PLATAFORMA ROBÓTICA PARA LA ASISTENCIA EN LA MOVILIDAD DE INVIDENTES

Gerardo Becerra, Jhon Páez.

Pontificia Universidad Javeriana, Facultad de Ingeniería,
Maestría en Ingeniería Electrónica,
Bogotá, Colombia
gbecerra@javeriana.edu.co

Abstract. El proyecto *Plataforma Robótica para la asistencia en la movilidad de Invidentes* se ha realizado en tres etapas: la primera es la implementación del algoritmo PRM para la planificación de rutas en un entorno conocido, la segunda consiste en la solución del problema de localización por el método de Monte Carlo y por ultimo, se desarrolla una red neuronal *backpropagation* para la navegación del robot. Se ha verificado su operación empleando el software Robot Soccer Simulator y se ha evaluado el desempeño del algoritmo propuesto. En este documento se describe dicha implementación y se presentan los resultados obtenidos.

Keywords: Algoritmo de Dijkstra, planificación de trayectorias, localización, Monte Carlo, redes neuronales, robótica, Matlab, C++, FIRA.

1 Introducción

La Organización Mundial de la Salud estima que 500 millones de personas en el mundo, es decir el 10% de la población tiene algún tipo de discapacidad. Además en la mayoría de países en conflicto se calcula que esta cifra podría alcanzar el 18% de la población total. De acuerdo con los datos arrojados por el Censo General 2005, realizado por el DANE, aproximadamente 2.640.000 presentan alguna limitación permanente, lo cual equivale al 6,4% del total de la población colombiana. Una de las características de la discapacidad es la personas es la perdida en la autonomía relacionada con el movimiento, el cual tiene causas hereditarias, congénitas o provocadas por accidentes.

Para los discapacitados visuales, los procesos de movimiento y orientación en diferentes entornos se constituyen en un elemento básico para el conocimiento espacial. Por este motivo se propone el desarrollo de un sistema robótico móvil que le ayude al sujeto invidente a navegar en un ambiente determinado.

Este informe presenta el desarrollo, implementación y análisis de un modelo computacional propuesto para la navegación de un robot móvil de tipo diferencial en un entorno conocido como propuesta de una plataforma robótica para la asistencia en la movilidad de invidentes. El ambiente propuesto es un área común de oficinas,

unidas por medio de corredores de diferentes medidas de largo y ancho, donde el piso es completamente plano sin desnivel.

El trabajo se ha realizado en tres etapas: la primera es la implementación del método conocido como *probabilistic road map* para la planificación de rutas en un entorno conocido, la segunda consiste en la solución del problema de localización por el método de Monte Carlo y por ultimo, se desarrolla una red neuronal *backpropagation* para la navegación del robot. Se ha verificado su operación implementado los diferentes algoritmos en MATLAB con el objetivo de comprobar de forma rápida y simple los modelos computacionales planteados en la fase anterior y en la tercera fase es la implementación en C++ de los algoritmos desarrollados anteriormente, realizando sobre éstos los análisis de eficiencia dentro modelo computacional.

2 Planificación de Trayectorias

El método PRM es una técnica que permite la creación de los mapas de rutas probabilísticos PRM. Esta técnica determina la conectividad entre los diferentes nodos disponibles en el espacio de configuración del robot. El desarrollo de esta técnica, tiene en cuenta varios elementos; inicialmente se deben buscar las rutas permitidas durante la trayectoria del estado inicial al estado meta. El segundo elemento importante es la aplicación del algoritmo de Dijkstra, que tiene como objetivo encontrar la ruta más corta entre la variedad de puntos y grafos creados anteriormente. En muchas ocasiones, las características de los mapas de ruta probabilísticos, tienden a intentar todas las conexiones, pero el costo computacional es muy alto [1]. Por este motivo se debe limitar el número de ciclos que permiten la generación del mapa y de la trayectoria en la ruta, también se limita la conexión de nodos en función de un radio previamente establecido.

2.1 Comprobación del método en Matlab

El método PRM es una técnica que permite la creación de los mapas de rutas En la fase dos, se implementaron los algoritmos matemáticos necesarios para la comprobación del los métodos propuestos. (PRM y Dijkstra). Inicialmente se diseño una imagen que representa el entorno de oficinas con diferentes obstáculos, al digitalizar la imagen en Matlab, esta quedará representada por una matriz de N x M donde los valores de cada espacio de la matriz quedaran con un peso de 0 ó 1 que representan la ausencia o presencia de obstáculos respectivamente.

Para colocar los puntos aleatorios $P_{(x,y)}$ dentro del ambiente, se implementó una matriz de (2 x m), donde m, es la cantidad de filas. Se deben filtrar estos puntos, ya que algunos pueden quedar dentro de los obstáculos y por lo tanto no son útiles para generar el grafo de la ruta más corta. Este proceso de filtro se hace por medio de una

rutina que recorre cada punto de la matriz del entorno y verifica que en la coordenada x_i, y_i el valor sea 0, lo que corresponde a un punto candidato para la generación de la ruta mas corta.

Otro filtro se usa para determinar la visibilidad entre los puntos, ya que es posible que en la implementación computacional del algoritmo, varios puntos estén dentro del espacio libre, pero no se puedan conectar por la presencia de obstáculos entre ellos como se muestra en la figura 1.

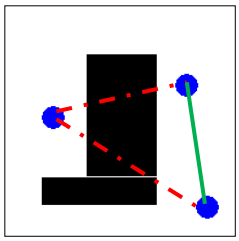


Fig. 1. Condición de visibilidad en los puntos.

Para determinar la visibilidad, se implementó un algoritmo que selecciona cada punto de la matriz y establece la condicion de visibilidad en una region determinada por una distancia $d = 100$. Para esto se implemento la ecuacion de la recta $y = mx + b$, que selecciona dos puntos y verifica que la recta que une los dos puntos, no pase por encima de un obstáculo. Con este recorrido se crea una matriz de adyacencia, que establecerá la distancia y la visibilidad entre todos los puntos.

	P1	P2	P3	P..	P..	P..	P..	P..	Pn-1	Pn
P1	1	20	0	40	...	...	...	...	...	...
P2	20	1	...	...	...	...	...	...	...	...
P3	0	...	1	...	...	...	...	...	...	...
P..	40	...	...	1	...	...	...	...	...	...
P..	...	...	...	...	1	...	...	...	...	...
P..	...	...	...	...	...	1	...	...	...	...
P..	...	...	...	...	...	...	1	...	...	...
Pn-1	...	...	...	...	...	...	...	1	...	...
Pn	...	...	...	...	...	...	...	...	1	...

Fig. 2. Matriz de adyacencia que establece la visibilidad y distancia de los puntos.

Finalmente, al tener cada los puntos completamente validos, se implementa el algoritmo de Dijkstra para obtener la ruta más corta en términos de distancia. Se implementó de la siguiente manera [3]:

Teniendo en cuenta que el peso de cada uno de los caminos (i, j) es $w(i, j) > 0$ y la marca del vértice x es $L(x)$. Al terminar, $L(z)$ es la longitud del camino mas corto de a a z.

1. *Inicialización*: Sea $L(a) := 0$. Para todo vértice $x \neq a, sea\ L(x) := \infty$. Sea T el conjunto de vértices.

2. *¿Terminó?*: Si $z\ no\ \in T$, ha terminado. ($L(z)$*es la longitud del camino mas corto de* $\boldsymbol{a}$ *a* $\boldsymbol{z}$).

3. *Pase al siguiente vértice:* Escoja $v \in T$, en donde $L(v)$ tiene un valor mínimo. Tome $T := T - \{v\}$

4. *Revise las marcas:* Para cada vértice $x \in T$, adyacente a v, se toma $L(x) := mín\{L(x), L(v) + w(v,x)\}$

5. *Pase a la instrucción 2.*

2.2 Implementación en C++

Para almacenar la información necesaria y realizar la planificación, se definieron varios arreglos que contienen la información del ambiente, de los nodos muestreados, del grafo de conectividad y del camino que se genera como resultado de la búsqueda.

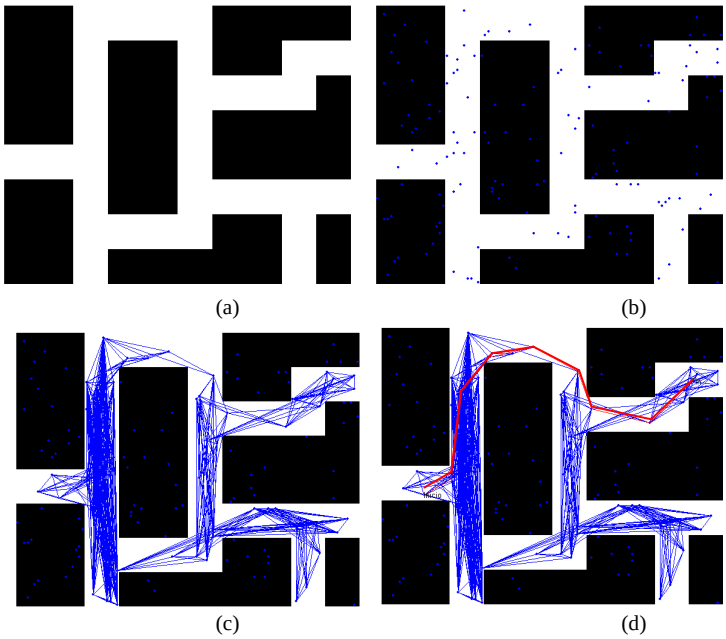


Fig. 3. (a) Ambiente definido para la verificación del algoritmo de planeación. Las zonas claras representan el espacio libre y las oscuras los obstáculos. Se assume que ya se ha calculado la diferencia de Minkowsky entre los obstáculos y el robot móvil (obstáculos agrandados) (b) Muestreo aleatorio de puntos en todo el ambiente. (c) Generación del grafo de visibilidad entre todos los nodos ubicados en el espacio libre. (d) Camino más corto obtenido por el algoritmo de Dijkstra entre los puntos de inicio y destino. Se utilizó Matlab para visualizar los arreglos generados por el algoritmo en C++.

De manera similar al proceso realizado en Matlab, el algoritmo en C++ requiere como entrada una definición del ambiente codificada en el arreglo Entorno (Fig. 3). Se realiza el muestreo aleatorio de un número determinado de nodos sobre todo el

ambiente y se procede a eliminar los que quedaron ubicados dentro de los obstáculos. Luego se calculan las distancias euclidianas comprendidas entre todos los pares posibles de nodos (i,j) y estos resultados se almacenan en el arreglo Dist. De esta manera se tienen dos arreglos que contienen la informacion completa de los nodos que van a utilizarse para la búsqueda: el arreglo Grafo contiene un listado indexable con las posiciones (x,y) de todos los nodos y el arreglo Dist contiene las distancias entre éstos.

Para determinar la conectividad de los nodos a través del espacio libre, se calcula para cada par de nodos la ecuación de la recta que los une y se realizó una iteración a lo largo de dicha recta, evaluando en cada punto si se presenta una intersección con algún obstáculo. Se observó que para los pares de nodos cuyas posiciones en el eje x eran cercanas, este método no lograba determinar correctamente si pasaban a través de un obstáculo, por lo cual se realizó el mismo barrido en el eje y, calculando la ecuación inversa de la recta.

Una vez obtenido el grafo de conectividad, se aplica el algoritmo de Dijkstra para realizar la búsqueda del camino con la distancia más corta entre el punto de inicio y el punto de destino. Los indices de los nodos del camino obtenido se almacenan en el arreglo Camino, el cual es utilizado por el robot móvil para realizar su recorrido.

El algoritmo implementado en C++ fue integrado a la función “Strategy” del software Robot Soccer Simulator, para realizar la comprobación visual del movimiento del robot de acuerdo con la planeación de rutas implementada.

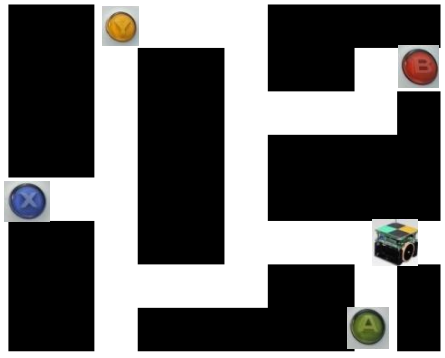


Fig. 4. Puntos de interés definidos dentro del ambiente.

Se realizó la definición dentro del ambiente de varios puntos de interés (Fig 4) hasta los cuales se desea realizar el desplazamiento del robot móvil, y se implementó la interfaz con el usuario a través de un control inalámbrico de Microsoft. El usuario selecciona el botón correspondiente al punto de destino hasta el cual desea desplazarse. El algoritmo recibe esta solicitud y agrega al listado de nodos la ubicación actual y la ubicación del punto de destino.

Para evaluar el desempeño del algoritmo de planificación, se realizaron diferentes experimentos modificando dos variables de interés que pueden influir en el desempeño: el número de nodos aleatorios muestreados uniformemente y la longitud máxima del camino que une dos nodos visibles entre sí.

Una característica que posee el uso de PRM para la generación del grafo es que si no se tiene una cantidad suficiente de nodos muestreados es posible que no se generen puntos en algunas regiones y éstas sean inaccesibles. Lo anterior es equivalente a afirmar que el algoritmo de PRM no es completo [4].

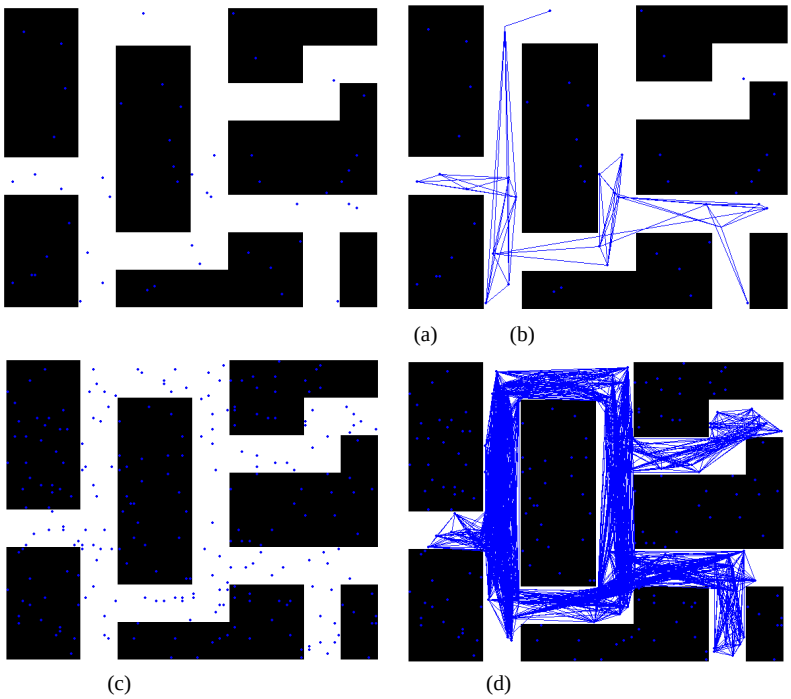
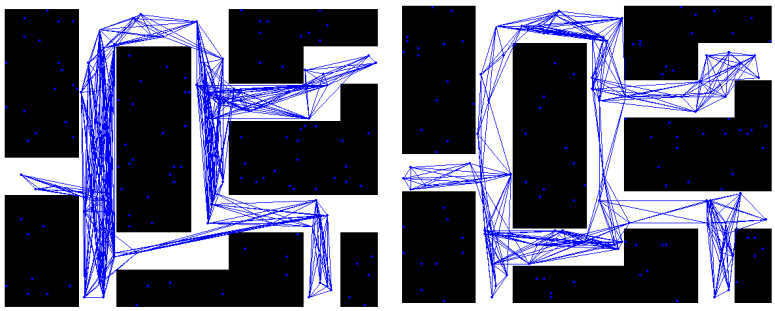


Fig. 5. (a) Muestreo aleatorio de 50 puntos. (b) Segmentos de visibilidad que conectan los nodos en el espacio libre. (c) Muestreo aleatorio de 250 puntos. (d) Segmentos de visibilidad que conectan los nodos en el espacio libre.

Con base en la Fig. 5 se puede afirmar que es más fácil generar un grafo que interconecte todas las zonas del espacio libre mientras mayor cantidad de puntos muestreados se tenga. Por ejemplo, en la Fig 7 (b) se puede observar un nodo ubicado en el corredor superior derecho que no tiene conectividad con los demás, y por lo tanto es inalcanzable desde otras regiones. Al realizar un muestreo con mayor cantidad de puntos, esta zona que antes era inaccesible ahora ya puede ser alcanzada por el robot móvil. Sin embargo, al realizar el muestreo de una mayor cantidad de puntos, la complejidad del grafo aumenta, lo cual podría llegar a afectar el desempeño del algoritmo, principalmente en el tiempo requerido para completarlo. Así que esta es una de las variables que serán observadas para hacer un estimativo del desempeño.



(a) (b)

Fig. 6. (a) Muestreo aleatorio de 150 puntos unidos con rutas que no tienen longitud máxima. (b) Muestreo aleatorio de 150 puntos con longitud máxima de ruta igual a 30.

Otra variable que puede llegar a impactar en el desempeño es la longitud máxima de los segmentos que unen los nodos que tienen visibilidad directa. En la Fig 6 (a) se realizó un muestreo de 250 puntos y se generaron todos los segmentos posibles. Nótese que se presenta una alta redundancia en los segmentos que comunican unas regiones con otras. En la Fig 6 (b) se limitó la longitud máxima de los segmentos a 30, obteniendo un grafo de menor complejidad. Se desea evaluar el grado de influencia de este parámetro en el desempeño general del algoritmo. Intuitivamente se podría suponer que al limitar la longitud máxima de los segmentos se tendrá un grafo de menor complejidad, mejorando el desempeño. También se tendrá una menor cantidad de rutas posibles entre los nodos y serán menos directas, haciendo necesario recorrer una distancia mayor para alcanzar el punto de destino. Esta hipótesis también fue evaluada a través de los experimentos realizados.

Se ejecutó el algoritmo completo de planeación cambiando el número de puntos muestreados y la longitud máxima de los segmentos y se midieron los tiempos de ejecución y la longitud obtenida para el camino encontrado como solución en el algoritmo de búsqueda. Debido a la naturaleza aleatoria del algoritmo, cada combinación de parámetros se ejecutó 20 veces y se calcularon los promedios de los resultados.

En la Fig. 7 se muestra gráficamente el comportamiento del tiempo empleado por el algoritmo para diferentes cantidades de nodos. Lo primero que puede concluirse a partir de esta gráfica es que el tiempo empleado por el algoritmo aumenta a medida que aumenta el número de nodos muestreados. Otra situación que se hace evidente con esta gráfica es que aparentemente los tiempos de ejecución del algoritmo no dependen en gran medida de la cantidad de segmentos entre nodos existentes en el grafo, ya que las gráficas para longitudes máximas de 30, 60 y 90 tienen un comportamiento muy similar.

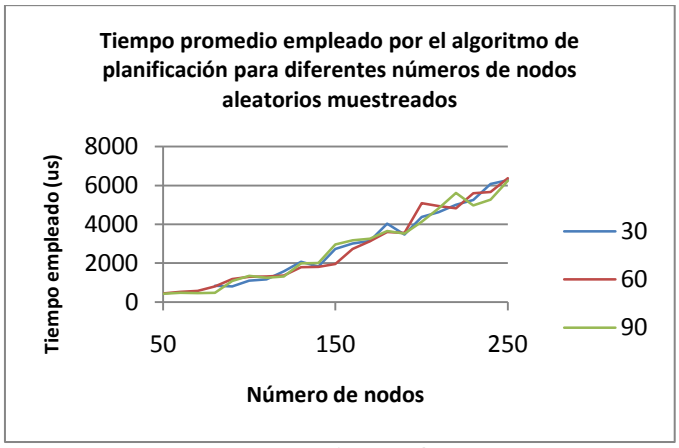


Fig. 7. Tiempo requerido por el algoritmo para diferentes números de nodos. Se muestran los resultados para cada experimento con máximas longitudes de segmentos entre nodos de 30, 60 y 90. Sólo se muestra la información de los ensayos que lograron encontrar el camino el 100% de los casos.

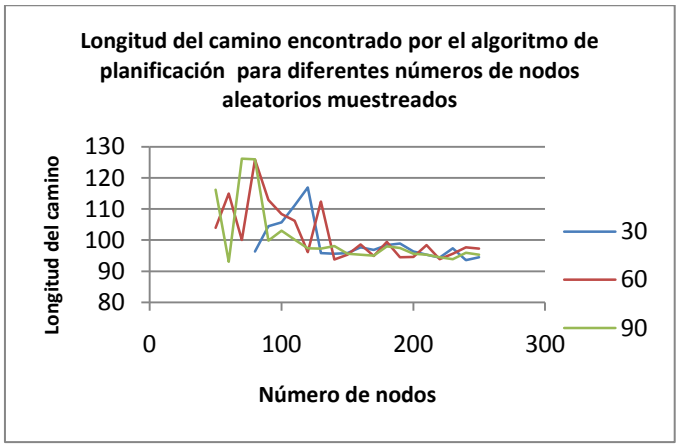


Fig. 8. Longitud de los segmentos resultado encontrados por el algoritmo para diferentes números de nodos. Se muestran los resultados para cada experimento con máximas longitudes de segmentos entre nodos de 30, 60 y 90. Sólo se muestra la información de los ensayos que lograron encontrar el camino el 100% de los casos.

En la Fig. 8 se muestra una gráfica con las longitudes de los caminos encontrados para diferentes números de nodos muestreados, identificando también las longitudes máximas de los segmentos entre nodos. Se observa que al aumentar la cantidad de nodos muestreados, las longitudes totales de los caminos de solución encontrados disminuyen hacia un límite inferior, el cual claramente está dado por el grafo de visibilidad en el cual las esquinas de los obstáculos son los nodos. También se puede observar que cuando se tiene una cantidad de nodos aleatorios suficiente, las longitudes de los caminos encontrados no dependen de la longitud máxima de los segmentos entre nodos. La causa de este comportamiento es que al muestrear cada

vez una mayor cantidad de puntos, éstos estarán más cercanos entre sí y el camino más corto obtenido por el algoritmo de búsqueda tenderá a parecerse cada vez más al camino que pasa por las esquinas de los obstáculos.

3 Localización

PARA que un robot móvil pueda planificar rutas y navegar a través de un ambiente de manera apropiada evitando las colisiones con los obstáculos que se encuentran en el recorrido, es necesario conocer con un alto grado de certeza su ubicación con relación a un marco de referencia global. En un ambiente real un robot móvil normalmente no dispone de esta información de manera directa, por lo cual es necesario realizar estimaciones a partir de la información obtenida por sus sensores. Recientemente este problema de estimación ha sido abordado usando técnicas probabilísticas que permiten modelar la incertidumbre en las diferentes variables a través de distribuciones de probabilidad. Como resultado, se ha logrado implementar algoritmos robustos y eficientes que logran solucionar el problema de localización.

3.1 Modelo de localización

En la figura 9 se presenta gráficamente el modelo de la localización de un robot móvil. Los nodos grises son conocidos: El mapa del ambiente m, las mediciones z obtenidas por los sensores y los controles u del robot. La meta de la localización es estimar las poses X del robot.

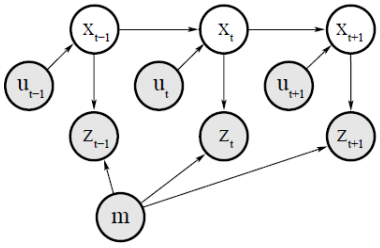


Fig. 9: Modelo de localización.

En general, todos los sistemas robóticos poseen algún grado de incertidumbre tanto en sus sensores como en sus actuadores. Para que un sistema de localización sea robusto ante la existencia de dicha incertidumbre, es importante implementar modelos que la tengan en cuenta utilizando distribuciones de probabilidad. En este caso particular se realizó la implementación de la Localización Monte Carlo (MCL), la cual posee ventajas en cuanto a facilidad de implementación y robustez frente a otros métodos de localización probabilística[1]. El algoritmo MCL requiere la

[1] Thrun et al, pág 274.

implementación de los modelos de movimiento y de medición del robot móvil. Una descripción de estos modelos se incluye en la siguiente sección.

Para la verificación de los resultados experimentales se utilizó la plataforma de simulación *Robot Soccer Simulator V1.5a* la cual modela un campo de futbol con dos equipos de robots Mirosot FIRA.

3.2 Modelo de movimiento

La cinemática de robots móviles normalmente se expresa con base en un conjunto de ecuaciones determinísticas[2], las cuales a partir de una velocidad translacional v, una velocidad angular ω y una pose inicial $x_{t-1} = (x \ y \ \theta)$ conocidas, indica la pose $x_t = (x' \ y' \ \theta')$ en la cual se encontrará el robot móvil después de transcurrido un tiempo Δt. Esta situación se presenta en la figura 10.

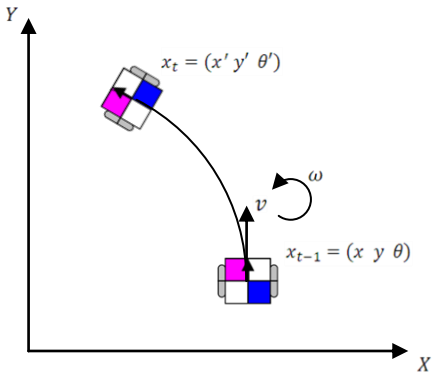


Fig. 10: Trayectoria de un robot móvil que se desplaza con velocidades v y ω constantes.

Sin embargo, esta única pose x_t calculada a partir de estas ecuaciones probablemente no corresponderá con la pose real del robot debido a la inexactitud de los actuadores y por esta razón es necesario cuantificar de alguna manera el error asociado. Esto se logra a través del modelo probabilístico de movimiento, el cual entrega como resultado una distribución de probabilidad $p(x_t|u_t, x_{t-1})$, donde u_t es el vector de control en el instante de tiempo t: $u_t = (v \ \omega)^T$. Esta distribución de probabilidad es aproximada a través de un conjunto finito de muestras aleatorias conocidas como *partículas*.

A continuación se muestra el algoritmo $sample\_motion\_model$[3], el cual dada una pose inicial x_{t-1} y el vector de control u_t aplicado al robot móvil en el tiempo t,

[2] Kim et al, pag 31.
[3] Thrun et al, pág 124.

entrega una partícula, es decir, una pose x_t estimada generada aleatoriamente con base en la distribución $p(x_t|u_t, x_{t-1})$:

sample_motion_model(u_t, x_{t-1}):

$\hat{v} = v + \textbf{sample\_normal\_dist}(\alpha_1 v^2 + \alpha_2 \omega^2)$

$\hat{\omega} = \omega + \textbf{sample\_normal\_dist}(\alpha_3 v^2 + \alpha_4 \omega^2)$

$\hat{\gamma} = \textbf{sample\_normal\_dist}(\alpha_5 v^2 + \alpha_6 \omega^2)$

$x' = x - \frac{\hat{v}}{\hat{\omega}} \sin\theta + \frac{\hat{v}}{\hat{\omega}} \sin(\theta + \hat{\omega}\,\Delta t)$

$y' = y + \frac{\hat{v}}{\hat{\omega}} \cos\theta - \frac{\hat{v}}{\hat{\omega}} \cos(\theta + \hat{\omega}\,\Delta t)$

$\theta' = \theta + \hat{\omega}\,\Delta t + \hat{\gamma}\,\Delta t$

$return\ x_t = (x'\ y'\ \theta')$

En este algoritmo los parámetros $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5$ y α_6 modelan la precisión en los actuadores del robot móvil. Mientras más preciso sea el movimiento del robot más pequeños serán estos valores. El algoritmo $sample\_normal\_dist$ genera muestras aleatorias de una distribución normal (aproximada) con media cero y varianza b^2, de acuerdo con el siguiente algoritmo:

sample_normal_dist(b^2):

$return\ \frac{1}{2}\sum_{i=1}^{12} rand(-b, b)$

Este algoritmo hace uso del teorema del límite central para aproximar una distribución normal a partir de la suma ponderada de variables aleatorias distribuidas uniformemente. En la figura 1 se observa la función de densidad de probabilidad de la suma de 12 variables aleatorias distribuidas uniformemente entre -1 y 1.

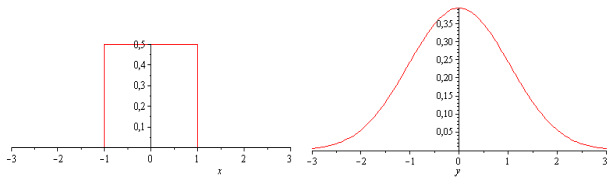


Fig. 11: Obtención de una distribución normal a partir del teorema del límite central.

En la figura 12 se observa el resultado de generar 500 muestras con el algoritmo $sample\_motion\_model$ para una pose x_{t-1} y un vector de control u_t dados. La dispersión de las partículas depende de los parámetros $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6$. Las zonas

con mayor concentración de partículas poseerán mayor probabilidad de ser alcanzadas por el robot cuando ejecute el movimiento. Mientras mayor sea el número de partículas muestreadas mejor será la representación de la distribución de probabilidad $p(x_t|u_t, x_{t-1})$.

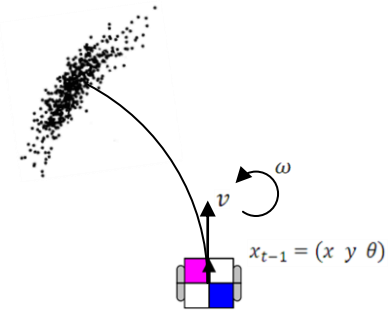


Fig. 12: Partículas generadas por el modelo de movimiento.

3.3 Modelo de Medición

El modelo probabilístico de medición se define como la distribución de probabilidad $p(z_t|x_t, m)$ donde x_t es la pose del robot, m es el mapa del ambiente y z_t es el conjunto de mediciones obtenidas en el tiempo t. Esta distribución indica el grado de verosimilitud que existe entre el conjunto de mediciones reales z_t y las mediciones que se obtendrían cuando el robot se encuentra en x_t dentro de un mapa m conocido.

En el caso particular del presente proyecto, los robots Mirosot no poseen ninguna clase de sensores a bordo, por lo cual fue necesario realizar su simulación. Se modeló un sensor laser telemétrico omnidireccional, el cual mide las distancias a los obstáculos más cercanos cada 4°, obteniendo un total de 90 mediciones (ver figura 13). Este modelo incluye ruido aditivo Gaussiano en las medidas de distancia, asemejándose de esta manera a un sensor real. Cuando no se detecta un obstáculo a una distancia menor al alcance máximo del sensor, éste reporta el valor máximo.

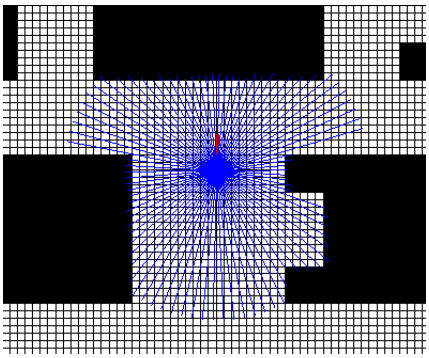


Fig. 13: Mediciones realizadas por el sensor simulado.

Se implementó el algoritmo $beam\_range\_finder\_model$, el cual calcula el grado de verosimilitud[4]:

$beam\_range\_finder\_model(z_t, x_t, m)$:

$q = 1$

$for\ k = 1\ to\ K\ do$

$calcular\ z_t^{k*}\ para\ la\ medición\ z_t^k\ usando\ proyección\ de\ rayo$

$p = z_{hit}\ p_{hit}(z_t^k | x_t, m) + z_{short}\ p_{short}(z_t^k | x_t, m)$

$q = q * p$

$return\ q$

La distribución $p_{hit}(z_t^k | x_t, m)$ modela la k-ésima medición ruidosa z_t^k de la distancia a un obstáculo ubicado a una distancia z_t^{k*}. La distribución $p_{short}(z_t^k | x_t, m)$ modela la medición de obstáculos no presentes en el mapa (por ejemplo personas moviéndose en el mismo ambiente). Las constantes z_{hit} y z_{short} permiten combinar las distribuciones p_{hit} y p_{short}. En el modelo completo normalmente se incluyen las distribuciones p_{max} y p_{rand} las cuales modelan fallas en sensores reales. En este caso, por ser un sensor simulado, se asume que en ningún caso se presentarán fallas y por eso no se incluyen estos términos.

La distribución p_{hit} se definió de la siguiente manera:

$$p_{hit}(z_t^k | x_t, m) = \begin{cases} \eta\ N(z_t^k; z_t^{k*}, \sigma_{hit}^2) & si\ 0 \le z_t^k \le z_{max} \\ 0 & otros\ casos \end{cases}$$

Donde $N(z_t^k; z_t^{k*}, \sigma_{hit}^2)$ es una distribución gaussiana univariada con media z_t^{k*} y varianza σ_{hit}^2. El término η es un normalizador definido como:

[4] Thrun et al, pág 158.

$$\eta = \left(\int_0^{z_{max}} N(z_t^k; z_t^{k*}, \sigma_{hit}^2)\, \mathrm{d}z_t^k \right)^{-1}$$

La distribución p_{short} se definió de la siguiente manera:

$$p_{short}(z_t^k | x_t, m) = \begin{cases} \eta\, \lambda_{short}\, e^{-\lambda_{short}\, z_t^k} & si\ 0 \le z_t^k \le z_t^{k*} \\ 0 & otros\ casos \end{cases}$$

El normalizador η está dado por:

$$\eta = \frac{1}{1 - e^{-\lambda_{short}\, z_t^{k*}}}$$

Al combinar estas distribuciones, se obtiene la distribución $p(z_t^k | x_t, m)$, la cual se representa en la figura 14.

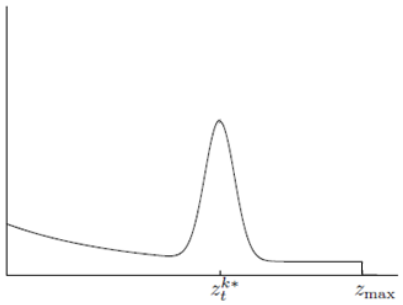


Fig. 14: Distribución combinada.

Finalmente, asumiendo que existe independencia en el ruido de las mediciones, se calcula la distribución deseada como el producto de las distribuciones de las *K* mediciones obtenidas:

$$p(z_t | x_t, m) = \prod_{k=1}^{K} p(z_t^k | x_t, m)$$

3.4 Filtro de Partículas

La Localización Monte Carlo se realiza utilizando el filtro de partículas especificado por el siguiente algoritmo[5]:

$\boldsymbol{Monte\_Carlo\_Localization}(X_{t-1}, u_t, z_t, m)$:
$\bar{X}_t = X_t = \emptyset$
$for\ m = 1\ to\ M\ do$
    $x_t^{[m]} = \boldsymbol{sample\_motion\_model}(u_t, x_{t-1}^{[m]})$
    $w_t^{[m]} = \boldsymbol{beam\_range\_finder\_model}(z_t, x_t^{[m]}, m)$
    $\bar{X}_t = \bar{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
$for\ m = 1\ to\ M\ do$
  $seleccionar\ i\ con\ probabilidad\ \propto\ w_t^{[i]}$
    $agregar\ x_t^{[i]}\ a\ X_t$
$return\ X_t$
$return\ q$

En resumen, el algoritmo toma inicialmente como entradas un conjunto de partículas X_{t-1}, generadas aleatoriamente, donde cada partícula es una hipótesis diferente de la pose del robot en el instante de tiempo $t-1$. Para cada una de las partículas $x_{t-1}^{[m]}$ se aplica el muestreo del modelo de movimiento, obteniendo una nueva partícula $x_t^{[m]}$desplazada de acuerdo con las velocidades dadas por u_t. Luego, para cada una de las nuevas partículas estimadas se calcula un peso $\omega_t^{[m]}$, el cual corresponde a la verosimilitud de la medición real z_t con la medición que se obtendría si el robot se encontrara en la pose dada por la hipótesis $x_t^{[m]}$. Las partículas que se encuentren cercanas a la posición real del robot en general tendrán pesos mayores a otras que se encuentren en otras posiciones. Si el ambiente presenta algún tipo de simetría (corredores o esquinas similares en diferentes lugares), las partículas generadas en dichos lugares tendrán pesos similares.

Posteriormente, se realiza una selección aleatoria de las partículas con probabilidad proporcional a su peso, obteniendo de esta manera las partículas con mayor probabilidad de encontrarse en una pose similar a la pose real del robot. El conjunto de partículas seleccionadas será utilizado como entrada del algoritmo en la siguiente iteración.

El robot continuará estimando las posiciones de las partículas de acuerdo con su modelo de movimiento, calculará la verosimilitud para cada una y las filtrará, obteniendo en cada iteración del algoritmo partículas que indicarán con un mayor grado de certeza su pose estimada.

[5] Thrun et al, pág 252.

3.5 Resultados de la simulación

La plataforma de simulación *Robot Soccer Simulator* (ver figura 15) permite desarrollar software de control para los robots móviles Mirosot – FIRA, el cual luego de algunos ajustes menores puede ser ejecutado en los robots físicos. Sin embargo, por estar enfocado al desarrollo de controladores para el juego de futbol robótico, no es posible modificar el ambiente simulado.

Fig. 15: Entorno de simulación utilizado.

Dado que la presencia de obstáculos en el ambiente es necesaria para que el robot móvil pueda detectarlos con sus sensores y realizar la localización, se modelaron obstáculos virtuales (ver figura 16) que no son visibles durante la simulación pero son detectados por el scanner laser del robot.

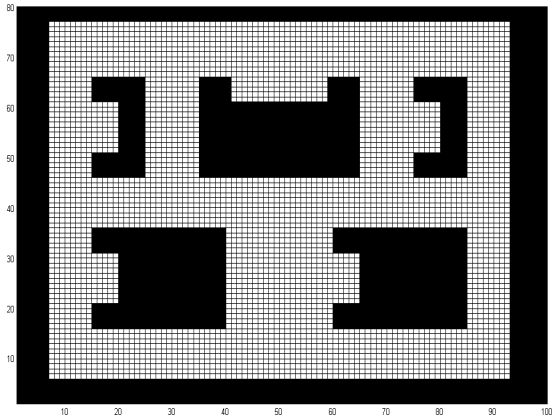


Fig. 16: Ambiente definido para los experimentos incluyendo obstáculos virtuales con formas arbitrarias.

Se utilizó un mando inalámbrico de XBOX 360 para generar el vector de control $u_t = (v\ \omega)^T$ que lleva al robot móvil a recorrer el espacio libre del ambiente.

Además este vector de control se toma como una de las entradas del algoritmo de localización.

En la figura 17 se observan 6 iteraciones sucesivas del algoritmo de localización. Los puntos rojos corresponden a las partículas que se toman como entrada del el algoritmo en esa iteración. Las partículas estimadas por el modelo de movimiento se muestran en color azul. Las partículas seleccionadas como las mejores se muestran en color verde. La ubicación real del robot móvil se encuentra identificada en cada ciclo con un punto negro.

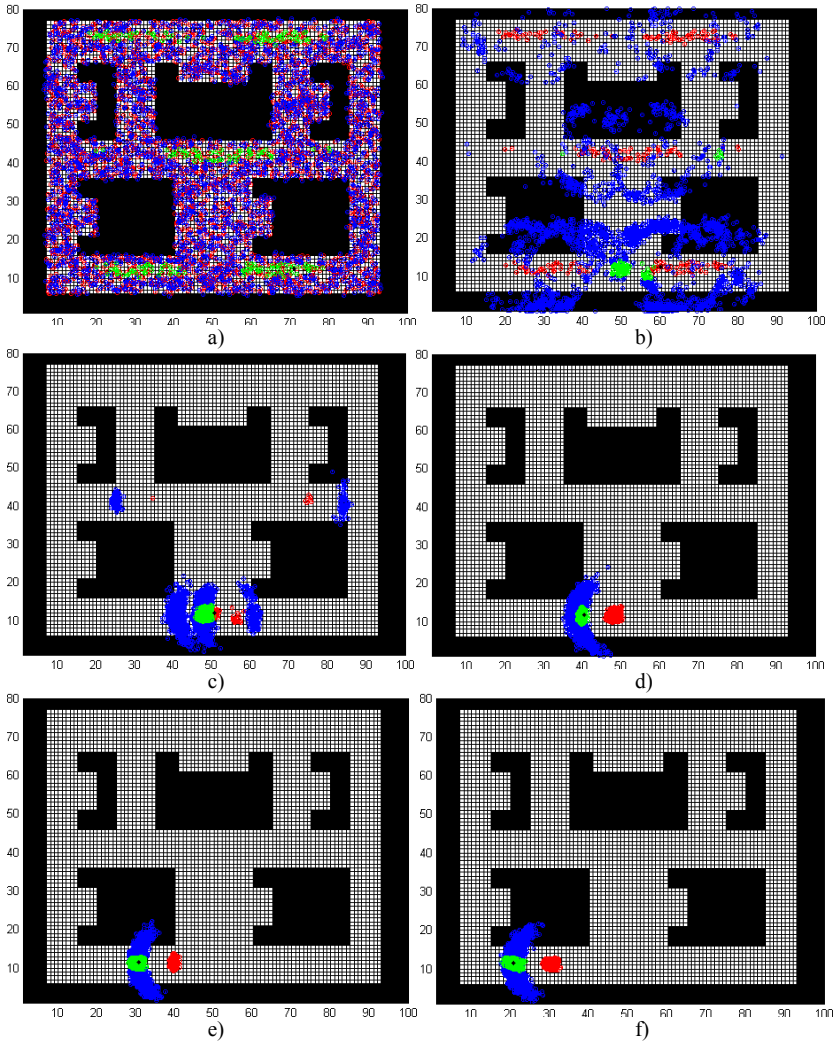


Fig. 17: Partículas generadas durante las primeras 6 iteraciones del algoritmo de localización.

En la primera iteración del algoritmo, se generan partículas aleatorias distribuidas uniformemente en todo el espacio libre. A medida que se realiza un nuevo ciclo, el algoritmo filtra las partículas, manteniendo solamente las correspondientes a las mejores hipótesis para la ubicación actual del robot móvil. En este caso después de 4 ciclos se generan partículas en una sola zona del mapa, indicando una alta probabilidad de que se encuentre en ese lugar. En las siguientes iteraciones del algoritmo, se continúan generando partículas para mantener actualizada la estimación de la localización del robot móvil.

Cuando el robot móvil se desplaza alguna distancia pequeña se espera que no cambien mucho las mediciones obtenidas por los sensores. Por esta razón no es necesario calcular en todos los ciclos del algoritmo la distribución del modelo de medición, disminuyendo así el costo computacional. Sin embargo debido a la inexactitud del modelo de movimiento, el error de localización (distancia entre la ubicación real y la ubicación estimada) tenderá a aumentar mientras no se realice nuevamente una actualización del modelo de medición. En la figura 10 se muestra el error de localización cuando se ejecuta dicha actualización cada 1, 5 y 10 ciclos del algoritmo.

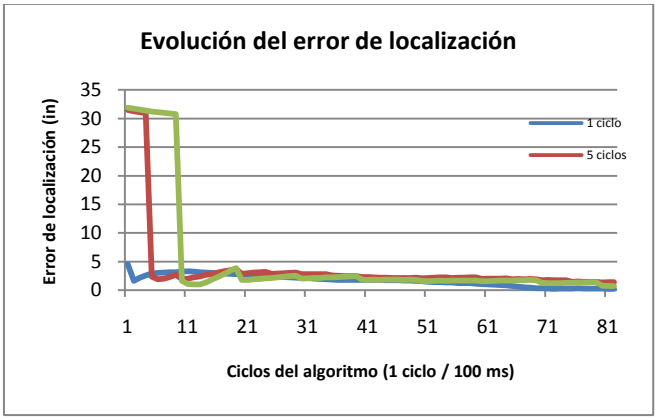


Fig. 18: Evolución del error de localización en el tiempo para diferentes frecuencias de actualización del modelo de medición.

Aquí se puede observar que el algoritmo requiere más ciclos para estimar correctamente la localización del robot cuando se ejecuta la actualización del modelo de medición con menor frecuencia. Cuando se ejecuta la actualización cada ciclo, el error de localización disminuye rápidamente y en general se mantiene en un valor menor al que se obtiene con una frecuencia menor. Aquí el costo computacional será más elevado, pero se comprobó que aún cuando se ejecuta la actualización en cada ciclo, el algoritmo logra producir la estimación de la localización en tiempo real. Para el cálculo de estos promedios se realizaron 10 ensayos para cada frecuencia, los cuales consistieron en recorrer una ruta fija mientras se ejecuta el algoritmo de localización.

3 Red Neuronal Artificial

Para la navegación autónoma del dispositivo en el entorno, se propone la implementación de una red neuronal artificial que le permita al robot evitar obstáculos por medio de la variación de las velocidades angulares ω y lineales v. Las redes neuronales artificiales son el resultado de una compleja operación entre abundantes lazos de realimentación junto con *no linealidades* de los elementos de proceso y cambios adaptativos de sus parámetros, que pueden definir incluso fenómenos dinámicos muy complicados [10].

Además, sus propiedades similares a las del cerebro como el aprendizaje, la adaptación y la generalización, permiten solucionar problemas de forma sencilla en el área de robótica móvil, como la navegación en entornos junto con el problema de localización.

En este proyecto se utilizó una red neuronal *backpropagation* para la navegación autónoma en el entorno (Figura 1). Nuestra red neuronal tiene 90 neuronas en la capa de entrada, 30 en la capa oculta y 2 en la capa de salida. Los datos de las neuronas de la capa de entrada corresponden a la información obtenida por la simulación de un sensor telemétrico y los datos de salida corresponden a la información de la velocidad angular ω y tangencial *v* del robot móvil. Por simplicidad en el proceso, se desarrolló el código de la red neuronal en MATLAB y los resultados obtenidos del entrenamiento y aprendizaje de la red los implementamos en el código de la plataforma FIRA, C++. A continuación se presentan los pasos desarrollados:

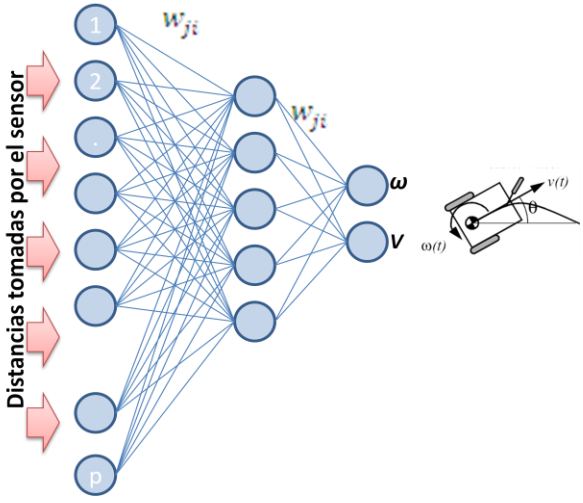


Fig. 19: Implementación y arquitectura de la red neuronal.

1. Se inicializan los pesos de las neuronas de la capa de entrada con valores aleatorios entre 1 y -1. Estos valores se guardan en una matriz *IW* de 30 filas x 90 columnas.

$$IW = \begin{bmatrix} iw_{1,1} & .. & .. \\ .. & .. & .. \\ .. & .. & iw_{30,90} \end{bmatrix}$$

2. Se inicializan de forma aleatoria los valores de los potenciales sinápticos de las neuronas de la capa oculta. Estos valores se guardan en una matriz $b1$ de 30 filas x 1 columna.

$$b1 = \begin{bmatrix} b1_{1,1} \\ .. \\ b1_{1,30} \end{bmatrix}$$

3. Se inicializan de forma aleatoria los valores de los pesos presenten entre la capa oculta y la capa de salida. Estos valores se guardan en una matriz LW de 2 filas x 30 columnas.

$$LW = \begin{bmatrix} lw_{1,1} & .. & .. \\ .. & .. & lw_{2,30} \end{bmatrix}$$

4. Se inicializan de forma aleatoria los valores de los potenciales sinápticos de las neuronas de la capa de salida. Estos valores se guardan en una matriz $b2$ de 2 filas x 1 columna.

$$b2 = \begin{bmatrix} b2_{1,1} \\ b2_{1,2} \end{bmatrix}$$

5. Para el entrenamiento de nuestra red, hemos tomado 800 patrones, donde cada uno corresponde a la información obtenida de las 90 mediciones del sensor en un ángulo de 360 grados, con sus respectivos valores de las velocidades ω y ν del robot.

$$Patrones = \begin{bmatrix} p_{1,1} & .. & .. \\ .. & .. & .. \\ p_{800,90} & & \\ \omega_{1,801} & & \omega_{90,801} \\ \nu_{1,802} & & \nu_{90,802} \end{bmatrix}$$

6. Calculamos las salidas para las neuronas ocultas en función de las neuronas de la capa de entrada. Para cada neurona de la capa oculta:

   a. $(a1)_{pj}^{h} = \frac{1}{\left(1+\exp\left(-\sum_{i=1}^{N} iw_{ji}^{h} x_{pi} + b1\right)\right)}$;

$$(a1)_{1}^{800} = \frac{1}{(1+\exp(-IW*P+b1))}$$

   En donde *h* hace referencia a la magnitud de la capa oculta, el subíndice *p*, al *p-ésimo* vector de entrenamiento, y j a la *j-ésima* neurona oculta.

7. Ahora se hacen los cálculos para obtener las salidas de las neuronas de la capa de salida.

a. $(a2)_{pj}^{h} = \sum_{i=1}^{N} lw_{ji}^{h} a1_{pi} + b2$;

$$(a2)_{1}^{800} = LW * a1 + b2$$

8. Para calcular los términos de error de cada una de las neuronas, desarrollamos los siguientes pasos:

   a. Calculamos el error entre la salida deseada y la obtenida $red2$ para cada una de las parejas de patrones correspondiente.

$$\delta_{pk}^{o} = \left(d_{pk} - y_{pk}\right) f_{k}^{o}(red_{pk}^{o}) \text{ ;}$$

$$E = S * LW * (P_{\omega v} - a2)$$

Donde:

$$S = \begin{bmatrix} s_{1,1} & \cdot\cdot & \cdot\cdot \\ \cdot\cdot & \cdot\cdot & \cdot\cdot \\ \cdot\cdot & \cdot\cdot & s_{30,30} \end{bmatrix}$$

Corresponde a una matriz diagonal

$$S_{i,j} = (1 - a1(i)) * a1(1)$$

d_{pk} , es la salida deseada de la red.

y_{pk} , es la salida obtenida en la red.

$f_{k}^{o}(red_{pk}^{o})$, es la derivada de la salida.

9. Ahora actualizamos los pesos, por medio de un algoritmo recursivo, empezando por las neuronas de la capa de salida y devolviéndonos hasta llegar a la capa de entrada.
   Para las neuronas de la capa de salida aplicamos la siguiente ecuación:

$$w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \Delta w_{kj}^{o}(t+1) \text{ ;}$$

$$\Delta w_{kj}^{o}(t+1) = \alpha\ \delta_{pk}^{o} y_{pj}$$

Para los pesos de la capa oculta:

$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \Delta w_{ji}^{h}(t+1) \text{ ;}$$

$$\Delta w_{ji}^{h}(t+1) = \alpha\ \delta_{pj}^{o} y_{pi}$$

De forma matricial, los pesos de actualizarían de la siguiente manera:

$$IW = IW + \alpha * (P_{\omega v} - a2) * P + \beta * E * P$$

$$b1 = b1 + \alpha * E + \beta * E$$

$$LW = LW + \alpha * E * a1 + \beta * E * a1$$

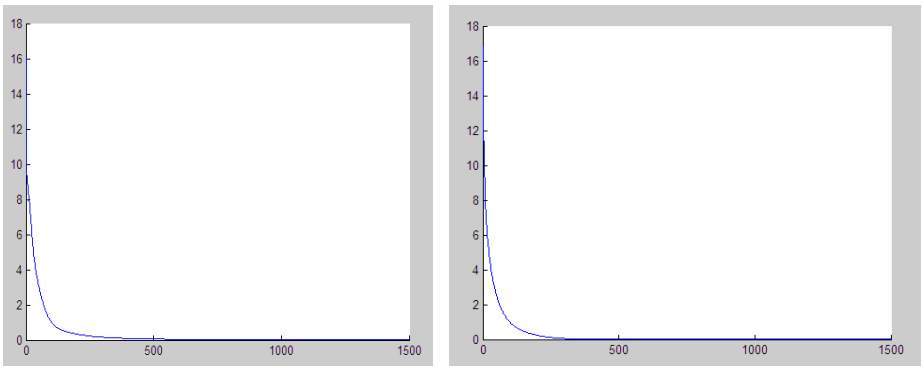
$$b2 = b2 + \alpha * E + \beta * E$$

10. El proceso se repite hasta que la red converge.

$$E_p = \sum_{k=1}^{M} \delta_{pk}^2; E_p = E_p + \sum |E^2|$$

3.1 Resultados del entrenamiento de la RNA

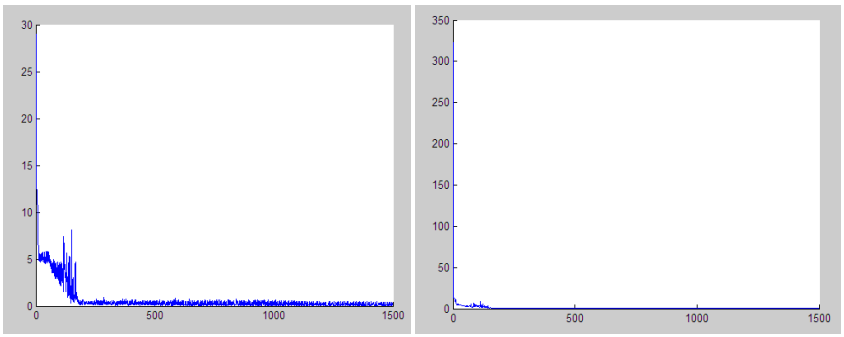
En la siguiente gráfica podemos observar la convergencia de la red neuronal artificial. Este proceso puede tomar mucho tiempo, y depende de la tasa de aprendizaje α (en este caso se utilizaron valores pequeños que oscilan entre 0,25 – 0,50).



a) b)

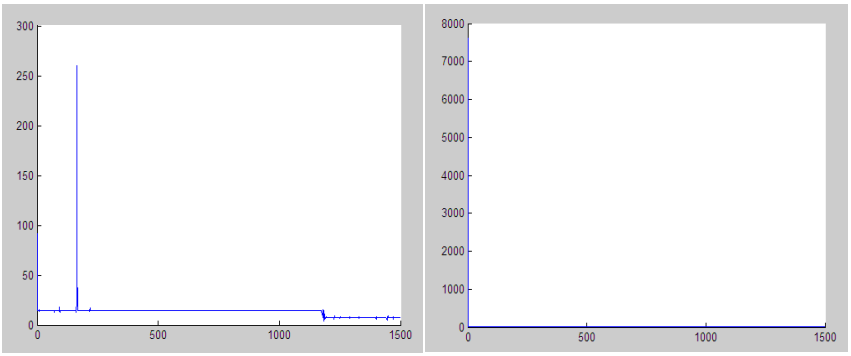
Fig. 20. Error normalizado durante el entrenamiento de la RNA con α=0,025 para a) 6 neuronas en la capa oculta y b) 30 neuronas en la capa oculta.

En las figuras 20 y 21 se observa que con estos parámetros el algoritmo de entrenamiento logra la convergencia del error a un valor mínimo. Sin embargo en la figura 22 con el parámetro de aprendizaje α igual a 1 no se logra dicha convergencia.



a) b)

Fig. 21. Error normalizado durante el entrenamiento de la RNA con α=0,5 para a) 6 neuronas en la capa oculta y b) 30 neuronas en la capa oculta.



a) b)

Fig. 22. Error normalizado durante el entrenamiento de la RNA con α=1 para a) 6 neuronas en la capa oculta y b) 30 neuronas en la capa oculta.

Conclusiones

Se comprobó la eficacia del método PRM implementado en este trabajo para determinar la ruta más corta entre dos puntos que se encuentran en un plano que presenta obstáculos. Es importante resaltar la incidencia de la cantidad de puntos y la condición de visibilidad en el costo computacional para la solución del algoritmo.

Se realizó la implementación de la Localización Monte Carlo (MCL), la cual posee ventajas en cuanto a facilidad de implementación y robustez frente a otros métodos de localización probabilística.

El uso de las redes neuronales artificiales en el proceso de navegación autónoma, es un método apropiado para el aprendizaje de los comportamientos del robot frente a los diferentes estados en los cuales se enfrentara durante la navegación. Sin embargo, la red diseñada no logra controlar apropiadamente el robot móvil para la navegación requerida durante el proceso de localización. Como trabajo futuro se propone implementar otro tipo de red neuronal.

La integración de los métodos de planificación de trayectorias, Montecarlo y redes neuronales artificiales en la navegación autónoma del robot móvil, permitió desarrollar la propuesta de plataforma robótica para la asistencia en la movilidad de invidentes de forma satisfactoria.

Referencias

[1] L. Jaillet. *Path Deformation Roadmaps*. Springer Tracts in Advanced Robotics. Vol 47. 2008.

[2] J. H. Kim, D. H. Kim, Y. J. Kim, K. T. Seow. *Soccer Robotics*. Springer Tracts in Advanced Robotics , Vol. 11. 2004.

[3] R. Johnsonbaugh. *Matemáticas Discretas*. Prentice Hall, 2005.

[4] S. S. Ge, *Autonomous Mobile Robots.* Boca Raton, FL: Taylor & Francis, 2006, pp. 385.

[5] C. Laffra. *Dijkstra's Shortest Path Algorithm*. Java Demo Applet. http://www.dgp.toronto.edu/people/JamesStewart/270/9798s/Laffra/DijkstraApplet.html

[6] López Justicia María Dolores. 2004. Aspectos evolutivos y educativos de la deficiencia visual.

[7] Bruce B. Blasch, William R. Wiener, Richard L. Welsh. Foundations of orientation and mobility. 1997.

[8] Sarmiento, López y otros (2006), La representación espacial en invidentes congénitos, Universidad Pedagógica Nacional.

[9] Harold Leitenberg, Modificación y terapia de conducta, 1982.

[10] Burgard, Wolfram, Dieter Fox, and Sebastian Thrun. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. London: The Mit Press, 2005. Print.