de desarrollar un simulador y coleccionar una gran cantidad de lecturas (mapa de observaciones), así como realizar búsquedas por similitud en estructuras tales como los árboles *kd* [Indyk 04] en sus versiones más eficientes. En cuanto al Modelo de Movimiento, en la gran mayoría de las implementaciones se utiliza el cálculo proporcionado por la odometría del robot, más un cierto nivel de error de tipo Gaussiano y con media cero.

Para el cálculo de la posición los métodos probabilísticos hacen uso de un esquema de actualización en dos fases:

1) Fase de Predicción.- Con base al comando de control $u(t)$ se hace una predicción de la distribución de probabilidad que indique la nueva posición del robot, sin tomar en cuenta las observaciones. De igual forma se hace una predicción de la observación Z_p^{k+1} que el robot deberá percibir en su nueva ubicación.

2) Fase de Actualización.- Se compara la nueva observación real Z_r^{k+1} con la observación supuesta Z_p^{k+1} y se actualiza la distribución de probabilidad de *X*, con base en dicha similitud.

Regla de Actualización de la Posición en el Filtro de Partículas de Monte Carlo

$$Bel(x_t) = h\, p(o_t \mid x_t) \times \int p(x_t \mid x_{t-1}, u_t) \times Bel(x_{t-1}) \times dx_{t-1} \qquad (2.2)$$

donde es el Modelo de Observación, $p(x_t \mid x_{t-1}, u_t)$ es el Modelo de Movimiento y el Modelo de Sensor viene dado por la figura 2.3:

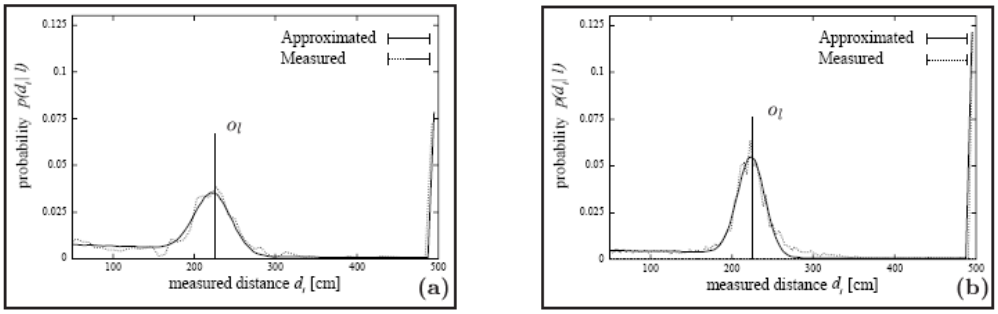


Figura 2.3 Probabilidad medida y aproximada de mediciones con (a) sonar y (b) láser, dada la distancia o_l al obstáculo más cercano en la dirección de sensado.

Para obtener primero se determina la observación supuesta s_{p_i} para cada valor individual s_i del sensor. Luego se obtiene de la gráfica anterior la $P(s_i \mid s_{p_i})$ finalmente

$$P(o_l \mid x_t) = \prod_{i=1}^{n} P(s_i \mid s_{p_i}) \qquad (2.3)$$

Nota:

Si comparamos la Ec. (2.2) con el algoritmo de Viterbi [Forney 73] para el cálculo de la Probabilidad de las Observaciones dado el Modelo $P(O \mid \lambda)$:

Procedimiento hacia adelante:

$$b_j = b_j(o_t) \times \sum_{i=1}^{n} a_{ij} \times b_i \text{, es decir} \qquad (2.4)$$

$$Bel(x_t) = p(o_t \mid x_t) \cdot \sum_{i=1}^{n} p(x_t \mid x_{t-1}) \cdot Bel(x_{t-1}) \qquad (2.5)$$

Se observa que (2.2) es el caso continuo de (2.5) para el caso de contemplar una transición entre estados del modelo dependiente de $u(t)$ ó $p(x_t \mid x_{t-1}, u(t))$.

2.2.3 Localización con Redes Neuronales Artificiales.

Diversas aproximaciones al problema de la localización robótica se han desarrollado utilizando las capacidades de generalización y aprendizaje de las Redes Neuronales Artificiales o RNAs. Algunos métodos utilizan redes de perceptrones para

asociar directamente las lecturas de los sensores del robot (también llamadas observaciones) con la ubicación del mismo dentro de un ambiente dado y con respecto a un origen de referencia con respecto a dicho ambiente. En [Choi 07] se utiliza una red de perceptrones para determinar un diferencial en el movimiento del robot, con base en la lectura real y una lectura simulada (virtual) para la posición supuesta del robot. En [Djekoune 01] se explora un método que utiliza una Red Neuronal de Regresión General (GRNN) para aproximar la relación entre características de alta dimensión con los estados del robot. Este método utiliza Análisis de Componentes Principales PCA.

Un aspecto importante ha sido el uso de técnicas como el Análisis de Componentes Principales o PCA para extraer las características principales de datos multidimensionales y reducir la dimensión de los mismos. En [Crowley 98] se presenta un método para estimar la posición de un robot utilizando PCA a partir de lecturas láser. En [Vlassis 02] se utiliza PCA para reducir la dimensión de las lecturas generadas a partir de sonares. También se ha utilizado PCA en [Nayar 94] y [Artac 02] para extraer características en imágenes para la localización robótica. En [Djekoune 01] se propone un algoritmo de localización utilizando visión estereoscópica donde el problema de la correspondencia para un conjunto de características extraídas a partir de un par de imágenes estereoscópicas es formulado como una minimización de una función, lo que es realizado mediante una red de Hopfield Bidimensional.

Por su parte las Redes Neuronales Artificiales de Kohonen son conocidas por su capacidad de realizar clasificación, reconocimiento, compresión de datos y asociación de forma no-supervisada [Lippman 87]. Estas redes son utilizadas para reconocer las características o *landmarks* utilizando mediciones láser con la finalidad de proveer coordenadas de tales características para realizar triangulación en [Hu 99]. En [Janet 95] se propone un algoritmo de localización global que utiliza una Red de Kohonen Autoorganizada. Utilizando dicha red el robot es capaz de determinar la habitación en la que se encuentra con base en mediciones con sonar. En [Crowley 98] se expone un método de localización basado en RNAs que utiliza un sensor láser de tipo SICK para la toma de lecturas. Se utiliza un mapa autoorganizado para extraer características que luego

son cotejadas con las características detectadas en tiempo de ejecución como medida de error en la localización

Finalmente, como una muestra de un enfoque híbrido que utiliza el preprocesamiento tanto de las lecturas de los sensores como de la estimación de la posición podemos citar la red de tipo difuso ART o Fuzzy ARTMAP, utilizada en [Racz 94] para posicionar a un robot móvil frente a cierto objeto mediante un anillo de sonares. Utiliza aprendizaje supervisado para asociar a un conjunto de lecturas de los sonares con las coordenadas del robot, para tareas tales como traspasar una puerta.

2.3. Resumen de los Métodos de Localización Robótica

Podemos decir que los métodos de localización actuales pueden ser clasificados en tres grandes rubros por el tipo de técnicas utilizadas:

a) *Métodos de Cotejamiento entre Lecturas (asociadores datos-datos)*.- Son métodos que pretenden asociar lecturas sucesivas de los sensores con los desplazamientos relativos del robot, determinando para ello una función f tal que

$$S_{new} = f(\Delta x, \Delta y, \Delta \theta, S_{ref}) \qquad (2.8)$$

La función *f( )* transforma las lecturas de referencia S_{ref} tomadas en la posición base X_{ref} en una nueva lectura S_{new} obtenida en la nueva posición X_{new} Algunos métodos no requieren de un mapa del ambiente pero son incapaces de resolver el problema de la localización global y son muy sensibles a errores altos en los sensores (como en el caso del sonar). Otros métodos realizan el cotejamiento entre una lectura real y una lectura simulada (o previamente almacenada) para resolver el problema de la localización global o el secuestro del robot, pero resultan muy sensibles a oclusiones y a errores elevados del sensor o difíciles de caracterizar (sensores de tipo sonar). Dentro de este rubro podemos incluir aquellos métodos que comparan la lectura más reciente del sensor contra una lectura esperada para una cierta ubicación o *Sensor-Map Matching*, ya

sea que se trate de un mapa métrico y la lectura esperada se simule o se trate de una búsqueda por máxima similitud en un conjunto de lecturas obtenidas en etapas de reconocimiento del ambiente por el robot.

Este tipo de métodos resultan ideales para resolver el problema del rastreo de la posición ya que no se requiere una representación del ambiente ya sea métrica o perceptual. Por la misma razón son incapaces de resolver el problema de la localización global o el secuestro del robot.

b) *Métodos Probabilísticos (asociadotes datos-modelo)*.- Estiman una distribución de probabilidad para una variable aleatoria *X*, que represente la probabilidad de que el robot se encuentre en una posición y orientación determinada, en función de las observaciones O_t actuales y pasadas de sus sensores y los comandos de control o el desplazamiento relativo u_{t-1} actual y pasado proporcionado por su odometría:

$$p(X \mid O_t, O_{t-1}, ..., u_{t-1}, u_{t-2}, ...) \tag{2.9}$$

Dichos métodos resultan más robustos en cuanto a su tolerancia a errores en los sensores, pero se requiere de un elevado poder de cómputo y de una representación del ambiente ya sea en forma de un mapa métrico que se toma como referencia para simular un modelo de sensor o en forma de una gran colección de muestras del ambiente. Inclusive en sus implementaciones más robustas, requieren de estructuras complejas que permitan obtener una posición estimada a partir de una lectura real. En su mayoría pueden proporcionar varias hipótesis simultáneas.

c) Métodos de Localización basados en Aprendizaje.- En su gran mayoría, los métodos basados en aprendizaje estudiados pretenden realizar ya sea un cotejamiento entre lecturas o un mapeo directo de la observación de los sensores con una ubicación del robot. Algunos de ellos utilizan Redes Neuronales Artificiales (RNAs) para estimar directamente una posición a partir de la lectura más reciente del sensor (asociación directa), sin embargo resultan extremadamente sensibles al ruido o a oclusiones. Otros

métodos utilizan la posición reportada por la odometría para estimar una observación supuesta para dicha posición (en la forma de un conjunto de características o *landmarks* proporcionada por la red) y realizar un ajuste en la posición con base en el entrenamiento de otra RNA. Estos métodos de igual forma son incapaces de predecir oclusiones o errores grandes en los sensores y resultan muy sensibles a problemas en la odometría o a la falta de la misma. Por otro lado en su totalidad proporcionan una hipótesis única de localización, lo que los hace poco útiles para resolver el problema del secuestro del robot.

Finalmente los métodos neuronales basados en redes autoorganizadas no proporcionan el nivel de exactitud que proporcionan tanto los métodos de cotejamiento de lecturas como los métodos probabilísticos, por lo que su aplicación se ve restringida a la toma de decisiones de alto nivel y no a tareas tales como la navegación o evasión de obstáculos.

2.4 Características Deseables en un Método de Localización

Luego de la investigación realizada de los distintos métodos de localización es posible formular las siguientes cualidades deseables en los métodos de localización y que fomentan su uso generalizado.

En orden de importancia:

1.- Ser tolerantes a oclusiones y a errores en los sensores.
2.- Operar en ambientes dinámicos.
3.- Contemplar para el cálculo de la posición tanto la secuencia de observaciones realizadas como los movimientos efectuados por el robot (odometría o comandos), no sólo la observación más reciente.
4.- Representación multimodal de la hipótesis de localización (más de una hipótesis).
5.- Adaptabilidad de Recursos de Hardware y Software.
6.- Estimación continua de la posición.

Capítulo 3

Localización Robótica Probabilística

3.1 Introducción

En robótica móvil es posible modelar el estado del robot, las mediciones de sus sensores y la información de control mediante variables aleatorias. Una variable aleatoria es capaz de tomar distintos valores y lo hacen de acuerdo a leyes probabilísticas específicas. La inferencia probabilística es el proceso de calcular dichas leyes que se derivan de otras variables aleatorias y los datos observados.

Utilizado la convención de [Thrun 05, Cap. 1], en principio es posible modelar el estado actual del robot mediante una variable aleatoria denominada X cuya función de distribución de probabilidad indique la factibilidad de que el robot se encuentre en la posición representada por el valor de dicha variable aleatoria (Fig. 3.1), donde $\boldsymbol{x}_1$ y $\boldsymbol{x}_2$ representan vectores de posición del robot (x, y, θ) mientras que $p(\bar{X} = x)$ constituye una distribución de probabilidad de la ubicación del robot.

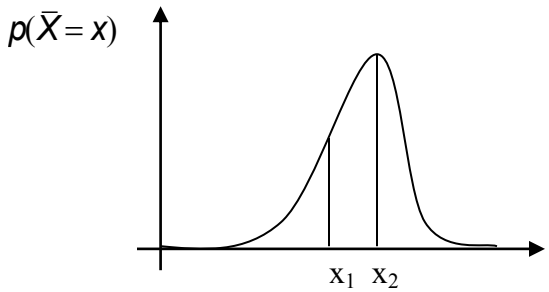


Figura 3.1 Densidad de probabilidad sobre x

Cabe aclarar que x_i no será un valor único, sino un vector que representará una *hipótesis de localización*. Por ejemplo, en la figura 3.1 es posible que

$$x_1 = (4.53, 2.25, 1.57), \quad x_2 = (5.53, 0.25, 0.71)$$

Un dato fundamental consiste en el hecho de que, al tratarse de probabilidades se debe cumplir que:

$$\int p(x)\,dx = 1 \tag{3.1}$$

en el caso de funciones de probabilidad continuas

$$\sum_{i=1}^{n} p(X = x_i) = 1 \tag{3.2}$$

en el caso de probabilidades de tipo discreto.

Lo anterior supone un problema doble

1) Cómo calcular una probabilidad multivariada que represente la estimación de la posición del robot, con base en las lecturas de sus sensores y la información de control y

2) Como asegurar que se cumplan (3.1) o (3.2) dependiendo del caso.

Si se asume la observación *s* como función de *x*, el problema de la localización puede ser considerado como de tipo *Markoviano*, es decir, que las observaciones para cierto estado *x*, dependan exclusivamente de dicho estado y cada estado represente un instante de tiempo discreto *t* (*Modelo de Markov*).

3.2 Modelos de Markov

Un modelo o cadena de Markov es una estructura de la forma

$$M := < S, s_0, R >$$

Donde *S* es el conjunto de estados del sistema, s_0 el estado inicial y *R* es una Relación de Accesibilidad que modela las transiciones entre estados como probabilidades de transición. A diferencia de los diagramas de transición de estados (como las cartas ASM), la relación de accesibilidad es de tipo probabilístico y se asume que la probabilidad de transición al siguiente estado depende exclusivamente del estado actual (propiedad de Markov o *Markov Assumption*).

Utilizando un modelo de Markov es posible evaluar la probabilidad de encontrarse en cierto estado x (que represente la posición y orientación actual del robot) en un instante determinado t.

Si el estado x es observable en todo momento, y por observable nos referimos a *cuantificable*, es decir, existe alguna forma inmediata de medirlo como un compás magnético y sensores de posición o GPS, se dice que se trata de un Modelo de Markov Observable. Por el contrario, si el estado no es cuantificable por completo (si se contara por ejemplo únicamente con el compás magnético para medir la orientación) el modelo sería Parcialmente Observable. Finalmente si no existe una manera directa de evaluar el estado actual del robot y es necesario inferirlo a partir de las observaciones (lecturas de sensores que no proporcionen directamente la posición del robot, como sensores láser, sonar o robots sin odometría interna), entonces se trata de un Modelo de Markov de variable Oculta, donde la variable oculta es el estado o posición del robot.

Una cadena de Markov no contempla en la determinación de las probabilidades de transición entre estados a los comandos de control u_t representados por la odometría del robot, es decir, la probabilidad de transición no depende de ninguna variable externa. Lo anterior supone un problema mayor ya que es preciso determinar un único valor de probabilidad de transición entre estados; sin embargo en su forma más general, dicha probabilidad dependerá en gran medida de los comandos de movimiento que se proporcionen al robot en el instante de su operación

3.3 Procesos de Decisión de Markov (MDPs)

Es posible tratar el problema de la toma de decisiones del robot (comandos dados por el usuario u_t) como un Proceso de Decisión de Markov o MDP (*Markov Decision Process*). En un proceso de decisión de Markov, para cada estado existe un conjunto de posibles *acciones*. Cada acción conlleva la probabilidad de transición a otro estado según los resultados de ejecutar dicha acción. Si como acciones se toma el conjunto de posibles comandos de control u_t, dado que u_t es continuo, entonces la cantidad de posibles acciones resultaría infinita lo que hace impráctico su uso.

Al igual que en el caso de las cadenas de Markov, si el estado no es completamente observable en todo momento, entonces se trata de un Proceso de Decisión de Markov Parcialmente Observable o *POMDP* (*Partially Observable Markov Decision Process*). Si el estado no puede ser directamente evaluado entonces se tiene un proceso de Markov Oculto o HMDP.

En resumen:

a) Si el estado $\overline{X}$ es por completo una función del tiempo *t* y sólo es posible conocerlo por medio de las observaciones (no es cuantificable por medios directos, i.e. no es *observable*), entonces es posible utilizar un Modelo de Markov de Variable Oculta y modelar el problema como una Cadena de Markov de Variable Oculta (*Hidden Markov Model*). Ejemplo: Robot sin sensores internos de posición (brújula o GPS) y sin odometría interna (estimación de u_t) o robots equipados con una cámara de video como único sensor.

b) Si el estado $\overline{X}$ es parcialmente observable, es decir, si puede ser calculado sólo bajo ciertas circunstancias (por ejemplo un entorno inmóvil) entonces se trata de un POMDP.

c) Si el estado $\overline{X}$ es por completo observable ó cuantificable en todo momento, es posible utilizar una Cadena de Markov (*Markov Chain*) o un Proceso de Decisión de Markov (*MDP*) para elaborar un modelo de localización.

3.4 Modelo Básico de Filtro

En los sistemas espacio-estado es posible definir [Dudek 00, p. 44]:

Modelo de Planta.- describe cómo cambia el estado del sistema $x(k)$ en función del tiempo y el control $u(k)$, también se le conoce como función de transición de estados.

$$x(k+1) = F\left[x(k), u(k)\right] + v(k)$$

donde Φ es la función de transición de estados, y $v(k)$ es una función de ruido.

Modelo de Sensor.- Indica cómo varían los datos de los sensores en función del estado del sistema

$$z(k+1) = h[x(k), x] + W(k)$$

donde $W(k)$ es una función de ruido, ξ es un modelo del medio ambiente y $h()$ es una función que describe las mediciones en función del estado del sistema.

Lo anterior implica que si existe h^{-1} entonces es posible obtener la posición (estado) $x(k+1)$ del robot, a partir de las observaciones $z(k+1)$ determinar la posición del robot en función de las observaciones que realiza.

3.5 Filtros Bayesianos

Desde el punto de vista de las probabilidades [Thrun 05, p. 27] si se tiene que

$$p(x_{k+1} \mid x_k, u_k, z_k) = p(x_{k+1} \mid x_k, u_k)$$

y

$$p(z_{k+1} \mid x_{k+1}, u_k, z_k) = p(z_{k+1} \mid x_{k+1})$$

donde $p(x_{k+1})$ es la nueva posición del robot (vista como función de densidad de probabilidad), x_k es la posición anterior, u_k es el comando de control, z_k son las observaciones anteriores y z_{k+1} corresponde a las observaciones en la nueva posición, se dice que se tiene un sistema completo, es decir, la nueva posición no depende de las observaciones anteriores y las observaciones sólo dependen de la posición, entonces es posible aplicar el teorema de Bayes para calcular la posición x del robot en función de las observaciones z de manera análoga a encontrar h^{-1} en el filtro básico:

$$p(x \mid z) = \frac{p(z \mid x)p(x)}{p(z)}$$

Los filtros Bayesianos operan en dos fases: *predicción* y *actualización*.

a) Fase de Predicción:

Se utiliza un *modelo de movimiento* para predecir la posición actual del robot en la forma de una función de densidad de probabilidad o PDF $p(x_k \mid Z^{k-1})$, tomando en cuenta únicamente el movimiento. Se asume la propiedad de Markov en donde x_k depende exclusivamente de x_{k-1} y del control conocido u_{k-1}, mientras que el modelo de movimiento se especifica como una densidad condicional $p(x_k \mid x_{k-1}, u_{k-1})$. La densidad predictiva sobre x_k es entonces obtenida por integración:

$$p(x_k \mid Z^{k-1}) = \int p(x_k \mid x_{k-1}, u_{k-1}) p(x_{k-1} \mid Z^{k-1}) dx_{k-1} \tag{3.1}$$

b) Fase de Actualización

En la segunda fase se utiliza un modelo de medición para incorporar información de los sensores para obtener $p(x_k \mid Z^k)$. Se asume que la medición z_k es condicionalmente independiente de mediciones anteriores Z^{k-1} dado x_k (la medición sólo depende el estado actual) y el modelo de medición es dado en términos de la similitud. Éste término expresa qué tan probable es que el robot se encuentre en la posición x_k dado que z_k es observada. La densidad sobre x_k se obtiene utilizando el teorema de Bayes:

$$p(x_k \mid Z^k) = \frac{p(z_k \mid x_k) p(x_k \mid Z^{k-1})}{p(z_k \mid Z^{k-1})} \tag{3.2}$$

Luego de la fase de actualización el proceso se repite recursivamente. En el instante t_0 el conocimiento sobre el estado inicial x_0 se asume en la distribución de la PDF inicial. En el caso de localización global, esta densidad podría ser una distribución

uniforme sobre todas las posibles posiciones. Para el rastreo de la posición la posición inicial a menudo es dada como la media y varianza de una Gaussiana centrada en x_0.

3.6 Filtro de Kalman

Una de las herramientas más utilizadas en la actualidad en los sistemas de espacio-estado para llevar una estimación de la posición de un robot móvil es el llamado Filtro de Kalman [Kalman 60], el cual se apoya en una serie de premisas (Fig. 3.2):

1.- Una función lineal de transición de estados (modelo de planta) de la forma:

$$x(k+1) = \mathrm{F}\,x(k) + \mathrm{G}u(k) + Cv$$

donde $x(k)$ es el estado anterior, Φ indica la variación del estado en ausencia de entradas, Γ es una matriz de transición entre estados, $u(k)$ es el comando de control y Cv es la matriz de covarianza del estado.

2.- Un modelo lineal de sensor de la forma:

$$z(k+1) = \mathrm{L}_E x(k) + W_i$$

donde $z(k+1)$ indica la medición del sensor al tiempo k+1, Λ_E es una matriz de dimensiones $m \times n$, donde m es la dimensión del vector de la medición del sensor y n es la dimensión del vector de estado $x(k)$ y W_i corresponde al ruido.

3.- Un modelo de ruido W_i con distribución Gaussiana y media cero.

En su etapa principal el filtro de Kalman hace una predicción de las mediciones de los sensores para el estado *k+1* aplicando las técnicas de los filtros Bayesianos mediante la fórmula

$Kt = \bar{S}_t C_t^T (C_t \bar{S}_t C_t^T + Q_t)^{-1}$ donde $\bar{S}_t, C_t$ y Q_t son matrices

conocida como Ganancia de Kalman, que requiere la inversión de matrices, implicando con ello un tiempo del orden de $O(k^{2.4})$ siendo k la dimensión del vector de mediciones [Thrun 05, p. 43] para la realización del cálculo de la posición.

3.7 EKF y UKF

No obstante se ha modificado el modelo general del filtro de Kalman conocido como EFK (*Extended Kalman Filter*) y UKF (*Unscented Kalman Filter*) dichas modificaciones pretenden hacer frente al problema de la linealidad, sin embargo, en ambos casos no se trabaja con el modelo real del sistema.

El caso anterior se complica cuando se utiliza sonar como medio de medición, ya que claramente su modelo de ruido no se comporta en absoluto en forma Gaussiana con media cero.

Sin embargo el problema principal radica en la obtención de la matriz Λ_E que, si consideramos el caso de poseer un anillo de 16 sonares, debe proporcionar un estimado de las lecturas de los 16 sensores en función exclusivamente del estado $x(k)$ del robot.

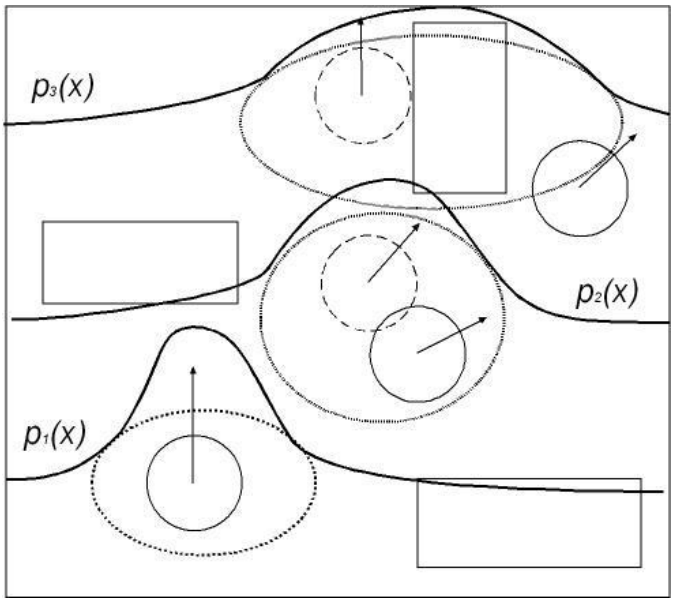


Figura 3.2 Distribuciones de probabilidad continuas con el Filtro de Kalman

Lo anterior se denomina "completitud" lo cual significa que es suficiente con conocer la posición del robot para poder estimar el valor entregado por los sensores, sin importar las mediciones anteriores.

En resumen, el filtro de Kalman realiza una estimación en forma continua de la nueva posición del robot a través de los parámetros de una función de densidad de probabilidad tipo Gaussiana, en función de su posición anterior y la información de control. Con base en la posición estimada calcula las mediciones de los sensores y realiza un ajuste final con base en las mediciones realizadas en la nueva posición.

Mientras más elevado sea el número de sensores más impráctico resultará utilizar el filtro de Kalman dada su complejidad y más difícil será encontrar un modelo de sensor apropiado.

3.8 Filtros de Partículas

En los métodos basados en muestreo o *sampling-based methods*, se representa una densidad $p(x_k \mid Z^k)$ por un conjunto de N muestras aleatorias o partículas $S^k = \{s_k^i; i = 1..N\}$ tomado de éste, debido a la dualidad esencial entre las muestras y la densidad a partir de la cual son generadas [Thrun 03]. A partir de las muestras podemos siempre construir aproximadamente la densidad original (Fig. 3.3).

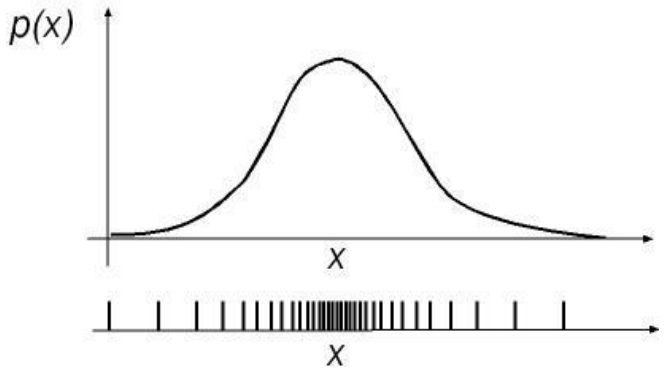


Figura 3.3. Ejemplo de distribución normal y muestreo de partículas con la misma densidad de probabilidad.

El objetivo es recursivamente calcular en cada iteración *k* el conjunto de muestras S_k que es tomado a partir de la distribución $p(x_k \mid Z^k)$.

3.8.1 Fase de Predicción

Se comienza a partir del conjunto de partículas S_{k-1} calculado en la iteración previa o en la fase inicial y se aplica el modelo de movimiento a cada partícula s_{k-1}^i muestreando a partir de la densidad $p(x_k \mid s_{k-1}^i, u_{k-1})$:

(i) para cada partícula s_{k-1}^i :

Se obtiene una muestra s'^i_{k-1} de la distribución $p(x_k \mid s_{k-1}^i, u_{k-1})$

Al hacer esto se obtiene un nuevo conjunto S'_k que aproxima una muestra aleatoria a partir de la densidad predictiva $p(x_k \mid Z^{k-1})$. La notación prima en S'_k indica que aún no se ha incorporado ninguna lectura de los sensores al instante *k*.

3.8.2 Fase de Actualización

En la segunda fase se toman en cuenta las mediciones z_k y se asigna un valor a cada una de las partículas en S'_k mediante el peso $m_k^i = p(z_k \mid s'^i_k)$, es decir, la similitud de s'^i_k dado z_k. Se obtiene S_k remuestreando a partir de este conjunto ponderado:

(ii) para j=1..N:

Se obtiene de S_k una muestra s_k^j a partir de $\{ s'^i_k, m_k^i \}$

El remuestreo selecciona con mayor probabilidad muestras s'^i_k que tienen una alta similitud (peso) asociadas con ellas y al hacer esto un nuevo conjunto S_k se obtiene y aproxima una muestra aleatoria de $p(x_k \mid Z^k)$. Existen implementaciones que logran hacer esto eficientemente en O(N) [Carpenter 97].

Luego de la fase de actualización se repiten los pasos (i) y (ii) recursivamente. Para inicializar el filtro, se comienza en el instante k=0 con una muestra aleatoria $S_0 = \{s_0^i\}$ de la distribución inicial $p(x_0)$, figura 3.4.

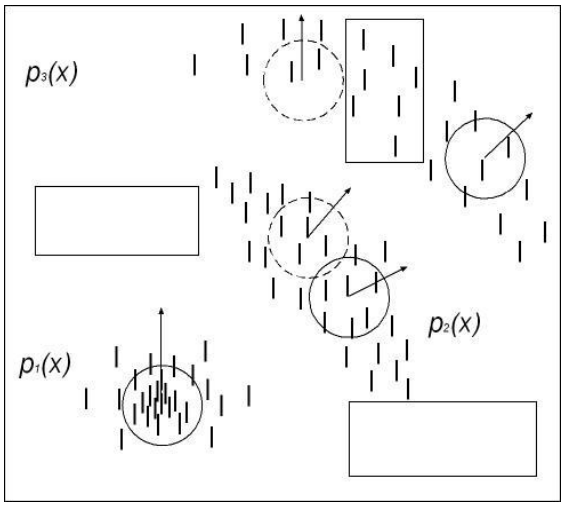


Figura 3.4. Aproximación de densidades de probabilidad mediante partículas.

3.9 Método de Localización de Monte Carlo (MCL)

En contraste al filtro de Kalman, el método de localización de Monte Carlo [Dellaert 99] utiliza un filtro de partículas para calcular la ubicación real del robot. El objetivo de dicho filtro es estimar el estado del robot en el instante actual k, dado el conocimiento sobre el estado inicial y todas las mediciones $Z^k = \{z_k, i = 1..k\}$ hasta el instante actual. El método calcula recursivamente la densidad $p(x_k \mid Z^k)$ en cada iteración, lo cual realiza en dos fases:

3.9.1 Fase de Predicción

Para tomar una muestra aproximada de la exacta PDF predictiva, se utiliza el modelo de movimiento y el conjunto de partículas S_{k-1} para construir la *densidad predictiva empírica* [Pitt 97]:

$$\hat{p}(x_k \mid Z^{k-1}) = \sum_{i=1}^{N} p(x_k \mid s_{k-1}^i, u_{k-1}) \tag{3.3}$$

La ecuación (3.3) describe una densidad de aproximación mixta a $p(x_k \mid Z^{k-1})$, consistente en una componente mixta $p(x_k \mid s_{k-1}^i, u_{k-1})$ igualmente ponderada por la muestra s_{k-1}^i. Para tomar una muestra de esta distribución mixta, se utiliza *muestreo estratificado* y se toma exactamente una muestra s'^i_k por cada uno de los *N* componentes mixtos para obtener S'_k.

3.9.2 Fase de Actualización

En la segunda fase deseamos utilizar el modelo de medición para obtener una muestra S'_k a partir del posterior $p(x_k \mid Z^k)$. En lugar de esto se utiliza la Ec. (3.3) y se muestrea a partir de la *densidad empírica posterior*:

$$\hat{p}(x_k \mid Z^k) \mu\ p(z_k \mid x_k)\hat{p}(x_k \mid Z^{k-1}) \tag{3.4}$$

en donde se utiliza directamente la regla de Bayes (sin el factor de normalización $p(z_k)$ en el denominador).

Esto es completado utilizando una técnica conocida como *muestreo por importancia* (*importance sampling*). Se utiliza para obtener una muestra aleatoria de una distribución difícil de estimar $p(x)$ mediante el muestreo desde una densidad $f(x)$ mucho más sencilla. En una acción correctiva, cada muestra es revalorada adjuntándole un *factor de importancia o peso* $w = p(x) / f(x)$. En el contexto del filtro de partículas desearíamos muestrear de $p(x) = \hat{p}(x_k \mid Z^k)$ y utilizamos como función de importancia $f(x) = \hat{p}(x_k \mid Z^{k-1})$ en tanto que ya se ha obtenido un conjunto de muestras S'_k a partir de éste en la etapa de predicción. Entonces se reasigna el peso de cada partícula según:

$$m_k^i = \frac{p(x)}{f(x)} = \frac{p(z_k \mid x_k)\hat{p}(x_k \mid Z^{k-1})}{\hat{p}(x_k \mid Z^{k-1})} = p(z_k \mid x_k) \tag{3.5}$$

El subsecuente *remuestreo por importancia* [Smith 92] necesita convertir el conjunto de muestras ponderadas no aleatorias de nuevo en un conjunto de muestras con igual peso $S_k = \{s_k^i\}$.

3.9.3 Resumen de Operación del Filtro

En resumen, el método de Monte Carlo posee las siguientes etapas:

1.- Inicialización. En la fase inicialización se genera un conjunto de partículas de acuerdo a la suposición (distribución) inicial de probabilidades de localización (Fig. 3.5). Si se desconoce por completo la ubicación inicial se supone una distribución uniforme sobre todo el espacio de localización (se supone la misma probabilidad para cada posición y orientación posibles en el ambiente).

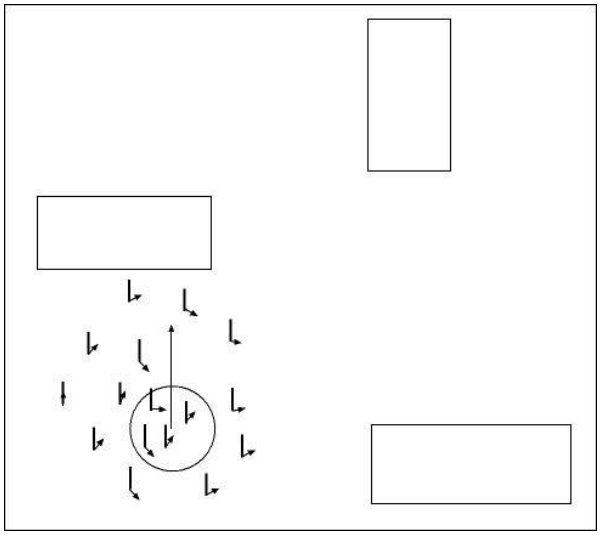


Figura 3.5. Inicialización del filtro de partículas de Monte Carlo para una ubicación conocida.

2.- Movimiento. Al recibir un comando de movimiento (o consultar con la odometría interna del robot el desplazamiento y giro efectuado por el robot en cierto

intervalo de tiempo) dicha estimación se toma como base para obtener la distribución de probabilidad del error en los movimientos del robot, (Fig. 3.6).

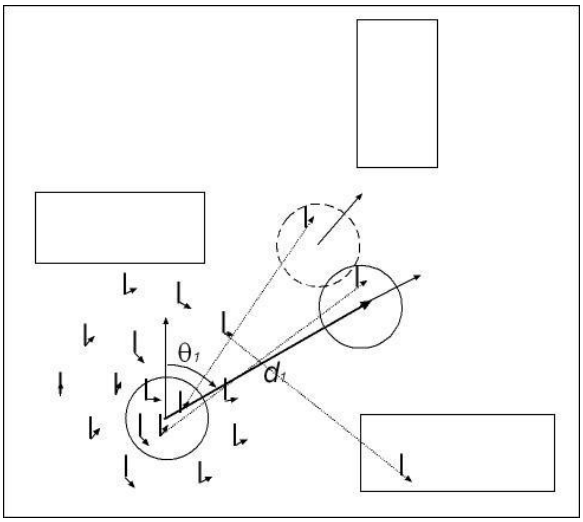


Figura 3.6. Estimación de desplazamiento con odometría.

De esta manera para cada partícula se genera un error de movimiento que, sumado al desplazamiento y giro estimados por la odometría dan como resultado una estimación directa de la nueva posición supuesta de la partícula dentro del ambiente (Fig. 3.7).

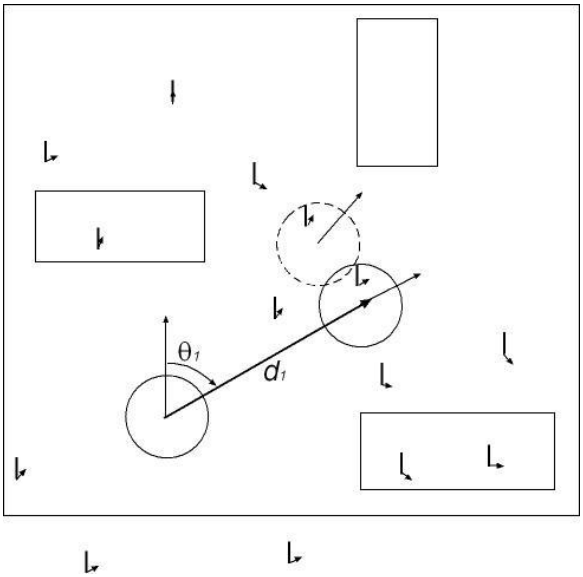


Figura 3.7. Desplazamiento de cada partícula de acuerdo con la odometría.

3.- Actualización. Al realizar una nueva observación con los sensores del robot, ésta es comparada contra la observación "supuesta" para cada partícula. Lo anterior supone la implementación de alguna técnica que permita obtener dicha observación supuesta (si se conoce la posición y orientación del robot), Lo anterior puede lograrse simulando la lectura del sensor si se conoce su forma de operación y se tiene un mapa del entorno. Otra forma de hacerlo es mediante la toma y almacenamiento exhaustivo de muestras en las fases previas a la localización, con el consiguiente problema de estimar la posición real del robot al momento de tomar la muestra, mediante algún tipo de dispositivo externo o midiendo directamente la posición del robot.

Una vez que se tiene la observación real y la supuesta, para cada partícula se obtiene la probabilidad de que la observación supuesta coincida con la real (Fig. 3.8). Es posible utilizar la distancia euclidiana entre las observaciones como medida de comparación mediante una fórmula del tipo:

$$p(z_k \mid x_k) = \frac{1}{e^{k|z_k^{sim} - z_k|}} \tag{3.6}$$

donde z_k^{sim} es la observación obtenida mediante el simulador para s_k^i y z_k es la observación real.

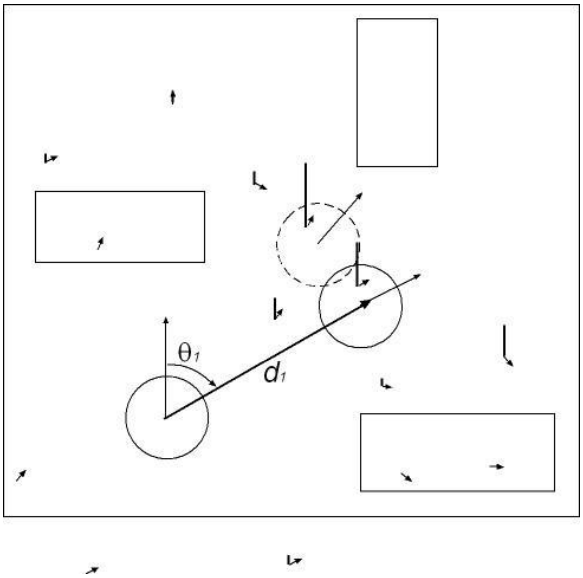


Figura 3.8. Factores de importancia (pesos) para cada partícula.

Finalmente se asigna a cada partícula un valor de coincidencia en función de la probabilidad encontrada.

4.- Remuestreo. Se vuelve a realizar un muestreo, utilizando el valor de coincidencia de cada partícula como parámetro para la generación de nuevas partículas. De esta manera las nuevas partículas se encontrarán distribuidas tomando en cuenta los valores de coincidencia de la muestra anterior. Con ello habrá un número mayor de nuevas partículas en las áreas donde exista un mayor coincidencia entre las observaciones reales y las supuestas. Aquellas posiciones erróneas con un alto nivel de coincidencia entre las observaciones supuestas y reales, al integrarse el movimiento del robot, irán disminuyendo paulatinamente a medida que las nuevas observaciones y movimientos sean tomados en cuenta mientras el robot avanza, (Fig. 3.9).

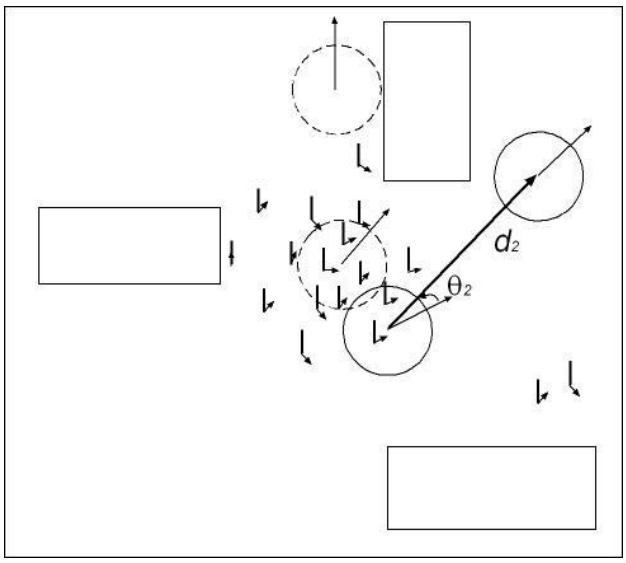


Figura 3.9. Remuestreo por importancia.

5.- Se repiten los pasos 2 a 4 hasta que las partículas se encuentran distribuidas en torno de cierta posición y orientación, lo cual es posible determinar mediante la desviación estándar del conjunto de partículas. Cuando ésta es menor a cierto valor ρ calculado de antemano, existe la certeza sobre la posición del robot, (Fig. 3.10).

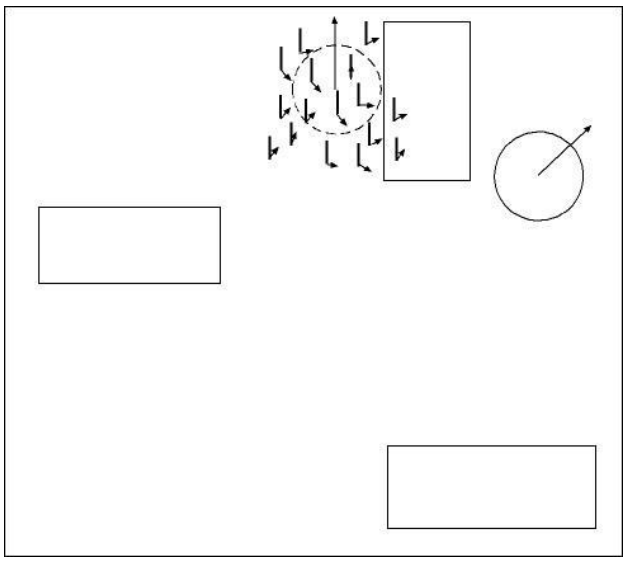


Figura 3.10. Convergencia a la ubicación real del robot.

Una vez que el robot se ha localizado correctamente el método continúa recibiendo nuevas observaciones y valores de la odometría mientras el robot se mueve en el ambiente.

3.9.4 Principales Ventajas

El método de localización de Monte Carlo posee las siguientes ventajas:

- Permite representar distribuciones multimodales y por lo tanto puede localizar globalmente un robot.

- Reduce drásticamente la cantidad de memoria requerida comparado con la localización discreta basada en celdas y puede integrar mediciones a una mayor frecuencia.

- Es más preciso que una localización discreta con ancho de celda fijo, dado que el estado representado en las partículas no se discretiza.

- Es fácil de implementar.

3.9.5 Desventajas

Su principal desventaja la supone el hecho de requerir generar la observación supuesta para cada partícula. Si se genera un número alto de partículas y se posee un sensor tipo láser, donde el número de lecturas sean 200 (y pueden llegar a 600), la complejidad de generar la observación supuesta para 200 * 10000 = 2 millones de lecturas individuales del láser (ya sea mediante simulación o una tabla de búsqueda) resulta demasiado costoso (10s para la actualización en nuestras pruebas, ver Cáp. 6). Por otro lado, si no se cuenta con una buena representación del ambiente ó mapa, no obstante se simulen las lecturas, éstas no corresponderán con las lecturas reales del ambiente y el método fallará. Adicionalmente el método sorprendentemente se degrada cuando los sensores son muy precisos como en el caso de las mediciones con láser, debido a que mediciones exactas del sensor evitarán que, eventualmente, una partícula cercana a la posición real posea una alta similitud con la observación real.

Otro problema radica en asegurar que el método aleatorio de muestreo pueda generar una partícula suficientemente cerca de la posición y orientación real, de lo contrario a medida que el robot se mueve y se obtienen nuevas observaciones, su valor irá disminuyendo y se obtendrá una localización errónea. Para poder evitar esto existen 2 técnicas: la primera es generar un número suficiente alto de partículas (en nuestros experimentos para un ambiente de 10 × 7 metros fueron necesarias alrededor de 10,000 partículas, con el consiguiente aumento del tiempo para generar las observaciones supuestas). El Segundo método es utilizar un kd-tree [Bentley 75] para poder estimar la probabilidad de cada observación con respecto a la posición. Este método se conoce como Dual MCL [Thrun 01], debido a que el conjunto de nuevas partículas se muestrea sólo de aquellas ubicaciones que correspondan lo más posible con la nueva observación, disminuyendo con ello el número de partículas

Finalmente otro factor fundamental en la localización corresponde a la obstrucción de los sensores por agentes móviles como pueden ser personas en el caso de robots de servicio o robots contrarios en el caso de robots bípedos que juegan futbol, lo

anterior supone nuevos factores a tomar en cuenta al momento de obtener la observación supuesta y evaluar la probabilidad de que dicha observación supuesta corresponda con la observación real en la ubicación actual del robot.

3.9.6 Principales Adaptaciones al Método de Monte Carlo

Para subsanar los problemas que supone el método de localización MCL los autores del mismo han propuesto una técnica conocida como MCL Mixto (o Mixture MCL) [Thrun 01] que combina el muestreo regular MCL con su "dual" o Dual MCL, que básicamente invierte el proceso de muestreo. Mientras el método regular inicialmente genera una posición aleatoria con base en la odometría y luego utiliza el error entre las mediciones real y supuesta para determinar la "importancia" de dicha partícula, Dual MCL genera posiciones utilizando la medición más reciente con los sensores del robot, entonces utiliza la odometría para verificar la correspondencia de esta suposición con la hipótesis previa de localización más los datos de la odometría.

La principal ventaja de esta adaptación radica en la reducción significativa del número de partículas necesarias para localizar la posición de robot, de hecho su rendimiento es muy bueno si el conjunto de partículas es pequeño (50 partículas según sus autores). Otra propiedad importante es que se recupera más rápido del problema de secuestro del robot que variaciones previas al MCL y opera bien cuando los modelos del sensor son precisos. Sin embargo su principal desventaja sigue siendo la necesidad de un modelo de sensor que permita un rápido muestreo de posiciones.

Para resolver lo anterior Mixture MCL utiliza conceptos estadísticos y árboles de densidad o *kd-tree* [Bentley 75] para "aprender un modelo a partir de los datos". Específicamente, en la fase de preprocesamiento las lecturas de los sensores son convertidas en un conjunto de características discriminantes y posiciones potenciales son generadas aleatoriamente utilizando los árboles generados. Una vez que el árbol ha sido construido el muestreo dual puede hacerse eficientemente.

Mediante un conjunto de *árboles kd*, cada uno de los cuales modela $p(x,o)$ para un subconjunto de observaciones *o* "similares", se particiona recursivamente el espacio de posibles posiciones en una forma que simplifica el muestreo a partir de $\bar{q} = p(o \mid x)/\pi(o)$, con como factor normalizador. Los datos utilizados para construir los árboles son pares $\langle x,o \rangle$ que se distribuyen de acuerdo a la distribución conjunta $p(x,o)$. Existen dos formas de realizar el muestreo:

1) El muestreo sintético o *synthetic sampling* es relativamente directo y puede hacerse en dos pasos: 1) Una posición *x* es muestreada a partir de una distribución uniforme y 2) Para dicha posición, se muestrea la observación de acuerdo con $p(o \mid x)$. El muestreo se repite hasta que un número suficiente de muestras haya sido generado (por ejemplo un millón). La forma más simple de generar dichas observaciones es a partir de una representación métrica del ambiente o mapa y un algoritmo de *ray tracing* para determinar cada observación. El conjunto de muestras resultante representa la distribución de probabilidad conjunta $p(x,o)$

2) Utilizar datos recolectados por un robot físico. Esto es preferible si no es fácil muestrear a partir del modelo de percepción $p(o \mid x)$. En general, muestrear a partir de dicho modelo resulta particularmente difícil cuando se utilizan cámaras, por lo que resulta un problema similar a la generación de gráficos por computadora.

Como puede verse, no obstante las mejoras al método tradicional de Monte Carlo mejoran su desempeño, persiste el problema de generar las lecturas supuestas $p(o \mid x)$, lo cual hace necesario contar con una representación *a priori* o mapa del entorno y realizar mediciones exhaustivas tanto para generar dicho mapa como para tratar de aproximar la distribución $p(o \mid x)$. Caso particularmente difícil resulta el uso de sensores de visión como único sensor dada la dificultad de poder obtener la lectura esperada de la cámara para una posición aleatoria, sin tener que realizar un proceso previo de toma de lecturas apoyado por localización basada en sensores láser o sonar para determinar la pareja $\langle x,o \rangle$ y construir un kd-tree eficiente.

Capítulo 4

Localización Robótica mediante Redes Neuronales Artificiales

4.1 Localización Mediante Redes Neuronales Progresivas (*Feed Forward*)

Algunos métodos de localización como [Choi 07] utilizan redes de perceptrones para determinar el diferencial ΔP_{loc} de movimiento del robot, con base en la lectura real $Sonar_{real}$ y una lectura simulada $Sonar_{virtual}$ para la posición supuesta $P_{supuesta}$ (Fig. 4.1):

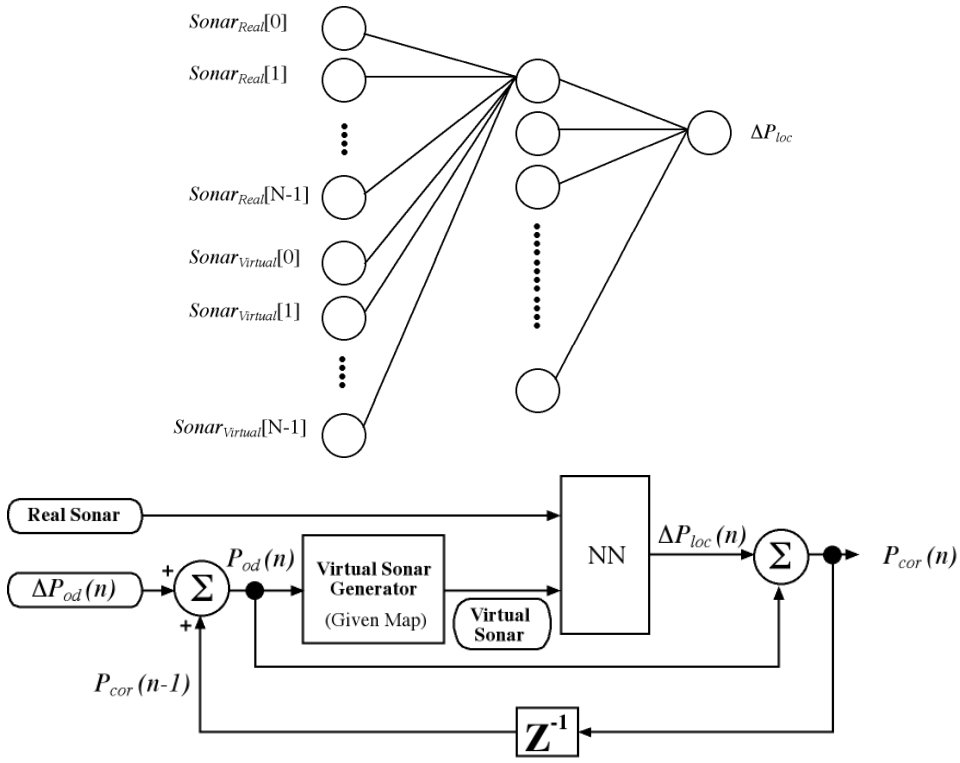


Figura 4.1 Red FFN de asociación de lecturas con posiciones del robot. Tomado de [Choi 07].

Este tipo de red aproxima la siguiente función:

$$\Delta P_{loc} = f^{-1}(Sonar_{real}) - f^{-1}(Sonar_{virtual}) \quad (4.1)$$

donde

$$Sonar_{real} = f(P_{real}),\ Sonar_{virtual} = f(P_{supuesta}) \quad (4.2)$$

El conjunto de entrenamiento se obtiene mediante simulación, sin embargo los autores recalcan que no es necesario conocer el mapa del ambiente donde el robot navegará en la fase de aprendizaje. Se simula tanto la observación para la posición real como para la posición supuesta dentro de cierto ambiente con suficientes características (corredores, esquinas cóncavas y convexas, etc.) y como se conoce de antemano ΔP_{loc} se genera un conjunto de vectores de entrenamiento $(Sonar_{real}, Sonar_{virtual}, \Delta P_{loc})$. Se seleccionan aleatoriamente $P_{supuesta}$ cercana a cada P_{real} de modo que ΔP_{loc} sea relativamente pequeña para facilitar el entrenamiento de la red.

Como puede observarse, este tipo de red neuronal aproxima a una Función de Ajuste o *matching* de observaciones, como en el caso de los métodos no neuronales referidos para este tipo de aproximaciones.

La principal desventaja de este método radica en la necesidad de generar una observación supuesta $Sonar_{virtual}$, lo cual hace necesario desarrollar con un simulador.

Finalmente en cuanto a la arquitectura de la red, los autores comentan que si se trata de entrenar a la red para calcular tres salidas simultáneas le resultará difícil aprender el desplazamiento en la posición y el ángulo simultáneamente. En su lugar proponen la siguiente arquitectura (Fig. 4.2):

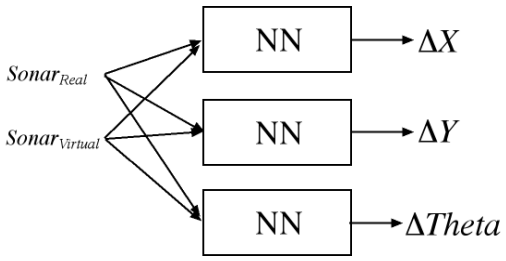


Figura 4.2. Arquitectura propuesta para la RNN. Tomado de [Choi 07].

Como puede observarse en la figura 4.2, se entrena por separado el diferencial en cada una de las dimensiones mediante retropropagación. Esto es comprensible ya que si se entrena una sola red con tres salidas, retropropagación de errores tomaría el error cuadrático medio de todas las salidas en lugar de tratar de disminuir al máximo el error en cada dimensión. En nuestra experiencia un error de 0.2 cm. en Δx no tiene el efecto que un error de 0.2 *rad* (11 grados), para el tipo de dimensiones del ambiente (por lo general de al menos de un par de decenas de metros cuadrados).

Cada Red de Perceptrones Multicapa tiene una estructura 32-16-16-1. Se utiliza retropropagación de errores como método de entrenamiento. El número de muestras de entrenamiento es 15,000, el factor de aprendizaje es 0.01 y se utiliza un *momentum* de 0.1. La función de activación es da tipo tangente hiperbólica, los rangos de entrada y salida van de -0.9 a 0.9. Se entrena con 1000 épocas.

Los resultados mostrados en la publicación reportaron un error en 3 de las 10 pruebas realizadas en un ambiente real. A primera vista parece que el error de odometría resultó mayor al entrenado con la red neuronal y por esta cuestión falló.

Otro caso de estudio se explica a continuación. En [Djekoune 01] se explora un método que una Red Neuronal de Regresión General (GRNN) para aproximar la relación entre características de alta dimensión con los estados del robot. Este método utiliza Análisis de Componentes Principales ó *PCA* [Jolliffe 02] para preprocesar imágenes obtenidas con una cámara omnidireccional (Fig. 4.3). El método extrae características del mapa de manera óptima y reduce las características correlacionadas. Los estados del

robot y las correspondientes características extraídas son utilizadas para entrenar la Red Neuronal (Fig. 4.4). Lo anterior permite al robot memorizar las características del ambiente y predecirlas dada su información de localización.

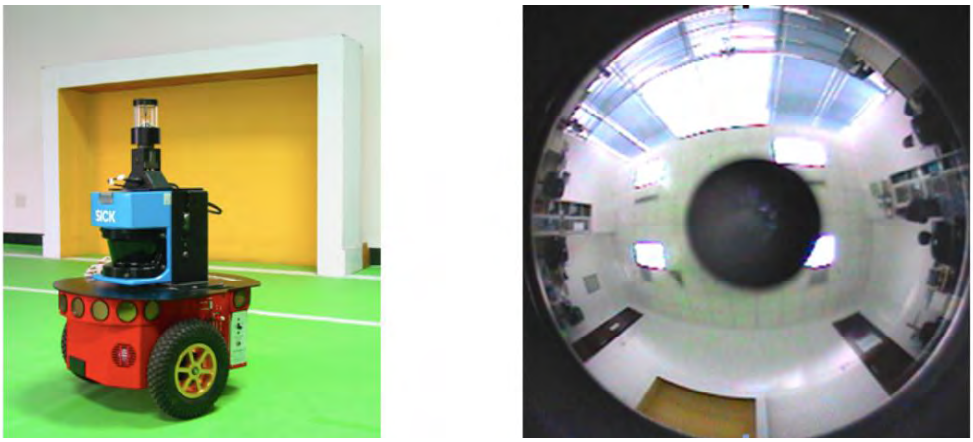


Figura 4.3. Robot Pioneer 3-DX (izq.) e imagen tomada con su cámara omnidireccional(der.). Tomado de [Djekoune 01].

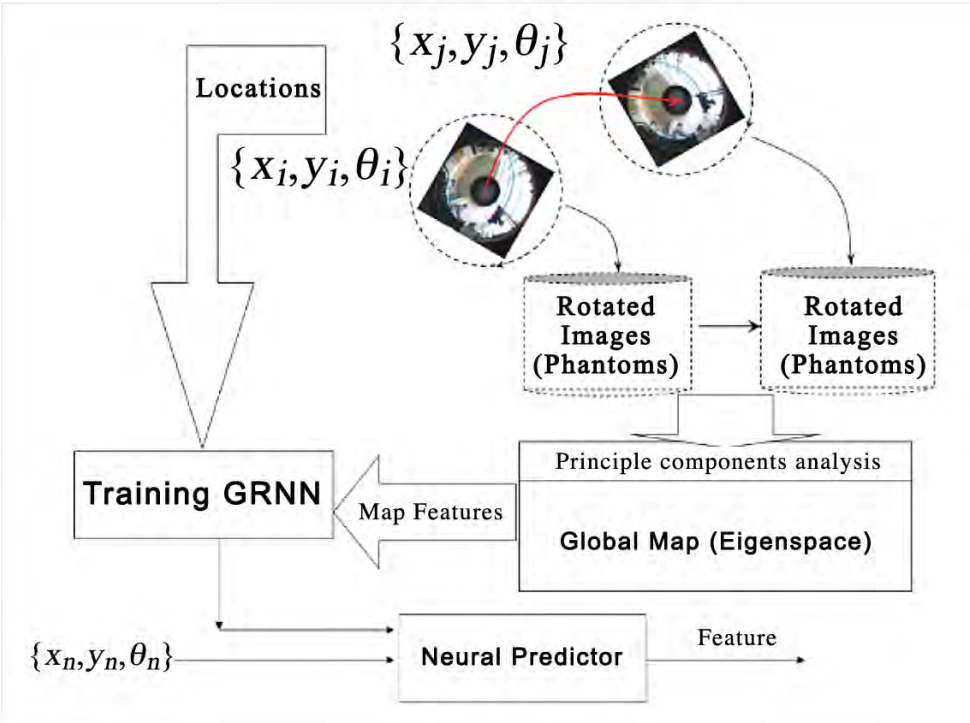


Figura 4.4. Esquema de aprendizaje de mapas basado en apariencia y GRNNs Tomado de [Djekoune 01].

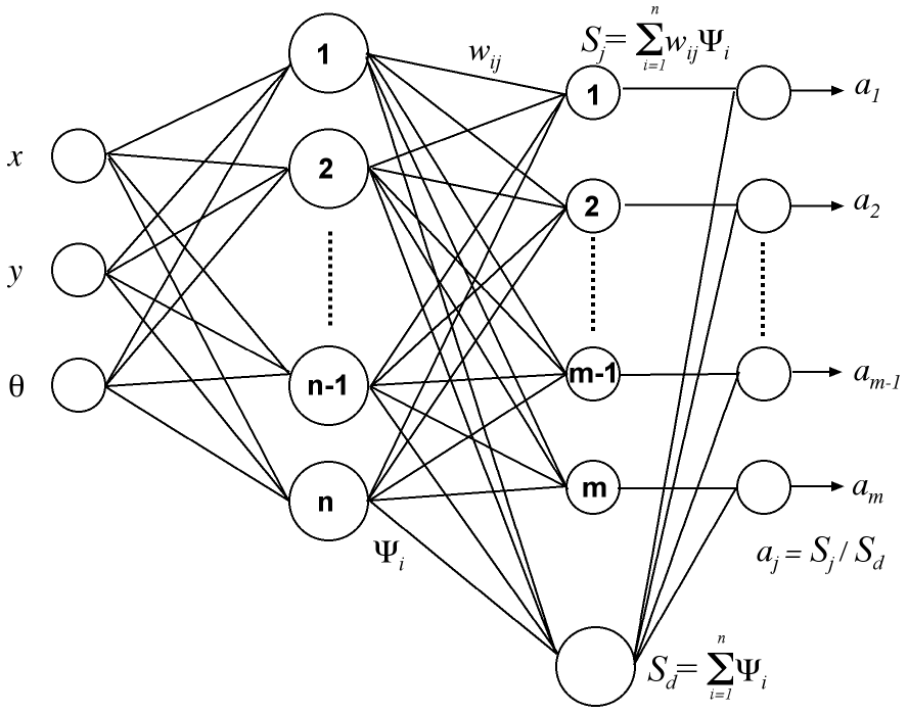


Figura 4.5. Red Neuronal de Regresión General GRNN. Tomado de [Djekoune 01].

La red Construida (Fig. 4.5) es una red de tipo *Feed Forward* con memoria, a menudo utilizada para la aproximación de funciones sin un modelo matemático preciso.

Dicha red consta de cuatro capas, entrada, patrón, adición y de salida. La capa de entrada recibe el vector de entrada proveniente del estado del robot y genera una salida Ψ_i. Las neuronas de la capa de adición calculan la suma ponderada de Ψ_i para el numerador S_i y una suma aritmética simple para el denominador S_d. Entonces la capa de salida realiza la división S_i / S_d para obtener la salida del sistema *A*.

En las pruebas reportadas por los autores este método fue capaz de proporcionar estimaciones correctas para 72 datos de prueba en 6 ubicaciones y ángulos distintos a los previamente entrenados.

Un aspecto importante de este método radica en el uso de un Análisis de Componentes Principales PCA para reducir la dimensionalidad de los vectores de entrada. Como resultado la Red no sólo almacena el patrón de entrenamiento, sino que

posee la habilidad de generalizar para poder predecir la escena que representa un intervalo intermedio a dos ubicaciones de referencia.

4.2 Localización Mediante Mapas Autoorganizados

En [Crowley 98] se expone un método de localización basado en RNAs que utiliza un sensor láser de tipo SICK para la toma de lecturas. Se utiliza un mapa autoorganizado para extraer características (esquinas).

Para la fase de entrenamiento de las esquinas se utilizan diferentes conjuntos de datos. Distintos conjuntos de datos se obtienen directamente del ambiente para las esquinas en distintas ubicaciones (r,θ) de un robot móvil con 10 distintas orientaciones desde 0^o hasta 180^o en incrementos de 15^o y con 5 diferentes distancias desde 1m hasta 6m en incrementos de 1m. En total se obtienen $50 \acute{} n$ conjuntos de entrenamiento para *n* tipos de esquinas (Fig. 4.6).

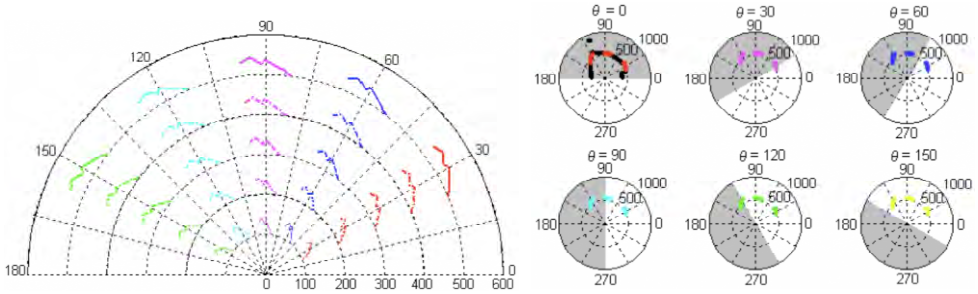


Figura 4.6. (a) Mediciones en distintas distancias. (b) Mediciones en distintas orientaciones. Tomado de [Djekoune 01].

En la práctica el robot móvil inicialmente utiliza su sensor láser para crear un mapa en forma de malla de ocupación. Las características de esquinas son entonces extraídas por el SOM sin que el robot se mueva, es decir, sin error odométrico. Con esto se obtiene una posición inicial de las características para poder compararlas posteriormente. Entonces el robot comienza a moverse, explora el ambiente con alguna estrategia y detecta las esquinas características en diferentes ubicaciones. Debido a los errores de localización es necesario que estas nuevas lecturas sean cotejadas en el mapa

del ambiente contra las lecturas inicialmente tomadas. Dada la información de odometría, la posición previa y la nueva ubicación de las esquinas características, el desplazamiento relativo es calculado.

En cuanto a la arquitectura de la red, ésta es entrenada con 250 vectores de entrenamiento (para 5 tipos de esquinas en total), la mínima distancia Euclidiana como medio de selección de la neurona ganadora

En sus resultados los autores reportan que este método por si solo no supera al algoritmo *Grid Map Matching* [Montesano 05] ni a [Yu 02], sin embargo su rapidez permite su utilización cada vez que se obtiene una nueva lectura del sensor. En combinación con *Grid Map Matching* se logran los mejores resultados.

En [Janet 95] se utilizan redes autoasociativas de Kohonen para localizar a un robot móvil dentro de una de 10 posibles habitaciones. El robot toma lecturas con un sensor de tipo sonar y construye una vista local que es alimentada a la red de Kohonen para encontrar por máxima similitud el patrón correspondiente a la habitación en que se encuentra. Este método únicamente proporciona la habitación donde el robot se encuentra, pero no su ubicación exacta y orientación dentro de dicho ambiente. La figura 4.7 muestra el proceso de comparación y selección de la habitación actual con respecto a los ambientes mostrados en la figura 4.8.

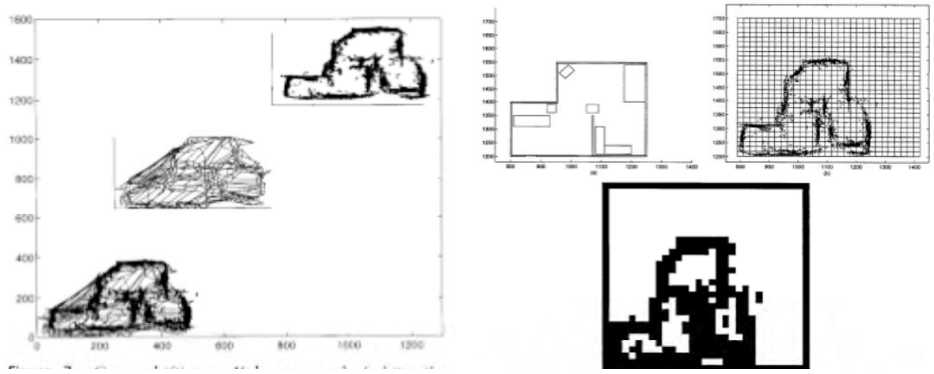


Figura 4.7. Transformación de las mediciones en una observación que se alimenta a la red. Tomado de [Janet 95].

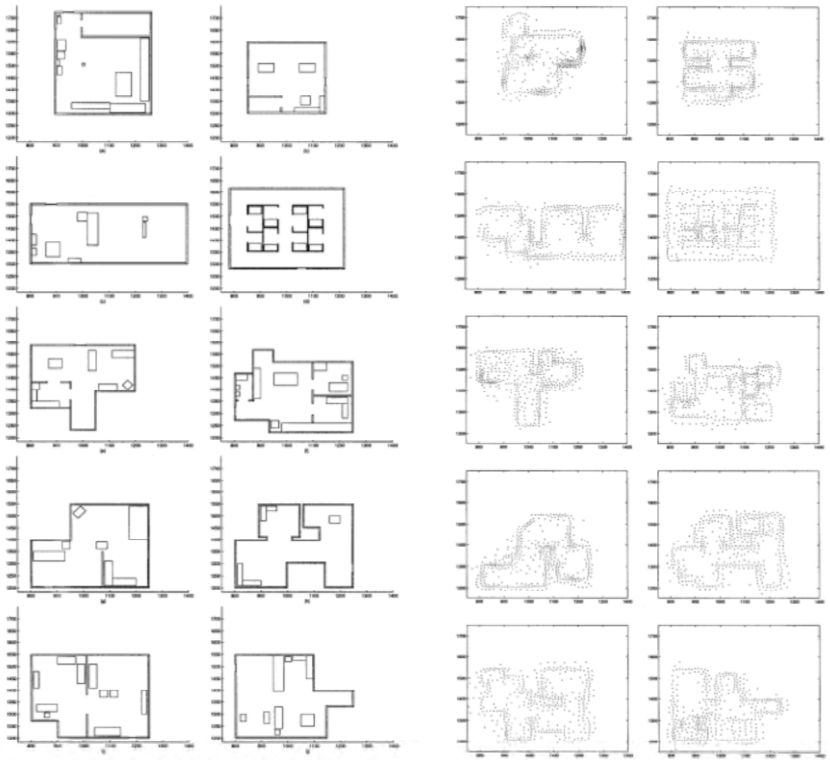


Figura 4.8. Mapas geométrico y sensorial para las habitaciones a reconocer.
Tomado de [Janet 95].

La gran desventaja de este método es que requiere que el ambiente sea completamente estático. Cualquier variación (como una puerta abierta o un cambio en el mobiliario) puede hacer que el método falle si la representación del ambiente no concuerda correctamente con la representación ideal del mismo almacenada en la Red de Kohonen.

4.3 Localización Mediante Otros Tipos de Redes

Una interesante alternativa que hace uso de las Redes ART (Adaptive Resonance Theory) es presentado en [Racz 94], donde se utilizan dos redes de este tipo para relacionar las lecturas de los sensores del robot con la ubicación estimada por la odometría del robot. Lo interesante de este método es que utiliza el concepto de Mapa Adaptable Difuso o Fuzzy-ARTMAP, mediante el cual dos redes ART (Fig. 4.9) independientes procesan, por un lado, las lecturas de los sensores del robot (11 sonares)

realizando un proceso de agrupamiento o c*lustering* no supervisado y por otro lado las ubicaciones (x, y, θ) , realizando la segunda ART otro proceso de *clustering* similar.

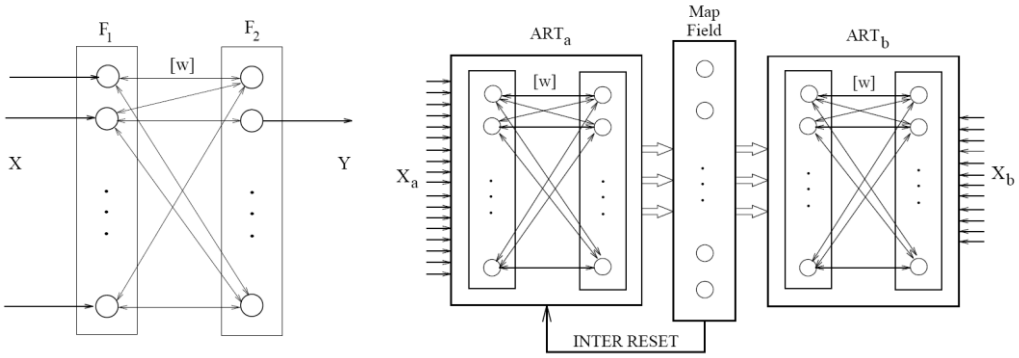


Figura 4.9. Red ART (izq.) y Red Fuzzy-ARTMAP (derecha). Tomado de [Racz 94].

En la fase de entrenamiento (Figs. 4.10 y 4.11), las lecturas de los sonares son preprocesadas y alimentadas a la primer ART. Por su parte las ubicaciones del robot son transformadas a una ubicación en forma de coordenadas difusas según el siguiente esquema, tanto para posición (x, y) con respecto a una puerta que se desea traspasar como para el ángulo θ con respecto a la misma.

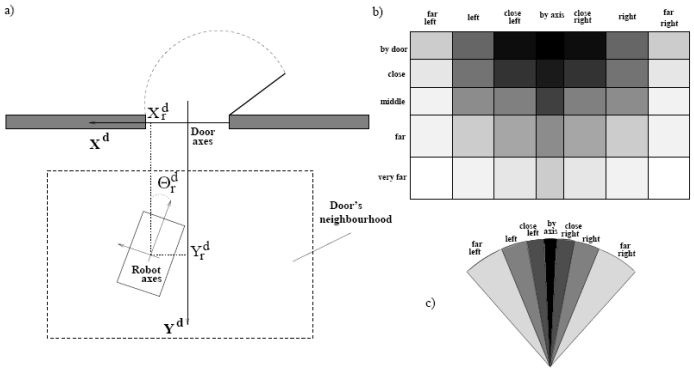


Figura 4.10. Representación difusa para posiciones y orientaciones. Tomado de [Racz 94]

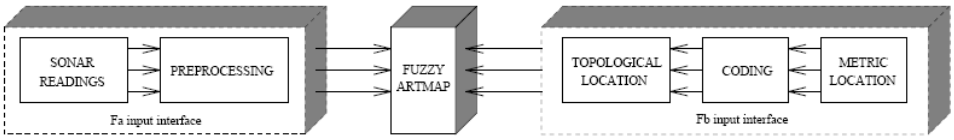


Figura 4.11. Entrenamiento de la asociación entre lecturas y posiciones difusas. Tomado de [Racz 94].

En la fase de operación, se alimenta a la primera red con las lecturas de los sensores del robot y proporciona como salida la predicción de la asociación correspondiente con la ubicación difusa estimada (Fig. 4.12).

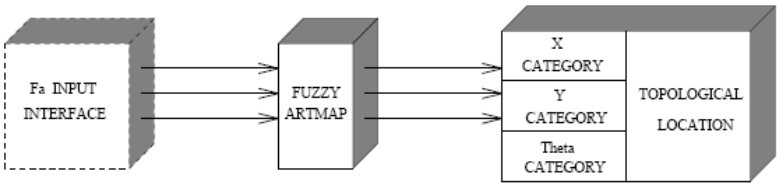


Figura 4.12. Fase de operación y predicción de la posición. Tomado de [Racz 94].

Como resultado, la red permite una localización relativamente eficiente (85% de aciertos) pero inferior con respecto a la mayoría de los métodos de localización que utilizan un enfoque probabilístico. Al analizar las causas de tales errores, los autores encontraron que su distribución se ajusta a la zona que limita una categoría de posición difusa con la siguiente debido a la discretización, de tal suerte que estiman que tales errores se deben a que en tales zonas la red es entrenada con valores muy similares de los sonares para proporcionar dos distintas clases de ubicaciones de salida (Fig. 4.13).

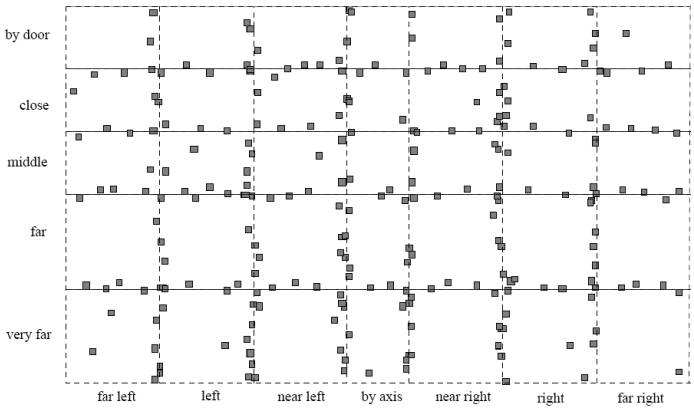


Figura 4.13. Ubicaciones de las posiciones erróneas. Tomado de [Racz 94].

4.4 Posibles Usos de RNAs en los Métodos de Localización Actuales

4.4.1 Métodos de Ajuste de Observaciones

Dado que este tipo de métodos pretenden encontrar una función *f* que realice un mapeo del espacio de posibles observaciones O al espacio de posibles ubicaciones *X* en algunos casos [Janet 95], es decir:

(4.3)

Es perfectamente posible obtener dicha función de aproximación aprovechando las características de aprendizaje de las RNAs, sin embargo en la presencia de oclusiones debidas a objetos o personas en movimiento este tipo de métodos fallan.

En otros casos [Choi 07], [Lu 97] se pretende encontrar

$$\Delta d = f^{-1}(O_t, O_{t-1}) \tag{4.4}$$

como una función inversa f^{-1} que proporciona el desplazamiento relativo del robot, en función de las observaciones anterior O_{t-1} y posterior al movimiento O_t. No obstante no se encontraron en la investigación métodos neuronales que directamente evalúen dicha función, ésto es perfectamente factible de realizar mediante una red de tipo Feed Forward. El problema con esta aproximación consiste en que, dado que no utilizan ninguna representación métrica o sensorial del ambiente en forma de mapa, no son capaces de localizar globalmente al robot o recuperarse del secuestro del mismo.

Por tal motivo se deben considerar complementarios a otro tipo de aproximación, sin embargo su efectividad es alta inclusive en la presencia de objetos inesperados en movimiento.

4.4.2 Optimización del Filtro de Partículas MCL (Monte Carlo)

El filtro de partículas de Monte Carlo [Thrun 99 y 99b] posee las siguientes ventajas:

a) Representación de distribuciones multimodales

b) Permitir un rápido recuperación en caso de secuestro del robot

c) Sencillo de programar

Sus desventajas son:

a) Es necesario que las partículas converjan a cierta ubicación para que se pueda tomar alguna acción.

b) Requiere de calcular rápidamente la observación o $p(o|x)$, ya sea mediante la toma exhaustiva de muestras o desarrollando algún simulador. En el primer caso requiere de un tiempo considerable y la supervisión humana, mientras que en el segundo caso requiere de considerable tiempo de proceso y simulación.

c) Se degrada en presencia de sensores con poco ruido al no inicializar ninguna partícula "suficientemente" cercana a la ubicación real.

Algunas alternativas [Thrun 00] se han propuesto para aminorar el tiempo de cálculo y actualización del factor de importancia de las partículas incluye

a) Almacenar en una tabla o *Lookup Table* las mediciones de los sensores, calculadas fuera de línea. Lo cual resulta difícil para el caso de sensores de visión.

b) Almacenar en un *árbol kd* las simulaciones de mediciones para generar rápidamente $p(o_t|x)$ u observación supuesta para cierta ubicación.

c) Utilizar un conjunto de *árboles kd* para calcular $p(x|o_t)$ o la distribución de probabilidad de la ubicación dada la observación más reciente. Lo anterior es realmente la función inversa f^{-1} donde $X=f^{-1}(O)$. Si bien un simulador es suficiente para encontrar f ya que $o=f(x)$ por la Suposición de Markov, su función inversa no es de fácil cálculo, por esa razón los autores del método Dual-MCL en [Thrun 00] han propuesto el uso de múltiples estructuras denominadas "bosque" de *árboles kd* para aproximar a tal función f^{-1}. En sus propias palabras:

"La idea básica es „aprender" un modelo de muestreo de la distribución conjunta $p(x,o)$ a partir de los datos, tal que las muestras de la distribución propuesta puedan ser

generadas con facilidad. La representación específica elegida aquí es de nuevo un conjunto de árboles *kd*, cada uno de los cuales modela *p(o|x)* para un subconjunto de observaciones „similares" *o*, cada una de las cuales particiona el espacio de todas las ubicaciones del robot en una forma que hace sencillo muestrear a partir de ella" .

[Thrun 00]

La idea básica de tales adaptaciones consiste en poseer una estructura que permita rápidamente obtener un conjunto de posibles ubicaciones *x* a partir de la observación actual *o*, es decir, que realice la función inversa f^{-1} . Según refieren sus autores, dada la alta dimensionalidad del vector de observación *o*, en la generación de la ubicación supuesta *x*, se extraen ciertas características de la observación más reciente, para poder alimentar con ellas al conjunto de *árboles kd*.

Como podrá observarse, la palabra *aprender* en el texto de los autores inmediatamente nos refiere al uso de alguna técnica de aprendizaje, por lo que se podría utilizar RNAs para "aprender" dicha función f^{-1} y realizar una optimización del método Dual-MCL.

Cabe aclarar que los autores propusieron dicha modificación para subsanar los problemas del método MCL ante errores altos en la odometría o en caso del secuestro del robot, sin embargo, el nuevo método Dual-MCL resulta muy sensible a oclusiones de los sensores al utilizar exclusivamente la observación más reciente para proponer una nueva ubicación.

4.4.3 Redes Neuronales de Localización Robótica

Si bien se analizaron previamente diversas aproximaciones neuronales al problema de la localización robótica, en su gran mayoría realizan un mapeo directo de las entradas u observación actual O_t a una posición *X* en el ambiente de operación. No se observó ningún caso donde se utilice más de una sola lectura (la observación más reciente) para el cálculo de la posición. No obstante en [Choi 07] se utiliza un cierto

factor de realimentación en el sistema, se utiliza exclusivamente una red progresiva o FFN (en realidad 3 redes de este tipo) para el cálculo del desplazamiento relativo del robot con respecto a una observación "supuesta" la cual es simulada.

En este punto se considera fundamental la incorporación de una "memoria de observaciones" y otra "memoria de desplazamientos" que permitan una mejor aproximación por parte de tales redes.

Inicialmente sería necesario transformar cada lectura de entrada en una representación de una dimensionalidad menor para facilitar el proceso de aprendizaje. Recordemos que a menudo se trabaja con sensores láser que pueden proporcionar vectores hasta de 600 lecturas en un rango de 0 hasta 500 cm. o cámaras web de, por lo menos, 320 × 240 píxeles.

Para tal efecto una de las técnicas mas utilizadas consiste en la Cuantificación Vectorial. Sin embargo la principal desventaja de este método radica en el deterioro de la lectura original. Si bien la cuantificación vectorial proporciona vectores tipo que guardan cierta similitud con el conjunto de vectores representado, ésta no proporciona una representación exacta de la lectura en bruto (como lo sería un simulador).

Por otra parte, uno e los factures mas importantes de los métodos de localización es sin duda la presencia de objetos móviles en el entorno, que introducen un factor de ruido por obstrucción difícil de modelar y filtrar. En este sentido la cuantificación vectorial presenta la ventaja de que, vista bajo el esquema de un mapa autoorganizado, puede permitir determinar cierta probabilidad de aparición de observaciones causadas por obstrucciones, si estas poseen cierto grado de similitud con el vector tipo.

Otra posibilidad de preprocesamiento de las observaciones sería utilizar la extracción de características como discontinuidades o puntos de inflexión. Dichas características podrían seguir apareciendo en la observación a pesar de errores en los sensores u oclusiones por objetos móviles.

De este modo se podría utilizar una arquitectura basada en RNAs para "aprender" a relacionar las observaciones y los movimientos más recientes del robot con las ubicaciones en el ambiente de operación ya sea en forma de desplazamientos relativos o una posición con respecto a algún marco de referencia común.

Distintos esfuerzos se han realizado para relacionar directamente las lecturas de los sensores de los robots con una ubicación. Ejemplos de ello son [Janet 95] y [Djekoune 01], sin embargo en ambos ejemplos se relacionan las lecturas del sensor con una ubicación dentro de un mapa topológico, es decir, no se proporciona la ubicación exacta (x,y,θ) dentro de algún sistema de coordenadas. En lugar de esto se identifica la habitación o la ubicación topológica donde se encuentra el robot, sin proporcionar una ubicación precisa en términos métricos.

Lo anterior se explica dadas las capacidades de generalización de las redes neuronales. Asumimos que la RNN se encuentra haciendo tareas donde distintas lecturas son mapeadas a un vector tipo o a una región dada. Sin embargo el problema fundamental consiste en asociar a un conjunto de 16 lecturas para el caso del sonar y al menos 68 lecturas (para el caso de sensor láser) con una posición en el espacio $R^3 : (x,y,\theta)$

Mientras mayor sea el numero de variables de entrada, mas difícil será el proceso de entrenamiento, por tal motivo se propone disminuir la dimensionalidad de los vectores de entrada, ya sea alimentando características presentes o no en la observación (preprocesamiento de la observación) o mediante la reducción a vectores tipo (cuantificación vectorial). La primera posibilidad presenta el problema de requerir tiempo de procesamiento para la extracción de las características, pero posee la gran ventaja de ser menos susceptible al ruido causado por la obstrucción parcial o total de algunas de las características presentes. El segundo método posee la ventaja de requerir menor tiempo de proceso (inclusive se puede realizar la cuantificación fuera de línea y utilizar tablas de búsqueda), sin embargo cualquier obstrucción del sensor proporcionará un vector tipo (observación) completamente distinto.

Capítulo 5

Localización Robótica mediante Visión Computacional

5.1 Estado del Arte en Localización Robótica con Visión

En años recientes debido al éxito de los filtros de Kalman y de Monte Carlo para localización robótica utilizando sensores láser, ha habido gran interés en sustituir los costosos sensores láser por cámaras de video de bajo costo, con el consabido problema de la generación de la densidad de probabilidad $p(o|x)$ entre la observación supuesta y la observación real.

Como alternativa a generar el modelo de sensor correspondiente a la imagen obtenida por la cámara se ha utilizado métodos que buscan características en la imagen observada, es decir, aproximan $p(o|x)$ mediante , donde $g(o)$ es una transformación sobre la imagen que intenta encontrar características diferenciadoras. Como ejemplos podemos citar el método de vSlam [Folkesson 05] que busca líneas en una imagen en blanco y negro, o el método *σSlam* [Elinas 05] que utiliza la transformada de puntos invariantes a la rotación y escala SIFT [Lowe 99] para obtener una representación de la imagen en forma de un conjunto de descriptores que son cotejados entre un par estéreo de imágenes, con la finalidad de obtener mediciones de profundidad de manera similar a lo que haría un láser. Ambos métodos utilizan la distribución espacial de las características o *features* encontradas en la imagen, sustituyendo la observación directa o (en este caso la imagen obtenida con la cámara) por el conjunto de características correspondiente a dicha imagen.

5.2 vSlam

vSlam [Karlsson 05] es un algoritmo basado en visión y odometría y permite la navegación a bajo costo en ambientes con alto tráfico de personas. No requiere de un mapa inicial y trata satisfactoriamente cambios dinámicos en el medio ambiente, por ejemplo, cambios de iluminación, objetos en movimiento y/o gente. Típicamente vSlam se recupera rápidamente de disturbios dramáticos como el problema del secuestro del robot.

El objetivo de vSlam es combinar imágenes e información odométrica en una forma que permite una construcción del mapa y una localización robustas. Lo anterior es importante ya que la información sensorial adquirida por el robot contiene gran cantidad de ruido difícil de modelar.

Desde el punto de vista de sistemas, vSlam puede ser descrito de la siguiente forma: Las entradas al sistema son la odometría y las imágenes, y la salida del sistema es la posición del robot y un mapa abstracto de vSlam (Fig. 5.1).

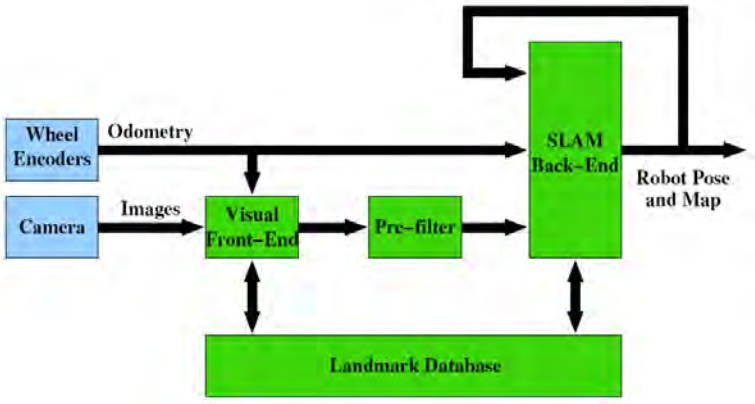


Figura 5.1. Diagrama de bloques de vSLAM. Tomado de [Karlsson 05].

El componente fundamental del mapa de vSlam son los *landmarks* que son "creadas" a lo largo de la trayectoria del robot. Un *landmark* visual es una colección de características únicas extraídas de la imagen. Cada marca o *landmark* es asociada con una *ubicación del landmark*, definida como la posición del robot (x, y, θ) cuando se crea la marca.

En el primer módulo de vSlam el *Visual Front End*, las imágenes son procesadas y comparadas con marcas previamente creadas. Si se efectúa un reconocimiento, un estimado de posición es calculado relativamente a la posición donde dicha marca fue creada por primera vez. Si no se efectúa un reconocimiento, el sistema intenta crear una nueva marca visual. Si la marca fue creada, ésta es almacenada en la Base de Datos de Marcas o *Landmark Database*.

Si una marca visual fue reconocida en el Visual Front End y una medición visual es calculada, entonces el segundo módulo, el pre-filtro o *Pre-filter*, determina la confiabilidad de la medición. Si la medición se considera no confiable, es rechazada y no será utilizada en ningún procesamiento futuro. Sin embargo si es aceptada se usa como entrada al módulo de SLAM.

El módulo de SLAM es un sistema realimentado que toma los datos de odometría y mediciones de posiciones relativas como entradas, para ser fusionadas y comparadas con el mapa actual. Primero la posición del robot es estimada basado en el mapa más actual y en las dos entradas. Entonces, el mapa es actualizado para reflejar la nueva información.

5.3 σSlam

Sigma Slam [Elinas 05] es un método de localización que se apoya en la localización de Monte Carlo pero posee algunas diferencias. Primeramente, no se limita a robots que ejecutan movimientos sobre un plano. El método se puede aplicar para movimientos en tres dimensiones (6 grados de libertad) desacoplando el modelo de la odometría propia del robot. Se derivan un estimado de los movimientos del robot a partir de mediciones visuales usando visión estéreo. En segundo lugar, se utilizan mapas dispersos de rasgos o *landmarks* naturales utilizando la transformada SIFT que es completamente invariante a cambios en la translación de la imagen, movimientos de escala y parcialmente invariante a cambios en la iluminación. La figura 5.2 muestra la ubicación en un mapa métrico de los *landmarks* encontrados por el método SIFT y utilizados para la localización del robot.

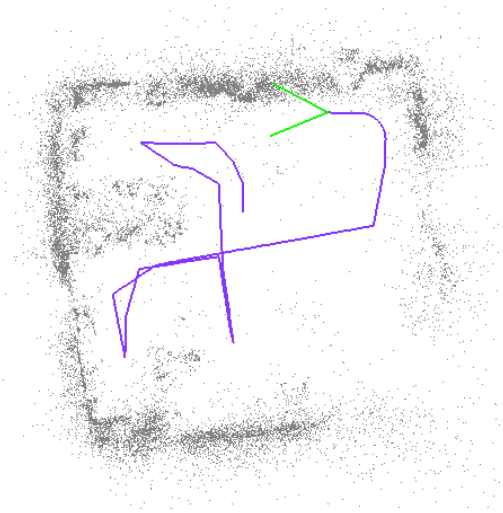


Figura 5.2. Vista aérea de un mapa de landmarks usado por σSLAM. Tomado de [Elinas 05].

5.4 Otras Representaciones

Menegatti et al. [Menegatti 04] representan imágenes mediante los coeficientes de Fourier de los componentes de las bajas frecuencias. Definen una métrica simple utilizando distancia euclidiana entre dichos coeficientes. Gross et al. [Gross 03] utilizan la distancia euclidiana entre los valores medios en RGB de las imágenes como métrica similar y emplean una estabilización de la luminancia y técnicas de adaptación al color para mejorar la exactitud de la correspondencia. Ulrich et al. [Ulrich 00] representan imágenes utilizando histogramas de color en los espacios normalizados RGB y HLS. Krose et al. [Krose 04] realizan un análisis de imágenes y almacenan las 20 primeras componentes principales. Cotejan imágenes comparando la distancia euclidiana entre estas componentes suavizadas por kernels Gaussianos locales.

5.5 Cuantificación Vectorial de Imágenes

Luego de analizar los métodos más actuales de SLAM con visión computacional, es posible concluir que en su totalidad en lugar de trabajar directamente con la imagen obtenida como modelo de sensor, trabajan con una transformación de dicha imagen, o sea, con una simplificación de la misma que sea susceptible de ser comparada en términos de distancia euclidiana, lo cual nos habla de un proceso de Cuantificación Vectorial de Imágenes, en donde se reduce la dimensionalidad del vector de las

observaciones a una representación mucho más simple. Lo anterior motiva el desarrollo de más y mejores algoritmos de cuantificación de imágenes y la preocupación por su desempeño en tiempo real ya que, tanto el método de vSlam como σSlam operan muy por debajo de los 15 cuadros por segundo sobre cámaras de baja resolución o Webcams.

Tanto para encontrar la matriz probabilidad de observación de símbolos *B* en un Modelo de Markov de Variable Oculta es posible utilizar un cuantificador vectorial (implementado mediante una red neuronal de agrupamiento) para clasificar las diferentes mediciones obtenidas durante los distintos recorridos del robot, mediante el mismo símbolo correspondiente a un tipo de observación. Lo anterior resulta especialmente útil si se posee un gran número de mediciones como las obtenidas con un anillo de sonares o láser.

Mediante dicho cuantificador es posible representar un vector de *n* dimensiones como un símbolo único v_k que represente la observación realizada en el estado actual.

Es posible determinar al inicio el número de clases, es decir, el cuantificador deberá clasificar las lecturas obtenidas, conociendo *a priori* el número *M* de símbolos.

Si se analizan nuevamente las muestras obtenidas durante cada recorrido y se alimentan al cuantificador, es posible obtener la probabilidad de observar los diferentes símbolos v_k para cada estado, y construir la matriz *B* del modelo de Markov.

5.6 Campos Aleatorios de Markov (Markov Random Fields)

Si se consideran a los estados de una cadena de Markov con una distribución espacial en forma de malla o látice y se conserva la propiedad de Markov, pero ahora el valor de cada estado dependerá exclusivamente del estado de sus vecinos, a diferencia de la cadena de Markov donde el estado siguiente depende exclusivamente del estado actual (dicho de otra forma cada estado depende únicamente del tiempo *t*, ya que para cada instante de tiempo *t* una transición es seleccionada), entonces se dice que se tiene un Campo Aleatorio de Markov *o Markov Random Field.*

Los campos aleatorios de Markov han tenido un área importante de aplicación en el procesamiento digital de imágenes, ya que es posible considerar a cada píxel de la imagen como dependiente de sus pixeles aledaños. De esta forma es posible extender la teoría de Markov al procesamiento de imágenes.

Si se construye un campo aleatorio de Markov a partir de una imagen y se encuentran las probabilidades para cada píxel o grupo de pixeles, entonces es posible comparar dos imágenes distintas según sus respectivos modelos. Desafortunadamente dichos modelos resultan muy sensibles a la traslación, rotación y escalamiento.

5.7 Transformada Discreta del Coseno DCT

La transformada discreta del coseno (DCT), también denominada transformada del coseno, es la más ampliamente utilizada en compresión de imágenes. Esta transformada cuenta con una buena propiedad de compactación de energía y es muy similar a la KLT Karhunen-Loève Transform), que produce coeficientes incorrelados, con la diferencia de que los vectores base de la DCT dependen sólo del orden de la transformada seleccionado, y no de las propiedades estadísticas de los datos de entrada.

La decorrelación de coeficientes es muy importante para compresión, ya que, el posterior tratamiento de cada coeficiente se puede realizar de forma independiente, sin pérdida de eficiencia de compresión. Otro aspecto importante de la DCT es la capacidad de cuantificar los coeficientes utilizando valores de cuantificación que se eligen de forma visual.

El proceso de transformación de la imagen consiste en calcular una serie de coeficientes que representan la amplitud de las frecuencias bidimensionales (Fig. 5.3) dentro de la imagen procesada (Fig. 5.4).

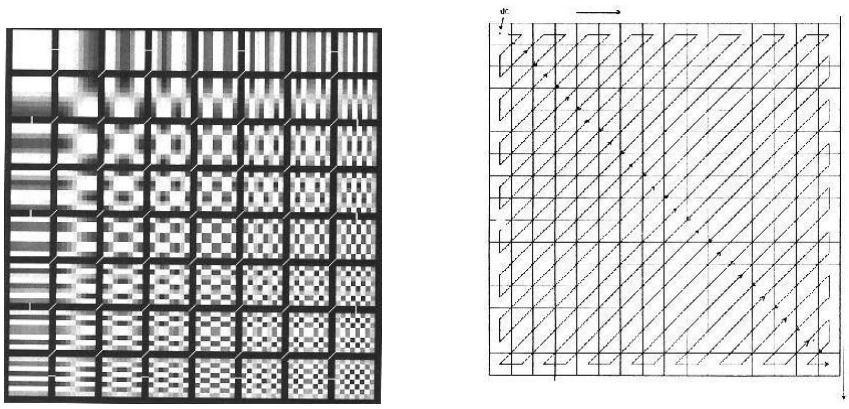


Figura 5.3. Imágenes base de la DCT. Modelo en Zig-Zag. Tomado de [UPM 11].

Conforme a la figura anterior, la transformada agrupa las frecuencias bajas en la parte superior izquierda de la imagen transformada y las frecuencias bajas en la parte inferior derecha. La eliminación de los coeficientes de las frecuencias altas permite la compresión de la imagen original.

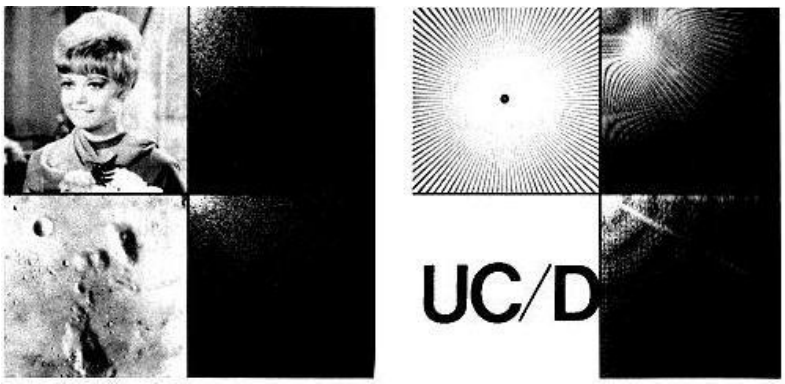


Figura 5.4. Distintas imágenes y sus respectivas DCT. Tomado de [UPM 11].

Una vez que se han obtenido los coeficientes de la imagen, es posible obtener la deferencia entre una imagen previamente procesada con la DCT y almacenada en una tabla de búsqueda, con la DCT de la imagen observada por la cámara al momento de navegar, siendo factible de utilizar el concepto de distancia euclidiana.

5.8 Transformadas SIFT y SURF

Diferentes aproximaciones al problema de la localización robótica mediante visión computacional se han probado con éxito, entre las más robustas se encuentran

aquellas que hacen uso de la extracción de características mediante métodos como PCA [Tamimi 04] o SIFT [Lowe 99] para conformar una base de datos de observaciones en base a uno o varios recorridos del robot por el ambiente de operación, dicha base de datos de observaciones es utilizada posteriormente como medio de obtener una observación supuesta para los filtros de partículas como el método de Monte Carlo [Bennewitz 06].

El método de Monte Carlo básicamente utiliza un Modelo Probabilístico de Movimiento (Modelo de Planta) para estimar los desplazamientos del robot con base en la odometría del mismo y un Modelo de Observación (Modelo de Sensor) para predecir una observación supuesta para cada una de las posiciones supuestas o "partículas" de la distribución de probabilidad condicional

La parte esencial del método de Monte Carlo reside en la forma de determinar

$$p(o_t \mid x_t) \tag{5.1}$$

que establece la probabilidad de que la observación actual o_t corresponda con la posición supuesta x_t para cada una de las posiciones x_{t_i} o "partículas" del filtro.

La forma más sencilla de hacer esto es mediante una función de distancia del tipo

(5.2)

donde o_t es la observación más actual obtenida con los sensores del robot y o_{sup} es la observación supuesta para la posición x_t.

Un ejemplo de función de distancia es la siguiente:

con *k* arbitraria (5.3)

Para poder determinar la observación supuesta o_{sup} cuando se utiliza visión computacional como principal sensor radica en la complejidad de obtener la imagen exacta que corresponda con la posición supuesta x_t para luego realizar la comparación. En lugar de ello comúnmente se recurre a una simplificación (extracción de características) de las observaciones, es decir:

$$p(o_t \mid x_t) = f(g(o_t), g(o_{\text{sup}})) \tag{5.4}$$

A diferencia de los métodos anteriores, la transformada SIFT [Lowe 99] y su variación conocida como SURF [Bay 06] proveen de un conjunto de vectores (o descriptores) con características de invariancia a la rotación, translación y escala. Dichos descriptores pueden ser utilizados como medida de similitud entre 2 imágenes (en blanco y negro), sin embargo por la gran cantidad de procesamiento requerido para su cálculo, no son susceptibles de ser implementados inclusive en procesadores con relativo poder de procesamiento (Como Pentium IV o Intel Core 2 Duo) , lo cual limita su integración en robots móviles o robots con poca capacidad de procesamiento.

La detección de objetos con método SIFT es un proceso de cuatro etapas. En la primera etapa una pirámide espacio-escala es construida mediante convoluciones sucesivas de la imagen original con una Gaussiana. Un operador cuasi Laplaciano se aplica en cada escala mediante la diferencia de dichas Gaussianas (DoG por sus siglas en idioma inglés) en imágenes adyacentes en la pirámide de escalas. La segunda etapa calcula máximos en la salida buscando en las octavas de la pirámide DoG, donde cada píxel es comparado contra sus vecinos espaciales en la misma escala y sus vecinos en las escalas superior e inferior. Si el píxel actual es el máximo o el mínimo entre dichos vecinos es marcado como un extremo. A continuación se ajusta una función cuadrática en tres dimensiones alrededor de los puntos extremos. Los extremos se filtran por contraste y por respuesta a los contornos (*edge response*), esto último utilizado el radio de curvaturas principales.

La tercera etapa crea los descriptores. A cada punto clave o *keypoint* se le asigna una orientación formando un histograma de las orientaciones de los pixeles en las escalas a los extremos del punto extremo y seleccionando la orientación dominante.

El descriptor es formado a partir de las orientaciones y magnitudes de los pixeles alrededor del *keypoint* muestreando sobre un arreglo de 16 × 6 pixeles alineado a la orientación del *keypoint*. Este arreglo es dividido en subregiones de 4 × 4 pixeles y entre cada subregión las orientaciones son utilizadas para llenar un histograma de orientaciones de ocho clases ponderadas por la magnitud de los pixeles y una Gaussiana para dar a los pixeles en los extremos un mayor peso que aquellos alejados de los mismos. La combinación de estos histogramas de orientación a partir de estas subregiones proporciona el descriptor de 128 dimensiones. Entonces es normalizado para proveer invariancia al cambio de iluminación.

La etapa final del método consiste en cotejar los *keypoints*. El autor utiliza tanto el primer como el segundo vecino cercano (*nearest neighbours*) para identificar las correspondencias. Cada *keypoint* en la base de datos es comparado con cada *keypoint* en la imagen de la escena mediante el cálculo de la distancia euclidiana entre ellos. Se mantiene un registro sobre cuáles pared de keypoints en la imagen son cercanos a los *keypoints* en la base de datos. Se calcula su radio de distancia y si éste es menor a cierto umbral el vecino cercano es considerado como una correspondencia con el *keypoint* de la base de datos. La correlación de los *keypoints* de la imagen con los *keypoints* en la escena de la imagen es entonces conocida y los *keypoints* que no poseen ninguna correspondencia son descartados.

En el método estudiado en [Bennewitz 06] se utiliza el método SIFT para determinar una serie de puntos o “descriptores” que resultan invariantes a transformaciones de rotación y escalamiento, lo cual resulta sumamente útil para comparar imágenes. La figura 5.5 muestra las características detectadas originalmente (izquierda) y durante la ejecución de la localización (derecha).

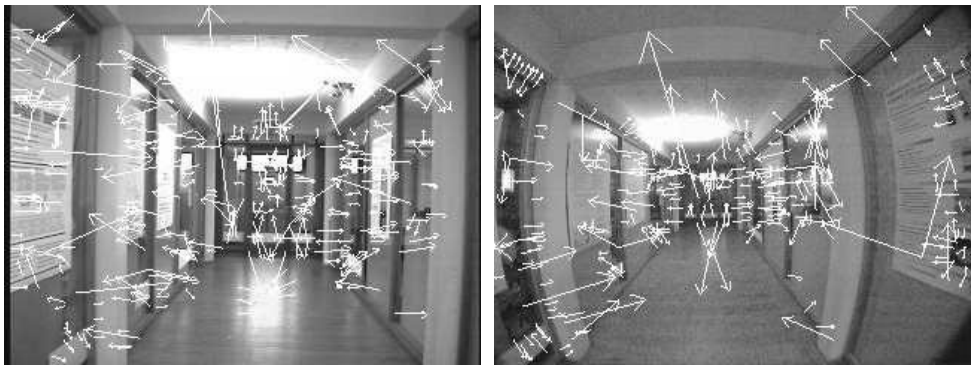


Figura 5.5. Detección de Características Invariantes con SIFT. Tomado de [Bennewitz 06].

En este caso $p(z|x)$ se obtiene a partir de la similitud de los descriptores encontrados en la imagen observada por la cámara del robot, contra los descriptores almacenados en una base de datos para una serie de posiciones que visitó el robot en una fase previa de adquisición de imágenes.

Desafortunadamente la precisión de este método no es tan buena si se utiliza exclusivamente una cámara y la odometría del robot durante la toma de muestras, ya que el error acumulado por la odometría del robot provocará que los descriptores almacenados para cierta posición posean variaciones de escala y desplazamientos. La mejor alternativa sería utilizar un sensor láser junto con la cámara para realizar las observaciones y un método de ajuste del desplazamiento con base en las lecturas del láser como el propuesto en [Lu 97].

A pesar de que el desempeño reportado por los autores de los métodos anteriores es aceptable, en una gran cantidad de aplicaciones y dispositivos (*hand-held* pc‟s, robots de tamaño reducido, etc.) la implementación de los mimos resulta inviable en términos de a) capacidad de procesamiento (especialmente en implementar el método SIFT para operar en tiempo real), b) imposibilidad de integrar dispositivos con un alto costo como los sensores láser (alrededor de U.S. $3,500.00) y c) reducidas dimensiones, lo que compromete el desempeño de los métodos basados en la extracción de características.

Capítulo 6

Comparativo de Métodos de Localización

6.1 Implementación de Métodos de Localización

Con la finalidad de poseer un marco práctico para la evaluación del método que se pretende desarrollar en el presente trabajo, se implementaron algunos de los principales métodos de localización presentados en la investigación. Lo anterior permitió evaluar a detalle los aspectos tanto teóricos como prácticos en su implementación y prueba. Con base en dicha implementación se determinaron las áreas de oportunidad que pueden implicar el uso de técnicas de aprendizaje basadas en RNAs.

6.2 Localización con Láser para Robots de Servicio

Con el método de creación de mapas métricos mediante Redes de Agrupamiento Borroso descrito en [Llarena 11, p. 97] (ver Apéndice B, sección B.5) se generó con la ayuda de los sensores del robot ARTURito un mapa métrico que sirvió de base tanto para realizar pruebas en simulador como en el ambiente real (Figs. 6.1 y 6.2).

Figura 6.1: Laboratorio de Biorobótica de la D.E.P.F.I., UNAM

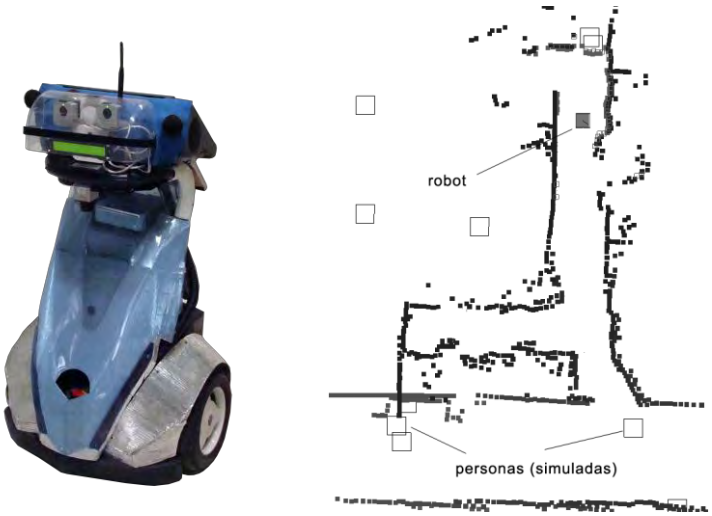


Figura 6.2. Robot ARTURito y mapa métrico generado con el sensor láser.

El robot ARTURito (Autonomous Ready-To-Use Robot) es un robot de servicio construido a partir de un modelo comercial del juguete Famosa Dareway[MR]. A dicho modelo le fue incorporado un sensor láser Hokuyo USG04Lx, una tarjeta de control diferencial basada en PIC específicamente desarrollada para tal efecto, una terminal POS Javelin Wedge[MR] para interactuar con el usuario y mostrar los mapas del entorno, un par de cámaras Unibrain Fire-i[MR], un teclado USB, un ruteador WiFi y una computadora Apple MacMini[MR] Core2Duo con 4Gb de RAM y procesador a 2.4 Ghz.

ARTURito es capaz de mantenerse en operación hasta por 40 minutos y tomar 10 muestras por segundo de hasta 728 lecturas con su sensor láser. La terminal POS se encarga del diálogo con el controlador diferencial de la base del robot vía puerto serial y calcular la hipótesis de odometría. También recibe los comandos de movimiento provenientes del módulo de control de alto nivel (piloto) vía sockets UDP (ver Fig. 6.3) e informar al piloto de la estimación odométrica actual. Lo anterior, conforme la arquitectura ViRbot (Fig. 6.3) descrita en [Savage 98]. Como la mayor parte del código de ViRbot se ejecuta en la computadora Apple Mac Mini, se desarrolló una librería de localización integrada con el Cartógrafo (que es el módulo que posee el mapa del ambiente usado para navegar y planear rutas). Dicha librería puede ejecutarse como un módulo por separado, lo que hace facil su integración a otras arquitecturas (ver el Apéndice B).

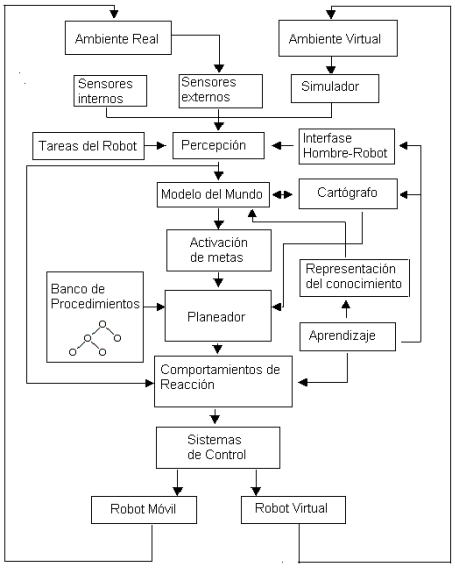


Figura 6.3: Esquema ViRbot de los módulos del ARTURito. Tomado de [Savage 98].

6.2.1 Pruebas en el Simulador

Con el mapa generado anteriormente se implementaron y probaron los siguientes métodos de localización en el simulador de ViRbot:

a) Búsqueda simple de la observación almacenada más similar a la lectura actual, dentro de un conjunto de observaciones simuladas fuera de línea.

Con ayuda de un simulador se obtuvo fuera de línea un conjunto simulado de 170000 observaciones con su respectiva ubicación de origen. Se desarrolló un método de localización por cotejamiento de lecturas que busca secuencialmente la observación más similar dentro del conjunto y presenta la ubicación correspondiente a dicha lectura como hipótesis de localización.

b) Red neuronal FFN 68□× 12 × 3, que asocia directamente las lecturas del láser con ubicaciones (x, y, θ) dentro del ambiente.

c) Método de localización por cotejamiento de lecturas por alineamiento de lecturas láser citado en [Lu 97].

d) Método de localización de Monte Carlo [Thrun 00] en distintas versiones:

1.- a) Observaciones simuladas en tiempo real y b) simuladas *a priori* (base de datos) con búsqueda rápida en árboles kd y modelo probabilístico de sensor propuesto por los autores en [Thrun 00], ver Fig. 2.3.

2.- Simulación de lecturas *en tiempo real* y calculando la distancia Euclidiana entre la observación real y aquella simulada (observación supuesta) como medida de distancia entre observaciones.

3.- Con un conjunto de observaciones simuladas *a priori* y mediante la búsqueda de la observación supuesta en árboles kd propuesta en [Thrun 01] con un conteo de correspondencias como medida de distancia.

Dichos métodos se probaron en la plataforma ViRbot desarrollada previamente, tanto para el caso de un ambiente estático (Fig. 6.2) como para el caso de oclusiones fortuitas ocasionadas por objetos en movimiento (básicamente personas). Como sensor se simuló un sensor láser con 68 lecturas para un ángulo de -110 a 110 grados (el frente de robot se considera a 0 grados). La Tabla 6.1, renglones 1 a 3 muestra los resultados de la búsqueda simple, la red FFN y el método de cotejamiento de lecturas laser. A continuación se discutirán las pruebas del método de Monte Carlo.

6.2.2 Experimentos con el Método de Monte Carlo

Dentro de las pruebas se evaluaron diferentes funciones de distancia entre observaciones (lecturas del sensor láser) para el método de localización de Monte Carlo:

a) Modelo Probabilístico de Sensor

Inicialmente se implementó el modelo de Sensor propuesto en [Thrun 01] por los autores del Método MCL (ver Fig. 2.3). Para obtener $p(o_t \mid x_t)$ primero se determinó la observación supuesta s_{p_i} para cada valor individual s_i del sensor láser con ayuda del simulador operando en tiempo real. Luego se obtuvo de la gráfica anterior la $p(s_{r_i} \mid s_{p_i})$ y finalmente

$$p(o_t \mid x_t) = \prod_{i=1}^{n} p(s_{r_i} \mid s_{p_i}). \tag{6.1}$$

Debido a que en el caso del ARTURito, el vector de lecturas de observación del láser posee 68 dimensiones a diferencia de las 16 dimensiones del modelo de sensor utilizado por los autores, en este caso $p(o_t \mid x_t) \to 0$ ya que el valor de $p(s_{r_i} \mid s_{p_i})$ para una correcta correspondencia es inferior a 0.05, y

$$0.05^{68} = 3.39 \acute{} \ 10^{-89} \tag{6.2}$$

que correspondería con el mayor valor de coincidencia entre observaciones.

Lo anterior supone un problema de representación, ya que es necesario trabajar con números muy pequeños. Una alternativa sería utilizar logaritmos para evitar caer en errores de representación numérica (valores *NAN* o *Not a Number*) en el cálculo de la probabilidad.

$$p(o_t \mid x_t) = \exp\left\{\frac{1}{n}\sum_{i=1}^{n} \ln\left[p(s_{r_i} \mid s_{p_i})\right]\right\} \tag{6.3}$$

La figura 6.4 y la Tabla 6.1 en el 5º renglón muestran el resultado de aplicar la función anterior. Se observó que tanto en el simulador como en el ambiente real el desempeño de la localización converge correctamente a la ubicación real, siempre y cuando no existan oclusiones en el 30% de las lecturas del sensor. Lo anterior se deduce de (6.1), ya que debido a que al tomar como eventos independientes cada lectura del sensor y multiplicarlos, la obstrucción de sensores debidos a movimientos de objetos u obstrucciones producen que rápidamente $p(o_t \mid x_t) \to 0$.

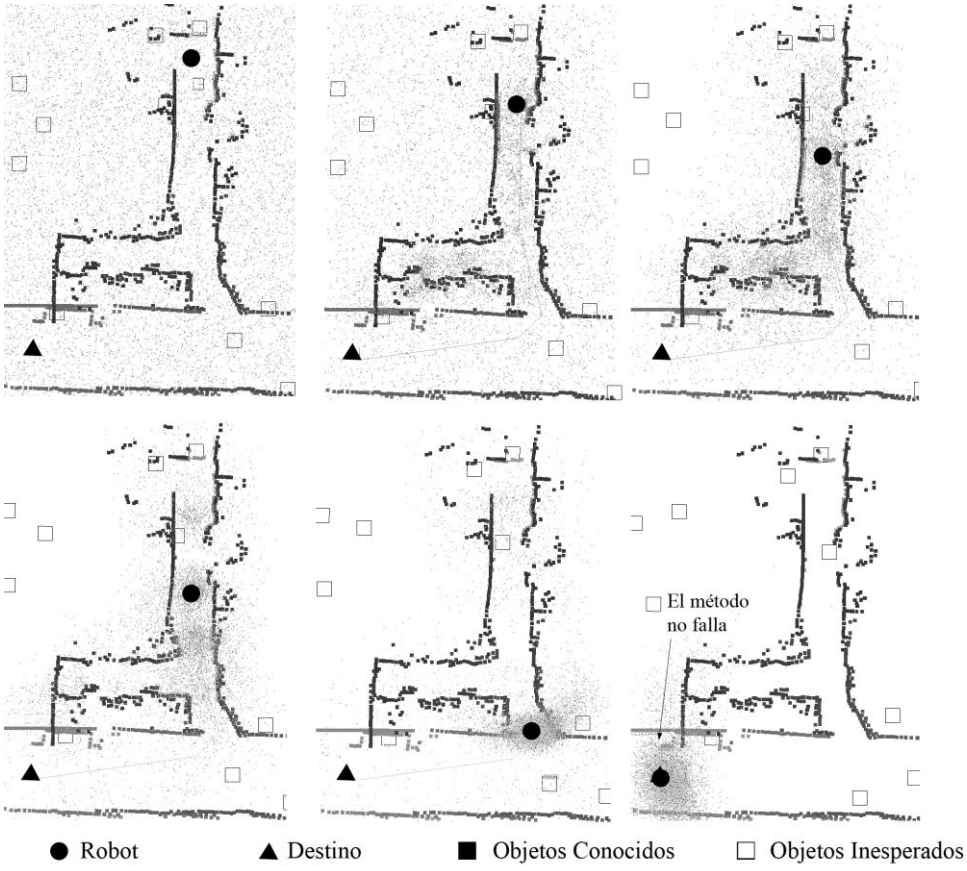


Figura 6.4. Prueba del Método de Monte Carlo con Modelo Probabilístico de Sensor.

La desventaja encontrada en éste método radica en el cálculo de $p(o_t \mid x_t)$ ya que es necesario evaluar 10000 partículas × 68 lecturas del láser = 680,000 evaluaciones de lecturas individuales en distribuciones de probabilidad de tipo Gaussiano, lo cual conlleva un tiempo considerable (1 s.) en un equipo con procesador Intel Core 2 Duo.

b) Distancia Euclidiana.

Como una manera de comparar contra un método simple de evaluar $p(o \mid x)$, se hicieron pruebas utilizando una medida de distancia euclidiana.

$$p(o \mid x) = e^{-kd}, \; donde \; k = 1 \; y \tag{6.4}$$

$$d = \mid o_{real} - o_{\sup} \mid \tag{6.5}$$

Al utilizar la distancia Euclidiana simple (Fig. 6.5 y Tabla 6.1, 4º renglón), se observó que el método MCL resulta sumamente sensible a los errores en la medición, ya que rápidamente $p(o \mid x) \to 0$ al aumentar *d*, debido a la dimensionalidad elevada de los vectores de observación (68 lecturas). Lo anterior si bien localiza rápidamente al robot de una manera correcta cuando se utiliza el simulador (Fig. 6.2), en el caso del ambiente real de operación localiza erróneamente al robot si existe una ligera variación en la ubicación del mobiliario o existen objetos y personas no contempladas originalmente en el mapa. La figura 6.5 muestra la distribución de las partículas calculada (nube de puntos) en presencia de objetos inesperados (cuadrados vacíos) al realizar una localización global.

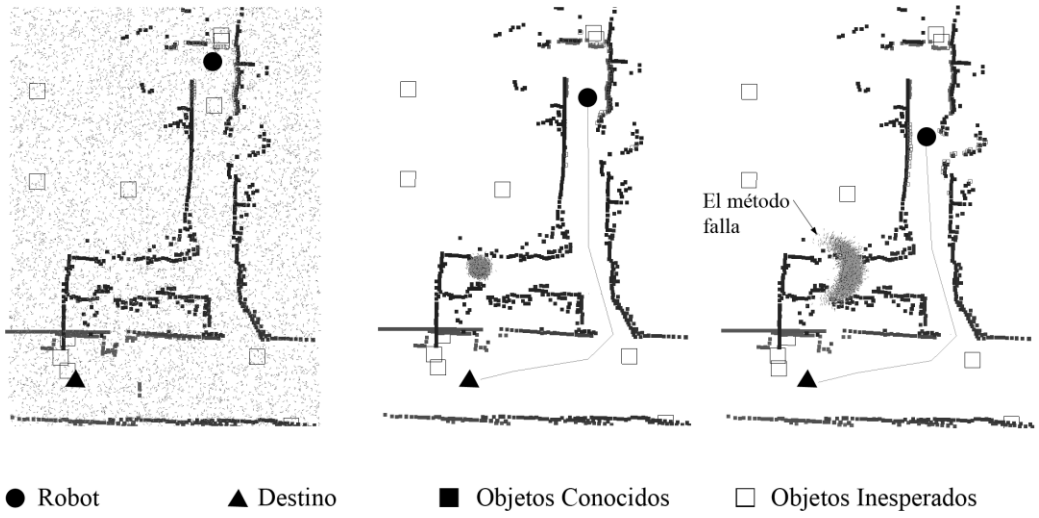


Figura 6.5. Prueba del Método de Monte Carlo con Modelo Euclidiano de Sensor.

c) Conteo Simple de Correspondencias.

Como alternativa al método anterior se determinó una función de distancia mediante un conteo simple de correspondencias individuales del láser correctas, es decir, se contó directamente en cuantas de las 68 lecturas individuales del láser la observación real coincide hasta cierto factor e = 10% con la observación supuesta para dicha ubicación. Bajo este esquema se calcula la función de probabilidad como

$$p(o \mid x) = \frac{l_{match}}{l_{total}} \tag{6.6}$$

donde $$l_{match} = \sum_{i=1}^{68} \begin{cases} 1 : si \mid o^i_{real} - o^i_{\sup} \mid < e \\ 0 : si \mid o^i_{real} - o^i_{\sup} \mid >= e \end{cases} \text{ y } l_{total} = 68 \qquad (6.7)$$

Como resultado de aplicar la función anterior se observa que $p(x_t \mid o_t, o_{t-1}, ..., x_{t-1}, x_{t-2})$ converge más lentamente a la ubicación correcta, sin embargo resulta menos sensible a los errores de observación, ya que la hipótesis de localización no converge de inmediato, permitiendo descartar aquellas ubicaciones erróneas (Fig. 6.6).

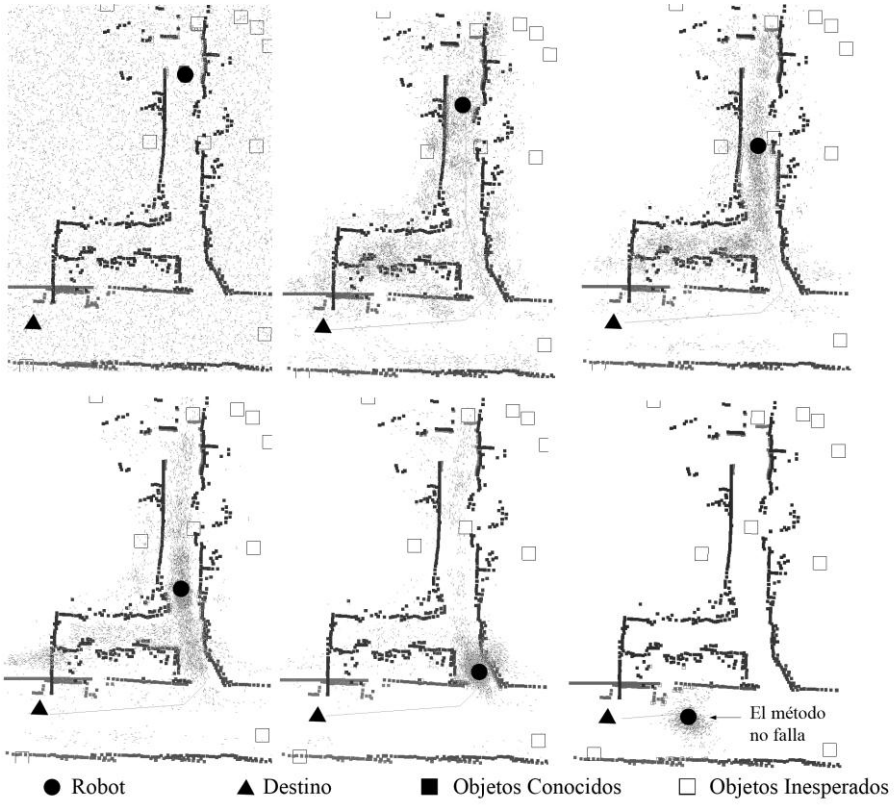


Figura 6.6. Prueba del Método de Monte Carlo con Conteo Simple de Correspondencias.

Con base en las pruebas anteriores, se concluye que la determinación de la función de evaluación de $p(o \mid x)$ representa un factor fundamental para garantizar la convergencia del método de localización de Monte Carlo. Mientras que una función muy "rígida" en términos de distancia entre observaciones podrá presentar una solución que converja rápidamente, no produce necesariamente una localización correcta, mientras que una función de distancia muy "laxa" (poco tolerante a errores) será más propensa a proporcionar la ubicación correcta pero luego de recorrer una distancia considerable.

6.3 Análisis Comparativo de los Métodos de Localización Implementados

Con base en las pruebas de los métodos implementados se obtuvo la siguiente tabla comparativa:

Métodos Analizados:

1.- Búsqueda secuencial de la observación más similar al conjunto almacenado.

2.- Red Neuronal de asociación directa.

3.- Localización por alineamiento de lecturas láser.

4.- Monte Carlo con simulación de lecturas y distancia Euclidiana.

5.- Monte Carlo con lecturas almacenadas previamente, búsqueda en árboles kd y distancia Euclidiana.

6.- Método de Monte Carlo con lecturas almacenadas, árboles kd para obtener de la observación esperada y conteo simple de correspondencias.

7.- Método de Monte Carlo con lecturas almacenadas, árboles kd para obtener de la observación esperada y modelo probabilístico de sensor.

Método	Tolera Oclusiones	Secuencia de Observaciones y comandos	Hipótesis Multi-modal	Adaptabilidad a Recursos de Hardware	Consumo de Recursos	Represen-tación continua
1. Búsqueda	No	No	No	No	Media	Si
2. Red FFN	No	No	No	No (al reducir la arquitectura se incrementa el error)	Media	Si
3. Alineamiento Láser	Si	Solo las 2 recientes	No	Si	Bajo	Si
4. MCL con simulación en t. real y dist. Euclidiana.	No	Si	Si	No	Alto	Si
5. MCL con base de datos, árboles kd y dist. Euclidiana.	No	Si	Si	No	Alto	Si
6. MCL con b.d., árboles kd y conteo de correspondencias	Media <50% ruido	Si	Si	No, pero aumenta su rapidez	Alto	Si
7. MCL con árboles kd y modelo de sensor	Media <50% ruido	Si	Si	No, disminuye su rapidez	Muy Alto	Si

Tabla 6.1. Comparativo entre métodos de localización con láser

Como resultado de la implementación de los métodos se encontraron los siguientes hallazgos:

1.- La búsqueda simple secuencial falla en la presencia de obstáculos inesperados. En ocasiones puede localizar al robot en un sitio completamente distinto al real, si existen observaciones muy similares para más de una ubicación. Presenta una sola hipótesis.

2.- La red FFN falla en la presencia de obstáculos inesperados. Si se presenta un error en alguna lectura no contemplado en el entrenamiento, de igual forma proporciona una ubicación errónea, al no contemplar el conjunto de movimientos sucesivos del robot. Presenta una sola hipótesis.

3.- El método de localización por alineamiento de lecturas del láser tuvo el mejor rendimiento para el caso de obstáculos inesperados e inclusive en movimiento. Sin embargo este algoritmo no es susceptible de ser utilizado para el problema de la localización global ya que proporciona únicamente los desplazamientos relativos entre movimientos consecutivos.

4.- MCL con la generación de la observación supuesta por simulación en tiempo real y distancia Euclidiana como medida de distancia, a pesar de no requerir recursos de memoria elevados para almacenar lecturas, requiere de un excesivo poder de cálculo (ray tracing) para sensores de tipo láser como el simulado, ya que en cada iteración al menos 10,000 partículas son reevaluadas para un mapa geométrico con 300 objetos para 68 lecturas del sensor, es decir 204 millones de cálculos por iteración. La distancia Euclidiana no obstante es un cálculo relativamente rápido, no resulta tolerante a oclusiones.

5.- MCL Con Árboles Kd para optimizar la búsqueda sobre el conjunto simulado de lecturas y distancia Euclidiana mejora desempeño en cuanto a rapidez de operación. Su rendimiento es adecuado para mapas estáticos, sin embargo falla de inmediato al existir obstáculos inesperados en movimiento al carecer de un modelo de sensor.

6.- MCL Con Árboles Kd y conteo de correspondencias obtuvo el mejor desempeño en cuanto a rapidez de operación. Frente a obstáculos inesperados tiene mejor rendimiento que la distancia Euclidiana.

7.- MCL Con Árboles Kd y el modelo de sensor propuesto por los autores, mejora la tolerancia a oclusiones y obstáculos inesperados, sin embargo en ocasiones falla al estar dichos objetos muy cerca del robot, bloqueando una gran cantidad de lecturas. El uso del modelo de sensor aumenta notoriamente el tiempo de actualización, ya que en cada iteración es necesario evaluar la ecuación (6.3) en la fase de actualización.

6.4 Localización Robótica para Robots Humanoides Futbolistas

Como resultado de la investigación y trabajos realizados, se decidió evaluar el problema de localización robótica bajo la perspectiva de los robots humanoides futbolistas, ya que visualmente dicho entorno de trabajo se encuentra mucho más restringido que el entorno de operación de un robot de servicio.

6.4.1 Reducción de la Complejidad del Ambiente de Operación

Es posible trasladar el problema de la localización robótica al escenario de los robots futbolistas para la competencia Robocup, donde se posee un ambiente de trabajo más controlado como el que se muestra en la figura 6.7.

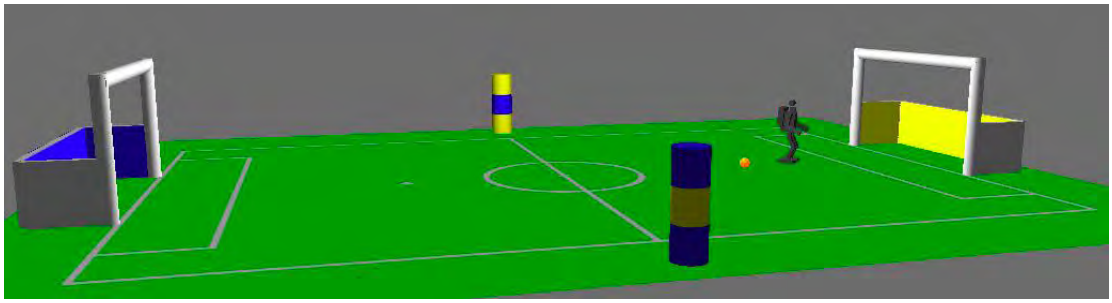


Figura 6.7. Ambiente de operación de los robot futbolistas.

Como puede observarse en la figura anterior, las dimensiones del terreno de juego son fijas (7 × 5 m), se poseen un par de marcas cilíndricas en posiciones fijadas de antemano y las porterías poseen un color característico. De igual forma los robots participantes deben ser de color predominantemente negro y poseer una marca distintiva (recuadro de color magenta o cian) para identificar a los robots del mismo equipo. Por otro lado la superficie por la cual se desplaza el robot posee un color uniforme (alfombra verde) y se encuentra marcada por líneas de un grosor uniforme (5 cm) que no se modifican durante toda la competencia.

Ventajas

Dado que el objetivo principal de la presente tesis se centra en el método neuronal de aprendizaje de la posición, más que en el procesamiento de la visión, el restringir la operación del robot a este tipo de escenario permite el uso de técnicas menos exigentes en cuanto a hardware como la segmentación por color o la extracción de características como líneas o esquinas ya que se poseen áreas de color uniforme tanto en los objetos del terreno de juego como en los propios robots.

Desventajas

Sin lugar a dudas la principal desventaja de utilizar esta plataforma consiste en la limitada capacidad de procesamiento disponible, ya que en este caso la competencia restringe a que todo el procesamiento se haga sobre el mismo robot, sin embargo, para el presente trabajo de tesis no se tomará en cuenta dicha restricción y se utilizará un equipo portátil conectado directamente al robot humanoide, ya que se pretende desarrollar un método suficientemente robusto, sin embargo se respetará el tipo de cámara utilizada por los robots futbolistas y el tipo de sensores permitidos en la competencia (giroscopio, acelerómetro y/o brújula electrónica). Por esta razón no se utilizarán sensores de tipo láser, sonar o infrarrojo.

Desde el punto de vista de localización el problema más complicado de resolver se refiere a la nula odometría del robot, ya que en este caso se trata de robots bípedos de tipo humanoide (figura 6.8).

Figura 6.8. Robot Humanoide Futbolista

Este tipo de robots no poseen un control de odometría ya que, a diferencia de un robot de servicio que se desplaza sobre ruedas, no se garantiza en ningún momento su desplazamiento sobre la superficie de juego debido a constantes choques y caídas. Lo único que se puede hacer es estimar una velocidad aproximada de desplazamiento en función de la instrucción de movimiento realizada por el robot (como correr, patear o simplemente permanecer inmóvil).

El problema más complicado a resolver será el hecho de que el robot en cualquier momento puede encontrarse boca abajo, boca arriba o mirando hacia abajo con cierto grado de inclinación que no le permita ver más que el terreno de juego, lo que haría sumamente complicado el poder establecer una observación "supuesta".

Lo anterior representa un problema para los métodos Bayesianos de localización como el método de Monte Carlo basados en estimaciones con respecto a la posición, sin embargo, dado que tanto las marcas como las líneas del campo de juego permanecen constantes durante todo el juego, éstas son susceptibles de ser tomadas en cuenta al

momento de obtener la observación actual que, dicho sea de paso, deberá ser una simplificación de la imagen observada por la cámara.

Con base en el criterio anterior, el problema de la localización se extenderá a un entorno de dimensión 5, es decir

$$\bar{X}_{C_t} = (x_c, y_c, f, j, y) \tag{6.8}$$

donde (x_c, y_c) corresponde a la posición del centro óptico de la cámara sobre el terreno de juego y (f, j, y) corresponde al ángulo *Pan*, *Tilt* y *Roll* de la cámara con respecto al sistema de coordenadas del mundo. La expresión anterior indica exclusivamente la localización exacta de la cámara con respecto al sistema de coordenadas del mundo, sin embargo lo que se pretende es encontrar la ubicación y orientación del robot con respecto a ese sistema de coordenadas.

Para encontrar la posición y orientación del robot dentro del terreno de juego

$$\bar{X}_{r_t} = (x_r, y_r, q_r) = f(x_c, y_c, q, f, j) \tag{6.9}$$

se utilizarán los ángulos de flexión de los motores que sostienen la cámara y las dimensiones del robot para realizar una transformación afín. Ambos parámetros pueden ser estimados. Además no se olvide que el robot puede incorporar un acelerómetro de 3 ejes que permita calcular la inclinación del torso en todo momento.

6.4.2 Localización con Filtros de Partículas para Robots Humanoides

Al igual que para el caso de los robots de servicio uno de los métodos más utilizados es el Método de Localización de Monte Carlo. En la actualidad diversos métodos son utilizados para procesar las imágenes y determinar $p(o \mid x)$, entre los cuales podemos citar desde detección de marcas mediante segmentación por color [Bruce 00], detección de líneas [Rofer 03] o transiciones de color, esquinas, uniones e intersecciones

de líneas [Bais 06]. En su mayoría dichos métodos relacionan el modelo de observación de dichas características respecto a la posición del robot, ya sea mediante triangulación simple o mediante un modelo predictivo de proyección tridimensional.

6.4.3 Implementación del Método de Localización MCL para Robots Humanoides

Con fines comparativos se implementó el método de Monte Carlo, tomando exclusivamente como observación la imagen proporcionada por una cámara modelo CMUCam3 desarrollada por la Universidad de Carneggie Melon en E.U.A (Fig. 6.9). Dicha cámara incorpora 64kb de memoria RAM, un procesador ARM a bordo y permite la programación en lenguaje C de rutinas de procesamiento de imagen y toma de decisiones mediante un compilador que se ejecuta bajo la plataforma Cygwin (en Microsoft Windows) y en Linux.

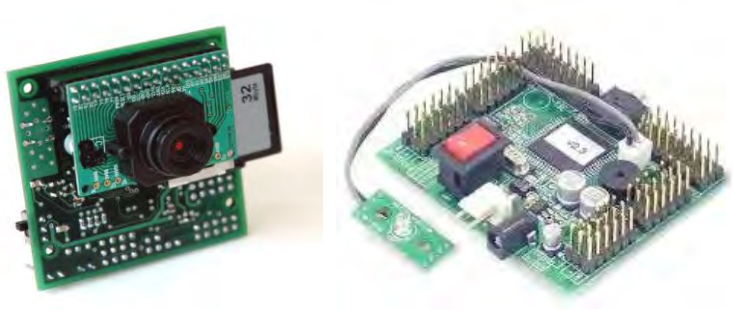


Figura 6.9. Sensor de visión CMUCAM y tarjeta MR-C3024 de control del robot.

Dicho sensor de visión posee un puerto serial incorporado, mediante el cual es posible enviar comandos de control a una tarjeta de control Micom MR-C3024 de la empresa Hitec, que integra un microcontrolador AT-MEGA 128 con 64kb en memoria flash y se ha programado con el código necesario para controlar un robot bípedo con 24 grados de libertad para realizar movimientos de caminado, giros, pateo y levantarse boca abajo y boca arriba, entre otros.

6.4.4 Detección de Marcas en el Entorno de Operación del Robot

Dadas las limitaciones de procesamiento de la CMUCam, básicamente se utiliza el método propuesto en [Bruce 00] para segmentar de manera rápida la escena mediante

umbrales RGB o YUV en 8 colores (cian, magenta, amarillo, azul, naranja, verde, blanco y negro), bajo un espacio de color YCrCb que puede ser obtenido de forma nativa por el sensor de visión CMOS de la CMUCam. La segmentación se realiza principalmente mediante rangos de color que son procesados fuera de línea para las 8 clases y almacenados en vectores de 256 dimensiones.

La rapidez de la segmentación radica en que sustituye las comparaciones del valor actual en la imagen por una consulta en un arreglo para cada componente de color del píxel y una operación *AND*.

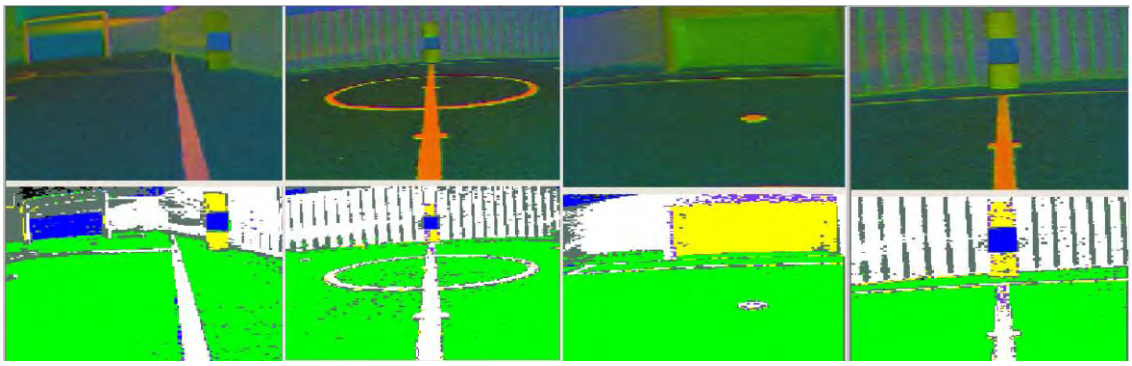


Figura 6.10. Segmentación por Color de Escenas de Juego.

Una vez Segmentada la imagen se ejecuta el algoritmo de formación de áreas de color conocido como RLE (*Run Lenght Encoded*) descrito en dicha publicación, que utiliza 4-conexionismo para conectar los píxeles segmentados en una sola área de color (figura 6.11).

Como resultado del método RLE completo descrito en [Bruce 00], se genera una lista ligada ordenada por área (en píxeles) para cada color. Dicha lista es tomada como referencia para detectar las porterías y marcas cilíndricas de color amarillo y azul.

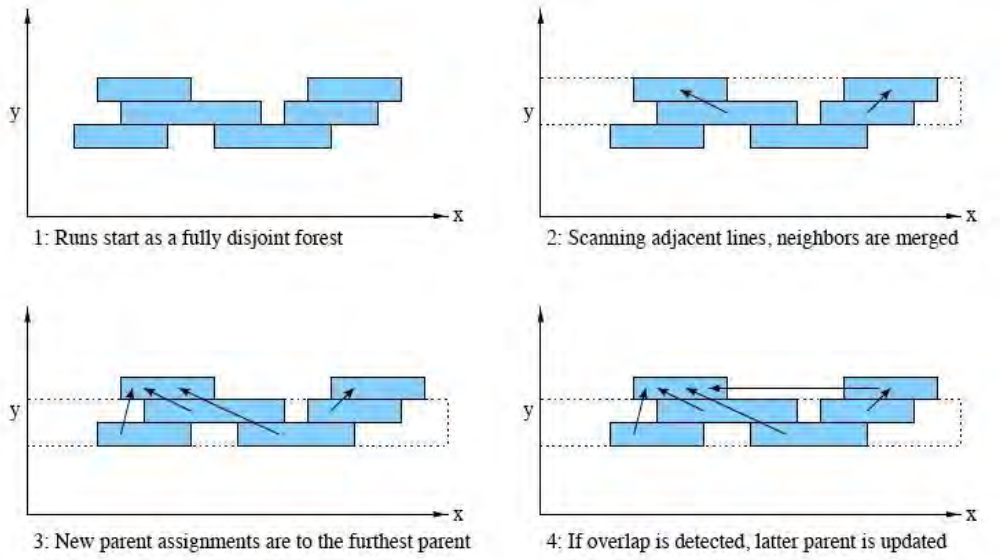


Figura 6.11. Algoritmo RLE (tomado de [Bruce 00]).

6.4.5 Modelo de Sensor

Como se indicó con anterioridad, el desempeño del método de MCL depende fundamentalmente del modelo de observación o $p(o|x)$. En este caso según lo indicado en [Sridharan 05] es posible efectuar una localización eficiente si $p(o|x_i)$ refleja la similitud entre la observación real y esperada para la posición x_i. Para tal efecto se construye un modelo simple de observación de los *landmarks* de la siguiente forma.

1.- Dado que es conocido el Campo de Visión o *FOV* de la Cámara , en principio es posible determinar para una ubicación y orientación arbitraria sobre el terreno de juego aquellas marcas (porterías y marcas cilíndricas) que se encuentran dentro del campo de visión del robot, siempre y cuando el ángulo de inclinación de la cámara permita observar dicha marca, para lo cual es posible hacer que el robot mire directamente hacia el frente al momento de pretender localizarse (figura 6.12).

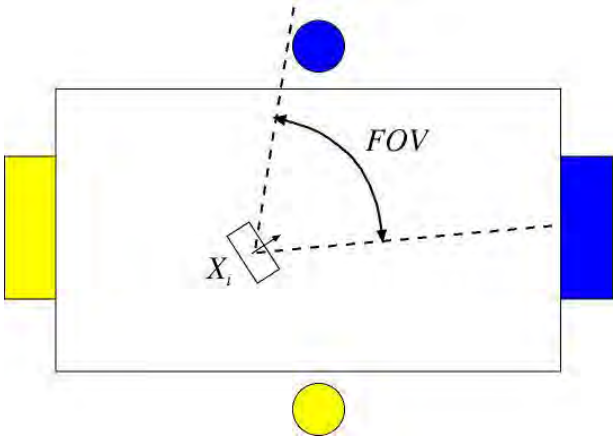


Figura 6.12. Determinación de Marcas Observables.

2.- Una vez que se han determinado los *landmarks* que deben de ser percibidos por el robot en su ubicación supuesta (es decir, para cada partícula del filtro MCL) es posible determinar una función de distancia como la que se muestra a continuación:

$$p(o \mid x) = \frac{l_{match}}{l_{total}} \tag{6.10}$$

donde

$$l_{match} = \sum_{i=1}^{4} \begin{cases} 1 : si \mid l^i_{real} - l^i_{\sup} \mid = 0 \\ 0 : si \mid l^i_{real} - l^i_{\sup} \mid \neq 0 \end{cases} li \in [0,1] \text{ y } l_{total} = 4 \tag{6.11}$$

es decir, mediante un conteo simple de los landmarks que se observan y que deberían observarse para la ubicación supuesta x_i. Estas ecuaciones son análogas a las ecuaciones (6) y (7) vistas para los robots de servicio. De ahí la importancia de evaluar su desempeño previamente.

6.4.6 Modelo de Planta

Como modelo de planta se determinó una velocidad de avance del robot (3 ± 1 cm/s) para aquellos comandos de caminar y correr en línea recta, mientras que se estableció una velocidad de giro de 15 ± 5 grados/s para aquellos comandos de girar. Para el comando de patear se estableció un desplazamiento del robot de (0±2 cm, 0±5º).

Al probar en el simulador, como era de esperarse, si el robot inicia su recorrido desde una ubicación desconocida, le toma mucho tiempo (80 segundos en promedio y un par de metros de recorrido) el localizarse correctamente (Fig. 6.13), debido a la modelo de sensor utilizado. Dado que un partido de futbol de robots cada tiempo dura 10 minutos, resulta excesivo el tiempo de localización si tomamos en cuenta que continuamente durante el juego hay interrupciones y desplazamiento de los robots, debido a colisiones y faltas.

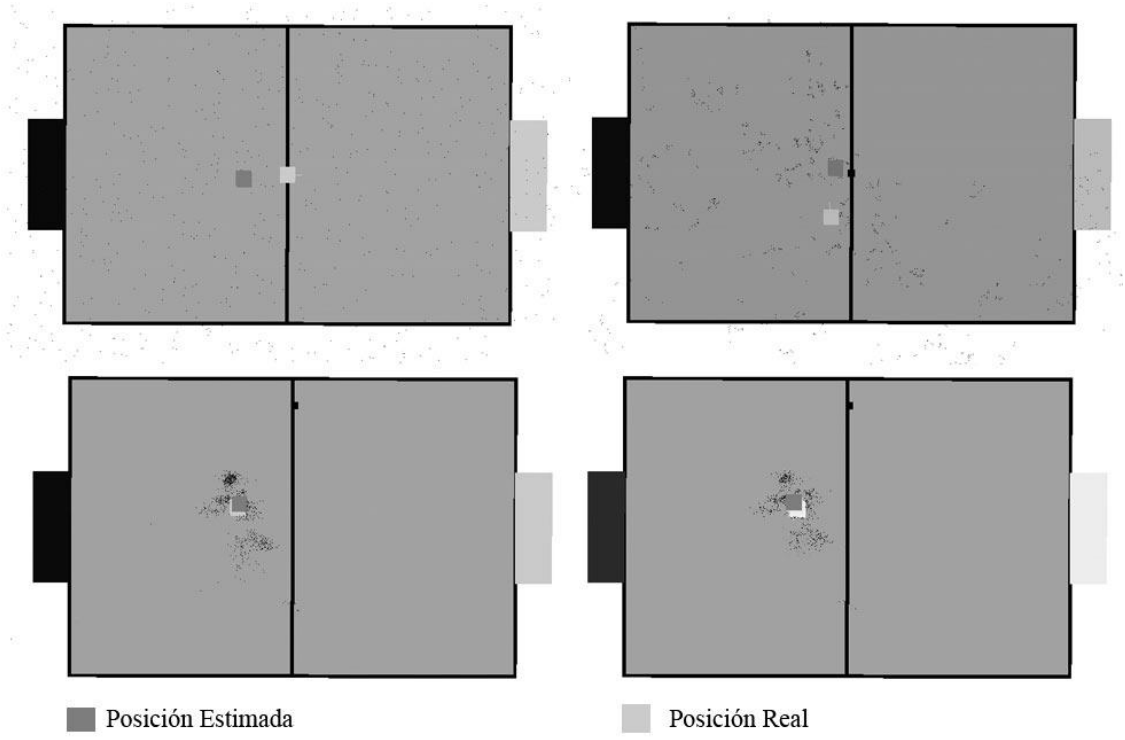


Figura 6.13. Operación del Método MCL para Robots Humanoides Futbolistas.

En conclusión, a pesar de que es posible localizar resultaría conveniente determinar visualmente la distancia a la cual se observa cada marca y la cantidad de pixeles visibles para integrarlo al modelo de sensor, en lugar de sólo tomar en cuenta si es observada o no cada marca, lo cual pudiese hacerse bajo un modelo donde sea predicha el área en píxeles que debe tener cada región de color detectada, para cada posición supuesta x_i.

Capítulo 7

Localización de Viterbi basada en Odometría

7.1 Introducción

El Apéndice A muestra (con el amable permiso de Springer Science+Business Media: Journal of Intelligent & Robotic Systems, DOI: 10.1007/s10846-011-9627-8) el artículo *in extenso* donde se presenta una amplia introducción y todos los detalles referentes al método desarrollado en el presente trabajo denominado "Localización de Viterbi basada en Odometría", OVL por sus siglas en inglés. El presente capítulo se centrará el los aspectos más relevantes del mismo para facilitar su comprensión y se discutirá el método propuesto así como la metodología seguida en su desarrollo. Si se desea abundar en los detalles se recomienda leer el artículo completo en dicho Apéndice.

7.1 Localización de Viterbi en Modelos Ocultos de Markov

Según lo propuesto en [Savage 05], si se considera cada ubicación $\boldsymbol{x_i}$ del robot (nodos de un mapa topológico) como un estado discreto (x_i, y_i, θ_i) dento del conjunto X y en cada estado se obtiene una observación o símbolo discreto s_k dentro del conjunto de símbolos observables Z, analizando la secuencia de símbolos observados (s_0, s_1, ... , s_t) es posible determinar, utilizando Modelo de Markov de Variable Oculta donde la variable oculta o *estado* es la ubicación o *pose* del robot, la secuencia de estados (ubicaciones discretas) más probable que haya generado las observaciones y con ello la posición actual (último estado observado) del robot.

Un HMM λ se denota por

$$\lambda = (A, B, P)$$

A = $\{a_{ij}\}$ es una matriz de distribución probabilidad de transición entre estados sobre $X \times X$, tal que

$$a_{ij} = p(x_t = x_j \mid x_{t-1} = x_i), \quad 1 \le i \le n \tag{7.1}$$

donde n es el número de estados en el modelo.

B = $\{b_j(k)\}$ es la matriz de distribución de probabilidad de observación de símbolos por cada estado $X \times Z$, tal que

$$b_j(k) = p(z_t = s_k \mid x_t = x_j), \quad 1 \le k \le m \tag{7.2}$$

donde z_t es el vector de observación en el tiempo t y s_k es el *k-ésimo* símbolo de la lista de posibles símbolos a observar, mientras que m es el número de símbolos observables.

Finalmente, $\Pi = \{\pi_i\}$ es el vector que indica la probabilidad inicial para cada estado

$$p_i = p(x_t = x_i), \quad 1 \le i \le n \tag{7.3}$$

Según lo anterior, para cada estado x_i del modelo λ deberá existir cierta probabilidad de observar cada uno de los símbolos $b_j(k)$, cierta probabilidad para cambiar de estado a_{ij} y, finalmente, una probabilidad inicial para cada uno de los estados π_i.

7.2.1 Ejemplo de Modelo de Markov de Variable Oculta (HMM)

A continuación se muestra un sencillo ejemplo de un HMM para la estimación del estado del clima en función de las observaciones.

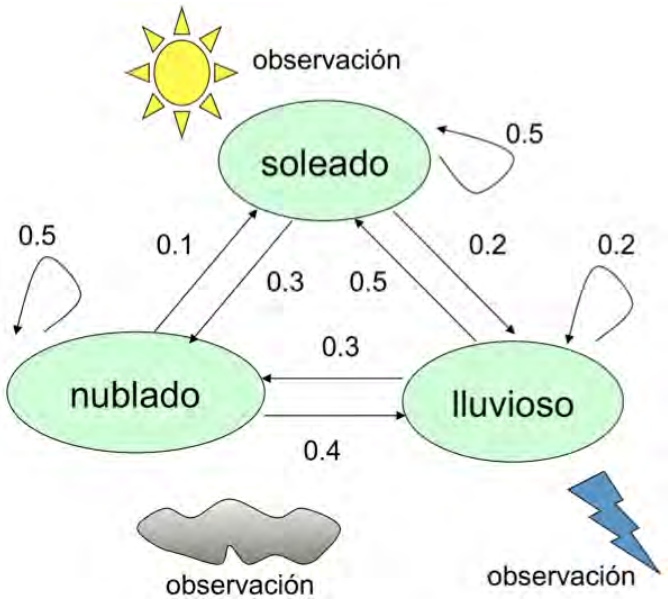


Fig. 7.1. HMM para el estado del tiempo $\lambda = (A, B, P)$. Tomado de [Rabiner 89].

Para cada estado del modelo λ existe cierta probabilidad de observar cada uno de los símbolos $B = \{b_{jk}\}$, cierta probabilidad para cambiar de estado $A = \{a_{ij}\}$ y una probabilidad inicial para cada uno de los estados $P = \{p_i\}$.

En este sencillo ejemplo es posible determinar que el número total de estados es tres: *soleado (1), nublado (2)*, y *lluvioso (3)*. De esta forma $1 \le i \le 3$, $1 \le j \le 3$ y por lo tanto la probabilidad de transición entre estados será

$$A = \{a_{ij}\} = \begin{bmatrix} .5 & .2 & .3 \\ .5 & .2 & .3 \\ 1 & .4 & .5 \end{bmatrix}$$

Dado que sólo existen tres símbolos (observaciones): *sol (1)*, *nube (2)* y *rayo (3)* entonces $1 \le k \le 3$ y en consecuencia

$$B = \{b_{jk}\} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Finalmente, si se desconoce la situación inicial del clima al utilizar el modelo se tiene que

$$P = \{p_i\} = \begin{bmatrix} .333 \\ .333 \\ .333 \end{bmatrix}$$

En el ejemplo descrito anteriormente se asume que cada símbolo será observado exclusivamente en cada uno de los estados, por lo tanto el ejemplo resulta trivial, sin embargo en la práctica puede suceder que más de un símbolo sea observado en cada estado. Por ejemplo

$$B = \{b_{jk}\} = \begin{bmatrix} 0.7 & 0.29 & 0.01 \\ 0.2 & 0.7 & 0.1 \\ 0.01 & 0.5 & 0.4 \end{bmatrix}$$

es un caso más realista en donde cada símbolo es observado en cada uno de los estados con cierta probabilidad mayor que cero.

7.2.2 Algoritmo de Viterbi

El algoritmo conocido como de Viterbi [Viterbi 67], permite determinar el conjunto de estados $\{x_0, x_1, ..., x_t\}$ que generan una serie de observaciones $\{z_1, z_2, ..., z_t\}$, basado en maximizar la probabilidad de observación de cada símbolo en cada estado del recorrido junto con la máxima probabilidad de transición entre estados

$$d_t(i) = \max_{x_1, x_2, ..., x_{t-1}} p(x_0, x_1, ..., x_t = i, z_1, z_2, ..., z_t \mid l)$$

El Algoritmo de Viterbi tiene dos formas de calcular $p(O|\lambda)$, el procedimiento hacia adelante o *progresivo* y hacia atrás o regresivo.

El procedimiento progresivo de Viterbi (*Viterbi Forward Procedure* VFP) tiene tres etapas:

Inicialización

$$a_1(i) = p_i b_i(z_1), \quad 1 \le i \le n \tag{7.4}$$

$\alpha_1(i)$ almacena la probabilidad conjunta $p(x_i, z_1)$ de estar en el estado i en t=0 y observar z_1, igual a la probabilidad π_i de iniciar en el estado i por la probabilidad de observar el símbolo z_1 en dicho estado, mientras n es el número de estados en el modelo.

Inducción

$$a_t(j) = b_j(z_t)\begin{cases}\sum_{i=1}^{n} a_{ij} a_{t-1}(i) & 1 \le t \le t_k \\ & 1 \le j \le n\end{cases} \tag{7.5}$$

donde t_k es el instante de tiempo discreto cuando la última observación z_k es realizada.

$\alpha_t(j)$ calcula la probabilidad conjunta $p(x_j, z_{0:t})$ de estar en el estado i en el tiempo t observando z_t. Al sumar la probabilidad $p(x_j \mid x_i, z_{0:t-1})$ de pasar al estado j a partir de cualquier estado i (probabilidad total), se obtiene $p(x_j\ , z_{0:t-1})$. Si este resultado se multiplica por la probabilidad de observar el símbolo z_t en el estado j entonces se obtiene $p(x_j, z_{0:t})$, porque $p(x_j, z_{0:t}) = p(z_t \mid x_j)\, p(x_j, z_{0:t-1})$.

Terminación

$$p(O \mid \lambda) = \sum_{i=1}^{n} a_t(i) \tag{7.6}$$

donde

$$a_t(i) = p(x_t = x_i, z_0, z_1, ..., z_t \mid \lambda). \tag{7.7}$$

Si se reescribe la Ec. (7.6), de acuerdo con (7.3) y (7.4) se obtiene

$$a_t(j) = p(z_t \mid x_j) \sum_{i=1}^{n} p(x_j \mid x_i)\, a_{t-1}(i) \tag{7.8}$$

$$= p(z_t \mid x_j)\, p(x_j,\ z_{0:t-1}) = p(z_{0:t}, x_j)$$

se puede notar que la Ec. (7.8) calcula la probabilidad conjunta de la secuencia de observaciones hasta z_t terminando en el estado x_j. Al sumar todas las $\alpha_t(j)$ con $1 \le j \le n$, se obtiene probabilidad de la secuencia de observaciones $p(z_{0:t})$ o la dado λ (Ec. 7.6).

El problema fundamental consiste en la determinación óptima de las matrices de probabilidad , ya que sería necesario que el robot efectuase la mayor cantidad posible de recorridos para determinarlas tal como se propone en [Savage 05], lo cual no siempre es factible y será motivo de una adaptación al método descrita en posteriores secciones.

Por otra parte, si se utiliza un mapa de ocupación espacial en forma de malla cuyas dimensiones sean conocidas, cada estado puede estar representado como una celda de dicha malla y se sabrá de antemano la cantidad de estados *n* del modelo λ.

7.2.3 Transición entre Estados

Según lo descrito en [Savage 05], cuando el robot se mueve aleatoriamente, dependiendo de la disposición de los objetos en el ambiente existe la posibilidad de ocupar ciertas posiciones o estados durante el recorrido y obtener la probabilidad de transición entre dichos estados a_{ij}. De esta forma habrá una mayor probabilidad de transición entre estados vecinos que entre estados (nodos) alejados unos de otros.

De igual forma, si se cuenta con una representación métrica (mapa) del ambiente, se propone en dicho artículo calcular a priori la probabilidad de transición mediante simulación del movimiento del robot siguiendo trayectorias aleatorias. Con ello se determinaría la matriz de transición entre estados **A**.

En cuanto a la matriz de observación de símbolos, Savage et al. proponen utilizar cuantificación vectorial para reducir la dimensionalidad del vector de observación (un anillo de 16 sonares) a un sólo símbolo o *vector tipo* s_k. Mediante el mismo proceso de simulación, mediante el uso de un mapa métrico y *ray tracing* se estimaría las observaciones supuestas para cada ubicación (estados discretos) y la correspondiente probabilidad de observación de símbolos denotada por la matriz **B**.

Finalmente, sería posible fijar la misma probabilidad inicial $\boldsymbol{\pi}_i = 1/n$ para cada uno de los estados o nodos navegables.

7.2.4 Obtención de la Secuencia de Estados Óptima con el Algoritmo de Viterbi

La operación del algoritmo de Viterbi puede ser vista mediante el siguiente diagrama:

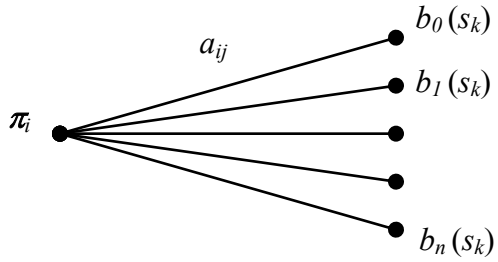


Fig. 7.2. Diagrama Trellis del Procedimiento Progresivo de Viterbi VFP.

Por cada estado i con $1 \leq i \leq n$, se tiene una probabilidad inicial π_i una probabilidad a_{ij} de transición a otro estado j y una probabilidad $b_j(s_k)$ de observar el símbolo s al tiempo k en el nuevo estado j.

Por cada símbolo observado al instante t se calcula

$$\delta_t(i) = \max_{x_1, x_2, \ldots, x_{t-1}} p(x_0, x_1, \ldots, x_t = i, z_1, z_2, \ldots, z_t \mid l) \tag{7.9}$$

si se consideran las observaciones como eventos independientes entre sí y sólo dependientes del estado actual (Propiedad de Markov), entonces

$$\delta_0(i) = \max_{1 \leq i, j \leq n} [\pi_i \times a_{ij} \times b_j(s_1)]$$

$$\delta_{t+1}(i) = \max_{1 \leq i, j \leq n} [\delta_t(i) \times a_{ij} \times b_j(s_{t+1})]$$

$$x_t = \arg\max_{1 \leq i \leq n} [\delta_t(i)]$$

De esta forma es posible con el VFP de manera recurrente calcular para cada símbolo observado, con base en las probabilidades inicial, de transición y de observación, así como en las observaciones anteriores, el estado actual del robot más probable.

7.2.5 Desventajas de la Localización de Viterbi contra los Filtros de Partículas

Luego de la investigación realizada, es posible concluir que la localización de Viterbi presentada en [Savage 05] presenta serias desventajas contra los métodos del estado del arte en localización robótica probabilística. Por citar las más importantes:

a) Aproxima el modelo de movimiento del robot $p(x_j \mid x_i)$ a partir de una función de densidad de probabilidad o PDF calculada mediante la simulación aleatoria de los movimientos del robot, descartando uno de los parámetros más importantes involucrados en la estimación de la posición del robot (especialmente para hacer el rastro de la posición o *Position Tracking*): la estimación del desplazamiento del robot con la odometría o u_t. La falta de dicha información hace al método incapaz de decidir entre dos ubicaciones aledañas a la ubicación real del robot con observaciones similares z_t ubicadas a la misma distancia de la ubicación previa $\boldsymbol{x}_{t-1}$, algo muy común en pasillos largos.

b) Hace una estimación discreta, esto es, localiza al robot en estados discretos o posiciones absolutas contra la ubicación continua que proporcionan los métodos de Kalman y de Monte Carlo.

c) En la determinación de la secuencia de estados más probable (Ec. 7.9) se descartan las trayectorias menos factibles de acuerdo a la probabilidad de observación, es decir, sólo se conserva la trayectoria más probable de ocurrencia. Es claro que si por errores en los sensores se observa un símbolo poco o nada frecuente para cierto estado, la localización falla al descartar el estado correcto al observar un símbolo poco probable.

7.3 Mejoras a la Localización de Viterbi para manejar Ubicaciones Continuas

La primera mejora importante a la Localización de Viterbi consistió en modificar el propio algoritmo de Viterbi para poder hacer una estimación continua de la posición del robot, basándonos en los Filtros de Partículas como el Método de Monte Carlo, donde se calcula la ubicación continua del robot a partir de un conjunto de muestras o "partículas" discretas mediante el concepto del valor esperado con la expresión

$$x_t = \sum_{i=1}^{n} x_i \times p(x_i \mid z_{0:t}) \tag{7.10}$$

para lo cual es necesario calcular la probabilidad de estar en cada estado del modelo al tiempo *t*.

Dado que el procedimiento progresivo de Viterbi o VFP calcula en cada iteración $p(z_{0:t}, x_j)$ con la Ec. (7.8), si este resultado se divide en cada iteración entre $p(z_{0:t})$ calculado con la (Ec. 7.6) el resultado es

$$\frac{p(z_{0:t}, x_j)}{p(z_{0:t})} = p(x_j \mid z_{0:t}) = Bel(t) \tag{7.11}$$

que corresponde exactamente con la estimación de ubicación calculada por el Método de Monte Carlo. Dado que la ubicación de los estados $\boldsymbol{x_i}$ es conocida y permanece fija durante toda la operación del método, con estas dos sencillas adaptaciones es posible hacer que la Localización de Viterbi maneje ubicaciones continuas al igual que los métodos del estado del arte.

7.4 Integración de la Odometría en el Cálculo de la Transición entre Estados

El siguiente paso consistió en modificar el algoritmo para integrar la odometría, lo que representó hacer que la matriz de transición entre estados **A** fuese dependiente de la odometría, lo cual supone calcularla en cada iteración del algoritmo justo después de recibir la información de desplazamiento u_t estimada con la odometría del robot.

Como a_{ij} debe representar la probabilidad de transición $p(x_j \mid x_i, u_t)$ entre dos estados dada la odometría u_t y cada estado corresponde con cierta región del espacio, si se considera una malla de estados como la mostrada en la Fig. 7.3 y un desplazamiento u_t, entonces es posible determinar una probabilidad de transición del estado i al estado j dependiendo de la distancia entre el centro del nodo i y el nodo j. A primera vista el considerar únicamente el centro de cada región (estado) pudiese parecer inapropiado, debido a que la probabilidad de transición entre estados estaría relacionada con el traslape de áreas entre los puntos dentro de la región comprendida por el estado i más el desplazamiento u_t con el área comprendida por el estado j. En la figura 7.3 se observa cómo el traslape de áreas se relaciona con que tan próximo resulta $x_i + u_t$ con x_j.

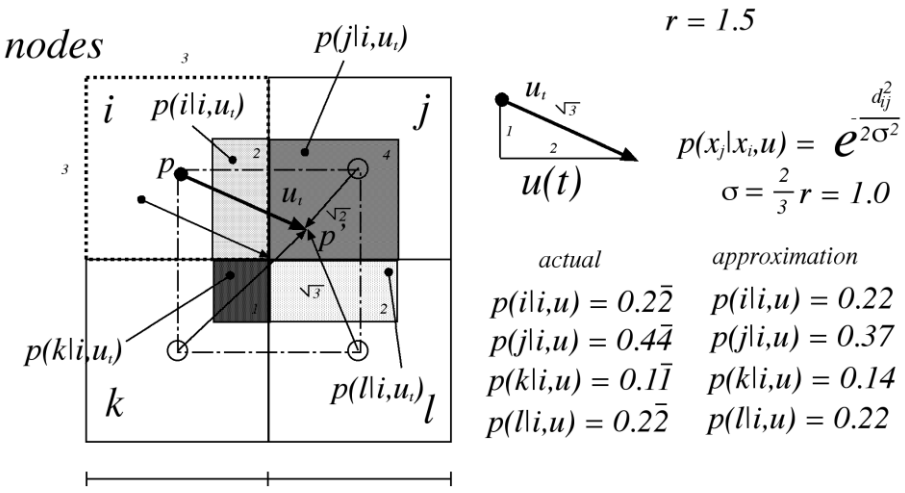


Fig. 7.3. Aproximación de la probabilidad de transición entre estados dada la odometría.

De esta manera, puede verse fácilmente que la probabilidad de pasar del estado i al estado j dada la odometría u_t, es proporcional a

$$d_{ij} = \left| x_i + u_t - x_j \right| \tag{7.12}$$

y con ello nos es posible hacer una aproximación mediante una PDF Gaussiana tal y como se indica en la Fig. 7.3. mediante

$$p(x_j \mid x_i, u_t) = e^{-\frac{d_{ij}^2}{2s^2}} \tag{7.13}$$

donde σ indica el grado de inclusión y correspondería para el caso de mapas con estados en una látice o rejilla cuadrada uniforme (mejor conocida como *gridmap*) a

$$s = \frac{2}{3} r \tag{7.14}$$

donde *r* es el radio del nodo.

Dado que la dimensión del estado $\boldsymbol{x} = (x, y, \theta)$ es tres, conviene determinar por separado un valor de σ para la orientación del robot (usualmente dentro del rango $[-\pi, \pi]$ o $[0, 2\pi]$) diferente del usado para las dimensiones *x* y *y*, debido a que generalmente estas últimas (expresadas en metros) resultan mucho mayores en valor absoluto para ambientes de operación reales (digamos 8 × 10 metros) con lo que la Ec. 7.14 ignoraría casi por completo diferencias grandes en la orientación del robot. De esta manera la Ec. 7.13 se transforma en

$$D(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\left(\frac{(x_j - x_i - u_{tx})^2}{2s_d^2} + \frac{(y_j - y_i - u_{ty})^2}{2s_d^2} + \frac{(q_j - q_i - u_{tq})^2}{2s_q^2}\right)} \tag{7.15}$$

Finalmente, sólo hace falta normalizar sobre la suma de todos los valores de transiciones posibles del estado *i* a cualquier estado *j*:

$$p(x_j \mid x_i, u_t) = \frac{D(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\sum_{j=1}^{n} D(\boldsymbol{x}_i, \boldsymbol{x}_j)} \tag{7.16}$$

con lo que es posible calcular cada elemento individual a_{ij} de la matriz de transición **A**, en función de u_t, es decir $\mathbf{A}(u_t)$.

7.5 Obtención de Símbolos mediante Mapas Autoorganizados

Para encontrar la matriz de probabilidad de observación de símbolos **B** es posible utilizar un cuantificador vectorial (implementado mediante una red neuronal de agrupamiento de Kohonen) para clasificar las diferentes mediciones obtenidas durante los

distintos recorridos del robot, proporcionando luego del entrenamiento el símbolo (índice dentro del arreglo neuronal) cuya observación o *vector tipo* (medición con los sensores del robot) resulte más similar a la observación actual del robot. Lo anterior resulta especialmente útil si se posee un gran número de mediciones como las obtenidas con un anillo de sonares o láser.

Mediante dicho cuantificador es posible representar un vector de *n* dimensiones como un símbolo único v_k que represente la observación realizada en el estado actual.

Es posible determinar al inicio en cuántas clases ó el cuantificador deberá clasificar las lecturas obtenidas, conociendo *a priori* el número *M* de símbolos deseados.

Si se analizan nuevamente las muestras obtenidas durante cada recorrido y se alimentan al cuantificador, es posible obtener la probabilidad de observar los diferentes símbolos v_k para cada estado, y construir la matriz *B* del modelo de Markov.

7.5.1 Implementación de Métodos de Reducción de la Dimensionalidad

Un aspecto importante visto en la investigación es la reducción de la dimensionalidad de las observaciones del robot, especialmente en el caso de sensores láser o visión computacional, ya que la alta dimensionalidad de las lecturas de dichos sensores hace complicada la tarea de clasificación, ajuste y entrenamiento en el caso de las RNAs.

Se hicieron distintas pruebas con métodos de reducción de la dimensionalidad basados en RNAs. Específicamente se probó una Red de Cuantificación Vectorial no supervisada y mapas autoorganizados de Kohonen en una versión unidimensional y bidimensional. Todas las redes se alimentaron con el conjunto de 170,000 lecturas simuladas del el ambiente de operación del robot utilizadas en los métodos de localización implementados (Fig. 7.4).

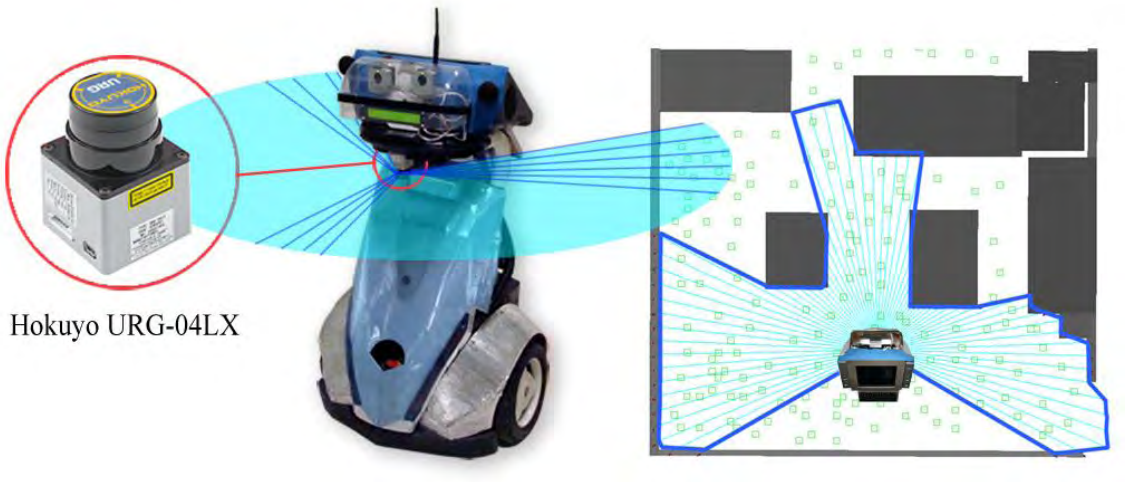


Fig. 7.4. (Izq.) Robot ARTURito y láser utilizado mostrando el plano de barrido. (Der.) observación simulada para el entrenamiento del Cuantizador Vectorial. Todas las lecturas se representan en el entrenamiento como si fuesen realizadas con el robot orientada hacia 90 grados.

El objetivo es observar el comportamiento de la observación proporcionada por la red, en presencia de ruido y oclusiones. El caso ideal sería encontrar una configuración donde la observación cuantificada varíe de alguna forma predecible o modelable en presencia de oclusiones (Fig. 7.5).

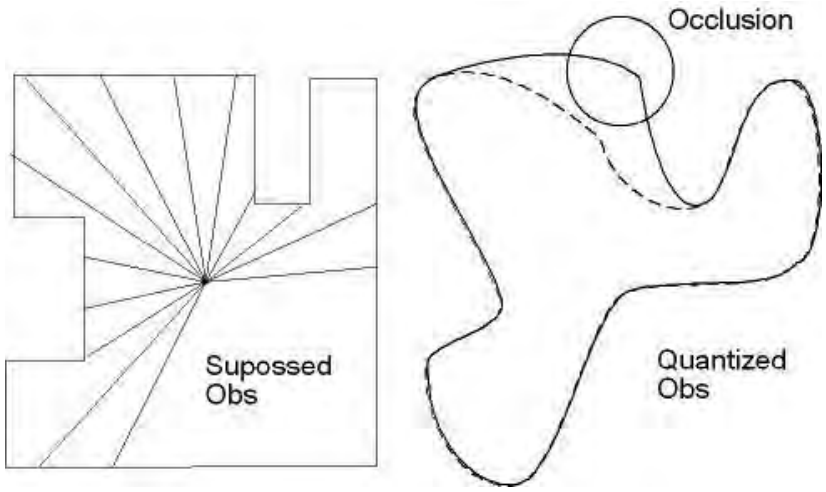


Figura 7.5. Ejemplo de observación supuesta (izq.) y observación cuantizada con oclusión (der.).

7.5.2 Red de Cuantificación Vectorial sin Relaciones de Vecindad

Las redes de cuantificación vectorial simple [Brío 02, p. 87] proporcionan la clasificación más similar a la lectura alimentada, no existe relación de similitud alguna entre los vectores tipo. De esta forma, si en cierto instante la red proporciona un vector tipo O_i, donde *i=2* por ejemplo, en presencia de ruido o un mínimo de oclusión, podría activarse otra neurona y proporcionar una salida activa o vector tipo O_j, donde *j=63* por ejemplo (Fig. 7.6). En este tipo de red, resultaría muy complicado predecir la salida en presencia tanto de ruido del sensor como de oclusiones debidas a objetos o personas en movimiento.

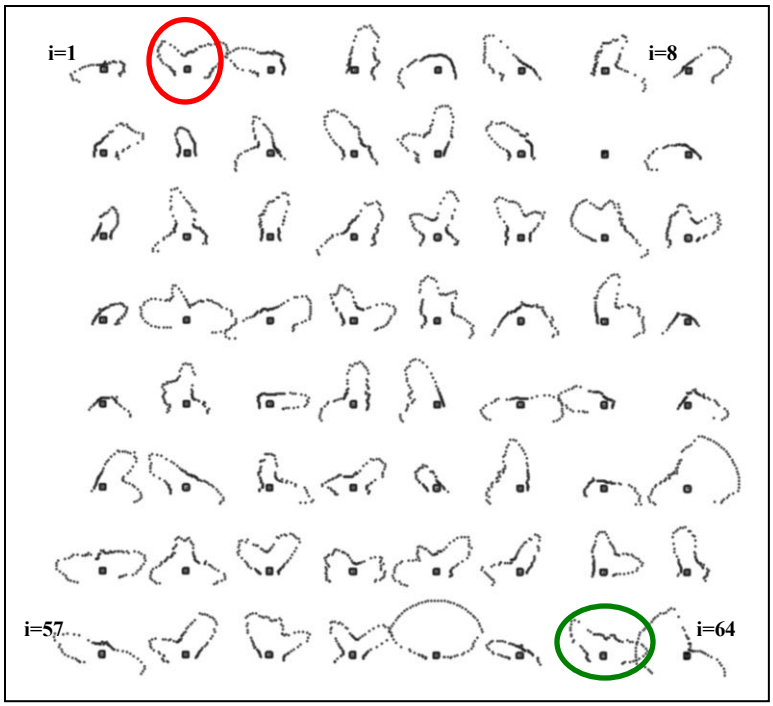


Figura 7.6. Resultado del entrenamiento de la Red Neuronal de Cuantificación Vectorial sin preprocesamiento de los datos y sin relación de vecindad. 64 vectores tipo. El vector tipo no. 2 y 63 se encuentran indicados por un ovalo rojo y verde respectivamente.

En la figura 7.6 se muestra el ejemplo anterior con el conjunto de vectores tipo obtenido durante el entrenamiento (300 épocas) de la Red de Cuantificación Vectorial no Supervisado (sin incorporar oclusiones). Se observa como vectores muy similares poseen índices muy distantes en la estructura (2 y 63 respectivamente). Por la misma razón este

tipo de Red no resulta adecuada para las tareas de reducción de dimensionalidad encaminadas a la localización.

7.5.3 Red de Kohonen Bidimensional

A continuación de implementó una Red de Kohonen 8 × 8 con relación de vecindad. Se alimentó nuevamente con el conjunto de 170,000 lecturas simuladas y se obtuvo la figura 7.7.

En esta figura se observa claramente como los perfiles (envolventes) de aquellas lecturas similares se encuentran próximas entre sí de una manera organizada. En términos de probabilidad podría decirse que existe cierta probabilidad de observar las lecturas próximas a la representación de la observación actual dentro de la estructura, lo que la hace factible de ser utilizada para nuestros propósitos.

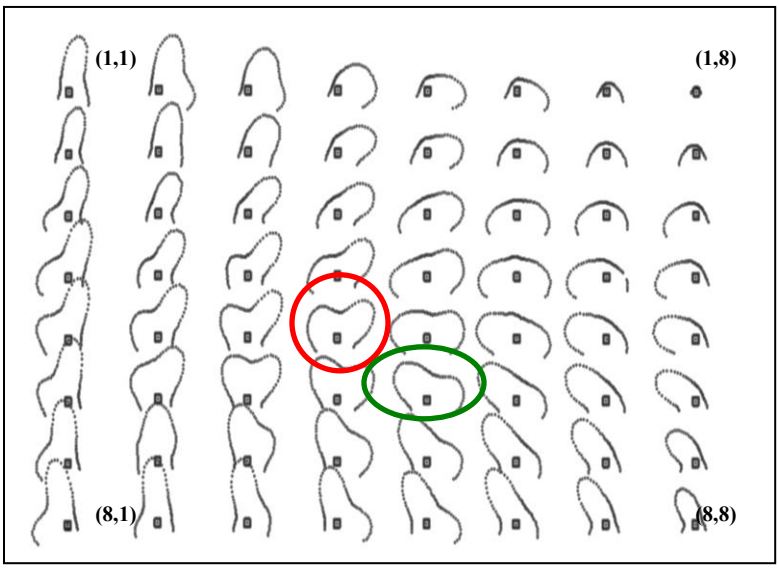


Figura 7.7. Resultado del entrenamiento de la Red de Kohonen 8 × 8 sin preprocesamiento de los datos y con relación de vecindad. 64 vectores tipo. Se muestra la representación de los vectores tipo de la Fig. 7.5 con y sin oclusiones.

De lo anterior se concluye que el mejor método neuronal de cuantificación evaluado fue la Red de Kohonen Bidimensional. En esta forma, cada lectura real con los sensores del robot puede ser transformada en un índice bidimensional dentro del arreglo de Kohonen [s_1 .. s_8, s_1 .. s_8] al alimentarla al mapa autoorganizado.

7.5.4 Modelo de Observación Tolerante a Oclusiones

Para evitar la necesidad de entrenar la red de Kohonen con muestras con ruido causado por oclusiones, dadas las características propias de tales mapas autoorganizados, se encontró que tales mapas proporcionan símbolos cercanos a la observación sin ruido para aquellas observaciones con oclusiones. Aprovechando esta propiedad se definió una probabilidad de observación con una relación de vecindad de tipo Gaussiano

$$G(s_k, s_i) = e^{\frac{-\left|g(s_i^s) - g(s_k)\right|^2}{2\sigma_{tol}^2}} \tag{7.17}$$

donde $g(s_i^s)$ y $g(s_k)$ son los índices 2D (i_i^s, j_i^s) y (i_k, j_k) respectivamente en el mapa aotoorganizado de Kohonen SOM (*Self Organized Map*), correspondientes a los símbolos s_i^s (símbolo correspondiente a la observación supuesta para el estado *i*) y s_k (símbolo del SOM correspondiente a la observación actual), mientras σ_{tol} (o la 'tolerancia') puede ser fijado en forma experimental, dependiendo de la factibilidad de ocurrencia de presentarse oclusiones y errores en los sensores (tal es el caso de ambientes con muchas personas).

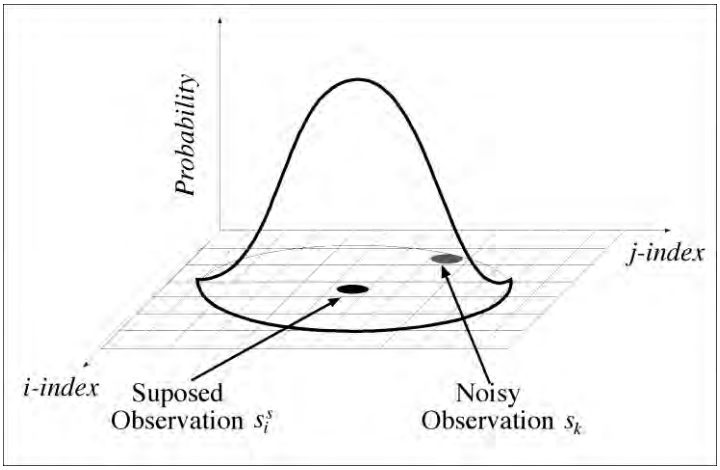


Figura 7.8. Gaussiana de probabilidad de observación de símbolos con ruido para una observación supuesta dentro del SOM de Kohonen utilizado para la Cuantificación vectorial.

El parámetro σ_{tol} da al Modelo de Observación representado por la matriz **B** la propiedad de "tolerar" algunas variaciones en los símbolos observados, producidos principalmente por oclusiones y reflexiones de la señal emitida por el sensor láser o sonar (Fig. 7.8).

La ecuación anterior evalúa la distancia entre la observación real (supuesta para cada estado) y el resto de los símbolos en el mapa. De esta manera, ninguna probabilidad de observación por símbolo es igual a cero y se aprovechan las características del mapa autoorganizado.

Al igual que para el caso de las transiciones, una normalización es necesaria

$$p(s_k \mid x_i) = \frac{G\left(s_k, V(S(x_i))\right)}{\sum_{n=1}^{m} G\left(s_n, V(S(x_i))\right)} \tag{7.18}$$

7.5.5 Obtención de la Matriz de Observación de Símbolos del HMM

Mediante el uso de la Ec. 7.18 es posible calcular para cada estado del modelo, la probabilidad de observar cada símbolo s_k, con base en un proceso de simulación previo, sin embargo en la práctica sería perfectamente posible que el robot entrenara el mapa de Kohonen con lecturas reales, a medida que se desplaza por el entorno y hasta que el operador determine que se han tomado un conjunto suficiente de lecturas representativo del ambiente de operación del robot.

7.6 Optimización del Método OVL para una Rápida Ejecución

Básicamente, la Localización de Viterbi con Odometría (Odometry-based Viterbi Localization OVL) se asemeja a un filtro de partículas donde la ubicación de las partículas (estados) permanece fija durante toda la ejecución del algoritmo. Gracias a esta propiedad, es posible calcular una observación supuesta ya sea por simulación o toma directa de muestras para cada estado agilizando enormemente el proceso de operación en tiempo real al conocer de antemano la observación supuesta para cada estado. El mayor

inconveniente del método OVL radica en la propia naturaleza del Algoritmo de Viterbi, donde en cada iteración todas las transiciones desde cualquier estado *i* a cualquier estado *j* son calculadas, es decir el algoritmo es $O(N^2)$. De esta forma, muchas transiciones cuya probabilidad será prácticamente cero hacen que el algoritmo se torne lento al requerir de tiempo de proceso.

Para subsanar lo anterior se determinó un *umbral de disparo*, esto es, dado que el Método de Viterbi es un algoritmo recursivo que en cada iteración actualiza la probabilidad de que el robot se encuentre en cada estado, almacenada en el vector

$$\Phi_t = \begin{bmatrix} p(x_1 \mid u_{0:t}) & p(x_2 \mid u_{0:t}) & \ldots & p(x_n \mid u_{0:t}) \end{bmatrix} \tag{7.19}$$

por la Ec. (7.8) se tiene que

$$\Phi_{t+1}[j] = \eta\, b_j(s_t) \sum_{i=1}^{n} \left(a_{ij}(u_t)\, \Phi_t[i] \right) \tag{7.20}$$

donde η es un normalizador correspondiente a

$$\eta = \sum_{j=1}^{n} \left[b_j(s_t) \sum_{i=1}^{n} \left(a_{ij}(u_t)\, \Phi_t[i] \right) \right] \tag{7.21}$$

Entonces si $\Phi_t[i] \approx 0$ implica que su contribución en la Ec. (7.20) será prácticamente cero al multiplicarse por un valor a_{ij} menor que uno. Por lo tanto, si sólo se calculan la operaciones $a_{ij}(u_t)\,\Phi_t[i]$ para $\Phi_t[i] \geq \varepsilon$, entonces se tendrá un ahorro de una operación en la Ec. (7.20) y *n* operaciones en la Ec. (7.21) por cada $\Phi_t[i] < \varepsilon$.

En consecuencia, si la operación $a_{ij}(u_t)\,\Phi_t[i]$ no será efectuada para $\Phi_t[i] < \varepsilon$ entonces tampoco será necesario calcular el preciso valor de a_{ij} con la Ec. (7.16) incrementando el ahorro de operaciones en el cálculo de la matriz **A**.

En las pruebas al método presentadas en el Apéndice A a manera de comparación, el cálculo de una actualización completa por iteración calculando todas las transiciones fue de 4s, comparado con sólo 0.02s por iteración con $e = 1/n$.

A pesar de que el dejar de calcular a contribución de muchas transiciones poco probables pudiese parecer que afecte demasiado el valor final de $F_t[i]$, el aumento en el error en la localización comparándolo contra el cálculo completo de transiciones es de tan solo 2mm, por lo que resulta despreciable.

Finalmente, la mayor ventaja que supuso el integrar el umbral de disparo ε es sin duda que resulta un inhibidor de transiciones. De esta manera, si la ubicación inicial del robot es desconocida, al fijar $F_t[i] = e''\ i$ automáticamente se forzará al método a calcular todas las transiciones desde todos los estados (Localización Global). A medida que el método vaya convergiendo, sucederá que $F_t[i] < e$ para algunos estados, descartándose el cómputo de su contribución automáticamente (Rastreo de Posición). Finalmente, si el robot es abruptamente desplazado a otra ubicación, rápidamente la probabilidad de observación para la ubicación supuesta decaerá hasta que suceda nuevamente que $F_t[i] = e''\ i$ forzando nuevamente al computo del total de transiciones (Secuestro del Robot). En resumen, el umbral de disparo es una medida que por si misma permite al método de Localización de Viterbi con Odometría (OVL) resolver el problema de la Localización del Robot en sus tres aspectos de una manera práctica y muy sencilla de implementar.

7.7 Ultimas Adiciones (Posteriores a la Publicación del Método)

La presente sección muestra las más recientes adiciones al método OVL, las cuales fueron realizadas posteriormente a la publicación del artículo referido en el Apéndice A.

7.7.1 Aplicación de Conceptos de Lógica Borrosa

La operación del método de Localización de Viterbi con Odometría descrita en [Llarena 11b] (Ver Apéndice A, Sección 4.5) puede ser vista como

$$\mathbf{F}_t = \frac{\mathbf{F}_{t-1}\mathbf{A} \ddot{A} \mathbf{B}\eta_t^T}{\mathbf{F}_{t-1}\mathbf{A}\mathbf{B}\eta_t^T} \tag{7.22}$$

donde $\mathbf{F}_{t-1}$ es el vector de hipótesis de probabilidad de la iteración anterior, **A** y **B** son las matrices de transición y observación del HMM respectivamente, el símbolo $\mathbf{A}$ denota el *producto dimensional* dado por

$$\boldsymbol{u} \ddot{A} \boldsymbol{v} = (u_1 v_1, u_2 v_2, ..., u_n v_n) \tag{7.23}$$

y el vector

$$\eta_t = \begin{bmatrix} z_t \circ s_1 & z_t \circ s_2 & \dots & z_t \circ s_m \end{bmatrix}^T \tag{7.24}$$

se propone como un vector unitario con la exacta correspondencia de la observación actual z_t respecto de cada posible símbolo del HMM $[s_1 .. s_m] \in Z$. Por ejemplo, si $z_t = s_1$:

$$\eta_t = \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix}^T$$

Al analizar el producto $\mathbf{B}v^T$

$$\begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{bmatrix} \begin{bmatrix} (z_k = s_0) \\ (z_k = s_1) \\ \dots \\ (z_k = s_{m-1}) \end{bmatrix} = \begin{bmatrix} b_1(z_t) \\ b_2(z_t) \\ \dots \\ b_n(z_t) \end{bmatrix} \tag{7.25}$$

es claro que el vector η_t actúa como un selector de probabilidad de observación del símbolo actual z_t dentro del modelo de observación representado por **B** y denotada por $b_j(z_t)$ en el Algoritmo de Viterbi.

Es posible afirmar que el vector n_t es de tipo binario (es decir, sus posibles valores corresponden el espacio vectorial de dimensión *m* de todos los vectores ortonormales y de módulo uno), ya que únicamente tiene un valor distinto de cero e igual a uno para aquel símbolo que corresponda con la observación actual, o dicho en términos prácticos, resulte *más similar a la observación tipo* representada en el mapa autoorganizado (SOM), sin embargo se puede dar el caso de que en términos de distancia Euclidiana, la observación actual resulte muy similar a más de un vector tipo o símbolo representado en el SOM. En este caso (el caso más general) podemos hablar de un grado de pertenencia o valor de similitud con respecto de cada uno de los vectores tipo representados en el SOM, es decir, bajo un concepto de *lógica borrosa* en lugar de la pertenencia exclusivamente a una sola clase o *lógica crisp*.

Bajo este nuevo enfoque, el vector n_t contendría ahora el grado de pertenencia o similitud de la observación actual z_t, respecto a cada vector tipo almacenado en el SOM, en la escala [0 ..1] tal que

$$n_t = \begin{bmatrix} m_1 & m_2 & \dots & m_m \end{bmatrix} \tag{7.26}$$

donde $0 \le \mu_k \le 1; \quad 0 \le k \le m$ y como función de pertenencia se tenga

$$m_k = e^{\frac{-|z_t - o_k|^2}{2s_k^2}} \tag{7.27}$$

donde o_k representa el k-ésimo vector del mapa autoorganizado SOM de Kohonen propuesto para realizar la reducción de la dimensionalidad del vector de observaciones.

Al tratarse de valores de pertenencia borrosos, no es requisito que la suma de los elementos en n_t sea igual a uno, con lo que se evita tener que normalizar. Por otro lado, dicha normalización no resulta necesaria ya que la Ec. (7.22) garantiza que la suma de los $F_{t+1}[i]$ sea igual a la unidad.

Matemáticamente resultará igualmente valido que el vector n_t sea de tipo borroso en la expresión (7.22) ya que se trata del mismo producto matricial sin restarle generalidad al método. Dado que el mapa autoorganizado de Kohonen calcula la distancia Euclidiana para estimar el vector tipo más similar a la observación actual, es perfectamente posible modificar la salida de la red de Kohonen para que, en lugar de proporcionar una salida alta (igual a uno o salida *MAX*) exclusivamente para el vector más similar y el resto de salidas igual a cero, pueda proporcionar directamente a la salida de la red el valor de la Ec. (7.27) obteniendo directamente los valores de pertenencia a cada una de las *m* clases de símbolos en el SOM.

La figura 7.9 muestra el beneficio que se obtiene en la hipótesis de localización con la variante borrosa del método o *fuzzy-OVL*. Como era de esperarse, el uso de un vector n_t de tipo borroso implica calcular necesariamente el producto $\mathbf{B}n_t$. En un experimento con los mismos datos utilizados en los experimentos con datos reales descrito en el Apéndice A arrojó que el cálculo de una iteración en fuzzy-OVL tardó 0.7s comparados con 0.02 para OVL con vector n_t crisp. No obstante el aumento de tiempo es considerable, el beneficio es claramente observable. Por otro lado, en procesadores con más de un núcleo o GPU (*Graphic Processing Unit*) el cálculo de $\mathbf{B}n_t$ podría hacerse en paralelo al cálculo de la matriz A. Disminuyendo considerablemente el tiempo de cálculo.

Figura 7.9. (Izq.) Odometría pura. (Centro) OVL. (Der.) Fuzzy-OVL

7.7.2 Representación Matemática del Cálculo de la Matriz A(u_t)

Si se tiene un vector con la estimación de desplazamiento por odometría en el sistema de referencia centrado en el robot y cuyo frente vea en dirección del eje x

$$\boldsymbol{u}_t = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \theta \\ 1 \end{bmatrix} \tag{7.28}$$

entonces es posible representar el desplazamiento relativo a la ubicación de cada estado (x_i, y_i, θ_i) en coordenadas mundiales como

$$\boldsymbol{x}_i' = \boldsymbol{R}_i \boldsymbol{u}_t = \begin{bmatrix} \cos\theta_i & -\operatorname{sen}\theta_i & 0 & x_i \\ \operatorname{sen}\theta_i & \cos\theta_i & 0 & y_i \\ 0 & 0 & 1 & \theta_i \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \theta \\ 1 \end{bmatrix} = \begin{bmatrix} x_i' \\ y_i' \\ \theta_i' \end{bmatrix} \tag{7.29}$$

De forma análoga se tiene que

$$\boldsymbol{x}_i' - \boldsymbol{x}_j = \boldsymbol{R}_{ij} \boldsymbol{u}_t = \begin{bmatrix} \cos\theta_i & -\operatorname{sen}\theta_i & 0 & (x_i - x_j) \\ \operatorname{sen}\theta_i & \cos\theta_i & 0 & (y_i - y_j) \\ 0 & 0 & 1 & (\theta_i - \theta_j) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \theta \\ 1 \end{bmatrix} = \begin{bmatrix} dx_{ij} \\ dy_{ij} \\ d\theta_{ij} \end{bmatrix} \tag{7.30}$$

Entonces, si se define la matriz

$$\Delta_{ij} = \begin{bmatrix} \frac{\cos\theta_i}{s_d\sqrt{2}} & \frac{-\operatorname{sen}\theta_i}{s_d\sqrt{2}} & 0 & \frac{(x_i - x_j)}{s_d\sqrt{2}} \\ \frac{\operatorname{sen}\theta_i}{s_d\sqrt{2}} & \frac{\cos\theta_i}{s_d\sqrt{2}} & 0 & \frac{(y_i - y_j)}{s_d\sqrt{2}} \\ 0 & 0 & \frac{1}{s_\theta\sqrt{2}} & \frac{(\theta_i - \theta_j)}{s_\theta\sqrt{2}} \end{bmatrix} \tag{7.31}$$

y las operaciones

$$v \times v = v^2 = \left(v_1^2 + v_2^2 + \ldots + v_n^2 \right) \tag{7.32}$$

$$e^v = \begin{pmatrix} e^{v_1} & e^{v_2} & \ldots & e^{v_n} \end{pmatrix} \tag{7.33}$$

y

$$e^A = \begin{bmatrix} e^{a_{11}} & e^{a_{12}} & \ldots & e^{a_{1n}} \\ e^{a_{21}} & e^{a_{22}} & \ldots & e^{a_{2n}} \\ & & \ldots & \\ e^{a_{m1}} & e^{a_{m2}} & \ldots & e^{a_{mn}} \end{bmatrix} \tag{7.34}$$

donde

$$v = \begin{pmatrix} v_1 & v_2 & \ldots & v_n \end{pmatrix} \quad A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ & & \ldots & \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}$$

Entonces la Ec. (7.15) puede ser representada como

$$D(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\left(D_{ij}\boldsymbol{u}_t\right)^2} \tag{7.35}$$

De esta manera, al construir la matriz $\mathbf{A}(\boldsymbol{u}_t)$ se tiene que

$$A(\boldsymbol{u}_t) = h\left(\begin{bmatrix} e^{-(D_{11}\boldsymbol{u}_t)^2} & e^{-(D_{12}\boldsymbol{u}_t)^2} & \ldots & e^{-(D_{1n}\boldsymbol{u}_t)^2} \\ e^{-(D_{21}\boldsymbol{u}_t)^2} & e^{-(D_{12}\boldsymbol{u}_t)^2} & \ldots & e^{-(D_{1n}\boldsymbol{u}_t)^2} \\ & & \ldots & \\ e^{-(D_{n1}\boldsymbol{u}_t)^2} & e^{-(D_{n2}\boldsymbol{u}_t)^2} & \ldots & e^{-(D_{nn}\boldsymbol{u}_t)^2} \end{bmatrix} \right) \tag{7.36}$$

donde h es un operador de normalización definido como

$$\eta A = \begin{bmatrix} a_{11}/\sum_{i=1}^{n} a_{1i} & a_{12}/\sum_{i=1}^{n} a_{1i} & \dots & a_{1n}/\sum_{i=1}^{n} a_{1i} \\ a_{21}/\sum_{i=1}^{n} a_{2i} & a_{22}/\sum_{i=1}^{n} a_{2i} & \dots & a_{2n}/\sum_{i=1}^{n} a_{2i} \\ \dots & & & \\ a_{m1}/\sum_{i=1}^{n} a_{mi} & a_{m2}/\sum_{i=1}^{n} a_{mi} & \dots & a_{mn}/\sum_{i=1}^{n} a_{mi} \end{bmatrix} = \begin{bmatrix} 1/\sum_{i=1}^{n} a_{1i} \\ 1/\sum_{i=1}^{n} a_{2i} \\ \dots \\ 1/\sum_{i=1}^{n} a_{mi} \end{bmatrix} \otimes \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & & & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Haciendo extensible la definición de *contracción vectorial dimensional* a matrices se tiene que

$$\boldsymbol{M}^{\oplus_r} = \begin{bmatrix} m_{1,1} + m_{1,2} + \dots + m_{1,n} \\ m_{2,1} + m_{2,2} + \dots + m_{2,n} \\ \dots \\ m_{n,1} + m_{n,2} + \dots + m_{n,n} \end{bmatrix} \quad \boldsymbol{M}^{\oplus_c} = \begin{bmatrix} m_{1,1} + m_{2,1} + \dots + m_{n,1} \\ m_{1,2} + m_{2,2} + \dots + m_{n,2} \\ \dots \\ m_{1,n} + m_{2,n} + \dots + m_{n,n} \end{bmatrix}^T$$

Por otro lado, dado que un vector no posee inversa, si se define la inversa de un vector $\boldsymbol{v}^{-1}$ en términos del producto dimensional como

$$\boldsymbol{v}^{-1} = \begin{pmatrix} 1/v_1 & 1/v_1 & \dots & 1/v_n \end{pmatrix} \tag{7.37}$$

lo que implica que

$$\boldsymbol{v} \ddot{A} \boldsymbol{v}^{-1} = \begin{pmatrix} 1 & 1 & \dots & 1 \end{pmatrix} = \overline{1} \tag{7.38}$$

Por lo tanto es posible representar el operador de normalización η como

$$\eta A = \left[A^{\oplus_r} \right]^{-1} \otimes A \tag{7.37}$$

Si se define el operador

$$A \ddot{A} A = A^{2\ddot{A}} = \begin{bmatrix} a_{11} \times a_{11} & a_{12} \times a_{12} & \dots & a_{1n} \times a_{1n} \\ a_{21} \times a_{21} & a_{22} \times a_{22} & \dots & a_{2n} \times a_{2n} \\ & & \dots & \\ a_{m1} \times a_{m1} & a_{m2} \times a_{m2} & \dots & a_{mn} \times a_{mn} \end{bmatrix} = \begin{bmatrix} a_{11}^2 & a_{12}^2 & \dots & a_{1n}^2 \\ a_{21}^2 & a_{22}^2 & \dots & a_{2n}^2 \\ & & \dots & \\ a_{m1}^2 & a_{m2}^2 & \dots & a_{mn}^2 \end{bmatrix} \tag{7.38}$$

donde los elementos a_{ij} de la matriz **A** son vectores y el resultado $\boldsymbol{A} \ddot{A} \boldsymbol{A} = \boldsymbol{A}^{2\ddot{A}}$ es una matriz escalar de dimensión $m \times n$, entonces (abusando de la notación matricial)

$$A(\boldsymbol{u}_t) = he^{-\begin{bmatrix} D_{11}\boldsymbol{u}_t & D_{12}\boldsymbol{u}_t & \dots & D_{1n}\boldsymbol{u}_t \\ D_{21}\boldsymbol{u}_t & D_{22}\boldsymbol{u}_t & \dots & D_{2n}\boldsymbol{u}_t \\ & & \dots & \\ D_{n1}\boldsymbol{u}_t & D_{n2}\boldsymbol{u}_t & \dots & D_{nn}\boldsymbol{u}_t \end{bmatrix}^{2\otimes}} = he^{-\left(\begin{bmatrix} D_{11} & D_{12} & \dots & D_{1n} \\ D_{21} & D_{22} & \dots & D_{2n} \\ & & \dots & \\ D_{n1} & D_{n2} & \dots & D_{nn} \end{bmatrix} \boldsymbol{u}_t \right)^{2\otimes}} = he^{-(D\boldsymbol{u}_t)^{2\otimes}} \tag{7.39}$$

Finalmente, es posible reescribir la Ec. (7.22) como

$$\mathrm{F}_t = \frac{\mathrm{F}_{t-1} he^{-(D\boldsymbol{u}_t)^{2\otimes}} \otimes \mathbf{B}\boldsymbol{n}_t^{\mathrm{T}}}{\mathrm{F}_{t-1} he^{-(D\boldsymbol{u}_t)^{2\otimes}} \mathbf{B}\boldsymbol{n}_t^{\mathrm{T}}} \tag{7.40}$$

haciendo explícitos el vector de observación actual $\boldsymbol{n}_t^{\mathrm{T}}$ y el vector de odometría $\boldsymbol{u}_t$ en el cálculo de la hipótesis de localización F_t.

Nota Importante: el producto dimensional $\otimes$ no debe ser confundido con el producto tensorial de Kronecker, operación la cual se define en términos de dos vectores $\boldsymbol{u}$ ($n \times 1$) y $\boldsymbol{v}$ ($1 \times m$) como

$$\begin{bmatrix} u_1 \\ u_2 \\ \dots \\ u_n \end{bmatrix} \ddot{A} \begin{bmatrix} v_1 & v_2 & \dots & v_m \end{bmatrix} = \begin{bmatrix} u_1 v_1 & u_1 v_2 & \dots & u_1 v_m \\ u_2 v_1 & u_2 v_2 & \dots & u_2 v_m \\ & & \dots & \\ u_n v_1 & u_n v_2 & \dots & u_n v_m \end{bmatrix} \tag{7.41}$$

7.8 Discusión del Método de Localización de Viterbi con Odometría OVL

El primer dato notable si se analizan las pruebas y resultados al método expuestos en el Apéndice A, es que OVL es capaz de calcular una posición continua del robot a partir de estados discretos. Dado que tanto el número como la ubicación espacial de los estados es determinada de antemano, OVL permite adaptarse a los recursos disponibles de memoria y capacidad de procesamiento.

Por otro lado, una de sus principales fortalezas se centra en su capacidad inherente de resolver los tres subproblemas de la localización sin adaptaciones adicionales: la Localización Global, El Rastreo de la Posición y el Secuestro del Robot. No obstante en su forma más general el Algoritmo de Viterbi implica el cálculo de transiciones desde cada estado y hacia todos los estados del modelo, dado que sólo algunas transiciones tienen relevancia numérica, el mismo *umbral de disparo* utilizado para activar la localización en sus tres subproblemas permite ignorar el cálculo de transiciones insignificantes para efectos de la localización.

Como en la sección anterior se mostró, al utilizar una reducción de la dimensionalidad de las observaciones para agilizar el método puede suceder que más de un vector tipo en el SOM sea muy similar en términos de distancia Euclidiana lo que significaría que más de un símbolo se observa. Al considerar observaciones de tipo difuso, en donde no sólo el símbolo más similar a la observación actual determina la probabilidad de observación, sino todos los símbolos del modelo contribuyen de alguna forma, los cambios en las transiciones del modelos se hacen más suaves y mucho más similares a la trayectoria real (ver Fig. 7.9). A pesar de que el método se ve afectado en su rendimiento, la calidad en la estimación de la ubicación amerita su uso.

De manera similar, a diferencia del Método de Localización de Viterbi en [Savage 05], el estado inicial del robot se estima con la Ec. (7.16) haciendo $\boldsymbol{u}_t$ igual a 0 y $\boldsymbol{x_0}$ igual a la posición inicial del robot. El efecto de dicha ecuación en el cálculo de la probabilidad inicial π_i y en las transiciones de $\boldsymbol{x_i}$ a $\boldsymbol{x_j}$ es servir de medio de descomposición del valor

continuo de posición $\boldsymbol{x_t}$ como una suma ponderada (valor esperado o *esperanza*) de la posición $\boldsymbol{x_i}$ de cada estado por su probabilidad calculada con la Ec. (7.16) en función de su distancia Euclidiana. Dado que el espacio vectorial *X* definido por las posiciones de los estados (x_i, y_i, θ_i) no es un espacio con vectores ortonormales, no existe una solución única para descomponer $\boldsymbol{x_0}$ en función de vectores en el espacio de *X*. En este sentido la Ec. (7.16) hace una buena aproximación de en función únicamente de aquellos vectores dentro del radio de influencia de *D* en la Ec. (7.15).

El principal problema relacionado con el método se centra en que es recomendable que la estimación de la posición (iteración) se realice cuando el desplazamiento del robot por lo menos iguala la distancia promedio entre los nodos (estados) del modelo, ya que para desplazamientos menores OVL encontrará mayor probabilidad de permanecer en el mismo estado que realizar una transición a otro. Bajo este escenario, si el robot se encuentra en pasillos largos o zonas con observaciones muy similares, al igual que el resto de los métodos de localización probabilísticos, el error de localización irá en aumento hasta que perciba una observación que resuelva la ambigüedad.

Otro problema menor se relaciona con la forma en que OVL determina cuándo realizar una Localización Global. Lo anterior sucede en pocas iteraciones luego de que lo observado no se asemeja a aquello que se supone el robot debe observar en una ubicación dada. En muchas ocasiones la discrepancia se debe a oclusiones provocadas principalmente por personas que se acercan deliberadamente al robot inclusive bloqueando los sensores. En este sentido en el futuro sería posible tratar de detectar cuando se trata de personas que bloquean los sensores para descartar las lecturas del sensor en esa dirección.

Capítulo 8

Conclusiones

8.1 Conclusiones

Comparado con los métodos probabilísticos más populares [Joseph 04, Fox 99] la Localización de Viterbi basada en Odometría, Redes Neuronales Artificiales y sensores Láser presenta serias ventajas: a diferencia del método de localización de Monte Carlo no requiere de una precisa estimación de las observaciones durante su operación, tampoco requiere de la inversión de matrices como el filtro de Kalman. A pesar de su naturaleza discreta proporciona estimaciones continuas. También permite manejar diversas hipótesis de manera simultánea tal y como lo hacen los métodos más novedosos del *estado del arte*. Debido al uso de Mapas de Kohonen Autoorganizados, OVL es capaz de manejar observaciones *n-dimensionales* (que puede incluir información visual o datos provenientes de diversos sensores) debido a que el trabajo de reducción de la dimensionalidad es realizado por Redes Neuronales Artificiales, éstas son perfectamente factibles de ser implementadas en hardware, liberando recursos de memoria y proceso.

Comparado con la Localización de Viterbi (VL) en [Savage 05], la versión optimizada de OVL implica importantes mejoras al método incluyendo:

1. Incorporación de un Modelo de Movimiento dependiente de la Odometría.
2. Capacidad de manejo de Localización Global, Rastreo de Posición y Localización Global al incluir un *umbral de disparo*.
3. Mayor número de estados (varios miles en OVL contra una docena en VL).
4. Estimaciones de posición continuas.

5. Sólo las transiciones más significativas son evaluadas.
6. Reducción de la dimensionalidad mediante Mapas Autoorganizados tolerantes a oclusiones y errores en los sensores.
7. Eliminación de la tendencia de la probabilidad a cero con el paso del tiempo mediante una normalización en cada iteración.

En conclusión, OVL es un Algoritmo Probabilístico No Degenerativo (NDPA) que es capaz de resolver eficazmente el problema de la localización robótica de manera *completa*, es decir, en sus tres subproblemas, basado tanto en odometría como en las observaciones. Ahorra memoria y tiempo de proceso debido a su naturaleza discreta pero provee estimaciones continuas. Finalmente, puede ejecutarse en computadoras con uno o más núcleos en tiempo real.

Como contribución al estado del arte el presente trabajo propone una representación matemática para la operación del Algoritmo de Viterbi en la resolución de $p(O|I)$ en un HMM, según:

$$\mathsf{F}_t = \mathsf{F}_{t-1}\mathbf{A} \ddot{\mathrm{A}} \mathbf{B}n_t^{\mathrm{T}} \tag{7.42}$$

donde

$$\mathsf{F}_t(i) = p(x_i, z_{0:t}) \tag{7.43}$$

y

$$p(O|I) = \mathsf{F}_{t-1}\mathbf{AB}n_t^{\mathrm{T}} \tag{7.44}$$

Si el algoritmo de Viterbi es modificado de acuerdo a la regla de Bayes para estimar el estado a partir de las observaciones, entonces (7.42) se transforma en

$$\mathsf{F}_t = \frac{\mathsf{F}_{t-1}\mathbf{A} \ddot{\mathrm{A}} \mathbf{B}n_t^{\mathrm{T}}}{\mathsf{F}_{t-1}\mathbf{AB}n_t^{\mathrm{T}}} \tag{7.45}$$

donde

$$\mathsf{F}_t(i) = p(x_i \mid z_{0:t}) \tag{7.46}$$

Donde las matrices de Transición **A** y Observación **B** permanecen constantes durante la operación del Algoritmo (HMM Clásico).

Es posible incorporar una estimación de transición en el cálculo dinámico de la matriz $\mathbf{A}_t$ según

$$\mathsf{F}_t = \frac{\mathsf{F}_{t-1}\mathbf{A}_t \ddot{\mathrm{A}} \mathbf{B}n_t^{\mathrm{T}}}{\mathsf{F}_{t-1}\mathbf{A}_t\mathbf{B}n_t^{\mathrm{T}}} \tag{7.47}$$

donde

$$\mathbf{A}_t = h e^{-(\Delta u_t)^{2\otimes}} \tag{7.48}$$

El planteamiento anterior facilita el estudio de los modelos que serían denominados Modelos Dinámicos de Markov de Variable Oculta o *Dynamic Hidden Markov Models* (DHMM) donde las transiciones dependen de variables de entrada como $\boldsymbol{u_t}$ (la odometría) para el caso de Localización Robótica. Esta propiedad pudiese aplicarse a problemas similares en muy diversos campos de aplicación.

En el caso más general, es posible que las probabilidades de observación representadas en **B** puedan sufrir modificaciones o refinamiento durante la ejecución del algoritmo (no analizadas en el presente trabajo), con lo que DHMM sería

$$\mathsf{F}_t = \frac{\mathsf{F}_{t-1}\mathbf{A}_t \ddot{\mathrm{A}} \mathbf{B}_t n_t^{\mathrm{T}}}{\mathsf{F}_{t-1}\mathbf{A}_t\mathbf{B}_t n_t^{\mathrm{T}}} \tag{7.49}$$

8.2 Trabajos Futuros

Actualmente se está trabajando para incorporar la versión optimizada de OVL a robots humanoides futbolistas como los descritos en el capítulo 6, los cuales utilizan una cámara web como único medio de observación. Se espera escribir un segundo artículo con los resultados obtenidos con dicho humanoide junto con las adaptaciones descritas en la sección 7.6.

Por otro lado el Método de Localización de Viterbi basado en Odometría por su propia naturaleza (Ec. 7.47) es susceptible de ser implementado en arquitecturas paralelas e incorporar Aprendizaje de Máquina para estimar las matrices **A** y **B** del HMM a medida que el robot explora su medio ambiente. En un futuro próximo se piensa experimentar en este sentido para evitar la necesidad de contar con un mapa métrico del entorno para simular las observaciones.

Finalmente, se pretende en un futuro desarrollar una implementación *on-chip* del algoritmo para poder ser incorporado como un dispositivo localizador a bordo de robots de servicio como el robot ARTURito o los robot humanoides futbolistas por su limitada capacidad de procesamiento.

Bibliografía

[Artac 02] Artac, M., Jogan, M., Leonardis, A.: *Mobile Robot Localization Using an Incremental Eigenspace Model*. Proceedings of the IEEE International Conference on Robotics and Automation, 1 (2002) 1025-1030

[Bais 06] A. Bais and R. Sablatnig, "Landmark based global self-localization of mobile soccer robots," in 7th Asian Conference on Computer Vision, ser. LNCS, vol. 3852, 2006, p. 842 – 851.

[Bay 06] Herbet Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded up robust features. *In European Conference on Computer Vision*, 2006.

[Bruce 00] J. Bruce, Tucker Balch, and Maria Manuela Veloso.
Fast and inexpensive color image segmentation for interactive robots.
Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '00), October, 2000, p. 2061 - 2066.

[Bennewitz 06] Bennewitz M., Stachniss C., Burgard W., and Behnke J. Sven.
Metric Localization with SIFT Features using a Single Camera.
EUROS 2006, European Robotics Symposium
March 16-18, 2006, Palermo / Italy

[Bentley 75] J. L. Bentley. Multidimensional bynary search trees used for associative searching. *Commun. ACM*. 18(9):509-517, 1975.

[Brío 02] Bonifacio Martín del Brío, Alfredo Sanz Molina, *Redes Neuronales y Sistemas Borrosos,* 2ª Edición, Alfaomega 2002 p. 129-132

[Carpenter 97] J. Carpenter, P. Clifford, and P. Fernhead, "An improved particle filter for non-linear problems", tech. Rep., Department of Statistics, University of Oxford, 1997.

[Choi 07] Choi, W. S. & Oh, S. Y. (2007). *Range Sensor-based Robot Localization Using Neural Network*, International Conference on Control, Automation and Systems, p. 230-234, 17-20.

[Crowley 98] Crowley, J. L., Wallner, F., Schiele, B.: *Position Estimation Using Principal Components of Range Data*. Proceedings of the IEEE International Conference on Robotics and Automation, 4 (1998) 3121-3128

[Dellaert 99] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte Carlo Localization for Mobile Robots," IEEE International Conference on Robotics and Automation (ICRA99), May, 1999.

[Djekoune 01] Djekoune, O., Achour, K.: *Vision-guided Mobile Robot Navigation Using Neural Network*. Proceedings of the 2nd International Symposium on Image and Signal Processing and Analysis, (2001) 355-361

[Du 99] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal Voronoi Tessellations: Applications and Algorithms. 41(4):637-676, December 1999.

[Dudek 00] Dudek, G., Jenkin, M.. *Computational Principles of Mobile Robotics*. Cambridge , University Press, 2000

[Elinas 05] Pantelis Elinas and J. Little, "*sigmaMCL: Monte-Carlo Localization for Mobile Robots with Stereo Vision*", Robotics: Science and Systems, Boston, MA, USA, June 2005.

[Elfes 89] A. Elfes. "Using occupancy grids for mobile robot perception and navigation", Computer, 22(6):46–57, 1989.

[Folkesson 05] Folkesson, J. Jensfelt, P. Christensen, H.I, *Vision SLAM in the Measurement Subspace*, Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference, 18-22 April 2005.

[Forney 73] Forney, G. D. *The Viterbi algorithm*. Proceedings of the IEEE 61(3):268–278, March 1973.

[Fox 99] D. Fox, W. Burgard, S. Thrun, "Markov localization for reliable robot navigation and people detection", In Modeling and Planning for Sensor-Based Intelligent Robot Systems. Springer Verlag, Berlin, 1999.

[Gross 03] H. Gross, A. Koenig, C. Schroeter, and H. Boehme, "Omnivision-based probalistic self-localization for a mobile shopping assistant continued," in Proceedings of IEEE/RSJ International Conference on

Intelligent Robots and Systems(IROS‟03), LasVegas, USA, 2003, p. 1505–1511.

[Haykin 96] S. Haykin. *Neural Networks: A Comprehensive Foundation*. New York, Macmillan College Publishing Company, 1996.

[Hu 99] Hu, H. S., Gu, D. B.: *Landmark-Based Navigation of Mobile Robot in Manufacturing*. Proceedings of the IEEE International Conference on Emerging Technologies.

[Indyk 04] Indyk, P. *Nearest neighbors in high-dimensional spaces*. Handbook of Discrete and Computational Geometry, chapter 39. Editors: Jacob E. Goodman and Joseph O'Rourke, CRC Press, 2nd edition, 2004.

[Jolliffe 02] Jollife, I.T. *Principal Component Analysis, 2nd Edition,* Springer Verlag. ISBN-13: 978-0387954424, 2002.

[Joseph 04] Joseph J. and LaViola Jr. *A comparison of unscented and extended kalman filtering for estimating quaternion motion*. Proceedings of the 2004 American Control Conference, IEEE Press, 2190-2195, June 2004.

[Janet 95] Janet, J. A., Gutierrez-Osuna, R., Chase, T. A., White, M., Luo, R. C.: *Global Self-localization for Autonomous Mobile Robots Using Self-organizing Kohonen Neural Networks*. Proceedings of the IEEE International Conference on Intelligent Robotics and Systems, 3 (1995) 504-509

[Karlsson 05] Karlsson, N.; Di Bernardo, E.;Ostrowski, J;Goncalves, L.;Pirjanian, P.;Munich, M. (2005). "*The vSLAM Algorithm for Robust Localization and Mapping*". Int. Conf. on Robotics and Automation (ICRA).

[Kalman 60] Kalman, R.E. 1960, *A new approach to linear filtering and prediction problems*. Trans. ASME, Journal of Basic Engineering 82:35-45.

[Kohonen 95] Kohonen, T. *Self-Organizing Maps*. Springer-Verlag, 1995.

[Krose 04] B. Krose, R. Bunschoten, S. Hagen, B. Terwun, and N. Vlassis, "Household robots look and learn," *IEEE Robotics and Automation Magazine*, vol. 11, no. 4, p. 45–52, December 2004.

[Latombe 91] Latombe, J. C., *Robot Motion Planning*, Kluwe Academic Publisher, 1991

[Lippman 87] Lippman, R.: *An Introduction to Computing with Neural Nets*. IEEE ASSP Magazine 4 (1987) 4-22

[Llarena 11] Llarena, A.; "*Redes Neuronales y Lógica Difusa en la creación de Mapas Topológicos*". Editorial Académica Española, ISBN 978-3-8454-8899-8, Noviembre 10, 2011.

[Llarena 11b] Llarena, A., Savage, J., Kuri, A. Escalante-Ramírez B.,"*Odometry.based Viterbi Localization with Artificial Neural Networks and Laser Range Finders for Mobile Robots*". Springer Science+Business Media: Journal of Intelligent & Robotic Systems, Sept. 2011, DOI: 10.1007/s10846-011-9627-8.

[Lowe 99] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh International Conference on Computer Vision(ICCV'99)*, Kerkyra, Greece, September 1999, p. 1150–1157.

[Lu 97] Lu, F. and Milios, Evangelos E., *Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans*, JIRS(18), No. 3, March 1997, p. 249-275. 9705

[Menegatti 04] E. P. E. Menegatti, M. Zoccarato and H. Ishiguro, "Image-based Monte-Carlo localisation with omnidirectional images," Robotics and Autonomous Systems, vol. 48, no. 1, p. 17–30, August 2004.

[Montessano 05] L. Montesano, J. Minguez, and L. Montano, "Probabilistic scan matching for motion estimation in unstructed environments", Procedings of the IEEE/RSJ Inte0l.l6igent Robots and Systems (IROS), vol. 1, pp. 1445-1450, 2005.

[Nayar 94] Nayar, S. K., Murase, H., Nene, S. A.: *Learning, Positioning, and Tracking Visual Appearance*. Proceedings of the IEEE International Conference on Robotics and Automation, 4 (1994) 3237-3244

[Pitt 97] M. K. Pitt and N. Shepard, "Filtering via simulation: auxiliary particle filters," tech. Rep., Department of Mathematics, Imperial College, London, October 1997.

[Rabiner 89] Rabiner, L. R., *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, *Proceedings of the IEEE*, vol. 77, no. 2, Feb. 1989, pgs 257 - 285.

[Racz 94] Racz, J. Dubrawski, A. *Mobile Robot Localization with an Artificial Neural Network*.

[Rofer 03] Röfer, T., Jüngel, M. (2003). Vision-Based Fast and Reactive Monte-Carlo Localization. In: Proceedings of the IEEE International

Conference on Robotics and Automation (ICRA-2003), Taipei, Taiwan. 856-861.

[Savage 98] Jesús Savage, Adalberto LLarena, Gerardo Carrera, Sergio Cuellar, David Esparza, Yukihiro Minami, Ulises Peñuelas; *ViRbot: A System for the Operation of Mobile Robots*. RoboCup 2007 Symposium, Atlanta, EU, 2007.

[Savage 05] J. Savage, M. Morales, E. Márquez, "The Use of Hidden Markov Models and Vector Quantization for Mobile Robot Localization", Robotics and Applications (RA 2005), IASTED, Boston, U.S.A.

[Smith 92] A.F.M. Smith and A.E. Gelfand, "Bayesian statistics without tears: a sampling-resampling perspective", *American Statician* 46(2), p. 84-88,1992

[Sridharan 05] Sridharan M., Kuhlmann G., and Stone P. Practical Vision-Based Monte Carlo Localization on a Legged Robot. In The IEEE International Conference on Robotics and Automation (ICRA 05), Barcelona, Spain, April 2005.

[Tamimi 04] Tamimi, H., Zell, A.: *Global Visual Localization of Mobile Robots Using Kernel Principal Component Analysis*. Proceedings of the IEEE International Conference on Intelligent Robots and Systems, 2 (2004) 1896-1901

[Thrun 00] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. *Robust Monte Carlo localization for mobile robots*. Artificial Intelligence, 128(1-2):99-141, 2000.

[Thrun 01] Thrun Sebastian, Fox Dieter, Burgard Wolfram, Dellaert Frank. *Robust Monte Carlo Localization for Mobile Robots*. Artificial Intelligence, Summer 2001

[Thrun 03] Thrun S., Montemerlo M.., Koller D., Wegbreit B., *FastSLAM: An Efficient Solution to the Simultaneous Localization And Mapping Problem with Unknown Data Association,* Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2003.

[Thrun 05] Thrun Sebastian, Burgard Wolfram, Fox Dieter. *Probabilistic Robotics*. MIT Press, Cambridge, Massachusetts, London, England 2005.

[Thrun 99] D. Fox, W. Burgard, and S. Thrun. *Markov localization for reliable robot navigation and people detection*. In Modeling and Planning for

Sensor-Based Intelligent Robot Systems. Springer Verlag, Berlin, 1999.

[Thrun 99b] D. Fox, W. Burgard, and S. Thrun. *Markov localization for mobile robots in dynamic environments*. Journal of Artificial Intelligence Research, 11:391-427, 1999.

[Ulrich 00] I. Ulrich and I. Nourbakhsh, "Appearance-based place recognition for topological localization," in Proc. of the IEEE International Conference on Robotics & Automation (ICRA'02), San Francisco, CA, April 2000, p. 1023–1029.

[UPM 11] Universidad Politécnica de Madrid. Departamento de Ingeniería Visual y Telecomunicaciones. Curso en Línea de Transformada Directa del Coseno. http://www.diac.upm.es/acceso_profesores/asignaturas/tdi/tdi/transformadas/pdf/dct.pdf

[Viterbi 67] Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, IEEE Transactions on Information Theory 13(2):260–269, April 1967. (The Viterbi decoding algorithm is described in section IV.)

[Vlassis 02] Vlassis, N., Motomura, Y., Krose, B.: *Supervised Dimension Reduction of Intrinsically Low-dimensional Data*. Neural Computation 14 (1) (2002) 191-215

[Yu 02] Jinxia Yu, Zixing Cai, and Zhuohua Duan. *Mobile Robot Self-localization Based on Feature Extraction of Laser Scanner Using Self-organizing Feature Mapping*

Con el amable permiso de Springer Science+Business Media: Journal of Intelligent & Robotic Systems, DOI: 10.1007/s10846-011-9627-8.

Apéndice A

Odometry-Based Viterbi Localization with Artificial Neural Networks and Laser Range Finders for Mobile Robots

Adalberto Llarena · Jesus Savage · Angel Kuri · Boris Escalante-Ramírez

Abstract This paper proposes an approach that solves the Robot Localization problem by using a conditional state-transition Hidden Markov Model (HMM). Through the use of Self Organized Maps (SOMs) a Tolerant Observation Model (TOM) is built, while odometer-dependent transition probabilities are used for building an Odometer-Dependent Motion Model (ODMM). By using the Viterbi Algorithm and establishing a trigger value when evaluating the state-transition updates, the presented approach can easily take care of Position Tracking (PT), Global Localization (GL) and Robot Kidnapping (RK) with an ease of implementation difficult to achieve in most of the state-of-the-art localization algorithms. Also, an optimization is presented to allow the algorithm to run in standard microprocessors in real time, without the need of huge probability gridmaps.

This work was supported in part by PAPIIT-DGAPA UNAM under Grants IN107609, IN113611 and IX100610, the Mexican Council of Science and Technology CONACYT and Posgrado en Ciencia e Ingeniería de la Computación PCIC, Universidad Nacional Autónoma de México UNAM.

A. Llarena (✉) · J. Savage · B. Escalante
Universidad Nacional Autónoma de México UNAM. Ciudad Universitaria no. 3000, Col. Copilco Universidad, Del. Coyoacán, México, D.F., C.P. 04360
e-mail: adallarena@aol.com

J. Savage · B. Escalante
e-mail: {savage, boris}@servidor.unam.mx

A. Kuri
Instituto Tecnológico Autónomo de México ITAM, Río Hondo No. 1, Col. Progreso Tizapán, México, D.F., C.P. 01080
e-mail: akuri@itam.mx

1. Introduction

In mobile robotics, one of the most basic problems to be solved is the Simultaneous Localization and Mapping or SLAM problem [1], where an autonomous robot must be capable of generating a representation often called a „map" of an unknown environment, at the same time that traverses it and gets localized into that environment representation.

Although robotic localization is considered a solved problem, it is far from being a closed investigation topic. It involves three aspects: a) data collecting and pre-processing to build an observation model, b) *a priori* motion kinematics knowledge for building a movement model (or odometer estimations, not always incorporated on cheaper or smaller robots) and c) an observation-movement relationship, used for updating the position estimation through time.

While computers are approaching the processing power of the human brain [2], the complexity of operating systems, algorithms and the sensorial robot information is also increasing. For this reason, algorithms and architectures suitable of being adapted or re-trained will constitute a good alternative in the future. The best example is the animal brain, capable of performing very complex tasks and adapting to the environment, all just done with neurons. Based on these premises, a kind of Artificial Neural Network (ANN) inspired in the way the brain organizes information, the Self Organized Map (SOM) has been widely used in robotics because its properties for organizing high-dimensional observation vectors. In this way, the nature has helped the researchers in finding novel ways to manage the increase in the data to be processed.

This paper is organized as follows: sections 1 to 3 present the general robotic localization problem and the most common solutions up to date. Sections 4 and 5 introduce the Viterbi Localization Method (VL) and propose a solution based on ANNs and odometry-based motion models named Odometry-dependent Viterbi Localization (OVL). Section 6 presents an error-tolerant observation model by using SOMs. Section 7 explains the full path reconstruction from discrete sample nodes. Section 8 shows the

method optimizations. Section 9 shows the OVL Algorithm. Sections 10 and 11 present the experiments and show the results. Finally, Sections 12 to 14 present the conclusions and the future work.

2. Previous Work

2.1 Robot Localization with Artificial Neural Networks

Since the first efforts for effectively locating a robot using ANNs the main purpose has been relating the robot pose as a function exclusively of the observations. Although this has been made possible using several kinds of ANNs such as Feed Forward Networks (FFNs) [3], Hopfield [4], Kohonen Self Organized Maps (SOMs) [5], Fuzzy-Adaptive Resonance Theory (Fuzzy-ARTs) [6], General Regression Neural Networks (GRNNs) [7] or Multi-Layered Perceptrons (MLPs) [8], such direct relationship between pose and observations makes it difficult to carry on a mixed hypothesis based on both odometry estimations and perceptual data because the displacement information is ignored.

2.2 Robot Localization and Hidden Markov Models

Probabilistic localization with Hidden Markov Models (HMMs) has been explored in the past by [9, 10] with good results. Unfortunately, with the discretization of the configuration space, large amounts of memory were required for storing a location probability distribution. For this reason they and other researchers started a new successful approach known as Monte Carlo Localization (MCL) [11]. The main feature of this method is the use of probabilistic Motion and Observation Models (MM and OM respectively). With the aid of these models, the new robot pose Probability Density Function (PDF) can be estimated based on odometry data and by comparing the actual observation against the stored model. This important modeling approach is also incorporated into the Kalman Filter (KF) Localization [12] technique and makes probabilistic methods tolerant to occlusions and sensor noise.

2.3 Viterbi Localization

In 2005, Savage et Al. [13] proposed a method that used the Viterbi Algorithm to solve the robot localization seen as a HMM with a fixed-probability state-transition model. That method was unable to give precise location estimations, due to the lack of an odometry-dependent Motion Model (ODMM). The present work extends that approach with an ODMM, incorporating rescaling techniques in order to avoid the tendency of probabilities to zero out after several iterations.

3. Robot Localization and HMMs

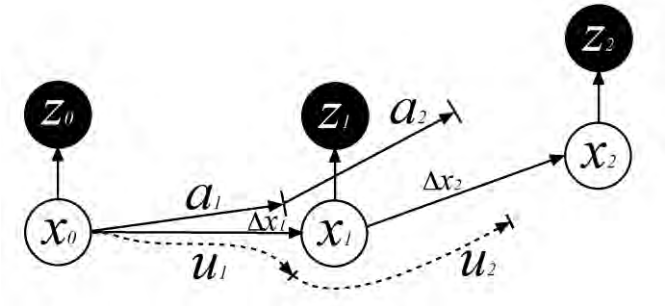
3.1 The Robot Localization Problem

The robot localization problem consists of determining the robot's pose with respect to some known references (a common origin in a Cartesian plane or environment landmarks), involving three basic aspects:

1) *Global Localization (GL)*. The mobile robot must determine its initial pose, based on the universe of known locations and sensor readings.
2) *Position Tracking (PT)*. The robot has prior knowledge about its previous location and keeps track of the position changes, through the integration of the successive displacements calculated by the PT algorithm.
3) *Robot Kidnapping (RK)*. The robot is abruptly taken out of its current location and it is repositioned into an arbitrary place of the working environment, but the robot is not informed about this change. As a consequence, the robot must be able to detect the position change and find its global location again.

Robot localization intends to determine the actual robot pose $\boldsymbol{x}_t$:(x_t, y_t, θ_t), where (x_t, y_t) are the robot coordinates in the Cartesian plane and θ_t is the robot"s heading, based on a set of control actions $a_{1:t\text{-}1}$, a set of odometry displacements estimations $u_{1:t}$, a set of previous visited locations $\boldsymbol{x}_{0:t\text{-}1}$ and a sensor readings set $z_{0:t}$. It is assumed that the robot starts at location $\boldsymbol{x}_0$ observing z_0 (Fig. 1).

Figure 1. Robot Localization Problem



A new control action a_1 is issued and the robot performs some displacement $\Delta \boldsymbol{x}_1$ according to its kinematics, arriving at location $\boldsymbol{x}_1$ with odometer estimation u_1 and observing z_1. A new action a_2 is issued, displacing the robot $\Delta \boldsymbol{x}_2$, arriving at $\boldsymbol{x}_2$, observing z_2 and estimating u_2. This process continues up to some time instant *t,* with a_t, $\Delta \boldsymbol{x}_t$, $\boldsymbol{x}_t$, u_t and z_t, where a re-localization method is launched to correct the robot position.

3.2 Classical Approaches in Robot Localization

Historically, two kinds of methods have been developed to solve the localization problem, *data-data* and *data-model* associations.

3.2.1 Data-data Associations

The *data-data* association calculates x_t as

(1)

The simplest way to do this association is by collecting and storing a set of observations at known locations $L:(x_i,z_i)$, gathered in a previous sampling phase. It is possible to find the closest observation z_k in L to a given observation z_t and assuming corresponding x_k as the true robot location. Although this association directly relates x_i with z_i and vice versa (apparently solving GL and RK but not PT), this procedure is not completely adequate because it finds the location most similar to a given observation, regardless of robot motions and sensor errors or occlusions.

Another kind of methods like [14] calculate successive robot displacements with x_t as $x_{t-1}+\Delta x_t$ where

(2)

By exploiting a relation between z_{t-1} and z_t they are able to find the exact amount of displacement Δx_t. They are adequate for calculating PT by integrating successive Δx_t but cannot therefore deal neither with GL nor RK.

Because *data-data* approximations directly relate position with observations they result very sensitive to sensor noise and occlusions. Moreover, they are incapable of deciding between two possible robot locations x_1 and x_2 with identical observation vectors, due to the lack of movement estimators.

3.2.2 Data-model Associations

The data-model associations calculate x_t by incorporating some robot and world models. While traversing the environment, some conditions must be met according with those models, thus:

(3)

where

(4)

is a set of models regarding the robot sensors S, the world model W, a location-observation relationship O and a motion model M.

3.3 The Sensor Model

The Sensor Model stores a representation about the way that specific sensors (like range finders, temperature sensors or digital cameras) behave under real world inputs. They model the sensor output z_k for a given measuring condition ϑ_r, based on specific sensor parameters ς_k according to

(5)

where ϑ_r is a measuring condition vector comprising important parameters such as the object's distance, scanning angle, surface incidence angle (for range scanners), surface color and roughness and etcetera. ς_k is the specific sensor parameter vector containing information such as sensibility, operative range, distortion and etcetera; used for predicting the noise added to z_k by the S function. For digital cameras ϑ_r is equivalent to extrinsic parameters while ς_k corresponds with the intrinsic parameters.

3.4 The World Model

The World Model W, often implemented trough a *map*, has a representation of the robot environment (such as the physical location of the objects and their attributes). It constitutes the reference frame where the robot is located into. It is important to have all the World Model's information as sensor independent as possible (the World Model must be *complete*). Regardless of the impossibility for any sensor to detect some features from a certain location inside the world (like black polished objects, invisible for the laser range finder) the World Model must incorporate as much characteristics as possible. In this sense, Sensor Fusion Methods [15, 16] are relevant to build more complete World Models than those conformed only with information coming from a single sensor.

Several kinds of maps are commonly used as world models such as occupancy grids [17], metric maps [18], topological maps [18, 19], feature maps [20] and many others. In any case the main goal in SLAM will be making the robot able to build by itself those maps, while exploring an unknown environment.

3.4 The Observation Model

The Observation Model (OM) O stores a direct association between locations x_r and observations z_r for a given sensor as

$$z_r = O(x_r) \tag{6}$$

Most of existing localization algorithms consider the environment as static. This issue simplifies the location procedure by applying the Markov Assumption [21] where the observation made at certain state is only dependent of the state itself (the current

robot's pose). Under normal operation conditions only a few section of the environment keeps static, basically the walls and some big furniture, the rest of objects (including human and robots) are constantly moving. For this reason, it is necessary to differentiate between *scene*, *view* and *observation*.

3.6 Scene, View and Observation

A *scene* corresponds with the instantaneous state W_t of all the actors inside the environment (i.e. the location of all the objects in the World Model). Inside a static environment as several methods propose, the world model does not change over time, so it warranties the applicability of the Markov Assumption.

A *view* is a particular section of the environment, only visible from a particular location and depending on some sensor parameters such as absolute location and heading, pan, tilt and field of view. In other words, a view is a portion of the environment suitable of being processed by the sensor at a given location.

An *observation* is the measure that a specific sensor performs from certain view. It is obvious that depending on the sensor model, and observation conditions, some of the world characteristics will not be perceivable by the sensor so the measure condition will lead to bad measuring errors produced for instance by lens distortion or moisture in case of having digital cameras.

3.7 Simulating the Observations

As an alternative to implementation of one Observation Model for every robot sensor by gathering a huge set of sample readings, the association x_r-z_r is often achieved by using a software simulator that predicts observation conditions ϑ_r from a given location x_r by using a world model *W*, having on mind specific sensor parameters ς_k as field of view, resolution (dimensionality), maximum and minimum measure ranges and etcetera:

$$J_r = O(W_t, x_r, V_k) \qquad (7)$$

The use of a simulation method is especially adequate when working with high dimensional observation vectors like cameras or range finders. By using ray-tracing techniques (for range based algorithms) or 3D renders and virtual cameras (for simulating the actual observation for vision systems) expected observations can be generated with the expense of processor and memory of course. Many authors build a readings [22] or features [23] databases with optimized algorithms like Kd-trees [11] to search for the direct or inverse location-observation value.

3.8 The Motion Model

Often referred as *Plant Model* in Probabilistic Localization [1], the Motion Model (MM) establishes the estimated movement the robot will perform under certain control action a_t based on odometry estimations u_t. Examples of these models are differential drive, omni-directional, synchro-drive or legged robot control. This model predicts the final robot location after performing a_t command.

Usually robots incorporate odometers to help the control modules in estimating the robot displacements based on u_t.

In general, a Motion Model is a transfer function that computes the new robot location x_{t+1}, in terms of current location x_t, robot constrains ρ_r and odometry data u_t.

(8)

The Motion Model imposes the restrictions a mobile robot must meet between consecutive movements. While a set of encoders is supposed to be mounted over the robot wheels and the robot dimensions and mechanics are known, the Motion Model establishes which locations transfers are valid (or more 'probable' in terms of probability theory) between consecutive movements, based either on odometry estimations u_t (for wheeled robots), the commanded movement a_t or a movement statistical analysis (for legged robots).

In this way, with the help of the abovementioned models, *data-model* associations match actual observations against those predicted by OM and MM.

3.8 Probabilistic Localization

The most widely used *data-model* localization approaches are of probabilistic nature, where the goal is estimating the posteriori Probability Density Function (PDF) $p(x_t)$ according to

(9)

where x_t is called the *robot state* representing the robot pose.

The most popular probabilistic localization methods are Kalman Filters (KFs) and Particle Filters (PFs). Both are basically *data-model* associators.

Probabilistic methods are also known as Bayesian Filters because they support their operation upon certain *a-priori* assumptions:

3.8.1 Markov Asumption

The Markov Assumption [21] considers observations z_t dependent only on the current state x_t (and the immediate previous and next states), this is

(10)

and therefore

(11)

in consequence

(12)

The Markov Assumption is not completely valid in crowded environments or scenarios with moving objects where observations are difficult to predict only in function of the current robot location. More robust methods incorporate also the robot observation history in order to give more accurate predictions.

3.8.2 Bayes Rule

In order to calculate $Bel(t)$ KFs and PFs apply Bayes Rule [24], having on mind Eq. (10) to (12) as

$$p(x_t \mid z_{0:t}, x_{0:t-1}, a_{1:t-1}, u_{1:t-1}) = \frac{p(z_t \mid x_t)\, p(x_t \mid u_{t-1})}{p(z_t)} \tag{13}$$

While KFs keep track of the position using parametric continuous Gaussian functions and a linear motion and observation models (something very difficult to achieve in real scenarios), PFs carry on a multi-hypothesis posterior PDF through a set of individual samples called 'particles', each one of them corresponding with a possible robot location.

3.9 Kalman Filters

Kalman Filters [12] have a linear Motion Model (so called the *Plant Model*) of the form

$$x(k+1) = \mathrm{F}\, x(k) + \mathrm{G} u(k) + \mathrm{C} v \tag{14}$$

where $x(k)$ is the previous state, Φ indicates the state variation in the absence of inputs, Γ is a state-transition matrix, $u(k)$ is the control command and C_v is the state covariance matrix.

They also consider a linear Observation Model

$$z(k+1) = \mathrm{L}_E x(k) + W_t \tag{15}$$

where $z_t(k+1)$ indicates the sensor reading at time $k+1$, Λ_E is a matrix relating observations with robot states $x(k)$. $W_t(k)$ is a zero-mean Gaussian noise function.

In its main stage, KFs compute a prediction of the sensor readings for the next state *k+1* by using Bayes Rule with

$$Kt = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1} \tag{16}$$

where $\bar{\Sigma}_t$, C_t and Q_t are matrixes. Kt is known as Kalman Gain and requires a matrix inversion with $O(k^{2.4})$ [25] being k the dimension of the observation vector.

Kalman Filters, carry on a robot location estimate based on the parameters of Gaussian functions, they assume a Gaussian, zero mean movement and observation errors. This, together with the linearity model assumption and matrix inversion makes difficult to implement KFs under real world conditions in limited devices. For this reason some improvements have been developed as the Extended Kalman Filter EKF [26], the Unscented Kalman Filter UKF [27] and the Information Filter IF [1]. KFs can carry on only one localization hypothesis, limiting their use in changing environments.

3.10 Monte Carlo Localization

Particle Filters surpass some of the KFs limitations by approximating the posterior PDF $p(x_t)$ by a set of samples called „particles". The particle density over the configuration space reflects the probability $p(x_t)$ calculated according to Bayes Rule as

(17)

where $Bel(x_t)$ is the pose belief at time t, $p(z_t \mid x_t)$ is the Observation Model, $p(x_t \mid x_{t-1}, u_t)$ is the Motion Model, $Bel(x_{t-1})$ the pose belief at time *t-1* and η is a normalizer.

The most popular PF is the Monte Carlo Localization method (MCL) [11]. MCL approximates Eq. (17) by randomly sampling locations according to the Motion Model $p(x_t \mid x_{t-1}, u_t)$ based on robot motions and odometer estimations u_t. After a sample set is generated, an importance weight w_i is calculated for every particle location x_i based on the Observation Model $p(z_t \mid x_t)$ After computing all pairs (x_i, w_i), every w_i is rescaled, multiplying its value by their individual scaling factor η_i (equivalent to $p(z_t)$ in Bayes Rule) calculated with

$$\eta_i = \frac{w_i}{\sum_{j=1}^{n} w_j} \tag{18}$$

where n is the size of the sample set in the PF.

Then, a method known as *Importance Sampling* renews the entire particle set by generating new random particles, based on importance factors w_i, generating more new samples in those locations with higher matching between real observations z_t and supposed observations z_x for that location. This process avoids the effects of progressive degradation of $Bel(x_t)$ due to the implicit computation of probabilities that tend to zero.

In many MCL implementations for laser range finders, the OM is computed by using a ray-tracing simulator in real-time, based on world model *W* or a set of stored observations *L* at known locations and an optimized-search method. Observation probability is calculated by evaluating a Probabilistic Sensor Model (as the one proposed in [11] for range scanners) for every simulated individual reading z_{x_i} and the actual individual sensor reading z_{r_i} according with

$$p(z_t \mid x_t) = \prod_{i=1}^{n} p(z_{t_i} \mid z_{x_i}) \tag{19}$$

By incorporating measuring errors and occlusion probabilities in the Probabilistic Sensor Model $p(z_{k_i} \mid z_{x_i})$ the MCL filter is capable to deal with both sensor errors and occlusions. By generating a small extra set of randomly samples all over the configuration space eventually MCL can recover from robot kidnapping.

Finally, as every particle has an importance weight associated to it, the current robot location can be computed as

$$E(x_t) = \sum_{i=1}^{n} (x_i \cdot w_i) \tag{20}$$

In conclusion, MCL has some important advantages

1) *Multi Hypotheses*. The sample set distribution allows managing many possible robot locations at once. Once the particles have converged to a narrow area, it can be assumed as the true robot pose.
2) *Completeness*. By having an evenly distributed initial particle set, MCL can handle GL. By adding some random particles across all the configuration space in the resampling process MCL can eventually recover from a RK. As robot motions u_t are used for sampling the new particle set, MCL

can handle PT.

3) *Memory Reduction*. It reduces drastically the amount of memory required by discrete localization techniques that use three-dimensional spatial grids as Markov Localization [9]. It is capable of integrating observations at high rates.
4) *Easiness of Implementation*. MCL It is easy to implement in systems with limited resources of memory and speed, because it does not require matrix inversion like KFs.

Unfortunately, MCL also has some disadvantages

1) *Sensor Error*. The MCL performance decays when having sensors with low error rates (like laser range finders).
2) *Simulation Processor Cost*. As each particle represent a possible robot location, the Observation Model must be able to predict a precise observation for each particle, this is often performed with the help of a software simulator. When managing laser range finders with hundreds of readings per scan or mounted vision cameras, the simulation process results very expensive in terms of processor consumption.
3) *Number of Particles*. The MCL performance depends mostly on the re-sampling process. At less one particle must be located very close to the actual robot position to ensure the method's convergence. In some scenarios bigger than a dozen of meters long and width, thousands of particles can be needed for performing GL or RK.
4) *Independent Sensor Readings*. As proposed in Eq. (19) individual sensor readings are considered as conditionally independent. The true is exactly the opposite: individual sensor readings z_{r_i} are highly correlated.

Some additions to MCL have been proposed by the authors to overcome some of these disadvantages like an inverse sampling model Mixture-MCL [11] for predicting locations with a Kd-tree forest storing observations, or the Rao-Blackwellized Particle Filters [29] that carries on a whole map with every particle. They both increase the performance of PFs but also their complexity and thus the required computer resources for running those algorithms.

In this form, spatial grids and particle filters have proved their applicability to the SLAM problem [11], but they require large amounts of processing time and memory resources. Topological localization [13] has proved its simplicity in terms of computational resources but they fail in the presence of occlusions and partial readings due to range sensors sensing limits. By the other hand, one fundamental issue is the

inherent complexity of the localization problem. It requires many resources for every square meter added to the environment.

3.2 Hidden Markov Models and Robot Localization

Hidden Markov Models (HMMs) [21] are probabilistic directed graphs, where the current state $x_i \in X$:$[x_1 .. x_n]$ is not directly observable (i.e. measurable). There is an observation $z_i \in Z$:$[s_1 .. s_m]$ associated with every state in the model and through the observation sequence and model parameters the hidden variable (the state) can be estimated. X is a set of states and Z is a set of observations (symbols).

The parameters of a Hidden Markov Model λ are

(21)

where $\mathbf{A} = \{a_{ij}\}$ is a state-transition probability distribution over $X \times X$ such as

(22)

$\mathbf{B} = \{b_j(k)\}$ is an observation probability distribution over $X \times Z$ (relating states with observed symbols) such that

(23)

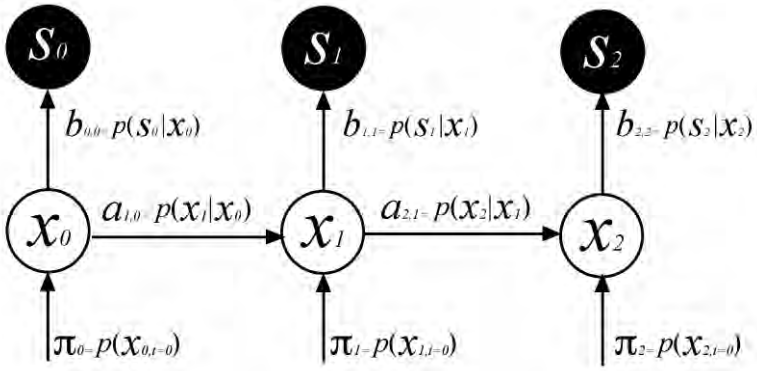
and $\mathbf{\Pi} = \{\pi_i\}$ is an initial state distribution

(24)

where n is the number of states and m is the number of symbols in the HMM.

A, **B** and $\mathbf{\Pi}$ can be expressed in matrix form: $\mathbf{A}(n \times n)$ is the state transition matrix, $\mathbf{B}(n \times m)$ is the observation matrix and $\mathbf{\Pi}(1 \times n)$ is the initial state probability matrix.

Figure 2. Robot Localization Problem as HMM.



4. Viterbi Localization

As proposed in [13], robot localization can be seen as the process of estimating the hidden variable in a HMM (the current state x corresponding to a discrete robot pose, see Fig. 2), based on the sensor observation sequence $\{z_0, z_1, \ldots, z_t\}$.

4.1 The Viterbi Algorithm

The Viterbi Algorithm (VA) [30] proposes a solution for estimating $p(O|\lambda)$, i.e. the probability of the observations O given the model λ, where $O = (z_0, z_1, ..., z_t)$ is the observation sequence and λ is the HMM. The Viterbi Algorithm has a two ways of calculating $p(O|\lambda)$, the forward and backward procedures.

The Viterbi"s forward procedure (VFP) has three steps:

Initialization

(25)

$\alpha_1(i)$ stores the joint probability $p(x_i, z_1)$ of being at state i at t=0 and observing z_1, equals to the probability π_i of starting at state i by the probability of observing the symbol z_1 at that state, while n is the number of states in the model.

Induction

$$\alpha_t(j) = b_j(z_t)\left[\sum_{i=1}^{n} a_{ij}\,\alpha_{t-1}(i)\right] \quad \begin{matrix} 1 \le t \le t_k \\ 1 \le j \le n \end{matrix} \tag{26}$$

where t_k *is* the time when the last observation z_k was taken.

$\alpha_t(j)$ computes the joint probability $p(x_j, z_{0:t})$ of being at state i at time t while observing z_t. By summing the probability $p(x_j \mid x_i, z_{0:t-1})$ of performing a transition to the state j from every state i (total probability) $p(x_j , z_{0:t-1})$ is obtained. If this result is multiplied by the probability of observing the symbol z_t at the state j then results $p(x_j, z_{0:t})$, because $p(x_j, z_{0:t}) = p(z_t \mid x_j)\, p(x_j, z_{0:t-1})$.

Termination

$$p(O \mid \lambda) = \sum_{i=1}^{n} \alpha_t(i) \tag{27}$$

where

□ . (28)

By rewriting Eq. (26), according with (23) and (24)

$$\alpha_t(j) = p(z_t \mid x_j) \sum_{i=1}^{n} p(x_j \mid x_i)\, \alpha_{t-1}(i) \tag{29}$$

can be noticed that Eq. (26) computes the joint probability of the observation sequence up to z_t finishing at state x_j. By summing all $\alpha_t(j)$ with $1 \leq j \leq n$, $p(z_{0:t})$ is obtained (Eq. 7).

4.2 Viterbi Algorithm vs. Probabilistic Localization Approaches

Main probabilistic localization methods estimate the robot location given the observation sequence, this is $p(x_j \mid z_{0:t})$. They apply the Bayes Rule (BR) to simplify this calculation as $p(x_j \mid z_{0:t}) = p(z_{0:t} \mid x_j)\, p(x_j) / p(z_{0:t})$.
If we take a look to the Particle Filters (PFs) update rule [11]

(30)

it can be seen that the BR is applied by having $\eta = p(z_t)$. If we compare Eq. (29) with Eq. (30) it can be seen that (29) is the discrete case of (30), except by two important absences: the term u_t (the motion estimation) and the normalizer η.

In a similar way, Eq. (22) corresponds to $p(x_t \mid x_{t-1}, u_t)$, the Particle Filter Motion Model M (the state-transition rule in Mobile Robotics, regardless of u_t) while Eq. (23) would precisely correspond to an Observation Model O (the relationship between locations and observations). In this sense, Particle Filters implement continuous versions of the Viterbi Algorithm with an u_t-dependent Motion Model. As shown above, Observation and Motion Models have their origin in the Viterbi Algorithm.

4.3 Viterbi Forward Procedure

The Viterbi Forward Procedure (VFP) in (29) computes the observation probability in a very similar way that PFs do, except by the normalizer η and the odometer estimation u_t. By modifying the VFP incorporating these parameters, robot pose estimation can be computed in a discrete form, without the need of solving a continuous model as Eq. (30) with the corresponding saving of time and resources. If the physical location of the states is known in advance, then the location-observation relationship can be built off-line, avoiding the need of precise observation estimations in real-time as PFs and KFs need [11, 12] often computed with the aid of a software simulator (through a metric map and ray-tracing techniques for simulating range scans at given locations).

By substituting (26) in (27) it can be shown that

$$\sum_{i=1}^{n} p(x_t, z_{0:t}) = p(z_{0:t}) \tag{31}$$

and therefore

$$p(x_t \mid z_{0:t}) = \frac{\alpha_t(i)}{\sum_{j=1}^{n} \alpha_t(j)} \tag{32}$$

that exactly corresponds to the rescaling factor η in PFs.

In order to solve Eq. (29), $p(x_j \mid x_i)$ can be computed as

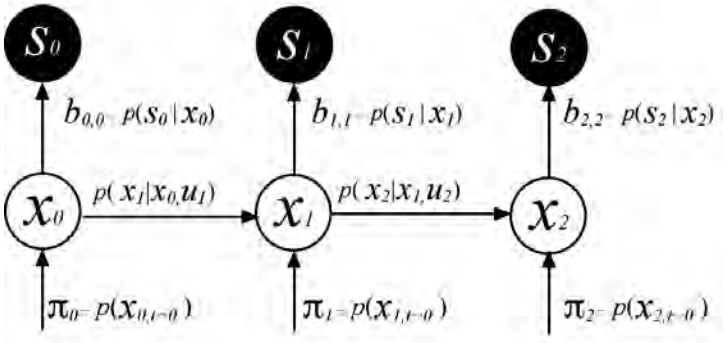
$$p(x_j \mid x_i) = p(x_{t+1} = j \mid x_t = i, u_t) \tag{33}$$

Unfortunately, due to the unpredictable nature of control actions $a_{1:t-1}$ that produce robot displacements estimations $u_{1:t}$, the former integral computation could result impractical, because it would be necessary to calculate *a priori* the overall transition probability between two consecutive robot locations, regardless of the control actions issued to the robot.

4.4 Disadvantages of Viterbi Localization against Particle Filters

The Viterbi Localization (VL) presented in [13] which approximated $p(x_j \mid x_i)$ through a PDF by randomly simulating robot displacements discards one of the most important parameters involved in the pose estimation (especially when solving Position Tracking), the odometry displacement estimation u_t. Also, this lack of odometry data makes the localization method incapable of deciding between two consecutive locations with similar observations z_t located at the same distance to the previous location $\boldsymbol{x}_{t-1}$, something very common in long corridors. An alternative to the computation of $p(x_j \mid x_i)$ is including u_t both in the HMM and the VA (Fig. 3).

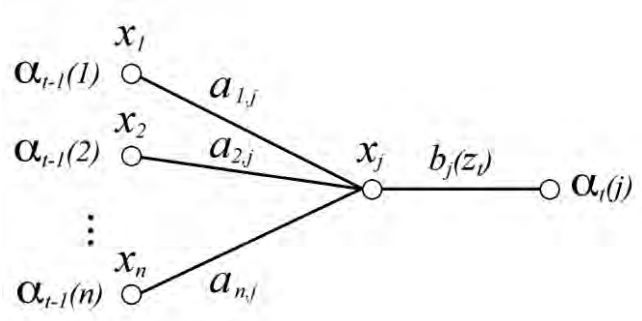
Figure 3. Modified HMM for robot localization.



4.5 Viterbi Localization Mathematical Foundations

Trellis diagrams are interesting tools to understand the VA (Fig. 4), since they show in a graphical form how the forward VA procedure works. As the Figure 4 depicts, in the induction phase of the forward procedure there is a summation that implies the state-transition probabilities a_{ij} that are part of the parameters of the HMM.

Figure 4. Trellis diagram of Viterbi‟s forward procedure in HMM.



By using matrix notation and substituting (26) in (27), the forward phase of the VA can be rewritten as

$$\alpha_t(x_t) = \eta \Phi_{t-1} \mathbf{A} \mathbf{B}_t \tag{34}$$

A and **B** are the state-transition and observation matrices of the HMM respectively, while

$$\Phi_{t-1} = [\alpha_{t-1}(1) \; \alpha_{t-1}(2) \; \ldots \; \alpha_{t-1}(n)] \tag{35}$$

is the previous belief vector and

$$\nu_t = [0 \; 0 \; \ldots \; 1 \; \ldots \; 0]^T \tag{36}$$

is a binary unitary vector with the exact correspondence of z_t with every symbol $[s_1 .. s_m] \in Z$. For instance, if $z_t = s_1$:

$$\nu_t = [1 \; 0 \; \ldots \; 0]^T .$$

By analyzing the product $\Phi_{t-1}\mathbf{A}$ it can be noticed that

$$\begin{bmatrix} \alpha_{t-1}(1) \\ \alpha_{t-1}(2) \\ \vdots \\ \alpha_{t-1}(n) \end{bmatrix}^T \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix} = \begin{bmatrix} p(x_1, z_{0:t-1}) \\ p(x_2, z_{0:t-1}) \\ \vdots \\ p(x_n, z_{0:t-1}) \end{bmatrix}^T \tag{37}$$

and similarly, the product $\mathbf{B}\nu^T$

$$\begin{bmatrix} b_{11} & b_{12} & \ldots & b_{1m} \\ b_{21} & b_{22} & \ldots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \ldots & b_{nm} \end{bmatrix} \begin{bmatrix} (z_k = s_0) \\ (z_k = s_1) \\ \vdots \\ (z_k = s_{m-1}) \end{bmatrix} = \begin{bmatrix} b_1(z_t) \\ b_2(z_t) \\ \vdots \\ b_n(z_t) \end{bmatrix} \tag{38}$$

Now, let us define a Dimensional Vector Product (DVP) as

(39)

and a Dimensional Vector Contraction (DVC) as

(40)

then, the dot product could be expressed as

(41)

By applying $\mathbf{\Phi}_{t-1}\mathbf{A} \otimes \mathbf{B}\,v^{\mathrm{T}}$ (giving more precedence to matrix multiplications than the dimensional product), a new $\mathbf{\Phi}_t$ is computed as

(42)

As $b_i(z_t) = p(z_t \mid x_i)$, when multiplying $[\mathbf{\Phi}_{t-1}\mathbf{A}]$ by $[\mathbf{B}\,v^{\mathrm{T}}]$ in Eq. (34) results (dot product formula):

$$\sum_{i=1}^{n} p(z_t \mid x_i)\, p(x_i, z_{0:t-1}) = p(z_{0:t}) \,. \tag{43}$$

It can be seen that Eq. (42) is included in Eq. (43), and

$$p(x_i \mid z_{0:t}) = \frac{p(x_i, z_{0:t})}{p(z_{0:t})} = \frac{\Phi_t(i)}{p(z_{0:t})} \,. \tag{44}$$

The forward procedure of VA calculates the total probability $p(O|\lambda)$ by having $\{z_0, z_1, ..., z_t\}$ as the observation sequence. After the initialization step when $\mathbf{\Phi}_0(i) = \pi_i\, b_i(z_0)$, in each iteration $\mathbf{\Phi}_t(i)$ computes

(45)

that is, the joint probability $p(x_i, z_{0:t})$. If summating all $\mathbf{\Phi}_t(i)$, with $1 \le i \le n$, on every iteration then $p(z_{0:t})$ is obtained, corresponding to $p(O|\lambda)$.

If every $\mathbf{\Phi}_t(i)$ in $\mathbf{\Phi}_t$ obtained with Eq. (42) is divided by $p(z_{0:t})$ (by using Eq. (44) before the next iteration, then

(46)

$$\frac{\Phi_t(i)}{\sum_{i=1}^{n}\Phi_t(i)} = \Phi_t(i) = Be_t(i) \tag{47}$$

As shown above, Viterbi‟s forward procedure can be easily adapted for carrying on a pose belief.

□

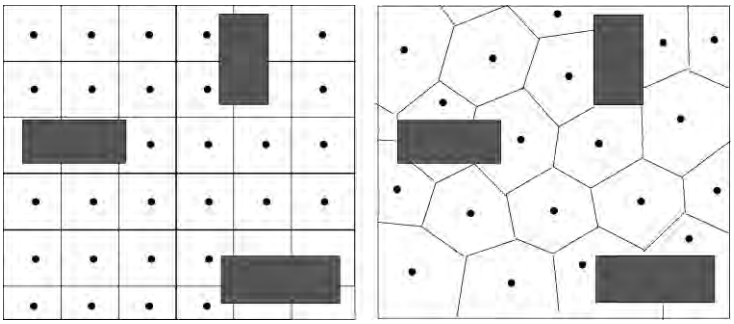
5. Motion Model Implementation

Based on a scan-matching algorithm as [14] or in the robot self-odometry, it is possible to build a Odometry-Dependent Motion Model (ODMM) with differential movement estimations u_t, plus the previous pose beliefs, in order to compute a complete motion estimation as Particle Filters do.

5.1 Partitioning the State-Space

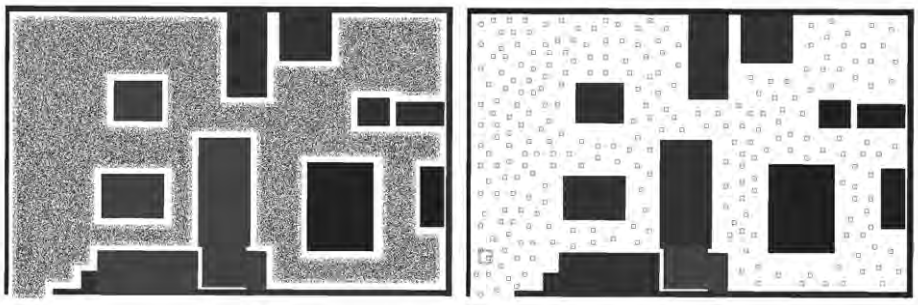
The Viterbi Algorithm operates in a discrete state-space. For this reason it is necessary to partition the working environment -the Configuration Space (CS)- into discrete regions. The CS is usually represented with occupancy grids (Fig. 5, left) or metric maps, then the partitioning can be easily done [31, 32]. In this work a Vector Quantization Network (VQN) was used because it provides locations uniformly distributed, similar to Voronoi Diagrams (Fig. 5, right).

Figure 5. Space-state partitions from [33]: (left) Occupancy gridmaps. (Right) Voronoi Diagrams.



From the CS representation, a random sampling L:(x_i, y_i, θ_i) of free locations and headings can be obtained (Fig. 6, left). By using a VQN fed with these vectors, after a few number of training epochs, the found vectors will correspond to the states of the HMM (Fig. 6, right).

Figure 6. Random Sampling (left) and Vector Quantization Network (right).

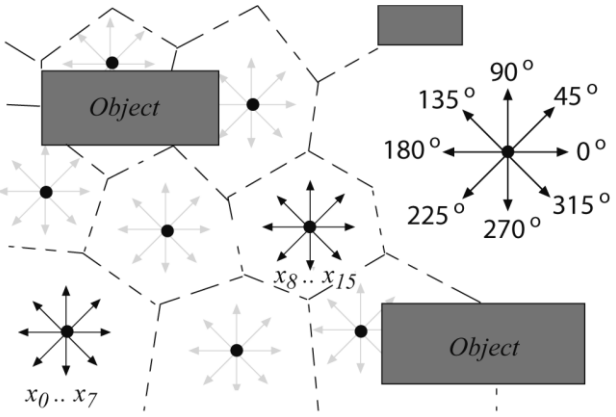


The advantage of this method is that the final number of states (nodes) can be decided in advance and the final locations will be uniformly-distributed.

5.2 Vector Quantization Considerations

Although Vector Quantization (VQ) can be performed in the three dimensions of the state-space (x, y and θ), more convenient (and faster) results are obtained by performing only a 2D state partitioning (x and y dimensions) and the final set of 3D states can be built by sharing every xy location of the 2D nodes with a group of absolute headings like [0, 0.78, 1.57, ... , 5.49], where every value stands for an absolute heading angle respect to the global coordinates frame expressed in radians (Fig. 7). By dividing 2Π by a fixed number of steps (eight in the former example) consecutive absolute headings can be defined from [0..2Π).

Figure 7. State absolute heading composition. The arrow indicates the node"s heading.



In this way, if a 2D node location is (3.2, 5.4) it is possible to set a group of eight nodes (states) with absolute locations (3.2, 5.4, 0), (3.2, 5.4, 0.78), ..., (3.2, 5.4, 5.49), and so on. This discretization reduces the memory needed for storing the node‟s location as it is assumed that for a given *xy* obstacle-free location the robot is able to be oriented in any direction. Otherwise, if VQ is performed over the 3 dimensions of the state-space, as θ is expressed in radians going either from [-π..π] or [0..2π], the VQN will select during training the same neuron *i* as the closest for a small neighborhood of (x_i, y_i) locations, regardless of θ_i (due to its smaller value compared with *x* and *y*). As a result, all found discretized nodes will have the same heading: 0 radians for θ_i in [π..π] or π radians for θ_i in [0..2π], because the VQN will compute the average θ_i. In this case all the HMM states would have the same absolute heading and it would be impossible to estimate the true robot‟s heading with that set of states if the robot is oriented in a different direction than the one all the states have.

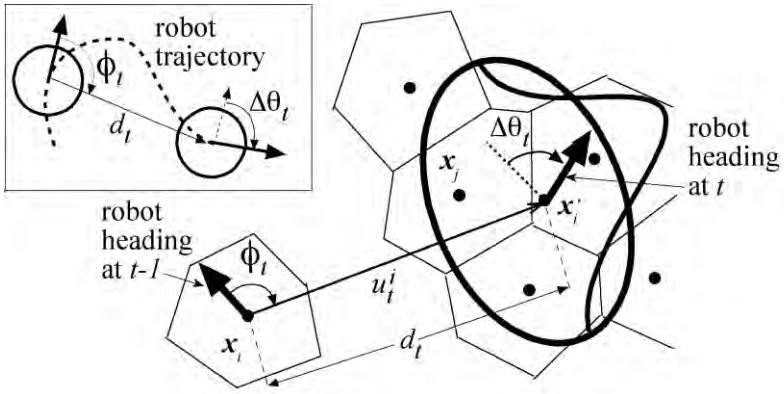
5.3 Transition Vectors

If the node (state) locations in absolute world coordinates $\mathbf{x}=(x_1, y_1, \theta_1)$ are considered as fixed, the Motion Model has to estimate the overall transition probability, given the odometry estimation u_t (or the a_t command in systems without odometers), this is $p(x_j \mid x_i, u_t)$, where u_t is:

$$u_t = (d_t, \phi_t, \Delta\theta_t) \tag{48}$$

where d_t is the distance between consecutive robot locations $\mathbf{x}_i$ and $\mathbf{x'}_i$, ϕ_t is the direction of the displacement (relative to the robot‟s front), and $\Delta\theta_t$ is the absolute heading change (Fig. 8, upper-left).

Figure 8. Actual robot movement and the Motion Model parameters.



For each state x_i it is possible to transform the odometry estimation u_t (given in a robot centered coordinate system) into an absolute displacement $u_t^i = (u_{t_x}^i, u_{t_y}^i, u_{t_\theta}^i)$ relative to each node absolute location $\boldsymbol{x}_i$ with

$$u_{t_x}^i = d_t \cos(\theta_i + \phi_t) \quad (49)$$

$$u_{t_y}^i = d_t \sin(\theta_i + \phi_t) \quad (50)$$

$$u_{t_\theta}^i = \Delta\theta_t \quad (51)$$

Once the absolute displacement has been calculated, it can be added to $\boldsymbol{x}_i$, computing a new absolute location

$$\boldsymbol{x}'_i = \boldsymbol{x}_i + u_t^i \quad (52)$$

The closer $\boldsymbol{x}'_i$ gets to every pose $\boldsymbol{x}_j$, the larger the probability of a true transition between x_i and x_j will exist.

One important issue is that, based on the connectivity network between nodes, not all transitions $p(x_j \mid x_i, u_t)$ would be valid, due to the existence of obstacles in the environment. In our case, all transitions are considered valid because under continuous movement conditions it is perfectly possible for a mobile robot to avoid an object by circumnavigating its contours. If location $\boldsymbol{x}_i$ is computed at time t-1 (before avoiding the obstacle) and the robot pose reaches $\boldsymbol{x}_j$ at time t (when the robot has surpassed the obstacle), u_t will reflect this change of pose, corresponding to a perfectly valid displacement.

5.4 Estimating Transition Probabilities

As $\boldsymbol{x}_i$ is an absolute state location and u_t is the robot-centered displacement estimation, the transition probability of reaching $\boldsymbol{x}_j$ starting from $\boldsymbol{x}_i$ will depend on how $\boldsymbol{x}'_i$ approximates to $\boldsymbol{x}_j$, in a global coordinates frame, this is:

$$p(x_j \mid x_i, u_t) = \frac{D(\boldsymbol{x}'_i, \boldsymbol{x}_j)}{\sum_{j=1}^{n} D(\boldsymbol{x}'_i, \boldsymbol{x}_j)} \tag{53}$$

where D is a distance-based PDF that can be a Gaussian function centered at $\boldsymbol{x}'_i$ (Fig. 6, right):

$$I(x_i, x_j) = e^{-\left(\frac{(x_{jw} - x_{iw})^2}{2\sigma_d^2} + \frac{(y_{jw} - y_{iw})^2}{2\sigma_d^2} + \frac{(\theta_{jw} - \theta_{iw})^2}{2\sigma_\theta^2} \right)} \tag{54}$$

where σ_d and σ_θ can be set depending on the *xy* node's spatial separation to become more or less inclusive (the *w* sub-index stands for „in world coordinates").

In this way, a new transition matrix **A** can be computed in each iteration, based on the u_t estimation by assigning

$$a_{ij} = p(x_j \mid x_i, u_t) \tag{55}$$

calculated with Eq. (53). As Figure 6 shows, $p(x_j \mid x_i, u_t)$ depends on how probable is that u_t takes $\boldsymbol{x}_i$ exactly into $\boldsymbol{x}_j$.

5.5 Complexity

Derived from (53) it can be seen that the computation of **A** matrix is $O(n^2)$ (where n is the number of nodes (states) in the configuration space) because it implies calculating a Euclidean distance between $\boldsymbol{x}'_i$ and every $\boldsymbol{x}_j$ given u_t, $\forall\ x_i, x_j \in X$. Based on this premise, the total number of required operations for the computation of **A** in each iteration could make this method incapable to run in real-time, considering that there will be only a small number of significant transitions for every node (those inside the influence ratio of D function) and the rest of transition probabilities would be almost

zero. These extra calculations can be avoided if Eq. (53) is computed only for those states into the neighborhood of $\boldsymbol{x}'_i$, having numerical significance for D, the remaining transitions can be assumed with probability near to zero.

6. Observation Model Construction

6.1 Random Sampling

Once a space partitioning has been computed, it is necessary to relate observations with locations for building the Observation Model (OM). This implies obtaining a Probability Density Function (PDF) of observations per discrete state. For this purpose, a set of observations (raw sensor vectors) must be obtained from random locations. The goal is to have a uniformly distributed observation sampling to relate enough observations with every discrete robot pose (state) of the model without introducing observation tendencies.

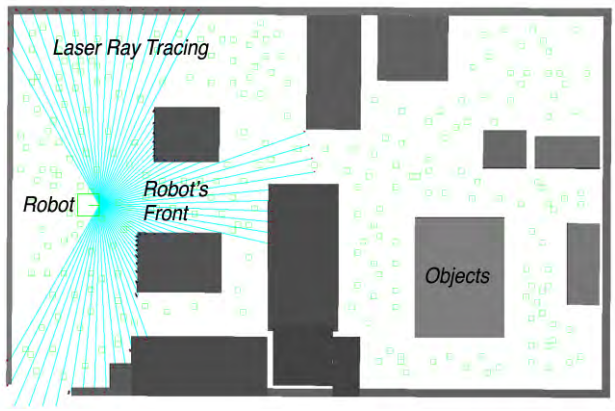
The main objective of this sampling is to have enough observations per state in order to build a complete observation PDF. If some observation probable to occur at a given state is not contemplated, the algorithm would fail because the OM does not relate that state-observation. In the next sections a Tolerant Observation Model (TOM) will be presented for dealing with such cases.

It is convenient using the same set of random locations generated in section 5.1 for obtaining the 'supposed' observation at each random location (Fig. 5), thus using a single set of random locations with their corresponding simulated observation for building the OM. While more similar the artificially generated observation to the actual sensor data, more precise the observation model will become. For this reason it will be important to use a method which provides as much realistic observations as possible.

6.2 Simulating Observations vs. Real Gathering

The best way to do this is by artificially generating those observations with a previously built World Model of the environment (metric map) and a software simulator with ray tracing (Fig. 9).

Figure 9. Metric map and simulated range scan (cyan).



The use of a metrical representation avoids the need of collecting as many samples as needed with the real robot for almost every navigable pose (the whole Configuration Space), something clearly impractical in environments as those where service robots must operate (Fig. 10).

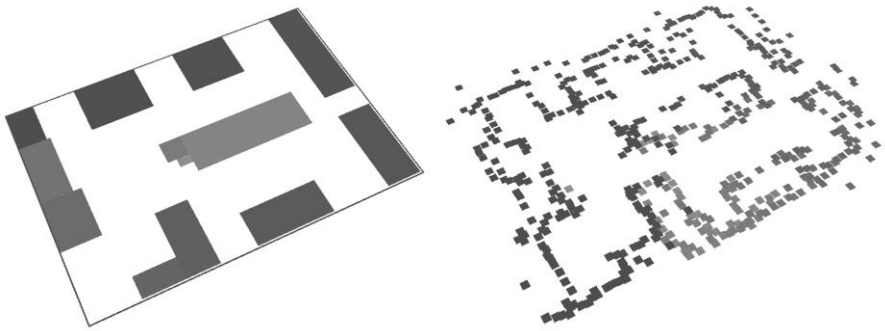
Figure 10. Typical environment for a service robot. (Left) Real env. (Right) Rough 3D model.

By the other hand, if a real robot is used for collecting observations inside the real environment, it is almost impossible to ensure that the OM will have enough observations for all navigable locations and headings (states), otherwise the Observation Model will be incomplete. The only case where using real samples for building the OM is convenient is in such cases where the mobile robot is always displacing along the same trajectories over a track or railroad.

6.3 Building a Metrical Representation

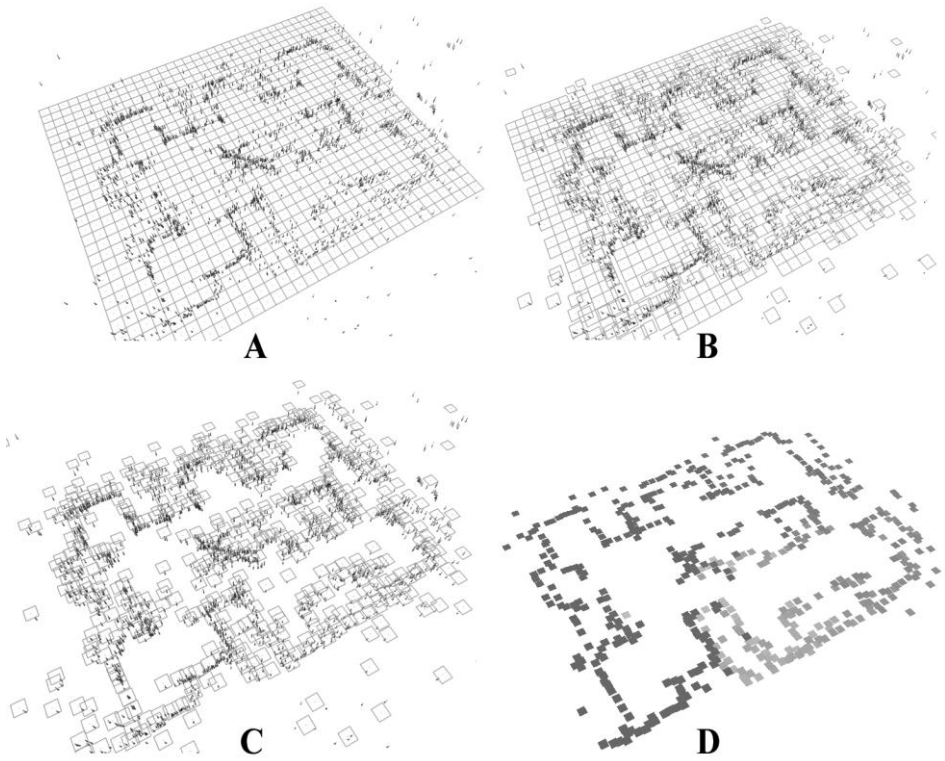
Although it is possible to use a rough polygonal representation by manually measuring the objects in the environment (Fig. 11, left), more accurate environment representations and simulations, as those obtained by generating a map of the environment using the actual robot sensors (Fig. 11, right), give better results because the accuracy of the simulated observations is greater.

Figure 11. Rough (left) and detailed (right) maps of the environment in Fig. 10.



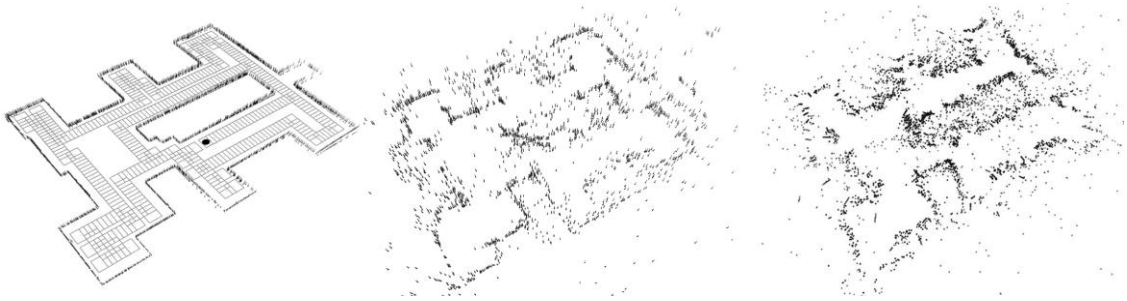
Both types of Metrical Maps were used in the present work for evaluating the proposed approach. A metrical representation of a working environment scanned with the actual robot sensors (a laser range scanner) was built by using a Clustering Artificial Neural Network (CANN) [34] with 10000 neurons (Fig. 12A). The network was fed with the (x,y) coordinates corresponding to projecting laser range scans as points into the xy plane. The CANN generates a metrical representation by placing clusters of a fixed size (10 cm × 10 cm) grouping the xy points (Fig. 12B). After 300 training epochs those neurons that were not updated (i.e. there are far away from any xy point) are removed (Fig. 12C) and a metrical representation is obtained (Fig. 12D). Each cluster represents an object.

Figure 12. Clustering ANN. A) Initialization. B) Training. C) Pruning. D) Final Map.



Once the detailed metric map is built with the CANN, the difference between actual observations (Fig. 13, center) and those simulated is notorious for the case of the rough map (Fig. 13, left) and the detailed map (Fig. 13, right).

Figure 13. Ray-tracing. (Left) Rough map. (Right) Detailed map. (Center) Real observations.



As it can be seen in Fig. 13, a detailed map will bring observations more similar to „real" ones. In the above example, Gaussian noise has been added to the ray-tracing so the resulting scan contour is not as sharp as Fig. 13 left.

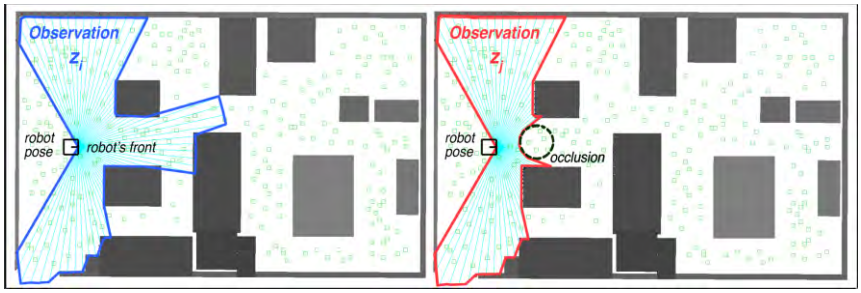
One important fact is that the small details (i.e. the high frequencies) corresponding with variations in the scans due to the presence of small objects, given the

nature of the Observation Model (OM) proposed in the next sections will have less importance in the OM than big changes. For this reason both type of metrical maps (rough and detailed) will be perfectly suitable for generating the OM. This issue will be discussed in the experiments section.

6.4 Simulating the Observations

The advantage of artificially generating the observations is that obviously is possible to generate a raw sensor vector (so called the 'supposed' observation) for any robot pose. Unfortunately, the 'real' sensor data (the observation made with the real robot at the given pose) could vary respect to that 'supposed' one, due to sensor noise and reflections. For this reason is very important adding noise to the generated observations as in [11].

Figure 14. Metric map, supposed (left) and occluded (right) simulated range scans (cyan).



Also, when simulating observations it is very important considering more random locations than states in the model (10 times or more), because each discrete state will comprise a region of poses and headings surrounding it. The Observation Model (OM) must consider as many different observations as possible in every region in order to relate states with real observations (environment dependency).

6.5 Lowering the Dimensionality

The next step consists in lowering the dimensionality of the artificially generated observation vectors, in order to simplify the OM construction. The idea is to build the observation-location relationship from quantized observations and quantized locations (states) instead of using the raw vectors.

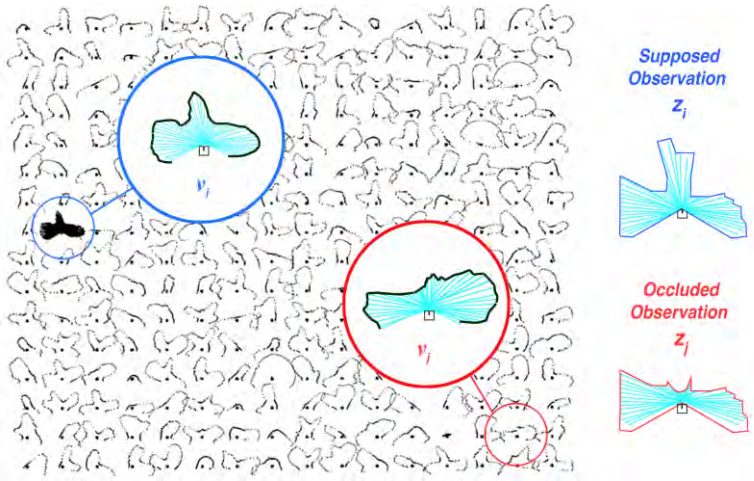
One important issue is that by quantizing the observations, a type vector index will substitute the whole multidimensional sensor observation, so every range scan will be labeled with a scalar (or a couple of scalars if a 2D VQ map is used). In the case of Robot

Localization this could imply gathering information from multiple scenarios before performing the VQ. Nevertheless, in this work we propose performing VQ exclusively using simulated observations for every evaluated scenario (this implies repeating the VQ for a new scenario), in order to avoid having type vectors whose probability of occur will be always practically zero thus saving memory and processing time. In further works we will perform VQ with simulated scans coming from many scenarios, comparing the results.

6.5.1 Vector Quantization ANNs

If a standard VQN [34] is used to obtain a set of observation vectors (to compute the Observation Model), the result will be similar to the one shown in Figure 15.

Figure 15. 256-units standard VQN and quantized observations of Fig. 14.



It can be seen that there is no similarity between consecutive vectors in the 2D VQN array, both in columns and rows (*r* and *c* indexes). Although this quantization of observations will solve the task, in case of having occlusions or large reading noise (like the one presented by black polished objects for laser scanners or soft materials in the case of sonar range finders), two found type vectors v_i:(r_i, c_i) and v_j:(r_j, c_j) (with and without noise respectively, see Fig. 15) would become very distant in the 2D VQ array (Fig. 8). As $p(z_t = s_k \mid x_i)$ must be calculated for building the OM, the distance $|r_j\text{-}r_i, c_j\text{-}c_i|$ between real and noisy observations in the VQ array would complicate the process of establishing an *a priori* observation probability for a given location x_i, because the actual observed symbol indexes (r_s,c_s), measured with real sensors, could vary substantially in the

presence of sensor noise or occlusions respect to the supposed observations stored in the OM. This distance between noisy and noise-free observations in the VQ array can be lowered with the help of a *neighborhood* relationship, facilitating the process of including noise and occlusions tolerance in the OM.

6.5.2 Vector Quantization with Self Organized Maps

Self Organized Maps (SOMs) [34] are a special kind of VQNs where a neighborhood relationship is defined. Like in a standard VQN, in a SOM the neuron whose weights are the closest to an input pattern is selected, but not only its weight is approached to the input vector during training, also are the weights of the neighbors in a lower proportion, according to

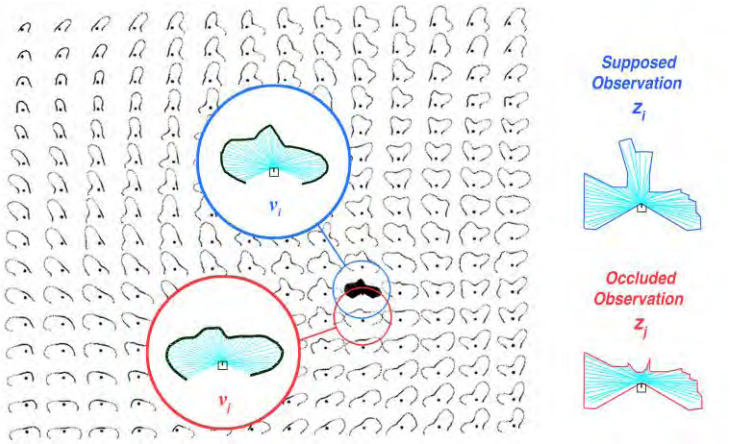
(57)

where $h(i\text{-}g)$ is a neighborhood update function in the SOM array, calculated based on the index separation $d=i-g$, as

$$h(d) = e^{\frac{-d^2}{\sigma_{SOM}^2}} \tag{58}$$

In this way, SOMs can organize the observations space in a more correlated form. When training a 16 × 16 units SOM with the samples described in Section 6.1, similar type vectors will be closer in the map (Figure 16).

Figure 16. 16-by-16 Kohonen SOM.



6.6 Building a Tolerant Observation Model

Once the SOM has been built, the next step will be identifying the Observation Model, by simulating the corresponding observations for locations x_i.

Every simulated sensor reading $z_i^s = S(x_i)$ is fed to the VQN V and a corresponding symbol $s_i^s = V(z_i^s)$ is obtained as the closest SOM vector, then a probability of observation can be established, based on a neighborhood relationship as

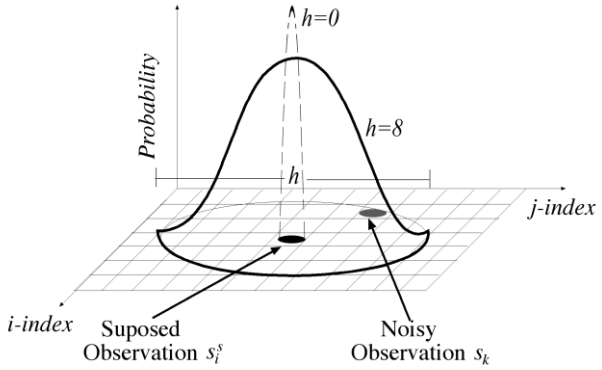
$$G(s_k, s_i) = e^{-\frac{\left[\gamma(s_i^s) - \gamma(s_k)\right]^2}{2\sigma_{tol}^2}} \qquad (59)$$

where $\gamma(s_i^s)$ and $\gamma(s_k)$ calculate the 2D indexes (i_i^s, j_i^s) and (i_k, j_k) respectively into the SOM, corresponding to symbols s_i^s and s_k (Fig. 16), while σ_{tol} (or the 'tolerance') can be set in experimental form, depending on the probability of having occlusions and sensor errors (as the case of populated environments). This parameter gives the OM the property of 'tolerating' some observation symbol variations, mainly produced by occlusions and sensor reflections. The formula

$$p(s_k \mid x_i) = \frac{G(s_k, V(S(x_i)))}{\sum_{n=1}^{m} G(s_n, V(S(x_i)))} \qquad (60)$$

calculates the Tolerant Observation Model probabilities, with S as a relationship between location and observations (a „simulator" or „sampler") and V gives the closest SOM vector s_i^s to the supposed observation z_i^s, given by the sampler.

Figure 17. Observation PDF based on SOM neighborhood.



The normalization in Eq. (60) avoids the need of an extra *k* parameter in Eq. (59) to ensure its integral equals to 1. Finally,

$$b_j = p(s_k | x_j) \tag{61}$$

can be calculated with Eq. (60) for building the observation probability matrix **B** in order to initialize the Tolerant Observation Model (TOM). By having a neighborhood of size *h=0* (only the current symbol s_i^s probability is updated) a non-tolerant observation model is obtained (Fig. 17).

Figure 18 shows the final Tolerant Observation Model construction pseudocode.

Figure 18. TOM construction pseudocode.

```
---------------------------------- initialization ------------------------------
1- Get the set of randomly sampled observations Ω : (x_i, z_i)
2- Set all B(i, j) to zero
3- For i=1 to number of nodes
        3.1 Set accumulator g(i) to zero
---------------------------------- quantization --------------------------------
4- For every sample ω_k ∈ Ω do
        4.1- Get the corresponding state index n_k by quantizing x_k
        4.2- Get the corresponding symbol s_k by quantizing z_k
---------------------------------- valuation ----------------------------------
        4.3- For all s_i in the neighborhood of s_k of size h
                4.3.1- Calculate p_ki = G(s_k , s_i) with Eq. (59)
                4.3.2- Increment index B(n_k , s_i) with p_ki
                4.3.3- Increment accumulated sum g(n_k) with p_ki
---------------------------------- normalization ------------------------------
5- For i = 1 to the number of states
        5.1 For j = 1 to the number of symbols
                5.1.1 Set B(i , j) = B(i , j) / g(i )
```

7. Path Reconstruction

As we are using a probabilistic method for pose and each node has a probability $Bel_t(i) = p(x_i \mid z_{0:t}, u_{1:t})$ assigned to it after the update phase, then, by using the expected value formula, a continuous robot location $\boldsymbol{x}_t$ can be computed from discrete samples (nodes)

with

$$\boldsymbol{x}_t = \sum_{i=1}^{n} [\boldsymbol{x}_i \; p(x_i \mid z_{0:t}, u_{0:t})] = \sum_{i=1}^{n} [\boldsymbol{x}_i \; \Phi'(i)] \tag{62}$$

after each iteration, being possible to build a path with every successive location calculated with Eq. (62).

8. Optimizations

The motion estimation proposed in this work has the time complexity of $O(n^2)$, with n as the number of discrete states in the model, because in each iteration the transition probabilities between every state x_i and the remaining states x_j must be calculated (total probability). Some assumptions can be done in order to reduce drastically the number of required operations without loosing generality and precision (this is especially useful when running the algorithm in standard non-parallel processors):

8.1 Node Location Selection

Because transitions between nodes $x_i:(x_{w_i}, y_{w_i}, \theta_{w_i})$ and $x_j:(x_{w_j}, y_{w_j}, \theta_{w_j})$ must be calculated in each iteration it results much simpler partitioning the free space with a 2D vector quantization network, in order to get the node's xy location. Once calculated, a group of nodes can share the same (x_{w_i}, y_{w_i}), only differing on their θ_{w_i} coordinates, starting from zero and with a heading at predefined regular intervals (see section 6.1). This can help to use one single distance estimation $(x_{w_j} - x_{w_i})$ and $(y_{w_j} - y_{w_i})$ in several transition updates.

8.2 Update Triggering

The belief update is calculated with $\boldsymbol{\Phi}_{t-1}\mathbf{A} \otimes \mathbf{B}\, v^{\mathrm{T}}$, but in general only a reduced number of pose beliefs in the $\boldsymbol{\Phi}_{t-1}$ vector will have a significant value. In this form, all the transition and observation calculations can be performed only for those $\boldsymbol{\Phi}_{t-1}(i) \geq \varepsilon$. If nested loops are used for calculating the a_{ij} transitions, such transitions can be considered as almost zero if $\boldsymbol{\Phi}_{t-1}(i) < \varepsilon$ (or a minimum probability value near to zero, in order to not discard any state transition in future computations). This optimization has the important role of

selecting which states (and state-transitions) must be evaluated during the pose belief update, acting as a trigger of GL after a RK as follows: if the robot is performing PT (the current robot pose is known) some $\Phi_0(i)$ will tend to one while the rest will tend to zero, due to the normalization after the observation update in Eq. (40). After a RK, $\Phi_t(i)$ will rapidly tend to a uniformly distributed probability value (due to the difference between real and supposed observations (as the robot is not informed about the displacement) equals to 1.0/*number_of_states*. If ε is set just below this uniformly distributed probability value then the method will trigger all the state-transitions evaluation after the RK (when $\Phi_t(i)$ reaches the uniform probability).

As soon as some $\Phi_t(i)$ increase their probability, the rest will continue lowering their values below ε, discarding them from future computations. This will be equivalent to performing GL with an unknown initial pose.

In a similar way, if the initial robot pose is unknown or a GL command is issued to the robot, it will be enough to set every $\Phi_t(i)$ equals to 1.0/*number_of_states*, forcing the method to reevaluate all state transitions (the operation of the update triggering will be demonstrated in the results section).

8.3 Observation Update Nesting

The main Viterbi update is calculated through Eq. (26) that is equivalent to

$$B(x_j)_t = \sum_{i=1}^{n} \left[p(x_j \mid x_i, u_t) B(x_i)_{t-1} b_j(z_t) \right] \tag{63}$$

By nesting the observation update into the transition update, only a couple of nested loops can be used to compute the belief update. Then, they can be triggered together by ε (Fig. 19).

Figure 19. Observation update nested in the transition evaluation cycle (pseudocode).

1. For i = 1 to number of states
  If $\Phi_{t-1}(i) \geq \varepsilon$ *then*
    1.1 Set $\alpha_{acc} = 0$
    1.2 Calculate $\mathbf{x'}_i$ *with Eq. (49)-(52)*
    1.3 For j = 1 to number of states
      1.3.1 Calculate $\mathbf{A}(i, j) = D(\mathbf{x'}_i, \mathbf{x'}_j)$ *with Eq. (54)*
      1.3.2 Increment α_{acc} *with* $\mathbf{A}(i, j)$
    1.4 For j = 1 to number of states
      1.4.1 Increment $\Phi_t(j)$ *with* $\mathbf{B}(j, s_t)\ \Phi_{t-1}(i)\ \mathbf{A}(i, j) / \alpha_{acc}$

This observation update nesting allows using a single vector array $\mathbf{A}(j)$ in each iteration, with $1 \leq j \leq$ *number of states*, instead of a whole transition square matrix $\mathbf{A}(i,j)$. The same loop (Fig. 18, step 1) is being used both for the calculation of the transition probabilities for the current state i to every state j and for the calculation of Eq. (45) so there is no need to store the transition probability values for previous states i, thus saving much memory space. According to this, it is possible to substitute the $\mathbf{A}(i,j)$ term in steps 1.3.1, 1.3.2 and 1.4 by $\mathbf{A}(j)$.

8.4 Memory Usage

With all the abovementioned optimizations, if *nodes* is the number of (x_i,y_i) quantized locations (see 8.1), then *nodes*×2 is the required memory for storing them. If *headings* is the number of absolute state headings (see 6.1), then *states* = *nodes* × *headings* is the number of states in the OVL and *states*×2 is the memory required for storing $\mathbf{\Phi}_t$ and $\mathbf{\Phi}_{t-1}$. If *symbols* is the number of neurons in the SOM (see 6.2) and *dimobs* is the dimensionality of the raw observation vector z_t, then *symbols*×*dimobs* is the memory necessary to store the SOM weights, *states* is the memory required to store the **A** vector and *states* × *symbols* is the memory required for storing the **B** matrix. Then the total memory usage will be:

$$Memory = nodes \times (2 + headings \times (symbols + 3)) + symbols \times dimobs \qquad (64)$$

9. Odometry-based Viterbi Localization Algorithm

In this section we present the final Odometry-based Viterbi Localization (OVL) Algorithm, shown in Figure 20.

Figure 20. Optimized OVL pseudocode for non-parallel microprocessors.

---------- initialization ----------

1.- Set $\xi = 1 \times 10^{-10}$, $\varepsilon = 1/(number\_of\_states) - \xi$, $\alpha_{acc} = 0$,

$\sigma_d = 2/3 \times avg\_node\_separation$, $\sigma\theta = \Pi/(2 \times num\_of\_absolute\_headings)$

3- For i = 1 to number_of_states

3.1- If Initial Position $\mathbf{x}_0$ *is known (Position Tracking)*

3.1.1- Set $\boldsymbol{\Phi}_0(i) = \mathbf{D}(\mathbf{x}_0, \mathbf{x}_i)\, \mathbf{B}(i, s_0) + \xi$ *with Eq. (54)*

3.2- else $\boldsymbol{\Phi}_0(i) = \varepsilon\, \mathbf{B}(i, s_0) + \xi$ *(Global Localization)*

3.3- Increment α_{acc} *with* $\boldsymbol{\Phi}_0(i)$

4- For i = 1 to number_of_states Set $\boldsymbol{\Phi}_0(i) = \boldsymbol{\Phi}_0(i) / \alpha_{acc}$

---------- recursion ----------

5- For t =1 to the number of observations

5.1- Get robot odometer estimation u_t *:*$(d_t, \phi_t, \Delta\theta_t)$

5.2- Get current symbol s_t *by quantizing* z_t

5.3- For i = 1 to number of states, Set $\boldsymbol{\Phi}_t(i) = \xi$

*---------- **A** matrix calculation ----------*

5.4- For i = 1 to number of states

If $\boldsymbol{\Phi}_{t-1}(i) \geq \varepsilon$ *then*

5.4.1- Set $\alpha_{acc} = 0$

5.4.2- Calculate $\mathbf{x'}_i$ *with Eq. (49)-(52)*

5.4.3- For j = 1 to number of states (or those in the neighborhood of $\mathbf{x'}_i$*)*

5.4.3.1- Calculate $\mathbf{A}(j) = D(\mathbf{x'}_i, \mathbf{x'}_j)$ *with Eq. (54)*

5.4.3.2- Increment α_{acc} *with* $\mathbf{A}(j)$

5.4.4- For j = 1 to number of states

Increment $\boldsymbol{\Phi}_t(j)$ *with* $\mathbf{B}(j, s_t)\, \boldsymbol{\Phi}_{t-1}(i)\, \mathbf{A}(j) / \alpha_{acc}$

---------- normalization ----------

5.5- Set $\alpha_{acc} = 0$

5.6- For i = 1 to number of states, Increment α_{acc} *with* $\boldsymbol{\Phi}_t(i)$

5.7- For i = 1 to number of states, Set $\boldsymbol{\Phi}_t(i) = \boldsymbol{\Phi}_t(i) / \alpha_{acc}$

---------- pose estimation ----------

5.8- Set $\mathbf{x}_t = (0,0,0)$

5.9- For i = 1 to number of states, Increment $\mathbf{x}_t$ *with* $\mathbf{x}_t\, \boldsymbol{\Phi}_t(i)$

5.10- Consider $\mathbf{x}_t$ *as the current robot pose*

9.1 Initialization

If the initial pose (x_0, y_0 θ_0) is known, Eq. (54) can be used to evaluate the initial pose probabilities Π_i (as it evaluates the probability for any x" to correspond with every state x_i):

$$\Pi_i = \quad (65)$$

By the other hand, if the initial pose is unknown, Π_i can be set to a uniformly distributed PDF, this is

$$. \quad (66)$$

forcing the method to evaluate all possible state transitions depending on u_1.

9.2 Recursion

9.2.1 Transition Probability Estimation

This section is the core of the Odometry-Based Viterbi Algorithm. First, steps 5.1 and 5.2 obtain the odometer estimation u_t and compute the current symbol s_t by quantizing the current observation z_t. Step 5.4.1 cleans the accumulator α_{acc} that serves to store the sum of state-transition probabilities. Step 5.4.2 calculates the new location x_i' given the state location x_i and u_t. Step 5.4.3 computes the corresponding state-transition probability $p(x_j \mid x_i, u_t)$ and stores that value in the temporal accumulator $\mathbf{A}(j)$.

Step 5.4.4 increments the state probability $\Phi_t(j)$ (already initialized with a value close to zero in step 5.3) with the probability of observation $p(z_t \mid x_j)$ multiplied by the previous state probability $\Phi_{t-1}(j)$, the state-transition probability $p(x_j \mid x_i, u_t)$ already stored as $\mathbf{A}(j)$ and divided by the probability sum α_{acc}. In this way, in a single loop, every state probability $\Phi_t(j)$ is incremented with the individual contribution of a transition to x_i from every x_j and normalized by α_{acc}.

Below step 5.4, by using a trigger value ε, only the overall contribution to $\Phi_t(j)$ of those states whose probability in the previous iteration $\Phi_{t-1}(j)$ was above the trigger value will be evaluated (update triggering). This important feature allows optimized-OVL to

save time, memory and managing GL, PT and RK (this will be discussed in the results section).

9.2.2 Normalization

Steps 5.5 to 5.7 compute Eq. (27) to obtain $p(x_i \mid z_{0:t})$. This normalization ensures

$$\sum_{i=1}^{n} p(x_i \mid z_{0:t}) = 1$$

9.2.3 Pose Estimation

Steps 5.8 to 5.10 compute the current robot pose with Eq. (62). Although this equation calculates only one localization hypothesis based on the expected value formula, OVL is always carrying-on a multi-hypothesis PDF through the state probability vector Φ_t. Other methods thet can be used for estimating multiple hypotheses are Clustering Artificial Neural Networks (CANNs) [20] and Radial Basis Functions (RBFs) [21]. By isolating a group of states with high-probability as a cluster, they can represent multiple localization hypotheses. When having only one cluster it would represent the true robot pose.

The advantage of using the expected value formula in Eq. (62) is that it can be also used for computing the average pose value for a group of states (those in the influence ratio of every found cluster).

9.3 Termination

As a difference with classical Viterbi Localization [13], after every update the robot pose is calculated with Eq. (62) so there is no need of backtracking (i.e. searching for the most probable individual state after every observation update). The proposed approach in fact calculates a continuous pose after every observation update, this is an important improvement against [13].

10. Experiments

10.1 Physical Robot

The robot ARTUR-ito (Autonomous Ready-To-Use Robot, Fig. 21) is a differential drive robot equipped with two computers: an Apple Mac Mini® with an Intel Core 2 Duo processor at 2.4 Ghz running Mac OS X Leopard, and a Javelin Wedge® Point-Of-Sales (POS) touchscreen computer with a Pentium MMX processor at 233Mhz, running Windows 98. The robot has on-board stereo vision and Wi-Fi communications.

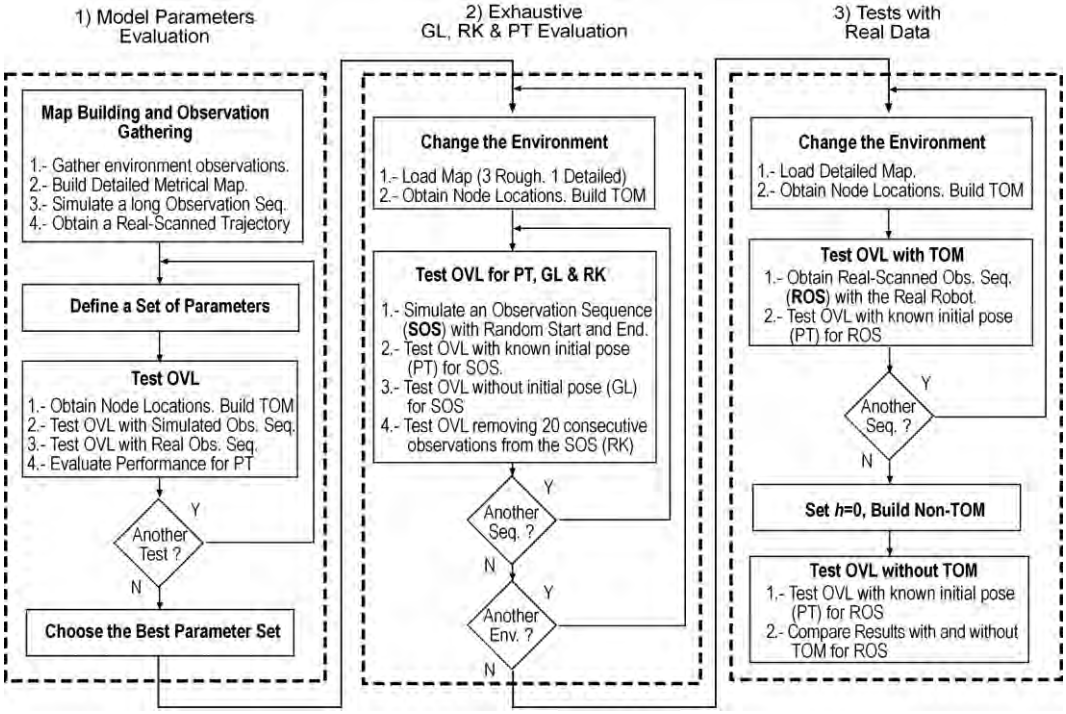
Figure 21. Robot ARTUR-ito

ARTUR-ito is equipped with a Hokuyo URG-04LX laser scanner, with a sensing range from 20 mm to 4 m and 240 degrees of field-of-view. Although this laser is able to provide almost 700 readings per scan. It was decided to use only 68 readings while conserving the maximum field-of-view. This is, one ray scan at every 3.53 degrees (from -120 to 120 degrees) conformed one 68-scans laser observation. The laser is situated at 80 cm high to the floor and the laser-scanning plane remains parallel to the floor (Fig. 21).

10.2 Experiments

Several experiments were conducted to evaluate the OVL performance in both real and simulated environments. The diagram in Figure 22 shows the experiment sequence. The initial task was defining the best set of parameters for OVL. Then an exhaustive evaluation of OVL solving PT, GL and RK both in detailed and rough maps was performed. Finally, tests with real data with and without a TOM were conducted to evaluate the aid of such kind of pose-observation relationships.

Figure 22. Experiment sequence for evaluating optimized-OVL.



10.3 Model Parameters Evaluation

10.3.1 Map Construction

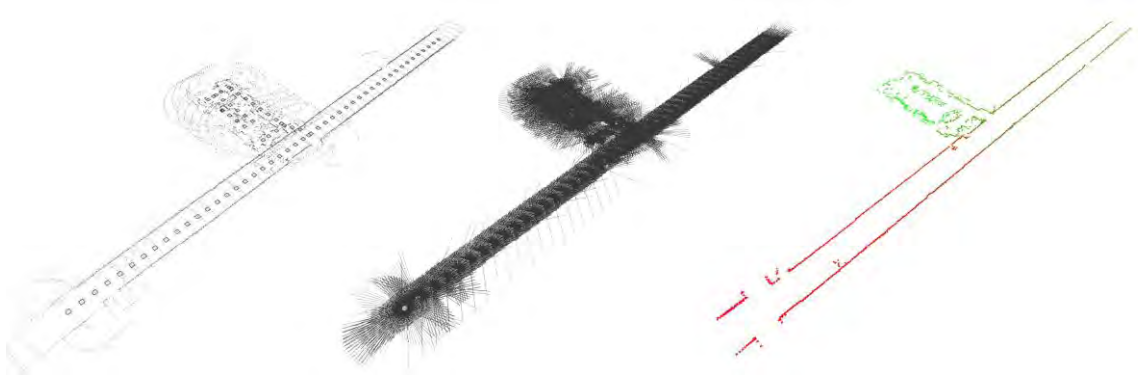
First, a detailed metrical map representation was built by manually displacing the real robot in the environment and gathering laser range readings. The real environment is the Biorobotics Laboratory of the National Autonomous University of Mexico UNAM and a long corridor (Fig. 23).

Figure 23. Bio-robotics laboratory and long corridor.

In order to minimize the initial pose error when building the map, the floor tiling (at every 30 cm) was used to set the robot at every 2.10 meters ± 3 centimeters, performing four scans at the absolute headings of zero, 90, 180 and 270 degrees at every robot stop (Fig. 24, left). This process took about 2 hours for scanning the 50 meters long corridor and the 5 × 7 m Biorobotics Laboratory.

With the set of gathered observations a metric map representation was built (Fig. 24, right) following the procedure described in section 6.3.

Figure 24. Detailed environment. Robot stops (left). Actual scans (center). Final map (right).

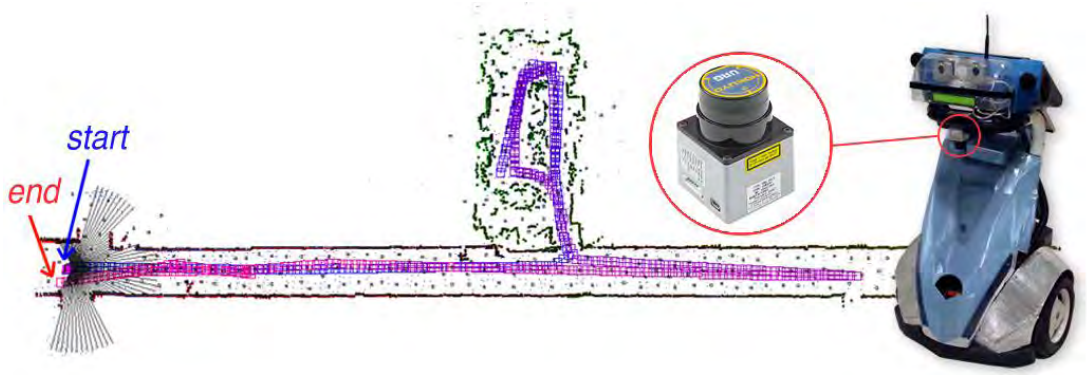


10.3.2 Best OVL Parameter Set

Five OVL parameters were evaluated to test their contribution to the overall OVL performance: a) the number of *xy* discrete nodes (see section 5.1), b) the number of absolute headings (see section 5.2), c) SOM size in neurons (see section 6.5.2), d) TOM"s neighborhood *h* (see section 6.6) and e) samples per state for building the TOM state-observation relationship (see section 6.4).

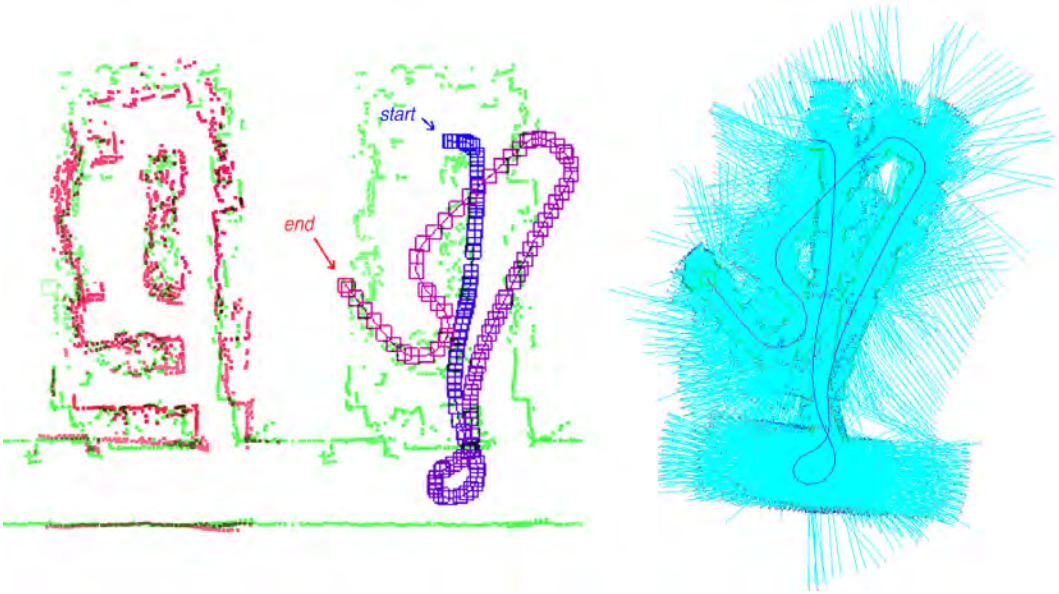
The procedure described from Section 5.1 to Section 6.2 was followed for building the set of states, training the SOM and building the TOM by artificially generating supposed observations at the navigable locations. Then, eight optimized-OVL PT trials with the same trajectory (a 258 meters path with 314 observations, see Fig. 25), one observation simulated roughly at every 80 cm in the detailed environment (Fig. 24) were run to determine the best set of parameters.

Figure 25. ARTUR-ito and 258 meters path used for the simulated tests.



After each simulated test, another OVL run with the same set of parameters but with a real observation sequence (Fig. 26) was run to compare the simulated OVL performance under real observation conditions, as the true robot pose in the real environment cannot be precisely evaluated under continuous robot motions. In order to introduce noise and occlusions some objects like chairs, lockers, boxes and small furnitures were moved (Fig. 26 left, red squares) from their original positions (Fig. 26 left, green squares).

Figure 26. Object Motions (Left). Odometry Estimation (Center). Sensor Readings (Right).



10.3.3 OVL Performance Evaluation

An average location error was calculated between the true robot pose and the one calculated with OVL for the set of 314 simulated observations as

$$E_{\theta} = \frac{1}{n}\sum_{i=1}^{n} |\theta_i - \theta c_i| \tag{67}$$

$$E_{xy} = \frac{1}{n}\sum_{i=1}^{n} \sqrt{(xr_i - xc_i)^2 + (yr_i - yc_i)^2} \tag{68}$$

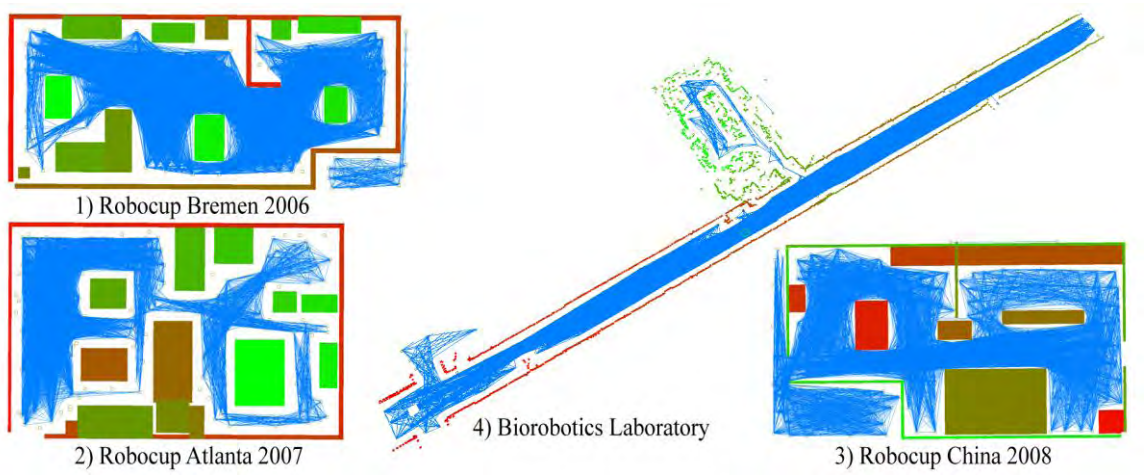
where n is the number of states in Φ_t, $(xr_i, yr_i, \theta r_i)$ is the actual robot pose at every step of the 314 observation"s trajectory, while $(xc_i, yc_i, \theta c_i)$ is the pose computed with the OVL algorithm with Eq. (62).

10.4 Exhaustive OVL Evaluation for PT, GL and RK.

With the set of parameters that provided the best trade-off between location error, processing time and noise tolerance, a test of 10 trials at random starting and ending locations in four different simulated scenarios was run to get the overall performance for the PT, GL and RK sub-problems (120 trials in total). The first, second and third

simulated environments correspond to robocup@home 2006, 2007 and 2008 competition scenarios. The fourth environment is the detailed map of the Biorobotics Laboratory (Fig. 27).

Figure 27. Maps used in the tests. The connectivity network is shown.



10.5 Tests with Real Data with and without a TOM

Ten trials were conducted to test the OVL performance with real data variations by randomly choosing a start and ending poses into the real environment (Fig. 23, Left) with the environment differences depicted in Fig. 26 left (red squares). Then, the experiment was repeated removing the Tolerant Observation Model, in order to prove the aid of the tolerance to sensor noise and occlusions.

10.6 Comparison against a State-of-the-Art Algorithm

A comparison of optimized-OVL vs. classical Monte Carlo Localization (MCL) [11] was performed. By using the ray-tracing simulator with the detailed environment, it was possible to simulate a „precise" supposed observation for MCL in real-time.

10.5 Method Performance on Small Processors

Finally, optimized-OVL was run in real-time on a Pentium MMX 133Mhz processor with 64 MB of RAM, incorporated in the robot ARTUR-ito (Fig. 21).

11. Results

11.1 Model Parameters Evaluation

Table 1. OVL PT Parameter tests for a 314 observations sequence.

#	*xy* nodes	Headings	SOM Symbols	States	Samples per State	Total OM Samples	*h*	Avg. $\|xy\ error\|$ Sim. (m)	Avg. $\|\theta\ error\|$ Sim. (m)	Total Processing Time (s)	Performance with real data.
1	256	16	256	4096	100	409600	8	0.33	0.06	6.16	Best, tolerates env. changes and errors
2	256	36	256	9216	10	92160	8	0.40	0.11	31.82	Not bad, just tolerates reflections
3	256	16	256	4096	10	40960	8	0.34	0.06	6.16	Not good, non tolerant to big changes
4	256	16	256	4096	20	81920	8	0.34	0.13	6.16	Not bad, just tolerates reflections
5	256	36	256	18432	10	184320	11	0.21	0.06	117.91	Good, tolerates most errors
6	64	36	256	4608	10	46080	6	0.11	0.06	6.7	Bad, fails on real environments
7	144	16	64	4096	20	81920	4	0.34	0.12	9.51	Good, tolerates most errors
8	256	16	144	4096	50	204800	6	0.30	0.08	8.26	Not good, non tolerant to big changes

The first column shows the number of experiment, then the number of absolute *xy* nodes (number of Vector Quantization neurons) is shown in column 2, followed by the number of absolute headings in the range of 2..2Π (3rd column).

The 4th column indicates the number of discrete symbols (neurons) used in the Self Organized Map while the 5th columns computes the total number of states in $\mathbf{\Phi}_t$ as the number of *xy* nodes multiplied by the number of absolute headings.

The 6th column indicates the number of randomly simulated observations per state. The 7th calculates the total simulated samples used for training the SOM and building the TOM as the number of states (5th column) multiplied by the number of simulated samples per state (6th column).

In the 8th column the *h* parameter -the „tolerance"- of the TOM is shown. *h*/2 is the neighborhood influence ratio of the Gaussian function that computes $p(z_t|x_t)$ with Eq.

(61). The Gaussian influence neighborhood function was estimated based on the work of Liu and Haralick [35] that helps to calculate σ^2_{tol} in Eq. (59), based on a Gaussian kernel of size *h*. They applied their theories in building Gaussian kernels for image processing.

Columns 9 and 10 show the average absolute location and heading errors respectively, measured with Eq. (67) and (68).

Column 11 computes the total processing time the OVL algorithm requires for calculating the 314 observations trajectory. It can be noticed how the number of states affects the method"s processing time.

Finally, the 12th column shows a qualitative analysis of the OVL performance with the real-data sequence of Fig. 26, analyzing the tolerance to environment changes, update time and sensor errors.

11.2 Best Parameter Set

Simulated (Table 1, columns 9 and 10) and real experiments (Table 1, column 12) gave the best results (a trade-off between speed and location error) with 256 *xy* discrete locations and 16 absolute headings, starting from 0 and up to 337.5 degrees in 22.5-degrees intervals (4096 states) for an average node separation of 0.68m. 10,000 simulated observations were used to train the 16×16 SOM and almost 410,000 for building a TOM (a minimum of 50 samples per state was experimentally determined and 100 as the optimum). This means at least one sample per dm^2 per absolute heading.

σ_{tol} was set to $0.1092 \times h/2 + 0.4335$ based on [35], this is 0.87 for h=8, while $\sigma_d = 2/3 \times avg\_node\_separation = 0.454\ m$ and $\sigma_\theta = \Pi/(2 \times headings) = \Pi/32$ was the best parameter set.

$\varepsilon = 1/Number\_of\_\ States - 1\times10^{-10}$ was found as the best trigger value (a uniformly distributed PDF) while consecutive observations z_t and displacements u_t should be taken

when the robot displacement surpasses the average *xy* node separation or when the robot"s heading change is above 2Π/*headings*, otherwise the method tends to remain in the same state or in a small state neighborhood.

11.3 Exhaustive OVL Evaluation for PT, GL and RK.

Table 2 (rows 1-3) and Figure 28 show the performance of OVL solving PT in three simulated environments. The 4th column in Table 2 shows the average deviation from the true pose (red path in Fig. 16). The 4th row shows the simulation results in the real-scanned environment (Fig. 23).

Table 2. OVL Performance tests for PT in simulated environments.

Env.	Map Dimensions Width & Height (m)	Avg. Node Separation (m)	Absolute Avg. Location Error (m)	Absolute Avg. Heading Error (rad)
1	13×7	0.49	0.16	0.06
2	11×7	0.30	0.19	0.12
3	12×5	0.33	0.19	0.08
4	50×12	0.68	0.25	0.10

It is notorious that the OVL accuracy is mainly related with the spatial separation between discrete nodes (not in an exact linear relation) as we are approximating the true robot pose from a discrete probability distribution. Because the method accuracy depends on several parameters (five in the Table 1) and we are testing the same parameter set in many environments, it is possible for instance that environments with more different observations than others would require bigger SOM elements. In such cases it will be desirable to vary some parameters and rebuilding the models.

In this way, the OVL accuracy becomes an optimization problem that could be treated in the future with multi-objective optimization techniques like Genetic Programming. By now the goal of this article is demonstrating that a parameter set can have an acceptable performance in many environments.

Figure 28. OVL PT in three simulated Environments. Red line: real path. Blue line: found path.

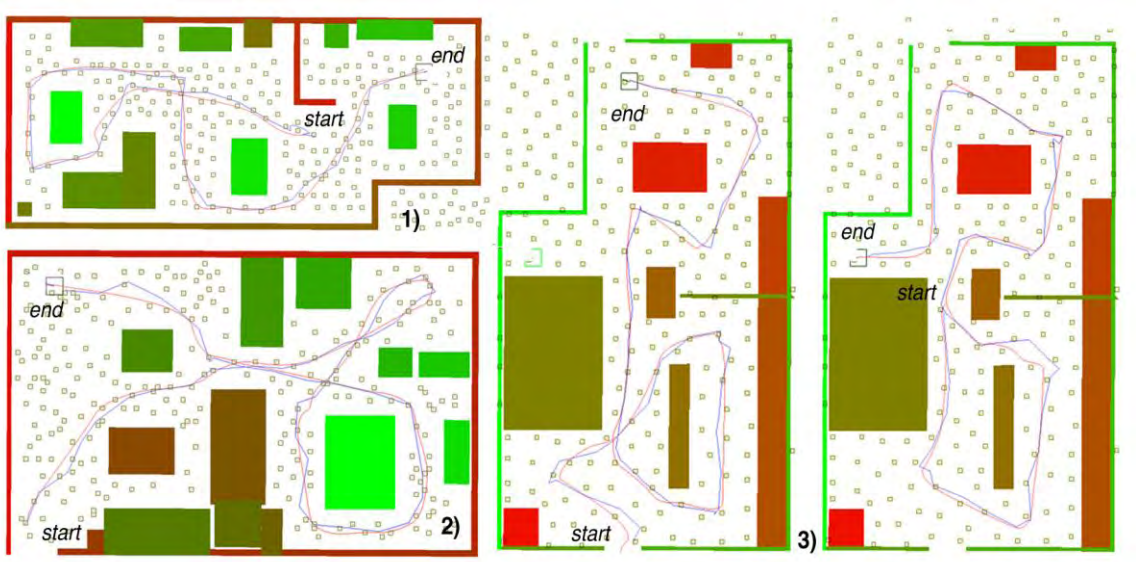


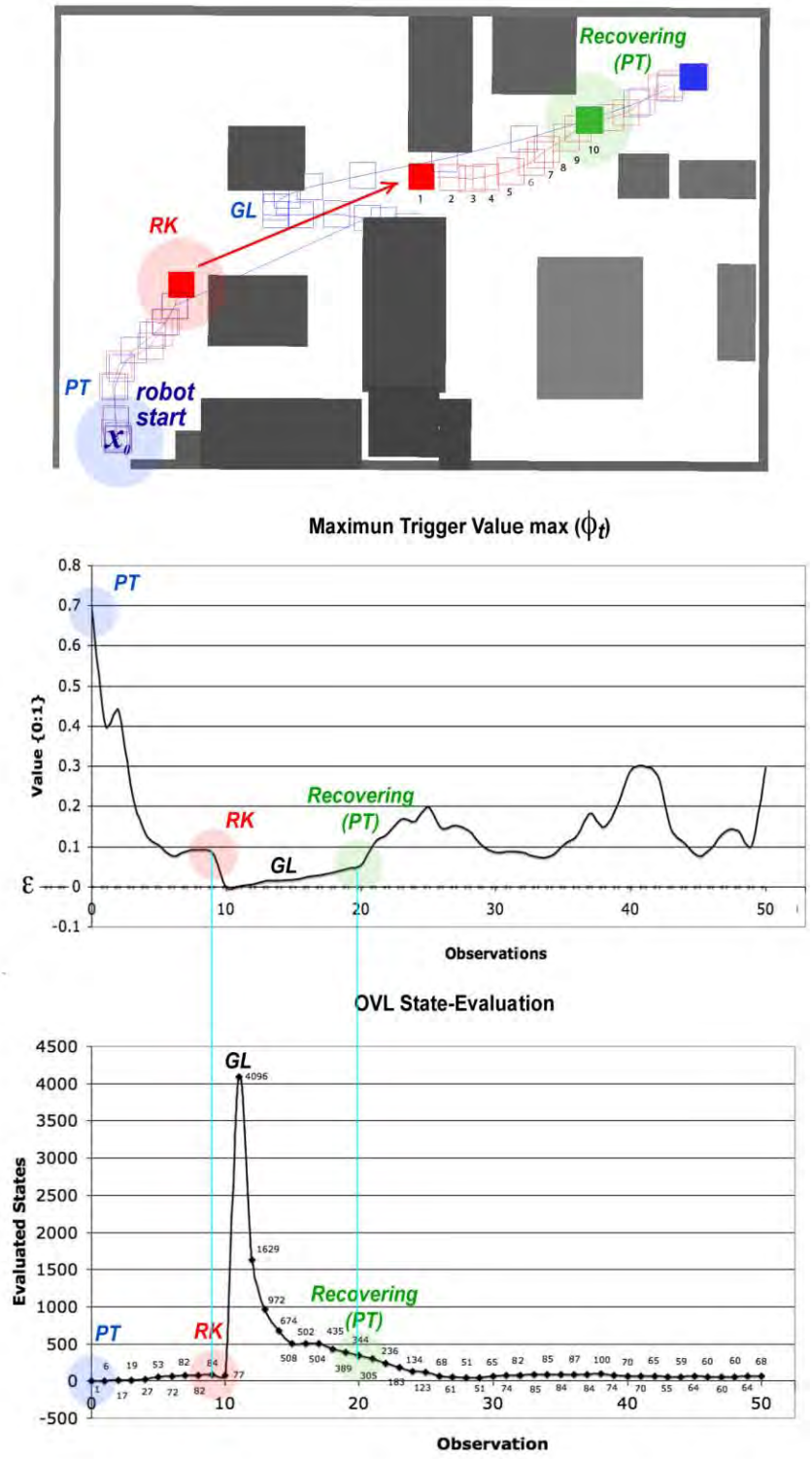
Figure 28 shows how it is possible computing a continous path from discrete samples (states). Despite the quantitative results in Table 2 show an average error above 16 centimeters, the comparison between the real motions path (red line) against the path calculated with optimized-OVL (blue line) showed that, despite in certain zones the calculated path is somewhat distant to the real one, in general OVL keeps track of the robot pose very well.

Based on the above results, it is possible to estimate that the OVL *xy* location error is about 50% of the average node separation and the angular error is also about 50% of 2Π/number_of_absolute_headings (32 in the tested parameter set). For instance, in order to have an average *xy* error of 5cms, it would be necessary at least three times more nodes (states) in the environments 1 to 3 and five times more states in the 4th environment.

In the GL tests, the method was able to estimate the true robot pose after 6 updates in average. After a RK, in only two updates the probability decay is enough to force GL to relocalize the robot in 10 updates in average.

Figure 29 demonstrates OVL over PT, RK and GL with the simulated environment number 2 for 4096 states (256 *xy* locations by 16 absolute headings).

Figure 29. OVL solving PT, RK and GL. Red path: true trajectory. Blue path: found trajectory.



In Fig. 29, the robot starts at the known location x_0 performing PT, thus the maximum trigger value max(ϕ_0) is near 1.0. After a RK, in only one update, the maximum trigger value max(ϕ_t) decays until reaching ε thus firing the GL and evaluating all the state-transitions (4096). Ten iterations later, OVL finds the true robot pose again, continuing with PT. As time passes, the number of evaluated states (trigger firing) reaches the rates before the RK (about 80 state-transitions).

It is notorious how the method can handle GL, PT and RK based both in the update triggering and the method initialization. Nevertheless, one problem arises when GL is fired when ϕ_t decays fast (as the case of having severe occlusions): the time consumed for evaluating 4096 state-transitions is considerably larger than the time needed for performing PT (4 seconds for 4096 transitions in GL against 0.02 seconds for 80 transitions in PT). Usually the Probabilistic Localization Methods as [11] limit the amount of change in the pose beliefs ϕ_t respect to ϕ_{t-1}, in order to limit the belief changes. This would help OVL to be less sensitive to observation changes.

The total amount of memory required by optimized-OVL was 1.07 MB. Compared with 16 MB required for storing only a full $\mathbf{A}(i,j)$ transition matrix for 4096 states, the saving is notorious.

11.4 Tests with Real Data with a TOM

Table 3 shows results in the environment 4 (real scanned) with real observations (Fig. 27, 4). The largest location errors were obtained in trajectories traversing a long corridor, but the method performance did not decay too much.

Table 3. OVL Performance tests for PT in a real environment.

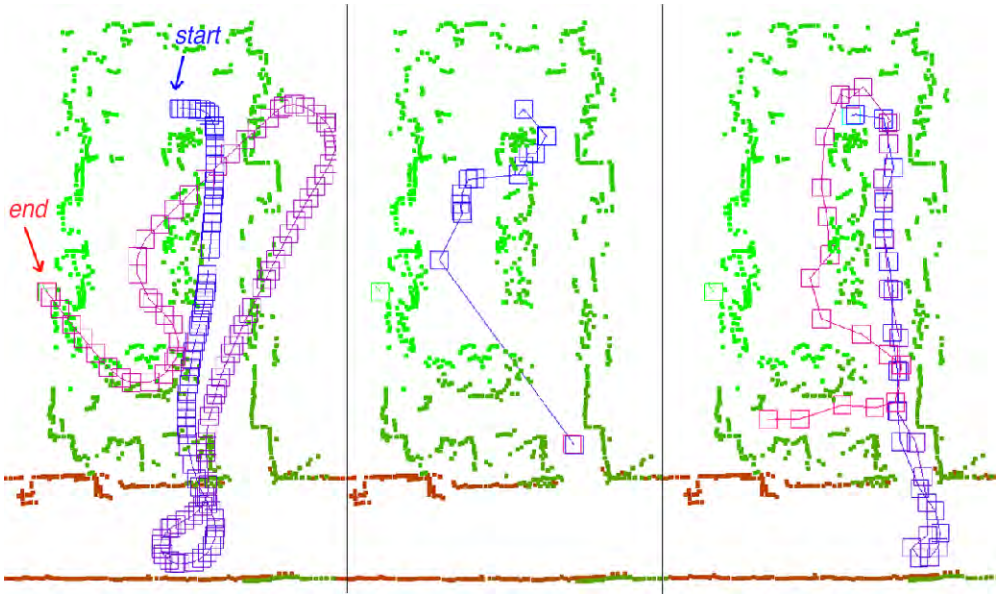
Run	Number of Observations	Accumulated Traveled Distance (m)	Absolute Avg. Location Error (m)	Absolute Heading Error (rad)
1	314	258.20	0.33	0.06
2	37	24.60	0.16	0.15
3	43	29.69	0.23	0.10
4	21	16.00	0.13	0.08
5	70	53.61	0.17	0.11
6	37	24.89	0.23	0.08
7	20	12.5	0.25	0.09
8	51	34.9	0.24	0.14
9	19	9.81	0.27	0.10
10	44	28.82	0.24	0.08

In this table, is evident that the OVL performance depends mostly on the chosen sequence path. The same isue also applies for Particle and Kalman Filters. For example, although the average node separation is the same for all the above tests (0.68) and all the tests use exactly the same TOM, the first sequence has a lower *xy* location performance (0.33) than the fourth (0.13). This is because the long first sequence crosses several times the long corridor (Fig. 25) compared with the small fourth sequence that reflects a robot motion inside de Biorobotics Lab (Fig. 26). In this sense, OVL presents the same problems than PFs and KFs: the long trajectories with very similar observations (i.e. the corridors).

11.4 Tests with Real Data without a TOM

Despite the use of a TOM has almost no effects in simulated environments, under real conditions having occlusions and sensor noise, it really helped to keep track of the true robot pose (Fig. 30). Without a TOM the robot would have easily got lost.

Figure 30. OVL PT with real observations. Left: pure odometry. Center: Non-TOM. Right: TOM.

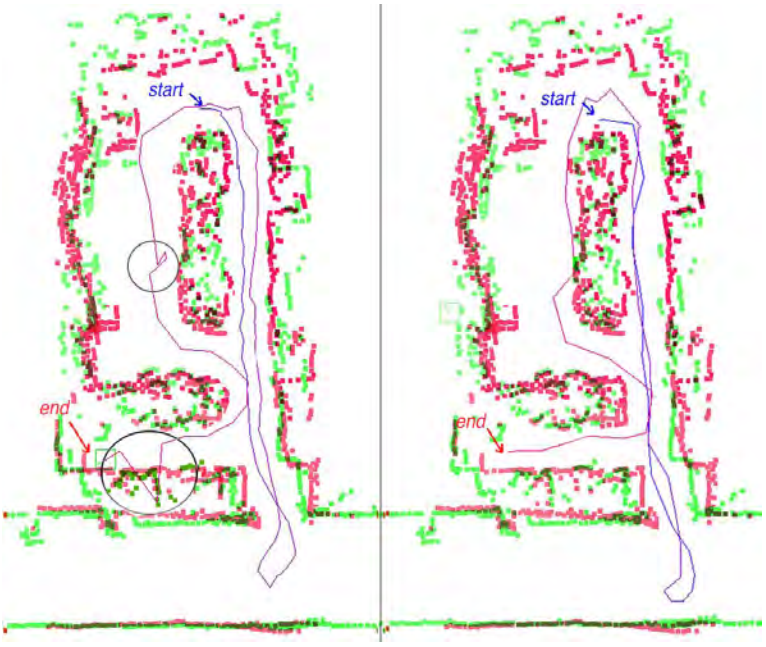


As the Tolerant Observation Model is basically a probabilistic tool for comparing high-dimensional observations, its applicability scope is not exclusive restricted to Viterbi Localization. In this way a TOM could be incorporated to those methods that require vector comparison. Even as an optimization to the Monte Carlo Particle Filter [11]. The main contribution of the Self Organized Maps is the easiness for arranging similar observations in the 2D map.

11.5 Comparison against a State-of-the-Art Algorithm

Figure 31 (left) shows a comparison of standard Monte Carlo Localization (MCL) using a real-time scan simulator and 5,000 particles against optimized-OVL (Fig. 31, right), for the same trajectory inside the Bio-robotics lab (Fig. 31, left) using the Hokuyo URG-04LX laser scanner with a 68-scans observation. MCL fails on two locations (circled) due to high changes in the environment (red squares) compared with the original mapped environment (green squares). The time spent by MCL per update was 5 seconds against 0.02 seconds of OVL. In the case of GL, about 10,000 particles are needed by MCL to correctly find the true robot pose (consuming 10 seconds per update). Even in the worst case, the 4 seconds needed by OVL for reevaluating 4096 states for GL results smaller.

Figure 31. Comparison between standard MCL (left) and OVL (right) on changing environments.



After comparing the performance of OVL against classical Monte Carlo Localization one question arises:

What would happen if a PF is used with similar intermediate simulated observation and motion process, would it be improved in efficiency as well?

The answer to this question is related with the way that both methods operate: the PFs efficiency rely in the precise prediction both of the next robot location after a displacement (by means of a resampling) and the „supposed" observation for that new location (by means of a simulator). In this way, PFs performance basically depends on „guessing" an appropriate location based on motions and having enough noise in the sensor to approximate the simulated observation to the real laser scan. In OVL the set of samples (the *states*) remains constant during all the algorithm execution and the efficiency depends mostly on the physical separation among those states and the relationship between locations and observations stored in the OM (off-line built).

For this reason, the PFs performance would decay by having a fixed location resampling set. Despite this, the Observation Model in PFs could improve its efficiency because a SOM allows naturally correlating and arranging observations into a „map", and the TOM stores that relationship. By searching for the closest state location stored in a TOM, PFs could estimate the observation for a given pose and compare it against the actual observation in a faster way than simulating the observation for a relatively high number of locations (the *samples*).

In any case, the weakness of most Probabilistic Localization approaches is the estimation of the next robot pose with the Motion Model (MM). If the MM does not represent with precision the way the mobile robot pose changes based on odometry, in a very few steps PFs and KFs can easily loose track of the robot. As OVL has a fixed set of „samples" represented by the fixed state locations, results less sensitive to imprecise motion estimations.

According to this, OVL can be suitable of being implemented in mobile robots whose motions are difficult to predict (as the case when the robot does not have odometers). Good examples of this are legged robots, nano-robots, and applications that calculate the pose based on accelerometers or by matching observations without odometry.

Finally, it is appropriate to mention that the performance of all Probabilistic Localization methods is mainly driven by the similarity between observations in different locations of the map. Similarities in the observations will therefore induce a strong uncertainty in the final robot pose.

11.6 Method Performance Evaluation

The average time per iteration, running the optimized version of OVL was 0.02s in an Intel Core 2 Duo processor, enough to run in real-time. Without the optimization, a full update took about 4 seconds.

The optimized-OVL tests running on a Pentium 133 Mhz computer of the ARTUR-ito robot were able to locate the robot with an average update time of 0.25 seconds. As the maximum robot speed is limited to 1 m/s, this update time is enough for using OVL in real-time.

12. Discussion

The first noticeable fact is that OVL can effectively calculate a continuous robot pose from a set of discrete locations. As a difference with previous discrete localization methods [9], there is no need of a huge 3D location probability array (the Env. 4 would need a 200×48×16 location matrix to reach the same error rates: 37.5 times more memory space than the $\mathbf{\Phi}$ vector for 4096 states).

The best attribute of OVL is the ability of managing without any further modification all the three sub-types of localization: GL, PT and RK. With an initial pose supposition it can handle with PT, if not, in a very small number of iterations it can find the true robot pose, solving GL, and automatically begins to keep track of the robot. If a RK is suddenly performed, the belief probability degenerates fast (due to the observation probability decay) forcing a GL, finding the true pose in 10 to 15 iterations. Also, the optimized version of OVL has proven its applicability to real-time applications running in processors with limited computing resources.

The main problem that causes the method to fail consists in evaluating the new pose when the robot displacement is below the average *xy* node separation (e.g. below 68 cm for env. #4), because the probability of remaining in the same state for a small displacement will be considerably higher than the probability of performing a true transition to another state. In fact, after many small displacements the robot could have changed from x_i to x_j, but the method would remain the robot at state x_i as long as the observation remain somewhat similar.

Another important issue is related with the GL firing when ϕ_t decays fast. As it is stated in this work, the method fires GL almost immediately after a RK. In future works

the method will be tested in crowded environments and the pose belief change will be quoted, in order to determine if OVL can be less sensitive to improbable observations. This was the main reason of integrating a TOM.

By the other hand, the main problem that compromises OVL performance in real-time is the variable processing time needed for the belief update, especially after a RK. The use of FPGA hardware could help the method to achieve the same update rates at high-speed, regardless of GL, PT or RK.

Related to this, results very important considering enough simulated observations when building the TOM, otherwise the method would be constantly performing GL due to the probability decay when in crowded environments.

13. Conclusions

Compared with the most widely used probabilistic pose estimation methods [11,12] the Odometry-based Viterbi Localization presents serious advantages: It is not necessary to estimate with high precision the observation at specific poses during operation as Monte Carlo Localization. Neither requires matrix inversion like Kalman Filters. Despite its discrete nature, it is able to estimate a continuous pose with limited resources. It can also handle with many hypotheses at once, as most of the state-of-the-art algorithms do. Also, because of the introduction of a Self Organized Map, OVL is able to use *n-dimensional* observations (including high-resolution visual information, or data coming from many sensors) because the hard work is performed by Vector Quantization Networks (perfectly suitable of being implemented in hardware).

Compared with the previous Viterbi Localization Approach by Savage et Al. [13], optimized-OVL has many important improvements:

1. The use of an odometry-dependent Motion Model.
2. Larger number of states (thousands against a dozen).

3. Continuous pose estimation. There is no need of „backtracking" for estimating the best discrete state sequence. All the trajectories (state-transitions) are considered.
4. Ability of handling GL, PT and RK by incorporating Update Triggering.
5. Self Organized Maps for building a Tolerant Observation Model.
6. No probabilistic tendency to zero out.

In conclusion, OVL is a Non-degenerative Probabilistic Algorithm (NDPA) that can effectively solve the entire robot localization problem, based both in odometry and observation information. It saves memory and processing time due to its discrete nature, but provides continuous pose estimations. Finally, it can run in single and multi-core microprocessors in real-time.

14. Future Works

Although in this paper we have started from a previously built map and statistically generated the Observation and Motion Models, by using an ANN architecture, a real-time learning procedure can be used for re-adapting these model parameters to the actual operation conditions (fine tuning). This architecture and its training will be presented in the future for adapting OVL to changing environments.

Also, some works using stereo and omni-vision cameras as the main sensor will be presented, because OVL generality easily allows using visual information for training the VQN. In this case, the narrowed field-of-view of computer cameras (45 degrees or so) compared with the field-of-view of the laser range scanners (240 degrees) and the two-dimensional visual information would turn very interesting the process of building the TOM. In this sense, stereoscopic cameras could help to provide depth information (as laser range finders do). Another important factor would be a variable illumination, thus the Vector Quantization of observations could incorporate some previous image feature extraction (like corners of SIFT [36]), instead of the raw sensors values: in this case three values for every pixel: red, green and blue.

15. References

[1] S. Thrun, W. Burgard, D. Fox,"Probabilistic Robotics", MIT Press, 2005

[2] A. Llarena, "Here Comes the Robotic Brain", Trends in Intelligent Robotics, Communications in Computer and Information Science, 2010, Volume 103, Part 2, 114-121

[3] W. S. Choi, and S. Y. Oh (2007), "Range Sensor-based Robot Localization Using Neural Network", International Conference on Control, Automation and Systems, p. 230-234.

[4] O. Djekoune, K. Achour, "Vision-guided Mobile Robot Navigation Using Neural Network". Proceedings of 2nd International Symposium on Image and Signal Processing and Analysis, (2001) 355-361.

[5] J. A. Janet, and R. Gutierrez, and T. A. Chase, "Autonomous mobile robot global self-localization using kohonen and region-feature neural networks", Journal of Robotic Systems, (1997), 263 – 282

[6] J. Racz, A. Dubrawski, "Mobile Robot Localization With an Artificial Neural Network", International Workshop on Intelligent Robotic Systems IRS'94, Grenoble, France, July 1994.

[7] K. Wang, W. Wang, Y. Zhuang, "Appearance-Based Map Learning for Mobile Robot by Using Generalized Regression Neural Network", ISNN (1) 2007: 834-842.

[8] M. Conforth, and Y. Meng, (2008), "An Artificial Neural Network Based Learning Method for Mobile Robot Localization", Robotics Automation and Control, Pavla Pecherkova, Miroslav Flidr and Jindrich Dunik (Ed.), ISBN: 978-953-7619-18-3, InTech,

[9] D. Fox, W. Burgard, S. Thrun, "Markov localization for reliable robot navigation and people detection", In Modeling and Planning for Sensor-Based Intelligent Robot Systems. Springer Verlag, Berlin, 1999.

[10] R. Simmons and S. Koenig, "Probabilistic Navigation in Partially Observable Environments", IJCAI '95, Montreal Canada, July 1995

[11] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo Localization for Mobile Robots," Artificial Intelligence Journal, 2001.

[12] S. Roumeliotis, G. Bekey, "Bayesian estimation and Kalman filtering: A unified framework for mobile robot localization". In Proc.2000 IEEE ICRA, San Francisco, CA, April 22-28. (2000) 2985–2992.

[13] J. Savage, M. Morales, E. Márquez, "The Use of Hidden Markov Models and Vector Quantization for Mobile Robot Localization", Robotics and Applications (RA 2005), IASTED, Boston, U.S.A.

[14] F. Lu, and E. Milios, "Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans", JIRS(18), No. 3, March 1997, p. 249-275. 9705.

[15] Crowley, J. L. & Demazeau, Y., "Principles and techniques for sensor data fusion", Signal Processing, Vol. 32, Nr. 1-2 , May (1993) , S. 5-27.

[16] A. Diosi and L. Kleeman, "Advanced sonar and laser range finder fusion for simultaneous localization and mapping", Proc. IROS, 2004.

[17] A. Elfes. "Using occupancy grids for mobile robot perception and navigation", Computer, 22(6):46–57, 1989.

[18] S. Thrun, J.-S. Gutmann, D. Fox, W. Burgard, and B. Kuipers, "Integrating topological and metric maps for mobile robot navigation: A statistical approach", Proceedings of the AAAI Fifteenth National Conference on Artificial Intelligence, 1998.

[19] J. Savage, A. Llarena, G. Carrera, S. Cuellar, D. Esparza, Y. Minami, and U. Peñuelas, "ViRbot: A System for the Operation of Mobile Robots", in Proc. RoboCup, 2007, pp.512-519.

[20] G.D.A. Barreto, A.F.R. Araújo, and H. Ritter, "Self-Organizing Feature Maps for Modeling and Control of Robotic Manipulators", presented at Journal of Intelligent and Robotic Systems, 2003, pp.407-450.

[21] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition", Proceedings of the IEEE, vol. 77, no. 2, Feb. 1989, pgs 257 - 285.

[22] Menegatti, A. Pretto, A. Scarpa, E. Pagello, "Omnidirectional vision scan matching for robot localization in dynamic environments", IEEE Transactions on Robotics, Vol. 22, 2003.

[23] J. Little S. Se, D. Lowe, "Vision-based mobile robot localization and mapping using scale-invariant features", Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pages 2051–2058, 2001.

[24] Jaynes, E. T., "Probability theory: the logic of science", Cambridge University Press. ISBN 9780521592710, 2003.

[25] T. Kailath, "Lectures Notes on Wiener and Kalman Filtering," Springer- Verlag, 1981.

[26] S. J. Julier and J. K. Uhlmann. "A new extension of the Kalman Filter to Nonlinear Systems", Proc. of AeroSense, The 11th Int. Symp. On Aerospace/Defense Sensing, Simulation and Controls, 1997.

[27] S. J. Julier and J. K. Uhlmann and H. Durrant-Whyte. "A new approach for nonlinear systems", In Proceedings of the American Control Conference, pages 1628-1632, 1995.

[28] A.H. Jazwinski, "Stochastic Processes and Filtering Theory", Academic Press, New York, NY, 1970.

[29] A. Doucet, N. de Freitas, K. Murphy, and S. Russell, "Rao-blackwellised particle filtering for dynamic Bayesian networks". In UAI, 2000.

[30] A. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm", IEEE Transactions on Information Theory 13(2), 260-269, 1967.

[31] S. Fortune. "A sweep line algorithm for Voronoi diagrams". Proceedings of the second annual symposium on Computational geometry. Yorktown Heights, New York, United States, pp.313–322. 1986. ISBN:0-89791-194-6.

[32] Y. Linde, A. Buzo, and R. M. Gray, "An Algorithm for Vector Quantizer Design". IEEE Transactions on Communications, 84-95, 1980

[33] J. C. Latombe, "Robot Motion Planning", 3rd Edition, Kluwer Academic Publisher, 1991.

[34] S. Haykin, "Neural Networks. A Comprehensive Foundation", 2nd Ed., Prentice Hall 1999.

[35] Gang Liu, Robert M. Haralick, "Two Practical Issues in Canny's Edge Detector Implementation," icpr, vol. 3, pp.3680, 15th International Conference on Pattern Recognition (ICPR'00) - Volume 3, 2000

[36] D. Lowe, "Object recognition from local scale-invariant features," in Proceedings of the Seventh International Conference on Computer Vision(ICCV'99), Kerkyra, Greece, September 1999, p. 1150–1157.

16. List of Figures

Apéndice B

Uso del Software rSLAM 1.0

B.1 Introducción

El software desarrollado durante el presente trabajo, denominado rSLAM 1.0 comprende un conjunto de librerías de creación de mapas métricos, topológico, cuantificación vectorial de lecturas láser, navegación autónoma y autolocalización, es decir, cubre todos los aspectos del problema de Localización Simultánea y Mapeo (SLAM por sus siglas en inglés) para un robot diferencial. Facilita la creación de mapas métricos y topológicos de acuerdo a lo referido en [Llarena 11].

En la versión presentada en este apéndice, se incorporaron rutinas de localización que le permiten al robot estimar su posición actual dentro del ambiente de operación.

B.1 Configuración

El primer paso previo a la ejecución de rSLAM consiste en la configuración del dispositivo láser. Para tal efecto es necesario abrir el archivo `robot.cfg`, ubicado en la carpeta `rVision/bin`. Una vez abierto es necesario localizar la siguiente línea:

```
TX8_NUM_LASER            68 radial 2 0 -119.2 119.5
```

En ella se indica el número de lecturas útiles del láser que vamos a tomar como una observación (de 0 a 660 para el láser Hokuyo URG04LX). Luego, el tipo de lectura del sensor (`radial` indica que el haz de luz del láser es emitido desde el mismo punto), la distancia relativa al centro del robot (visto como una circunferencia desde arriba) en

decímetros (20 cms en ejemplo anterior), el ángulo relativo al frente del robot en el cual se encuentra el centro óptico de emisión del haz de luz láser (en el mismo ejemplo éste se encuentra situado a 20 cms hacia el frente del robot o 0^o). Finalmente se deben indicar los ángulos inicial y final del barrido del láser que corresponderá a nuestras lecturas útiles tomadas como una observación (119.2 a -119.5 para el URG04LX, es decir, el haz de luz barre de izquierda a derecha). Si hubiera necesidad de colocar de cabeza el láser, bastaría con cambiar el signo de los ángulos inicial y final de barrido para que rSLAM pueda tomar correctamente las lecturas. De manera análoga es posible definir los parámetros de un segundo láser mediante la etiqueta TX8_NUM_LASER2.

Una vez hecha la configuración del número de lecturas del láser, es necesario configurar los puertos de conexión. Para ello se debe ubicar el siguiente grupo de líneas:

```
;*** LASER PORTS ***/
LASERPORTWIN                COM5            COM8
LASERPORTLIN                /dev/ttyACM0    /dev/ttyACM1
LASERPORTMAC                /dev/ttyACM0    /dev/ttyACM1
LASERPORTWCE                COM1            COM2
```

En ellas es necesario especificar (para el sistema operativo utilizado: Windows, Linux, Mac OSX y Windows CE) los puertos COM, correspondientes al primer y segundo láseres. Por omisión, rSLAM tratará de conectar el láser por el puerto serie a la máxima tasa de transmisión posible, sin embargo es posible fijar dicha tasa mediante la línea:

```
LASERBAUD                   57600           57600
```

donde las tasas válidas son: 1200, 2400, 4800, 9600, 14400, 28800, 57600 y 155200 bps.

B.2 Ejecución del Programa

rSLAM fue desarrollado bajo lenguaje ANSI C++ por lo que puede ser ejecutado en sistemas operativos Windows, Mac Os X y Linux entre otros. En cualquiera de sus

versiones la sintaxis de comandos es la misma y el desplegado de mensajes se hace vía consola de terminal.

Para ejecutar el programa basta con teclear en modo terminal (Mac OS X y Linux) el comando ./`rSLAM` o ejecutar en Windows el archivo ejecutable con el mismo nombre `rSLAM.exe`.

rSLAM, además de incorporar rutinas para resolver el problema SLAM, integra un poderoso motor de sistemas expertos denominado CLIPS, desarrollado por la NASA en E.U.A. y de libre distribución y copia. Una vez que se ejecute el programa aparecerá el prompt de CLIPS

```
CLIPS>
```

B.4 Principales Comandos

1) Carga de un ambiente de trabajo (mapa).- rSLAM cuenta con un directorio con algunos mapas métricos previamente elaborados. Para cargar un mapa previamente elaborado se usa el comando:

```
CLIPS> (ldmap <ruta>)
```

Donde <ruta> indica la ruta absoluta o relativa desde el directorio de ejecución del programa hacia el archivo de mapa métrico. Por ejemplo el comando:

```
CLIPS> (ldmap ../maps/map.LBR)
```

Lee desde el directorio de mapas el archivo map.LBR. Luego de cargar el mapa se presenta en pantalla en una ventana de OpenGL:

```
Reading input file ...
World dimensions:  (-13,42) to (66,153)
Number obstacles: 630

Making Map ...
World dimensions:(-13.81, 42.61) to (66.93, 153.38)
Map Rows: 37 , Map Columns: 27
Total Cells: 999
Safedist not connect 1.000000
Free Cells: 628
```

Figura 1. Carga de mapa métrico

2) Posición Inicial.- La ubicación inicial del robot puede ser fijada con el siguiente comando:

```
CLIPS>(setpos <x> <y> <orientación en radianes>)
```

En caso de desconocer la ubicación inicial, éste comando se puede omitir y el algoritmo de localización se encargará de encontrar la posición actual del robot (por omisión <0,0,0>), con base en los desplazamientos y observaciones sucesivas.

3) Visualización de Partículas.- Por omisión, rSLAM incorpora un filtro de partículas con el Método de Monte Carlo como medio de localización. Dado que el software cuenta con una ventana de desplegado donde es posible ejecutar algunos comandos de manera manual, únicamente pulsando únicamente una tecla en el teclado mientras dicha ventana se encuentra activa (presione la tecla *h* en el teclado para ver una lista completa de los comandos disponibles).

Presione la tecla "=" para ver la distribución inicial de un conjunto de partículas.

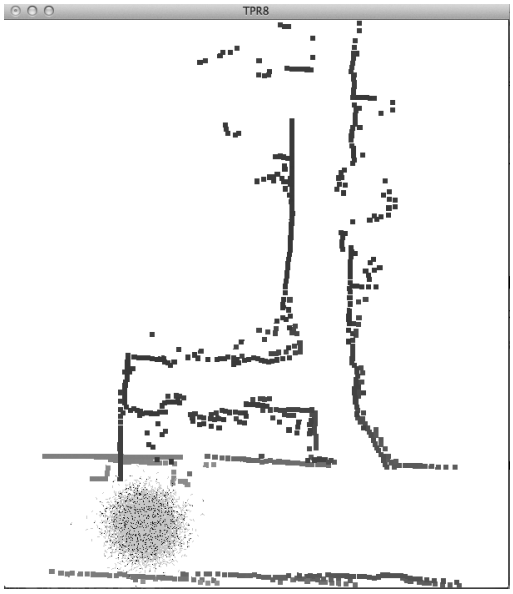


Figura 2. Distribución inicial de partículas (hipótesis de localización)

4) Comandos de Movimiento.- Para simular el movimiento del robot (y por consiguiente la actualización de la nube de partículas), ejecute el siguiente comando de desplazamiento:

```
CLIPS>(mv <distancia_en_dm> <ángulo_giro_relativo_al_frente_del_robot>)
```

Por ejemplo, el comando

```
CLIPS>(mv 10 0)
```

hará que el robot se desplace un metro en línea recta hacia el frente (manteniendo su orientación actual) y se actualice la posición de la nube de partículas.

Luego de que el robot recorre un metro (distancia acumulada), de forma automática se dispara el la fase de actualización del filtro de partículas, mediante el desplazamiento de las propias partículas y el posterior cotejamiento contra la observación más reciente, obteniendo una nueva posición global.

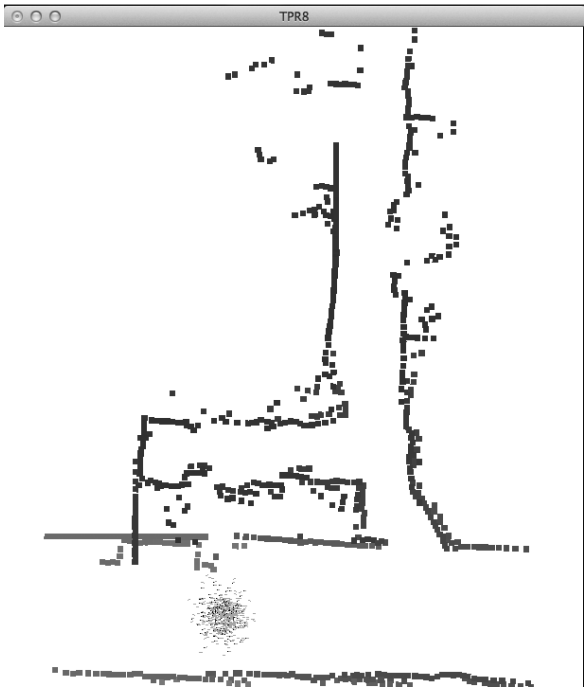


Figura 3. Nueva ubicación de la nube de partículas
(luego de la fase de actualización de las observaciones)

Cabe hacer la aclaración que el sistema requiere de un sensor láser (modelo Hokuyo URG04LX o similar) conectado a través de un puerto USB o serial a la computadora. En la fase de actualización de la observación se tomará una nueva lectura luego de cada movimiento sucesivo del robot, la cual es cotejada contra un archivo LUT (Look Up Table) generado previamente, el cual almacena lecturas simuladas con ayuda del mapa métrico del ambiente y optimizada para búsquedas rápidas mediante un Árbol Kd. El proceso de creación de la LUT es automático y se detalla a continuación.

5) Creación del Archivo de Observaciones (Tabla de Búsqueda o Look Up Table LUT).- Para evitar el uso de simulación en tiempo real de las lecturas del láser para cada ubicación de la nube de partículas (lo cual puede involucrar incluso más de un minuto para 1000 partículas y 68 lecturas del láser) se utiliza un método fuera de línea. Luego haber definido los parámetros del sensor láser en el archivo de configuración (número de lecturas, ángulos de barrido y máxima distancia), al momento de cargar un mapa del ambiente (comando `ldmap`), el propio sistema rSLAM genera una observación simulada para un total de 180,000 ubicaciones aleatorias dentro de la zona navegable del

mapa métrico cargado. Por ejemplo, si se están utilizando 64 lecturas del laser, la LUT contendrá 180,000 vectores de dimensión 67, 64 lecturas láser más la ubicación (x, y, θ) en la cual se simuló la lectura.

Una vez calculadas las observaciones simuladas para la LUT, ésta es almacenada en un archivo de tipo texto, con la misma ubicación y nombre del archivo del mapa métrico (más la extensión `.lt` que indica que se trata de la LUT para dicho mapa). Este archivo es leído cada vez que el mapa métrico es cargado. Con las 180,000 lecturas se construye un árbol *kd* que resulta muy útil para acelerar notablemente la búsqueda de la observación supuesta más cercana a la posición de cada partícula, durante la fase de actualización de la observación del filtro de partículas de rSLAM.

En resumen, el proceso de creación y carga de la LUT se hace de manera autónoma por el sistema y no requiere en absoluto la intervención del usuario.

6) Actualización de la posición estimada por el filtro de partículas.- Para conocer la estimación actual de la ubicación del robot, es necesario ejecutar el siguiente comando:

```
CLIPS>(pos)
```

el cual imprime la posición actual estimada del robot. Al ejecutarlo en nuestro ejemplo anterior proporcionará:

```
X = 5.000, Y = 55.000, Angle = 0.000, Pan = 0.000, Tilt = 0.000,
Secur = 1.0000
```

Los valores Pan y Tilt indican la orientación de la cabeza del robot, sin embargo no resultan de utilidad en el ejemplo actual. Por su parte el valor “Secur” o *secureness*, indica la confiabilidad de la estimación de ubicación en la escala de cero a uno. Éste dato tiene que ver con la dispersión de la nube de partículas y es de suma importancia, ya que

puede haber ocasiones en las cuales el método no ha convergido aún a una nube de puntos distribuida alrededor de la ubicación real, cuya varianza sea baja y por lo tanto su valor de *secureness* sea cercano a uno. En caso contrario, si la nube se encuentra muy dispersa en el ambiente o hay más de una sola región de probabilidad, la varianza de las partículas será alta, por lo tanto el valor de *secureness* será próximo a cero.

7) Comunicación a través de Sockets UDP.- El sistema rSLAM, ha sido provisto de un canal de comunicación por sockets UDP denominado `COMM_IN_PORT`, el cuál actúa como la entrada estándar del sistema. De igual forma, la salida estándar es direccionada al socket UDP `COMM_OUT_PORT` del equipo especificado en `REMOTE_COMM_HOST` del archivo de configuración en las siguientes líneas:

```
;*** SOCKETINPUT ***
REMOTE_COMM_HOST            127.0.0.1
COMM_IN_PORT                     9000
COMM_OUT_PORT                    9001
```

Al utilizar el siguiente comando:

```
CLIPS> (mysockOn)
```

se direcciona la entrada y salida estándar (ES y SE respectivamente) a los puertos indicados en el archivo de configuración.

Para interrumpir el direccionamiento de la entrada y salida estándar (ES y SE) al socket UDP, es necesario transmitir el siguiente comando a través del propio socket UDP que actúa como entrada estándar:

```
(mysockOff)
```

Recuerde que mientras esté redireccionada la ES los comandos que escriba directamente en la consola de terminal no tendrán ningún efecto.

8) Procesos por lotes o *batch*.- Es posible escribir un archivo por lotes o *script* de CLIPS que se ejecute cada vez que se inicia rSLAM. Para ello deberá modificar el contenido del archivo de arranque de CLIPS especificado en el archivo de configuración mediante la siguiente línea:

```
TX8_CLIPS_FILE          virbot.clp
```

De esta forma, el contenido de dicho archivo (`virbot.clp` en el ejemplo) podría ser:

```
(ldmap ../maps/map.LBR)
(setpos 5 10 0)
(mysockOn)
```

lo que indica que, de manera automática al ser ejecutado, rSLAM cargará el mapa `../maps/map.LBR`, fijará la posición inicial del robot en (`5 10 0`) y direccionará la ES y SE al socket UDP indicado en el archivo de configuración.

7) Integración de rSLAM con otros sistemas.- La manera más sencilla de integrar rSLAM como un módulo de SLAM en sistemas de control robótico es mediante los sencillos comandos arriba mencionados. Bastará con elaborar una interfaz o API que abra un socket UDP con rSLAM y que transmita (ejecute) los siguientes comandos en secuencia (luego de la carga del mapa métrico y fijación de la ubicación inicial):

```
(mv <distancia_proveniente_de_odometría> <ángulo_estimado_de_giro>)
(pos)
```

Una vez que los comandos anteriores sean transmitidos a rSLAM, es posible extraer de la cadena que retorna el sistema (luego del comando `pos`), tanto la posición estimada como su confiabilidad, calculada con el algoritmo de localización de rSLAM.

B.5 Creación del Mapa Métrico con rSLAM

Como se mencionó al principio del presente Apéndice, rSLAM es capaz de realizar las principales tareas de SLAM (Simultaneous Localization and Mapping). Una de esas tareas es la creación del mapa métrico.

Para tal efecto, será necesario ejecutar una serie de comandos de manera secuencial para acumular las lecturas del láser y generar un mapa métrico del ambiente.

1) Modo de Creación de Mapa.- Por omisión, al momento de cargar un mapa inmediatamente rSLAM entra en modo de localización y utiliza cada nueva lectura como entrada del algoritmo de localización. Sin embargo, es posible indicar al sistema que se desea utilizar las lecturas para elaborar un mapa mediante el siguiente comando:

```
CLIPS> (mkmap)
```

el cual indica al sistema que se encuentra en modo de creación de mapa y hace que no utilice el algoritmo de localización.

2) Acumulación de Lecturas.- A partir de este punto, todas las lecturas realizadas por el robot serán acumuladas para formar un nuevo mapa métrico. Por lo tanto, debido al propio error de odometría del robot, es conveniente mover manualmente al robot físico en línea recta (por ejemplo un metro hacia el frente) y ejecutar los siguiente comandos:

```
CLIPS> (setpos 0 0 0)
CLIPS> (mv 1 0)
CLIPS> (look)
```

los cuales moverán la ubicación del robot en rSLAM un metro en línea recta hacia el frente y forzarán a que se tome una nueva lectura (comando `look`). Una vez en su nueva posición, se recomienda girar al robot físico 180 grados y tomar otra lectura en la misma ubicación (x, y) pero en la dirección contraria (para cubrir los 360 grados de observación para esa ubicación). De igual forma es necesario mover la posición del robot en rSLAM y forzar a una nueva observación mediante los comandos:

```
CLIPS> (mv 0 3.1416)
CLIPS> (look)
```

Luego, se gira manualmente al robot real 180 grados para continuar con el movimiento hacia el frente y desplazarlo otro metro en la misma dirección:

```
CLIPS> (mv 0 3.1416)
CLIPS> (mv 10 0)
CLIPS> (look)
```

El proceso anterior se repite, moviendo al robot real a intervalos regulares de un metro de espaciamiento o menos. De igual forma, se recomienda que donde el robot deba girar esto se haga preferentemente a intervalos de 90 grados (ya que en muchas ocasiones esto facilita el correcto posicionamiento del robot, especialmente cuando se poseen pisos de loseta cuadrada o rectangular que sirven como referencia de alineamiento).

3) Generación del Mapa.- Una vez que se haya recorrido la mayor cantidad posible de ubicaciones navegables del ambiente (el usuario deberá determinar en qué momento observa en la ventana de desplegado del rSLAM que existen "suficientes" lecturas), se ejecuta el siguiente comando para procesar las lecturas y generar un nuevo mapa métrico mediante el algoritmo de agrupamiento o *clustering* difuso especificado en [Llarena 11, p. 97]:

```
CLIPS> (stopmap)
```

Dicho comando generará una serie de grupos o *clusters* como una serie de pequeños objetos cuadrados cuyos lados miden el tamaño especificado (en decímetros) en la siguiente línea del archivo de configuración:

```
TX8_CLUSTER_RATIO        0.5
```

Nota: se recomienda tener cuidado de no fijar el tamaño de los clusters en un valor por encima de los 20 cms, ya que si bien esto reducirá el número final de los mismos, como éstos representan objetos fijos del ambiente, se corre el riesgo de que exista muy poca área libre navegable en el mapa final o de que la representación métrica no posea el espacio suficiente en pasillos estrechos para que el robot pueda navegar por los mismos con cierto margen de seguridad especificado en la línea:

```
TX8_SAFE_DIST            1.0
```

4) Almacenamiento del Mapa.- Una vez generado el nuevo mapa, éste se mostrará en la ventana gráfica. Para almacenarlo en disco es necesario ejecutar el siguiente comando:

```
CLIPS> (savemap ../maps/demo.map)
```

indicando la ruta y nombre de almacenamiento, relativos al directorio rSLAM/bin.

Con lo anterior se cubren los aspectos básicos de uso de rSLAM. En el futuro se prevé publicar un manual de uso más completo.