# Trabajo Práctico Final

99319 Cuenca Matilde mcuenca@fi.uba.ar
99404 Guarino Paulo pguarino@fi.uba.ar

7 de septiembre de 2020

## 1. Introducción

En el siguiente informe se describe la implementación de los algoritmos implementados para la resolución de los problemas de patrullaje y exploración del robot móvil tipo Roomba, contando con datos de odometría y con lecturas de un lidar. En el problema de patrullaje, el robot es "despertado" en algún punto de un ambiente con mapa conocido y debe realizar un trayecto que visite tres puntos de interés indicados. En el escenario de exploración, el robot comienza en un ambiente desconocido explorar en un tiempo determinado, con el objetivo de generar un mapa del entorno. Para ambos problemas se desarrolló un algoritmo en Matlab utilizando los conocimientos adquiridos en el curso y estudiando la bibliografía recomendada del mismo.

Los algoritmos fueron validados mediante simulaciones siguiendo las indicaciones provistas por la cátedra para permitir una transición directa entre el simulador y la plataforma robótica real. Se hicieron algunas modificaciones al simulador original para facilitar la escritura del código, como agregarle a la clase que maneja el lidar el ángulo del mismo respecto del ángulo de dirección del robot.

## 2. Consideraciones de diseño

En primer lugar, debemos tener en cuenta algunas consideraciones para realizar los desafíos. Sabemos que si el robot detecta una colisión, su sistema interno deshabilitará sus motores y la misión habrá fracasado. Por lo tanto, debe implementar una función que permita prevenir choques. Se programó una rutina que, en caso que se detecte un obstáculo en la dirección actual, se anulen las velocidades de comando y se retome la misión con una nueva dirección. Para ello, miramos las mediciones en un intervalo de $[-\pi/3, pi/3]$ con respecto al ángulo cero del robot, como observamos en la figura 2.1, y teniendo en cuenta la pose del lidar con respecto al robot. Si la medición mínima supera cierto umbral, se definen las velocidad de movimiento nulas.

Otra consideración a tener en cuenta para ambos desafíos es cómo definir las velocidades de comando del robot en función de la trayectoria que queramos realizar. Como no queremos desviarnos mucho de la trayectoria, evaluamos la distancia a recorrer en una misma dirección. Si esta distancia es grande, podemos permitirnos una velocidad lineal grande. Si la distancia es corta, es conveniente reducir la velocidad para prepararnos para girar o eventualmente llegar al objetivo de forma más suave. Finalmente, si detectamos que debemos girar, se determina la velocidad de giro de manera proporcional al ángulo. Si por algún motivo, nos desviamos mucho del camino deseado, se vuelve a calcular para evitar posibles colisiones.

Además, al definir las velocidades de comando debemos tener en cuenta los límites de velocidad recomendados para mantener la integridad física del sistema y para minimizar los errores de medición (debido a vibraciones, etc). Intentamos mantenerlas dentro del intervalo recomendado, aunque se superó ligeramente la velocidad máxima lineal en caso de una distancia grande en una misma dirección.

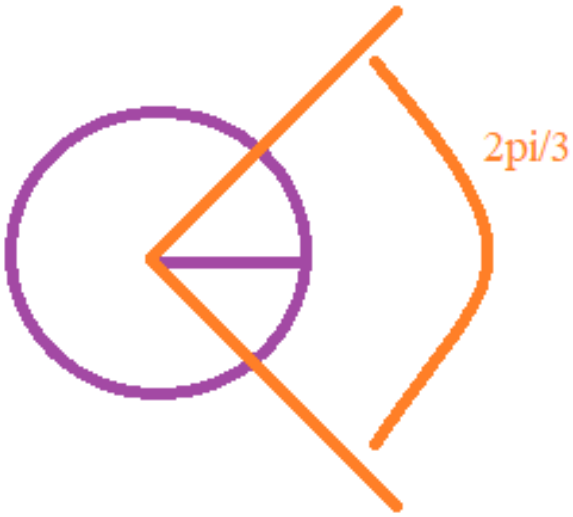


**Figura 2.1:** Protección contra choques

## 3. Patrullaje

El robot comienza en un entorno conocido y debe visitar los tres puntos definidos por las siguientes coordenadas:

| | |
|---|---|
| Punto A | 2,9 3,7 |
| Punto B | 1,3 3,7 |
| Punto C | 1,5 1,9 |

Podemos separar este desafío en dos sub-problemas, localización y planeamiento.

Para resolver el problema de localización, se decidió utilizar el algoritmo de filtro de partículas. Luego de una primera inicialización, el primer paso en cada iteración de este algoritmo consiste en generar partículas usando el modelo de movimiento, donde cada partícula es una potencial pose del robot, y en conjunto todas representan la distribución de la pose. Luego, se calcula el peso $w$ de cada partícula según el likelihood de las observaciones. Finalmente se debe remuestrear, es decir reemplazar partículas pocos probables por otras más probables. Para esto se toman nuevas partículas con una probabilidad proporcional al likelihood de las observaciones.

La dificultad se dio sobretodo en adaptar el cálculo de los pesos $w$, que deben calcularse a partir de las mediciones, la pose de la partícula y todas las celdas ocupadas, a un mapa de grilla, cuando en trabajos anteriores se había utilizado el filtro de partículas pero con un conjunto de landmarks. Como el mapa fue provisto de antemano, se decidió armar una matriz de forma tal que cada elemento $i, j$ de la matriz contenga la distancia de la celda $i, j$ a la celda ocupada más cercana.

De esta forma, simplemente evaluamos la matriz en la celda donde suponemos el obstáculo medido (dada tal partícula, para todas las mediciones) y finalmente calculamos $p(z_t|x_t, m)$ para obtener el likelihood que corresponde al peso de cada partícula.

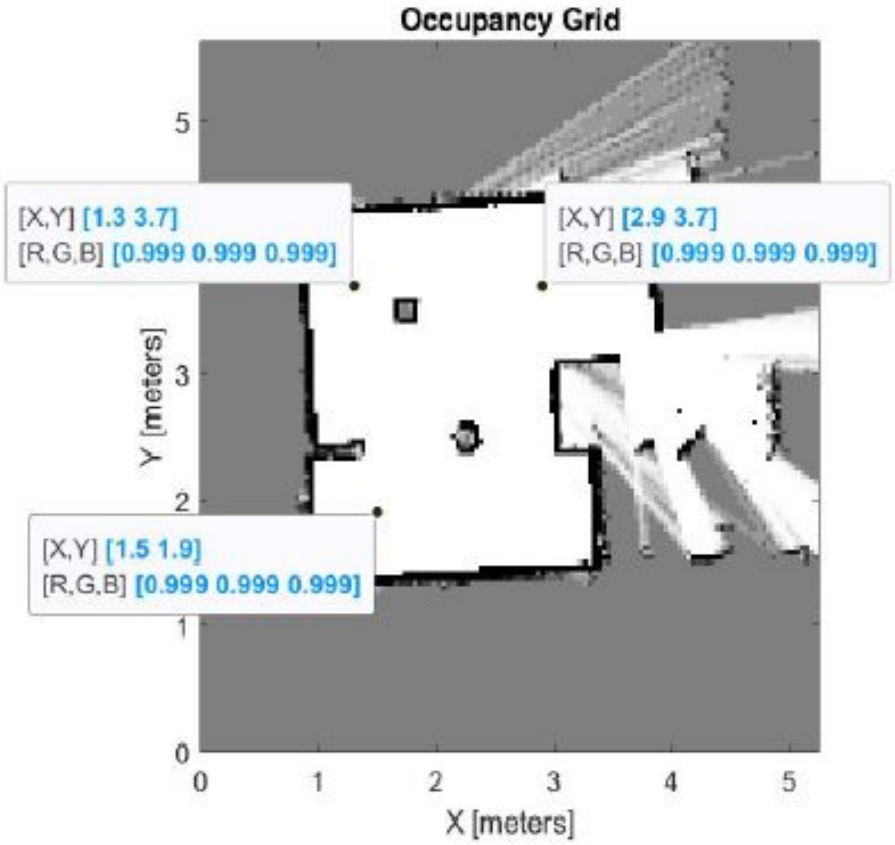


**Figura 3.1:** Mapa y coordenadas por las que debe pasar el robot

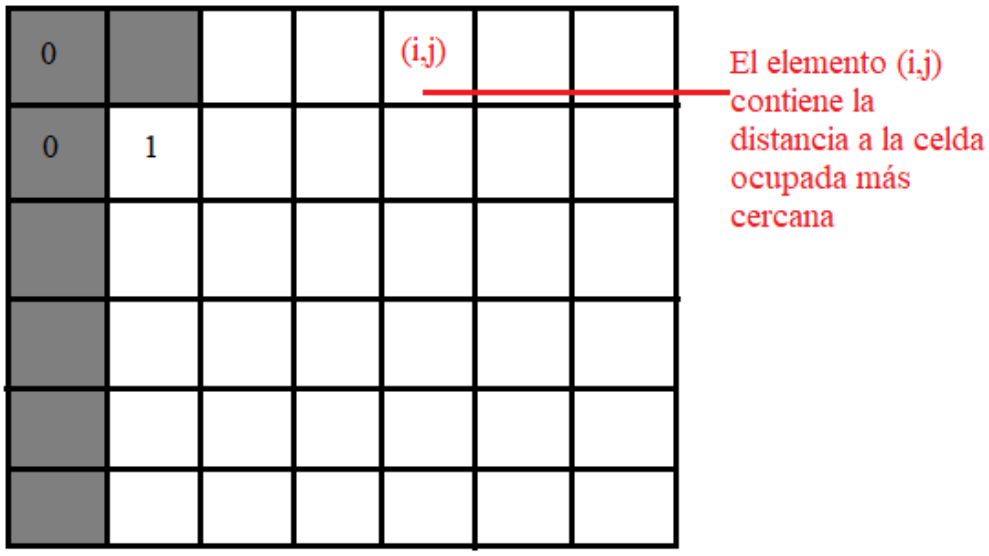


**Figura 3.2:** Matriz de distancias implementada

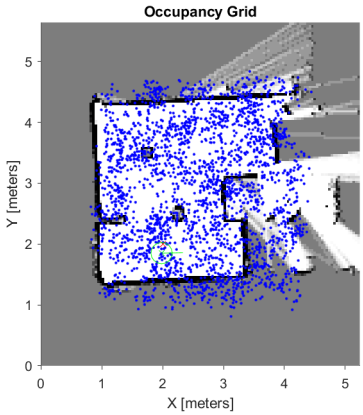


**Figura 3.3:** Partículas en el instante inicial

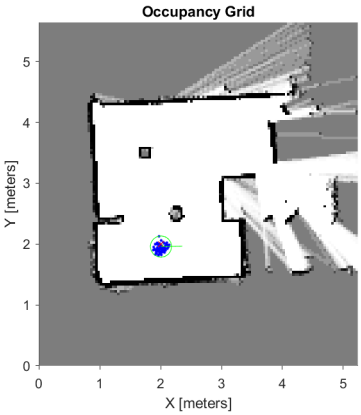


**Figura 3.4:** Partículas luego de 2 iteraciones

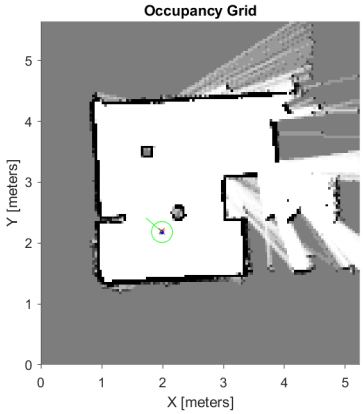


**Figura 3.5:** Partículas luego de 120 iteraciones

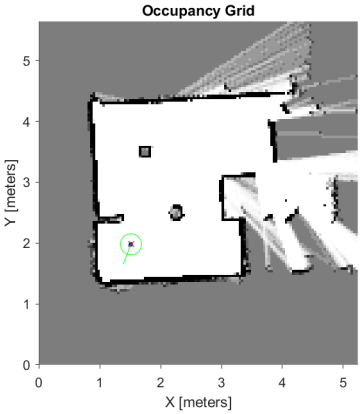


**Figura 3.6:** Partículas en el instante final

**Figura 3.7:** Evolución de las partículas

En un principio trabajábamos con 250 partículas, ya que permitía que el programa pueda seguir la tasa de muestreo de los sensores del robot, y seguían con mucha precisión y con baja varianza muestral la pose del robot simulado. Sin embargo, la localización inicial fallaba en distintos intentos, y dependía completamente de cómo se inicializaban las partículas, que estaban distribuidas con pose y dirección aleatorias según una distribución uniforme sobre la zona libre del mapa. Una vez que las partículas convergían a un punto, por más que este estuviese extremadamente alejado de la pose del robot, las partículas nunca lograban localizar al robot, a menos que el ruido de la odometría fuera tal que algunas llegasen a moverse hacia el robot. La razón por la que esto pasaba es por la naturaleza misma del campo de likelihood: no tienen en cuenta el camino del rayo del lidar, sino únicamente el punto donde termina. Esto hace que partículas que están en una zona del mapa desconocida y a una distancia a la pared mucho más cercana que la del robot, tengan un peso no despreciable frente a partículas cercanas al robot, en especial si estas últimas tienen una dirección muy distinta a la del robot. Por lo tanto, para solucionar este problema que llevaba a que fuese imposible recuperar una buena estimación del robot, las partículas inicialmente son 2500, 10 veces más que en el otro caso, y se les agrega ruido en los primeros pasos para asegurarse que se distribuyan uniformemente y así obtener una primera estimación correcta. Luego se descartan partículas y se prosigue el resto del programa con 250.

En la figura 3.7, se muestra cómo van evolucionando las partículas con el tiempo. La figura 3.3 muestra las 2500 partículas en posición aleatoria sobre toda la superficie del mapa donde es posible encontrar al robot. Luego de dos iteraciones, podemos ver que las partículas ya convergieron a la pose del robot, es decir, la estimación de la pose se encuentra prácticamente en el centro del robot. Se graficaron en dos momentos más para mostrar que la localización se mantiene con el tiempo. Para observar este resultado con más detalle, ver el vídeo adjuntado particles.mp4.

Luego, para resolver el problema de planeamiento, se utilizó el algoritmo $A^*$ junto a consideraciones relacionadas a las dimensiones del robot a la hora de armar la trayectoria, además de la heurística que prioriza las distancias más cortas. Para tomar en cuenta el tamaño del robot, debimos asignarle costo muy alto a aquellas celdas que se encuentran a distancia de otras celdas ocupadas menor al radio del robot, ya que no entraría y chocaría. Además, para lograr un camino más suave y menos susceptible a errores aumentamos el costo de manera proporcional al ángulo de rotación. Esto además nos facilitó el control del robot, ya que estamos priorizando caminos derechos. En la figura 3.11 podemos ver los distintos planeamientos obtenidos, para ir desde la pose inicial al primer objetivo, del primer objetivo al segundo, y del segundo al tercero. Ninguno se acerca demasiado a las paredes, y prioriza caminos derechos.

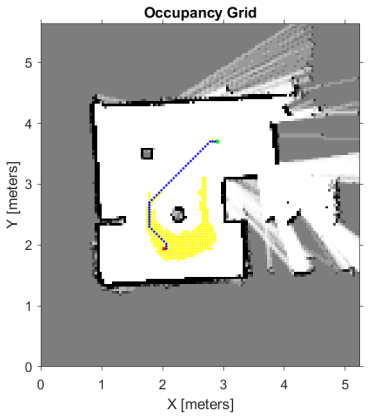


**Figura 3.8:** Planeamiento obtenido hasta goal 1

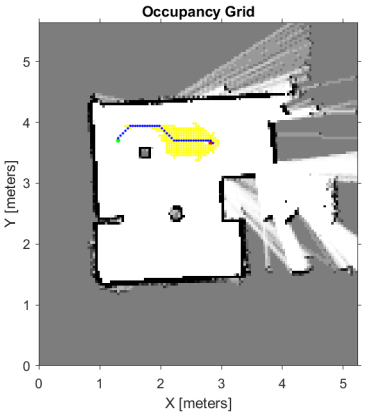


**Figura 3.9:** Planeamiento obtenido desde goal 1 hasta goal 2

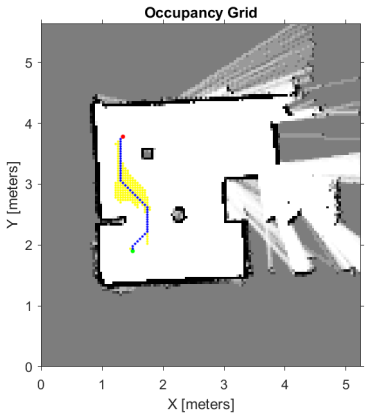


**Figura 3.10:** Planeamiento obtenido desde goal 2 hasta goal 3

**Figura 3.11:** Planeamientos para llegar a los distintos objetivos

En la figura 3.12 podemos observar la trayectoria realizada por el robot. Se ve que, efectivamente, esta trayectoria se encuentra sobre los puntos a los que se deseaba ir. Para una observación más completa de los resultados obtenidos en este problema, ver el vídeo adjuntado bajo el nombre trayectoria.mp4.

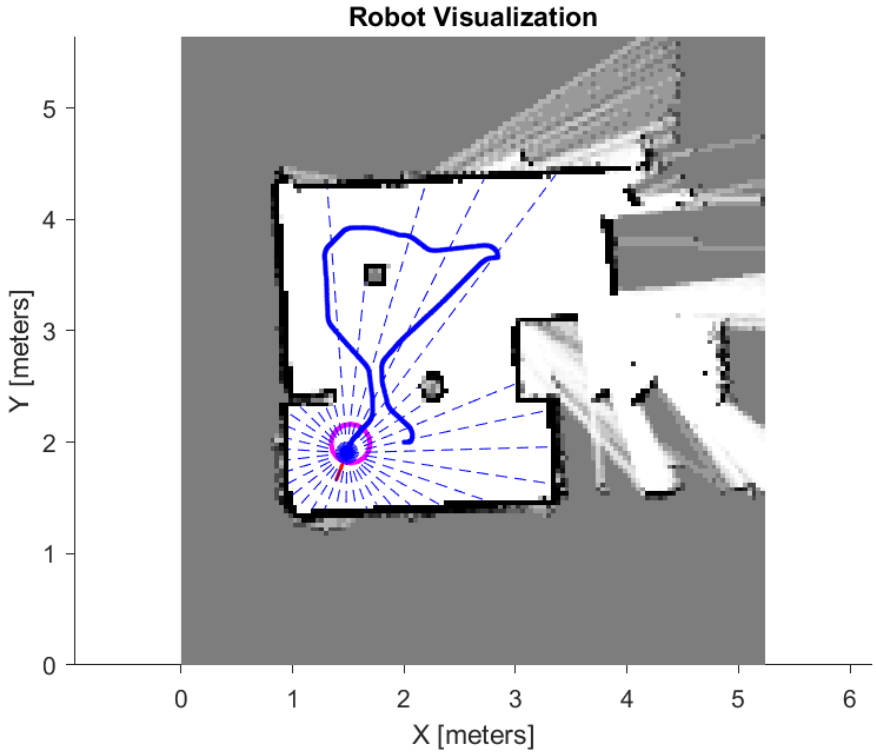


**Figura 3.12:** Recorrido realizado

# 4. Exploración

En este desafío, el robot se despierta en un entorno desconocido y debe generar un mapa de ocupación del mismo, explorando el entorno. En este caso nos enfrentamos a un problema más complejo que al del desafío anterior ya que debemos localizarnos y planear al igual que antes pero sin conocer el mapa. Para resolver la localización y mapeo, se decidió utilizar el algoritmo de FastSLAM ya que considero apropiado para resolver el problema y se contaba con una implementación inicial hecha una trabajo anterior.

Al igual que el algoritmo del ejercicio anterior, primer paso consiste en la generación de partículas. Nuevamente, cada una de estas partículas representa una posible pose del robot, pero además cada partícula tiene asociado un mapa que actualizamos con el nuevo conjunto de mediciones. Luego, calculamos los pesos $w$ de cada partícula. Si bien no podemos implementar exactamente el mapa de likelihood como en el desafío anterior ya que el mapa es desconocido, la idea es similar. Se define un vector con las celdas ocupadas del mapa de cada partícula y se busca la distancia desde el punto donde se supone que se encuentra el obstáculo medido (dada tal partícula, para todas las mediciones), hasta la celda ocupada más cercana. Con todas esas distancias, podemos fácilmente calcular $p(z_t|x_t)$. Finalmente, se realiza un remuestreo donde se reemplazan a las muestras poco probables por otras más probables, idéntico al del ejercicio anterior.

Durante las pruebas de simulación, se encontró que apenas se despertaba el robot este recorría dando muchos giros sobre sí mismo, tardaba mucho tiempo en localizarse, y generaba un mapa muy difuso. Encontramos que esto radica en el hecho de que hay más incerteza en la odometría cuando el robot realiza giros. Si se combina esto con la gran incerteza que tiene el robot en su posición apenas es despertado, lleva a que la impresiones iniciales del mapa sean muy pobres, lo cual empeora la localización, y gener un efecto en cadena que resulta en el robot completamente perdido y a veces incluso sin posibilidad de planear caminos para la exploración. Para intentar remediar esto, se programo el robot para que prefiera realizar caminos rectos apenas es despertado si fuera posible. Esto permite tener un "buen comienzo" (un decente mapa inicial con una buena localización de baja incerteza) en la tarea de exploración.

Para decidir hacia donde queremos explorar, además de considerar costos similares a los del primer ejercicio, planteamos la matriz de entropía del mapa a partir de la de probabilidad de ocupación $p$,

$$H_{xy} = p \cdot log(p) + (1 - p) \cdot log(1 - p)\,. \tag{4.1}$$

La entropía nos permite saber dónde vamos a poder aprender más sobre el mapa. Esta es máxima cuando la probabilidad de la celda es $p = 0{,}5$, es decir cuando no tenemos información sobre la celda. Luego, planteamos la matriz de entropía binaria $H_b$, cuyo elemento $i, j$ vale 1 si no tenemos información de la celda, o sea la probabilidad de esta en un intervalo pequeño al rededor de $p = 0{,}5$. Cada vez, por lo tanto, planeamos hasta el primer punto no explorado que encontremos, lo que equivale a $H_b(i, j) = 1$.

Se tomaron algunas consideraciones particulares para la generación del mapa dado los resultados de algunas simulaciones. Por ejemplo, para evitar que el giro de una partícula por unos instantes lleve a que la probabilidad de ocupación de un punto vacío aumente, y que luego en otro instante pueda ser erróneamente tomado como ocupado, en cada iteración del mapa, los puntos de los cuales no se conoce la suficiente información pierden información si no son actualizados.

Debido a la mayor intensidad computacional del algoritmo, se utilizaron solo 75 partículas, aunque esto no generó mayores inconvenientes ya que en este caso no se necesita encontrar la ubicación inicial del robot respecto al mapa, simplemente se inicializan todas las partículas en un mismo punto.

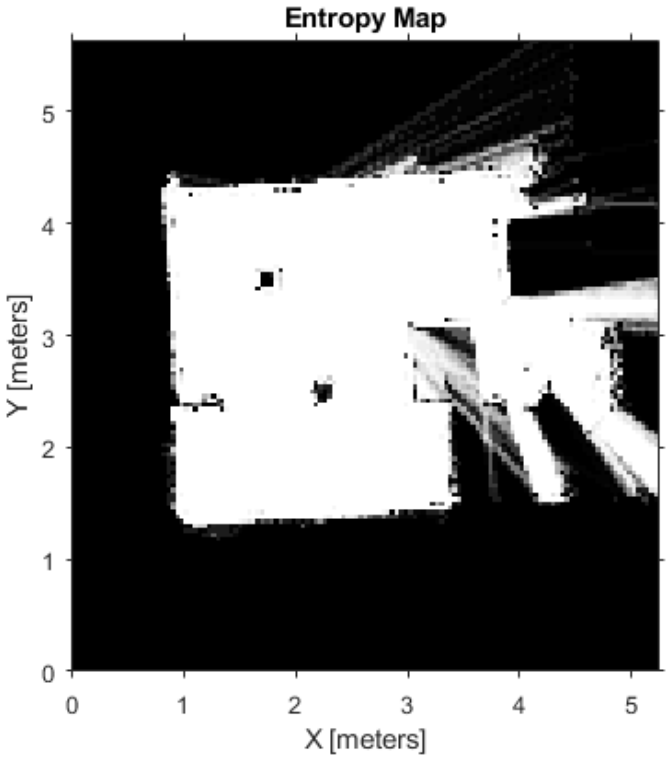


**Figura 4.1:** Matriz de entropía

Simulando durante 3 minutos, para replicar las condiciones de la prueba, se obtuvo el mapa que se muestra en la figura 4.2. Para observar los resultados obtenidos en este problema, ver el vídeo adjuntado mapping.mp4.

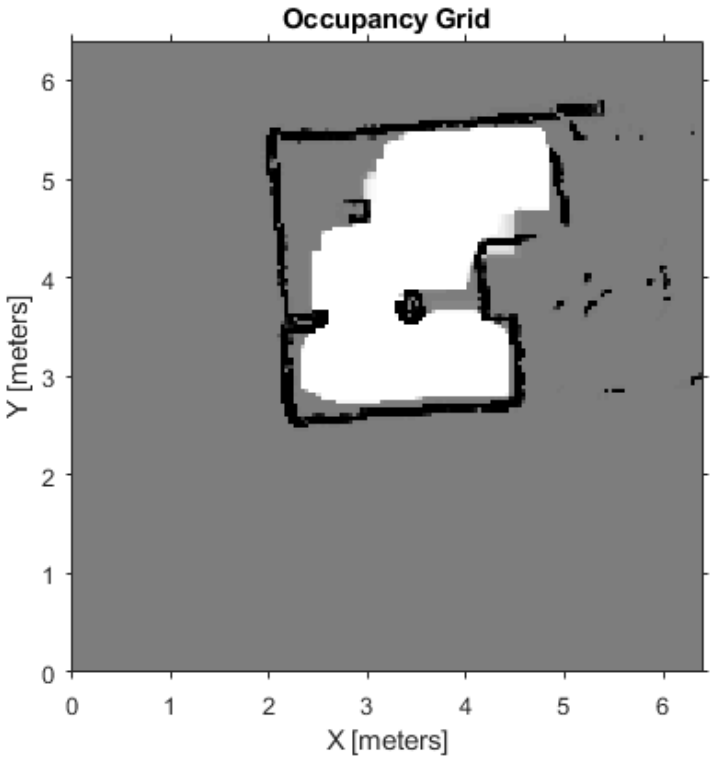


**Figura 4.2:** Mapa obtenido

# 5. Conclusiones

Se lograron diseñar, implementar y validar mediante simulación algoritmos que permitieran resolver los problemas de patrullaje en un ambiente conocido y exploración de un entorno desconocido para el robot Roomba dado. Para lograr esto, se utilizaron los conocimientos adquiridos a lo largo de toda la materia, como el uso de filtro de partículas para resolver la localización y su uso en FastSLAM. Todos estos algoritmos se tuvieron que adaptar a los problemas específicos del trabajo, siendo de particularidad dificultad el SLAM e incluso la localización misma con un mapa de grilla. Se adaptaron también las funciones de costo, y se tuvieron que hacer varias iteraciones de cada función para lograr la eficiencia necesaria para mantener las iteraciones a menor o igual tiempo que el período de muestreo. Además se detallaron las consideraciones de diseño de los algoritmos y se presentaron los resultados obtenidos con las simulaciones.