Técnicas de IA aplicadas a la robótica

Módulo de planificación

Pablo Francisco Pérez Hidalgo

05/05/2010

Índice

Introducción

A lo largo de esta sección de comentarán algunos detalles que deben ser conocidos para comprender el fin del módulo de planificación y algunas de las decisiones tomadas a lo largo de su diseño y desarrollo.

Planteamiento del problema a resolver

Este documento forma parte de una serie de tres entregas, en las cuales se desarrollan las componentes esenciales de un robot móvil según la aproximación planteada en la asignatura "Técnicas de Inteligencia Artificial Aplicadas a la robótica". Dicha aproximación de robótica móvil plantea la construcción del software de control para un robot y, dicho software, debe satisfacer los siguientes requisitos funcionales:

- El robot sólo necesita ser controlado en lo que a su movimiento en un único plano del espacio respecta.
- Dicho control debe basarse en tres fuentes de datos:
  - Mapa o representación del plano sobre el cual se mueve el robot.
  - Datos recogidos por los sensores del robot:
    - 8 Sensores de distancia a obstáculos (usando tecnología de ultrasonidos).
    - Contabilidad odométrica del espacio recorrido.
  - Lectura de marcadores visuales situados en el espacio que puede ser recorrido por el robot. Esta lectura se realiza a través de la adquisición de imágenes por medio de una cámara.

La toma de decisiones de movimiento se basa en las tres fuentes de datos citadas. Dichas fuentes manejan, cada una, un nivel de abstracción diferente y son tomadas con diferentes prioridades a la hora de tomar decisiones. Cada módulo maneja los datos proporcionados por una fuente y toma una decisión en función de sus propios cálculos y los del módulo anterior en lo que a nivel de abstracción respecta. La Figura 1 pretende explicar esta estructura modular.

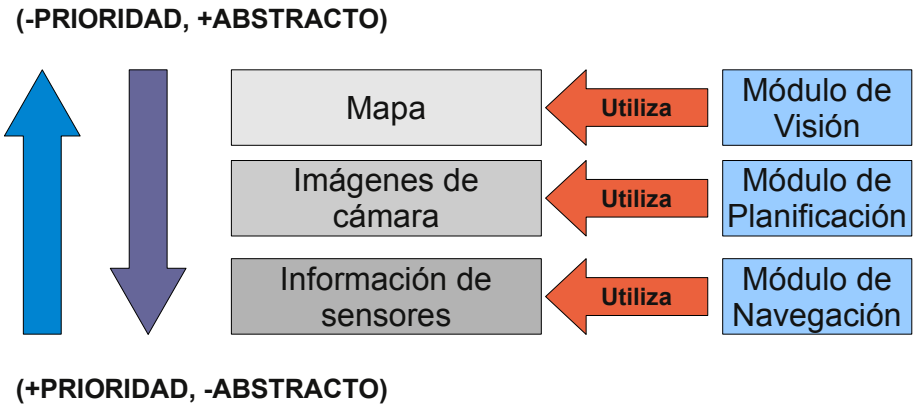


Figura 1: Estructura del SW controlador.

El orden de desarrollo de los módulos es el siguiente:

1. Módulo de Navegación.
2. Módulo de Planificación.
3. Módulo de visión

Dado que esta es la segunda entrega, el problema abordado es el de la planificación. Como ya debe saber el lector, suponiendo que conoce el contenido de la primera entrega, el módulo de navegación es capaz de llevar al robot de un punto a otro sin chocar bajo las limitaciones mencionadas en la subsección *Carencias de un sistema reactivo como único controlador*, dichas limitaciones hacen necesario el poder utilizar un mapa para que el robot "no se pierda" ante situaciones de máximos locales de la función objetivo del controlador reactivo. Para poder utilizar un mapa, el módulo de navegación debe ser capaz de interpretarlo y proporcionar, en base a este, una ruta o conjunto de hitos que definan un camino desde la posición del robot hasta la posición objetivo. Existen dos formas fundamentales de realizar esta tarea que varían según la representación del espacio que utilizan:

- **Forma 1:** Planificación en base a representación métrica del espacio.
- **Forma 2:** Planificación en base a representación topológica del espacio.

El explicar las diferencias entre ambos tipos de representaciones del espacio no es el objetivo de este documento y pueden ser encontradas en un gran número de libros e, incluso, en las transparencias de la asignatura.

Al comenzar el desarrollo de este módulo, ya se contaba con una implementación de planificación basada en la representación métrica del espacio. Dicha solución consiste en calcular la distancia hasta el punto objetivo desde cada celda en las que se divide el plano (grid). Para dar lugar al camino que, desde la celda que representa la posición inicial, toma la decisión de acercarse a las celdas adyacentes con distancia al punto objetivo decreciente. La obtención de las distancias desde cada celda al punto objetivo se realiza mediante un algoritmo de programación dinámica. Este método de planificación presenta los siguientes problemas:

- Es muy sensible a la presencia de obstáculos ya que el módulo de planificación

(cuyas decisiones son más prioritarias) puede que desvíe al robot a celdas con mayor distancia hasta el objetivo. Hay que buscar una solución de compromiso para la granularidad de la ruta (mayor cuanto menor es la distancia entre puntos consecutivos de esta) ya que cuanto mayor es esta, más susceptible es la ruta a ser ignorada por culpa de las decisiones tomadas en el módulo de Navegación con el fin de evitar obstáculos.

- Pequeños cambios en la representación del espacio pueden dar lugar a grandes cambios en la planificación.
- Una corrección en la posición del robot supone la necesidad de volver a realizar la planificación, es decir, generar una nueva ruta.
- La complejidad algorítmica crece demasiado rápido cuando crece el tamaño de la entrada.

Debido a estas limitaciones, se construye un nuevo módulo de planificación basado en una representación topológica del plano. Antes de entrar en detalles sobre esta implementación, se detallarán las limitaciones de los controladores reactivos ante estímulos sensoriales como únicos controladores y se introducirán algunos conceptos matemáticos necesarios para el entendimiento de la solución topológica.

Carencias de un sistema reactivo como único controlador

Aunque estas carencias ya están detalladas en la primera entrega y se han expuesto levemente en secciones anteriores, se recogen con el fin de motivar el esfuerzo que supone el desarrollo de un buen módulo de planificación.

El sistema de toma de decisiones reactivo (implementado en el módulo de Navegación) basa, como ya se ha comentado, sus conclusiones únicamente en la posición actual del robot (calculada de forma aproximada gracias a la contabilidad odométrica), en los datos proporcionados por los sensores de distancia (a partir de ahora se les llamará “sonar” en el documento) y en el conocimiento de la posición geométrica del punto objetivo.

El sistema reactivo ha sido implementado mediante un modelo neuronal (perceptrón) que, desde un punto de vista matemático, no es más que una función polinómica y lineal cuyas variables son los datos de los 8 sensores sonar y el valor de la función a optimizar (concretamente a minimizar). La función a minimizar no es más que la diferencia de ángulo entre el encabezamiento del robot y la dirección que habría que seguir si se pudiera llegar al punto objetivo en línea recta. La función polinómica determina pues el sentido y velocidad de giro del robot en cada paso. Por otro lado, otra función lineal cuya entrada es la velocidad y sentido de giro, determina la velocidad de avance del robot.

Lo expuesto puede parecer una solución completa pero existe una afirmación en dicho método que lo condena como controlador total del robot: *La dirección que habría que seguir si se pudiera llegar al punto objetivo en línea recta*. Es trivial el darse cuenta de que en un edificio existen habitaciones y, por tanto, en una planta hay puntos que no

son accesibles desde otros puntos con una trayectoria rectilínea. Esto hace que la función polinómica que determina la velocidad de giro (e, indirectamente, la velocidad de avance; Es decir, todas las decisiones de movimiento del robot) caiga en mínimos locales de la función a optimizar lo que se traduce en un movimiento circular e incesante del robot sobre un centro imaginario.

Es necesario tener información acerca de las habitaciones y de que puntos pertenecen a que habitaciones con el fin de poder determinar un camino (lista ordenada de puntos o hitos) que cumpla que entre dos puntos consecutivos no existan mínimos locales para la función que toma las decisiones en el sistema reactivo. Para ello se desarrolla y aplica el módulo de navegación: Para obtener caminos, también llamados rutas.

Conceptos aplicados en el desarrollo del módulo de planificación

El módulo de planificación desarrollado se basa en una búsqueda de rutas a partir de una estructura matemática que, aunque se genera a partir de la representación métrica del plano (a partir del GRID), contiene únicamente, información topológica de este: Esta estructura es un grafo cuyos nodos representan las *habitaciones* del plano y cuyas aristas representan la accesibilidad entre habitaciones (hay arista entre dos habitaciones si y sólo si una está conectada con la otra).

Se hace necesario el definir o concretar el concepto de *habitación*.
Una habitación no es más que un conjunto de puntos conexo y todos ellos envueltos por una envolvente convexa, es decir, un conjunto convexo de puntos conexos. De forma que, dentro de una habitación (mientras el robot se mueva entre los puntos que componen un conjunto habitación) no existen óptimos locales para la función de toma de decisiones del módulo de navegación. Ello implica que dentro de una habitación, el robot puede guiarse completamente gracias al módulo de navegación sin posibilidad de entrar en bucles infinitos.

Ha surgido en varias ocasiones, a lo largo de este documento, la palabra GRID. El GRID es una representación métrica del plano por el que se moverá el robot. Se genera dividiendo dicho plano en una cuadrícula (matriz) en la que cada elemento tiene un valor lógico alto o bajo. Si es alto, todos los puntos del espacio representados por dicho elemento se consideran como pertenecientes a un obstáculo (por ejemplo, una pared) y si es bajo, se considera que todos los puntos que son representados por el elemento pueden ser ocupados por el robot.

La representación topológica del espacio, no es más que la representación de todos los puntos de cada habitación por su nodo correspondiente (conocido como centro de la habitación aunque no represente el centro geométrico de esta) y la representación de la accesibilidad entre habitaciones por medio de aristas entre sus respectivos nodos representantes. No se han considerado pesos entre aristas aunque podría haberse hecho usando experiencias anteriores (otros recorridos) o distancias (basadas en diversas normas) entre los puntos representantes de las habitaciones conectadas por las aristas.

Proceso de planificación y descripción de los algoritmos que lo componen

En esta sección se describe el proceso mediante el cual, una imagen del mapa del plano por el que se puede mover el robot (a partir de ahora se le llamará *plano universo* en este documento) acaba convirtiéndose en una plan de ruta, es decir, en un camino óptimo desde el punto origen al punto destino u objetivo. A este proceso se le conoce como *proceso de planificación*.

Estructura en "pipeline" del tratamiento de los datos de entrada

Se considera que el proceso de planificación es un *pipeline* en el que, en cada etapa, se aplica un algoritmo. A cada una de estas etapas se le llama fase. La siguiente figura () resume los algoritmos aplicados de principio a fin del proceso de planificación junto con las entradas y salidas de cada uno de estos.

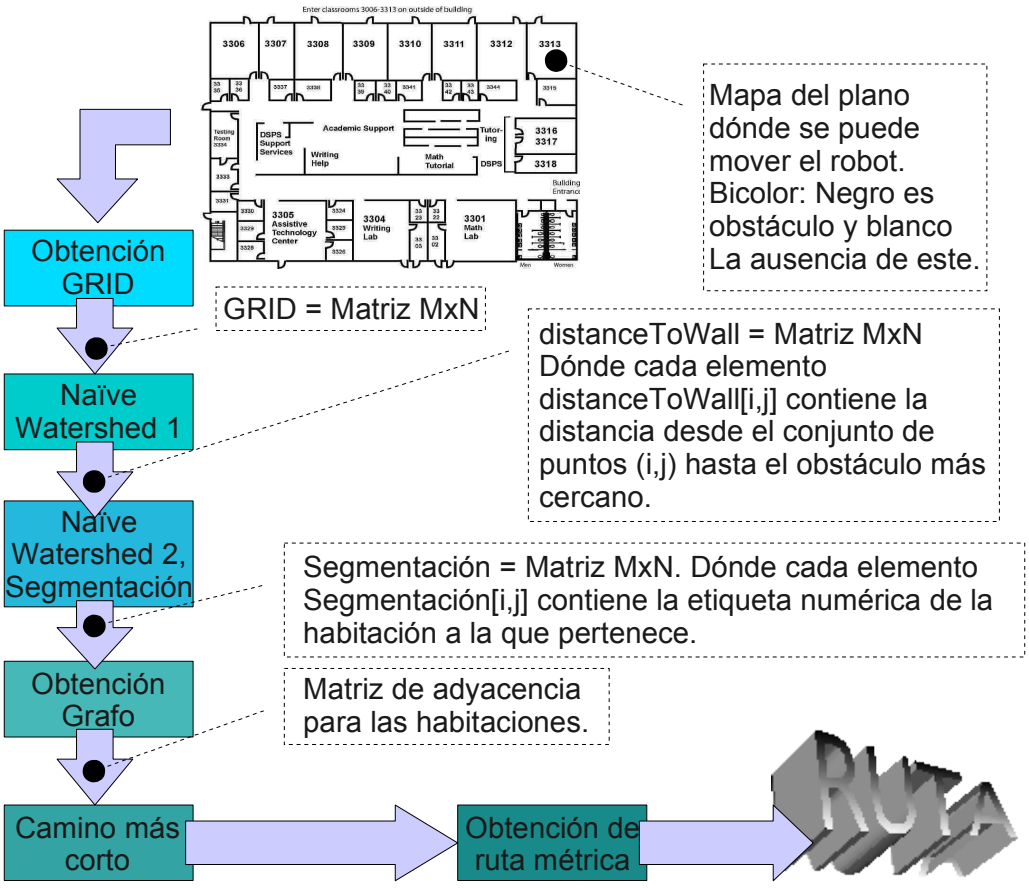


Figura 2: Algoritmos aplicados, presentados en orden junto con sus entradas y salidas.

Fase 0: Construcción de un "grid" a partir de un mapa"

La imagen que determina el mapa a utilizar es, realmente, una representación en una estructura de datos computacional de una imagen real. Esta representación es, por tanto, digital. Es por ello por lo que la construcción del GRID no es más que una simplificación de la representación a través de la reducción de los datos almacenados.

Este proceso no ha sido programado por el autor del documento sino que ya había sido proporcionado por el equipo docente en la primera aproximación de construcción de módulo de planificación: *Planificación a través de representación métrica del plano*. Por esta razón, no se incluirán muchos detalles acerca de como se realiza esta fase sino que se presentan algunas nociones de como hacerlo.

Suponiendo que se dispone de una biblioteca para Python capaz de manejar imágenes, el mapa inicial es representado por medio de una matriz (de tamaño HxW) en la que, cada elemento, indica el nivel de intensidad de cada color primario presente en el punto de la imagen (x,y).

Si se desea construir un GRID de MxN elementos, basta con submuestrear una de cada

$floor\left(\frac{H}{M}\right)$ filas y una de cada $floor\left(\frac{W}{N}\right)$ columnas de la imagen. Tomando, como calores de cada elemento de la nueva matriz (GRID), una función de los puntos encerrados por la cuadrícula resultante que de una representación fiel de la presencia o no de obstáculos en el elemento de la rejilla. Algunas posibilidades para esta función son:

- Marcar el elemento como obstáculo si hay al menos un punto en la celda del color que toman los obstáculos.
- Marcar el elemento como obstáculo si todos los puntos de la celda son del color de un obstáculo.
- Realizar la media de color de los puntos de la celda y considerarla como obstáculo si dicha media supera un umbral.

Hay muchísimas más posibilidades y la que utilice o deje de utilizar el código proporcionado no es del todo relevante a la hora de desarrollar el nuevo proceso de planificación.

Fase 1: Segmentación

Una vez que se dispone del GRID, se puede proceder al inicio de lo que realmente constituye el primer paso en el proceso de planificación, la segmentación de la rejilla. De hecho, podría decirse que la planificación tiene como entrada el GRID del mapa y no considerar la generación de este como parte de la planificación.

El objetivo de la segmentación es clasificar los puntos del GRID por habitaciones. Para ello se utiliza como herramienta una versión del algoritmo Watershed que en este documento se conoce como *Naïve Watershed*.

Algoritmo Naïve Watershed:

En esta fase se pretende crear una representación intermedia del mapa que facilite la segmentación de este en habitaciones. Para ello se ha utilizado una versión simplificada del algoritmo *Watershed* que será llamada en este documento: *Naïve Watershed*.

El algoritmo Watershed (la versión no simplificada) recibe como entrada una imagen digital, en escala de grises, (representada por medio de una matriz) y proporciona, como salida, otra imagen digital (también, y siempre a partir de ahora, representada como una matriz) en la que todos los puntos que proporcional el mismo óptimo local de una función determinada.
En el caso que nos ocupa, lo que se busca es etiquetar como pertenecientes a una misma habitación aquellos puntos encerrados en una misma componentes **conexa y convexa** que cumple que todos los puntos que la componen son conexos con un mismo punto que,a su vez, se caracteriza por ser el que se encuentra a más distancia de las paredes de entre los pertenecientes a la componente.

Para ello se ha utilizado el algoritmo Naïve Watershed que es capaz de hacer precisamente este etiquetado concreto. La primera parte del algoritmo queda descrita en la siguiente figura extraída de los apuntes de la asignatura:

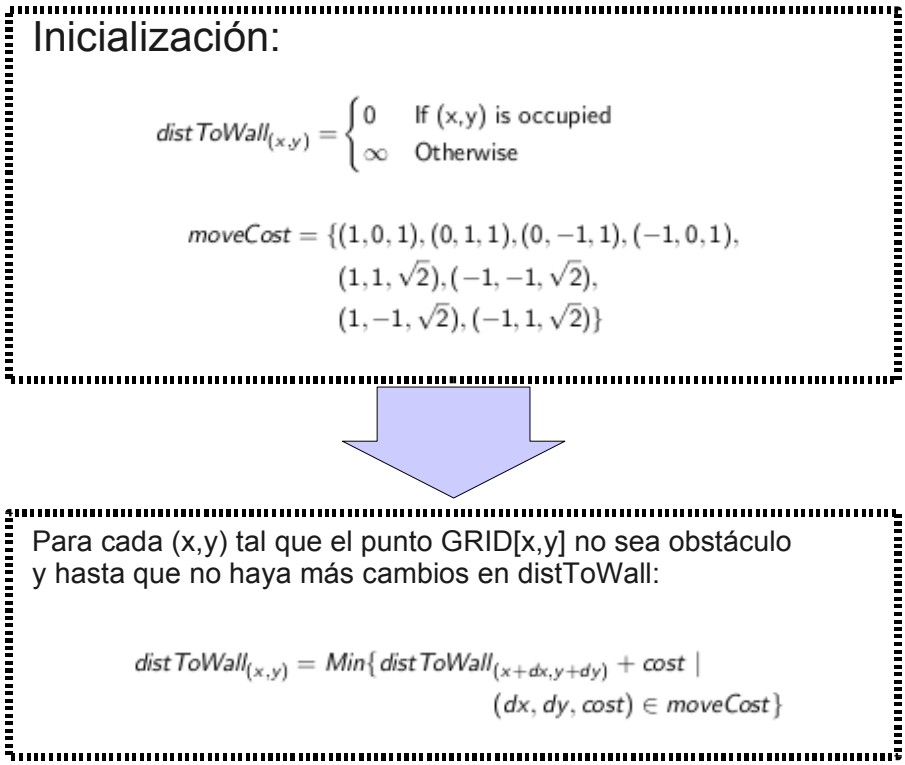


Figura 3: Parte 1 del algoritmo Naïve Watershed, obtención de la función que debe maximizar el punto conexo con el resto de puntos de cada habitación.

Una vez que se tiene la función a maximizar para cada punto de la rejilla (dichos valores están almacenados en la matriz distanceToWall) el algoritmo procede al etiquetado de los puntos que pertenecen a una misma habitación, segunda parte del algoritmo Naïve Watershed.

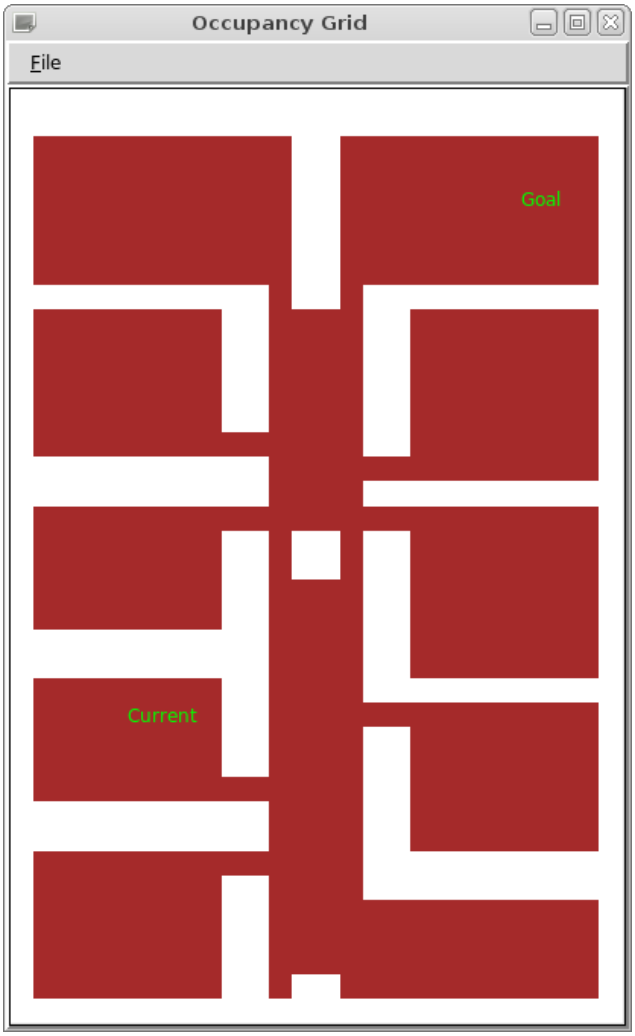
La segunda parte del algoritmo se encuentra resumida en la siguiente descripción en pseudocódigo:

```
1  Clasificación = Matriz de 0's de tamaño MxN
2  Para cada fila → i
    2.1        Para cada columna → j
       2.1.1      Si la celda (i,j) no está ocupada por ningún obstáculo.
          2.1.1.1   Máximo = 0
          2.1.1.2   Calcular el máximo local al que se puede llegar desde la celda
             (i,j), es decir, para cada celda pertenece a Adyacentes(i,j)
             2.1.1.2.1          (l,m) = Adyacente(i,j) tal que distanceToWall(l,m)
                mayor que ese mismo valor para cualquier otro adyacente de (i,j).
             2.1.1.2.2          Si no exite (l,m) tal que distanceToWall(l,m) >
                distanceToWall(i,j)
                2.1.1.2.2.1       Clasificar la celda (i,j) con la etiqueta
                   distanceToWall(l,m), esto es: Clasificación(i,j)
```

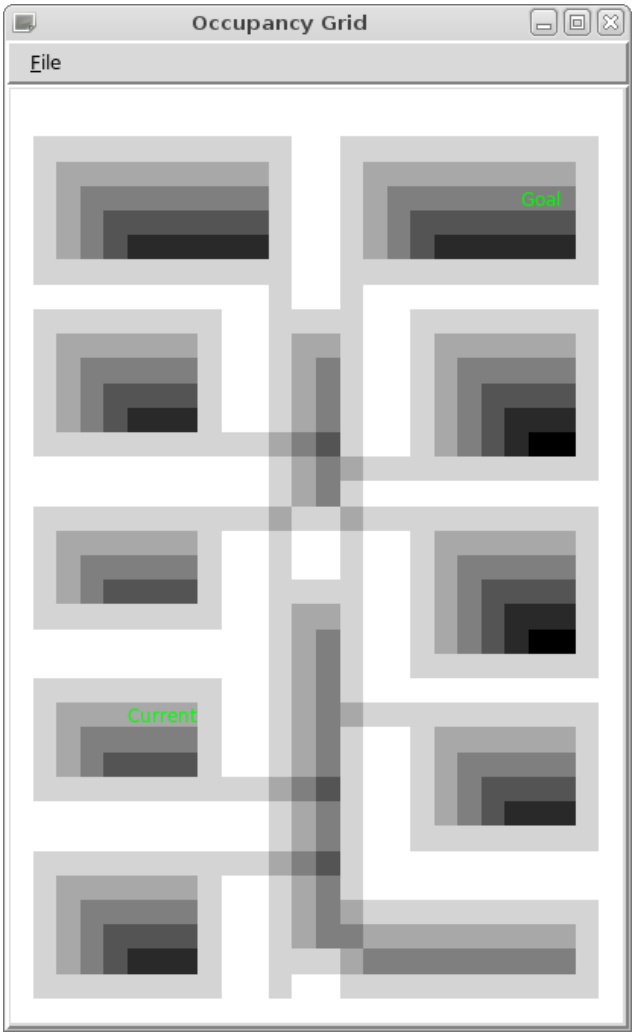
=distanceToWall(l,m).
2.1.1.2.2.2 Cambiar de celda (i,j) según bucle principal.
2.1.2 En caso contrario, ignorar la celda (i,j) y cambiar la celda objetivo, (i,j), según el bucle principal

La segunda parte del algoritmo genera una matriz intermedia (Clasificación) en la que cada celda tiene como valor la distancia a la pared del punto conexo con la celda que alcanza una distancia máxima con el conjunto de puntos obstáculos.
La forma de distinguir habitaciones, y concluir la segmentación, es encontrando los bordes, es decir, dos celdas adyacentes entre sí y con un valor en la matriz de clasificación diferente. Ese es el cometido de la tercera parte del algoritmo.

Antes de pasar a detallar la tercera parte de algoritmo de segmentación, la siguiente tabla , muestra un ejemplo de la evolución del cálculo de la matriz distanceToWall.

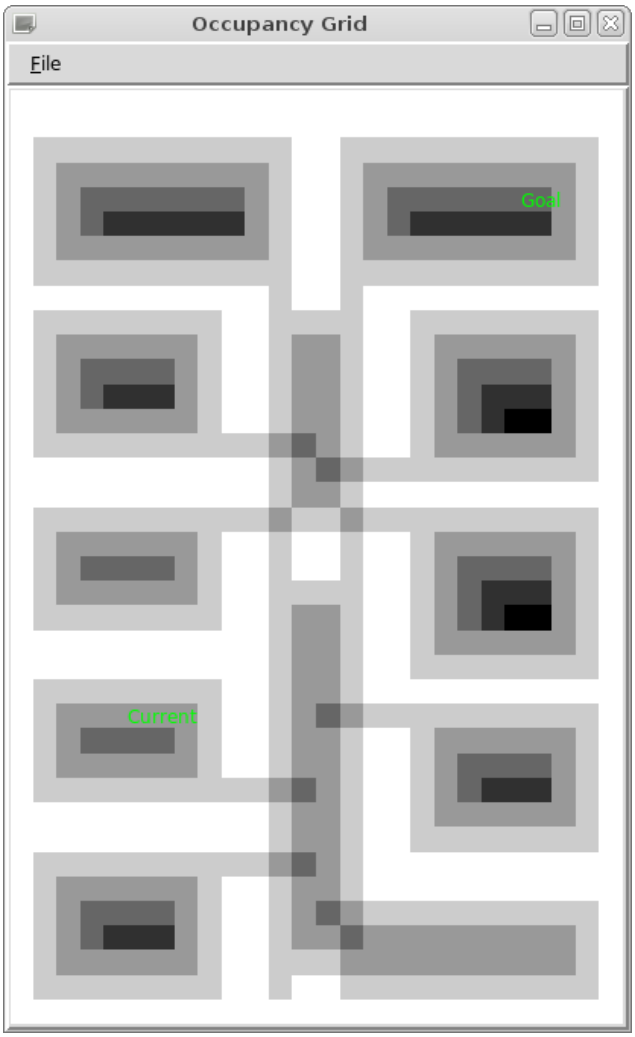


***GRID**: En rojo se muestran las celdas no ocupadas por obstáculos y en blanco las ocupadas.*

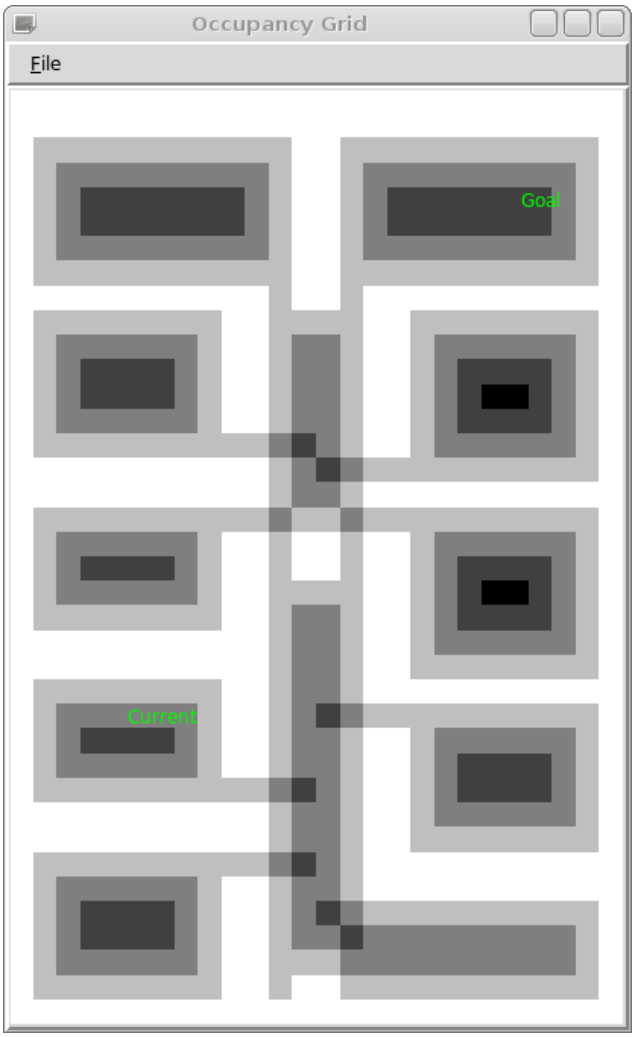


***Iteración 1**: La distancia a las paredes de cada celda se indica mediante la oscuridad de su color. Blanco significa pared (celda ocupada), cualquier otro color representa una celda libre y cuanto más oscuro es el*

color de las celdas más distancia hay entre estas y la pared.



Iteración 2



Iteración 3: Tras esta iteración, no hay más cambios de distanceToWall por lo que la primera parte del algoritmo para.

La tercera parte del algoritmo es la que se encarga de etiquetar todos los puntos de una misma habitación con la misma etiqueta. Es el paso final de la segmentación y además de etiquetar los puntos asocia cada etiqueta de habitación con el centro que la representa, el que representa el máximo local en la distancia a paredes.

Para ello, a cada centro se le asigna una etiqueta incremental. Esta asignación se realiza empezando por los centros de la esquina superior izquierda, recorriendo por filas de izquierda a derecha y de arriba a bajo.
A la hora de clasificar las celdas, estas también se recorren de izquierda a derecha y de arriba a abajo. De forma que a cada una de ellas se les asigna el primer centro no utilizado en otra habitación cuyo máximo local coincide con el de Clasificación[celda].

Si se aplican las fases 1, 2 y 3 del algoritmo de segmentación al ejemplo de la última tabla se obtiene como resultado el representado por la Figura 4.

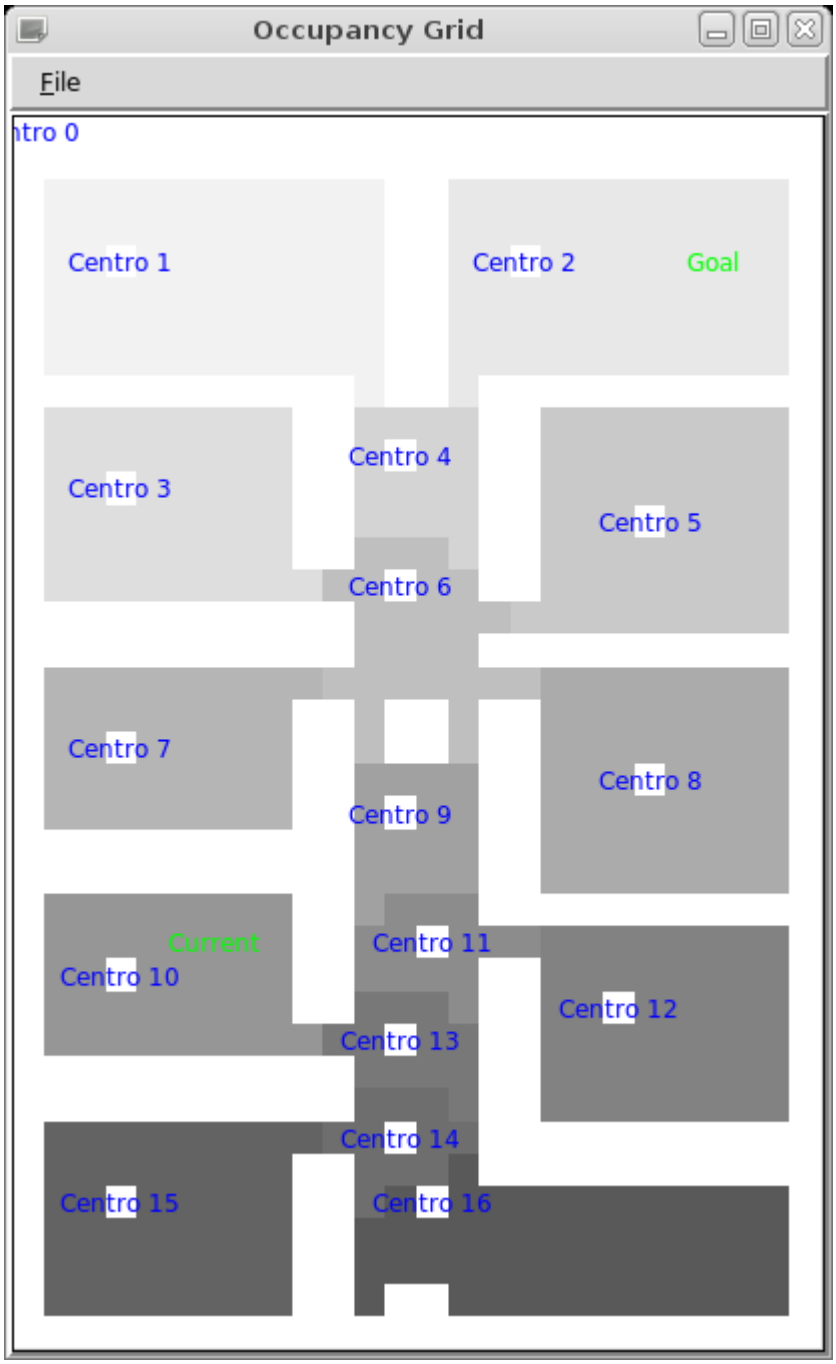


Figura 4: Resultado proporcionado por el algoritmo de segmentación. En diversos tonos de grises se muestran las habitaciones detectadas. Las celdas blancas aisladas rotuladas en azul indican los máximos de distancia a las paredes de cada habitación

Fase 2: Obtención de representación topológica

Si se desea ser del todo riguroso en la clasificación de cada una de las acciones realizadas en el proceso de planificación, gran parte de la segmentación formaría parte de esta fase. No obstante, se ha decidido dedicar esta fase (fase 2) a la construcción de

la estructura topológica finalmente utilizada para la búsqueda del camino óptimo: El grafo.

Así pues, para esta fase, se asume que ya se ha obtenido la matriz Segmentación y se parte de ella como datos de entrada.

El siguiente pseudocódigo resume el algoritmo implementado para realizar esta tarea:

1 **nnodos** = NºHabitaciones obtenidas en la segmentación.
2 **MA** = Matriz de ceros de tamaño NNODOSxNODOS, será la matriz de adyacencia del grafo, cada celda contendrá un valor lógico alto o bajo según la pareja de nodos representada por la celda sea conexa o no (estén conectados los dos miembros de la pareja entre sí o no).
MAF = Matriz de nodos inicialmente con todas sus celdas con valor (0,0). Contiene las coordenadas (i,j) de aquellas celdas que actúan como frontera entre dos habitaciones adyacentes. Sólo tienen sentido los valores de aquellas entradas que en la matriz de adyacencia tienen valor lógico alto (1, True, Cierto, …).
3 Para cada celda, (i,j), no ocupada por obstáculos del GRID:
3.1 Para cada celda adyacente a (i,j) no ocupada por obstáculos->(l,m):
3.1.1 Si $Segmentacion(l,m) \neq Segmentacion(i,j)$ entonces:
3.1.1.1 MA(NºHabitacion(Segmentacion(l,m)), NºHabitacion(Segmentacion(i,j))) = 1
3.1.1.2 MA(NºHabitacion(Segmentacion(i,j)), NºHabitacion(Segmentacion(l,m))) = 1
3.1.1.3 MAF(NºHabitacion(Segmentacion(i,j)), NºHabitacion(Segmentacion(l,m))) = (i,j)
3.1.1.4 MA(NºHabitacion(Segmentacion(l,m)), NºHabitacion(Segmentacion(i,j))) = (l,m)

Para el ejemplo mostrado en la segmentación, el grafo obtenido es el siguiente.

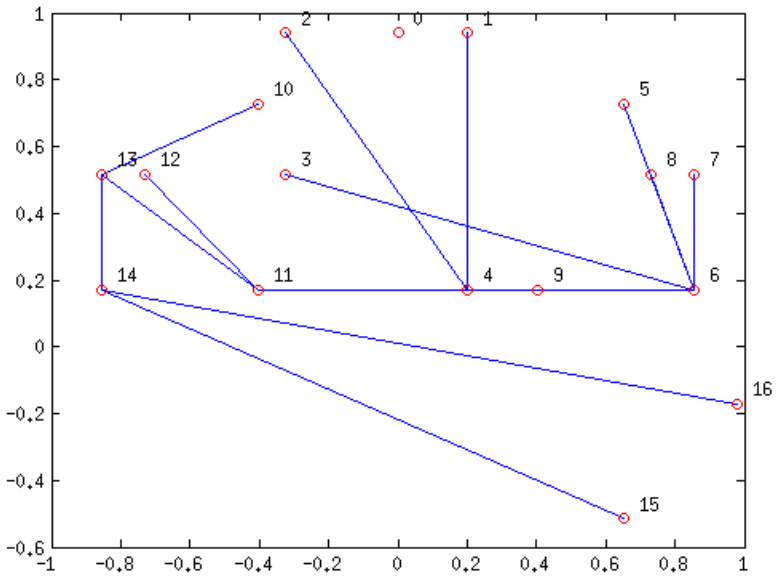


Figura 5: Grafo de conectividad entre habitaciones, las posiciones de los nodos no se corresponden con la de los centros de las habitaciones sino con una proyección sobre el plano de una estructura bucky

Fase 3: Búsqueda del camino más corto y generación final del plan

Una vez que se dispone del grafo de conectividad entre habitaciones, basta con aplicar un algoritmo de búsqueda del camino más corto. En este caso, y siguiendo la recomendación del equipo docente, se ha utilizado un algoritmo de búsqueda del camino más corto A*, del cual hay que destacar que:

- Busca el camino más corto en la representación topológica (el grafo), por lo tanto genera una lista de habitaciones por las que pasar en el camino desde la habitación origen hacia la habitación donde está el punto objetivo.
- La función heurística de estimación de la distancia hasta el objetivo (H) se basa en la representación métrica del plano y utiliza dos distancias:
  - Distancia de Manhattan: $D_{Manhattan}((i,j),(l,m))=|i-l|+|j-m|$
  - Distancia euclídea: $D_{eu}((i,j),(l,m))=\sqrt{(i-l)^2+(j-m)^2}$
- $H((i,j),(l,m))=D_{eu}((i,j),(l,m))+D_{Manhattan}((i,j),(l,m))$

El algoritmo A* proporciona el camino topológico más corto. Para obtener un camino en términos métricos:

```
1  Sea rutaNodos el camino óptimo proporcionado por el algoritmo A*.
2  ruta = Lista de ordenada de celdas del grid con un único elemento: Celda Origen
```

3 Para cada elemento de rutaNodos, en orden y empezando por el segundo (iterador → i)
  3.1 ruta.añadirAlFinal(MAF(i))
  3.2 ruta.añadirAlFinal(Centro(i))
4 **PLAN = ruta**

El resultado final sobre el ejemplo anterior es el indicado por el camino de baldosas rojas en la siguiente figura.

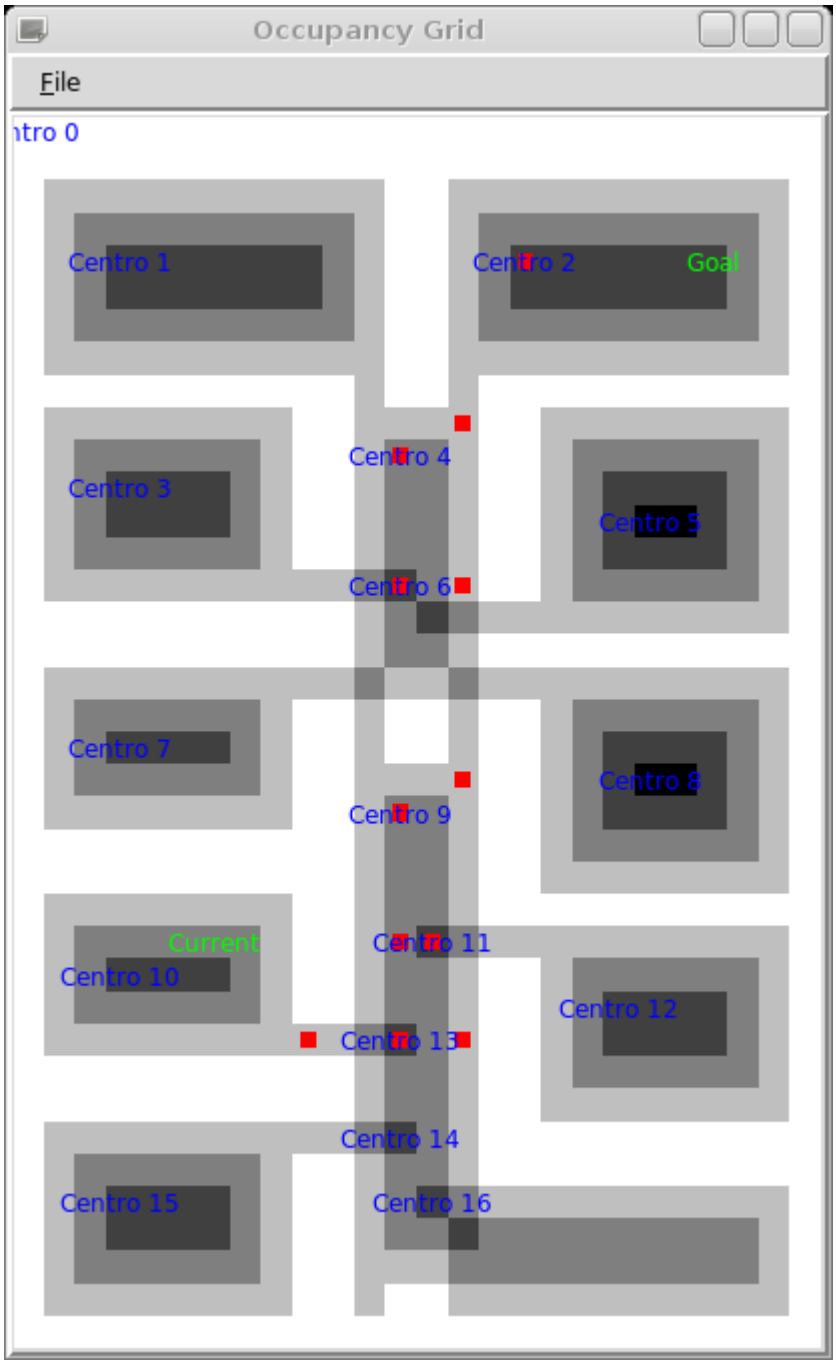


Figura 6: El camino planificado está indicado por las baldosas rojas