
# Universidad Carlos III de Madrid

## Máster Universitario de Robótica y Automatización
## Planificación de tareas y movimientos de robots

# ??

*Autores:*
David Estévez Fernández
Enrique Fernández Rodicio
Javier Isabel Hernández

*Profesores:*
Alberto Jardón Huete
Ramón Barber Castaño

12 de mayo de 2015

# Índice general

# Capítulo 1

# Introducción

Introduccion

# Capítulo 2

# Explicación artículos

**Introducción de esta parte si hiciese falta**

## 2.1. Artículo principal

**Aquí se habla del artículo principal**

El artículo principal, *A Note on "Solving the Find-Path Problem by Good Representation of Free Space"*, es una crítica a un artículo de Brooks, llamado *Solving the Find-Path Problem by Good Representation of Free Space*. En esta sección se llevará a cabo una introducción al artículo original de Brooks, explicando los puntos principales. Posteriormente, se explicarán las críticas que el artículo principal realiza sobre el artículo de Brooks.

### 2.1.1. Solving the Find-Path Problem by Good Representation of Free Space

El método que Brooks expone en este artículo se basa en la representación del espacio libre como conos generalizados. Un cono geralizado es un cono truncado que tiene una base y una tapa formada por cilindros (figura 2.1).

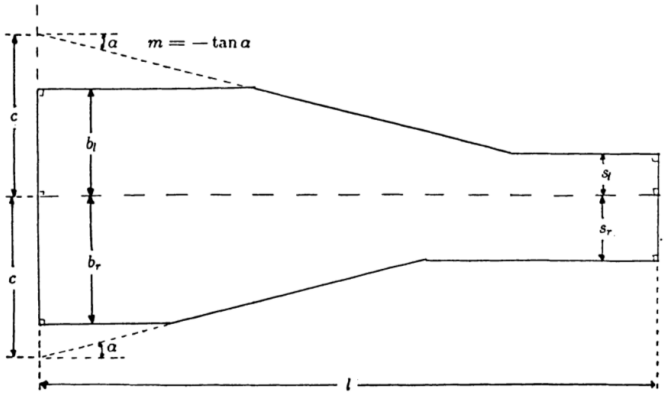



Figura 2.1: Cono generalizado en un plano 2D

El algoritmo propuesto es el siguiente. Primero, el mapa del entorno es analizado, y se calculan los conos generalizados que proporcionarán las trajectorias sin colisión. Este paso puede ser realizado en un tiempo $O(n^3)$. A continuación se procesan estos conos por parejas, comprobando para cuáles de ellos existe intersección entre sus columnas. La columna del cono generalizado es el eje de revolución del mismo. Hasta este punto, el algoritmo es independiente del objeto que se deseee mover por el entorno y, por tanto, sólo es necesario ejecutar esta parte del algoritmo una vez por cada entorno (suponiendo que el entorno es estático).

Cada intersección debe ser comprobada y anotada con el rango angular que describe las orientaciones del objeto en las que está garantizado que ese objeto quede contenido dentro del cono generalizado. De esta forma, se especifican las orientaciones o rotaciones permitidas para el objeto en cada una de las intersecciones.

Finalmente, se genera el grafo que contiene todas las rutas posibles, comprobando que, para cada intersección, el ángulo requerido para viajar de una intersección a la siguiente está contenido dentro del rango calculado en el paso anterior. Este grafo nos daría todas los caminos libres de colisión que existen en el entorno, asegurándonos que si viajamos desde un nodo A a un nodo B de este grafo con nuestro objeto no existirá colisión con ningún obstáculo. El hecho de usar las columnas de los conos generalizados nos grantiza también que el objeto recorrerá el camino lo más alejado posible de los obstáculos.

Una vez tenemos el grafo, se puede realizar la búsqueda del camino más corto de un nodo a otro del grafo usando un algoritmo de búsqueda en grafos tal como el $A^*$. El camino resultante será un conjunto de tramos rectos en los que el objeto se desplará usando translaciones puras y un conjunto de nodos en los que el objeto se reorientará mediante rotaciones puras para dirigirse al siguiente nodo.

El método para generar los conos generalizados está ampliamente detallado en el artículo, y se explica a continuación. En primer lugar, se define el lado libre de una arista del obstáculo como el lado hacia el que apunta la normal a esa superficie (la parte de fuera del polígono). Se considera que dos aristas forman parte de un cono generalizado si se cumplen las siguientes condiciones: al menos un vértice de la arista debe encontrarse en el lado libre de la otra arista y el producto escalar de las normales que apuntan hacia afuera debe ser negativo. Esta última condición asegura que los dos lados libres de las aristan se encuetren en el mismo lado.

Una vez se han calculado los conos generalizados, se obtienen las columnas de los mismos mediante la bisectriz del espacio libre entre las aristas de los obstáculos y se extiende el cono desde los vértices de forma paralela a su columna. Tras generar cada cono, se deben comprobar colisiones con todos los obstáculos, pues puede suceder que al generar un cono éste tenga un obstáculo dentro y no se encuentre totalmente en el espacio libre. Si esto sucede, la intersección se ha de proyectar sobre la columna del cono para obtener la parte del cono que pertece al espacio libre.

### 2.1.2. A Note on "Solving the Find-Path Problem by Good Representation of Free Space"

Este artículo, escrito en 1997 por Wang, es una crítica al artículo de Brooks. En él, se explica que las condiciones de tranlaciones puras y rotaciones puras que son un requisito indispensable para la navegación usando el algoritmo de Brooks no son válidas para todos los tipos de robot móvil. Por tanto, el algoritmo de representación del espacio libre mediante conos generalizados sólo es aplicable a cierto tipo de robots móviles.

En primer lugar comenta las limitaciones en el caso de robots diferenciales, en cuyo caso las limitaciones sólo se aplican al punto de referencia que se ha de elegir para el sistema de referencia del robot. Para que las condiciones de rotación pura y translación puras se cumplan, el punto seleccionado debe encontrarse en el eje de rotación de las ruedas móviles, en el punto de corte con la mediatriz del eje.

En el caso de los robots de tres ruedas, tipo triciclo (figura 2.2) y los de cuatro, de tipo coche (con ruedas directrices y ruedas fijas), al ser robots no holonómicos, además de la limitación del punto de referencia colocado en el punto medio del eje de las ruedas traseras, se añade una limitación extra para la rotación pura. De esta forma, si el robot posee un rango de movimiento en la rueda directriz menor de $90^o$, el robot no podrá realizar rotaciones puras. Esto se puede apreciar mediante las ecuaciones cinemáticas del robot de tipo triciclo:

$$\dot{x_R} = -v\cos(\alpha)\sin(\theta)$$
$$\dot{y_R} = v\cos(\alpha)\cos(\theta)$$
$$\dot{\theta} = \frac{v}{L}\sin(\alpha)$$


Figura 2.2: Robot tipo triciclo

Para un punto situado en la rueda delantera, donde $\theta$ es el ángulo formado con la horizontal, $\alpha$ es el ángulo de giro de la rueda directriz, $v$ es la velocidad de la rueda directriz y L es la distancia entre el eje trasero y la rueda directriz. Para un punto situado en el punto medio del eje trasero, las ecuaciones cinemáticas se convierten en:

$$\dot{x_R} = -vsin(\alpha + \theta)$$
$$\dot{y_R} = v\cos(\alpha + \theta)$$
$$\dot{\theta} = \frac{v}{L}\sin(\alpha)$$

Con lo que podemos observar que sólo existe rotación pura para un robot de tipo triciclo si la rueda directriz no tiene limitado el rango de giro, y puede ir desde -90º hasta 90º. En todos los demás casos la rotación pura es imposible y, por tanto, no se podría aplicar el algoritmo de Brooks.

## 2.2. Artículo complementario

El articulo complementario que se ha estudiado durante la realización de este proyecto tiene por título *"Quasi-Randomized Path Planning"*, y fue escrito por M. Branicky, S. La Valle, K. Olson y L. Yang.. En el se propone un enfoque distinto a la hora de muestrear el espacio por el que se moverá el robot, pasando a distribuir los nodos de forma cuasi-aleatoria, en vez de hacerse por completo aleatoriamente.

Cuando se trabaja en un espacio de configuraciones con un alto número de variables a considerar, las aproximaciones deterministas clásicas desarrolladas para la planificación de trayectorias dejan de ser aplicables, o son demasiado costosas. Para estos casos, en la década de los 90 se comenzarom a desarrollar enfoques aleatorios. Estos métodos pasaron a ser mas usados que los deterministas por dos motivos:

1. Permitían resolver un problema con una alta multidimensionalidad sin la necesidad de explorar todas las alternativas.
2. Si se ve el problema como un enemigo a vencer, a menudo se puede evitar la derrota aplicando una estrategia aleatoria, cosa que no ocurre con una determinista.

Sin embargo, no es posible conseguir una serie de datos realmente aleatorios, ya que cualquier posible implementación simplemente llevará a una secuencia determinista de números pseudoaleatorios que van a seguir una cierta función de probabilidad. De aquí se sacó la idea de desarrollar un metodo cuasi-aleatorio que permitiese solucionar el problema de la generación de trayectorias de forma más eficaz.

A la hora de generar la secuencia de muestras pseudo-aleatorias con las que va a trabajar el método aleatorio de planificación, se busca el conjunto de puntos que optimicen el valor de dos parámetros, la discrepancia y la dispersión. La discrepancia para un conjunto P formado por N muestras de d-dimensiones en $[0,1]^d$ viene dada por la ecuación 2.1:

$$D_N(P) = \sup_j |\frac{A(J)}{N} - \mu(J)| \tag{2.1}$$

Donde J es cualquier subconjunto n-rectangular perteneciente a $[0,1]^d$, $\mu(J)$ es su medida n-dimensional y A(J) es el número de puntos que pertenecen a la unión entre P y J. En cuanto a la dispersión, hace referencia a la máxima distancia a la que cada punto de un conjunto puede estar respecto al punto mas cercano perteneciente a la misma secuencia. Por lo general, los conjuntos de puntos que presentan una baja discrepancia también poseerán una baja dispersión.

Se desarrollaron cuatro distintos conjuntos de muestras:

1. Cuadrículas: Consiste en la cuantificación uniforme de cada uno de los ejes de coordenadas.
2. Enrejado: Generalización que mantiene la estructura de una cuadrícula, pero es generado por un conjunto de bases generalmente no ortogonales que buscan obtener una baja discrepancia.
3. Método cerrado: No requiere una estructura de vecindad para las muestras, cuyo número debe ser conocido a priori.
4. Método abierto: No es necesario conocer a priori el número de muestras. Por lo general, se obtiene una discrepancia más alta entre las muestras que con el método cerrado

El primer grupo es un subconjunto del segundo, que a su vez es un subconjunto del tercero. Para probar los resultados que ofrece el uso del método abierto y el cerrado desarrollaron una variante del algoritmo Probabilistic Road Map (de ahora en adelante PRM), mientras que para probar los otros dos conjuntos optaron por una variación del Lazy PRM.

El algoritmo PRM y sus variantes se pueden dividir en dos fases. La primera consisitiría en la generación aleatoria de nodos dentro del mapa con el que se está trabajando. Estos nodos se conectan entre sí para crear un grafo, siempre evitando generar nodos o conexiones encima de obstáculos. Una vez obtenido dicho grafo, la segunda fase consiste en encontrar el camino mas corto entre el punto inicial y el final, a través de las distintas conexiones que conforman el grafo.

En cuanto al Lazy PRM, sigue la misma estructura que el PRM clásico, con una diferencia. Durante la primera fase, al generar el grafo, no se tienen en cuenta los obstáculos al distribuir los nodos y crear las conexiones entre ellos. Después, se usa un algoritmo para encontrar el camino a través del grafo (en este caso se ha usado el A*) y se genera una trayectoria. Por último, se comprueba si alguno de los tramos del camino colisiona con un obstáculo. Si esto ocurre,

se elimina la conexión o el nodo que colisione y se vuelve a buscar el camino. Esto se repite hasta que se encuentre una solución factible. Por lo general, la comprobación de colisiones es bastante costosa computacionalmente, por lo que empleando este método se puede reducir el tiempo de cálculo en entornos con una gran cantidad de nodos y de obstáculos.

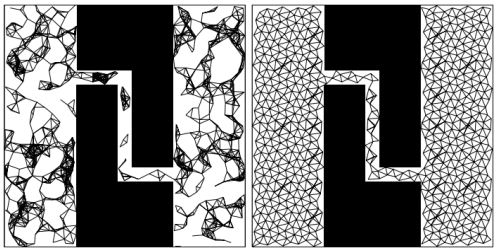


Figura 2.3: Comparación entre PRM y Q-PRM

En la figura 2.3 se puede observar el resultado de aplicar un muestreo aleatorio del espacio libre para generar los nodos. Al coger puntos al azar se da lugar a que existan grandes regiones del espacio en la que no existan nodos, mientras que puede haber otras con un mayor numero de puntos de los que son realmente necesarios. Como consecuencia directa de esto, en casos donde el espacio sea limitado, como puede ser mapas con una gran densidad de obstáculos o pasillos estrechos, podría llegar a no encontrarse un camino, al no presentar nodos lo suficientemente cerca como para que exista una conexión entre ellos.

Para tratar de solucionar esto, se han desarrollado conjuntos deterministas de puntos, con los que se trata de explorar de manera mas efectiva el espacio libre, además de reducir la discrepancia entre puntos. En conccreto, en este paper se utilizan tres distinos tipos de datos de entrada, conocidos como puntos de Hammersley, puntos de Halton y puntos de Faure. Se hablará mas en profundidad acerca de estos conjuntos en la sección 4.2.

Con esto se consigue solucionar algunos de los problemas que presentan las aproximaciones totalmente aleatorias sin que desaparezcan las ventajas que estos métodos presentan frente a los métodos deterministas clásicos.

## 2.3. Comparativa

Realmente no se puede hacer una comparación directa entre el artículo principal y el complementario, ya que el artículo principal consiste en una crítica al trabajo de R. Brooks y no presenta ningún método de planificación. Sin embargo si que se podria establecer una comparación entre el trabajo de Brooks que trata sobre representar el $C_free$ como conos generalizados y el método basado en muestras pseudo-aleatorias.

En primer lugar se puede empezar comentando que ambos métodos presentan una estructura similar. Se comienza muestreando el $C_free$ por el que se puede mover el robot y posteriormente se usan dichas muestras para buscar un camino usando alguno de los múltiples algoritmos disponibles, como pueden ser el $A^*$ o el Dijkstra.

La mayor diferencia se encuentra en el modo de muestrear el espacio. El Q-PRM soluciona esto creando una secuencia determinista de muestras pseudo-aleatorias que conformarán un grafo a través del cual se buscará el camino mas rápido, mientras que en el caso del método de los conos generalizados se busca representar el espacio libre empleando dicha figura geométrica para luego buscar el camino. En la figura 2.4 se puede observar como trabajan ambos métodos

Esta es una de las desventajas del algoritmo planteado por Brooks, ya que necesita representar todo el $C_free$ para encontrar una trayectoria, mientras que en el Q-PRM solo necesitas saber que los nodos y las conexiones que conforman el grafo no colisionan con ningún obstáculo. Además de esto, la complejidad de implementar el Q-PRM es bastante inferior, ya que utiliza conjuntos de muestras ya definidos en el campo de las matemáticas, por lo que todo se reduce a elegir cual de los distintos grupos se va a utilizar. En cuanto a la crítica que Yongji Wang hace al trabajo de Brooks, igualmente sería aplicable al algoritmo Q-PRM, ya que también sería necesario poder realizar movimientos de rotación pura.

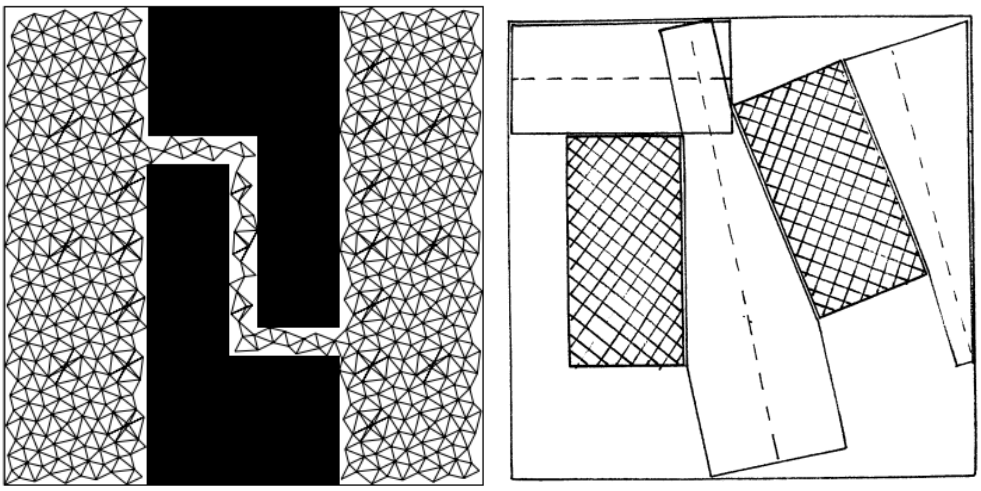


Figura 2.4: Muestreo del escenario por parte de ambos métodos

# Capítulo 3

# Caso de uso

## 3.1. Justificación del algoritmo

??

## 3.2. Problema

**Descripción del problema que nos hemos inventado (CEABOT??)**

## 3.3. Ámbito de uso

?? limitaciones??

# Capítulo 4

# Implementación

**Introducción si es necesario**

## 4.1. Simulación

**Hablar sobre la simulación aquí**

Para testar el funcionamiento del algoritmo se han realizado diferentes experimentos sobre el entorno de simulación robótica V-REP [ javi ]. V-REP es un simulador muy versátil que permite diferentes opciones a la hora de integrarse en un proyecto. Por un lado, está basado en el lenguaje Lua y ofrece un interprete integrado en el propio simulador, sin embargo, también permite utilizar programas externos en diversos lenguajes de programación como son: C++, Python o Java. Además, V-REP está integrado en ROS [ javi ] y pone a disposición del usuario los paquetes necesarios para realizar simulaciones de proyectos basados en ROS. En el caso que nos ocupa, se ha utilizado el lenguaje Python a través de la API proporcionada por V-REP para el control remoto de la simulación.

### 4.1.1. Robot seleccionado

Como base para implementar los algoritmos, se ha seleccionado un robot móvil con dos grados de libertad y configuración holonómica. V-REP contiene varios modelos de robots reales de este tipo, tales como: el javi , javi o el Pioneer 3D-X. Hemos utilizado este último dado que es un robot bastante usado en tareas de navegación, y su relación entre tamaño y prestaciones es adecuada para nuestro problema.

[ javi foto ]

El Pionner 3D-X es un robot móvil impulsado por dos ruedas de movimiento diferencial. Junto a sus motores equipa encoders incrementales de 500 posiciones. También incluye sensores ultrasonicos de posición. Sin esta aplicación no ha sido necesario utilizar ningún sensor interno del robot. El Pionner 3D-X tiene unas dimensiones de 38.1x45.5x23.7 cm . Su velocidad se acerca a los 1.6 metros por segundo y tiene una capacidad de carga de hasta 23 kg.

### 4.1.2. Desarrollo del controlador del robot

La API remota de V-REP proporciona acceso al control de los actuadores del robot. Tomando como base la modificacion de parámetros de bajo nivel de los actuadores (velocidad y sentido de giro), se ha desarrollado un controlador que permite al robot desplazarse entre dos puntos con un error menor a 10cm. Teniendo en cuenta el tamaño del robot, se considera que es una precisión muy razonable.

El primer controlador implementado ha sido la función de giro y orientación. En nuestra aplicación, es muy importante realizar giros controlados, ya que repercutirán con gran importancia en la suavidad de los movientos del robot, así como en el tiempo ocupado en realizar la prueba. Por una parte, se requiere conocer en todo momento la orientación del robot respecto al eje de coordenadas global del simulador. Al mismo tiempo, el controlador debe permitir modificar esta orientación con el objetivo de dirigir el robot correctamente hacia los puntos objetivos. A continuación, en la figura [ javi ] se muestra un diagrama de estados en el que se define el modo de actuación del robot tras una orden de giro.

[ javi ]

El segundo controlador del robot corresponde a su traslación. Ya que el robot constituye el punto de inicio de cualquier trayectoria que sea desarrollada, se debe conocer su posición inicial antes de correr el algoritmo de planificación. También,

durante la simulación, se requiere un conocimiento en tiempo real de la posición global del robot. Este controlador nos permitirá definir el desplazamien y la consecución de objetivos del robot. En el siguiente diagrama de estados, representado en la figura [ javi ], se muestra la implementación de este controlador.

### 4.1.3. Escenarios de simulación

Los escenarios propuestos se han diseñado para probar la eficacia del algoritmo frente a diferentes situaciones problemáticas. Entre los puntos críticos de los escenarios destacan: tuneles con una altura definida, pasillos estrechos y zigzagueos. El uso de bloques sólidos modulares permite realizar configuraciones precisas para probar de forma directa las capacidades del algoritmo. Se han diseñado 3 escenarios con un area de javi m.

- Escenario A: Túneles y obstáculos de diferente altura. Puede observarse en la figura javi .
- Escenario B: Zigzagueo constante y pasillos estrechos. Puede observarse en la figura javi .
- Escenario C: Integración de túneles, zigzagueos y pasillos estrechos. Puede observarse en la figura javi .

### 4.1.4. Extracción de datos de la simulación

Uno de los puntos mas interesantes al trabajar con V-REP es la generación de planos a partir de los escenarios. La aplicación desarrollada está programada para funcionar en dos dimensiones, sin embargo, para crear los mapas se han tenido en cuenta tres dimensiones. El motivo de esto es obtener un funcionamiento óptimo del robot en situaciones cuya cruzabilidad se vea comprometida. Los casos más numerosos son aquellos en los que el robot se enfrenta al paso bajo un obstáculo elevado.

Para extraer el plano se ha utilizado una cámara aérea de proyección ortogonal. Su rango de visión abarca la superficie completa del escenario y está situada a una altura ligeramente superior a la del robot. Dicho de otra forma, la cámara captará todos los objetos que se encuentren en el escenario que intercedan en el volumen mínimo sobre el que puede garantizarse la cruzabilidad.

En las imágenes de la figura [ javi ] se puede observar que los obstáculos cuya altura es superior al robot, no se tienen en cuenta a la hora de procesar el algoritmo.

Las imágenes son transferidas desde el simulador al programa principal, dónde se les aplica el algoritmo y comienza la ejecución.

## 4.2. Algoritmo

En esta sección se presenta el algoritmo que se ha implementado para el caso de uso que se ha tratado. El algoritmo se puede ver en el flujograma de la figura **??**.

**Insertar flujograma!**

En primer lugar se procesa la imagen obtenida del escenario (mapa) para extraer los obstáculos. A continuación, se generan los puntos (muestras) que corresponderán a los nodos del roadmap y se conectan siguendo un criterio de distancias. Por último, cuando un punto inicial y un punto final son pedidos, se incluyen en el grafo y se calcula el camino más corto mediante un algoritmo de búsqueda en grafos, tal como el Djisktra o el $A^*$.

### 4.2.1. Procesamiento del mapa

A partir de la imagen del escenario capturada desde el simulador, se realiza un procesamiento de la misma usando técnicas de visión por computador. Se comienza por realizar un umbralizado de manera que el resultado es una imagen binaria en la que los obstáculos son píxeles blancos y el espacio libre píxeles negros.

A la imagen binaria se le realiza un etiquetado de objetos para obtener los contornos de los distintos obstáculos, los cuales son posteriormente simplificados para reducir el número de puntos que definen el obstáculo, mejorando la eficiencia del algoritmo de detección de colisiones.

Estos contornos que definen a los obstáculos son usados en posteriores partes del algoritmo. En la figura **??** se especifica el diagrama de flujo del procesamiento del mapa.

### 4.2.2. Muestreo del espacio libre

El siguiente paso para la generación del roadmap es el muestreo del espacio libre. En el algoritmo original este muestreo se realiza de forma aleatoria, generando nodos del roadmap repartidos aleatoriamente por todo el entorno. La mejora que introduce el artículo que es objeto de estudio con este caso práctico es la generación de dicho muestreo a partir de conjuntos de puntos cuasi-aleatorios. Estas distribuciones de puntos tienen la ventaja de ocupar el espacio libre de forma más eficiente, por lo que con menos cantidad de puntos se puede abarcar un área mayor y, por tanto, más partes delicadas del mapa. Estas partes delicadas pueden ser, por ejemplo, estrechamientos del espacio libre entre obstáculos. Los autores del artículo dan una medida de la ocupación eficiente del espacio libre con un parámetro que ellos llaman *discrepancia*:

$$D_N(P) = \sup_j \left| \frac{A(J)}{N} - \mu(J) \right|$$

En la que P es un conjunto de N puntos d-dimensionales, $\{x_0, ..., x_{N-1}\}$ en $[0,1]^d$, J es cualquier subconjunto rectangular de $[0,1]^d$, $\mu()$ es su **measure** y A(J) es el número de puntos contenido en $P \cap J$.

Para el muestreo de puntos cuasi-aleatorios hemos usado dos distribuciones de puntos: el conjunto de Hammersley y el conjunto de Halton. Estas distribuciones se generan a partir de una semilla para un número arbitrario de dimensiones. Estas distribuciones se pueden calcular de la siguiente manera:

#### Conjunto de Hammersley

Dados $d-1$ números primos distintos $p_1, p_2, ..., p_{d-1}$ el i-ésimo punto del conjunto es dado por la expresión:

$$\left( \frac{i}{N}, r_{p_1}(i), ..., r_{p_{d-1}(i)} \right), \qquad i = 0, 1, ..., N-1$$

#### Conjunto de Halton

Dados $d$ números primos distintos $p_1, p_2, ..., p_d$ el i-ésimo punto del conjunto es dado por la siguiente expresión:

$$(r_{p_1}(i), r_{p_2}(i), ..., r_{p_d}(i))$$

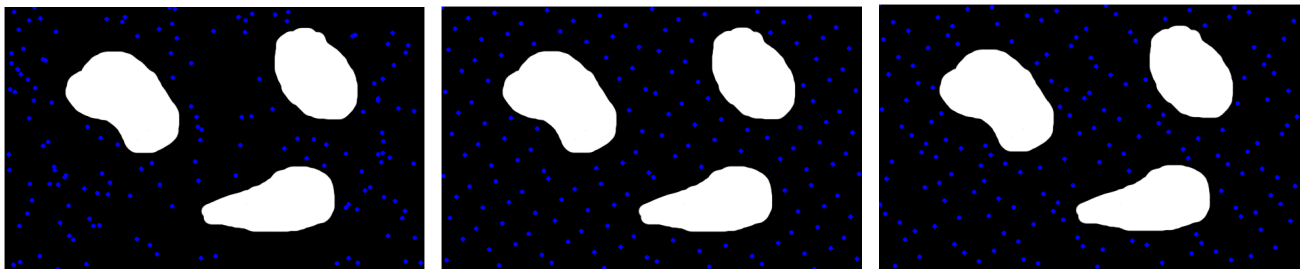
En ambos casos, la función $r_p(i)$ se obtiene escribiendo los dígitos de la notación basada en $p$ en orden inverso. Por ejemplo, para la expresión $i = a_0 + a_1 p + a_2 p^2 + a_3 p^3 + ...$ donde $a_j \in \{0, 1, ..., p-1\}$ la función $r_p(i)$ sería:

$$r_p(i) = \frac{a_0}{p} + \frac{a_1}{p^2} + \frac{a_2}{p^3} + \frac{a_3}{p^4} + ...$$

Para nuestra implementación hemos usado una biblioteca, perteneciente al paquete cgkit que nos proporcionaba estos conjuntos de puntos de manera cómoda y rápida, ahorrándonos tiempo de desarrollo y depuración de errores. La figura 4.1 muestra una comparativa de los distintos métodos de generación de puntos de muestreo en un escenario ficticio.

## 4.3. Resultados

**Resultados y explicación de cómo afecta cada cosa al resultado**



(a) Muestreo aleatorio (b) Muestreo con puntos Hammersley (c) Muestreo con puntos Halton

Figura 4.1: Comparativa de los distintos modos de muestreo del espacio libre

# Capítulo 5

# Conclusiones

**Conclusiones y resultados y esas cosas**