UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

“NAVEGACIÓN AUTÓNOMA DE UN ROBOT MÓVIL USANDO TÉCNICAS PROBABILÍSTICAS DE LOCALIZACIÓN Y MAPEO BASADAS EN MÉTODOS MONTE CARLO SECUENCIALES”

TESIS

PARA OPTAR EL GRADO ACADÉMICO DE MAESTRO EN CIENCIAS CON MENCIÓN EN AUTOMÁTICA E INSTRUMENTACIÓN

ELABORADO POR

IVÁN ARTURO CALLE FLORES

ASESOR

Dr. ARTURO ROJAS MORENO

LIMA – PERÚ

2014

A mis padres Rene Y Fausta, por alentarme siempre a lograr mis más anhelados sueños, y a mis queridos hermanos Marylia y Eduardo.

AGRADECIMIENTO

A todas aquellas personas que me apoyaron en el desarrollo del presente trabajo, mis más sentidas gracias. A mi asesor, el Dr. Arturo Rojas Moreno, el agradecimiento por brindarme su asesoría en el presente trabajo de tesis, y sentar las bases de gran parte de mi conocimiento en lo que se respecta a la teoría de control moderno. A aquellos buenos profesores de la Maestría en Automática e Instrumentación, así como al personal de la oficina de Post Grado de la FIEE-UNI. También a la FIM-UNI por su apoyo en la construcción del robot móvil desarrollado en el presente trabajo.

A aquellos investigadores del área de robótica e inteligencia artificial de diversas partes de mundo, por ser mi principal y casi única guía en la investigación y desarrollo del presente trabajo, el cual, por su novedad y muy reciente desarrollo, aún no cuenta con investigadores peruanos con conocimientos en el mismo. Espero haber abierto el camino para que nuevas generaciones de compatriotas, se animen y desarrollen investigaciones en estos temas tan apasionantes y de relevancia actual como es la navegación autónoma de robots móviles usando técnicas probabilísticas.

Finalmente deseo agradecer a mis padres. Mi padre Rene Calle Zúñiga, por su constante apoyo tanto material como espiritual, y por siempre alentarme a seguir adelante. Mi madre Fausta Flores Salas, por su cariño y afecto, y por siempre estar pendiente de mi familia.

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

El presente trabajo de tesis consiste en el desarrollo e implementación de un robot móvil para tareas de navegación autónoma usando técnicas probabilísticas de localización y mapeo basadas en método de Monte Carlo secuenciales. Estas técnicas, que actualmente representan el estado del arte en la robótica móvil, son las que le permiten a un robot estimar tanto su configuración como el mapa de su entorno de navegación usando la data "ruidosa" de sus sensores y actuadores. De esta manera, estas técnicas le proveen al robot capacidades de navegación autónoma en el sentido que este en todo momento sabe dónde está "localización", y es capaz de construir mapas de su entorno de navegación "mapeo".

Además de implementar los algoritmos de localización y mapeo usando métodos Monte Carlo secuenciales estándar, en la presente tesis también se ha propuesto algunas mejoras a estos algoritmos estándar con el fin de obtener mejores resultados tanto en la estimación de estado como en el tiempo de ejecución. Estas mejoras representan uno de los principales aportes de la presente tesis. Adicionalmente, también se ha implementado un sistema de planeamiento usando el algoritmo A*, con el fin de dotar al robot el nivel de cognición que le permitirán encontrar las trayectorias óptimas con el fin que este llegue a sus metas; y un sistema de control de movimiento basado en el controlador PID, con el fin que el robot sea capaz de seguir las trayectorias planificadas. Estos cuatro componentes son los que hacen que nuestro robot sea autónomo, y por tal, sea capaz de desarrollar tareas al más alto nivel.

Como resultado de este trabajo se tiene el primer robot móvil en nuestro país con la capacidad de desarrollar tareas de navegación autónoma al más alto nivel usando las técnicas del estado del arte del campo de la robótica móvil. Este robot, bautizado como "R2D2-00: El Robot Autónomo", representa el punto de partida en la construcción de robots móviles cada vez más avanzados en nuestro país

Abstract

The present work consists in the development and implementation of a mobile robot for autonomous navigation tasks using probabilistic techniques of localization and mapping based on sequential Monte Carlo methods. These techniques, which currently represent the state-of-the-art, allow a robot to estimate both its configuration and the map of its environment using the noisy measurements of its sensors. Thus, these techniques provide our robot with autonomous navigation capabilities, in the sense that, the robot at any time knows where it is “localization”, and is able to construct a map of its environment “mapping”.

In addition to the implementation of the localization and mapping algorithms using standard Monte Carlo techniques, in the present work it has also been suggested some improvements to these standard algorithms in order to get better estimation results as well as reduced execution times. These improvements are one of the main contributions of the present work. Additionally, it has also been implemented a planning component using the A* algorithm, in order that the robot will be able to find the optimal paths to reach its goals; and a motion control component based on the well-know PID controller so that the robot will be able to follow the path previously defined by its planning component. These four components allow our robot to be considered as autonomous, and able to do tasks at the highest levels.

As a result of the present work, it has been developed the first mobile robot in my country, Peru, with the capacity to do autonomous navigation tasks at the highest level using the state-of-the-art techniques of the field of mobile robotics. This robot, named “R2D2: The Peruvian Autonomous Robot”, represents the point of departure in the development of more advanced mobile robots in my country.

INTRODUCCIÓN

La presente investigación se refiere al diseño e implementación de un robot móvil diseñado para tareas de navegación autónoma en entornos reales. El problema de la navegación autónoma de robots móviles tanto en ambientes cerrados como en exteriores ha recibido considerable atención en los últimos años, debido principalmente a la gran cantidad de aplicaciones que se le pueden dar a estos sistemas.

Para que un robot pueda realizar tareas como transporte de objetos en una casa o industria, búsqueda y rescate en áreas peligrosas o inhóspitas, servicio de guía en museos y hospitales, tareas de monitoreo o supervisión, etc., este necesita tener capacidades de navegación autónoma al más alto nivel (Con el término “al más alto nivel” nos referimos a la capacidad de realizar tareas complejas en entornos reales. A diferencia de por ejemplo los seguidores de línea, robots sumo, etc., los que si bien son autónomos, solo son capaces de realizar tareas simples en entornos altamente controlados). Para tal fin, cualquier robot que se precie de ser autónomo debe tener cuatro componentes: localización, mapeo, planeamiento y control.

En la presente tesis se han implementado estos cuatro componentes, los cuales combinados con los componentes de control de los sensores y actuadores del robot, han dado como resultado el primer robot móvil completamente autónomo desarrollado en nuestro país con la capacidad de desarrollar tareas al más alto nivel. Este robot, bautizado como “R2D2-00: El Robot Autónomo”, ha demostrado con éxito su capacidad de navegación autónoma en diferentes ambientes de la Universidad Nacional de Ingeniería. De esta manera, con el presente trabajo se ha hecho un gran aporte en la investigación y desarrollo en el campo de la robótica móvil en nuestro país.

Tal como se verá en la presente tesis, el uso de las técnicas probabilísticas, especialmente aquellas basadas en los métodos Monte Carlo, permiten el desarrollo de robots con capacidades de navegación autónoma en entornos reales (como los ambientes de la FIM-UNI, o la sección de posgrado de la FIEE-UNI). De esta manera, estas técnicas hacen posible el desarrollo de sistemas robóticos capaces de desarrollar nuevas tareas como: servicio de guía, transporte de objetos, monitoreo, mapeo, etc.

CAPÍTULO I
ESTIMACIÓN DE ESTADO Y EL MÉTODO MONTE CARLO

El objetivo del presente capitulo es la descripción del concepto de estado, estimación de estado y su importancia en la navegación autónoma de un robot móvil. Tanto en la robótica como en muchas otras áreas de la ciencia, la tarea de estimación de estado es un requerimiento esencial con el fin de realizar tareas de control sobre un sistema. Por ejemplo, para que un robot pueda llegar a una posición deseada, el requisito esencial para realizar control es que el robot estime su posición y orientación en cualquier instante de tiempo. Tal como se muestra en este capitulo, el filtro de Bayes es el filtro probabilístico más general y optimo en la tarea de estimación de estado, y las técnicas de Monte Carlo proveen una manera de implementar el filtro de Bayes usando técnicas basadas en muestras o simulaciones.

Se debe resaltar que todas las variables a tratar en el presente trabajo (el estado, los controles, las mediciones de los sensores, etc.) se modelan como variables aleatorias. Por tal motivo, se usaran los conceptos y terminología básica de la teoría de la probabilidad [1], [2]. Además se debe notar que, por motivos de conveniencia, en el caso de variables aleatorias continuas, cuando se usa el término distribución se está haciendo referencia a la función de densidad de probabilidad.

1.1 Estado y Estimación de Estado

Es la colección de todos los aspectos del robot y su entorno que es necesario tomar en cuenta en la tarea de navegación autónoma de un robot (Figura 1.1). Algunos estados son del tipo estático (por ejemplo la posición de las paredes), mientras que otros son del tipo dinámico (configuración del robot, posición de las personas, etc.). Por lo general, en el caso de los robots móviles, se escoge como vector de estado a la configuración del robot (en el problema de localización), y el mapa de su entorno (en el problema del mapeo).

Figura 1.1. Entorno de navegación de un robot (Fuente: Imagen del robot Rhino, Universidad de Bonn, Alemania)

Uno de los objetivos principales de todo robot móvil es la estimación de estado usando las mediciones de sus sensores. Por ejemplo, en la Figura 1.2 podemos ver en acción un algoritmo probabilístico que está estimando tanto la posición del robot (círculo rojo) como el mapa de ocupación de su entorno (las partes en negro representan obstáculos detectados por los sensores, las partes en blanco representan espacios libres).

Figura 1.2. Estimación de estado (Fuente http://robots.stanford.edu/videos.html)

El problema de estimación es muy importante en muchas áreas de la ciencia y la ingeniería, y es comúnmente formulado desde un enfoque probabilístico debido a que tanto las mediciones de los sensores como la dinámica del sistema bajo estudio están sujetos a ruido y/o incertidumbre. El filtro de Bayes que se describe en el subcapítulo 1.3 provee el marco más general en la tarea de estimación de estado usando métodos probabilísticos, y una manera de implementar este filtro es usando los métodos Monte Carlo que se describen en la subcapítulo 1.4. Nótese que a lo largo del presente trabajo el estado será denotado por la variable x, y el valor especifico del estado en el tiempo t estará dado por x_t.

1.2 Interacción de un Robot con su Entorno

Existen dos tipos de interacciones entre un robot y su entorno. Aquellas interacciones en las que el robot cambia el estado del sistema, que se da cuando el robot usa sus actuadores; y aquellas interacciones en las que el robot obtiene información sobre el sistema, que se da cuando el robot usa sus sensores.

1.2.1 **Acciones de Control.** Son las acciones que tienen como objetivo cambiar el estado del sistema (robot + entorno). Para esto el robot ejerce fuerzas sobre su entorno a través de sus actuadores. Un ejemplo de estas acciones es el movimiento de los motores de las ruedas de un robot, el cual tiene como objetivo mover el robot hacia una nueva configuración (Figura 1.3).

Figura 1.3. Acciones de control (Fuente: http://www.robot-electronics.co.uk/)

La acción de control ejecutada en el intervalo $(t-1;t]$ se denota por la variable u_t, y la secuencia de acciones desde t_1 hasta t_2 se describe como:

$$u_{t_1:t_2} = u_{t_1}, u_{t_1+1}, u_{t_1+2}, ..., u_{t_2} \tag{1.1}$$

El efecto de aplicar una acción de control u_t es la transición o cambio de estado del sistema. En general esta transición de estado es el tipo estocástico (debido a perturbaciones o fuentes de ruido no modeladas por el sistema), por lo que la dinámica probabilística de la transición de estado se puede expresar como:

$$p(x_t \mid u_t, x_{t-1}) \tag{1.2}$$

Y recibe el nombre de *función de densidad de probabilidad de transición de estado*, que define la evolución estocástica del estado en función de las acciones de control, y se lee como la probabilidad del estado x_t desde el estado x_{t-1} aplicando las acciones de control u_t. Este modelo es el primer componente que se necesita para realizar la tarea de estimación de estado en cualquier tipo de sistema. En el Capítulo 2 se formulara este modelo en el caso específico del robot móvil implementado en el presente trabajo.

1.2.2 **Acciones de Percepción**. Son las acciones que ejecuta el robot con el fin de obtener información sobre el estado del mundo haciendo uso de sus sensores. Ejemplos de estas acciones son por ejemplo las mediciones de un sensor laser (Figura 1.4), sensor ultrasónico, las imágenes obtenidas por una cámara de video, etc.

La data generada por las acciones de percepción recibe el nombre de mediciones, y en el instante t, las mediciones se denotan por la variable z_t. La secuencia de mediciones desde t_1 hasta t_2 se describe como:

$$z_{t_1:t_2} = z_{t_1}, z_{t_1+1}, z_{t_1+2}, ..., z_{t_2} \tag{1.3}$$

Figura 1.4. Acciones de percepción. (Fuente: Imagen del robot Minerva, Universidad de Carnegie Mellon, Estados Unidos)

El efecto de aplicar una acción de percepción z_t es tratar de obtener información sobre el estado del mundo x_t en un instante de tiempo dado. En general, las mediciones z_t son del tipo estocásticas debido al ruido intrínseco de los sensores así como al ruido debido a perturbaciones externas, por lo que la dinámica probabilística de la generación de mediciones se puede expresar como:

$$p(z_t \mid x_t) \tag{1.4}$$

Y recibe el nombre de *función de densidad de probabilidad de las mediciones,* que define la evolución estocástica de las mediciones en función de estado del sistema, y se lee como la probabilidad de la medición z_t , dado el estado x_t . Este modelo es el segundo componente que se necesita para realizar la tarea de estimación de estado en cualquier tipo de sistema. En el Capítulo 3 de la presente tesis se formulara este modelo en el caso específico del sensor laser que se usara en nuestro robot móvil.

1.3 Filtro de Bayes

El filtro de Bayes es el método más general en la tarea de estimación de estado y es usado ampliamente en muchas áreas de la ciencia y la ingeniería. Algoritmos tales como

el filtro de Kalman [3], modelos ocultos de Markov [4], filtro de partículas [5], etc., se pueden ver como implementaciones de este filtro aplicados a casos específicos. En el área de la robótica móvil, este filtro representa el núcleo de cualquier algoritmo basado en técnicas probabilísticas.

El objetivo del filtro de Bayes es hallar la función de probabilidad posterior sobre el estado usando toda la dada disponible. En el caso específico de un robot móvil, la data disponible consta de la data generada por los sensores y los actuadores $\{z_{1:t}, u_{1:t}\}$, por lo que la función de probabilidad sobre el estado está dada por

$$p(x_t \mid z_{1:t}, u_{1:t}) \tag{1.5}$$

Y se lee como la *función de densidad de probabilidad del estado* x_t *dada la data de los sensores* $z_{1:t}$*, y la data generada por los actuadores* $u_{1:t}$. Una característica del filtro de Bayes es que nos permite calcular la distribución de probabilidad sobre el estado $p(x_t \mid z_{1:t}, u_{1:t})$ en el instante t a partir de la distribución de probabilidad previa $p(x_{t-1} \mid z_{1:t-1}, u_{1:t-1})$ en el instante $t-1$ mediante la siguiente ecuación:

$$p(x_t \mid z_{1:t}, u_{1:t}) = \eta \cdot p(z_t \mid x_t) \int p(x_t \mid u_t, x_{t-1}) p(x_{t-1} \mid z_{1:t-1}, u_{1:t-1}) dx_{t-1} \tag{1.6}$$

Es decir, el filtro de Bayes es del tipo recursivo, y es esta propiedad lo que lo hace adecuado para una serie de tareas de estimación de estado en tiempo real. Debe notarse la aparición de los términos $p(z_t \mid x_t)$ y $p(x_t \mid u_t, x_{t-1})$ que representan la función de densidad de probabilidad de las mediciones y la función de densidad de probabilidad de transición de estados anteriormente descritos.

Para la validación del filtro de Bayes, el único requisito es que el sistema bajo análisis cumpla con la *propiedad de Markov*. Esta propiedad establece que si se conoce el estado actual del sistema, el futuro es independiente del pasado. Esta propiedad, que también es comúnmente usada en la teoría de control, en el caso de los robots móviles solo se cumple cuando el entorno de navegación es del tipo estático. Debido a que los ambientes de navegación del robot son dinámicos esta condición no se satisface. Sin

embargo, en la práctica las diversas implementaciones del filtro de Bayes han demostrado un buen desempeño en entornos reales [6], [7], [8].

1.3.1 Derivación Matemática

A continuación se mostrara como se obtiene la ecuación (1.6) que describe el filtro de Bayes. El primer paso consiste en aplicar el teorema de Bayes a la densidad de probabilidad posterior $p(x_t \mid z_{1:t}, u_{1:t})$ de donde se tiene:

$$p(x_t \mid z_{1:t}, u_{1:t}) = \eta \cdot p(z_t \mid x_t, z_{1:t-1}, u_{1:t})\, p(x_t \mid z_{1:t-1}, u_{1:t}) \tag{1.7}$$

Donde η es el factor de normalización. Ahora, aplicando la propiedad de Markov que expresa el hecho que las mediciones son independientes de otras variables si se conoce el estado actual $p(z_t \mid x_t, z_{1:t-1}, u_{1:t}) = p(z_t \mid x_t)$ se tiene:

$$p(x_t \mid z_{1:t}, u_{1:t}) = \eta \cdot p(z_t \mid x_t)\, p(x_t \mid z_{1:t-1}, u_{1:t}) \tag{1.8}$$

Si ahora aplicamos las leyes probabilísticas de la suma y el producto al término $p(x_t \mid z_{1:t-1}, u_{1:t})$ tenemos:

$$p(x_t \mid z_{1:t}, u_{1:t}) = \eta \cdot p(z_t \mid x_t) \int p(x_t \mid x_{t-1}, z_{1:t-1}, u_{1:t})\, p(x_{t-1} \mid z_{1:t-1}, u_{1:t}) dx_{t-1} \tag{1.9}$$

Finalmente, aplicando la propiedad de Markov a los términos $p(x_t \mid x_{t-1}, z_{1:t-1}, u_{1:t})$ y $p(x_{t-1} \mid z_{1:t-1}, u_{1:t})$ se completa la derivación del filtro de Bayes:

$$p(x_t \mid z_{1:t}, u_{1:t}) = \eta \cdot p(z_t \mid x_t) \int p(x_t \mid x_{t-1}, u_t)\, p(x_{t-1} \mid z_{1:t-1}, u_{1:t-1}) dx_{t-1} \tag{1.10}$$

Nótese que el único requisito para hacer la derivación del filtro de Bayes, es que el sistema bajo estudio cumpla la propiedad de Markov. En el caso que el estado x_t corresponda a una variable discreta la integral de la ecuación (1.10) se reemplaza por una sumatoria.

1.3.2 Implementaciones más Comunes

El filtro de Bayes descrito en la ecuación (1.6) es la solución más general al problema de estimación de estado. Dependiendo del tipo de problema o sistema a analizar, la implementación de este filtro se puede hacer de manera analítica o de manera aproximada. A continuación se muestra de manera resumida algunas de las implementaciones más comunes del filtro de Bayes en el caso de la robótica móvil.

a) Filtro de Kalman. El filtro de Kalman es la solución óptima del filtro de Bayes en el caso de sistemas gausianos lineales. Es decir aquellos sistemas en donde la transición de estado y de medidas están dadas por ecuaciones de la forma:

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t \tag{1.11}$$

$$z_t = C_t x_t + \delta_t \tag{1.12}$$

Donde las matrices A, B y C describen la dinámica lineal de la transición de estado y la generación de mediciones; el término ε_t representa el ruido gaussiano con media cero y covarianza R_t en la transición de estados; y el término δ_t representan el ruido representa el ruido gaussiano con media cero y covarianza Q_t en las mediciones.

Entonces, bajo estas condiciones, la función de densidad de probabilidad de transición de estado, y de las mediciones está dado por:

$$p(x_t \mid u_t, x_{t-1}) = N\left(x_t; A_t x_{t-1} + B_t u_t, R_t\right) \tag{1.13}$$

$$p(z_t \mid x_t) = N\left(z_t; C_t x_t, Q_t\right) \tag{1.14}$$

Donde $N\left(x; \mu, P\right)$ representa la distribución de probabilidad gausiana de la variable x con media μ y covarianza P. Por otro lado, si se asume que la distribución inicial sobre el estado está dada por una distribución gaussiana, se cumple que la distribución posterior $p(x_t \mid z_{1:t}, u_{1:t})$ sobre el estado en cada instante de tiempo t, está dada por una distribución gaussiana con media μ_t y covarianza Σ_t.

$$p(x_t \mid z_{1:t}, u_{1:t}) = \frac{1}{\left|2\pi\Sigma_t\right|^{1/2}} \exp\left(-\frac{1}{2}\left(x_t - \mu_t\right)^T \Sigma_t^{-1} \left(x_t - \mu_t\right) \right) \tag{1.15}$$

El filtro de Kalman que se muestra en la Tabla 1.1 nos permite calcular en cualquier instante de tiempo los parámetros μ_t y Σ_t de la distribución posterior sobre el estado. Entre las líneas L1y L2 se realiza el paso de predicción, y en las líneas L3 y L5 se realiza el paso de corrección. Se debe notar que este filtro representa la solución más óptima en el caso de sistemas gausianos lineales.

Tabla. 1.1. Filtro de Kalman (Fuente http://www.probabilistic-robotics.org/))

$Kalman \quad filter(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$
$L1: \ \bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
$L2: \ \bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
$L3: \ K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
$L4: \ \mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
$L5: \ \Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
$return \quad \mu_t, \Sigma_t$

b) Filtro extendido de Kalman. El problema principal del filtro de Kalman es que solo es aplicable a sistemas gausianos lineales, mientras que la mayoría de robots y sensores que se usan en la robótica móvil son del tipo no lineal. El filtro extendido de Kalman, es la versión no lineal del filtro de Kalman aplicados a sistemas gausianos no lineales. Es decir aquellos sistemas en donde la transición de estado y de medidas están dadas por ecuaciones de la forma:

$$x_t = g(u_t, x_{t-1}) + \varepsilon_t \tag{1.16}$$

$$z_t = h(x_t) + \delta_t \tag{1.17}$$

Donde las funciones $g(\cdot)$ y $h(\cdot)$ describen la dinámica no lineal de la transición de estado, y la generación de mediciones; el término ε_t representa el ruido gaussiano con media cero y covarianza R_t en la transición de estados; y el término δ_t

representan el ruido representa el ruido gaussiano con media cero y covarianza Q_t en las medidas.

La idea básica de este algoritmo, es linealizar estas ecuaciones con el fin de aplicar el filtro de Kalman, lo que se traduce en el hecho que las distribuciones posteriores sobre el estado se siguen representando por distribuciones gaussianas. En el caso del filtro de extendido de Kalman esta linealización se hace usando la serie de Taylor de primer orden de donde se obtienen las aproximaciones (1.18) y (1.19). Nótese que la linealización se hace alrededor del estado de mayor probabilidad.

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}} (x_{t-1} - \mu_{t-1})$$
$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t\ (x_{t-1} - \mu_{t-1}) \tag{1.18}$$

$$h(x_t) \approx h(\bar{\mu}_t) + \frac{\partial h(\bar{\mu}_t)}{\partial x_t} (x_t - \bar{\mu}_t)$$
$$h(x_t) \approx h(\bar{\mu}_t) + H_t\ (x_t - \bar{\mu}_t) \tag{1.19}$$

Donde las matrices G_t y H_t son las matrices Jacobianas de las ecuaciones de transición de estados y de mediciones. Entonces, reemplazando las ecuaciones (1.18), (1.19) en el filtro de Bayes (1.6), se tiene el filtro extendido de Kalman que se muestra en la Tabla 1.2.

Tabla. 1.2. Filtro extendido de Kalman (Fuente: http://www.probabilistic-robotics.org/)

$Extended\ \ Kalman\ \ filter(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$
$L1:\ \bar{\mu}_t = g(u_t, \mu_{t-1})$
$L2:\ \bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
$L3:\ K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
$L4:\ \mu_t = \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t))$
$L5:\ \Sigma_t = (I - K_t H_t) \bar{\Sigma}_t$
$return\ \ \mu_t, \Sigma_t$

Este filtro ha sido, y es, extensivamente usado en la robótica móvil tanto en tareas de localización y mapeo [9], [10]. Sin embargo, la desventaja principal de este algoritmo es que no puede trabajar con los sensores laser y ultrasónicos debido a

que estos presentan un ruido no gausiano. Además no puede solucionar el problema de localización global que se da cuando un robot no tiene ninguna información sobre su posición inicial [11], [12].

c) **Métodos basados en celdas.** Los métodos basados en celdas, que proveen una solución óptima del filtro de Bayes en el caso de sistemas discretos, se han usado en la robótica móvil mediante la discretización del espacio de estados continuo usando histogramas. Esta idea se puede apreciar en la Figura 1.5, en donde se muestra la discretización del estado de un robot de tipo 1D.

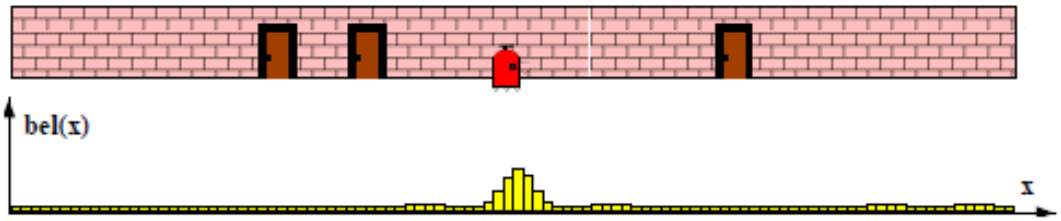


Figura 1.5. Discretización del espacio continúo de un robot en 1D (Fuente: http://www.probabilistic-robotics.org/)

Si bien estas técnicas no presentan las restricciones del filtro de Kalman, su principal inconveniente es la cantidad de memoria y carga computacional requeridas para actualizar la distribución posterior $p(x_t \mid z_{1:t}, u_{1:t})$ [13], [14]. Por este motivo, estas técnicas hacen difícil la operación del robot en tiempo real. Nótese que en el caso del robot diferencial las celdas de discretización están dadas por una matriz de tres dimensiones (Figura 1.6)

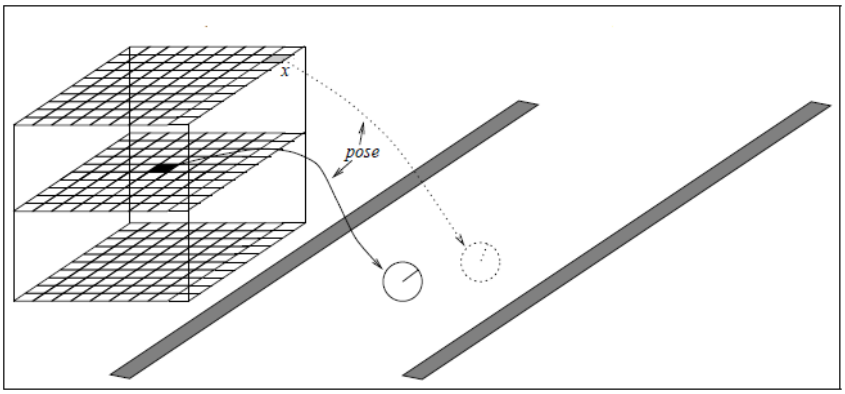


Figura 1.6. Discretización del espacio continúo del robot diferencial (Fuente: http://www.probabilistic-robotics.org/)

1.4 El Método Monte Carlo

Tal como se vio en la sección anterior, debido al problema de la no linealidad de la mayoría de sistemas no es posible hallar de manera exacta la distribución de probabilidad posterior por lo que es necesario hacer una serie de aproximaciones. El método Monte Carlo [15] es un método numérico para implementar el filtro de Bayes sin la necesidad de hacer algún tipo de linealización; lo cual se consigue usando técnicas de muestreo numérico o simulaciones.

La idea principal de los métodos Monte Carlo es la representación de la distribución de probabilidad posterior mediante una serie de muestras, cada una de ellas con un peso o factor de importancia. En base a estas muestras podemos calcular la distribución de probabilidad las mismas usando métodos de estimación de la densidad, y además podemos hallar algunas cantidades estadísticas de interés como la esperanza. A. continuación se hará una breve introducción de estos conceptos los cuales se usaran más adelante en los capítulos localización y mapeo.

1.4.1 Conceptos Básicos

La idea de los métodos Monte Carlo es representar la distribución posterior por una serie de muestras y sus pesos, es decir esta distribución se aproxima como:

$$p(x_{0:t} \mid z_{1:t}, u_{1:t}) \approx \left\{ x_{0:t}^{(m)}, w_t^{(m)} \right\}_{m=1,\ldots,M} \tag{1.20}$$

Donde cada $x_{0:t}^{(m)}$ es una muestra o hipótesis del estado hasta el tiempo t, y los parámetros $w^{(m)}$ son los pesos o factores de importancia de cada muestra. Los factores de importancia, como su nombre sugiere, determinan la importancia de cada muestra en la tarea de estimación de estado. Entonces, la distribución de probabilidad posterior en base a las muestras está dada por:

$$p(x_{0:t} \mid z_{1:t}, u_{1:t}) \approx \sum_{m=1}^{M} w_t^{(m)} \delta(x_{0:t} - x_{0:t}^{(m)}) \tag{1.21}$$

Que es una aproximación discreta de la distribución de probabilidad del estado. Además, en base a estas muestras también es posible hallar la esperanza de cualquier función f con respecto a la distribución $p(x_{0:t} \mid z_{1:t}, u_{1:t})$ como:

$$E[f] \approx \frac{1}{M} \sum_{i=1}^{M} f(x_{0:t}^{(m)}) \tag{1.22}$$

Conforme se incremente el número de muestras, el error de estas aproximaciones tendera a cero. Entonces, si bien se puede aproximar una distribución usando muestras de la misma, la pregunta es ¿de dónde consigo estas muestras? El algoritmo SIS, que se muestra a continuación provee el método más general para la solución de este problema. Este algoritmo forma la base de todos los métodos Monte Carlo secuenciales que se usaran en el presente trabajo.

1.4.2 El Algoritmo SIS (Sequential Importance Sampling)

La idea básica del algoritmo SIS [16], [17] es obtener muestras de una distribución a partir de otra distribución de la cual es fácil obtener muestras. Para ilustrar este concepto, supongamos que deseamos aproximar una función de probabilidad $f(\cdot)$ a partir de sus muestras, pero lo único que tenemos a mano son muestras $x^i \sim g(x),\ i = 1, ..., N_s$, de una función $g(\cdot)$ llamada distribución propuesta (Figura 1.7).

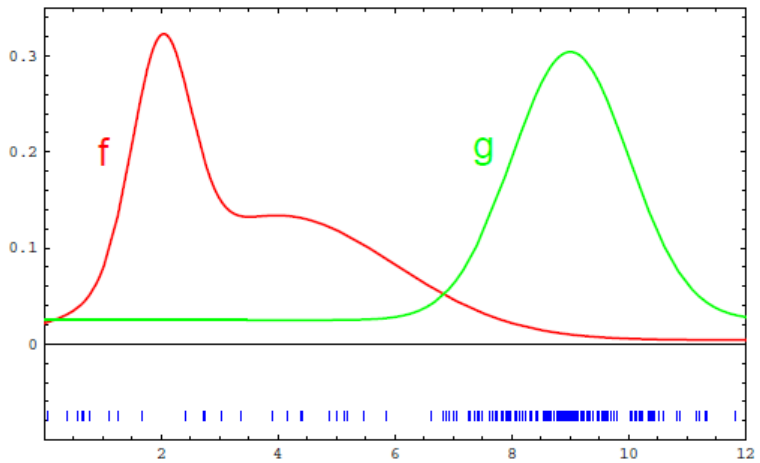


Figura 1.7. Muestras de la distribución propuesta (Fuente: http://www.probabilistic-robotics.org/

Se puede demostrar que una muestra y^i de la función $f(\cdot)$ se obtiene a partir de la muestra x^i de la función $g(x)$ simplemente adjuntando el siguiente peso a dicha muestra:

$$w^i = \frac{f(x^i)}{g(x^i)} \tag{1.23}$$

Este resultado se puede ver en la Figura 1.8 en donde se aprecia que las muestras ubicadas en regiones de mayor probabilidad de $f(\cdot)$ reciben mayor peso, lo que los hace más importantes a la hora de estimar dicha densidad. De donde se tiene que las muestras y^i acompañadas de sus pesos son muestras de la distribución f .

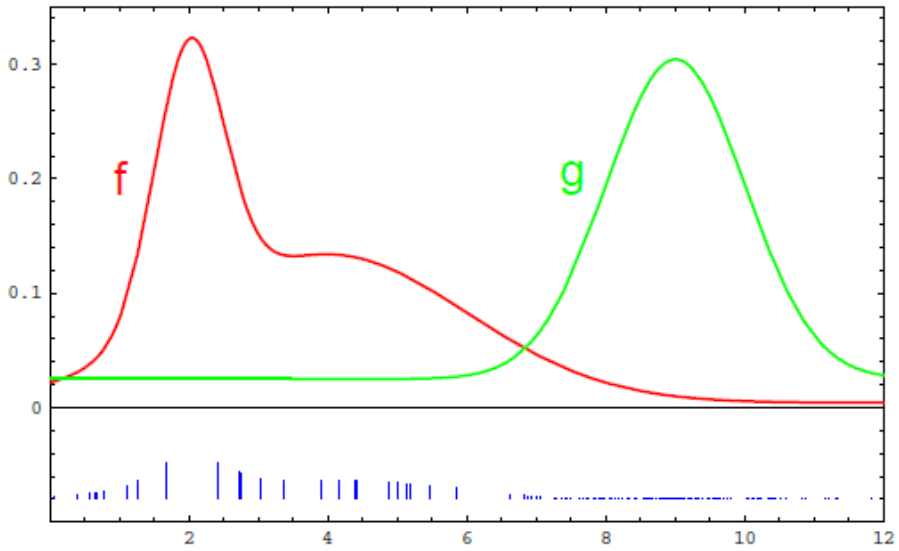


Figura 1.8. Muestras de la distribución deseada (Fuente: http://www.probabilistic-robotics.org/)

Retornando al caso de la estimación de la distribución posterior $p(x_{0:t} \mid z_{1:t}, u_{1:t})$, si suponemos que tenemos muestras de una distribución propuesta $q(x_{0:t} \mid z_{1:t}, u_{1:t})$, los pesos de las muestras estarían dadas por:

$$w_t^i = \frac{p(x_{0:t}^i \mid z_{1:t}, u_{1:t})}{q(x_{0:t}^i \mid z_{1:t}, u_{1:t})} \tag{1.24}$$

Si además, la distribución propuesta se puede descomponer como:

$$q(x_{0:t} \mid z_{1:t}, u_{1:t}) = q(x_t \mid x_{0:t-1}, z_{1:t}, u_{1:t}) q(x_{0:t-1} \mid z_{1:t-1}, u_{1:t-1}) \tag{1.25}$$

Entonces uno puede obtener muestras $x_{0:t}^i \sim q(x_{0:t} \mid z_{1:t}, u_{1:t})$ aumentando cada una de las muestras del conjunto anterior $x_{0:t-1}^i \sim q(x_{0:t-1} \mid z_{1:t-1}, u_{1:t-1})$ con la nueva muestra $x_t^i \sim q(x_t \mid x_{0:t-1}^i, z_{1:t}, u_{1:t})$. Para derivar una expresión matemática para la actualización de pesos (2.24), es conveniente hacer las siguientes operaciones

$$\begin{aligned}
p(x_{0:t} \mid z_{1:t}, u_{1:t}) &= \eta\, p(z_t \mid x_{0:t}, z_{1:t-1}, u_{1:t})\, p(x_{0:t} \mid z_{1:t-1}, u_{1:t}) \\
&= \eta\, p(z_t \mid x_t)\, p(x_{0:t} \mid z_{1:t-1}, u_{1:t}) \\
&= \eta\, p(z_t \mid x_t)\, p(x_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t})\, p(x_{0:t-1} \mid z_{1:t-1}, u_{1:t}) \\
&= \eta\, p(z_t \mid x_t)\, p(x_t \mid x_{t-1}, u_t)\, p(x_{0:t-1} \mid z_{1:t-1}, u_{1:t}) \\
&= \eta\, p(z_t \mid x_t)\, p(x_t \mid x_{t-1}, u_t)\, p(x_{0:t-1} \mid z_{1:t-1}, u_{1:t-1})
\end{aligned} \tag{1.26}$$

Reemplazando las ecuaciones (1.26) y (1.25) en la expresión (1.24) se tiene la siguiente relación:

$$\begin{aligned}
w_t^i &\propto \frac{p(z_t \mid x_t^i)\, p(x_t^i \mid x_{t-1}^i, u_t)\, p(x_{0:t-1}^i \mid z_{1:t-1}, u_{1:t-1})}{q(x_t^i \mid x_{0:t-1}^i, z_{1:t}, u_{1:t})\, q(x_{0:t-1}^i \mid z_{1:t-1}, u_{1:t-1})} \\
w_t^i &\propto \frac{p(z_t \mid x_t^i)\, p(x_t^i \mid x_{t-1}^i, u_t)}{q(x_t^i \mid x_{0:t-1}^i, z_{1:t}, u_{1:t})}\, w_{t-1}^i
\end{aligned} \tag{1.27}$$

Que representa una ecuación recursiva de actualización de pesos. Si además se tiene $q(x_t \mid x_{0:t-1}, z_{1:t}, u_{1:t}) = q(x_t \mid x_{t-1}, u_t)$, que representa la asunción de Markov, se tiene la siguiente ecuación de actualización de pesos:

$$w_t^i \propto \frac{p(z_t \mid x_t^i)\, p(x_t^i \mid x_{t-1}^i, u_t)}{q(x_t^i \mid x_{t-1}^i, u_t)}\, w_{t-1}^i \tag{1.28}$$

Donde se observa claramente que solo se necesita almacenar en memoria, las variables x_t^i, z_t y u_t para el cálculo del peso w_t^i de cualquier muestra. Este hecho permite la implementación de este algoritmo en tiempo real ya que solo se necesita las

mediciones actuales de los sensores. Por otro lado, la función de densidad de probabilidad sobre el estado actual en el tiempo t se puede aproximar como:

$$p(x_t \mid z_{1:t}, u_{1:t}) \approx \sum_{m=1}^{M} w_t^{(m)} \delta\left(x_t - x_t^m\right) \tag{1.29}$$

Se puede probar que si el número de muestras M es muy grande ($M \to \infty$), la aproximación de la distribución posterior (1.29) se acerca a la verdadera distribución posterior $p(x_t \mid z_{1:t}, u_{1:t})$.

En resumen, el algoritmo SIS consiste de una propagación recursiva de los pesos y de las muestras conforme se actualizan las mediciones de los sensores z_t y actuadores u_t. Este algoritmo se muestra en la Tabla 1.4

Tabla. 1.4. Filtro SIS (Fuente: http://www.probabilistic-robotics.org/)

$Algoritmo\ \ SIS(\{x_{t-1}^i, w_{t-1}^i\}, u_t, z_t)$
$L1:\ \ for\ \ i = 1:M$
$L2:\quad x_t^i \sim q(x_t \mid x_{t-1}^i, z_t, u_t)$
$L3:\quad w_t^i = \dfrac{p(z_t \mid x_t^i)\, p(x_t^i \mid x_{t-1}^i, u_t)}{q(x_t^i \mid x_{t-1}^i, z_t, u_t)}$
$L4:\ end$
$return\ \ \{x_t^m, w_t^m\}$

1.4.3 El Problema de la Degeneración

El algoritmo SIS presentado en la sección anterior es el núcleo principal de los métodos Monte Carlo secuenciales. Sin embargo, está sujeto al problema de la degeneración, que se refiere al hecho que después de algunas iteraciones, la mayor parte de las muestras (sino todas) tienen un peso o importancia muy pequeño. Esto implica que estas muestras no son representativas de la función de probabilidad que quieren aproximar, por lo que la calidad de la aproximación se vuelve pésima. Una medida adecuada de este fenómeno se obtiene usando el tamaño efectivo de las muestras que se define como:

$$N_{eff} = \frac{N}{1 + \mathrm{var}(w_k^{*i})} \tag{1.30}$$

Donde $w_k^{*i} = p(x_t^i \mid z_{1:t}, u_{1:t}) / q(x_t^i \mid x_{t-1}^i, u_t)$ es el peso verdadero de la muestra, y N es el número de muestras. Debido a que esta expresión no puede evaluarse directamente, una aproximación de la ecuación (1.30) puede obtenerse como:

$$N_{eff} \approx \frac{1}{\sum_{i=1}^{Ns} \left(\tilde{w}^{[i]} \right)^2} \tag{1.31}$$

Donde $\tilde{w}^{[i]}$ representa los pesos normalizados. Entonces si se tiene $N_{eff} \leq N$, esto indica un problema severo de degeneración. Existen dos maneras de superar este problema. a) Usando una gran cantidad de muestras, lo cual es impráctico debido a que no se tiene infinitos recursos computacionales, b) Escogiendo una adecuada distribución propuesta, y c) Mediante el uso de técnicas de re muestreo. Estas dos últimas alternativas se describen brevemente a continuación.

a) Elección de la función de densidad propuesta. Con el fin de evitar el problema de degeneración, una opción es escoger la función de densidad propuesta de tal manera que se aproxime lo mejor posible a la distribución deseada. El problema con esta elección es que en la práctica es muy difícil conseguir una función propuesta que sea lo suficientemente buena.

b) Re-muestreo. La segunda manera de evitar el problema de degeneración es mediante el uso de técnicas de re-muestreo. La idea de estas técnicas es eliminar aquellas muestras que tienen poco peso y concentrarnos en aquellas que tienen gran peso. De esta manera el conjunto de muestras resultante aproxima de mejor manera la distribución deseada.

El paso de muestreo consiste en la generación (mediante un proceso de re-muestreo) de un nuevo conjunto de muestras $\left\{ x_k^{(m*)} \right\}_{m=1}^{N}$ a partir de las muestras que aproximan la distribución de la forma siguiente:

$$p(x_t \mid z_{1:t}, u_{1:t}) \approx \sum_{m=1}^{M} w_t^{(m)} \delta(x_t - x_t^{(m)}) \tag{1.32}$$

De tal manera que después del re-muestreo todas las muestras tengan igual peso $w_t^{(m)} = 1/M$; y lo más importante, que la mayor cantidad de muestras este en las regiones de mayor probabilidad.

CAPÍTULO II
CINEMÁTICA PROBABILÍSTICA DEL ROBOT MÓVIL

El objetivo de este Capítulo es la descripción cinemática del movimiento del robot móvil implementado en el presente trabajo desde un enfoque probabilístico. Es decir, la obtención del modelo probabilístico de movimiento del robot. La importancia de dicho modelo es que no solo permite describir con más exactitud el movimiento real "ruidoso" de nuestro robot, sino que es un componente principal en la formulación de los algoritmos de Localización y Mapeo. Nótese que este tipo de modelos probabilísticos son los que se usan en los algoritmos que representan el estado del arte en la robótica.

2.1 Configuración Cinemática

La cinemática describe el efecto de las acciones de control en la configuración de un robot. En el caso de los robots móviles que operan en ambientes planos, el estado cinemático, más conocido como configuración, está dado por la locación y orientación del robot (Figura 2.1.) en un sistema coordenado.

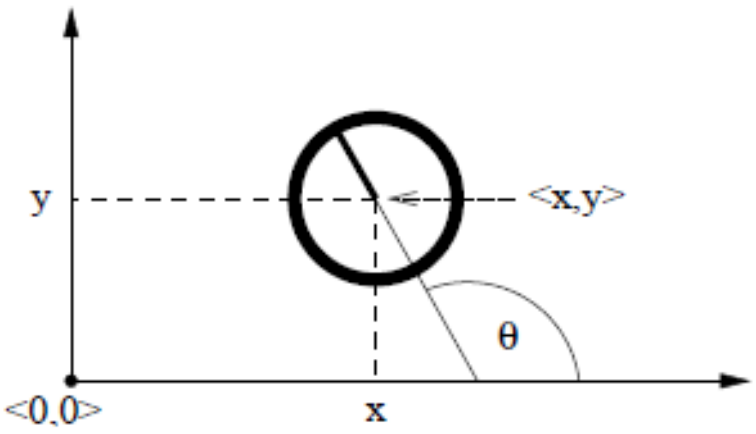


Figura 2.1. Estado o configuración de un robot móvil que opera en terrenos planos (Fuente: http://www.probabilistic-robotics.org/)

La locación del robot está dada por las variables x e y que definen sus coordenadas respecto de un sistema de coordenadas global, y su orientación está dado por la variable θ que se mide respecto al eje X del sistema global. Entonces la configuración o estado del robot está dada por el vector:

$$x = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{2.1}$$

2.2 Cinemática Probabilística

La cinemática probabilística de un robot móvil busca la descripción probabilística del movimiento del robot usando funciones de probabilidad [18], [19]. Tal como se mencionó en el Capítulo 1, en el campo de la robótica móvil es imposible predecir o modelar con gran precisión el movimiento del robot debido a las diversas fuentes de ruido (deslizamiento de las ruedas, mal diseño mecánico del robot, pisos resbaladizos, resolución limitada de los encoders, etc.); por tales motivos es más conveniente describir el movimiento del robot usando modelos probabilísticos.

Tal como se mostró en el capítulo 1, el modelo probabilístico de transición de estado que describe el movimiento de un robot está dado por la siguiente función de densidad de probabilidad:

$$p(x_t \mid u_t, x_{t-1}) \tag{2.2}$$

Que se lee como: "la probabilidad del estado x_t dado el estado inicial x_{t-1}, y la acción de control u_t". Con el fin de ilustrar este concepto, en la Figura 2.2 se muestra la función de densidad de probabilidad de transición de estado para el caso de un robot móvil que se mueve con una velocidad constante a lo largo del eje x. En este ejemplo, la posición inicial x_{t-1} del robot es el origen de coordenadas, la velocidad aplicada u_t es de 1 m/s, y el tiempo de movimiento es de 4 segundos. Nótese que bajo estas condiciones la posición final "ideal" estaría dada por $x_t = x_{t-1} + v \cdot \Delta t = 4m$.

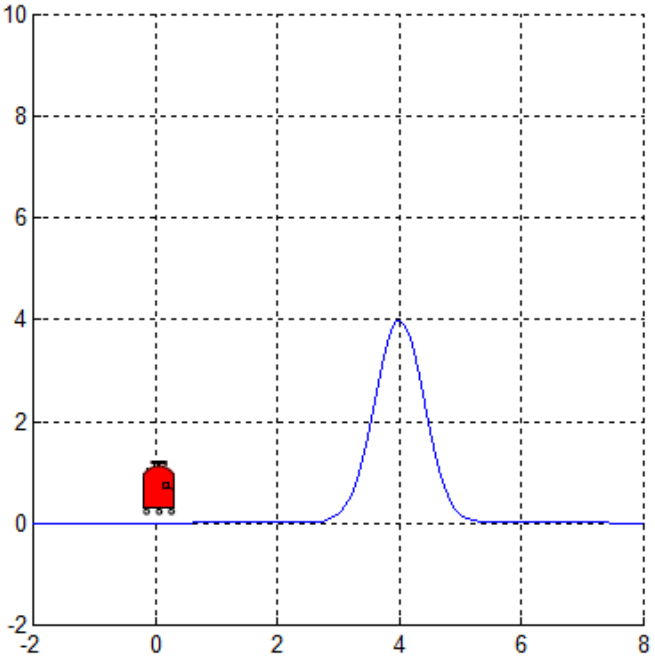


Figura 2.2. Función de densidad de probabilidad de transición de estado de un robot que se mueve en 1D (Fuente: Elaboración propia)

Como puede notarse, la función de densidad de probabilidad de transición de estado asigna a cada posible posición un valor de probabilidad de ser el siguiente estado. En este caso el punto $x_t = 4$ recibe una mayor probabilidad debida a que bajo las leyes de las físicas, este debería ser la siguiente posición "ideal" del robot. Sin embargo, a los puntos cercanos a esta posición también se les asignan alta probabilidad de ser el siguiente estado debido a que en la práctica, el robot puede terminar en estos puntos debido al ruido en su movimiento. Por otro lado, las posiciones muy lejanas a $x = 4$ tienen baja probabilidad (casi cero), ya que es muy improbable que el robot termine acabando en estas posiciones.

2.3 Modelo Probabilístico del Robot Móvil

En esta sección se describirá el modelo probabilístico del movimiento $p(x_t \mid u_t, x_{t-1})$ del robot móvil implementado en la presente tesis. Este modelo es esencial no solo para la formulación de algoritmos de navegación, sino para la simulación del movimiento real del robot en la computadora. Para mayor detalle sobre el diseño mecánico y los diversos componentes del robot referirse al Apéndice A.

Nótese que de los diversos tipos de robots móviles basados en ruedas (Figura 2.3) los robots del tipo diferencial presentan una serie de ventajas en la navegación en interiores [18], razón por la cual el robot implementado en el presente trabajo es del tipo diferencial. Esto significa que el movimiento está dado por 2 motores los cuales están conectados a dos ruedas de tracción. Además, se tienen un par de ruedas locas con el fin de dotar estabilidad al sistema (ver Apéndice A).

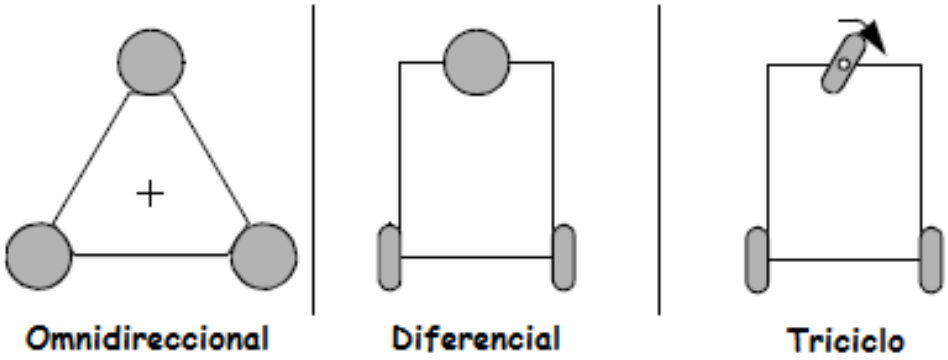


Figura 2.3. Tipos de robots móviles con ruedas más usados (Fuente: Elaboración propia)

2.3.1 Señales de Control

El control de movimiento de los robots del tipo diferencial se realiza a través de 2 velocidades: La velocidad translacional, que permite que el robot se desplace de atrás hacia adelante y viceversa; y la velocidad rotacional, que permite al robot realizar giros. Entonces la señal de control en el tiempo t se denota por u_t, y está dado por:

$$u_t = \begin{pmatrix} v_t \\ w_t \end{pmatrix} \tag{2.3}$$

Donde, v_t representa la velocidad translacional, y w_t representa la velocidad rotacional, en donde los valores positivos representan giros en sentido anti horario, y los valores negativos representan giros en sentido horario. Estas velocidades están establecidas por el módulo de control de movimiento del robot (Capítulo 6) con el fin que este siga una trayectoria deseada.

2.3.2 Modelamiento del Movimiento Ideal

Para hallar las ecuaciones que describen la cinemática “ideal o libre de ruido” de nuestro robot se asume que las velocidades v_t y w_t que se aplican, se mantienen constante en cada instante de tiempo. Entonces bajo esta asunción se tiene que en cada instante Δt de tiempo el robot realiza un movimiento circular alrededor de un centro de rotación, tal como se muestra en la Figura 2.4.

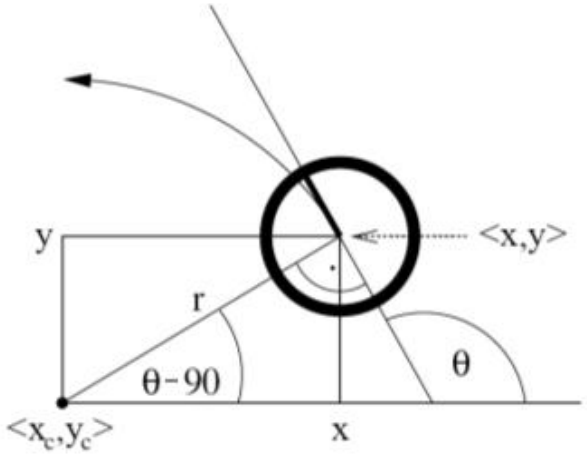


Figura 2.4. Modelamiento del movimiento ideal del robot (Fuente: http://www.probabilistic-robotics.org/)

Usando relaciones geométricas sencillas de realizar se puede demostrar la siguiente ecuación de transición de estado:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v}{w}\sin(\theta) + \frac{v}{w}\sin(\theta + w\Delta t) \\ \frac{v}{w}\cos(\theta) - \frac{v}{w}\cos(\theta + w\Delta t) \\ w\Delta t \end{pmatrix} \tag{2.4}$$

Que relaciona la transición de estado del robot luego de aplicar las acciones de control. Nótese que esta ecuación se refiere al tipo mas general de movimiento que es del tipo “translacional + rotacional”. Para el caso del movimiento del tipo “translacional” puro, que se da cuando $w = 0$, se requiere la formulación de una ecuación especial de transición de estado.

En la Figura 2.5, se muestran los resultados de la implementación de la ecuación ideal de movimiento del robot, donde el círculo en azul representa la posición inicial del mismo. En la primera parte el robot realiza un movimiento del tipo “translacional + rotacional” en donde el robot gira alrededor de un centro de rotación (representado por el circulo negro); luego el robot realiza un movimiento del tipo “translacional”; para finalmente realizar un nuevo giro y terminar en la posición representada por el círculo rojo.

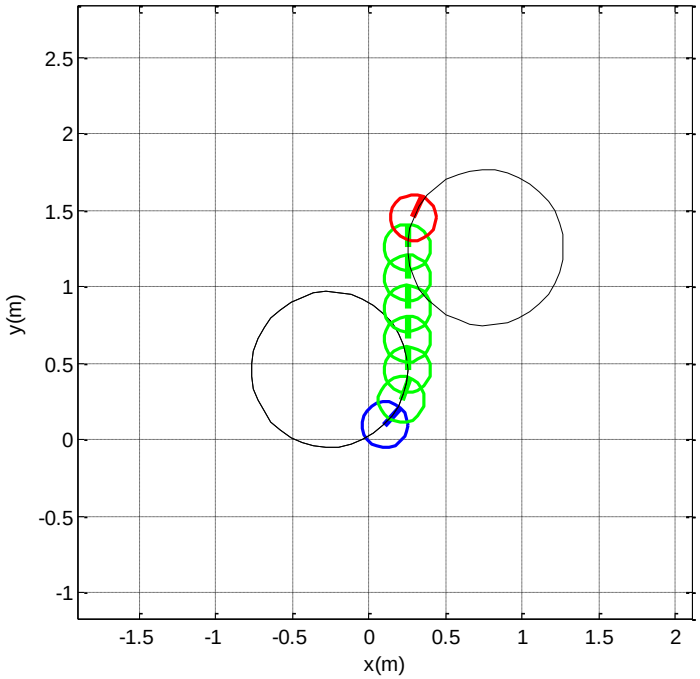


Figura 2.5. Movimiento ideal del robot móvil (Fuente: Elaboración propia)

2.3.3 Modelamiento del Movimiento Real

En la realidad, el movimiento del robot está sujeto a ruido, y esto se expresa por el hecho que la posición final del robot siempre va a diferir de la posición final predicha por el modelo “ideal” del robot. Esto implica que las velocidades actuales (necesarias para llegar a la posición final “real” del robot) difieren de las velocidades comandadas. Esta diferencia representa un error o incertidumbre en las velocidades. Entonces, asumiendo que este error tiene una función de densidad de distribución gausiano, podemos modelar las velocidades “reales” del robot como variables aleatorias “gaussianas” con media dada por la velocidad “ideal”, y varianza que depende del ruido en el movimiento, de donde se tiene la siguiente ecuación:

$$\begin{pmatrix} \hat{v} \\ \hat{w} \end{pmatrix} = \begin{pmatrix} v \\ w \end{pmatrix} + \begin{pmatrix} \varepsilon_{\alpha_1 v^2 + \alpha_2 w^2} \\ \varepsilon_{\alpha_3 v^2 + \alpha_4 w^2} \end{pmatrix} \tag{2.5}$$

Esta ecuación expresa el hecho que las velocidades reales son del tipo aleatorio, en donde la velocidad comandada es la que tiene mayor probabilidad. El nivel de incertidumbre está dado por los parámetros α_i los cuales definen el nivel de ruido presente que depende de factores como: el material de las llantas, la calidad del piso (rugoso, suave, resbaloso, etc.), la precisión mecánica en la construcción del robot (dimensionamiento al milímetro del diámetro de las ruedas, el uso de ruedas perfectamente iguales, eje correctamente alineado, etc.), la calidad del controlador de velocidad de las ruedas (tiempo de establecimiento, etc.), etc. En la Figura 2.6 se muestra un ejemplo de una función de densidad de probabilidad de la velocidad "real" del robot cuando la velocidad comandada es de 1 m/s y el ruido tiene una varianza de 1.

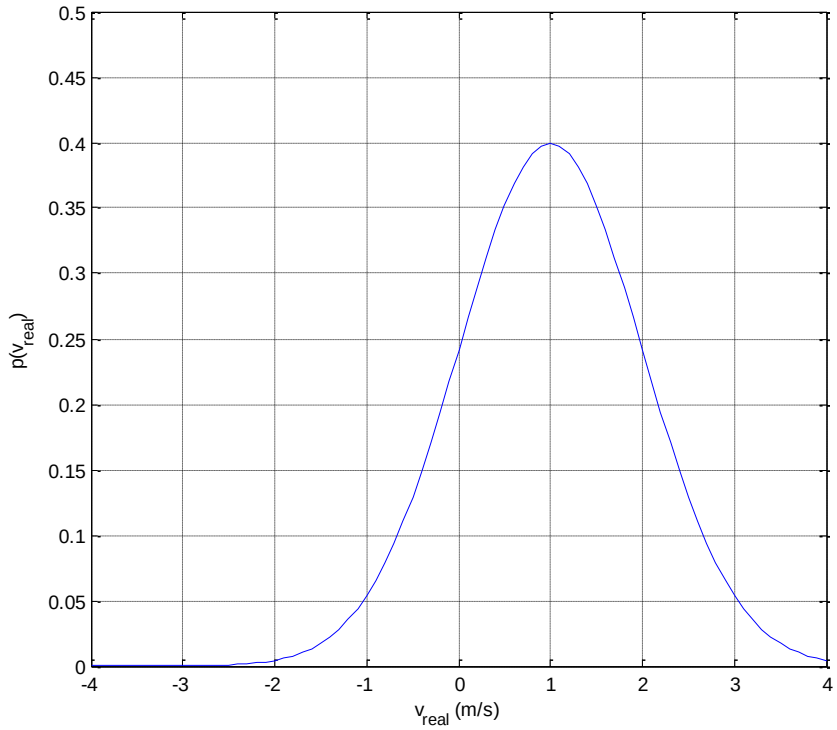


Figura 2.6. Función de probabilidad de la velocidad real del robot (Fuente: Elaboración propia)

Entonces, las ecuaciones que describirían el movimiento real del robot, se obtendrían reemplazando las velocidades "aleatorias" reales dadas en la ecuación (2.5) y en la ecuación de cambio de estado (2.4). Para esto, se asume que, aunque las

velocidades reales son diferentes de las comandadas, estas aún son constantes en cada instante de tiempo con el fin que el robot se mueva alrededor de un centro de rotación. Esta asunción de movimiento circular, genera una degeneración, en el sentido que algunos puntos del espacio estado tendrían cero probabilidad de ser la siguiente posición del robot. Entonces, con el fin de evitar este problema, se asume que el robot al llegar a la ubicación final realiza un pequeño giro sobre su eje. De donde el modelo para hallar la orientación final estaría dado por:

$$\theta' = \theta + \hat{w}\Delta t + \hat{\gamma}\Delta t \tag{2.6}$$

Donde $\hat{\gamma} = \varepsilon_{\alpha_5 v^2 + \alpha_6 w^2}$ es la rotación final del robot, que está dado por un numero aleatorio gaussiano de media cero y desviación estándar dado por los parámetros α_5 y α_6. Entonces, la ecuación que define el movimiento real del robot estaría dada por la siguiente ecuación:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{\hat{v}}{\hat{w}}\sin(\theta) + \frac{\hat{v}}{\hat{w}}\sin(\theta + \hat{w}\Delta t) \\ \frac{\hat{v}}{\hat{w}}\cos(\theta) - \frac{\hat{v}}{\hat{w}}\cos(\theta + \hat{w}\Delta t) \\ \hat{w}\Delta t + \hat{\gamma}\Delta t \end{pmatrix} \tag{2.7}$$

Para definir el movimiento probabilístico del robot es necesario definir los parámetros de ruido $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6$. En la práctica es imposible definir un valor exacto de estos valores, ya que el ruido no solo depende del robot sino del entorno en el que este opere (cada ambiente de navegación presente diferentes tipos de pisos, y es imposible modelar todos los posibles casos). Lo más conveniente es desarrollar pruebas en diferentes tipos de pisos y estimar un valor promedio para estos valores de ruido. Luego de realizar diversas pruebas (que básicamente consisten en comparar las posiciones reales con las posiciones ideales para hallar el error en el movimiento), los parámetros estimados de ruido de nuestro robot fueron:

$$\alpha_1 = 2.2e-3 \qquad \alpha_2 = 1.8e-5 \qquad \alpha_3 = 2.0e-2$$
$$\alpha_4 = 2.0e-2 \qquad \alpha_5 = 1.0e-4 \qquad \alpha_6 = 1.2e-4$$

2.3.4 Función de Probabilidad de Movimiento

En la Tabla 2.1 se muestra el algoritmo que implementa la función de densidad de probabilidad de movimiento $p(x_t \mid u_t, x_{t-1})$ del robot. Este algoritmo permite evaluar la probabilidad de cualquier estado x_t dado un estado inicial x_{t-1} y la acción de una señal de control u_t. La idea básica del algoritmo es hallar el modelo inverso del robot. Es decir, hallar la señal de control $\hat{u}_t = [\hat{v}; \hat{w}; \hat{\gamma}]$ que permite la transición del estado x_{t-1} a x_t (líneas 1 a 8); para luego evaluar esta señal en la función de probabilidad de la velocidad $u_t = [v; w]$ del robot (líneas 9, 10, y 11).

Tabla. 2.1. Algoritmo para generar la función de densidad de probabilidad de movimiento. (Fuente: http://www.probabilistic-robotics.org/)

$Salidas: p(x_t \mid u_t, x_{t-1})$

$Entradas: x_t, u_t, x_{t-1}$

1: $\mu = \frac{1}{2}\frac{(x-x')\cos\theta + (y-y')\sin\theta}{(y-y')\cos\theta - (x-x')\sin\theta}$

2: $x^* = \frac{x+x'}{2} + \mu(y-y')$

3: $y^* = \frac{y+y'}{2} + \mu(x'-x)$

4: $r^* = \sqrt{(x-x^*)^2 + (y-y^*)^2}$

5: $\Delta\theta = a\tan 2(y'-y^*, x'-x^*) - a\tan 2(y-y^*, x-x^*)$

6: $\hat{v} = \frac{\Delta\theta}{\Delta t} r^*$

7: $\hat{w} = \frac{\Delta\theta}{\Delta t}$

8: $\hat{\gamma} = \frac{\theta'-\theta}{\Delta t} - \hat{w}$

9: $error\_v = v - \hat{v}$

10: $error\_w = w - \hat{w}$

11: $p = gaus(error\_v, \alpha_1 v^2 + \alpha_2 w^2) \cdot gaus(error\_w, \alpha_3 v^2 + \alpha_4 w^2)$
$\cdot gaus(\hat{\gamma}, \alpha_5 v^2 + \alpha_6 w^2)$

En este algoritmo, la función "$gaus(e, \sigma^2)$" implementa la función de probabilidad gausiana con media '0' y varianza σ^2. Nótese que la probabilidad total es el producto de las probabilidades individuales.

2.4 Resultados obtenidos

Luego de haber implementado el algoritmo de movimiento descrito en la Tabla 2.1, a continuación se muestran los resultados obtenidos para los diferentes tipos de movimiento que es capaz de realizar nuestro robot móvil. Los valores numéricos mostrados en la barra derecha de las Figuras (2.7), (2.8) y (2.9) representan los valores de la función de densidad de probabilidad para diferentes posiciones del robot.

2.4.1 Movimiento Translacional + Rotacional

En la Figura 2.7 se muestra los resultados en el caso que el robot realice un movimiento del tipo "translacional + rotacional". En esta figura, el círculo rojo describe el estado inicial del robot $x_{t-1} = [0,0,0]^T$. El arco de color azul representa la acción de control $u_t = [v_t, w_t]^T$, donde $v_t = 0.2 \ \ m/s, \ \ w_t = pi/16 \ \ rad/s$ con un delta de tiempo $\Delta t = 5s$. Y el círculo de color magenta representa el estado final $x_t = [0.8469, 0.4527, 0.9817]^T$ que se obtendría en caso de que el robot realice un movimiento perfecto "libre de ruido".

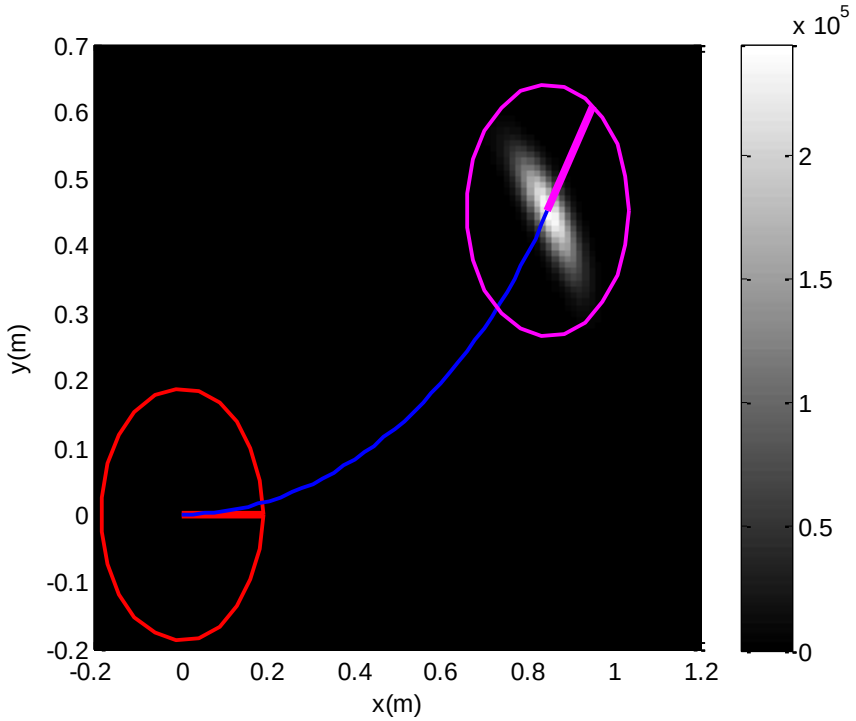


Figura 2.7. Función de probabilidad de movimiento del robot – Translación + Rotación (Fuente: Elaboración propia)

2.4.2 Movimiento Translacional

En la Figura 2.8 se muestra los resultados en el caso que el robot realice un movimiento del tipo “translacional”. En esta figura, el círculo rojo describe el estado inicial del robot $x_{t-1} = [0,0,0]^T$. Las velocidades están dadas por $v_t = 0.2\ \ m/s,\ \ w_t = 0\ \ rad/s$ con un tiempo de $\Delta t = 5s$. Y el círculo de color magenta representa el estado final $x_t = [1,0,0]^T$ que se obtendría en caso de que el robot realice un movimiento perfecto “libre de ruido”.

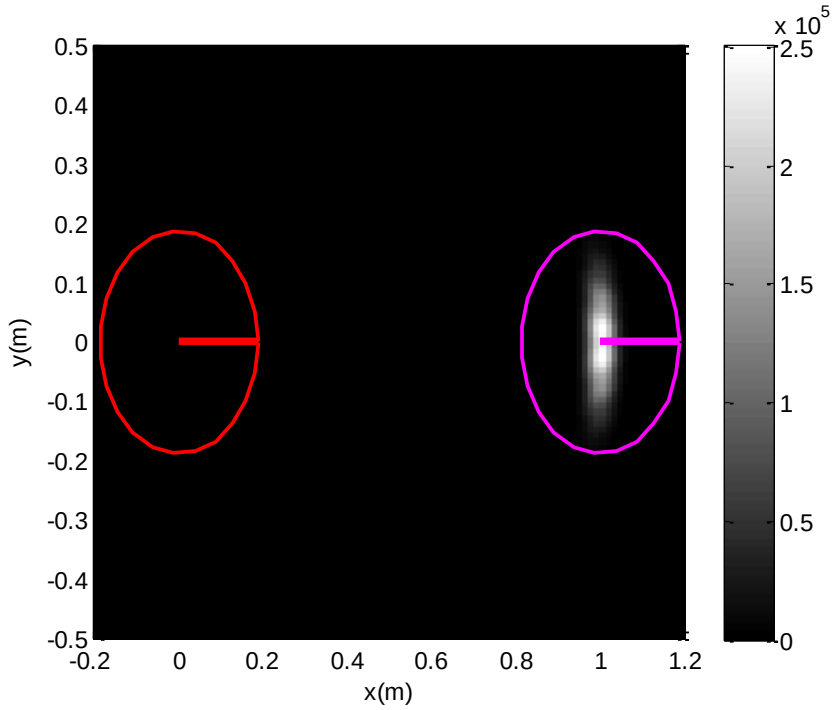


Figura 2.8. Función de probabilidad de movimiento del robot – Translación (Fuente: Elaboración propia)

2.4.3 Movimiento rotacional

En la Figura 2.9 se muestra los resultados en el caso que el robot realice un movimiento del tipo “rotacional”. En este experimento, el círculo es la configuración inicial del robot $x_{t-1} = [0,0,0]^T$. Las velocidades están dadas por $v_t = 0\ \ m/s,\ \ w_t = pi/16\ \ rad/s$ con un tiempo de $\Delta t = 5s$. La configuración final del robot está dada por el círculo en magenta. Nótese que en este caso las posiciones final e inicial son muy cercanas (dado que el robot solo gira sobre su eje).

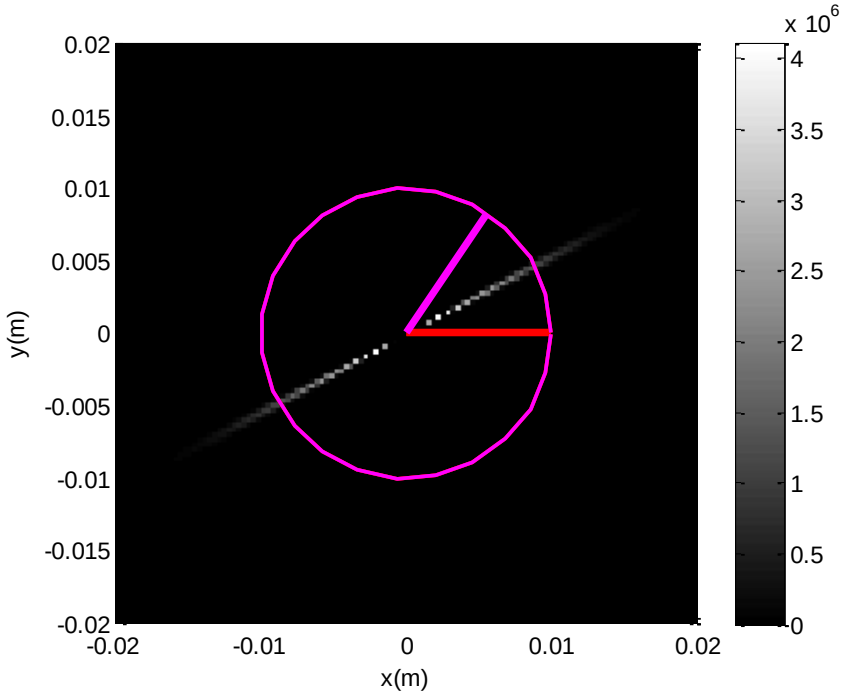


Figura 2.9. Función de probabilidad de movimiento del robot – Rotación (Fuente: Elaboración propia)

Nótese que en todos los experimentos, las configuraciones cercanas a la configuración final “ideal” del robot también reciben una gran probabilidad, esto, en concordancia con los posibles resultados del movimiento “real” del robot, en donde las configuraciones “reales” finales tienden a estar cerca de la configuración “ideal” esperada.

CAPÍTULO III
MAPAS DE OCUPACIÓN Y MODELO PROBABILISTICO DEL SENSOR LASER

El objetivo del presente capitulo es la descripción probabilística tanto del entorno de navegación de nuestro robot móvil, así como de los sensores que este usa para realizar tareas de navegación autónoma. En el caso de la descripción de los entornos de navegación, nos centraremos en los llamados “mapas de ocupación”, por ser los más adecuados tanto en tareas de localización así como de planeamiento. Respecto a los sensores, el objetivo será hallar el modelo probabilístico del sensor laser Hokuyo URG 04LX – UG01 usado en el presente trabajo. Nótese que este tipo de modelos probabilísticos son los que se usan en los algoritmos que representan el estado del arte en la robótica móvil.

En este capítulo solo se hará una introducción a los mapas de ocupación. Para mayor detalle sobre la construcción tanto manual como automática de estos mapas referirse al Capítulo 5 del presente trabajo.

3.1 Mapas

Para que un robot pueda navegar de manera autónoma en un ambiente dado este necesita tener un mapa de dicho ambiente. Este mapa no solo le permite al robot conocer las características y atributos de su entorno de navegación, sino que le permite explicar el proceso de la generación de las mediciones de sus sensores.

En la robótica móvil se han diseñado diferentes tipos de mapas [18], es decir, maneras de representar el entorno de navegación. En el presente trabajo nos centraremos únicamente en los llamados mapas de ocupación, debido a que son los más adecuados para trabajar con los sensores laser, y además, debido a que se pueden usar para el desarrollo de algoritmos de planeamiento.

3.1.1 Mapas de Ocupación

Los mapas de ocupación permiten la descripción del entorno de navegación del robot entre espacios navegables y no navegables. Para esto se hace una discretización del mundo en celdas, de tal manera que las celdas que corresponden a un espacio ocupado reciben un valor de 1 (negro) y las celdas que corresponden a un espacio libre reciben un valor de 0 (blanco), tal como se observa en la Figura 3.1.

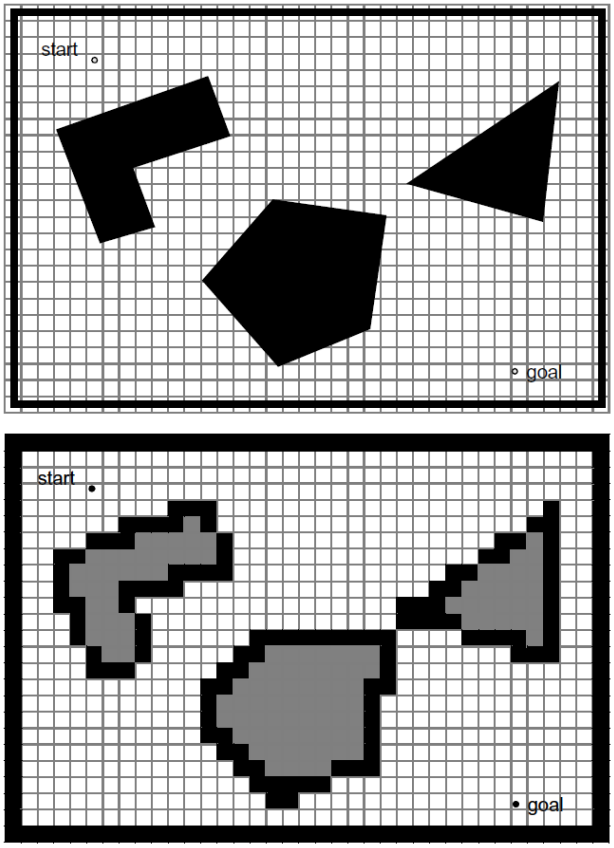


Figura 3.1. Mapa de ocupación (Fuente: http://www.mobilerobots.ethz.ch/)

Estos mapas son considerados del tipo 'volumétrico' debido a que describen en detalle cada posición del entorno de navegación. La ventaja de estos mapas es que permiten el desarrollo de algoritmos de planeamiento, ya que muestran de manera explícita los espacios ocupados y desocupados del entorno de navegación.

El único parámetro de control en la construcción de este tipo de mapas es el tamaño de las celdas. Si se usa un valor pequeño se tendrá más calidad en los mapas, pero se requerirá más memoria para poder almacenar los mismos. Si se usa un valor

grande, los mapas tendrán menos calidad pero requerirán menos memoria. En la práctica, la gran mayoría de robots móviles usan celdas en el rango de 4 a 15 cm.

3.1.2 Resultados Obtenidos

Con el fin de realizar las pruebas de simulación de los algoritmos de Localización y Mapeo se usara el mapa de ocupación que se muestra en la Figura 3.2, el cual representa el segundo piso del hogar del autor (cabe señalar que no se han considerado las puertas ni los accesorios que se encuentran en un hogar común tales como las puertas, mesas, etc.).

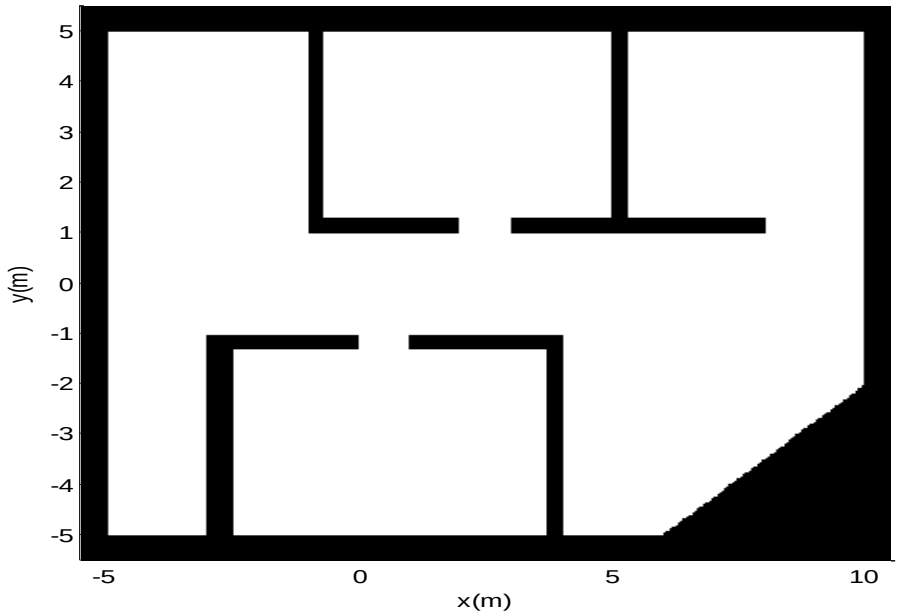


Figura 3.2. Mapa de ocupación del segundo piso del hogar del autor (Fuente: Elaboración propia)

En este mapa, el tamaño seleccionado de las celdas es de 5 cm, lo cual resulta en una matriz de ocupación de dimensión 220×784. Si se hubiese elegido una resolución de 4cm el tamaño de la matriz de ocupación seria 275×980. Y si se hubiese elegido una resolución de 10 cm el tamaño de la matriz seria 110×392. La ventaja de usar una resolución de 5 cm es que el mapa de ocupación construido es de buena calidad, y no requiere tanta memoria para poder almacenarlo. Finalmente, nótese que el mapa de ocupación no es otra cosa que una matriz que representa los espacios ocupados y libres del entorno de navegación del robot.

3.2 Sensores Laser

Los sensores laser son los sensores más usados en el campo de la robótica móvil hoy en día, y se usan con el fin de medir distancias al objeto más cercano en una dirección dada. En particular, los sensores laser hacen un barrido emitiendo señales de luz en diferentes direcciones. De esta manera, el robot es capaz de percibir la presencia de paredes y obstáculos en una serie de direcciones (Figura 3.3).

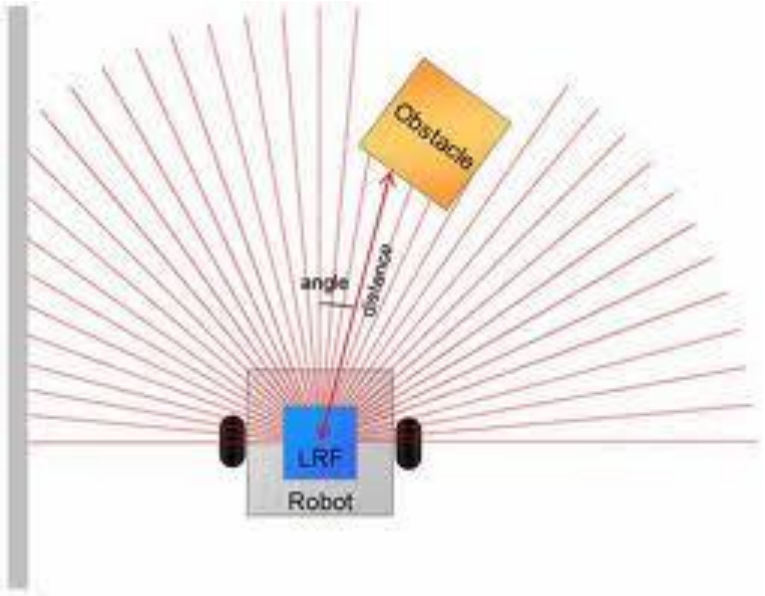


Figura 3.3. Funcionamiento del sensor laser (Fuente: http://msdn.microsoft.com/en-us/library/bb483042.aspx)

En la Figura 3.4 se muestran las dos marcas de sensores laser más usados: SICK y Hokuyo. La ventaja de los sensores de la marca SICK es que proveen un mayor rango de alcance (de hasta 80 metros); sin embargo, la gran desventaja es que son bastante caros (el SICK LMS 200 tiene un costo aproximado de 7,000.00 dólares). Los sensores de la marca Hokuyo, son más accesibles, aunque tienen un menor rango. Estos dos sensores son los más usados por los sistemas robóticos más avanzados a nivel mundial.

El sensor usado en el presente trabajo es el sensor laser Hokuyo URG-04LX-UG01 el cual tiene un alcance de hasta 5.6 m, un precio aproximado de 1,500.00 dólares, y una interfaz de comunicación de alta velocidad usando el protocolo USB 2.0. Además se debe notar que este sensor no requiere de una fuente de alimentación externa. En el Apéndice A se muestra todas las características de este sensor.

Figura 3.4. Sensores laser SICK y HOKUYO (Fuente: http://www.sick.com/)

3.2.1 Mediciones Ideales del Sensor Laser

En la teoría, la mediciones ideales de los sensores laser debería ser la distancia "libre de ruido" hacia el objeto más cercano tal como se ve en la Figura 3.5. En esta figura, el láser se encuentra situado sobre el robot y encuentra las distancias perfectas hacia las paredes en una serie de direcciones. En esta imagen las líneas en magenta representan los rayos de luz emitidos por el láser.

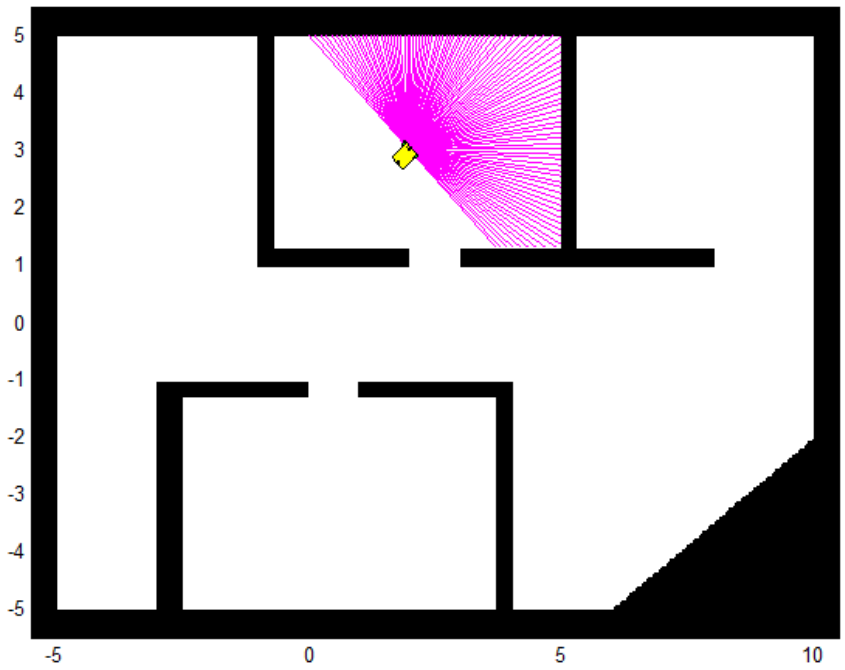


Figura 3.5. Mediciones ideales del sensor laser (Fuente: Elaboración propia)

3.2.2 Mediciones Reales del Sensor Laser

En la práctica, las mediciones de los sensores están sujetas a diversas fuentes de ruido, lo que provoca que estas mediciones difieran de las mediciones “esperadas” por el robot [18], [19]. Estas fuentes de ruido se deben a la presencia de personas en el ambiente, el ruido propio del sensor, etc. En la Figura 3.6 se muestran las mediciones “reales” entregadas por el sensor en el caso de nuestro mapa. Comparando estos resultados con los obtenidos en la Figura 3.5 claramente se puede observar los efectos del ruido en las mediciones del sensor.

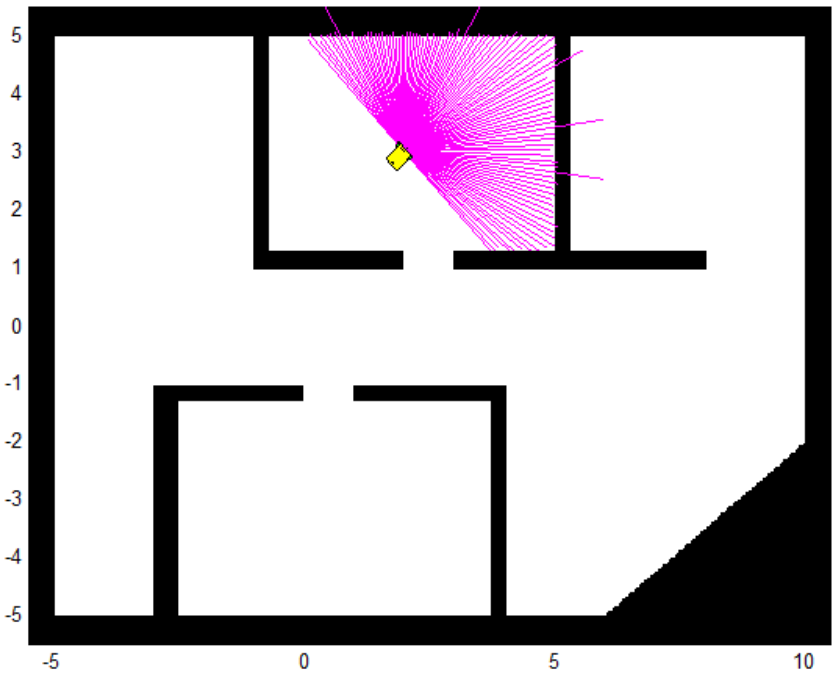


Figura 3.6. Mediciones reales del sensor laser (Fuente: Elaboración propia)

3.3 Modelo Probabilístico del Sensor Laser

Tal como se mencionó en el Capítulo 1 del presente trabajo, el modelo probabilístico de un sensor está dado por $p(z \mid x)$, que nos indica “la probabilidad de las mediciones z, dado que el robot está en el estado x”. La ventaja de usar este tipo de modelos es que nos permiten modelar y manipular las diversas fuentes de ruido en las mediciones “reales” de cualquier tipo de sensor. En esta sección se mostrara este modelo en el caso específico de nuestro sensor laser.

El modelo seleccionado para describir la dinámica del sensor laser, es el comúnmente llamado “beam based model” el cual está basado en el proceso físico que generan las mediciones del sensor laser en un rayo dado [19], [20]. En este modelo se reconoce que existen 4 fuentes de ruido que afectan las mediciones del sensor laser:

a. Ruido gausiano local. Se refiere al ruido local de valores pequeños que afectan las mediciones correctas del sensor. Este ruido es debido principalmente a errores en la resolución del sensor o condiciones atmosféricas presentes. Estos errores se pueden modelar con una distribución gausiana cuya media está dada por la medición “ideal” z^{k*} del sensor y una desviación estándar σ_{hit}.

$$p_{hit}(z_t^k \mid x_t, m) = \begin{cases} \eta N\left(z_t^k; z_t^{k*}, \sigma_{hit}^2\right), & if\ 0 \le z_t^k \le z_{\max} \\ 0, & c.c \end{cases} \tag{3.1}$$

Este modelo se puede apreciar en la Figura 3.7. Nótese que en este modelo la desviación estándar σ_{hit} define la cantidad de ruido.

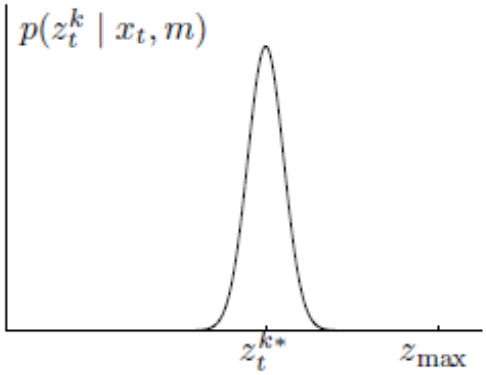


Figura 3.7. Modelo basado en rayos del sensor laser – Ruido local (Fuente: http://www.probabilistic-robotics.org/)

b. Ruido debido a objetos inesperados. Se da cuando la medición obtenida es mucho menor que la medición esperada por el robot. Este ruido es debido principalmente a la presencia de personas u objetos que normalmente se desplazan en el entorno de navegación del robot. Matemáticamente, este ruido se modela usando una función exponencial ya que los obstáculos más cercanos tienen más probabilidad de ser medidos que los obstáculos más lejanos. Entonces la función de densidad de probabilidad tiene la forma:

$$p_{short}(z_t^k \mid x_t, m) = \begin{cases} \eta\lambda_{short} e^{-\lambda_{short} \cdot z_t^k}, & if\ 0 \le z_t^k \le z_t^{k*} \\ 0, & c.c \end{cases} \tag{3.2}$$

Esta densidad de probabilidad se puede apreciar en la Figura 4.8, donde se observa que la probabilidad decae de manera exponencial.

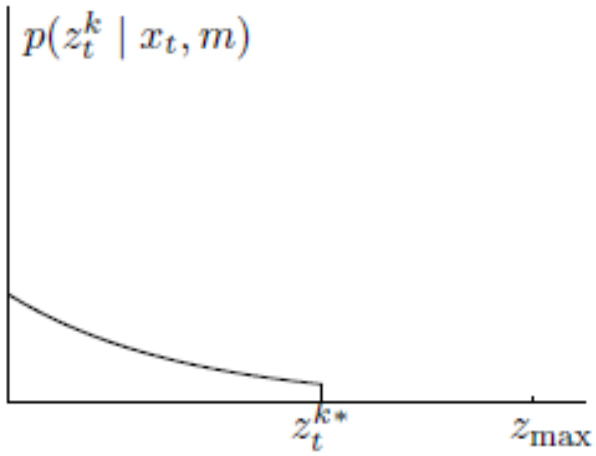


Figura 3.8. Modelo basado en rayos del sensor laser – Ruido debido a objetos inesperados (Fuente: http://www.probabilistic-robotics.org/)

c. Ruido debido a fallas. Se da cuando el sensor no detecta ningún obstáculo aun cuando en su campo de acción existen obstáculo y/o paredes. La principal fuente de este error es debido al fenómeno de "reflexión especular" que se da cuando la pared está en un ángulo respecto a la dirección del rayo láser. Otra fuente de error, es cuando los objetos o paredes son de color negro y por tal, no permiten la reflexión del rayo de luz del sensor láser

Estadísticamente esta fuente de error se modela usando una distribución de masa centrada en la medida máxima $z_{\max}$.

$$p_{\max}(z_t^k \mid x_t, m) = \begin{cases} 1, & if\ z_t^k = z_{\max} \\ 0, & c.c \end{cases} \tag{3.3}$$

Este modelo se muestra en la Figura 3.9, donde se puede ver claramente que la probabilidad solo es diferente de cero en el punto $z_{\max}$.

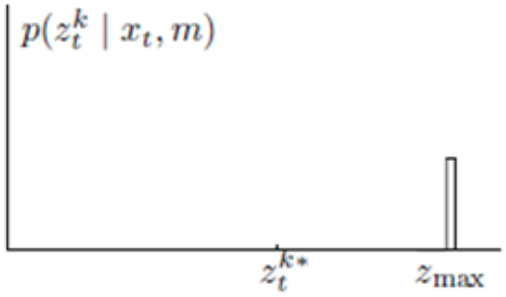


Figura 3.9. Modelo basado en rayos del sensor laser – Ruido debido a fallas (Fuente: http://www.probabilistic-robotics.org/)

d. Ruido aleatorio. Los sensores laser pueden producir mediciones que son del todo inexplicables. Una posible explicación de este ruido se debe al fenómeno de "*cross talk*", que son las interferencias entre las mediciones de los rayos contiguos del sensor. Estadísticamente, este ruido se modela usando una distribución uniforme.

$$p_{rand}(z_t^k \mid x_t, m) = \begin{cases} 1/z_{\max}, & if\ 0 \le z_t^k < z_{\max} \\ 0, & c.c \end{cases} \tag{3.4}$$

Tal como se puede apreciar en la Figura 4.10, en este modelo cualquier punto en el intervalo $[0, z_{\max}]$ tiene la misma probabilidad.

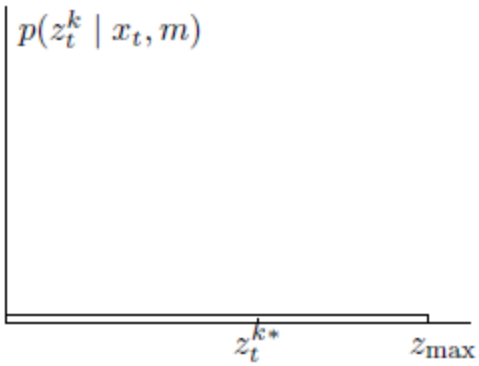


Figura 3.10. Modelo de probabilidad de medida del sensor laser – Ruido aleatorio (Fuente: http://www.probabilistic-robotics.org/)

Entonces, estas diferentes fuentes de error son las que afectan las mediciones "ideales" del sensor laser, por lo que la función de probabilidad de las mediciones $p(z \mid x)$ resultante del sensor está dada por la suma ponderada de las funciones de densidad de probabilidad individuales tal como se muestra a continuación:

$$p(z_t^k \mid x_t, m) = \begin{pmatrix} z_{hit} \\ z_{short} \\ z_{\max} \\ z_{rand} \end{pmatrix}^T \cdot \begin{pmatrix} p_{hit}(z_t^k \mid x_t, m) \\ p_{short}(z_t^k \mid x_t, m) \\ p_{\max}(z_t^k \mid x_t, m) \\ p_{rand}(z_t^k \mid x_t, m) \end{pmatrix} \tag{3.5}$$

Este resultado se puede apreciar en la Figura 3.11, donde la función de probabilidad total es la suma de las probabilidades individuales. Nótese que en este modelo, los parámetros $z_{hit}, z_{short}, z_{max}$ y z_{rand}, definen los pesos de cada componente.

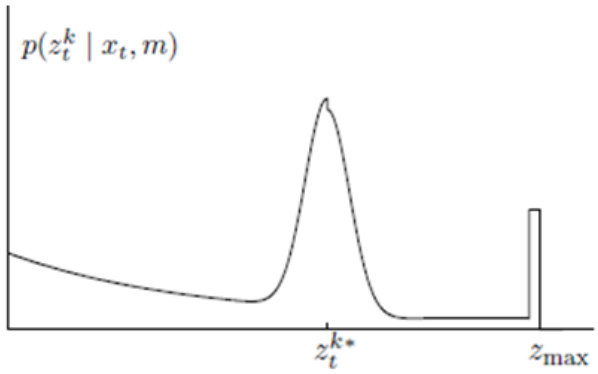


Figura 3.11. Modelo probabilístico del sensor laser basado en rayos (Fuente: http://www.probabilistic-robotics.org/)

En la Tabla 3.1 se muestra el algoritmo que permite evaluar la probabilidad de las mediciones (3.5) para cualquier posible combinación de los parámetros x_t y z_t; y para cualquier cantidad de rayos "K" emitidos por el sensor laser.

Tabla. 3.1. Algoritmo para evaluar la probabilidad de las mediciones del sensor laser (Fuente: http://www.probabilistic-robotics.org/)

$Salidas: p(z_t \mid x_t, m)$
$Entradas: z_t, x_t, m$
1: $q = 1$
2: *for* $k = 1:K$
3: *Calcular* z_t^{k*} *para* z_t^k
4: $p = z_{hit} \cdot p_{hit}(z_t^k \mid x_t, m) + z_{short} \cdot p_{short}(z_t^k \mid x_t, m)$
5: $+ z_{\max} \cdot p_{\max}(z_t^k \mid x_t, m) + z_{rand} \cdot p_{rand}(z_t^k \mid x_t, m)$
6: $q = q \cdot p$
7: *Retornar* q

3.4 Resultados Obtenidos

Luego de haber implementado el algoritmo de movimiento descrito en la Tabla 3.1, a continuación se muestran los resultados obtenidos usando el mapa de ocupación descrito en la sección 3.1.2. El objetivo en esta sección es analizar la función de densidad de probabilidad de las mediciones $p(z \mid x)$ del sensor laser de nuestro robot móvil.

3.4.1 Primera Prueba

En esta prueba el estado del robot fue $x = [5, 4, pi/2]^T$, y el número de rayos usados por el sensor laser fue de $K = 25$. Esta prueba se puede ver en la figura 3.12 donde podemos apreciar el robot y las mediciones del sensor laser.

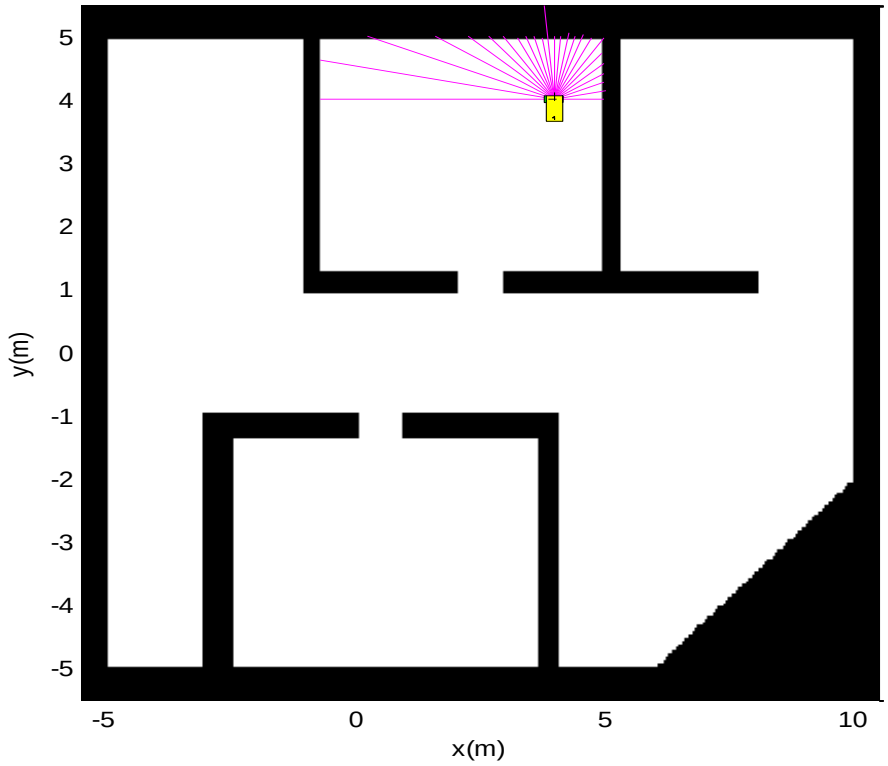


Figura 3.12. Primera prueba (Fuente: Elaboración propia)

En la Figura 3.13 se puede ver el modelo probabilístico del sensor $p(z \mid x)$ obtenido para las condiciones anteriormente descritas.

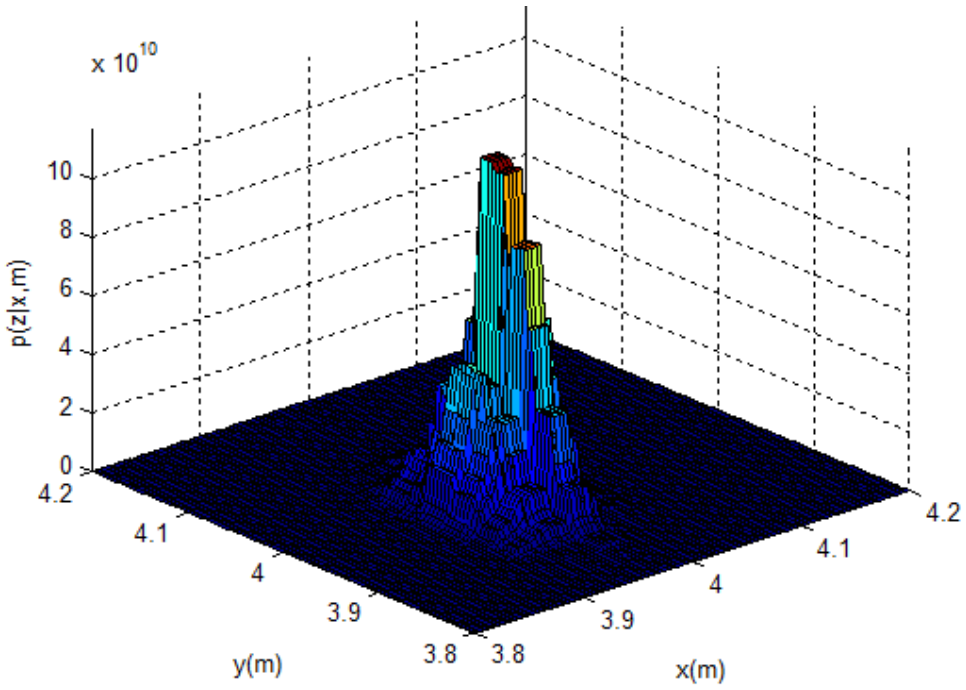


Figura 3.13. Primera prueba – Resultados obtenidos (Fuente: Elaboración propia)

3.4.2 Segunda Prueba

En esta prueba el estado del robot fue $x = [7, 0, 0]^T$, y el número de rayos usados por el sensor laser fue de $K = 15$ (Figura 3.14).

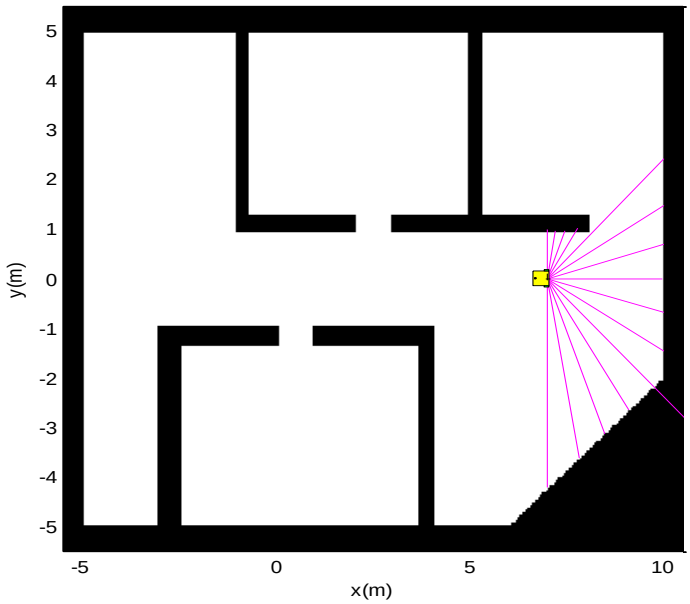


Figura 3.14. Segunda prueba (Fuente: Elaboración propia)

En la Figura 3.15 se puede ver el modelo probabilístico del sensor $p(z \mid x)$ obtenido para las condiciones anteriormente descritas.

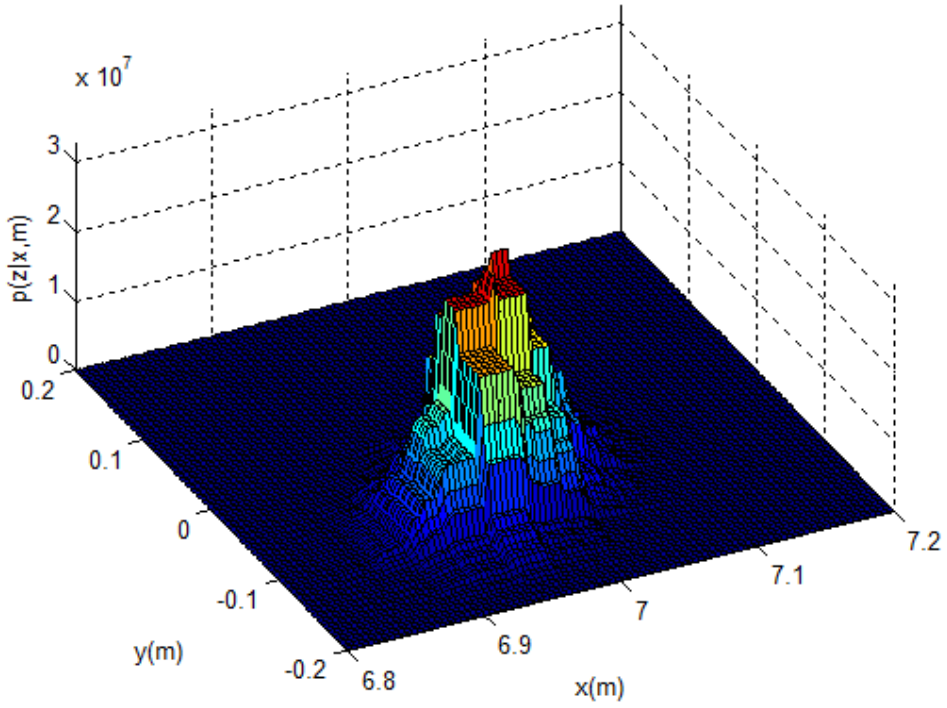


Figura 3.15. Segunda prueba – Resultados obtenidos (Fuente: Elaboración propia)

3.4.3 Conclusiones

De los resultados anteriores puede concluirse primeramente que, el modelo de la probabilidad de las mediciones $p(z \mid x)$ del sensor laser es del tipo continuo por tramos. Esto es debido a la discretización del entorno de navegación en celdas. Conforme se use celdas de menor resolución, esta función de probabilidad se aproximara más a una función del tipo continua.

Por otro lado debe notarse que cuando se usa una mayor cantidad de rayos del sensor laser “K”, las funciones de densidad de probabilidad $p(z \mid x)$ tienen un valor más considerable, y están mucho más centradas alrededor de la configuración actual del robot.

CAPÍTULO IV
LOCALIZACIÓN USANDO MÉTODOS MONTE CARLO

El objetivo del presente capitulo consiste en la resolución del problema de la localización usando métodos Monte Carlo. La localización es de suma importancia en la navegación autónoma de robots móviles, ya que para realizar esta tarea, el requisito fundamental es que el robot sepa cuál es su configuración en cualquier instante de tiempo. En este capítulo se mostrara el algoritmo de localización Monte Carlo estándar, y se verá las maneras de mejorarlo con el fin de tener un algoritmo más robusto y eficiente. Este último punto es una de las principales contribuciones del presente trabajo de tesis.

4.1 El Problema de la Localización

El problema de la Localización se refiere a la estimación de la configuración (posición + orientación) del robot respecto al mapa que este tiene de su entorno de navegación, usando únicamente las mediciones "ruidosas" de sus sensores (Figura 4.1). Frecuentemente, este problema ha sido señalado como "*el problema más fundamental en el diseño de robots móviles con capacidades de navegación autónomas*" [18].

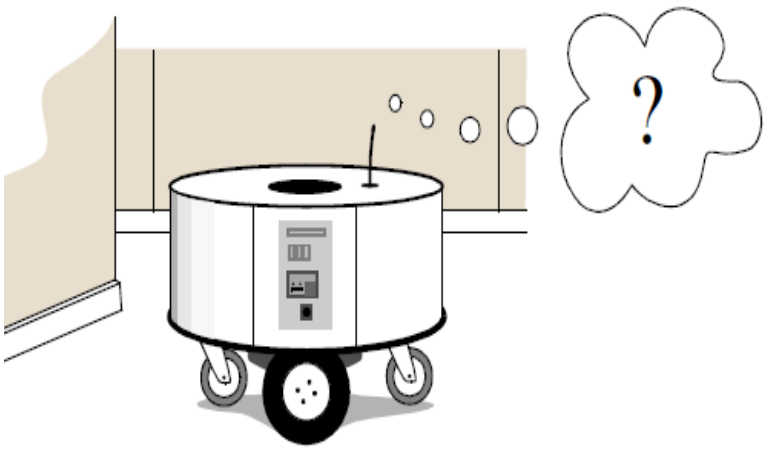


Figura 4.1. El problema de la localización (Fuente: http://www.mobilerobots.ethz.ch/)

Todos los algoritmos probabilísticos que tienen como objetivo resolver el problema de la localización tienen como punto de partida el Filtro de Bayes descrito en la sección 1.3 del presente trabajo. La diferencia entre estos algoritmos se da en la manera de modelar las leyes probabilísticas que modelan la dinámica del sistema: $p(x_t \mid u_t, x_{t-1})$ y $p(z_t \mid x_t)$, y la manera de modelar la distribución $bel(x_t) = p(x_t \mid z_{1:t}, u_{1:t})$.

4.2 Localización usando Métodos Monte Carlo Estándar

Los algoritmos de localización usando métodos Monte Carlo secuenciales recientemente se han vuelto muy populares en el campo de la robótica, y esto es debido principalmente a que: son muy robustos, pueden trabajar con los sensores laser y ultrasónicos, y no presentan las restricciones de los algoritmos basados en el filtro de Kalman. En esta sección se describirá el algoritmo Monte Carlo estándar usado comúnmente en la tarea de localización, y en la sección siguiente se verán maneras de mejorar este algoritmo con el fin de hacerlo más robusto y eficiente.

4.2.1 Conceptos Básicos

Tal como se vio en la sección 1.4, la idea básica de los métodos Monte Carlo es trabajar con las muestras de una variable aleatoria en vez de trabajar directamente con la función de probabilidad de la misma. El algoritmo SIS descrito en la Tabla 1.3 representa el algoritmo más básico que implementa el filtro de Bayes usando el método Monte Carlo, por lo que en el caso del problema de la localización representa el punto de partida a partir del cual se van a derivar los algoritmos especialmente diseñados para el caso de los robots móviles.

El algoritmo SIS (ver sección 1.4.2) requiere el uso de una función propuesta $q(x_{0:t} \mid z_{1:t}, u_{1:t})$ de tal forma que sea fácil tomar muestras. En el método Monte Carlo estándar [18], [19] esta función usualmente se escoge como la función de probabilidad de movimiento:

$$q(x_t \mid x_{t-1}, z_t, u_t) = p(x_t \mid x_{t-1}, u_t) \tag{4.1}$$

Esta elección se debe a que es posible tomar muestras de esta función de distribución. Entonces, reemplazando esta distribución en la ecuación (1.28) se tiene la siguiente ecuación de actualización de pesos de cada muestra:

$$w_t^i \propto p(z_t \mid x_t^i) w_{t-1}^i \tag{4.2}$$

4.2.2 Algoritmo de Localización

En la Tabla 4.1 se muestra el algoritmo SIS adaptado para la tarea de localización de los robots móviles. Las líneas L1, L2, L3 y L4 son casi idénticas a las líneas presentes en el algoritmo SIS, excepto por las modificaciones descritas en la sección anterior. Las líneas restantes hacen la etapa de re muestreo con el fin que el algoritmo no sufra el problema de la degeneración (ver sección 1.5.3).

Tabla. 4.1. Algoritmo de Localización Monte Carlo secuencial estándar (Fuente: http://www.probabilistic-robotics.org/)

$Algoritmo \;\; Localizacion\_MC(\{x_{t-1}^m\}, u_t, z_t)$
$L1: \;\; for \;\; m = 1:M$
$L2: \quad x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
$L3: \quad w_t^{[m]} = p(z_t \mid x_t^{[m]})$
$L4: \; end$
$L5: \; for \;\; i = 1:M$
$L6: \quad Muestrear \; "x^{[m]}" \; con \; probabilidad \; \alpha \; w_t^{[m]}$
$L7: \quad x_t^{[i]} = x_t^{[m]}$
$L8: \; end$
$return \;\; \{x_t^m\}$

En este algoritmo la línea 2 se refiere a la generación de muestras de la función propuesta (4.1) que en este caso es la función de probabilidad de movimiento. En la línea 3, se asigna a cada muestra un peso o factor de importancia mediante el que se especifica que muestras son más relevantes en base a las medidas del sensor. Desde la línea 5 hasta la línea 8 se hace el llamado re muestreo que consiste en quedarnos con las muestras que tengan mayor probabilidad. Nótese que en la línea 6 se genera una muestra de un índice de acuerdo a los pesos de las muestras. Y en la línea 7 se guarda la muestra que corresponde a este índice.

4.2.3 Resultados

Luego de haber implementado el algoritmo descrito en la sección anterior, a continuación se muestran los resultados obtenidos usando el mapa descrito en la sección 3.1.2. El objetivo de estas pruebas es comprobar si el filtro Monte Carlo estándar es capaz de estimar correctamente la configuración del robot en cualquier instante de tiempo. Además se busca analizar como varia el desempeño del algoritmo bajo diferentes números de muestras.

En las Figuras 4.2, 4.3 y 4.4 que se muestran a continuación las líneas en rojo representan la trayectoria real del robot, y las trayectorias en azul representan la trayectoria estimada usando el método de localización Monte Carlo estándar. Si estas líneas coinciden esto significa que el algoritmo es capaz de localizar correctamente al robot. Además, con el fin de cuantificar los errores en la estimación, para cada prueba se ha realizado 10 experimentos, y se han obtenido los errores RMS de cada experimento, los cuales se reportan en las tablas anexas 4.2, 4.3, y 4.4.

a. Prueba 01: Usando $M = 5$ muestras

En esta prueba se analiza el desempeño del algoritmo cuando se usa una cantidad muy pequeña de muestras.

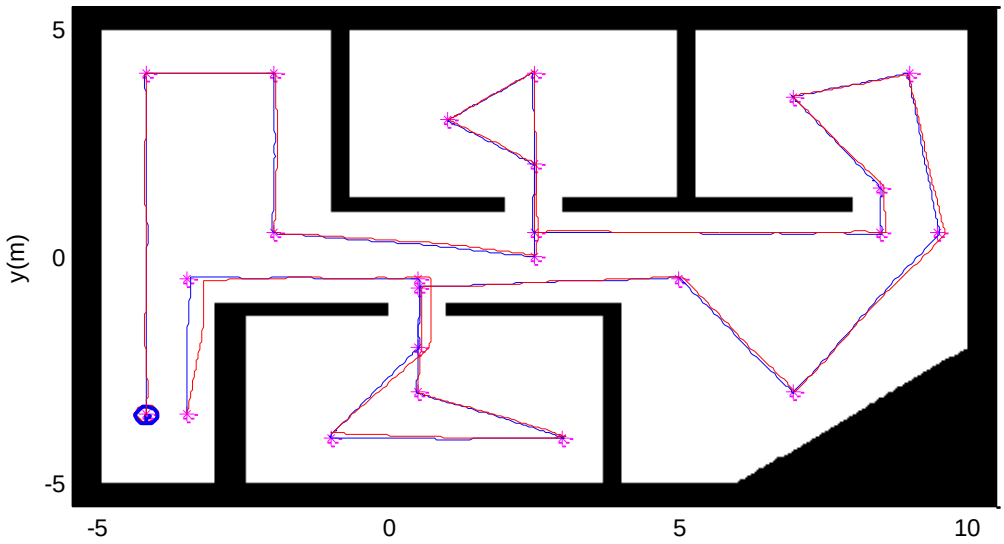


Figura 4.2. Localización usando el método Monte Carlo estándar – M=5 muestras

(Fuente: Elaboración propia)

Tabla. 4.2. Error RMS usando el método Monte Carlo estándar – M=5 muestras (Fuente: Elaboración propia)

Experimento	$E_{RMS}(x)$	$E_{RMS}(y)$	$E_{RMS}(\theta)$
1	0.0830	0.0334	0.0364
2	0.0302	0.0502	0.0217
3	0.0335	0.0223	0.0118
4	0.0315	0.0297	0.0220
5	0.0266	0.0506	0.0246
6	0.0408	0.0318	0.0257
7	0.0774	0.0281	0.0263
8	0.0464	0.0635	0.0333
9	0.0553	0.0763	0.0311
10	0.0387	0.0443	0.0237

b. Prueba 02: Usando $M = 10$ muestras

En esta prueba se analiza el desempeño del algoritmo cuando se usa una cantidad reducida de muestras.

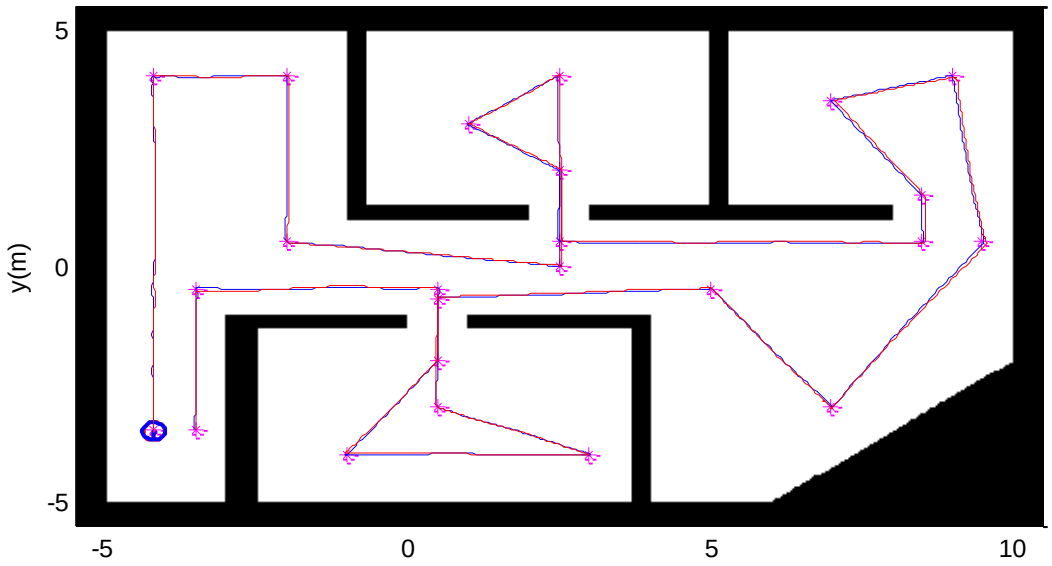


Figura 4.3. Localización usando el método Monte Carlo estándar – M=10 muestras (Fuente: Elaboración propia)

Tabla. 4.3. Error RMS usando el método Monte Carlo estándar – M=10 muestras (Fuente: Elaboración propia)

Experimento	$E_{RMS}(x)$	$E_{RMS}(y)$	$E_{RMS}(\theta)$
1	0.0178	0.0299	0.0172
2	0.0328	0.0254	0.0180
3	0.0335	0.0223	0.0118
4	0.0218	0.0324	0.0181
5	0.0315	0.0190	0.0158
6	0.0335	0.0251	0.0209
7	0.0251	0.0317	0.0202
8	0.0219	0.0379	0.0154
9	0.0240	0.0278	0.0175
10	0.0250	0.0234	0.0170

c. Prueba 03: Usando $M = 30$ muestras

En esta prueba se analiza el desempeño del algoritmo cuando se usa una cantidad moderada de muestras.

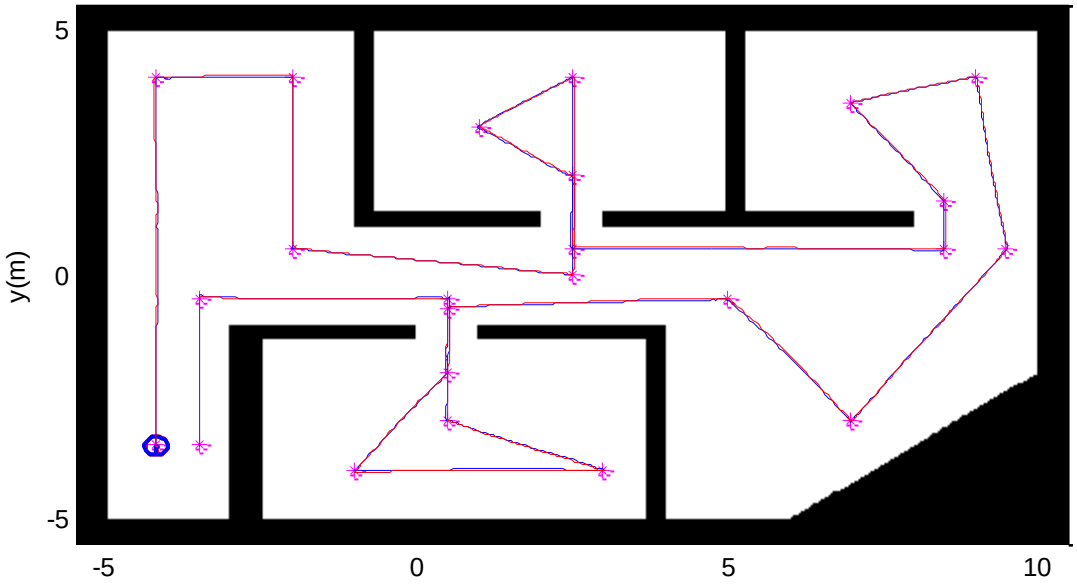


Figura 4.4. Localización usando el método Monte Carlo estándar – M=30 muestras (Fuente: Elaboración propia)

Tabla. 4.4. Error RMS usando el método Monte Carlo estándar – M=30 muestras (Fuente: Elaboración propia)

Experimento	$E_{RMS}(x)$	$E_{RMS}(y)$	$E_{RMS}(\theta)$
1	0.0138	0.0126	0.0134
2	0.0182	0.0175	0.0120
3	0.0161	0.0161	0.0141
4	0.0151	0.0213	0.0131
5	0.0144	0.0156	0.0134
6	0.0122	0.0174	0.0133
7	0.0121	0.0201	0.0136
8	0.0179	0.0191	0.0124
9	0.0184	0.0219	0.0122
10	0.0184	0.0254	0.0159

En base a los resultados anteriores, la primera conclusión es que el algoritmo de localización Monte Carlo estándar es capaz de estimar con gran precisión la configuración del robot en cualquier instante de tiempo. Esto se desprende del hecho que la trayectoria estimada por el algoritmo (trayectoria en azul) es similar a la trayectoria real del robot (trayectoria en rojo).

Por otro lado, tal como se puede observar de los resultados obtenidos en las Tablas, 4.2, 4.3 y 4.4 la calidad de la estimación depende de la cantidad de muestras usadas. Cuando se usan pocas muestras, el error de estimación es considerable; y a medida que se usan más muestras, el error de estimación tiende a disminuir. Idealmente se podrían usar una gran cantidad de muestras con el fin de tener errores aún más pequeños, pero la desventaja de dicho enfoque es que la carga computacional crece de manera considerable, lo que haría que el robot no funcione en tiempo real.

Finalmente, se debe mencionar que una desventaja de este algoritmo es que al hacer el paso de re-muestreo en cada instante de tiempo, puede presentar problemas de perdida de la diversidad [21]. Esto se debe al que luego de hacer el paso de re-muestreo hay muchas muestras que se repiten, lo que ocasiona que la perdida de la diversidad de las muestras.

d. Eficiencia del Algoritmo de Re-muestreo Estándar

Finalmente, otro punto importante a analizar es la eficiencia del algoritmo de re-muestreo estándar. En Figura 4.5 se muestran los diferentes tiempos de ejecución de este algoritmo para diversos tamaños de muestras.

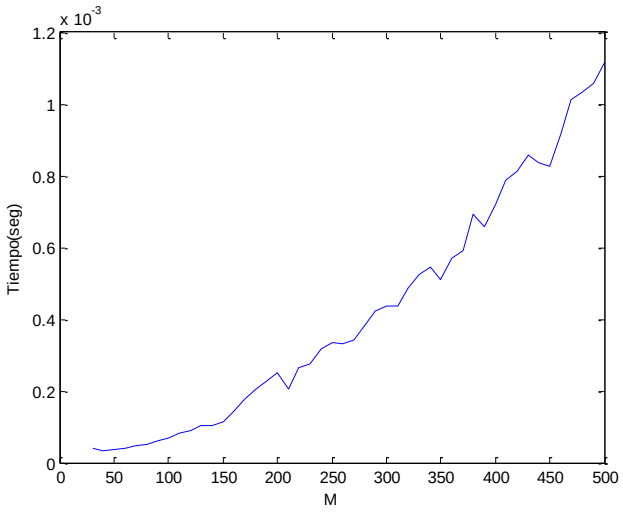


Figura 4.5. Tiempo de ejecución del algoritmo de re-muestro estándar (Fuente: Elaboración propia)

Como puede notarse el tiempo de ejecución está en el orden de los milisegundos, y tal como se muestra en [19] tiene un orden de operaciones de $O(M \ln(M))$. En el diseño de robots autónomos, es de suma importancia el uso de algoritmos de re-muestreo que sean robustos y eficientes. Por lo que en la siguiente sección se propondrá el uso de técnicas de re-muestreo más sofisticadas.

4.3 Localización usando Métodos Monte Carlo Mejorados

El algoritmo Monte Carlo estándar descrito en la sección anterior trabaja muy bien en la tarea de localización; sin embargo, se le pueden hacer una serie de mejoras con el fin de mejorar su eficiencia y hacerlo más robusto. En esta sección se describirán estas mejoras, y se compararan los resultados obtenidos con los resultados que se tienen usando el método Monte Carlo estándar.

4.3.1 Mejoras Propuestas

En base a las observaciones hechas en la sección 1.4.3, se pueden formular las siguientes mejoras al algoritmo Monte Carlo estándar.

a) Reducción de la frecuencia de re-muestreo. La primera mejora que se puede hacer es la reducción de la frecuencia de re-muestreo. La idea es integrar las mediciones en diversos instantes de tiempo usando la siguiente formula de actualización de pesos

$$w_t^{(m)} = \begin{cases} 1 & si\ se\ ha\ hecho\ re\ muestreo \\ p(z_t \mid x_t^{(m)}) w_{t-1}^{(m)} & si\ no\ se\ ha\ hecho\ re\ muestreo \end{cases} \tag{4.3}$$

La elección sobre la frecuencia de re-muestreo no es muy simple, ya que si el re-muestreo se realiza con mucha frecuencia se puede incurrir en el problema de perdida de diversidad; si por el contrario, el re-muestreo se hace con poca frecuencia, las muestras pueden caer en regiones de pequeña probabilidad presentando el problema de degeneración.

Tal como se discutió en la sección 1.4.3 una medida del fenómeno de degeneración está dada por el tamaño efectivo de las muestras N_{eff}. Entonces, tal como se muestra en [21], la idea es hacer el paso de re-muestreo cada vez que el tamaño efectivo de las muestras sea menor que $N_s / 2$. Donde N_s se refiere al número de muestras. De esta manera se logra tener un re-muestreo del tipo adaptivo, en el sentido que este solo se realiza cuando es necesario.

Lo que se logra al tener un re-muestreo adaptivo es que se reduce el problema de la perdida de la diversidad. Otra ventaja adicional que se gana al hacer el re-muestreo solo cuando es necesario es el ahorro de costos computacionales. Debido a que el robot debe operar en tiempo real, cada ahorro en tiempo computacional es muy importante ya que de esta manera podremos ejecutar algoritmos que demanden altos tiempos de ejecución.

b) Uso de algoritmos de re-muestreo más sofisticados. La segunda mejora que se puede hacer al algoritmo Monte Carlo estándar, es el uso de algoritmos de re muestreo más sofisticados, que sean más inmunes al problema de la varianza. En la

práctica existen dos algoritmos que se pueden usar para hacer el paso de re-muestreo: el muestreo estratificado y el muestreo sistemático [17]. La gran ventaja de usar este tipo de algoritmos es que: cubren el espacio de muestra de manera sistemática, presentan menos problemas de varianza, y su eficiencia esta en el orden de operaciones $O(\log(M))$ [19]. Esta última ventaja permite reducir el tiempo computacional en el re-muestreo de las muestras.

En la Tabla 4.5 se muestra el algoritmo de re-muestreo sistemático implementado en el presente trabajo. Las entradas de este algoritmo son las muestras con sus respectivos pesos, y la salida es el nuevo conjunto de muestras. Nótese que las nuevas muestras, tienen igual peso, y representan mejor la distribución bajo estudio (sección 1.4).

Tabla. 4.5. Algoritmo de re-muestreo sistemático (Fuente: http://www.probabilistic-robotics.org/)

$Algoritmo\ \ Low\_variance\_sampler\left(\left\{x_t^{(m)}\right\},\left\{w_t^{(m)}\right\}\right)$

$L1:\ \ r = rand(0:M^{-1})$

$L2:\ \ c = w_t^{(1)}$

$L3:\ \ i = 1$

$L4:\ \ for\ \ m = 1:M$

$L5:\ \ \ \ U = r + (m-1)\cdot M^{-1}$

$L6:\ \ \ \ while U > c$

$L7:\ \ \ \ \ \ i = i + 1$

$L8:\ \ \ \ \ \ c = c + w_t^{(i)}$

$L9:\ \ \ \ end$

$L10:\ \ \ \ \bar{x}_t^{(m)} = x_t^{(i)}$

$L11:\ end$

$return\ \ \left\{\bar{x}_t^{m}\right\}$

4.3.2 Algoritmo de Localización

Luego de adaptar todas las mejoras propuestas descritas en la sección anterior al algoritmo Monte Carlo estándar, en la Tabla 4.6 se muestra el algoritmo de localización Monte Carlo mejorado. Donde las mejoras se refieren a la reducción de la frecuencia de muestreo, y al uso de técnicas de re muestreo más sofisticadas

Tabla. 4.6. Algoritmo de localización Monte Carlo secuencial mejorado. (Fuente: Elaboración propia)

$Algoritmo \;\; Localizacion\_MC(\{x_{t-1}^m\},\{w_{t-1}^m\},u_t,z_t)$
$L1:\;\; for \;\; m=1:M$
$L2:\quad x_t^m \sim p(x_t \mid u_t, x_{t-1}^m)$
$L3:\quad w_t^m = p(z_t \mid x_t^m)\cdot w_{t-1}^m$
$L4:\; end$
$L5:\;\; N_{eff} = 1/\sum_{m=1}^{M}\left(w_t^m\right)^2$
$L6:\;\; if \;\; N_{eff} < M/2$
$L7:\quad Ejecutar\ algoritmo\ de\ remuestreo$
$L8:\quad w_t^m = 1$
$L9:\; end$
$return \;\; \{x_t^m\}$

En este algoritmo las primeras cuatro líneas son idénticas a las del algoritmo Monte Carlo estándar con la diferencia de la acumulación de los pesos que se da en la línea 3. En la línea 5 se halla el tamaño efectivo de las muestras. Desde la línea 6 hasta la línea 9 se hace el paso de re-muestreo solo si el tamaño efectivo de las muestras es menor que $M/2$. En la línea 7 se ejecuta el algoritmo de re-muestreo descrito en la Tabla 4.5.

4.3.3 Resultados

Luego de haber implementado el algoritmo descrito en la sección anterior, a continuación se muestran los resultados obtenidos usando el mapa descrito en la sección 3.1.2. El objetivo de estas pruebas es comprobar si el filtro Monte Carlo mejorado es capaz de estimar la configuración del robot en cualquier instante de tiempo. Además se busca analizar el desempeño del algoritmo bajo diferentes números de muestras.

En las Figuras 4.6, 4.7 y 4.8 que se muestran a continuación las líneas en rojo representan la trayectoria real del robot, y las trayectorias en magenta representan la trayectoria estimada usando el método Monte Carlo. Además, con el fin de cuantificar los errores en la estimulación, para cada prueba se ha realizado 10 experimentos, y los errores RMS de cada experimento se reportan en las Tablas anexas 4.7, 4.8 y 4.9. Nótese que los errores RMS en las variables x, y están en metros, y los errores RMS en la variable θ están en radianes.

a. Prueba 01: Usando $M = 5$ muestras

En esta prueba se analiza el desempeño del algoritmo cuando se usa una cantidad muy pequeña de muestras.

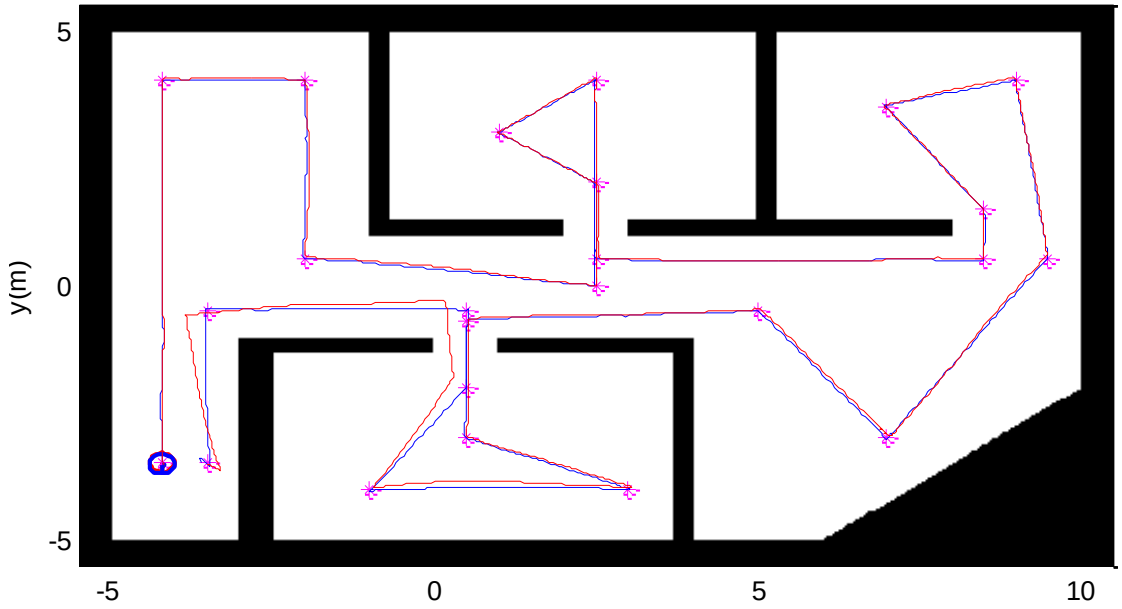


Figura 4.6. Localización usando el método Monte Carlo mejorado – M=5 muestras (Fuente: Elaboración propia)

Tabla. 4.7. Error RMS usando el método Monte Carlo mejorado – M=5 muestras (Fuente: Elaboración propia)

Experimento	$E_{RMS}(x)$	$E_{RMS}(y)$	$E_{RMS}(\theta)$
1	0.0596	0.0727	0.0372
2	0.0276	0.0265	0.0239
3	0.0371	0.0592	0.0260
4	0.1032	0.0638	0.0630
5	0.0328	0.0411	0.0269
6	0.0283	0.0349	0.0224
7	0.0278	0.0417	0.0227
8	0.1669	0.1201	0.0587
9	0.0224	0.0767	0.0326
10	0.0235	0.0340	0.0265

b. Prueba 02: Usando $M = 10$ muestras

En esta prueba se analiza el desempeño del algoritmo cuando se usa una cantidad reducida de muestras.

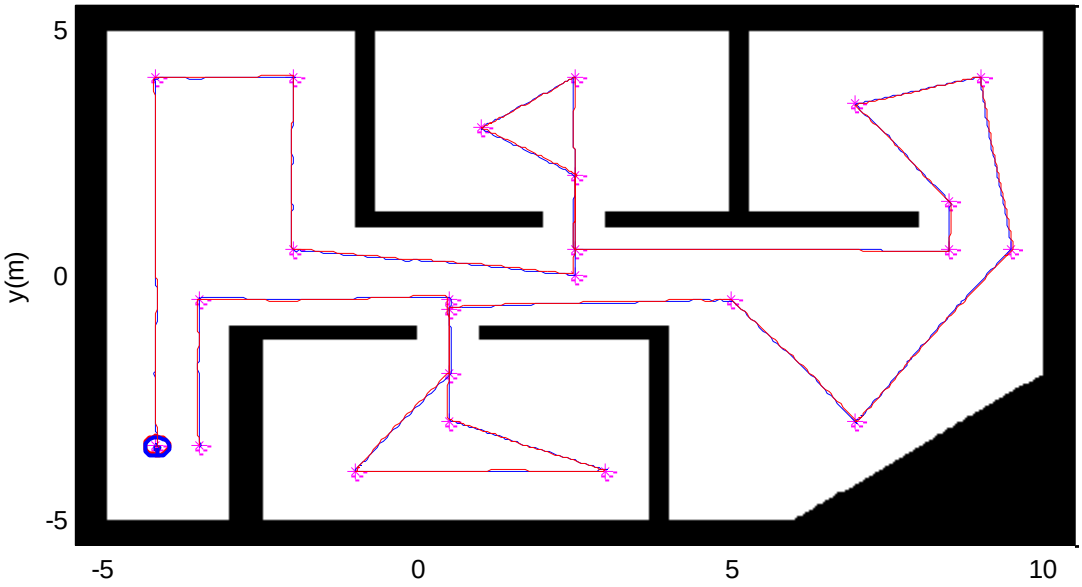


Figura 4.7. Localización usando el método Monte Carlo mejorado - M=10 muestras

(Fuente: Elaboración propia)

Tabla. 4.8. Error RMS usando el método Monte Carlo mejorado – M=10 muestras

(Fuente: Elaboración propia)

Experimento	$E_{RMS}(x)$	$E_{RMS}(y)$	$E_{RMS}(\theta)$
1	0.0181	0.0224	0.0182
2	0.0148	0.0400	0.0162
3	0.0212	0.0310	0.0205
4	0.0216	0.0578	0.0269
5	0.0237	0.0206	0.0163
6	0.0185	0.0283	0.0185
7	0.0180	0.0494	0.0221
8	0.0204	0.0324	0.0190
9	0.0236	0.0270	0.0206
10	0.0235	0.0240	0.0186

c. Prueba 03: Usando $M = 30$ muestras

En esta prueba se analiza el desempeño del algoritmo cuando se usa una cantidad moderada de muestras.

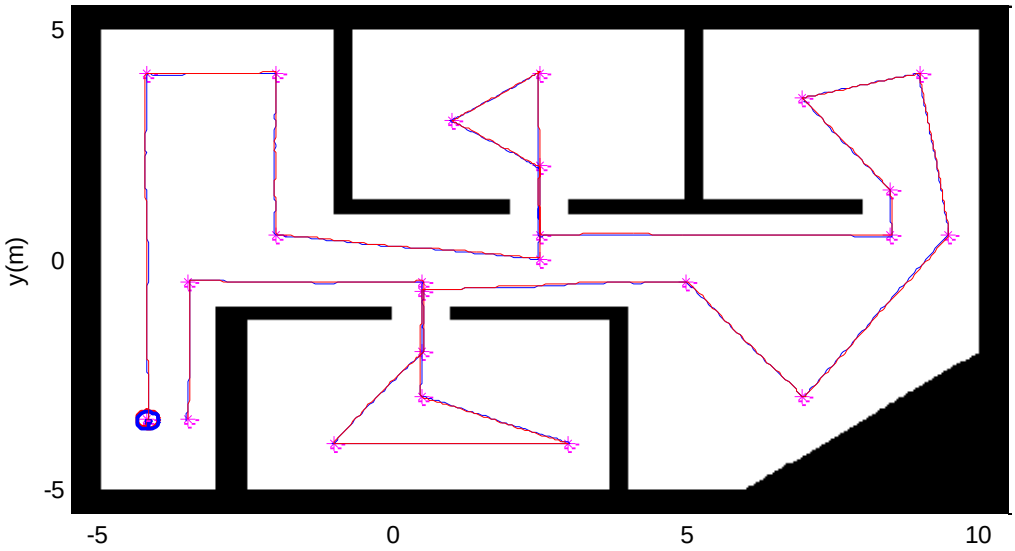


Figura 4.8. Localización usando el método Monte Carlo mejorado – M=30 muestras

(Fuente: Elaboración propia)

Tabla. 4.9. Error RMS usando el método Monte Carlo mejorado – M=30 muestras

(Fuente: Elaboración propia)

Experimento	$E_{RMS}(x)$	$E_{RMS}(y)$	$E_{RMS}(\theta)$
1	0.0138	0.0145	0.0145
2	0.0162	0.0166	0.0157
3	0.0118	0.0192	0.0153
4	0.0120	0.0190	0.0150
5	0.0155	0.0171	0.0165
6	0.0166	0.0324	0.0157
7	0.0161	0.0166	0.0148
8	0.0125	0.0200	0.0159
9	0.0117	0.0180	0.0152
10	0.0142	0.0179	0.0152

En base a los resultados anteriores, la primera conclusión es que el algoritmo de localización Monte Carlo mejorado es capaz de estimar con gran precisión la configuración del robot en cualquier instante de tiempo. Esto se desprende del hecho que la trayectoria estimada por el algoritmo (trayectoria en azul) es similar a la trayectoria real del robot (trayectoria en rojo). Y al igual que en el caso del algoritmo Monte Carlo estándar, el error en la estimación disminuye al aumentar el número de muestras. Comparando los resultados obtenidos con los del algoritmo Monte Carlo estándar, se tiene que los errores en la estimación son comparables. Esto se puede apreciar al ver las Tablas 4.10 y 4.11 donde se muestran los errores RMS promedio de ambos algoritmos al usar diferentes números de partículas.

Tabla. 4.10. Errores RMS promedio usando el método Monte Carlo estándar (Fuente: Elaboración propia)

	Eje X(m)	**Eje Y(m)**	**Orientación(rad)**
$M = 5$	0.0463	0.0430	0.0257
$M = 10$	0.0267	0.0275	0.0172
$M = 30$	0.0157	0.0187	0.0133

Tabla. 4.11. Errores RMS promedio usando el método Monte Carlo mejorado (Fuente: Elaboración propia)

	Eje X(m)	**Eje Y(m)**	**Orientación(rad)**
$M = 5$	0.0529	0.0571	0.0340
$M = 10$	0.0203	0.0333	0.0197
$M = 30$	0.0140	0.0191	0.0154

Nótese que los errores de ambos algoritmos al estimar la posición del robot están en el orden de los centímetros, y el error en la estimación de la orientación no pasa de los dos grados sexagesimales. Estos resultados nos muestran la gran precisión de ambos algoritmos en la estimación de la configuración del robot.

Finalmente, si bien en el diseño final de nuestro robot móvil se podría usar un número de muestras reducido, en la práctica se recomienda usar un número de muestras moderado, tal como $M = 30$, debido a que en los entornos reales de navegación existen una serie de factores que generan ruidos que no han sido modeladas por el algoritmo. Además, esto se hace con el fin que el sistema de localización sea más robusto.

d. Eficiencia del Algoritmo de Re-muestreo Estándar

Finalmente en la Figura 4.9 se muestran los tiempos de ejecución del algoritmo de re-muestreo sistemático para diversos tamaños de muestras. Comparando estos resultados con los de la Figura 4.5 puede notarse que los tiempos de ejecución del algoritmo de re-muestreo sistemático son mucho menores que los del algoritmo de re muestreo estándar por un factor aproximado de 10.

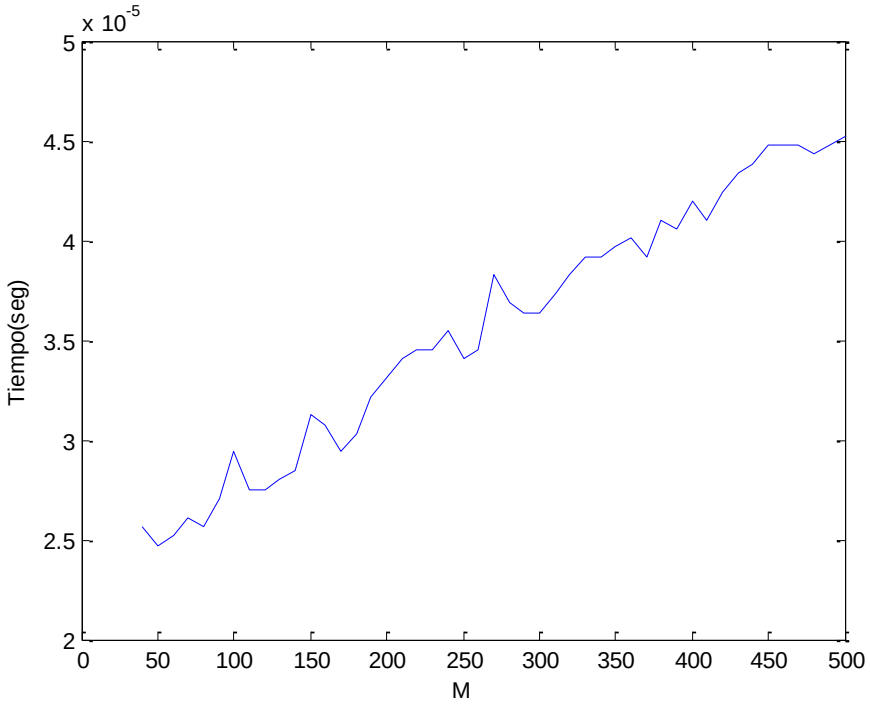


Figura 4.9. Tiempo de ejecución del algoritmo de re-muestro sistemático (Fuente: Elaboración propia)

4.4 Conclusiones

Como puede desprenderse de los experimentos anteriores, los métodos Monte Carlo secuenciales proporcionan una herramienta poderosa para estimar la configuración del robot en cualquier instante de tiempo usando las mediciones de los sensores laser. Los resultados obtenidos nos muestran que el error en la estimación de la posición está en el orden de los centímetros, y el error en la estimación de la orientación no pasa de los dos tres grados sexagesimales.

La gran ventaja del algoritmo Monte Carlo mejorado presentado en este trabajo sobre el algoritmo Monte Carlo estándar, es que es más robusto al problema de pérdida de la diversidad de las muestras, al disminuir la frecuencia de re-muestreo; y al hacer uso de algoritmos de re-muestro más avanzados se logra disminuir los problemas de la varianza en las muestras. Además, estas mejoras permiten reducir los tiempos computacionales requeridos por el algoritmo de estimación, siendo este punto muy importante, ya que el robot debe operar en tiempo real.

CAPÍTULO V
MAPEO USANDO MÉTODOS MONTE CARLO

El objetivo del presente capitulo consiste en la implementación de métodos de mapeo usando métodos Monte Carlo. El mapeo es uno de los problemas más importantes en la navegación autónoma de robots móviles, ya que un requisito esencial para la navegación autónoma de un robot es que este tenga un mapa de su entorno de navegación. En este capítulo se mostraran los resultados de la implementación del algoritmo de mapeo Monte Carlo estándar, y su versión mejorada, el algoritmo Monte Carlo Rao-Blackwellized mejorado.

5.1 El Problema del Mapeo

El problema del mapeo se refiere a la estimación del mapa de navegación de un robot usando las mediciones “ruidosas” de sus sensores (Figura 5.1). Debido a la complejidad del mapeo, este problema representa el tema central de investigación en las universidades y centros de investigación más avanzados a nivel mundial (MIT, CMU, Stanford, etc.).

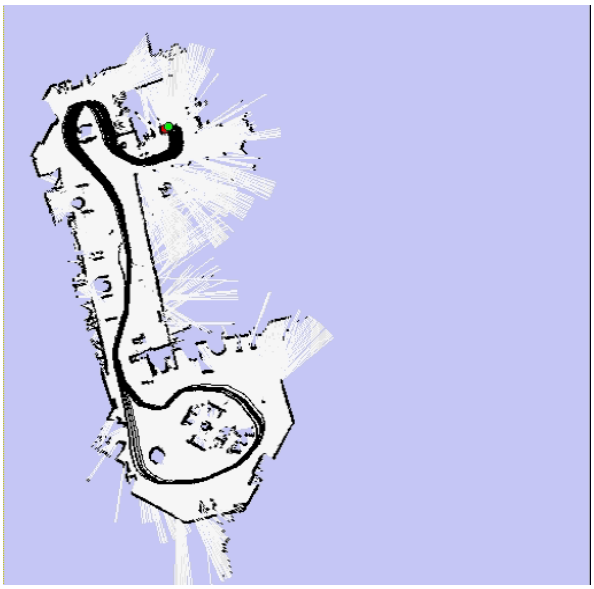


Figura 5.1. El problema del mapeo o construcción de mapas (Fuente http://robots.stanford.edu/videos.html)

5.1.1 Construcción Manual de Mapas

La manera más simple de obtener un mapa de navegación es mediante la intervención de un operador humano. Es decir, se encarga a una determinada persona que, en base a un dimensionamiento del entorno de navegación y una inspección visual de los espacios ocupados y no ocupados, construya un mapa de ocupación que describa el entorno entre espacios ocupados y no ocupados. Para esto el primer paso es la obtención del plano arquitectónico del entorno de navegación (Figura 5.2).

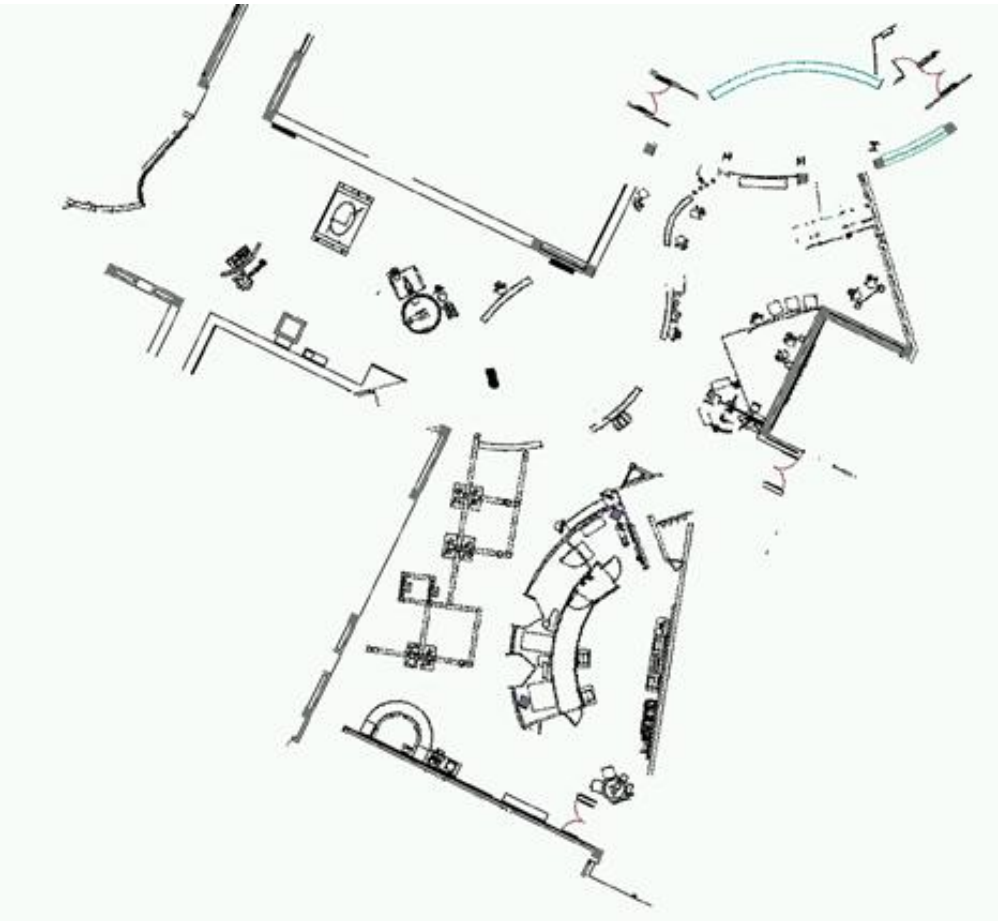


Figura 5.2. Plano arquitectónico de un entorno de navegación (Fuente: http://www.probabilistic-robotics.org/)

El siguiente paso consiste en la construcción de la matriz o mapa de ocupación (Figura 5.3). Tal como se mencionó en el Capítulo 3, en esta matriz las celdas que representan espacios ocupados tienen un valor de 1, y las celdas que corresponden a espacios libres tienen un valor de 0. Los espacios no explorados tienen un valor de 0.5.

Figura 5.3. Mapa de ocupación de un entorno de navegación (Fuente: http://www.probabilistic-robotics.org/)

El principal problema del enfoque de la construcción manual de mapas es que si queremos que el robot desarrolle tareas de navegación en entornos nuevos, es necesario que el operador humano construya los mapas de navegación. Esta tarea es usualmente muy tediosa ya que por lo general no se cuenta con los planos arquitectónicos de dichos ambientes; y aun si se contara con los mismos, estos no reflejan fielmente el entorno de navegación (presencia de sillas, mesas, etc.).

5.1.2 Construcción Automática de Mapas

Para que un robot se considere completamente autónomo debe ser capaz de construir mapas de su entorno de navegación de manera automática usando solamente las mediciones de sus sensores [20]. Este problema, que es el centro principal de la investigación en el campo de la robótica móvil, es muy difícil de resolver debido a que el

problema de estimar mapas involucra el manejo de espacios estados de cientos y hasta miles de dimensiones.

Para que un robot pueda construir un mapa de su ambiente de navegación se requiere que este conozca su configuración. Por el contrario, para que el robot conozca su configuración se requiere que este tenga un mapa de su entorno. Debido a esta dificultad, el problema del mapeo es comúnmente llamado como el problema de *localización y mapeo simultaneo (SLAM – Simultaneous localization and mapping)* [18], [19]. Entonces, de aquí en adelante cuando se mencione el término de mapeo nos estaremos refiriendo al problema de localización y mapeo simultáneo.

5.2 Mapeo usando Métodos Probabilísticos

Al igual que en el caso del problema de localización, todos los algoritmos probabilísticos que tienen como objetivo resolver el problema del mapeo tienen como punto de partida el Filtro de Bayes descrito en la sección 1.3 del presente trabajo. La diferencia entre estos algoritmos se da en la manera de modelar o representar las leyes probabilísticas $p(x_t \mid u_t, x_{t-1})$ y $p(z_t \mid x_t)$, y la función de densidad de distribución posterior $bel(x_t) = p(x_t \mid z_{1:t}, u_{1:t})$.

Debe notarse que las técnicas probabilísticas, son las únicas que han demostrado éxito en el problema del mapeo. Esto debido a que el nivel de ruido/incertidumbre es mucho mejor manejado usando la teoría de probabilidades y técnicas estadísticas, que usando técnicas del tipo determinísticas. Tanto en la robótica móvil como en otras áreas de la ciencia, las técnicas probabilísticas representan el estado del arte.

5.2.1 El Filtro de Bayes en el Mapeo

En el problema del mapeo, el estado "y" a estimar comprende el mapa del entorno y la configuración del robot. Si denotamos el mapa por m y la configuración del robot por x, entonces el vector de estado está dado por:

$$y = (m, x) \tag{5.1}$$

Reemplazando esta ecuación en la ecuación (1.6) tendremos la ecuación del filtro de Bayes en el caso del problema del mapeo.

$$p(x_t, m_t \mid z_{1:t}, u_{1:t}) = \eta \cdot p(z_t \mid x_t, m_t) \iint p(x_t, m_t \mid u_t, x_{t-1}, m_{t-1}) \\ p(x_{t-1}, m_{t-1} \mid z_{1:t-1}, u_{1:t-1}) dx_{t-1} dm_{t-1} \tag{5.2}$$

Si asumimos que el entorno de navegación no cambia en el tiempo, tendremos $m_t = m_{t-1}$. De donde se tiene que el término $p(x_t, m_t \mid u_t, x_{t-1}, m_{t-1})$ es diferente de cero solo sí $m_t = m_{t-1}$. Reemplazando esta condición en (5.2) se tiene:

$$p(x_t, m_t \mid z_{1:t}, u_{1:t}) = \eta \cdot p(z_t \mid x_t, m_t) \int p(x_t \mid u_t, x_{t-1}, m_{t-1}) p(x_{t-1}, m_{t-1} \mid z_{1:t-1}, u_{1:t-1}) dx_{t-1} \tag{5.3}$$

Eliminando los índices de tiempo de las variables del mapa, resulta en:

$$p(x_t, m \mid z_{1:t}, u_{1:t}) = \eta \cdot p(z_t \mid x_t, m) \int p(x_t \mid u_t, x_{t-1}, m) \cdot p(x_{t-1}, m \mid z_{1:t-1}, u_{1:t-1}) dx_{t-1} \tag{5.4}$$

Finalmente, asumiendo que el mapa *m* no tiene efecto en la descripción probabilística del movimiento del robot tenemos la ecuación (5.5), que representa el punto de partida de muchos algoritmos de mapeo

$$p(x_t, m \mid z_{1:t}, u_{1:t}) = \eta \cdot p(z_t \mid x_t, m) \int p(x_t \mid u_t, x_{t-1}) \cdot p(x_{t-1}, m \mid z_{1:t-1}, u_{1:t-1}) dx_{t-1} \tag{5.5}$$

5.2.2 Mapeo usando el Método Monte Carlo Estándar

En el problema del mapeo, el cálculo de la distribución posterior que se muestra en la ecuación (5.5) es usualmente intratable debido a la gran dimensionalidad del espacio estado. Como se mencionó anteriormente, el problema del mapeo es la gran dimensionalidad del espacio de mapas, debido a las casi infinitas combinaciones de posibles mapas. Por lo que muchas veces se requiere hacer ciertas asunciones con el fin de tener algoritmos que aproximen el cálculo de la distribución (5.5).

La familia de algoritmos SLAM basada en los sensores del tipo 'range-bearing' y la asunción de ruido gausianos en los sensores y los actuadores, puede aproximar esta distribución posterior usando distribuciones gausianas [19], [22]. Sin embargo estos algoritmos no pueden usarse con los sensores laser, ni con los mapas de ocupación que son el objeto del presente trabajo. El ejemplo más conocido de esta familia, es el algoritmo EKF-SLAM, que está basado en el filtro extendido de Kalman.

En el caso de las aproximaciones discretas realizadas por los métodos Monte Carlo estándar, la cantidad de muestras requeridas para cubrir de manera adecuada el espacio estado es muy grande. Este problema se vuelve aún mayor conforme la dimensión del entorno de navegación crece en longitud. Este hecho hace inaplicable el uso de los métodos Monte Carlo estándar de manera directa ya que al considerar una cantidad casi infinita de muestras, se requerirían una cantidad infinita de recursos computacionales. En las secciones posteriores se describirán las maneras de superar este obstáculo haciendo uso de técnicas más avanzadas.

5.2.3 Mapeo con Configuraciones Conocidas

Antes de pasar a ver los algoritmos de mapeo, es necesario revisar el algoritmo de *"mapeo con configuraciones conocidas",* el cual es un caso especial del problema del mapeo que se da cuando el robot conoce su configuración en cualquier instante de tiempo. Es decir, se asume que el robot conoce su configuración en el momento que los sensores realizan sus mediciones.

Las técnicas de mapeo con configuraciones conocidas buscan resolver la siguiente distribución $p(m \mid x_{1:t}, z_{1:t})$. Es decir, la función de probabilidad sobre el mapa dado de que se conoce las configuraciones $x_{1:t}$ del robot en las cuales se han tomado las mediciones $z_{1:t}$ [23]. La idea central de estos algoritmos es asignar una probabilidad a cada 'celda' del mapa de ocupación en función de su cercanía al cono de ocupación que modela las mediciones del sensor laser (Figura 5.4).

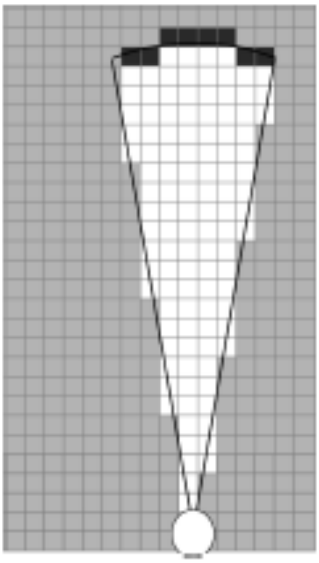


Figura 5.4. Cono de ocupación que modela las mediciones de un sensor laser (Fuente: http://www.probabilistic-robotics.org/)

En esta figura los puntos que están fuera del alcance del cono que encierra la medición del sensor laser reciben un valor neutro debido a que no se puede afirmar ni negar la presencia de un obstáculo en estas posiciones. Los puntos que están dentro del cono y que están cerca del rango de la medición obtenido por el sensor reciben una alta probabilidad de contener espacios ocupados ya que es altamente probable que estas posiciones contenga el obstáculo que género la medida del sensor. Los puntos que están dentro del cono pero con menor rango que las mediciones del sensor reciben poca probabilidad de contener espacios ocupados ya que es poco posible que contengan obstáculos.

Para ilustrar el funcionamiento del algoritmo de mapeo con posiciones conocidas, en la Figura 5.5 se muestran las mediciones de un sensor laser (que emite nueve rayos de luz) en un ambiente de navegación y en la Figura 5.6 se muestran los resultados del mapeo usando la posición conocida del robot. Como puede notarse, en las posiciones fuera de los conos que representan los rayos laser la probabilidad de ser un espacio ocupado es 0.5 ya que no se puede afirmar ni negar la presencia de un obstáculo, por lo que estos espacios están representados en color gris. En los puntos dentro del cono que coinciden con las mediciones de los sensores se tienen las posiciones con alta posibilidad de tener un espacio ocupado, que se representan por puntos en blanco. Y los puntos dentro de los conos pero con menor rango reciben poca probabilidad de tener espacios ocupados, por lo que se representan por espacios en blanco.

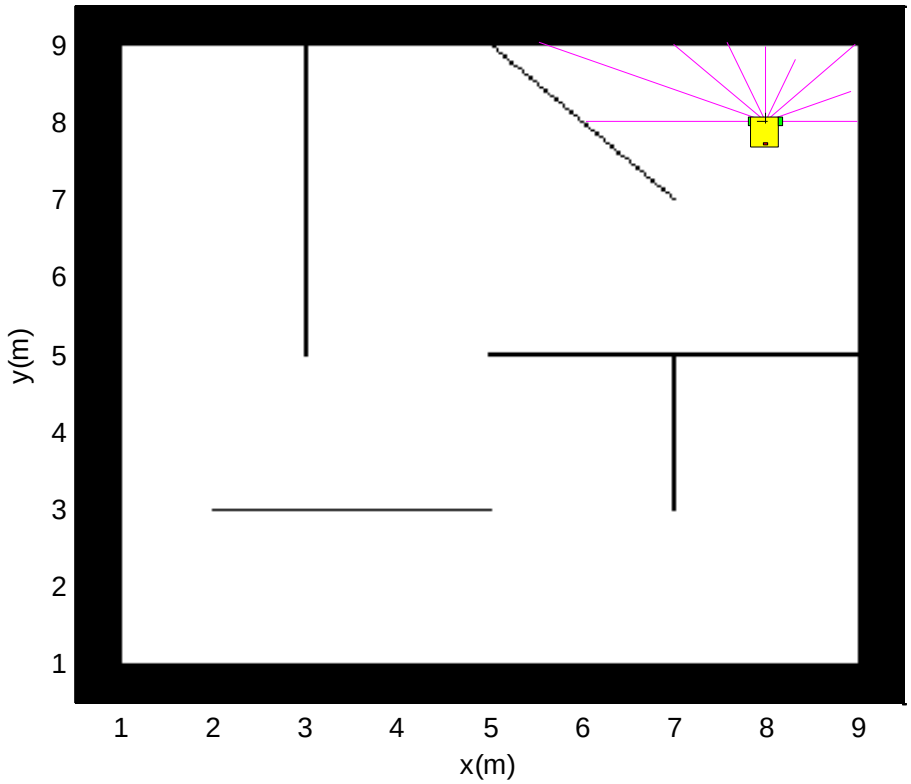


Figura 5.5. Mediciones de distancia de un sensor laser (Fuente: Elaboración propia)

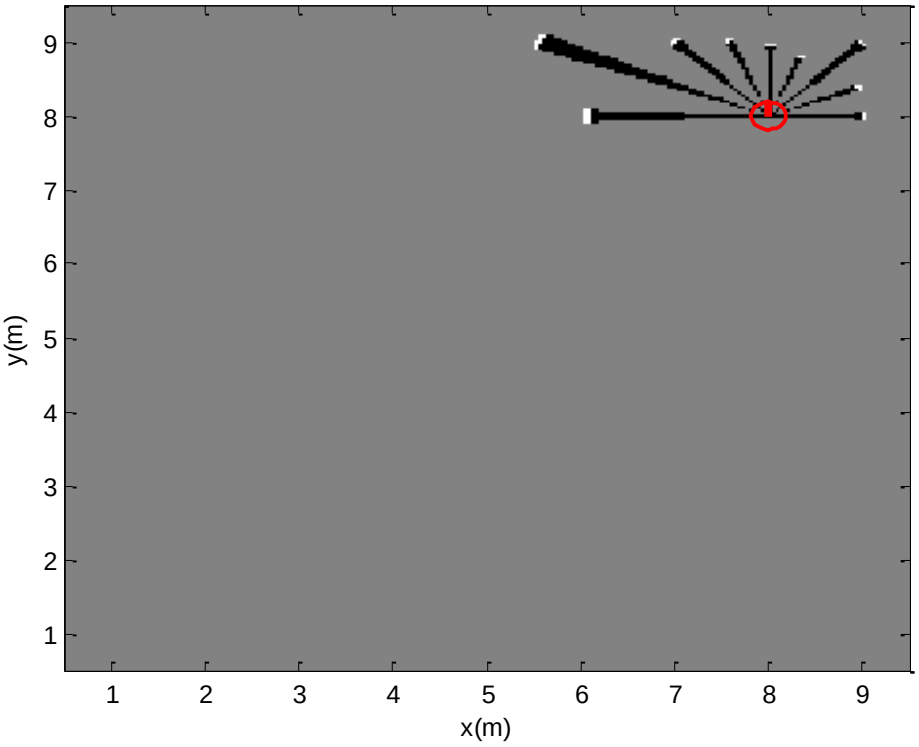


Figura 5.6. Mapeo con posiciones conocidas (Fuente: Elaboración propia)

5.3 Mapeo usando Métodos Monte Carlo Rao-Blackwellized

En esta sección nos centraremos en los algoritmos que constituyen el eje central de la investigación de la presente tesis: los algoritmos de mapeo usando métodos Monte Carlo Rao-Blackwellized. Como se mencionó en la sección anterior, el principal problema del método Monte Carlo estándar en el problema del mapeo es la gran dimensionalidad del espacio estado (debido al estado correspondiente al mapa). Tal como se verá en esta sección el uso de las técnicas del tipo Rao-Blackwellized hace tratable el problema de mapeo usando las técnicas Monte Carlo.

5.3.1 Conceptos Básicos

Una de las desventajas más importantes de los métodos Monte Carlo es el muestreo en espacios de alta dimensionalidad [21], como en el caso del problema de mapeo donde las dimensiones del espacio estado pueden estar fácilmente en el orden de los miles lo que hace inaplicable el uso de este tipo de técnicas.

Sin embargo, en algunos casos, los sistemas bajo estudio tienen una parte de su estructura que se puede manejar de manera analítica. La idea central de los algoritmos del tipo Rao-Blackwellized es explotar estas estructuras con el fin de usar métodos analíticos en esta parte de la estructura del problema, y usar métodos Monte Carlo en la parte restante. De esta manera hace posible la aplicación de los métodos Monte Carlo ya que la dimensionalidad del espacio de muestras se reduce de manera considerable.

En el caso específico del problema de mapeo la idea principal de los métodos del tipo Rao-Blackwellized es expresar la distribución posterior $p(x_{1:t}, m \mid z_{1:t}, u_{1:t})$ sobre el mapa y la trayectoria del robot mediante la siguiente factorización.

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t}) = p(m \mid x_{1:t}, z_{1:t}) \cdot p(x_{1:t} \mid z_{1:t}, u_{1:t}) \tag{5.6}$$

En esta factorización el primer término se refiere a la estimación del mapa dado que se conocen las configuraciones del robot y las mediciones de los sensores; y el segundo término se refiere a la estimación de las configuraciones del robot dado las mediciones de

los sensores y actuadores. El modelo probabilístico grafico que representa esta dinámica es el que se muestra en la Figura 5.7.

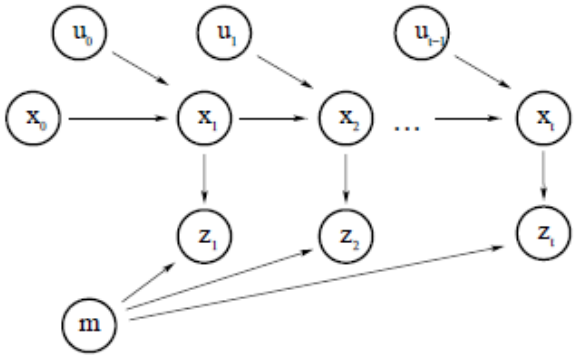


Figura 5.7. Modelo probabilístico grafico del problema del mapeo (Fuente: http://www.probabilistic-robotics.org/)

La importancia la factorización mostrada en la ecuación (5.6) es que permite redefinir el problema de estimar de manera conjunta el mapa y las configuraciones, en un problema donde se estiman de manera independiente tanto el mapa y la trayectoria del robot [24]. El punto principal aquí es reconocer que el primer término puede ser calculado analíticamente usando la técnica de '*mapeo con configuraciones conocidas*' que fue descrito en la sección anterior dado del presente trabajo. La estimación del segundo término, es decir las configuraciones del robot, se puede hacer usando el método Monte Carlo descrito en el Capítulo 4 de la presente tesis.

De esta manera, al usar los métodos Monte Carlo en combinación con las técnicas Rao-Blackwellized es posible resolver el problema de mapeo de manera eficiente, ya que el proceso de muestreo de las técnicas Monte Carlo solo se hace en el espacio de trayectorias (sin considerar el espacio de mapas de ocupación) reduciendo de manera considerable la dimensión del espacio de muestreo. Finalmente se debe comentar que estas técnicas actualmente representan el estado del arte en el problema del mapeo y son usados comúnmente en los robots más avanzados a nivel mundial.

5.3.2 Algoritmo de Mapeo

En la Tabla 5.1 se muestra el algoritmo SIS descrito en el Capítulo 1 adaptado para la tarea de mapeo usando los algoritmos Monte Carlo Rao-Blackwellized. Como puede notarse este algoritmo es muy similar al algoritmo de localización presentado en la Tabla

5.1 del capítulo anterior. La principal diferencia es que no solo se estiman las configuraciones del robot sino también el mapa de ocupación del entorno de navegación.

Tabla. 5.1. Algoritmo de mapeo usando el método Monte Carlo Rao-Blackwellized
(Fuente: http://www.probabilistic-robotics.org/)

$Algoritmo\ \ Mapeo\_MC\_Rao\_Blackwellized(\{x_{t-1}^{[k]}, m_{t-1}^{[k]}\}, u_t, z_t)$

$L1:\ \ for\ \ k = 1:M$

$L2:\quad x_t^{[k]} \sim q(x_t \mid u_t, x_{t-1}^{[k]})$

$L3:\quad w_t^{[k]} = \dfrac{p(z_t \mid x_t^{[k]}, m_{t-1}^{[k]}) \cdot p(x_t^{[k]} \mid x_{t-1}^{[k]}, u_{t-1})}{q(x_t^{[k]} \mid x_{t-1}^{[k]}, u_{t-1})}$

$L4:\quad m_t^{[k]} = update\_occupancy\_grid(z_t, x_t^{[k]}, m_{t-1}^{[k]})$

$L5:\ end$

$L6:\ \ for\ \ i = 1:M$

$L7:\quad Muestrear\ "m"\ con\ probabilidad\ \alpha\ w_t^m$

$L8:\quad x_t^{[i]} = x_t^{[m]}$

$L9:\ end$

$return\ \ \{x_t^{[k]}, m_t^{[k]}\}$

En este algoritmo, la línea L2 corresponde a la generación de muestras $x_t^{[k]}$ de la distribución propuesta $q(x_t \mid u_t, x_{t-1})$. La línea L3 se refiere al paso de asignación de pesos de las muestras. La línea L4 se encarga de actualizar los mapas de cada una de las muestras usando el método de mapeo con configuraciones conocidas descrito anteriormente. Finalmente en las líneas L6 y L9 se hace el paso de re-muestreo. Como puede notarse, en este algoritmo cada muestra representa una posible configuración del robot; y además, cada muestra tiene su propio mapa que representa la creencia de la muestra sobre la forma de su entorno de navegación.

Respecto a la forma de la distribución propuesta $q(x_t \mid u_t, x_{t-1})$, al igual que en el caso del problema de localización descrito en el Capítulo 4, en el algoritmo de mapeo Monte Carlo Rao-Blackwellized se puede hacer uso de la función de densidad de distribución de probabilidad de movimiento $p(x_t \mid u_t, x_{t-1})$ como la distribución propuesta. De esta manera, los pesos de cada muestra estarían dados simplemente por la expresión $w_t^{[k]} = p(z_t \mid x_t^{[k]}, m_{t-1}^{[k]})$, que es la función de probabilidad de las mediciones descrita en el Capítulo 3 del presente trabajo.

Por otro lado cabe notar que el parámetro principal de control del algoritmo se refiere al número de muestras a usar. Tal como se vio en el Capítulo 1, el uso de una gran cantidad de muestras permite una mejor precisión en la estimación pero vuelve al algoritmo muy costoso en términos computacionales. Por otro lado, el uso de pocas muestras permite que el algoritmo pueda operar en tiempo real (es decir el robot puede construir su mapa en línea mientras se desplaza por un ambiente), pero la estimación puede ser muy pobre o incluso totalmente errónea.

Antes de mostrar los resultados del mapeo, en la Tabla 5.2 se muestran los tiempos de ejecución de las operaciones más costosas en el proceso de mapeo. Como puede deducirse, al aumentar el número de muestras aumenta el tiempo de ejecución de manera proporcional, haciendo que, pasado cierto límite, el algoritmo de mapeo no pueda operar en tiempo real. Nótese que la operación más costosa está dada por la construcción del mapa, en la que el robot dado las mediciones de los sensores laser debe actualizar el mapa de ocupación (Figura 5.5 y 5.6). Para lograr este tiempo relativamente pequeño se deben hacer uso de técnicas altamente optimizadas de programación, caso contrario se pueden obtener tiempos de ejecución muy grandes (en una primera implementación se obtuvieron tiempos de alrededor de 5 segundos).

Tabla. 5.2. Tiempos de ejecución (por muestras) de las operaciones más costosas
(Fuente: Elaboración propia)

Operación	**Tiempo de ejecución**
Construcción de mapa	80 ms
Calculo de las probabilidades	1.3 ms
Otras operaciones	10 ms

5.3.3 Resultados

Luego de haber implementado el algoritmo de mapeo descrito en la Tabla 5.1 a continuación se mostraran los resultados obtenidos en diferentes pruebas, usando diferentes números de muestras. Para tal fin se usara como en entorno de navegación el que se usó en el capítulo de localización, el cual se muestra en la Figura 5.8. Entonces, el objetivo de estas pruebas, es evaluar el desempeño del algoritmo de mapeo en función de que tan bien sea capaz de construir este mapa.

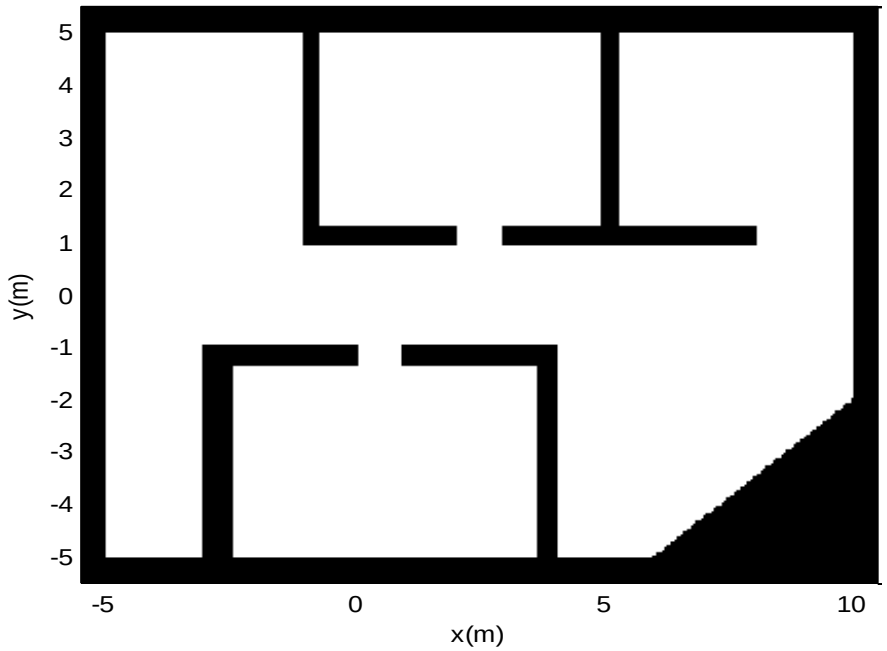


Figura 5.8. Mapa de ocupación de prueba en el mapeo (Fuente: Elaboración propia)

Debe notarse que este entorno de navegación es de un tamaño considerable, en el eje “x” la dimensión es de 15 metros, y en el eje “y” la dimensión es de 10 metros. Con el fin de construir mapas de ocupación de una gran precisión, se ha escogido usar una resolución del tamaño de las celdas de 5 cm, lo que resulta en una matriz de ocupación de 220×320. Esto resulta en una dimensión del espacio de mapas de 70400.

Como puede notarse, la dimensión del espacio de mapas es enorme, lo que hace inaplicable el uso de las técnicas Monte Carlo estándar en el problema de mapeo. Si bien se puede usar una resolución mayor con el fin de reducir esta dimensionalidad, el problema sería que los mapas construidos sean de menor calidad, incapaces de capturar todas las características del entorno de navegación. La desventaja de usar una menor resolución es que, si bien se puede obtener mapas de mejor calidad, los tiempos de ejecución son mucho más largos.

Luego de haber definido el entorno de navegación del robot a continuación se muestran los resultados del mapeo usando diferentes tamaños de muestras M. Debido a la naturaleza estocástica del algoritmo de mapeo, para cada valor de M se realizan varios experimentos con el fin de tener una estimación promedio del funcionamiento del algoritmo de mapeo.

a) Pruebas cuando M = 10

En estas pruebas se analiza los resultados del mapeo usando el método Monte Carlo Rao-Blackwellized usando $M = 10$ muestras. Las Figuras 5.9, 5.10, 5.11, 5.12, 5.13, 5.14, 5.15, 5.16, 5.17 y 5.18 corresponden a los mapas de mayor probabilidad. Solo en el caso de la primera prueba se muestran los mapas de ocupación de otras muestras. Como se recuerda, en este método, cada muestra lleva su propio mapa.

Primera prueba

Figura 5.9. Mapeo usando el método Monte Carlo Rao-Blackwellized cuando M=10 - Primera prueba (Fuente: Elaboración propia)

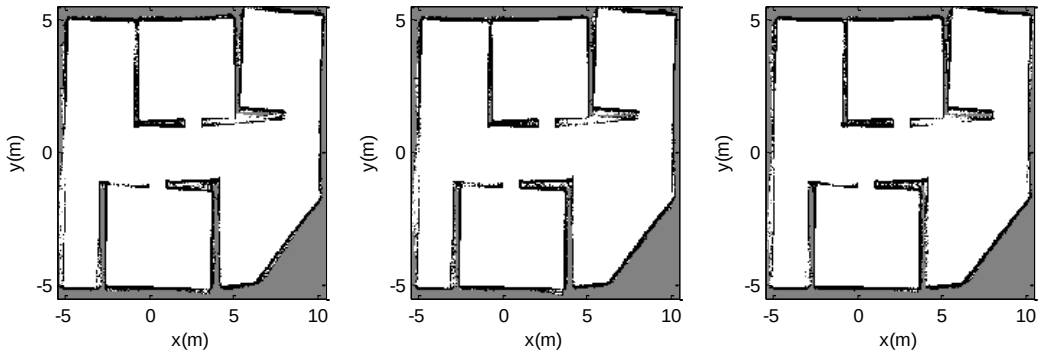


Figura 5.10. Mapas de ocupación de otras muestras (M=10) - Primera prueba (Fuente: Elaboración propia)

Segunda prueba

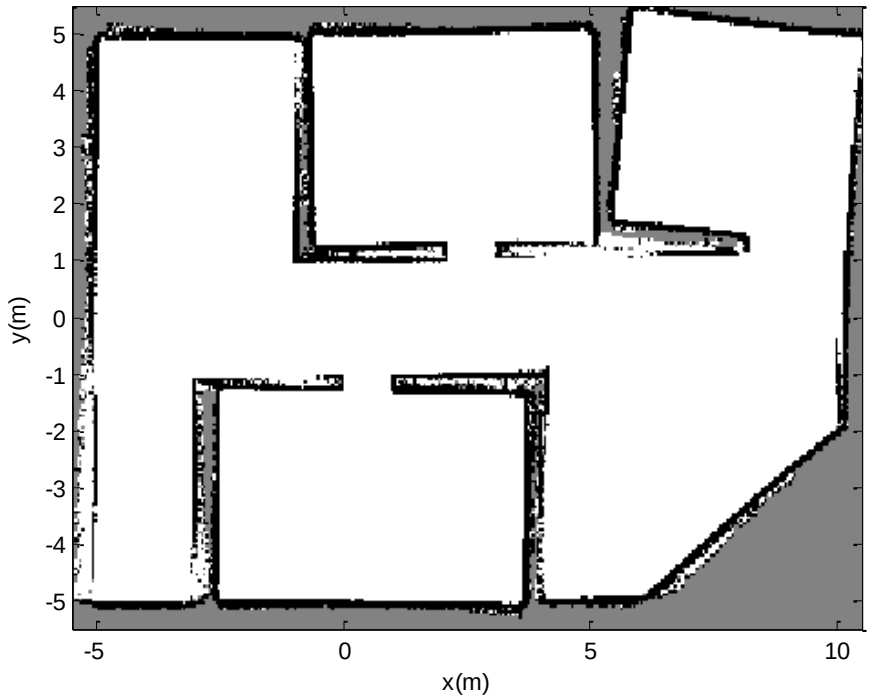


Figura 5.11. Mapeo usando el método Monte Carlo Rao-Blackwellized (M=10) - Segunda prueba (Fuente: Elaboración propia)

Tercera prueba

Figura 5.12. Mapeo usando el método Monte Carlo Rao-Blackwellized (M=10) - Tercera prueba (Fuente: Elaboración propia)

b) Pruebas cuando M = 30

Primera prueba

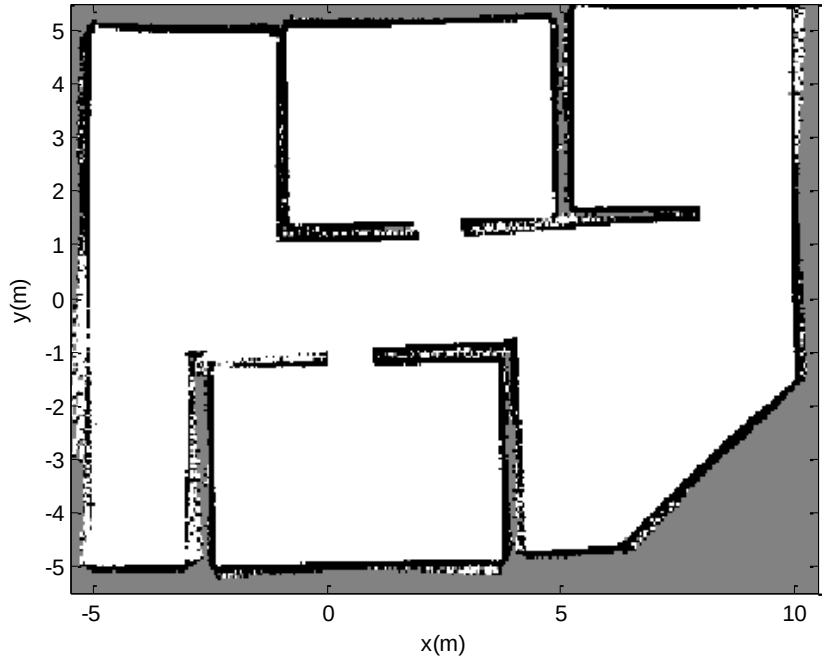


Figura 5.13. Mapeo usando el método Monte Carlo Rao-Blackwellized cuando M=30 - Primera prueba (Fuente: Elaboración propia)

Segunda prueba

Figura 5.14. Mapeo usando el método Monte Carlo Rao-Blackwellized cuando M=30 - Segunda prueba (Fuente: Elaboración propia)

Tercera prueba

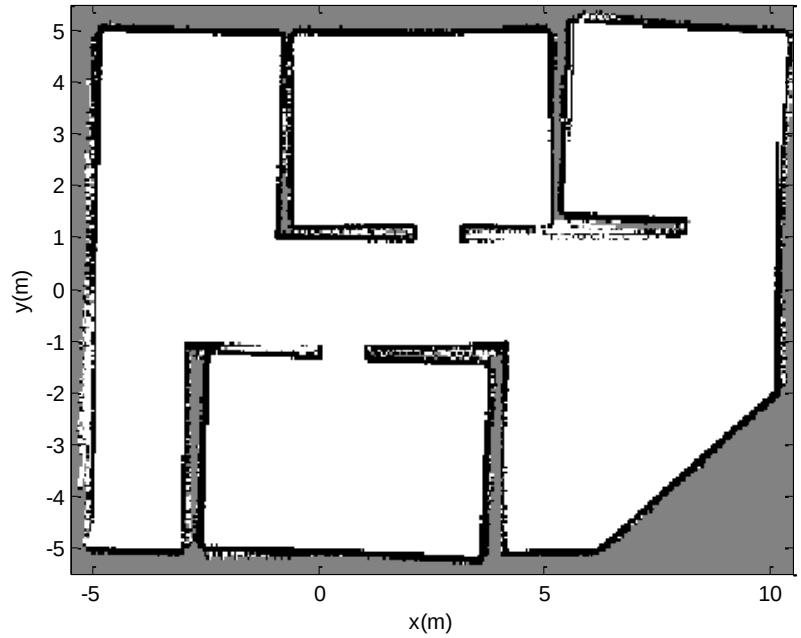


Figura 5.15. Mapeo usando el método Monte Carlo Rao-Blackwellized (M=30) - Tercera prueba (Fuente: Elaboración propia)

c) Pruebas cuando M = 50

Primera prueba

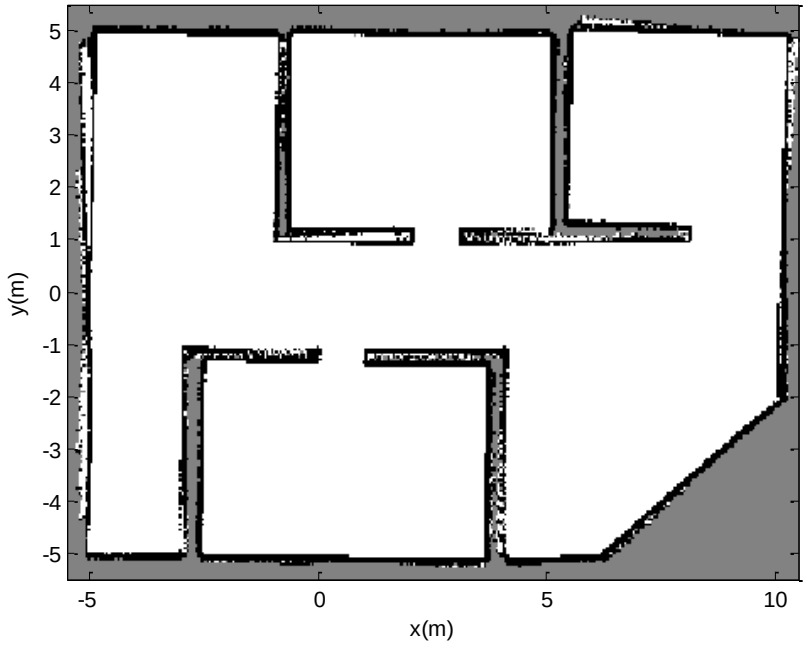


Figura 5.16. Mapeo usando el método Monte Carlo Rao-Blackwellized (M=50) - Primera prueba (Fuente: Elaboración propia)

Segunda prueba

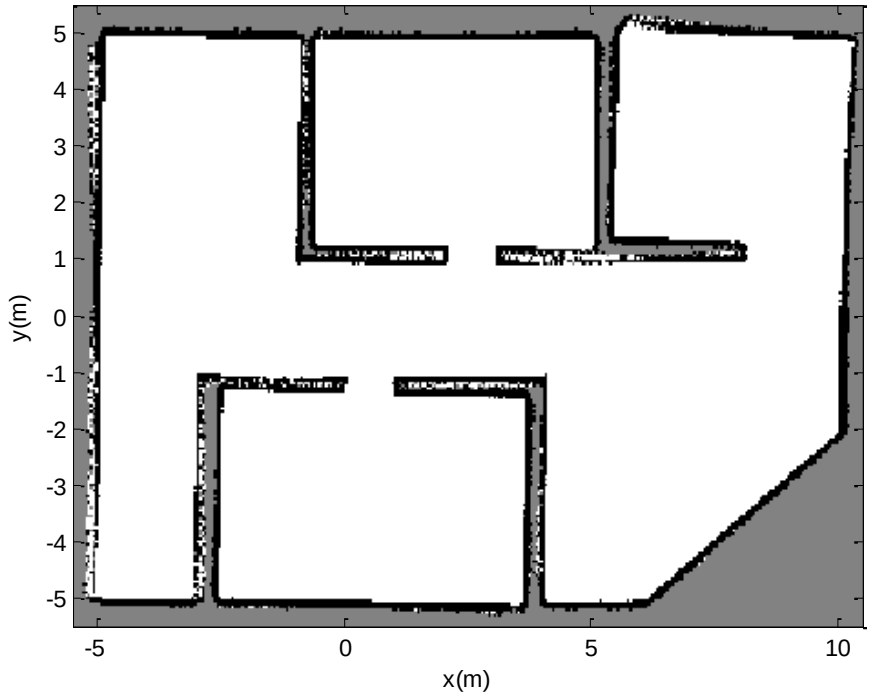


Figura 5.17. Mapeo usando el método Monte Carlo Rao-Blackwellized cuando M=50 - Segunda prueba (Fuente: Elaboración propia)

Tercera prueba

Figura 5.18. Mapeo usando el método Monte Carlo Rao-Blackwellized (M=50) - Tercera prueba (Fuente: Elaboración propia)

5.3.4 Conclusiones

Tal como puede deducirse de los experimentos anteriores, los mapas adquiridos usando el filtro Monte Carlo Rao-Blackwellized estándar no son de buena calidad ya que no reflejan con mucha exactitud el mapa del entorno real. Sin bien en algunos casos los mapas son parecidos al mapa del entorno real, en ciertas regiones estos presentan inconsistencias que los hacen inadecuados en su uso como mapas de ocupación.

Conforme se incrementó el número de muestras, los mapas obtenidos muestran ciertas mejoras. En teoría, se podría hacer más experimentos usando cada vez más cantidad de muestras, sin embargo, la dificultad de este enfoque, es que el tiempo de ejecución se hace demasiado extenso (en los experimentos usando M=50 muestras, el tiempo de ejecución del algoritmo fue de aproximadamente 3 horas y media), y además se llega a un punto en que la capacidad de la memoria de la computadora se agota.

El principal inconveniente del algoritmo Monte Carlo Rao-Blackwellized es la función de distribución propuesta. Tal como se comentó en el Capítulo 1, en los algoritmos basados en muestras, la calidad de la estimación depende en gran parte de la calidad de la distribución propuesta. En al caso del algoritmo Monte Carlo Rao-Blackwellized estándar se usó como función propuesta la función de movimiento $p(x_t \mid x_{t-1}, u_t)$. El principal problema de esta distribución es su gran nivel de incertidumbre (Figura 5.19), lo que hace que se requiera una gran cantidad de muestras con el fin de cubrir adecuadamente todas las regiones de esta distribución.

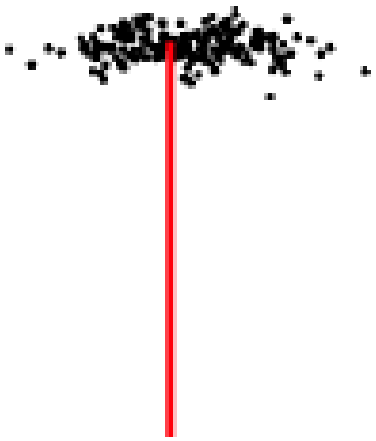


Figura 5.19. Muestras de la función de densidad de probabilidad de movimiento (Fuente: Grisetti, et. [25])

5.4 Mapeo usando Métodos Monte Carlo Rao-Blackwellized Mejorados

Tal como se pudo observar en la sección anterior, la aplicación del método de mapeo Monte Carlo Rao-Blackwellized estándar no provee buenos resultados en la construcción de mapas de alta calidad de los entornos de navegación del robot. Con el fin de remediar estos problemas en esta sección se mostraran una serie de mejoras, descritas en [25], con el fin de tener un algoritmo que sea capaz de construir mapas de navegación de alta calidad que puedan usarse para realizar tareas de navegación autónoma.

5.4.1 Mejoras Propuestas

a) Mejoramiento de la distribución propuesta. Tal como se mencionó en el Capítulo 1, la calidad de los algoritmos del tipo Monte Carlo depende en gran medida de la distribución propuesta $q(x_t \mid x_{t-1}^i, z_t, u_t)$. Una mejor distribución propuesta resulta en un mejor desempeño del filtro de estimación. En el caso del algoritmo de mapeo Monte Carlo Rao-Blackwellized estándar, la distribución propuesta fue la distribución de transición de movimiento $p(x_t \mid u_t, x_{t-1})$.

La desventaja de usar esta distribución es que es no es óptima especialmente cuando se usa sensores de alta precisión, como es el caso de los sensores laser. Este problema se puede ver claramente en la Figura 5.20 donde se puede notar que la función de densidad de probabilidad de los sensores es mucho más precisa.

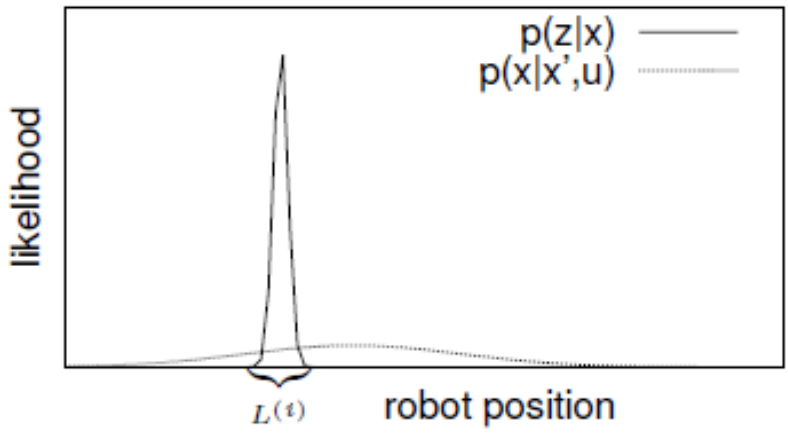


Figura 5.20. Distribución de las mediciones vs. Distribución de movimiento (Fuente: Grisetti, et. [25])

Con el fin de superar este problema es posible considerar una nueva distribución propuesta que tenga en cuenta las mediciones de los sensores. De esta manera la distribución propuesta estaría mucho más concentrada en las regiones de alta probabilidad según las mediciones del sensor. Tal como se muestra en [21] la distribución:

$$p(x_t \mid x_{t-1}, u_t, m_{t-1}, z_t) = \frac{p(z_t \mid x_t, m_{t-1})\, p(x_t \mid x_{t-1}, u_t)}{p(z_t \mid x_{t-1}, m_{t-1}, u_t)} \tag{5.7}$$

es la distribución optima con respecto a la varianza de los pesos de las muestras. De esta manera, al tomar en cuenta la medición del sensor, la varianza de estimación se reduce de manera considerable, lo que mejora notablemente la calidad de la distribución propuesta. La nueva distribución propuesta (Figura 5.21) es mucho más precisa que la distribución propuesta estándar (Figura 5.20)

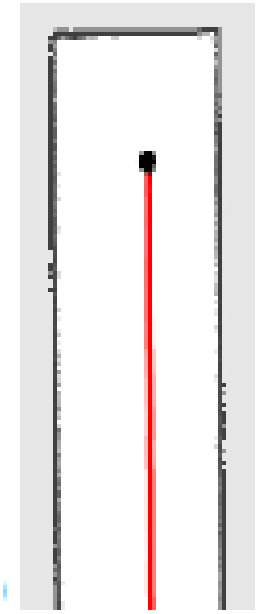


Figura 5.21. Distribución propuesta que considera las mediciones (Fuente: Grisetti, et. [25])

Al usar la distribución propuesta en la ecuación (5.7), se tiene que la ecuación de actualización de los pesos toma la forma:

$$w_t^{(i)} = w_{t-1}^{(i-1)} \cdot \frac{p(z_t \mid m_{t-1}^{(i)}, x_t^{(i)})\, p(x_t^{(i)} \mid x_{t-1}^{(i)}, u_t)}{p(x_t \mid x_{t-1}^{(i)}, m_{t-1}^{(i)}, z_t, u_t)} \tag{5.8}$$

Reemplazando la expresión en (5.7) en (5.8) resulta en la siguiente ecuación de actualización de pesos:

$$w_t^{(i)} = w_{t-1}^{(i-1)} \cdot \int p(z_t \mid x') p(x' \mid x_{t-1}^{(i)}, u_t) dx' \tag{5.9}$$

Esta ecuación contiene una integral que tiene como argumento el producto de la distribución de las medidas con la distribución de movimiento.

b) Aproximación eficiente de la distribución propuesta. Con el fin de manejar de manera eficiente el proceso de generación de muestras y actualización de pesos es conveniente hacer una aproximación de la distribución propuesta $p(x_t \mid x_{t-1}, u_t, m_{t-1}, z_t)$. La idea principal es aproximar esta distribución usando una distribución gausiana alrededor del punto máximo de la función de probabilidad de medidas. Para hallar este punto máximo es necesario hacer una búsqueda de la forma:

$$\hat{x}_t^{(i)} = \arg\max_x p(x \mid z_t, m_{t-1}^{(i)}, x_t'^{(i)}) \tag{5.10}$$

Donde $x_t'^{(i)}$ es el punto de partida propuesto que puede estar dado por una muestra de la función de probabilidad de movimiento. Para hacer esta búsqueda se puede hacer uso de técnicas de optimización (como el algoritmo de gradiente ascendente). Entonces, dado el punto máximo de probabilidad $\hat{x}_t^{(i)}$ y el área que lo contiene, lo siguiente es realizar un muestreo en esa área con el fin de hallar la media y la covarianza de la distribución gaussiana. Los parámetros $u_t^{(i)}$ y $\sum_t^{(i)}$ se hallan usando:

$$u_t^{(i)} = \frac{1}{\eta^{(i)}} \cdot \sum_{j=1}^{K} x_j \cdot p(z_t \mid m_{t-1}^{(i)}, x_j) \cdot p(x_j \mid x_{t-1}^{(i)}, u_t) \tag{5.11}$$

$$\sum_t^{(i)} = \frac{1}{\eta^{(i)}} \cdot \sum_{j=1}^{K} (x_j - \mu_t^{(i)}) \cdot (x_j - \mu_t^{(i)})^T \cdot p(z_t \mid m_{t-1}^{(i)}, x_j) \cdot p(x_j \mid x_{t-1}^{(i)}, u_t) \tag{5.12}$$

$$\eta^{(i)} = \sum_{j=1}^{K} p(z_t \mid m_{t-1}^{(i)}, x_j) \cdot p(x_j \mid x_{t-1}^{(i)}, u_t) \tag{5.13}$$

Como puede notarse, en el cálculo de los parámetros de la distribución gaussiana se toma en cuenta las distribuciones de movimiento y de mediciones. Entonces al hacer una aproximación gausiana de la distribución propuesta, el proceso de generación de muestras se puede hacer de manera eficiente debido a que existen métodos numéricos bien fundamentados que nos permiten tomar muestras de distribuciones gausianas.

Respecto a la ecuación de actualización de pesos (5.9), esta se puede implementar eficientemente usando la siguiente aproximación:

$$\begin{aligned} w_t^{(i)} &\simeq w_{t-1}^{(i-1)} \cdot \sum_{j=1}^{K} p(z_t \mid m_{t-1}^{(i)}, x_j)\, p(x_j \mid x_{t-1}^{(i)}, u_t) \\ &\simeq w_{t-1}^{(i-1)} \cdot n^{(i)} \end{aligned} \tag{5.14}$$

Donde $n^{(i)}$ es el valor previamente calculado en la ecuación (5.13). De esta manera la actualización de pesos es fácil de implementar, y desde el punto de vista computacional nos proporciona un gran ahorro de tiempos de ejecución.

c) Reducción de la frecuencia de muestreo. Al igual que en el caso del problema de localización otra mejora que se puede hacer al algoritmo es reducir la frecuencia de re muestreo. La idea es integrar las mediciones en diversos instantes de tiempo usando la siguiente formula de actualización de pesos

$$w_t^{(m)} = \begin{cases} 1 & si\ se\ ha\ hecho\ re\ muestreo \\ w_{t-1}^{(m)} \cdot n^{(m)} & si\ no\ se\ ha\ hecho\ re\ muestreo \end{cases} \tag{5.15}$$

El paso de re muestreo se realiza cada vez que el tamaño efectivo de muestras N_{eff} es menor de $N/2$ donde N es el número de muestras. De esta manera, se consigue una reducción considerable del problema degeneración de muestras. Es decir se garantiza una buena diversidad de muestras, y se disminuyen los problemas de las muestras repetidas. Esto punto es importante, ya que de esta manera se puede trabajar con diferentes hipótesis sobre el estado del robot.

5.4.2 Algoritmo de Mapeo

Luego de adaptar todas las mejoras propuestas descritas en la sección anterior al algoritmo de mapeo Monte Carlo Rao-Blackwellized estándar, en la Tabla 5.3 se muestra el algoritmo de mapeo Monte Carlo Rao-Blackwellized mejorado. Este algoritmo más conocido como '*gmapping*' constituye el estado del arte en el problema de mapeo, y es usado por los robots y móviles más avanzados a nivel mundial. La contribución de la presente tesis consiste en su implementación para la tarea de navegación autónoma de nuestro robot móvil.

En este algoritmo se empieza usando el algoritmo de optimización para buscar el punto máximo $x_t^{'(i)}$. En las líneas 5 a 7, se hace un muestreo alrededor de la región del punto máximo. En las líneas 8 a 19 se hace el cálculo de la distribución propuesta usando la aproximación gaussiana, para esto se hace el cálculo de la media y la covarianza. En la línea 21, se toma una muestra de la distribución gaussiana; esta muestra representa la nueva hipótesis sobre la configuración de una muestra dada. En la línea 23 se hace la actualización de pesos de las muestras. En la línea 25 se hace la actualización del mapa de ocupación usando las medidas de los sensores y la nueva hipótesis sobre la configuración. Finalmente entre las líneas 28 a 31, se hace el paso de re muestreo.

5.4.3 Resultados

Luego de haber implementado el algoritmo de mapeo Monte Carlo Rao-Blackwellized mejorado descrito en la Tabla 5.3, a continuación se muestran los resultados obtenidos usando el mismo entorno de navegación que se usó en las pruebas del algoritmo Monte Carlo Rao-Blackwellized standard. Esto con el fin de comparar los resultados obtenidos usando ambos algoritmos.

Tal como se verá en las Figuras 5.22-5.31 la cantidad de muestras necesarias para obtener mapas de alta calidad es mucho menor que las requeridas en las pruebas anteriores. Este hecho se refleja en un tiempo de mapeo mucho más rápido, y en el menor uso de recursos computacionales. Por lo que este algoritmo de mapeo puede usarse cuando se requiera que un robot haga mapeo en tiempo real.

Tabla. 5.3. Algoritmo de mapeo usando el método Monte Carlo Rao-Blackwellized mejorado (Fuente: Grisetti, et. [25]))

$Algoritmo\ \ Mapeo\_MC\_Rao\_Blackwellized\_mejorado(\{x_{t-1}^{[k]}, m_{t-1}^{[k]}\}, u_t, z_t)$

$L01:$ *for* $k = 1:M$

$L02:$ $//Ejecutar\ algoritmo\ Scan-matching$

$L03:$ $\hat{x}_t^{(i)} = scan\_matcher(x_{t-1}^{(i)}, z_t, m_{t-1}^{(i)})$

$L04:$ $//Aproximacion\ gausiana\ de\ la\ distribucion\ propuesta$

$L05:$ *for* $j = 1:K$

$L06:$ $x_k \sim \{x_j \mid |x_j - \hat{x}_t^{(i)}| < \Delta\}$

$L07:$ *end*

$L08:$ $u_t^{(i)} = (0,0,0)^T$

$L09:$ $n^{(i)} = 0$

$L10:$ *for* $x_j \in \{x_1, ..., x_K\}$

$L11:$ $u_t^{(i)} = u_t^{(i)} + x_j \cdot p(z_t \mid m_{t-1}^{(i)}, x_j) \cdot p(x_j \mid x_{t-1}^{(i)}, u_t)$

$L12:$ $n^{(i)} = n^{(i)} + p(z_t \mid m_{t-1}^{(i)}, x_j) \cdot p(x_j \mid x_{t-1}^{(i)}, u_t)$

$L13:$ *end*

$L14:$ $u_t^{(i)} = u_t^{(i)} / n^{(i)}$

$L15:$ $\sum_t^{(i)} = 0$

$L16:$ *for* $x_j \in \{x_1, ..., x_K\}$

$L17:$ $\sum_t^{(i)} = \sum_t^{(i)} + (x_j - \mu_t^{(i)}) \cdot (x_j - \mu_t^{(i)})^T \cdot p(z_t \mid m_{t-1}^{(i)}, x_j) \cdot p(x_j \mid x_{t-1}^{(i)}, u_t)$

$L18:$ *end*

$L19:$ $\sum_t^{(i)} = \sum_t^{(i)} / n^{(i)}$

$L20:$ $//Muestreo$

$L21:$ $x_t^{(i)} \sim N(u_t^{(i)}, \sum_t^{(i)})$

$L22:$ $//Actualizacion\ de\ pesos$

$L23:$ $w_t^{(i)} = w_{t-1}^{(i)} \cdot n^{(i)}$

$L24:$ $//Actualizacion\ del\ mapa$

$L25:$ $m_t^{(i)} = update\_map(m_t^{(i)}, x_t^{(i)}, z_t)$

$L26:$ *end*

$L27:$ $//Ejecutar\ algoritmo\ de\ remuestreo$

$L28:$ $N_{eff} = 1 / \sum_{i=1}^{M} \left(w_t^{(i)}\right)^2$

$L29:$ *if* $N_{eff} < M/2$

$L30:$ $w_t^{(i)} = 1$

$L31:$ *end*

return $\{x_t^{[k]}, m_t^{[k]}\}$

a) Pruebas cuando M = 5

Primera prueba

.

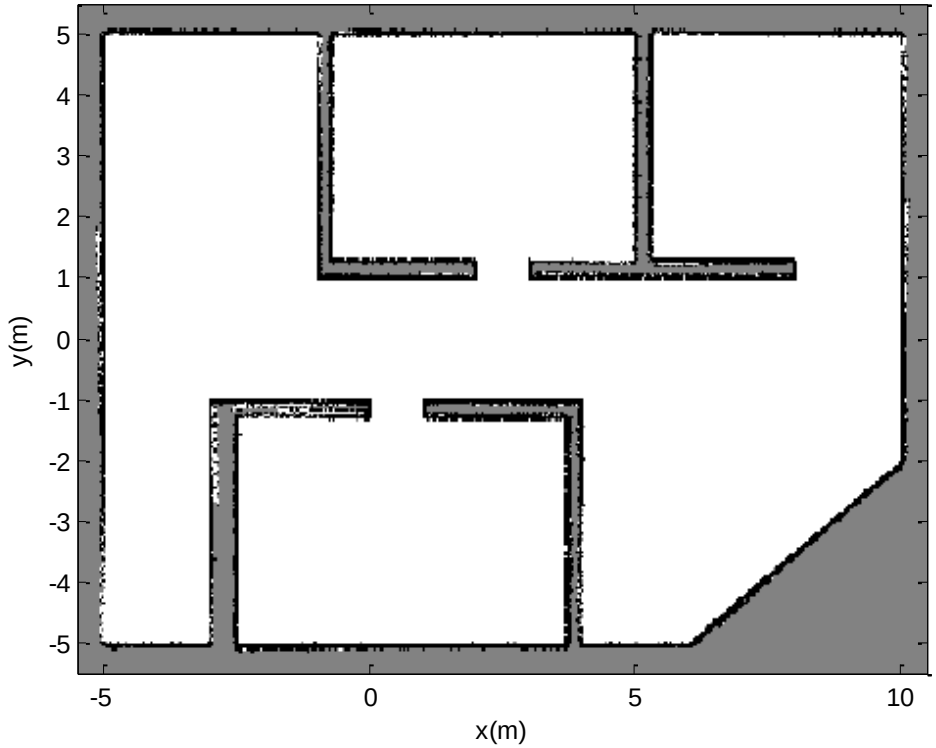


Figura 5.22. Mapeo usando el método Monte Carlo Rao-Blackwellized mejorado (M=5) - Primera prueba (Fuente: Elaboración propia)

A continuación se muestran los mapas de otras muestras que tienen menor probabilidad. En las pruebas posteriores solo se mostraran los mapas de las muestras de mayor probabilidad.

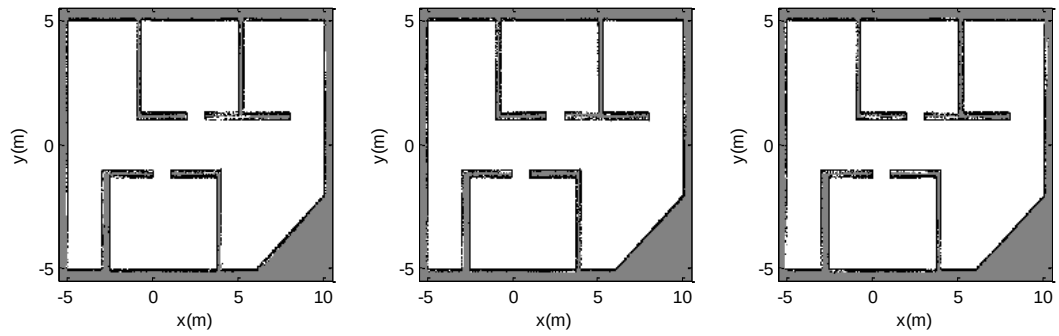


Figura 5.23. Mapas de ocupación de otras muestras cuando M=5 - Primera prueba (Fuente: Elaboración propia)

Segunda prueba

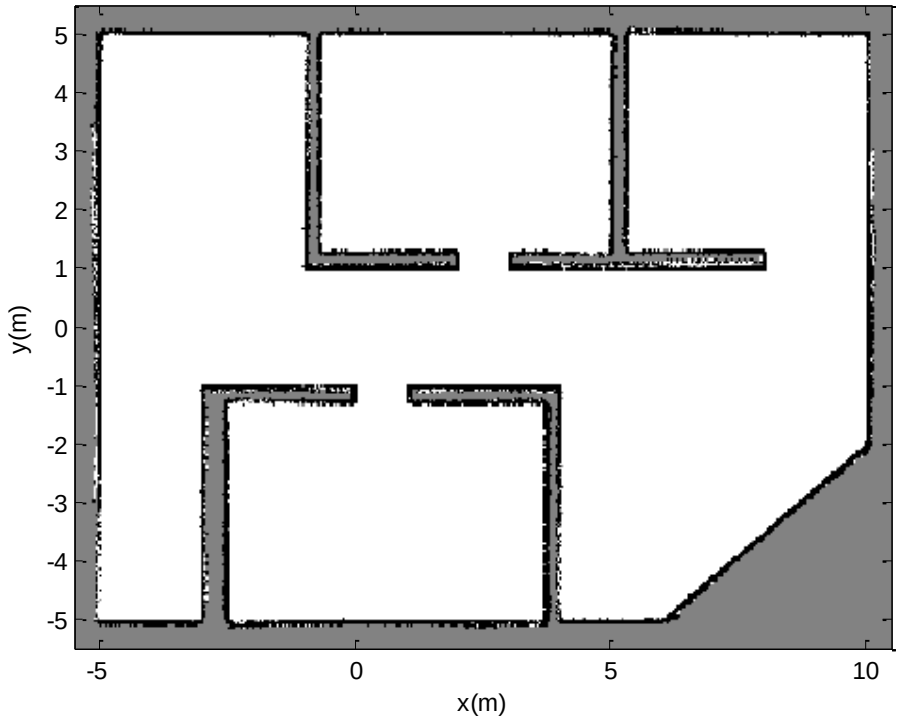


Figura 5.24. Mapeo usando el método Monte Carlo Rao-Blackwellized mejorado (M=5) - Segunda prueba (Fuente: Elaboración propia)

Tercera prueba

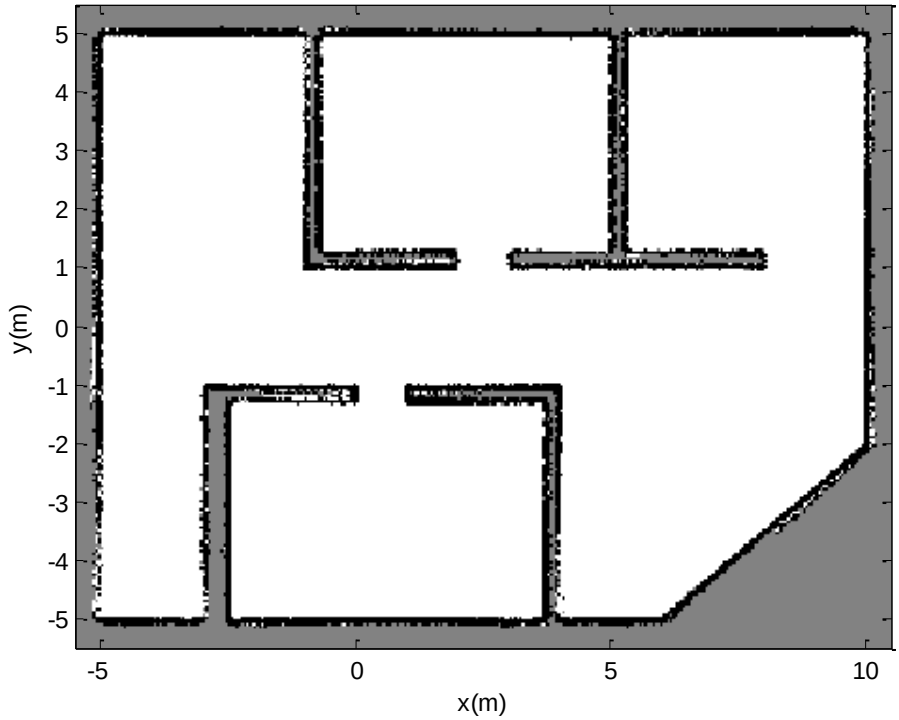


Figura 5.25. Mapeo usando el método Monte Carlo Rao-Blackwellized mejorado (M=5) - Tercera prueba (Fuente: Elaboración propia)

b) Pruebas cuando M = 10

Primera prueba

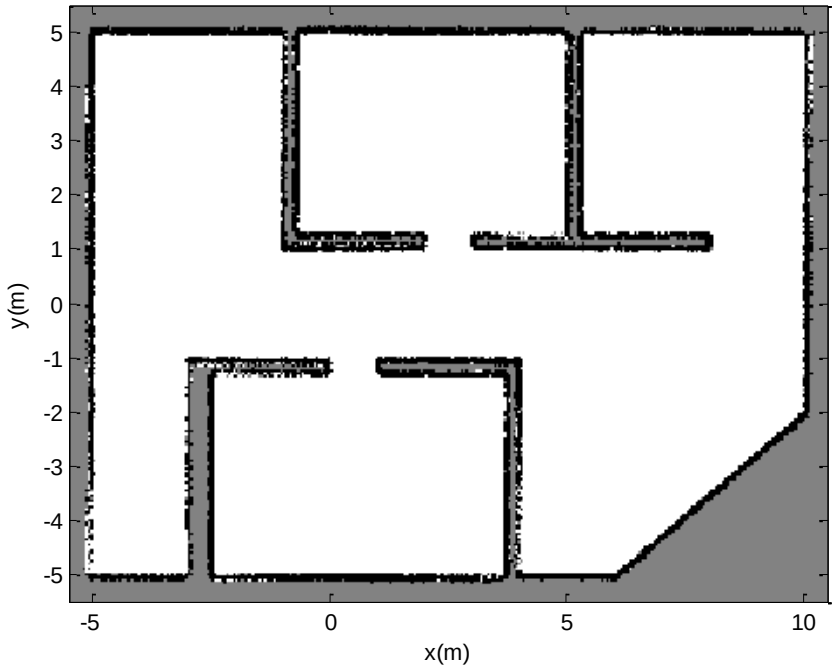


Figura 5.26. Mapeo usando el método Monte Carlo Rao-Blackwellized mejorado (M=10) - Primera prueba (Fuente: Elaboración propia)

Segunda prueba

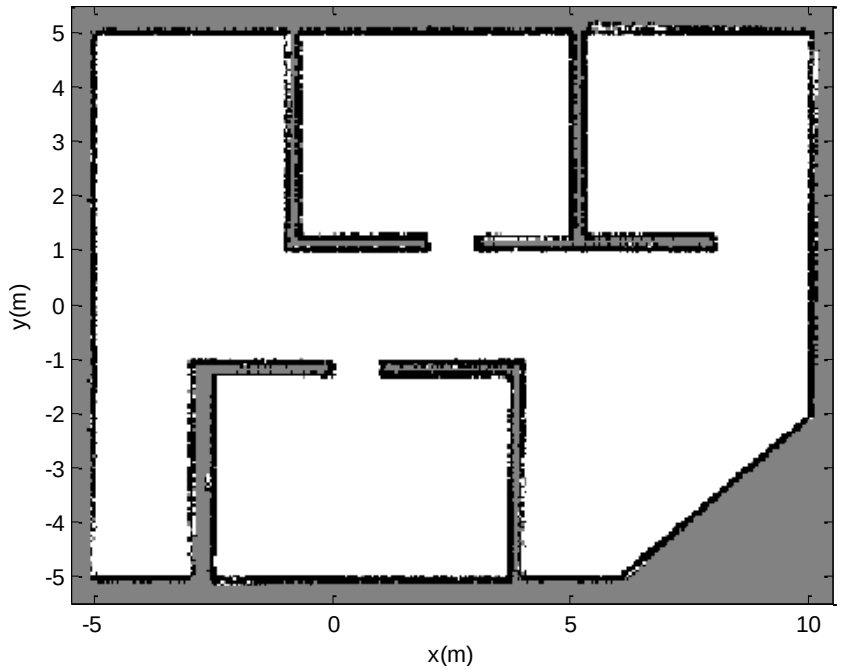


Figura 5.27. Mapeo usando el método Monte Carlo Rao-Blackwellized mejorado (M=10) - Segunda prueba (Fuente: Elaboración propia)

Tercera prueba

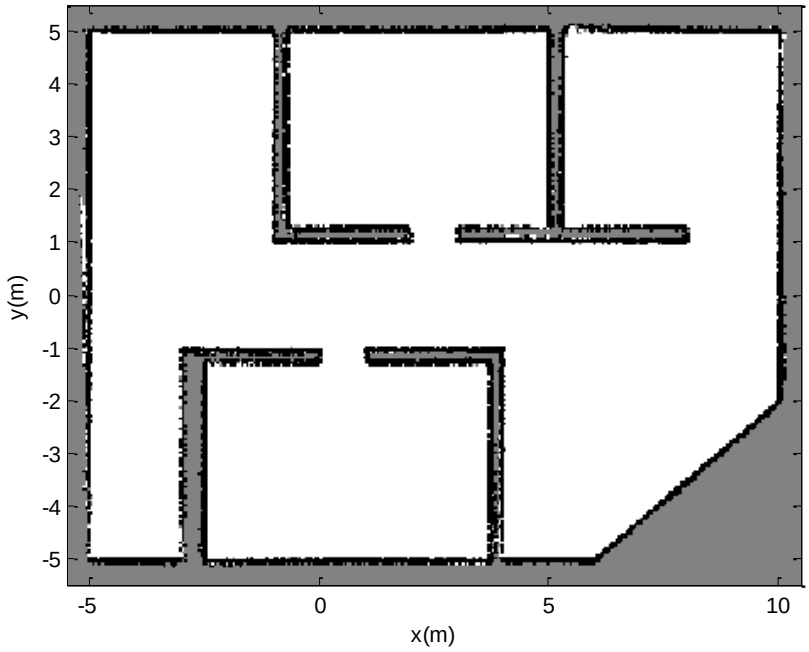


Figura 5.28. Mapeo usando el método Monte Carlo Rao-Blackwellized mejorado (M=10) - Tercera prueba (Fuente: Elaboración propia)

c) Pruebas cuando M = 15

Primera prueba

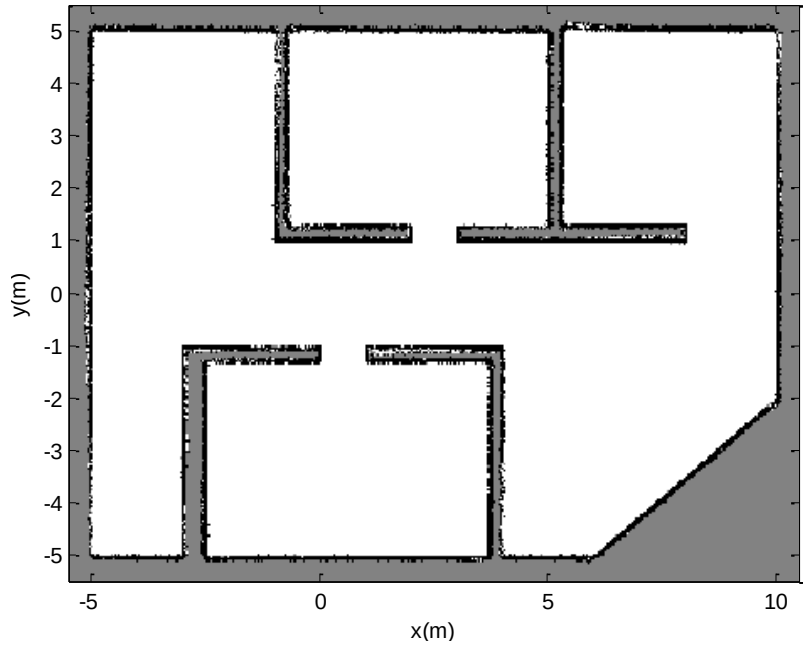


Figura 5.29. Mapeo usando el método Monte Carlo Rao-Blackwellized mejorado (M=15) - Primera prueba (Fuente: Elaboración propia)

Segunda prueba

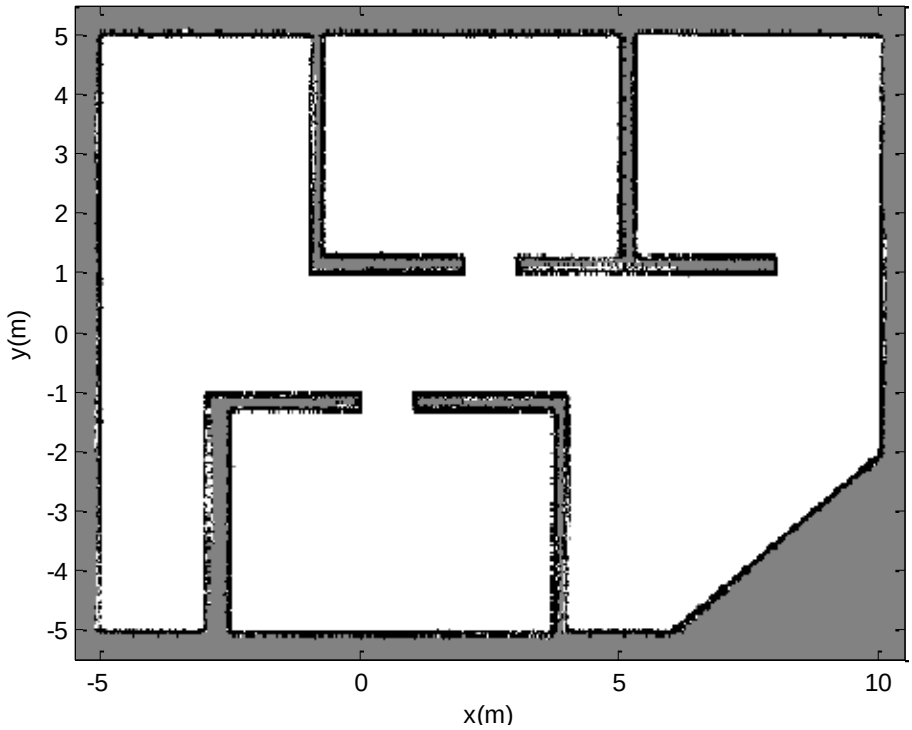


Figura 5.30. Mapeo usando el método Monte Carlo Rao-Blackwellized mejorado (M=15) - Segunda prueba (Fuente: Elaboración propia)

Tercera prueba

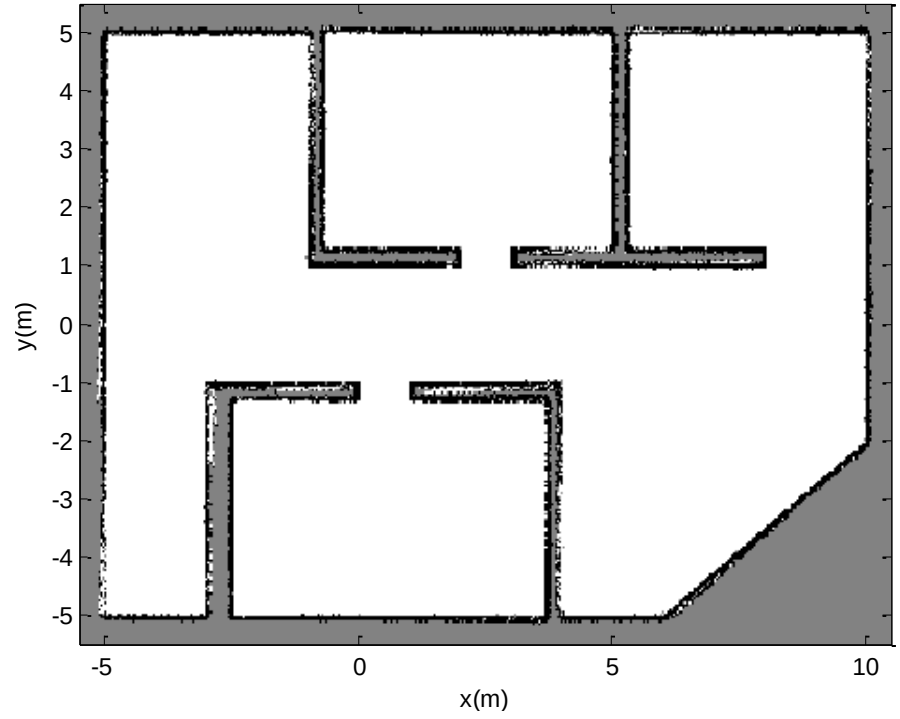


Figura 5.31. Mapeo usando el método Monte Carlo Rao-Blackwellized mejorado (M=15) - Tercera prueba (Fuente: Elaboración propia)

5.4.4 Conclusiones

Como puede desprenderse de los resultados obtenidos, el algoritmo de mapeo Monte Carlo Rao-Blackwellized mejorado es capaz de construir mapas de navegación de alta calidad usando un número reducido de muestras. Estos mapas son consistentes con el mapa real del entorno de navegación, y por tal el robot puede usar estos mapas en su tarea de navegación autónoma.

Estas mejoras se deben en gran parte al uso de una mejor distribución propuesta, la cual al tomar en cuenta no solo la señal de control u sino también las mediciones de los sensores z, permite tener una distribución con una menor varianza y mucho más centrada alrededor de la configuración real del robot. Esto a su vez permite el uso de una menor cantidad de muestras, ya que el área de probabilidad es mucho más reducida.

Finalmente, cabe notar que el uso de un menor número muestras en la construcción de mapas de alta calidad, permite que el robot pueda construir mapas en tiempo real, es decir mientras esta en movimiento en cierto ambiente de navegación.

CAPÍTULO VI
PLANIFICACION Y CONTROL DE MOVIMIENTO

Hasta este punto se han desarrollado los componentes básicos que son necesarios para que un robot móvil pueda navegar en un ambiente dado. Es decir, dado un mapa del entorno de navegación, el robot tiene la capacidad de conocer en cada instante de tiempo su configuración respecto a este mapa; y si el robot no tiene el mapa lo puede construir de manera autónoma. Ahora prestaremos atención al nivel de cognición, que representa el componente de toma de decisiones cuyo fin es lograr que el robot logre sus metas, y de esta manera pueda realizar tareas asignadas.

El componente de cognición está relacionado a la competencia de planificación del movimiento cuya tarea se puede definir como: “dado un conocimiento total o parcial de su entorno y una meta, planificar la serie de acciones con el fin de llegar a la meta de manera eficiente y confiable”. En este capítulo se desarrollara el componente de planificación de movimiento del robot móvil usando unos de los algoritmos más usados en la Inteligencia Artificial – el algoritmo A*. Además se desarrollara el componente de control de movimiento que hará posible que el robot llegue a la meta por el camino planificado. Para implementar este último componente se hará uso de los conocidos controladores PID.

6.1 Representación del entorno

El primer paso en la mayor parte de los sistemas de planificación de trayectorias es la transformación del entorno de navegación, que es un espacio continuo, en un modelo “mapa” discreto en donde se puedan aplicar algoritmos de planificación [18]. Para esto, existen dos enfoques principales: la descomposición usando celdas, y la esqueletización. Estas representaciones reducen el problema de planeamiento en espacios continuos a un problema de búsqueda en espacios discretos usando grafos.

Para el desarrollo del componente de navegación de nuestro robot móvil se ha escogido la representación del entorno usando celdas debido a las ventajas que esta representación presentan sobre los métodos basados en la esqueletización [18]. Esta representación del entorno, que ya fue vista en el Capítulo 3 del presente trabajo, es usada por la mayoría de algoritmos de planeamiento que hoy representan el estado del arte en la robótica móvil.

6.2 Búsqueda usando Grafos

Las técnicas de búsqueda usando grafos, desarrolladas en el campo de la Inteligencia Artificial, se han usado ampliamente en la comunidad de la robótica para la tarea de planificación de trayectorias debido a su fuerte sustento matemático y su capacidad de acomodarse a entornos dinámicos [1], [26], [27]. En la mayoría de estos métodos se distinguen dos pasos principales: la construcción del grafo, donde los nodos de búsqueda se insertan al grafo y se conectan vía arcos; y la búsqueda, donde se busca una solución óptima en el espacio de nodos. A continuación se describirán estos pasos de manera resumida en el caso de la planificación de trayectoria de un robot móvil.

a) Construcción del grafo: En el caso del robot móvil, los nodos del grafo representan las posiciones del robot; y los arcos representan las conexiones entre estos. Dependiendo del algoritmo de búsqueda a usar, los nodos son añadidos al grafo en una forma determinada. Usualmente en esta construcción se empieza por el nodo que contiene la posición de partida, y usando todas las acciones disponibles se van generando los nuevos nodos del grafo.

Para la construcción del grafo en el caso del planeamiento de un robot móvil usualmente se hacen las siguientes asunciones respecto del robot y su entorno.

- **Omni-direccionalidad:** El robot es capaz de moverse desde su posición actual a cualquier posición adyacente (Figura 6.1.)
- **Movimiento determinístico:** Se asume que el movimiento del robot es preciso. Es decir si su intención es ir a una posición adyacente, el robot llegara a esta posición.

- **Única posición deseada:** El robot tienen como objetivo llegar a solo una posición deseada. Si se desea llegar a otra posición, se debe ejecutar nuevamente el componente de planificación con este nuevo objetivo.
- **Representación binaria del mapa:** Los espacios libres tienen un valor de 0, y los espacios ocupados tienen un valor de 1.
- **Costo de navegación:** El costo de navegar por una celda está dado por un número positivo.

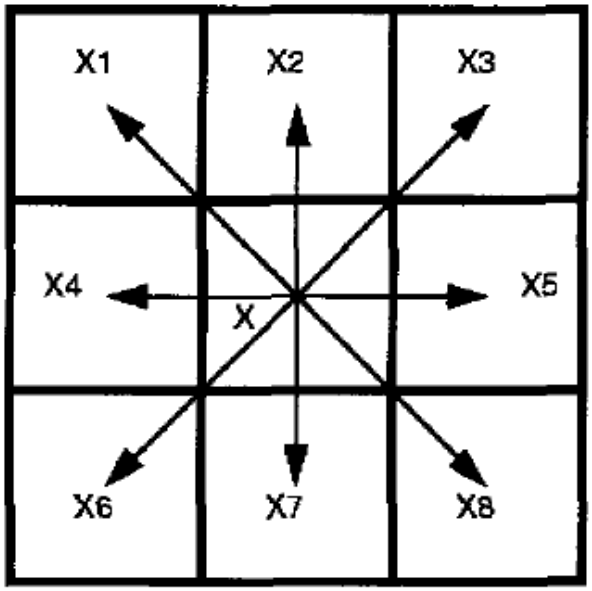


Figura 6.1. Movimientos posibles del robot (Fuente: http://www.mobilerobots.ethz.ch/)

b) Búsqueda en el grafo. El segundo paso consiste en buscar en el grafo construido el camino “optimo” entre la posición actual del robot y la posición deseada. El término óptimo en este caso se refiere a la minimización o maximización de cierto criterio de optimización, por ejemplo el camino mas corto. Diversos criterios de búsqueda dan lugar a diferentes algoritmos de búsqueda [1], [26], [27].

c) Evaluación de los algoritmos de búsqueda. Con el fin de evaluar el desempeño de los diversos algoritmos de búsqueda comúnmente se usan los siguientes criterios de evaluación:

- **Completo:** Se dice que un algoritmo es completo si tiene la garantía de encontrar una solución si es que esta existe.

- **Optimo:** Si dice que un algoritmo es óptimo si es capaz de encontrar la trayectoria óptima “más corta” dentro de un conjunto posible de trayectorias.

Una vez presentados los conceptos básicos de la representación del entorno y los métodos de búsqueda basados en grafos, a continuación se presentara uno de los algoritmos más populares de planificación de trayectorias, el algoritmo A*. Se ha escogido este algoritmo no solo porque es óptimo y completo, sino por su capacidad de aplicación práctica en el caso de la planificación de trayectorias en tiempo real en entornos dinámicos. Tal como se verá en la siguiente sección este algoritmo puede generar trayectorias “optimas” de manera muy eficiente (esta eficiencia se consigue además al aplicar técnicas de programación optimizadas a la hora de implementar el algoritmo).

6.3 Planeamiento usando el Algoritmo A*

El algoritmo A* es un método de búsqueda del tipo informado ampliamente usado y estudiado en el campo de la Inteligencia Artificial [9]. Esto es debido a su gran eficiencia, y al hecho que es del tipo óptimo y completo. Es decir el algoritmo A* es capaz de encontrar una solución si esta exista existe, y esta solución es la más óptima de entre todas las posibles soluciones. En esta sección se describirá de manera resumida los conceptos básicos de este algoritmo, y se mostraran los resultados de la implementación del mismo.

6.3.1 Definiciones Básicas

En el proceso de búsqueda usando grafos por lo general se trabaja con una serie de conjuntos con el fin de guardar información sobre el proceso de búsqueda. En el caso del algoritmo A* se trabaja con dos conjuntos:

a) **Conjunto abierto (OPEN).** Es el conjunto que contiene los nodos que están siendo analizados en determinado momento. Estos nodos son los que están listos para ser expandidos con el fin de generar nuevos nodos. Este conjunto también es llamado la “frontera” por qué divide el espacio explorado del no explorado

b) Conjunto cerrado (CLOSED). Es el conjunto que contiene a los nodos que ya han sido analizados. Este conjunto tiene la finalidad de eliminar las redundancias en el proceso de generación de nodos (elimina los caminos redundantes).

Por otro lado, el algoritmo A* al ser del tipo informado hace uso de heurísticas o conocimientos específicos acerca de un problema con el fin de ayudar al algoritmo en el proceso de búsqueda. De esta manera, el proceso de búsqueda es mucho más rápido, y los requerimientos de memoria para guardar los conjuntos abierto y cerrado disminuyen considerablemente. Este conocimiento heurístico, está definido por una función heurística dada por:

$h(n)$ = *Costo estimado de la ruta más corta desde el estado que corresponde al nodo "n" a la posición deseada.*

En el caso de la planificación de trayectorias de los robots móviles las heurísticas se refieren a las distancias más cortas desde una posición dada hasta la meta. Dependiendo de la métrica que se use para definir una distancia, se tienen diferentes tipos de heurísticas. En la práctica las heurísticas más usadas son:

$$h(n) = abs(n(x) - goal(x)) + abs(n(y) - goal(y)) \quad (6.1)$$

$$h(n) = \sqrt{(n(x) - goal(x))^2 + (n(y) - goal(y))^2} \quad (6.2)$$

Donde $n(x)$, $n(y)$ son las coordenadas "x" e "y" del nodo n que contiene la posición actual, y $goal(x)$, $goal(y)$ son las coordenadas "x" e "y" de la meta. La heurística (6.1) es comúnmente llamada la heurística de 'Manhattan' (debido a que hace uso de la métrica de Manhattan), y la heurística (6.2) es comúnmente llamada la heurística 'Euclidiana'.

El procedimiento de búsqueda del algoritmo A* consiste en la expansión de los nodos que tengan el menor valor del costo estimado total " f " dado por

$$f(n) = g(n) + h(n) \quad (6.3)$$

Donde $g(n)$ es el costo del camino desde la posición inicial hasta el nodo "n". Entonces si estamos buscando la solución menos costosa, el algoritmo A* siempre seleccionara los nodos con menor valor " f " en el conjunto abierto; de esta manera, la solución encontrada por el algoritmo siempre será la más óptima [1], [27].

6.3.2 El Algoritmo A*

En la tabla 6.1 se describe en pseudocódigo el algoritmo A*. Las entradas del algoritmo son la posición inicial del robot, la meta y la heurística a usar en el proceso de planificación de trayectoria del robot móvil.

Tabla 6.1. Algoritmo de búsqueda A* (Fuente: Norvig, et. [1])

```
Funcion Astar (inicio, meta, heuristica)
L1   Nodo ← Posición inicial
L2   OPEN ← Establecer "Nodo" como el único elemento
L3   CLOSED ← Establecer como conjunto vacío
L4   loop do
L5      if EMPTY(OPEN)
L6          Return "falla"
L7      Nodo ← POP(OPEN)  /* Escogemos el nodo con menor costo "f"*/
L8      if Nodo == "Meta"
L9          Return "solución encontrada"
L10     CLOSED ← Ingresamos "Nodo"
L11     for action in ACTIONS(Nodo)
L12         child ← CHILD-NODE(Nodo, action)
L13         if child is not in CLOSED or OPEN
L14             OPEN ← INSERT(child)
L15         else if child is in OPEN con mayor costo
L16             reemplazar el anterior nodo "child" con el nuevo "child"
```

En las líneas L1 a L3 se inicializa en conjunto abierto OPEN con un solo elemento que corresponde a la posición inicial, y el conjunto cerrado CLOSED que se define como vacío. En la línea L4 se declara un lazo infinito, mediante el cual buscamos una solución si esta existe, y retornamos "falla" si esta no existe. En las líneas L5 y L6 se revisa el conjunto abierto OPEN, si este está vacío se retorna "falla" y se finaliza el algoritmo. En la línea L7 se escoge el nodo con menor costo "f" del conjunto abierto eliminándolo de este conjunto. En las líneas L8 y L9, se revisa si este nodo corresponde al estado deseado, indicando que se ha encontrado una solución en caso de ser así. En la línea 10 se establece que el nodo bajo análisis pertenece al conjunto cerrado CLOSED, indicando de

esta manera que el nodo ya ha sido analizado. En las líneas siguientes, se expande el nodo seleccionado usando todas las posibles acciones (Figura 6.1), se añaden los nodos generados en la lista abierta OPEN (L13 y L14), y si se ha descubierto un camino con menor costo (L15 y L16) se escoge este como el nuevo camino.

6.3.3 Resultados

A continuación se muestran los resultados de la implementación del algoritmo A* usando el mapa de navegación usado en el capítulo de localización. En este mapa las celdas tienen una resolución de 4 cm, lo que resulta en una matriz de ocupación de dimensión 275×400. Tal como se comentó en el Capítulo 5, si bien se puede usar una resolución más pequeña con el fin de tener mejores resultados, la desventaja es que los tiempos computaciones crecen de manera excesiva.

a) Primera prueba. En esta prueba se evalúa el desempeño del algoritmo en la planificación de trayectorias en distancias largas. Para esto se escogió como punto de partida la coordenada $(36, 42)$ y como punto de llegada la coordenada $(287, 228)$. Luego de ejecutar el algoritmo, la trayectoria hallada se muestra en la Figura 6.2.

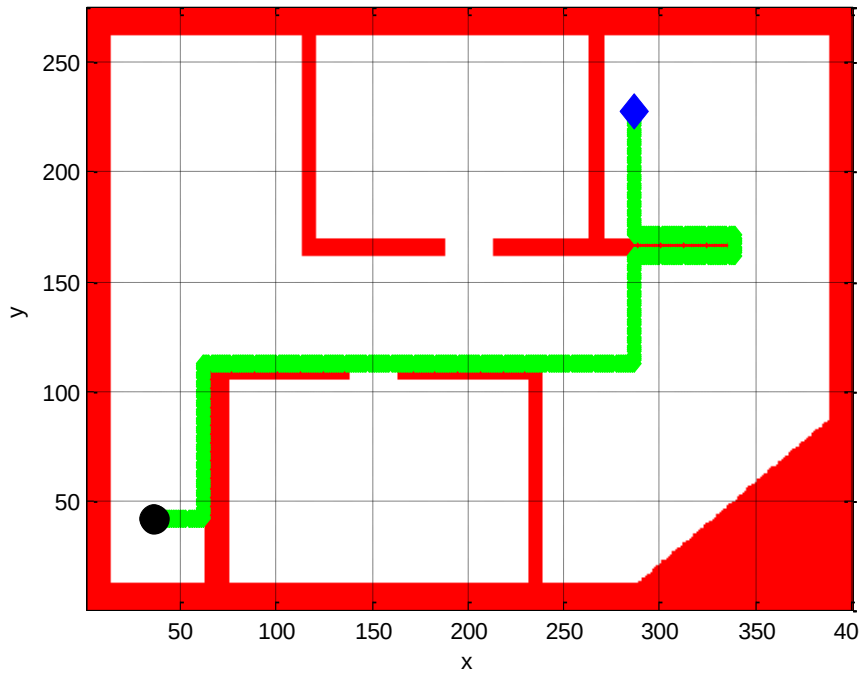


Figura 6.2. Planificación de trayectoria en distancias largas (Fuente: Elaboración propia)

Como puede observarse la trayectoria hallada es efectivamente la más óptima entre la posición de partida y la posición de llegada. El tiempo de ejecución del algoritmo para hallar esta trayectoria tiene un promedio de 6.0 segundos. Este tiempo es relativamente pequeño considerando la magnitud del entorno de navegación del robot.

b) Segunda prueba. En esta prueba se evalúa el desempeño del algoritmo en la planificación de trayectorias en distancias moderadas. Para esto se escogió como punto de partida la coordenada $(101,46)$ y como punto de llegada la coordenada $(139,189)$. Luego de ejecutar el algoritmo, la trayectoria hallada se muestra en la Figura 6.3.

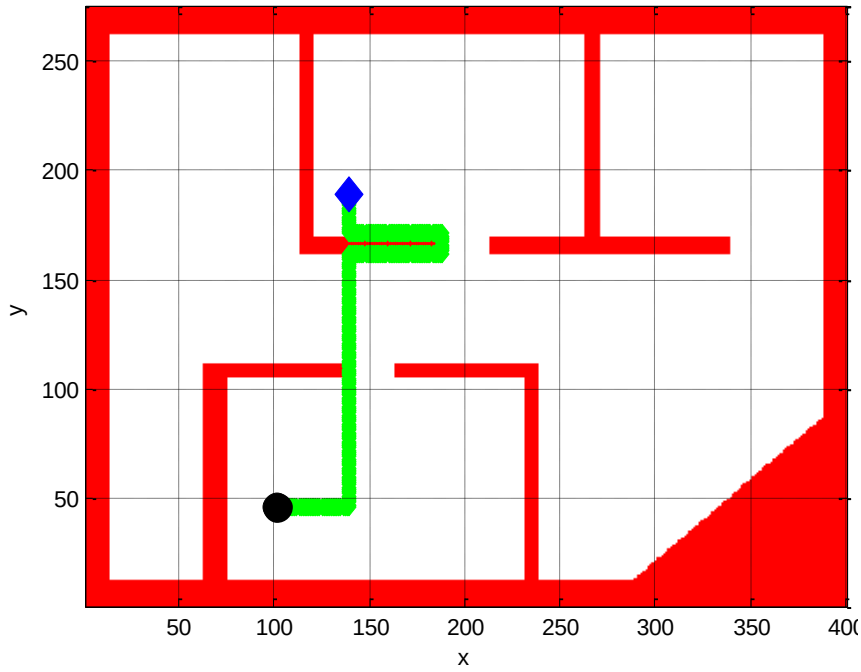


Figura 6.3. Planificación de trayectoria en distancias moderadas (Fuente: Elaboración propia)

En este caso también se tiene que la trayectoria hallada es la más óptima. Por otro lado el tiempo de ejecución del algoritmo en este caso fue de 1.5 segundos.

c) Tercera prueba. En esta prueba se evalúa el desempeño del algoritmo en la planificación de trayectorias en distancias cortas. Para esto se escogió como punto

de partida la coordenada $(105, 45)$ y como punto de llegada la coordenada $(160, 73)$. Luego de ejecutar el algoritmo, la trayectoria hallada es la mostrada en color verde en la Figura 6.4.

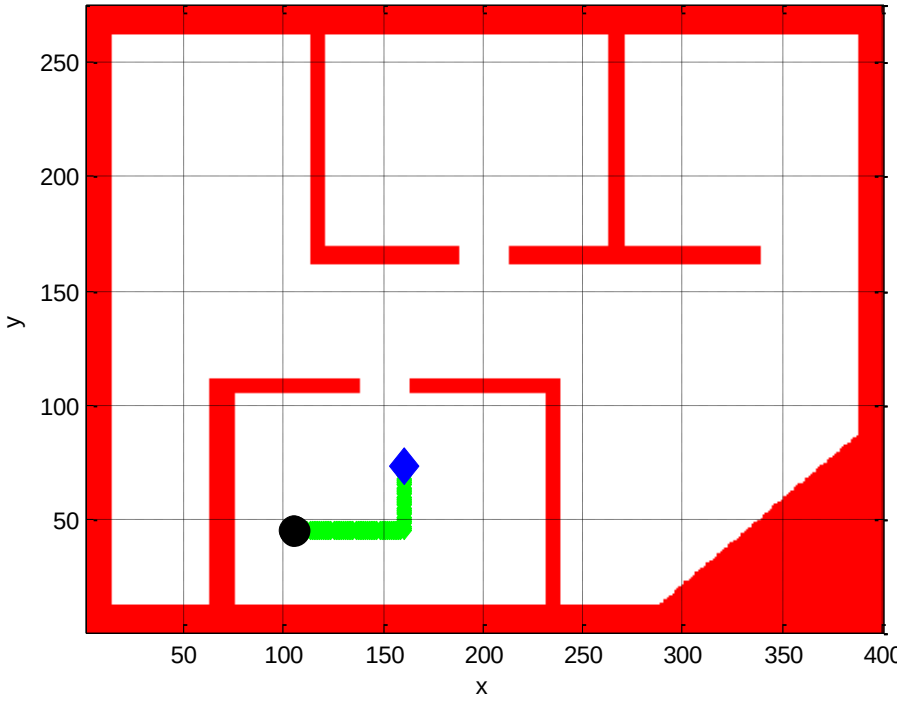


Figura 6.4. Planificación de trayectoria en distancias cortas (Fuente: Elaboración propia)

En este caso el tiempo de ejecución del algoritmo para hallar esta trayectoria tiene un promedio de 0.2 segundos. Nótese que en este caso si bien la distancia corta estaba dada por la línea recta que une el punto inicial y final, la trayectoria obtenida es la más óptima de acuerdo a la heurística de 'Manhattan'

De los resultados obtenidos se desprende que el algoritmo A* implementado en el presente trabajo está altamente calificado para la generación de trayectorias. Además este mismo algoritmo puede usarse para la tarea de re-planeamiento, que se da cuando el robot encuentra en su camino un obstáculo inesperado, y requiere generar un nuevo camino con el fin de llegar a la meta. Otros algoritmos de evitación de obstáculos se muestran en [28], [29], [30].

6.4 Control de Movimiento

Una vez que se ha establecido la trayectoria que el robot debe seguir para llegar a su meta, se deben establecer las acciones de control en cualquier instante de tiempo $u_t = [v_t, w_t]^T$ adecuadas con el fin de lograr tal objetivo. Es decir se debe desarrollar un componente de control de movimiento que asegure que el robot siga cierta trayectoria deseada. En esta sección se describirá como el controlador del tipo PID es capaz de lograr tal objetivo.

6.4.1 Conceptos Básicos

La tarea central del control de movimiento es asegurar que el robot siga la trayectoria establecida por el componente de planeamiento. La tarea de este componente, es asegurar que en cada instante de tiempo la distancia del robot a la trayectoria deseada sea cero. Esta distancia, que es usualmente llamada CTE (cross-talk error), es la variable a ser controlada por el sistema de control de movimiento (Figura 6.5).

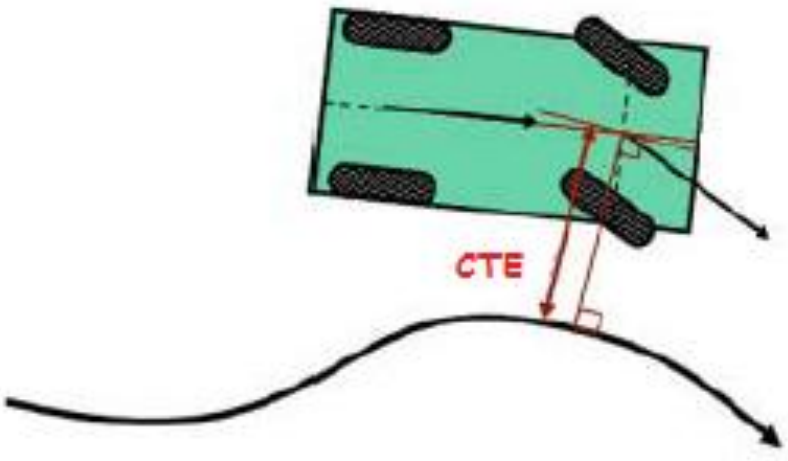


Figura 6.5. Problema del control de movimiento de un robot móvil (Fuente: http://www.mobilerobots.ethz.ch/)

Debido a que la forma de las trayectorias generadas por el sistema de planificación de trayectorias está dada por la unión de líneas rectas (Figuras 6.2, 6.3, y 6.4), esta distancia estará dada simplemente por la distancia de un punto a la recta, la cual puede hallarse de manera muy simple usando métodos geométricos.

6.4.2 Control PID

El controlador PID es uno de los algoritmos más usados tanto en la industria como en los centros de investigación [31], [32]. En el caso de la industria se estima que en más del 90% de los casos se usa este tipo de controlador. Esto es principalmente debido a que es fácil de implementar como a sus propiedades de estabilidad en la mayor cantidad de sistemas. La versión más general de este algoritmo tiene la forma

$$u(t) = K_P\left[e(t) + \frac{1}{T_i}\int_0^t e(t)dt + T_d\,\frac{de(t)}{dt} \right] \tag{6.4}$$

Donde $u(t)$ es la señal de control, $e(t)$ es la señal de error del sistema, K_P es la ganancia proporcional, T_i es la constante de tiempo integral, y T_d es la constante de tiempo derivativo. Adaptando esta señal de control al caso de problemas de control en tiempo discreto, como es el caso del control movimiento de nuestro robot móvil, se tiene la siguiente ley de control

$$u(k) = K_P e(k) + K_I \sum_{i=0}^{k}\left[T\,\frac{e(i-1)+e(i)}{2} \right] + K_D\left(\frac{e(k)-e(k-1)}{T} \right) \tag{6.5}$$

Donde $e(k)$ es la señal de error del sistema en el instante k, K_P es la ganancia proporcional, K_I es la ganancia integral, K_D es la ganancia derivativa, y T es la frecuencia de muestreo. En el caso de nuestro robot la señal de error $e(k)$ está dada por el error CTE.

6.4.3 Resultados

Luego de haber implementado la ley de control PID descrita en la sección anterior, a continuación se muestran los resultados obtenidos usando diferentes valores de las ganancias PID. Para realizar estos experimentos se va a usar el problema que se muestra en la Figura 6.6. En este problema el robot debe seguir la línea de 10 metros de color magenta, dado que la posición inicial del robot está a 20 cm de dicha línea. El objetivo de estas pruebas es establecer la influencia de las ganancias en el movimiento

del robot, para más adelante hallar las ganancias óptimas que le permitan al robot seguir la trayectoria deseada en el menor tiempo y con el menor error posible.

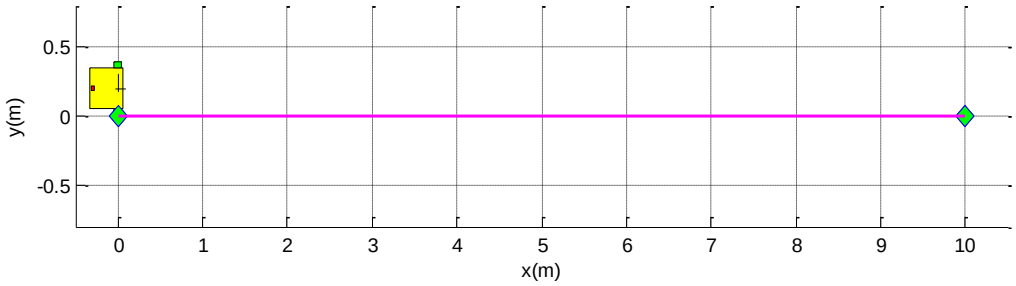
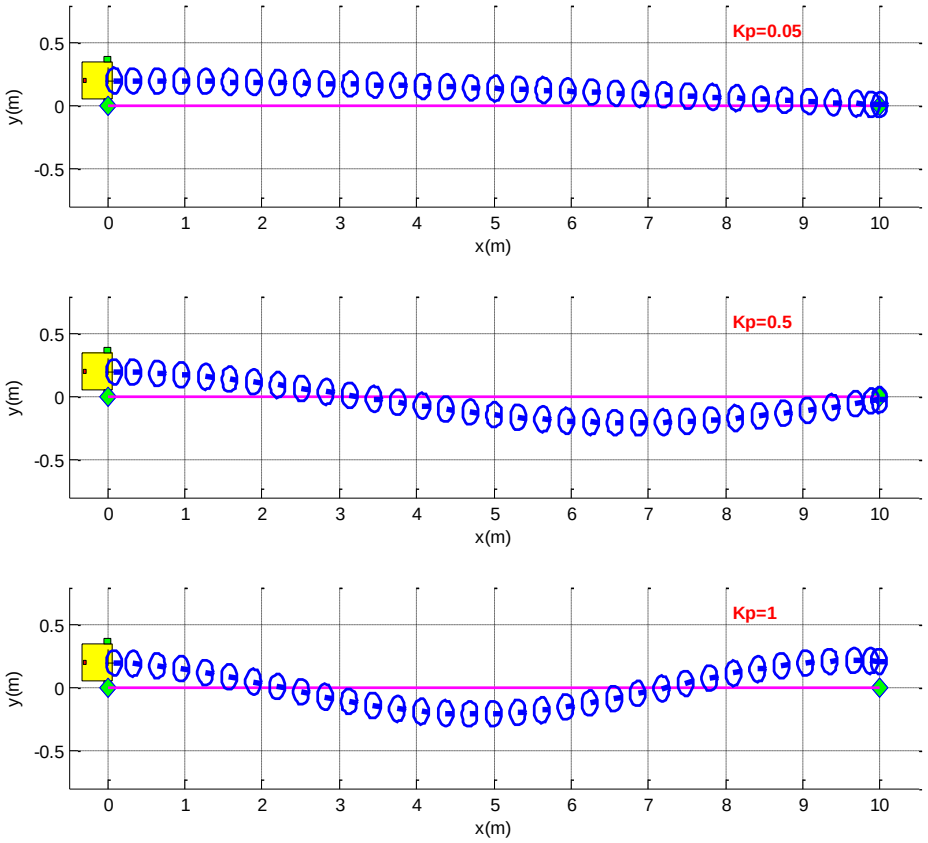


Figura 6.6. Test de prueba para el diseño del controlador PID (Fuente: Elaboración propia)

Primera prueba - Influencia de la ganancia proporcional: En estas pruebas se analiza la influencia de K_P. (Considerando $K_D = 1,\ K_I = 0.005$)



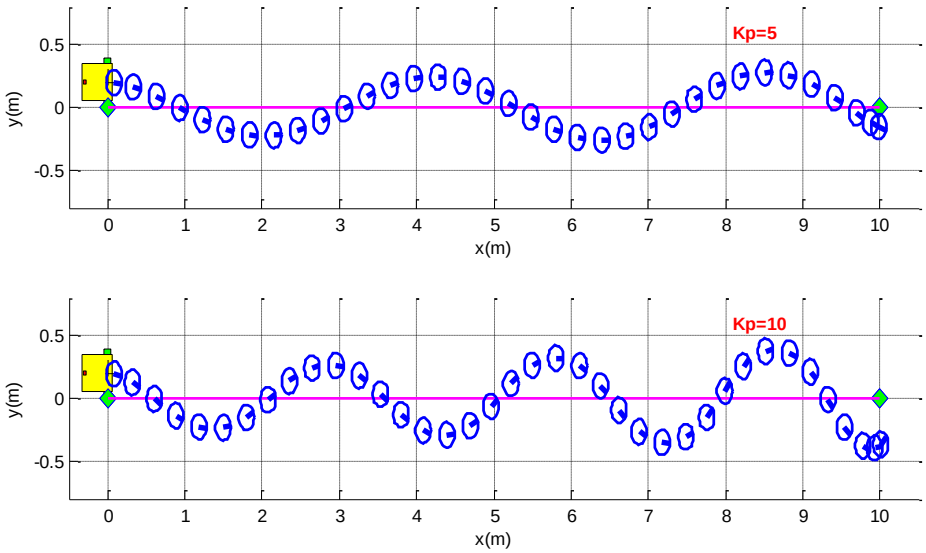
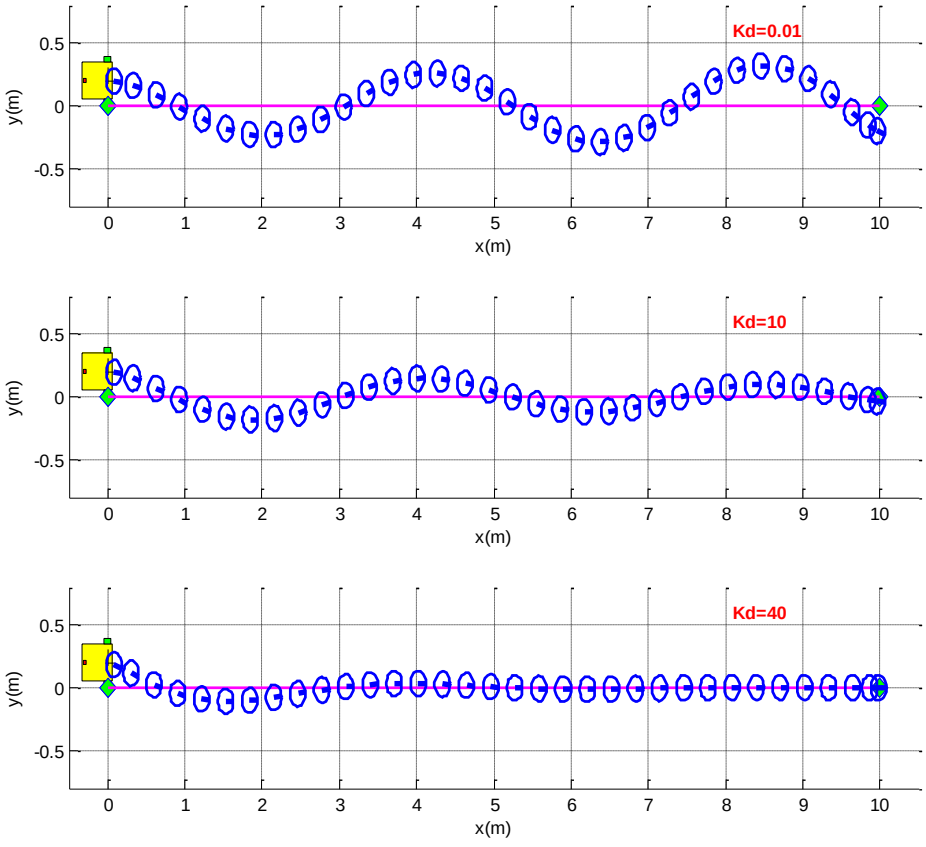


Figura 6.7. Control de movimiento – Influencia de la ganancia proporcional (Fuente: Elaboración propia)

Segunda prueba - Influencia de la ganancia derivativa: En estas pruebas se analiza la influencia de K_D. (Considerando $K_P = 5,\ K_I = 0.005$)



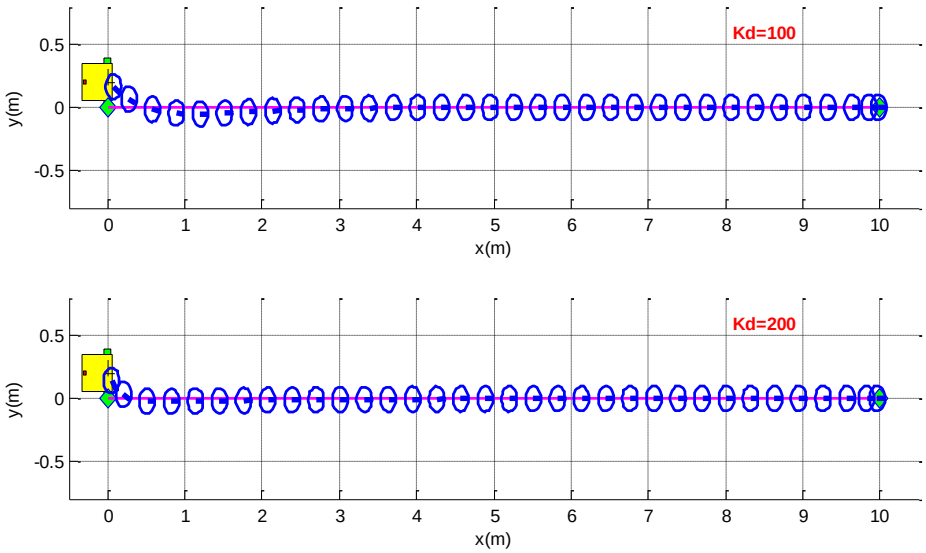
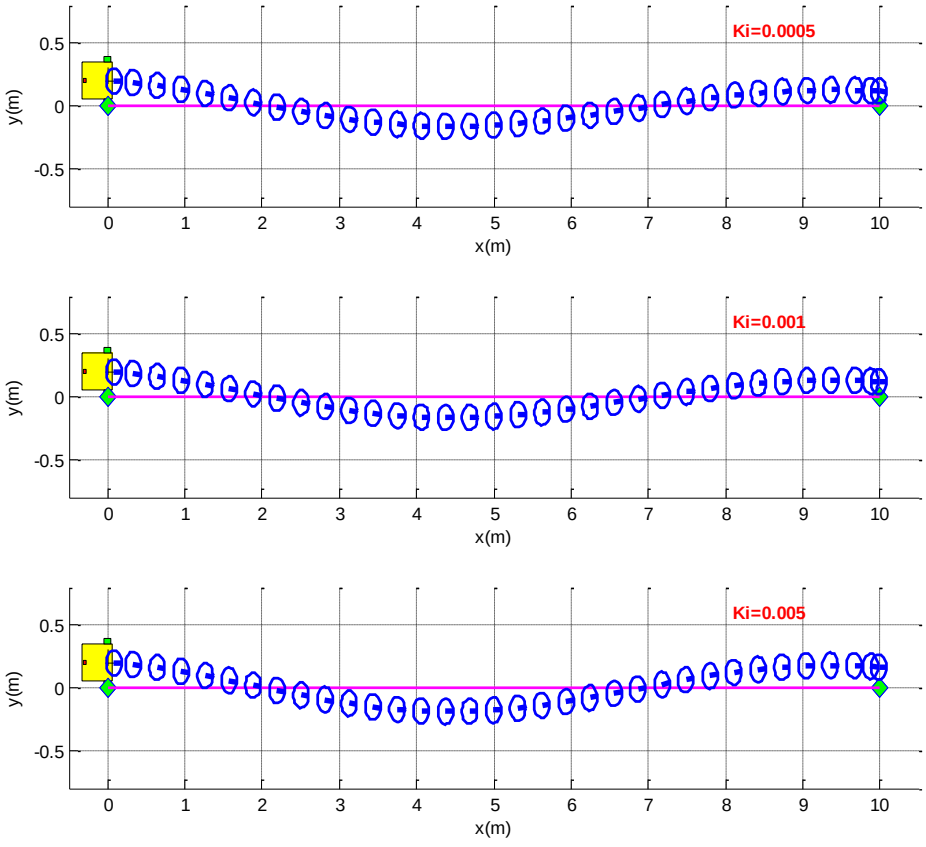


Figura 6.8. Control de movimiento – Influencia de la ganancia derivativa (Fuente: Elaboración propia)

Tercera prueba - Influencia de la ganancia integral: En estas pruebas se analiza la influencia de K_I. (Considerando $K_P = 1,\ K_D = 5$)



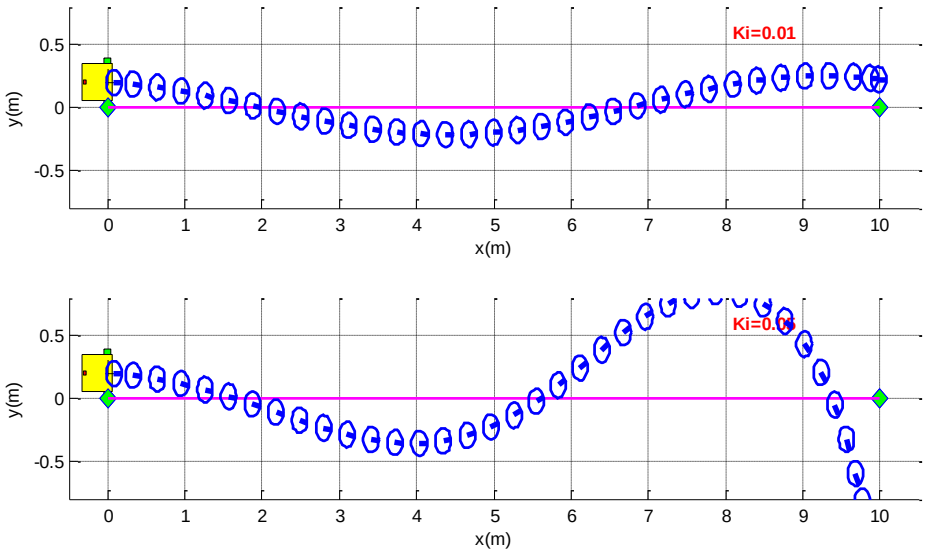


Figura 6.9. Control de movimiento – Influencia de la ganancia integral (Fuente: Elaboración propia)

De los resultados anteriores puede concluirse primeramente que la ganancia proporcional K_P por sí sola no es capaz de realizar un control muy adecuado. Valores pequeños de esta ganancia tienden a reducir el error de manera muy lenta, mientras que valores grandes resultan en errores del tipo oscilatorio (Figura 6.7). Los resultados nos permiten comprobar el hecho de que el controlador del tipo proporcional es del tipo marginalmente estable. Se debe notar que aunque se usen ganancias pequeñas el error aun será oscilatorio.

Respecto a la ganancia derivativa K_D, el objetivo principal de este tipo de señal es añadir una amortiguación al sistema con el fin que el error converja de manera suave a su valor estacionario. Tal como puede apreciarse en la Figura 6.8, al aumentar esta ganancia, se lograr que el robot llegue a la trayectoria deseada de manera más rápida y más suave, sin la presencia de oscilaciones. Tal como se mostró en [33], otra manera de interpretar a los controladores PD es como sistemas que moldean la energía del sistema controlado con el fin que el punto mínimo de energía coincida con la posición deseada.

Finalmente, respecto a la ganancia integral K_I su principal objetivo es la reducción de la señal de error que se acumula en el tiempo. De los resultados de la Figura 6.9 se tiene que se debe usar valores pequeños de esta ganancia, ya que valores muy grandes vuelven al sistema inestable (esto debido a que el error acumulado crece muy

rápidamente, y la señal de control integral se vuelve muy grande respecto a las señales proporcionales y derivativas)

Con el fin de tener un controlador que tenga el mejor desempeño, se debe hacer un proceso de sintonización que no es otra cosa que el proceso de búsqueda de las ganancias K_P, K_I, K_D optimas que resulten en el mejor control de la señal de error. Lamentablemente no existe un método analítico que nos permita hallar de manera automática las ganancias optimas, por lo que la única manera de hallarlas es mediante la experimentación. Luego, de haber realizado varios experimentos se seleccionaron los siguientes valores de las ganancias $K_P = 1, K_I = 0.005$ y $K_D = 50$ (Figuras 6.10 y 6.11).

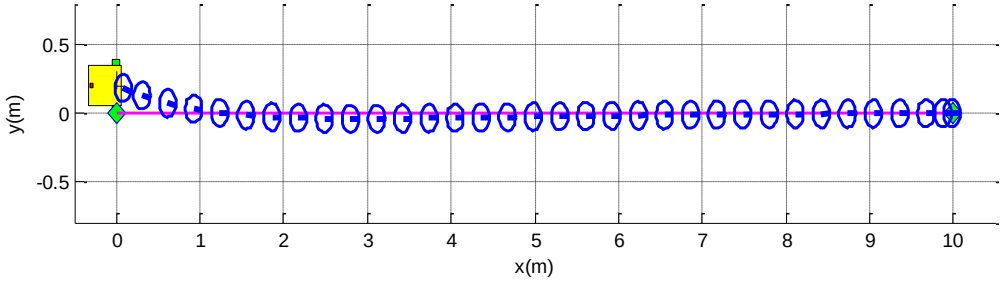


Figura 6.10. Control PID seleccionado para el control de movimiento del robot (Fuente: Elaboración propia)

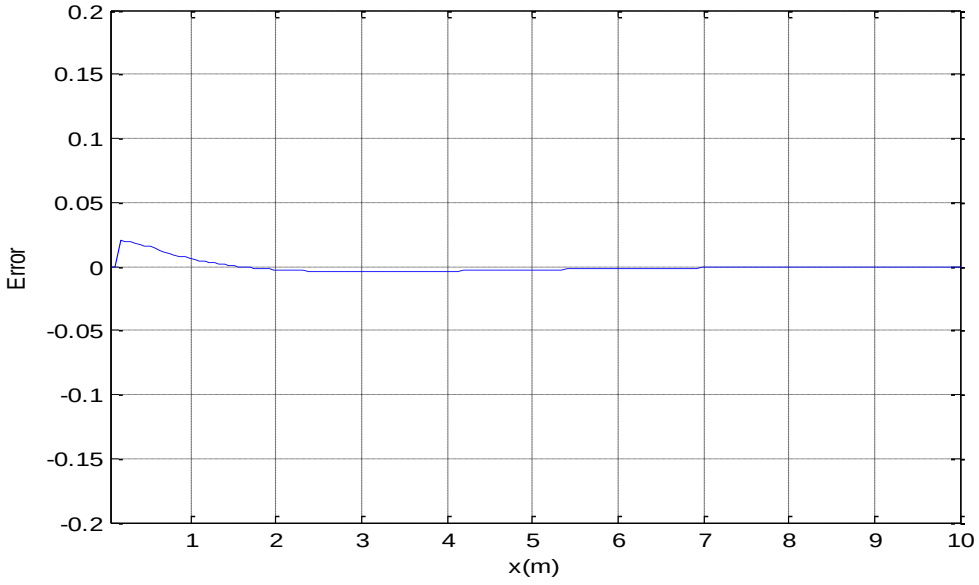


Figura 6.11. Señal de error del controlador PID seleccionado (Fuente: Elaboración propia)

Como puede verse en la Figura 6.10, el controlador propuesto es capaz de lograr que el robot siga la trayectoria deseada con la menor oscilación posible. De la Figura 6.11 puede observarse que el error tiene una forma suave y tiende asintóticamente a cero.

Este controlador PID con las ganancias seleccionadas junto con el componente de planeamiento desarrollado en la sección anterior, son los encargados del movimiento del robot. Ambos componentes garantizan que el robot llegue a las posiciones deseadas de manera óptima y confiable. Ambos componentes, en combinación con los componentes desarrollados en capítulos anteriores (localización y mapeo) permiten que nuestro robot móvil sea capaz de desarrollar tareas de navegación autónoma al más alto nivel.

CAPÍTULO VII
ARQUITECTURA DEL SOFTWARE DE NAVEGACIÓN AUTÓNOMA

Una vez desarrollado todos los componentes necesarios para que un robot tenga competencias de navegación autónoma la tarea siguiente es el diseño de la arquitectura del software de navegación del robot. Es decir la tarea de combinar adecuadamente todos estos componentes en un solo sistema de software de navegación autónoma. En este capítulo se describirá la importancia y la dificultad en la implementación del software de navegación autónoma del robot, la importancia de usar técnicas de programación altamente optimizadas, y los criterios que se deben seguir en la implementación del software de navegación. Finalmente se muestra la arquitectura del sistema de navegación de nuestro robot móvil.

7.1 Software y Librerías de Navegación Autónoma

A diferencia de los manipuladores robóticos que usan las técnicas de control moderno (control adaptivo, deslizante, etc.), el nivel y cantidad de software requerido por las técnicas probabilísticas de navegación autónoma es mucho mayor. Por ejemplo en trabajos tales como control de manipuladores robóticos de 2 o 3 grados de libertad, la cantidad de líneas de código (en MATLAB) esta usualmente en el rango de 300 a 400 [34]. En el caso de nuestro robot móvil, se ha escrito aproximadamente 16,000 líneas de código (en MATLAB). Este código contiene la implementación de los algoritmos descritos en capítulos anteriores, así como las librerías necesarias para poder controlar los diversos sensores usados por el robot (el sensor laser, los motores, los encoders). Es por este motivo que las técnicas modernas de navegación autónomas de robos móviles están ubicadas dentro del campo de la Inteligencia Artificial y las Ciencias de la Computación.

Debido a la magnitud del desarrollo de software requerido para la implementación de robots autónomos, a nivel mundial se han lanzado diversas iniciativas para el desarrollo de librerías de navegación autónoma. La librería “CARMEN – Carnegie Mellon Navigation Toolkit” [35], desarrollada en el instituto de robótica de la Universidad “Carnegie Mellon”, fue una de las primeras en la implementación de algoritmos de navegación probabilísticos. Esta librería se usó exitosamente en el robot Minerva, que hacia servicio de guía en un museo; y el robot Nursebot, que fue diseñado para servir como asistente a las personas de la tercera edad en los centros para adultos (Figura 7.1).

Figura 7.1. El robot Minerva (izquierda) y el robot Nursebot (derecha) (Fuente: http://www.probabilistic-robotics.org/)

El proyecto “STAIR: Stanford Artificial Intelligence Robot” (Figura 7.2), desarrollado en el laboratorio de Inteligencia Artificial de la Universidad de Stanford [36], represento un hito más importante ya que no solo se implementó librerías de navegación autónoma, sino además librerías de procesamiento de voz, visión artificial, manipulación robótica, etc., dando como resultado uno de los robots más avanzados a nivel mundial, capaz de desarrollar como navegar en ambientes con la capacidad de abrir puertas, hacer inventariado, usar los ascensores, traer objetos, etc.

Figura 7.2. STAIR: Stanford Artificial Intelligence Robot (Fuente: http://stair.stanford.edu/)

El proyecto STAIR dio lugar más adelante al desarrollo del robot PR2 (Figura. 7.3), uno de los robots más avanzados a nivel mundial en la actualidad, y a la creación de la empresa Willow Garage, la que actualmente se encarga del desarrollo y mantenimiento del software "*ROS: Robot Operating System*" [37], el cual tiene como objetivo final ser el Linux de la robótica. Esta librería no solo es usada por el robot PR2 sino por una serie de robots muy avanzados en diferentes partes del mundo.

Figura 7.3. El robot PR2 (Fuente: http://www.willowgarage.com/)

Teniendo como fuente de inspiración las librerías antes mencionadas, fue que se desarrolló el software de navegación autónoma de nuestro robot “R2D2-00”. En el caso de nuestra librería de navegación, el código esta hecho íntegramente en Matlab, a diferencia de las librerías mencionadas anteriormente que están hechas en C/C++ y solo se pueden ejecutar en el sistema operativo Linux.

Con el fin de tener una implementación altamente eficiente se hizo uso de técnicas de programación Matlab altamente optimizadas (como operaciones vectorizadas, pre asignación de memoria, uso de manejadores, etc.) con el fin de conseguir una eficiencia lo más cercana posible a C/C++. Tal como se verá en el capítulo de resultados experimentales, nuestras librerías de navegación permiten que el robot sea capaz de operar en tiempo real.

7.2 Criterios de Programación en la Implementación del Software

En la implementación del software de navegación autónoma es importante seguir criterios adecuados de programación con el fin de poder manejar la gran cantidad de líneas de código necesarias (en nuestro caso se escribió aproximadamente 16,000 líneas de código), que el software sea fácilmente manejable por el usuario, y que sea adaptable a nuevos cambios y/o mejoras. Teniendo en cuenta estas recomendaciones, en la implementación del software de navegación de nuestro robot se siguió los siguientes principios (que son usados por las librerías anteriormente mencionadas):

a) Modularidad

El software del robot R2D2-00 fue diseñado de manera modular, en donde cada componente se construyó como un módulo separado e independiente. Esto significa que en la implementación de cada componente de navegación (localización, mapeo, planeamiento, control) y en la implementación de las librerías de control (laser, motores), se hizo uso de las técnicas de programación orientada a objetos. Aun cuando el uso de estas técnicas introduce pequeños retardos en los tiempos de ejecución, sus principales ventajas son:

- **Flexibilidad**. El usuario puede usar diferentes tipos de configuraciones en la navegación del robot. Es decir se pueden usar diferentes tipos de robots y sensores simplemente definiendo diferentes tipos de objetos de la clase base.

- **Confiabilidad**. Al producirse un error, se puede verificar de manera rápida el modulo responsable de dicho error. Por otro lado, en el proceso del desarrollo del software es más fácil desarrollar pruebas de funcionamiento de cada componente por separado.

- **Extensibilidad**. La modificación y/o actualización de los diferentes componentes es mucho más fácil. Además el desarrollo e incorporación de nuevos componentes como visión, audio, etc. es mucho más manejable.

La modularidad es de gran importancia en la robótica móvil debido a la gran cantidad de líneas de código de cada componente, y a la necesidad de tener un sistema que sea fácilmente adaptable a los cambios en software y/o hardware del robot. Este criterio se usa no solo en el desarrollo de las librerías para los sistemas robóticos más avanzados a nivel mundial, sino también en otras áreas de la ingeniería.

b) Robustez

El software del robot R2D2-00 fue diseñado de manera robusta de tal manera que pueda afrontar diversos tipos de fallas. En particular los diversos métodos de las clases muestran en pantalla mensajes de error y/o emergencia cuando ha sucedido algún evento que perjudica el buen desenvolvimiento del sistema. Por ejemplo cuando, no se ha instalado adecuadamente un sensor, cuando las velocidades exceden sus valores límites, etc., se muestra en pantalla un mensaje con el respectivo error, y las acciones a tomar con el fin de superar el inconveniente.

c) Eficiencia

Otro componente esencial del software de navegación es la eficiencia en los tiempos de ejecución. Uno de los principales inconvenientes de los algoritmos de navegación probabilísticos es que requieren un gran tiempo y recursos computacionales en su funcionamiento. Este hecho significa que se debe tener sumo cuidado a la hora de implementar estos algoritmos; una mala implementación de los mismos puede generar tiempos de ejecución demasiado grandes, lo que haría que el robot no sea capaz de operar en tiempo real.

En la implementación de las librerías de navegación de nuestro robot móvil se usaron las técnicas de programación en Matlab más avanzadas (operaciones vectorizadas, pre asignación de memoria, etc.) con el fin de minimizar el tiempo de ejecución de los diversos algoritmos. Tal como se puede apreciar en los videos que acompañan el presente trabajo, nuestro robot es capaz de operar en tiempo real.

7.3 Arquitectura del Software de Navegación

Una vez definido los criterios de programación a seguir, el siguiente paso es el diseño de la arquitectura del software de navegación, la cual se refiere a la adecuada combinación de los principales componentes que son necesarios para la navegación autónoma de un robot móvil (Figura 7.4). Además se debe tener en cuenta también, la instalación de los componentes que se encarga del control del hardware del robot.

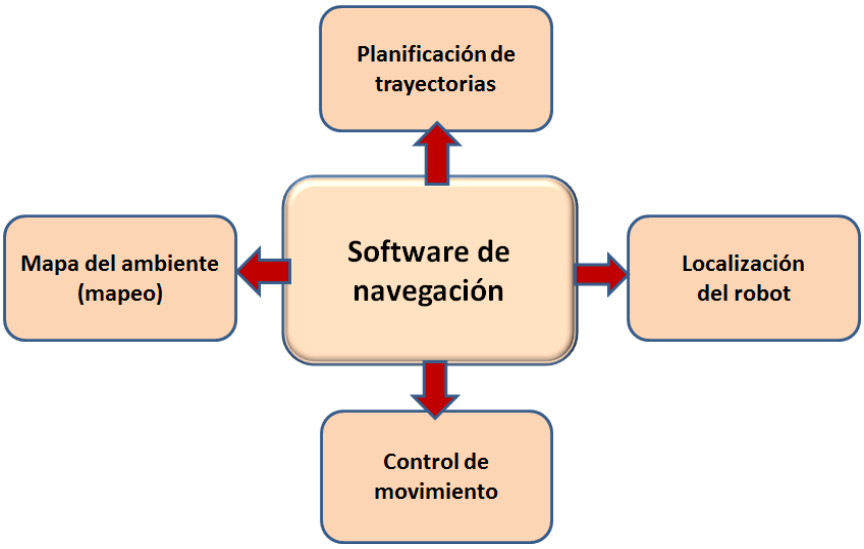


Figura 7.4. Componentes del sistema de navegación del robot móvil (Fuente: Elaboración propia)

Además de los componentes mostrados en la Figura 7.4, se debe añadir los módulos que se encargan del control del hardware del robot (sensores y actuadores). Por otro lado, en el diseño de la arquitectura de navegación se debe tener en cuenta los procesos que deben operar en tiempo real y los que operan fuera de línea. En el caso de tareas de navegación autónoma, de ahora en adelante se va a asumir que el robot tiene a mano el mapa de su ambiente de navegación. En caso contrario, el procedimiento para adquirir dichos mapas fue descrito en el Capítulo 6 de la presente tesis.

Tomando en cuenta el nivel de cognición de cada uno de los módulos del robot, la arquitectura de software del robot se dividió en 3 capas (Tabla 7.1). En la capa más baja, se encuentran los módulos que interactúan directamente con el hardware del robot. En la capa intermedia, se encuentran los módulos de navegación del robot. Y en la capa superior se encuentran los módulos de interacción que se encargan de definir tareas y la interacción con el usuario.

Tabla 7.1. Arquitectura del software de navegación autónoma (Fuente: Elaboración propia)

Módulos de interacción con el usuario (asignación de tareas, interfaz de usuario)
Módulos de navegación autónoma (localización, mapeo, planeamiento, control de movimiento, evitación de obstáculos)
Módulos de interface con el hardware (motores, sensores)

Esta arquitectura permite que el usuario pueda definir tareas al más alto nivel, mediante los módulos de interacción con el usuario. Los otros niveles no son accesibles al usuario, y están controlados directamente por el robot. De esta manera, nuestro robot puede ser usado por cualquier tipo de usuarios de manera fácil y sencilla.

Finalmente, se debe notar que las tareas de navegación autónoma están definida por el siguiente proceso: Primero, el usuario, a través de la interfaz de usuario, le asigna una tarea al robot. Por ejemplo una tarea como "*ve a la oficina de posgrado de la FIEE y toma una foto*", para lo cual se le suministra al robot las coordenadas de la oficina de posgrado y se activa el módulo de visión. Segundo, una vez definida la tarea, el módulo de planeamiento se encarga de planificar el camino más óptimo con el fin de llegar a la meta desde la posición actual del robot. Tercero, una vez definido el camino, el robot empieza su proceso de navegación, haciendo uso de sus componentes de localización, control de movimiento y evitación de obstáculos. Finalmente cuando el robot completa su tarea asignada, está listo para la ejecución de una nueva tarea.

CAPÍTULO VIII
RESULTADOS EXPERIMENTALES

En este capítulo se describirá los resultados experimentales obtenidos en la navegación autónoma del robot móvil "*R2D2-00: El Robot Autónomo*" (Figura 8.1) en diversos ambientes de la Universidad Nacional de Ingeniería. El objetivo de estas pruebas es comprobar si el robot es capaz de desplazarse de manera autónoma en entornos reales donde hay personas, objetos inesperados, y un nivel considerado de ruido en el movimiento del robot, debido a los diversos tipos de pisos (los cuales pueden presentar baches, agujeros, etc.). Los videos de las pruebas de navegación que se muestran a continuación se incluyen en el CD que acompaña al presente trabajo.

Figura 8.1. R2D2-00: El Robot Autónomo (Fuente: Elaboración propia)

8.1 Navegación Autónoma en la Sección de Posgrado de la FIEE-UNI

La primera prueba de navegación autónoma se desarrolló en la sección de posgrado de la FIEE-UNI (Figura 8.2). Las características de este ambiente es que contiene un nivel de ruido moderado, debido a que hay pocas personas que se desplazan en dicho ambiente, y debido a que el piso no contiene imperfecciones considerables (baches, agujeros, etc.)

Figura 8.2. Ambiente de navegación – Primera prueba (Fuente: Elaboración propia)

En la Figura 8.3 se muestra el mapa de ocupación de este ambiente. Este mapa contiene el pasadizo principal, y la sección A (en posteriores experimentos se van a considerar todos los salones). Como puede notarse el mapa no es fiel reflejo del ambiente de navegación real (ya que no contiene algunos detalles presentes en el ambiente real como la mesa ubicada en el pasadizo, y la mesa y las carpetas del salón A). En este mapa, los puntos en rojo son los puntos de paso por los cuales debe pasar el robot en su movimiento. El objetivo de esta prueba es que el robot empiece en la posición (0,1) haga el recorrido definido por los puntos en rojo, y vuelva al punto de partida, comprobando los resultados de las simulaciones con los resultados en la operación real del robot.

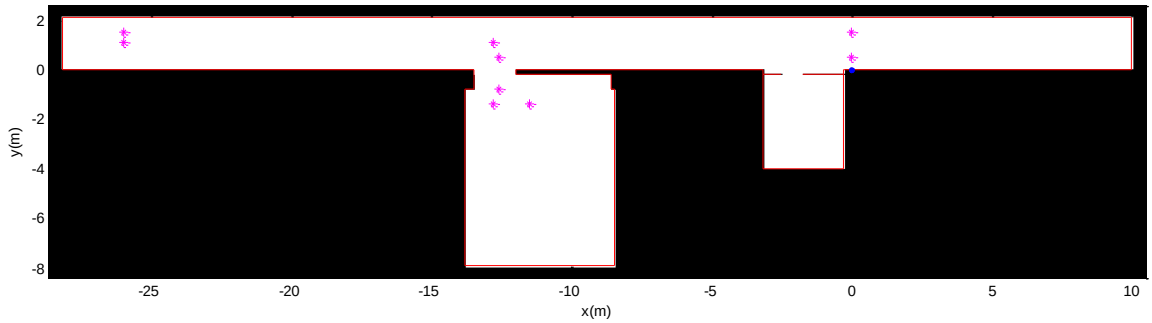


Figura 8.3. Mapa del ambiente de navegación – Primera prueba (Fuente: Elaboración propia)

En la Figura 8.4 se muestra la trayectoria recorrida según las estimaciones del robot. Comparando estos resultados con el resultado del movimiento real del robot, se tiene que el robot estuvo correctamente localizado en cualquier instante de tiempo, lo cual le permitió llegar a los puntos de paso sin ningún problema, para finalmente llegar a la posición final deseada. Debe notarse, que si el robot no tendría la capacidad de localizarse "correctamente" en el ambiente dado, este no sería de pasar exactamente por los puntos de paso establecido. Si el robot no estuviese correctamente localizado este creería estar pasando por las trayectorias deseadas, pero realmente, este estaría realizando trayectorias totalmente distintas a las esperadas (lo que en la práctica haría que el robot se choque con puertas, paredes, etc.).

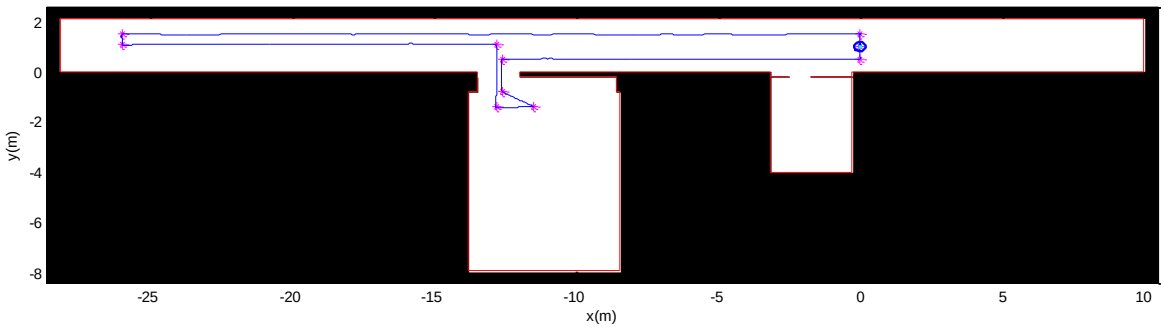


Figura 8.4. Trayectoria recorrida según el robot – Primera prueba (Fuente: Elaboración propia)

En la Figura 8.5 se muestra el error en el control de trayectoria del robot durante todo el proceso de navegación. Como puede notarse este error siempre estuvo acotado con un margen de 5cm. Este resultado garantiza el buen funcionamiento del módulo de control de movimiento, el cual tiene como objetivo que el robot siga la trayectoria establecida por su módulo de control de trayectoria.

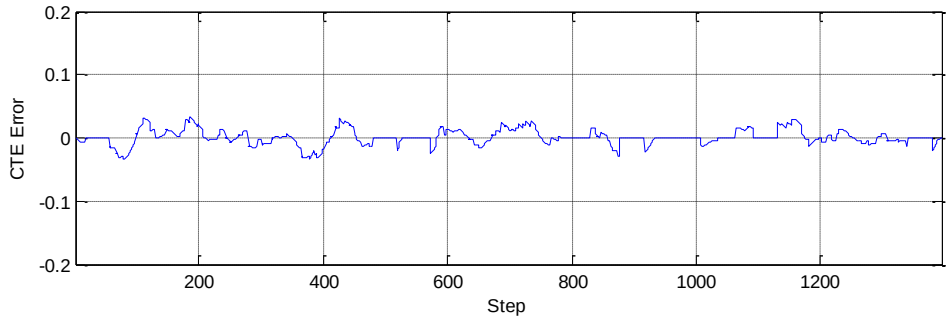


Figura 8.5. Error en el control de trayectoria – Primera prueba (Fuente: Elaboración propia)

Finalmente, en la Figura 8.6 se muestran las velocidades de los motores del robot. Estas velocidades fueron controladas por el módulo de control de movimiento, y como se observa la velocidad máxima requerida fue de aproximadamente 5 rad/s.

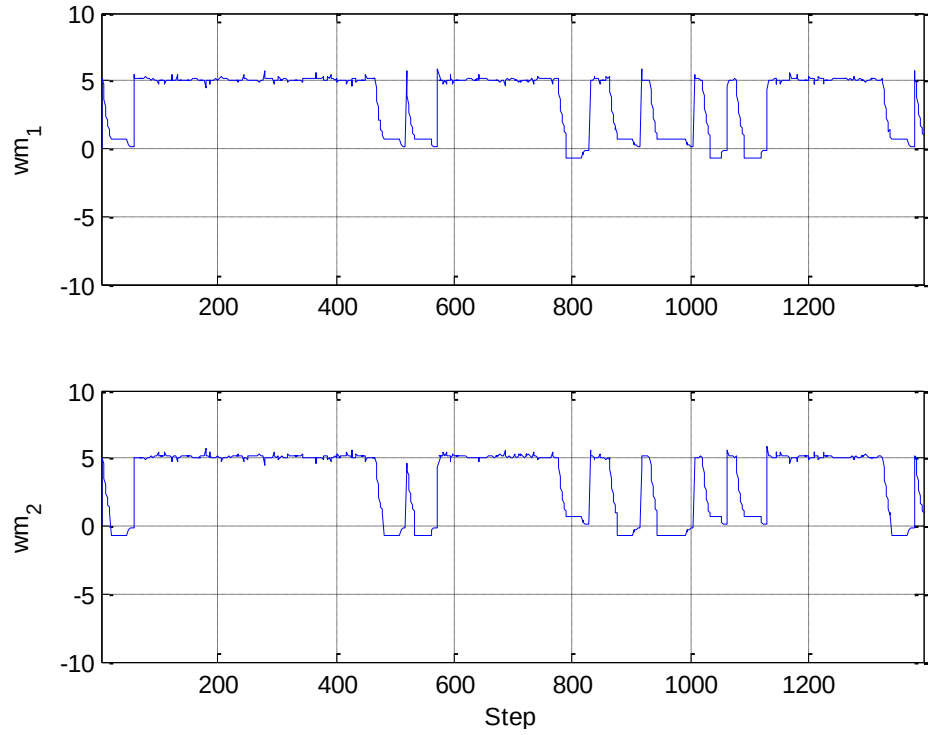


Figura 8.6. Velocidades angulares (rad/s) de los motores del robot – Primera prueba (Fuente: Elaboración propia)

Finalmente, debe señalarse que esta prueba se realizó varias veces, y en todas ellas el robot fue capaz de realizar la tarea de navegación establecida. De esta manera se puede garantizar que nuestro robot es capaz de realizar tareas de navegación autónoma en este ambiente de navegación, el cual tiene un nivel de ruido/incertidumbre moderado.

8.2 Navegación Autónoma en el Segundo Piso de la FIM-UNI

La segunda prueba de navegación autónoma se desarrolló en el segundo piso de la facultad de Ingeniería Mecánica de la UNI (Figura 8.7). Este ambiente, a diferencia de la sección de posgrado de la FIEE, tiene un nivel de ruido/incertidumbre muy elevado, debido a la mayor presencia de personas (estudiantes y profesores), debido a las múltiples imperfecciones del piso de este ambiente y debido a la presencia de objetos no especificados en el mapa de navegación (mesas, tachos de basura, etc.). Además se debe agregar que este ambiente es considerablemente más grande. Todas las dificultades anteriores hacen que la navegación autónoma de cualquier robot en este ambiente sea todo un reto.

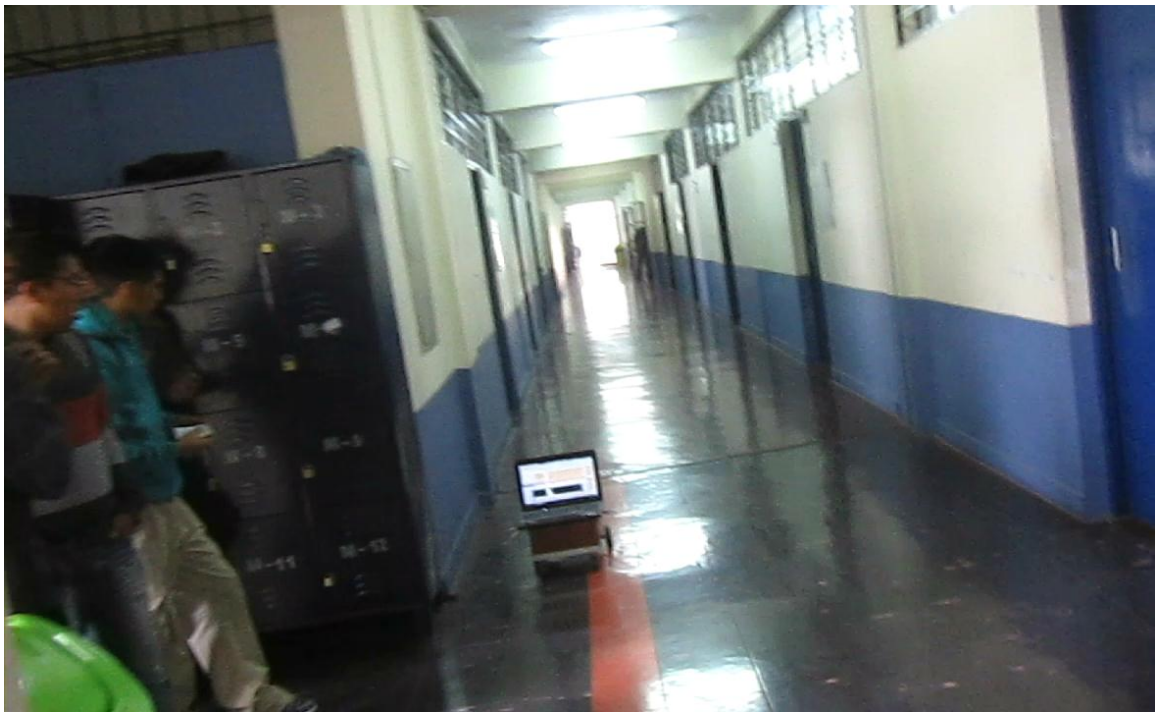


Figura 8.7. Ambiente de navegación – Segunda prueba (Fuente: Elaboración propia)

Respecto a las imperfecciones del piso de este ambiente, en la Figura 8.8 se muestran los casos más relevantes encontrados en el desarrollo del presente experimento de navegación autónoma. La primera imperfección se refiere a la presencia de un agujero de una profundidad considerable; y la segunda se refiere a la presencia de un canal a lo largo del pasadizo. Ambas imperfecciones representa una fuente de ruido "considerable" en el movimiento del robot.

Figura 8.8. Imperfecciones en el piso del ambiente de navegación – Segunda prueba (Fuente: Elaboración propia)

En la Figura 8.9 se muestra el mapa de ocupación de este ambiente. Al igual que en el caso del mapa de la sección de posgrado, este mapa no contiene la presencia de algunos detalles como (basureros, casilleros, etc.). En este mapa, los puntos en rojo son los puntos de paso por los cuales debe pasar el robot en su movimiento. El objetivo de esta prueba es que el robot empiece en la posición (0,0) haga el recorrido definido por los puntos en rojo, y termine en la posición (8,1).

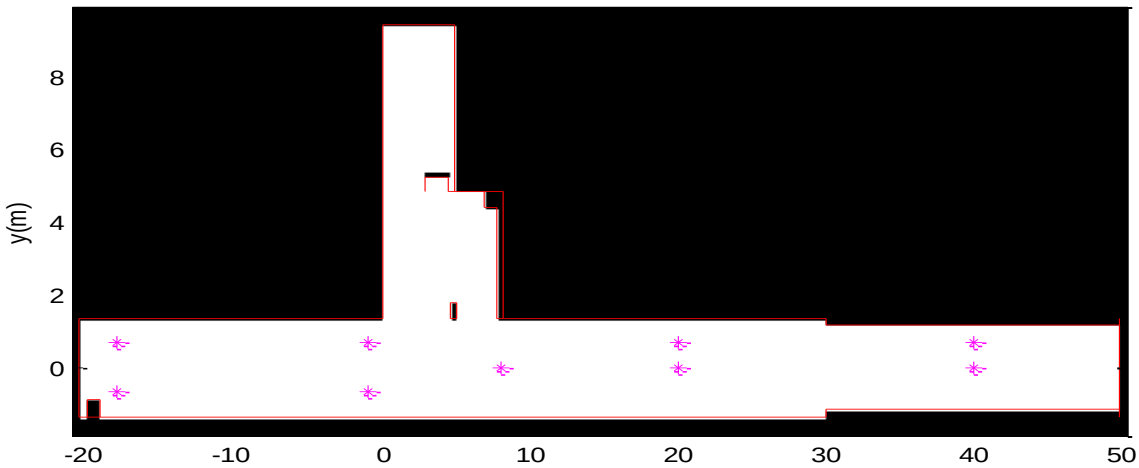


Figura 8.9. Mapa del ambiente de navegación – Segunda prueba (Fuente: Elaboración propia)

En la Figura 8.10 se muestra la trayectoria recorrida según la estimación del robot. Comparando estos resultados con el recorrido real del robot, se puede comprobar que

efectivamente el sistema de localización Monte Carlo mejorado es capaz de estimar la configuración del robot aún bajo condiciones extremas. Si el robot hubiese estado mal localizado no hubiese sido capaz de realizar la tarea asignada en el experimento real, y hubiese terminado realizando una trayectoria muy distinta a la deseada.

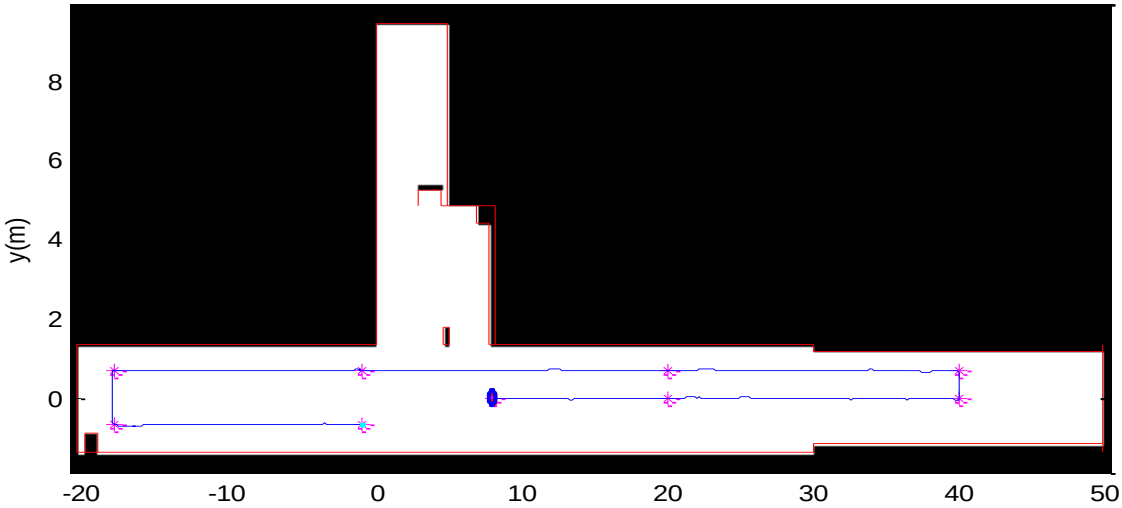


Figura 8.10. Trayectoria recorrida según el robot – Segunda prueba (Fuente: Elaboración propia)

En la Figura 8.11 se muestra el error en el control de trayectoria del robot durante todo el proceso de navegación. Al igual que en el experimento desarrollado en la sección de posgrado, este error siempre estuvo acotado con un margen de 5 cm.

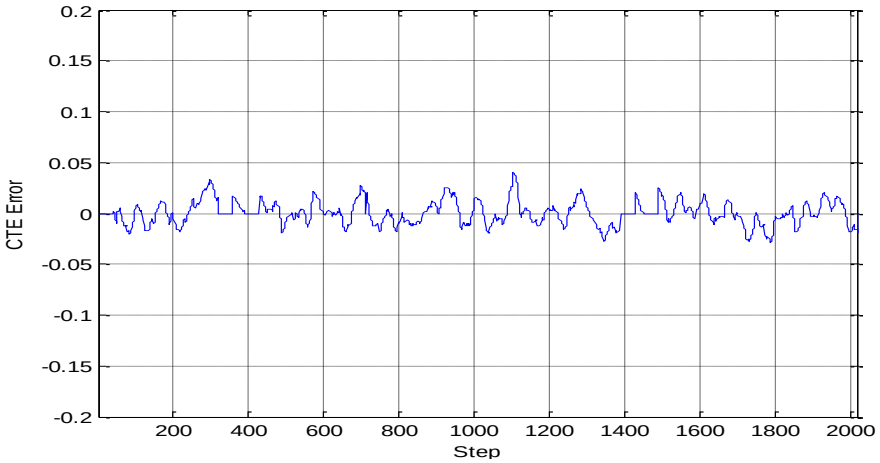


Figura 8.11. Error en el control de trayectoria – Segunda prueba (Fuente: Elaboración propia)

Finalmente, en la Figura 8.12 se muestran las velocidades de los motores del robot. Estas velocidades fueron controladas por el módulo de control de movimiento, y como se observa la velocidad máxima requerida fue de aproximadamente 5 rad/s.

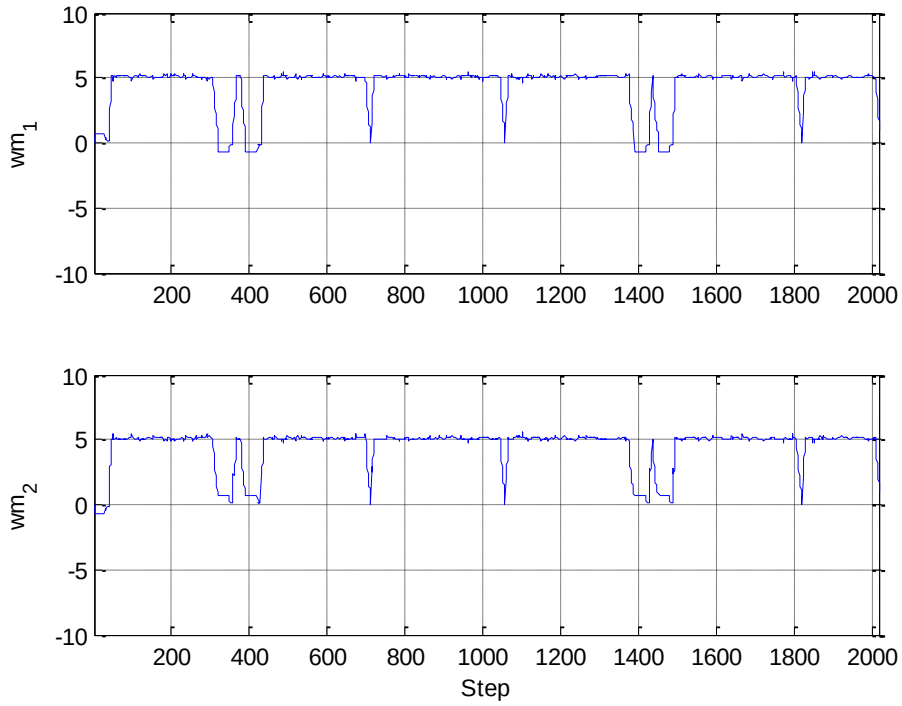


Figura 8.12. Velocidades angulares (rad/s) de los motores del robot – Segunda prueba (Fuente: Elaboración propia)

Finalmente, debe señalarse que esta prueba se realizó varias veces, y en todas ellas el robot fue capaz de realizar la tarea de navegación establecida. De esta manera se puede garantizar que nuestro robot es capaz de realizar tareas de navegación autónoma en este ambiente de navegación, el cual tiene un nivel de ruido/incertidumbre considerable.

8.3 Conclusiones

De los resultados anteriores puede concluirse que el robot móvil implementado en la presente tesis es capaz de realizar tareas de navegación autónoma en entornos dinámicos “reales”, los cuales están sujetos a diversas fuentes de ruido. Esta capacidad hace que nuestro robot pueda realizar tareas complejas como monitoreo, servicio de transporte de objetos, servicio de guía, etc.

CONCLUSIONES Y RECOMENDACIONES

1. En el problema de la localización, el algoritmo Monte Carlo mejorado presentado en esta tesis, presenta una serie de ventajas sobre el algoritmo Monte Carlo estándar. Al disminuir la frecuencia de re-muestreo es más robusto al problema de pérdida de la diversidad de las muestras; y al hacer uso de algoritmos de re-muestro más avanzados se disminuye el problema de la varianza en las muestras. Este algoritmo nos permite estimar la posición de nuestro robot con un error promedio de 3 cm, y la orientación con un error aproximado de 0.02 radianes Tanto en las simulaciones como en los resultados experimentales se comprobó el correcto funcionamiento del algoritmo Monte Carlo mejorado presentado en esta tesis.

2. En el problema de la construcción automática de mapas o mapeo, el algoritmo de mapeo Monte Carlo Rao-Blackwellized mejorado implementado en la presente tesis es capaz de construir mapas de navegación de alta calidad, coherentes con los mapas reales de los ambientes de navegación. Estas mejoras se deben en gran parte al uso de una mejor distribución propuesta, la cual al tomar en cuenta no solo la señal de control sino también las mediciones de los sensores, permite tener una distribución con una menor varianza y mucho más centrada alrededor de la configuración real del robot. La importancia de este resultado, está en la capacidad de nuestro robot de construir mapas de alta calidad de cualquier ambiente de navegación.

3. Tanto en el problema de localización como de mapeo, los algoritmos mejorados implementados en la presente tesis permiten el ahorro de los recursos computacionales necesarios para ejecutar los mismos. En la localización, el paso de re-muestreo se hace solo cuando es necesario; y en el mapeo, se requiere un número menor de muestras para la construcción de mapas de alta calidad. Estos ahorros son muy importantes ya que el robot debe operar en tiempo real, y además a que los recursos de memoria en una PC no son infinitos. Estos ahorros de recursos computacionales hacen posible la inserción de otros módulos adicionales (módulos de visión, audio, etc.).

4. El módulo de planeamiento basado en el algoritmo A* implementado en el presente trabajo permite dotar a nuestro robot con el nivel de cognición necesario para que este pueda tomar decisiones con el fin de lograr sus metas. Específicamente, este módulo le permite al robot encontrar la trayectoria más óptima con el fin de llegar a una meta determinada. La adecuada implementación de este algoritmo nos ha permitido el cálculo de estas trayectorias en tiempos relativamente pequeños.

5. El módulo de control de movimiento basado en el controlador PID implementado en el presente trabajo, permite que el robot siga las trayectorias calculadas por el módulo de planeamiento. De esta manera, este módulo garantiza que el robot pueda llegar a las posiciones, o haga las tareas, determinadas por el usuario.

6. Con el fin de integrar los módulos de localización, mapeo, planeamiento y control en un solo sistema de navegación, se ha hecho uso de una arquitectura de 3 capas. En la capa más baja, se encuentran los módulos que interactúan directamente con el hardware. En la capa intermedia, se encuentran los módulos de navegación del robot. Y en la capa superior se encuentran los módulos de interacción que se encargan de definir las tareas y la interacción del robot con el usuario.

7. En el presente trabajo se ha implementado exitosamente el robot bautizado como *"R2D2-00: El Robot Autónomo"*, el primer robot móvil autónomo desarrollado en nuestro país capaz de desarrollar tareas al más alto nivel (tareas como transporte de objetos, monitoreo, etc.). Este robot, al contar con un software de navegación que consta de los módulos de localización, mapeo, planeamiento y control es capaz de realizar tareas de navegación autónoma en entornos reales no estructurados. De los resultados obtenidos en las pruebas experimentales se puede afirmar el correcto funcionamiento de nuestro robot en diferentes tipos de entornos.

8. Se recomienda el uso del software de alto nivel como MATLAB, Octave, o Python en la implementación de algoritmos probabilísticos de navegación autónoma. Esto principalmente debido a gran cantidad de líneas de código necesario a escribir (nuestro software de navegación consta de aproximadamente 16,000 líneas de código). No se recomienda el uso de otras herramientas software como LabVIEW.

9. Mejorar la velocidad de ejecución del software de navegación ya sea implementando una versión en C/C++, o usando una combinación de códigos MATLAB/C++ mediante archivos del tipo MEX. Esta última posibilidad es la más conveniente, debido a que

nos permite tomar lo mejor de los dos lenguajes, es decir, la velocidad de ejecución en C/C++, y las herramientas de manejo de datos y visualización de MATLAB.

10. Implementar una versión del software de navegación del robot usando software libre (Linux). Es un hecho ya establecido, que el software libre es mucho más conveniente, robusto y eficiente en aplicaciones científicas.

BIBLIOGRAFÍA

[1] Russell, S., Norvig, P., *Artificial Intelligence: A Modern Approach*. 3rd edition. New York, Prentice Hall International, 2010.

[2] DeGroot, M. H, Schervish, M. J. *Probability and Statistics*. Addison Wesley 3rd Edition, 2001.

[3] Kalman, R. E. 1960. "*A new approach to linear filtering and predictions problems"*. Trans. ASME, Journal of Basic Engineering. 1960.

[4] L. R. Rabiner. "*A tutorial on hidden Markov models and selected applications in speech recognition"*. In Proceedings of the IEEE, 1989.

[5] A. Doucet. "*On sequential simulation-based methods for Bayesian filtering"*. Technical Report CUED/F-INFENG/TR 310, Cambridge University, Department of Engineering, Cambridge, UK, 1998.

[6] H. F. Durrant-Whyte. "*An Autonomous guided vehicle for cargo handling applications"*. International Journal of Robotics Research, 15(5), 1996.

[7] Burgard, W. A. B. Cremers, D. Fox, D. Hahnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. "*Experiences with an interactive museum tour-guide robot"*. Artificial Intelligence 114:3-55, 1999.

[8] Pineau, J., M. Montemerlo, N. Roy, S. Thrun, and M. Pollak. "*Towards robotic assistants in nursing homes: challenges and results"*. Robotics and Autonomous Systems 42:271-281., 2003.

[9] J. J. Leonard and H. F. Durrant-Whyte. "*Directed Sonar Sensing for Mobile Robot Navigation"*. Kluwer Academic Publishers, Boston, MA, 1992.

[10] G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba. "*An experimental and theoretical investigation into simultaneous localization and map building (SLAM)"*. Lecture Notes in Control and Information Sciences: Experimental Robotics VI, Springer, 2000.

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

[12] S. Thrun., D. Fox, and W. Burgard. "*Monte Carlo localization with mixture proposal distributions".* Proceedings of the AAAI National Conference on Artificial Intelligence, AAAI, Austin, TX, 2000.

[13] W. Burgard, D. Fox, D. Hennig, and T. Schmidt. "*Estimating the absolute position of a mobile robot using position probability grids"*. In Proc. Of the Fourteenth National Conference on Artificial Intelligence (AAAI-96), pp. 896-901, 1996.

[14] D. Fox, W. Burgard, and S. Thrun. "*Active Markov localization for mobile robots"*. Robotics and Autonomous Systems, 25(3-4): 195-207, 1988.

[15] N. Gordon, D. Salmond, and A. F. Smith. "*Novel Approach to Non Linear and Non-Gaussian Bayesian State Estimation"*. IEE Proceedings-F, Volume 140, pp. 107-113, 1993.

[16] B. Ripley. *Stochastic Simulation*. Wiley, New York, 1987.

[17] J. S. Liu and R. Chen. "*Sequential Monte Carlo Methods for Dynamical Systems"*. Journal of the American Statistical Association, 1998, Volume 93, pp. 1032 -1044.

[18] Siegwart, R., Nourbakhsh, I., Scaramuzza, D., *Introduction to Autonomous Mobile Robots*. 2nd edition. Cambridge, MA, MIT Press 2011.

[19] Thrun, S., Burgard, W., Fox, D., *Probabilistic Robotics*. Cambridge, MA, MIT Press 2005.

[20] S. Thrun. *"A probabilistic online mapping algorithm for teams of mobile robot"* International Journal of Robotics Research, 20(5):335-363, 2001.

[21] A. Doucet. *"On sequential simulation-based methods for Bayesian filtering"*. Technical report, Signal Processing Group, Dept. of Engineering, University of Cambridge, 1998.

[22] G. Dissanayake, H. Durrant-Whyte, and T. Bailey. "*A computationally efficient solution to the simultaneous localization and map building (SLAM) problem".* In ICRA 2000 Workshop on Mobile Robot Navigation and Mapping, 2000.

[23] H. P. Moravec. "*Sensor fusion in certainty grids for mobile robots"*. AI Magazine, pages 61-74, summer 1988.

[24] D. Hahnel, W. Burgard, D. Fox, and S, Thrun. *"An efficient FastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements"*. In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pages 206-211, Las Vegas, NV, USA, 2003.

[25] G. Grisetti, C. Stachniss, W. Burgard. *"Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling".* In Proc. Of the IEEE Int. Conference on Robotics & Automation (ICRA), pages 2443-2448, Barcelona, Spain, 2005.

[26] Latombe, J. C., *Robot Motion Planning*. Norwood, MA, Kluwer Academic, 1991.

[27] Nilsson, N. J., *Principles of Artificial Intelligence*. Tioga Publishing Company, 1980.

[28] Khatib, O., "*Real-Time obstacle avoidance for manipulators and mobile robots"*. International Journal of Robotics Research 5, no. 1, 1986.

[29] I. Ulrich, J. Borenstein. *"VFH*: Local obstacle avoidance with look-ahead verification".* Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, May, 2000.

[30] D. Fox, W. Burgard, S. Thrun. *"The dynamic window approach to collision avoidance"*. IEEE Robotics and Automation Magazine 4: 23-33, 1997.

[31] K. Ogata*. Modern Control Engineering*. Prentice Hall, 5th Edition, 2009.

[32] R. C. Dorf, R. Bishop. *Modern Control Systems*. Addison Wesley.

[33] I. Calle Flores. *Modelamiento y control de sistemas físicos usando el enfoque port-Hamiltoniano*. Tesis de pregrado. Facultad de Ingeniería Mecánica - Universidad Nacional de Ingeniería, 2010.

[34] A. Rojas Moreno. *Control no lineal multivariable – Aplicaciones en tiempo real*. Editorial EDUNI, 2012.

[35] M. Montemerlo, N. Roy, S. Thrun, D. Hahnel, C. Stachniss, and J. Glover. *"CARMEN – the Carnegie Mellon robot navigation toolkit"*, 2002.

[36] A. Ng, S. Gould, M. Quigley, A. Saxena, E. Berger. *"STAIR: Hardware and Software Architecture"*. AAAI 2007 Robotics Workshop, 2007.

[37] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, A. Ng. *"ROS: An open-source Robot Operating System"*. Open-source software workshop of the International Conference on Robotics and Automation (ICRA), 2009.

APÉNDICE A
R2D2-00: EL ROBOT AUTÓNOMO

En este apéndice se presenta de manera resumida los diversos componentes (sensores, actuadores, componentes electrónicos, componentes, mecánicos, etc.) que conforman el robot móvil (bautizado como R2D2-00) desarrollado en la presente tesis.

Tal como se mencionó en el Capítulo 1, los módulos de entrenamiento robóticos para realizar investigación en los temas de navegación autónoma (localización, mapeo, planeamiento) están usualmente en los miles de dólares. Un ejemplo claro son los robots móviles de la empresa Pionner (los robots móviles más usados en la comunidad de la robótica móvil) que tienen un costo superior a los USD 20,000.00. El módulo robótico desarrollado en el presente trabajo tiene un costo (solo considerando la parte de hardware) aproximado de USD 2,500.00. Este bajo costo, combinado con el software de navegación amigable al usuario, hace que este módulo robótico sea el ideal para realizar investigación en el tema de navegación autónoma de robots móviles.

A.1 Sistema de Locomoción

Tal como se mencionó en el capítulo 3, el robot móvil “R2D2-00” implementado en la presente tesis es del tipo diferencial. Esto significa que el movimiento del robot está dado por dos motores alineados a lo largo de un mismo eje. Por otro lado, con el fin de garantizar la estabilidad al sistema, el robot cuenta con dos ruedas locas en la parte posterior. Este sistema de locomoción se muestra en la Figura A.1.

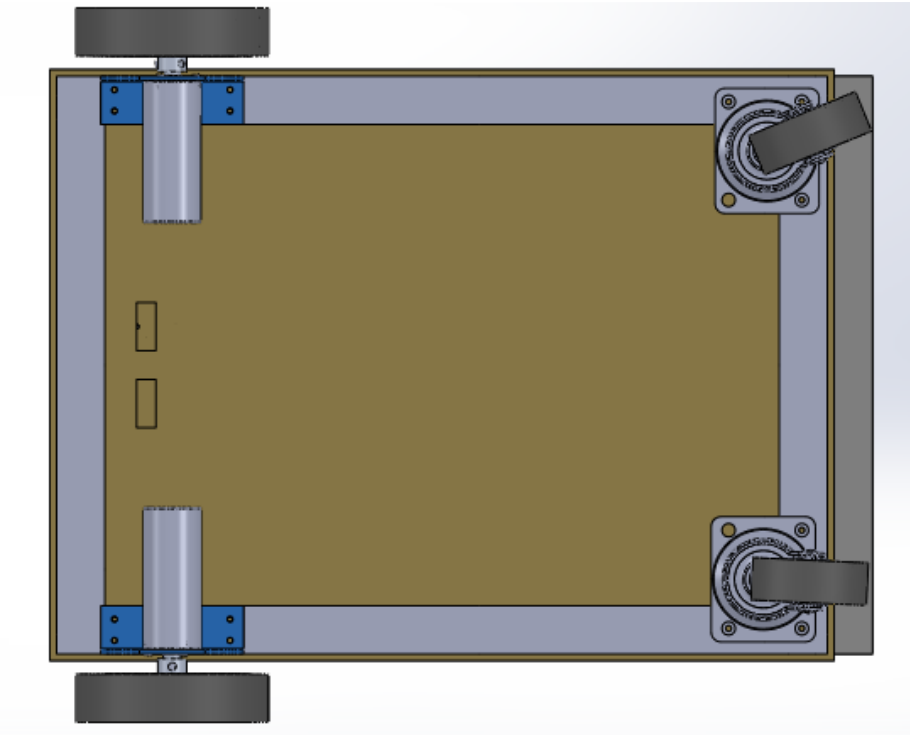


Figura A.1. Sistema de locomoción del robot R2D2-00 (Fuente: Elaboración propia)

Los motores usados son del tipo continuo de 12V, los cuales tienen incorporados encoders y cajas de reducción (Figura A.2). Estos motores son ideales para su aplicación en los robots móviles, ya que tienen incorporados los sistemas anteriormente mencionados, y además porque sus especificaciones técnicas (Tabla A.1) los hace adecuados para su uso en los robos móviles.

Figura A.2. Motores DC del robot R2D2-00 (Fuente: Elaboración propia)

Tabla A.1. Especificaciones técnicas de los motores DC del robot R2D2-00 (Fuente: http://www.robot-electronics.co.uk/)

Voltaje nominal	12V
Torque nominal	1.5Kg/cm
Velocidad nominal	170rpm
Corriente nominal	530mA
Velocidad sin carga	216rpm
Corriente sin carga	150mA
Potencia nominal	4.22W
Resolución del encoder	360

Las ruedas usadas (Figura A.3) tienen un diámetro de 100mm y un ancho de 20mm. Estas ruedas se conectan fácilmente a los motores DC anteriormente mencionados, y al estar hechas de goma, presentan una buena fricción.

Figura A.3. Ruedas del robot R2D2-00 (Fuente: http://www.robot-electronics.co.uk/)

A.2 Sistema Electrónico

El sistema electrónico del robot R2D2-00 consta de dos tarjetas. La primera tarjeta se encarga de suministrar potencia a los motores DC (Figura A.4). Esta tarjeta recibe la energía de las baterías del robot, y según las señales de control enviadas por la tarjeta de control, envía las señales PWM a los motores.

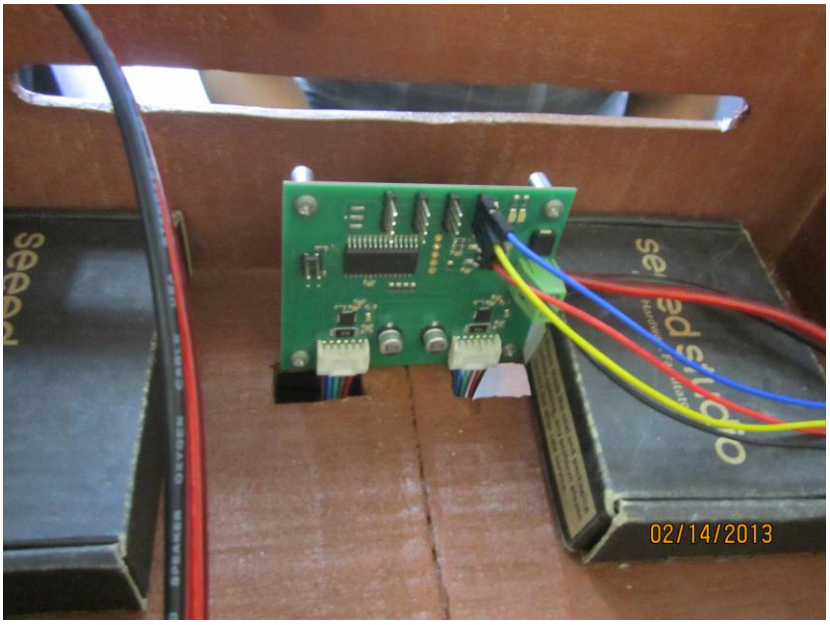


Figura A.4. Tarjeta de potencia de los motores DC (Fuente: Elaboración propia)

La segunda tarjeta (Figura A.5) se encarga del control de los motores a través de la tarjeta de potencia de los motores DC, y de la recepción de las señales de control enviadas desde la laptop del Robot. El procesador principal de esta tarjeta es un DSP TMS320F28069 de la empresa Texas Instrument, y la comunicación con la PC es a través del puerto USB.

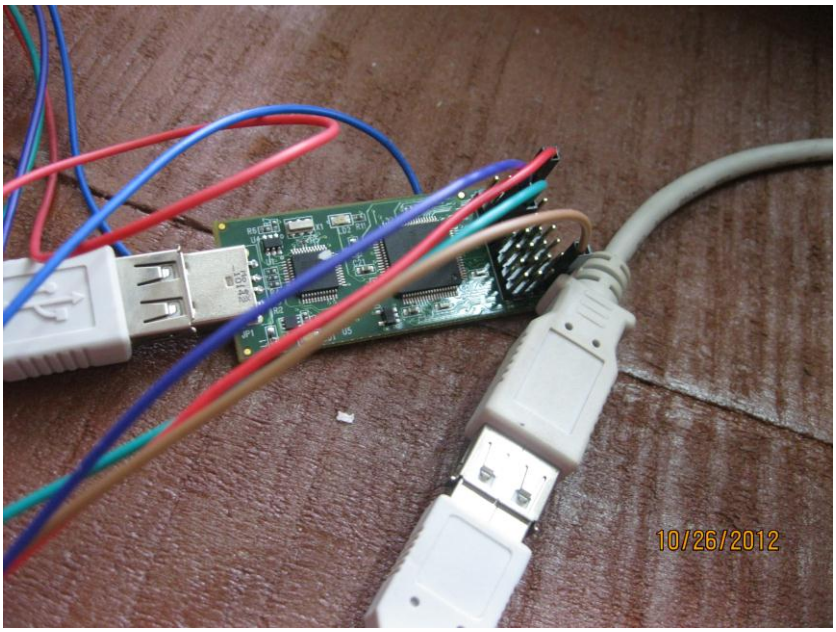


Figura A.5. Tarjeta de control (Fuente: Elaboración propia)

A.3 Sensor Laser Hokuyo URG 04LX-UG01

El principal sensor del robot “R2D2-00” es un sensor laser con el cual se miden las distancias hacia los obstáculos más cercanos. Específicamente se usó el sensor URG 04LX – UG01 de la marca Hokuyo (Figura A.6)

Figura A.6. Sensor laser Hokuyo URG 04LX – UG01 (Fuente: http://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx_ug01.html)

Entre las principales características de este sensor (ver Tabla A.2) podemos resaltar las siguientes: Requiere una fuente de alimentación de 5V DC la cual es proporcionada directamente por el puerto USB de la Laptop, tiene una frecuencia de muestreo de 100mseg/scan, y tiene una resolución de 1mm. Una lectura típica de este sensor se muestra en la Figura A.7.

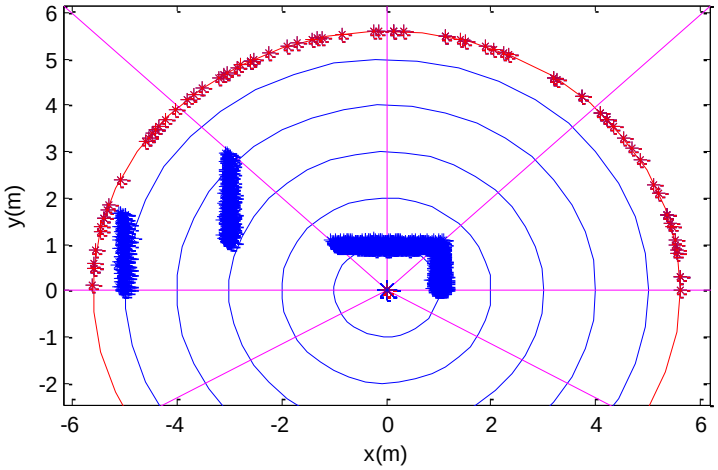


Figura A.7. Lectura típica del sensor laser Hokuyo URG 04LX – UG01 (Fuente: Elaboración propia)

Tabla A.2. Especificaciones técnicas del sensor Hokuyo URG 04LX – UG01 (Fuente: http://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx_ug01.html)

Product Name	Scanning Laser Range Finder
Model	URG-04LX-UG01
Light source	Semiconductor laser diode (λ=785nm), Laser safety Class 1 (21 CFR part 1040.10 and 1040.11) Laser power: 0.8mW or less (Class 1 compliant by scanning)
Power source	5V DC ±5% (Supplied by USB bus power)
Current consumption	500mA or less (Rush current 800mA)
Detection distance and standard object	Accuracy:60-4,095mm (white paper 70mm x 70mm or bigger)* Detectable range: 20-5,600mm
Accuracy	Described in the data sheet attached to each unit Guaranteed accuracy: 0.06-1m: ±30mm* , 1-4m: 3% of the detected distance* (Standard object: white paper 70mm x 70mm)
Resolution	1 mm
Scan Angle	240°
Angular Resolution	Approx.0.36° (360°/1024)
Scan Time	100msec/scan
Interface	USB Version 2.0 FS mode (12Mbps) SCIP2.0
Ambient (Temperature/Humidity)	-10 ~ 50°C / 85% or less (without dew and frost)
Preservation temperature	-25 ~ 75°C
Ambient Light Resistance	10000Lx or less (Sunlight)
Vibration Resistance	Double amplitude 1.5mm 10 ~ 55Hz, 2 hours each in X, Y and Z direction, and 98m/s^2 55Hz ~ 150Hz in 2 minutes sweep, 1 hours each in X, Y and Z direction
Impact Resistance	196 m/s^2, 10 times each in X, Y and Z direction
Protective Structure	Optics : IP64 Case : IP40
Insulation Resistance	10MΩ for DC 500Vmegger
Weight	Approx. 160 g
Case	Polycarbonate
External dimension (W×D×H)	50×50×70mm (Reference design sheet No. C-40-3362)

A.4 R2D2-00: El Robot Autónomo

El robot R2D2-00 (Figura A.8) resulta de la combinación e instalación de todos los componentes anteriormente descritos en un chasis de madera de forma rectangular. Este chasis además contiene las baterías que se encargan de alimentar los motores DC, y los diversos componentes electrónicos del robot. Además, nótese que el robot tiene un sistema de visión estéreo mediante el cual es capaz de tomar imágenes. En posteriores trabajos se espera agregar un manipulador robótico con el fin que nuestro robot sea capaz de manipular objetos.

Figura A.8. R2D2-00: El robot autónomo (Fuente: Elaboración propia)