

# Localización y trayectorias

Cristian González García  
gonzalezcristian@uniovi.es

Material original de Jordán Pascual  
Espada

v 1.2.1 Noviembre 2022

# Localización

# Localización I

- Sirve para saber
  - **Donde esta el robot en el momento actual**
  - **Posición respecto al medio ambiente**
- **Obtenida / calculada por medio de datos recogidos del entorno**
  - Sensores
  - Actuadores
    - Como soporte de ayuda: movimiento ultrasonidos
  - Lógica (Software)

## Localización II

- Para localizar el robot **hace falta información del entorno**
- Normalmente **mecanismos de entrada**
  - Visión por computador
  - Sistema de Posicionamiento Global (GPS)
  - Sensores de ultrasonidos
  - Brújulas
  - Encoders
  - Balizas/Códigos QR
- El robot transforma en coordenadas su posición, ejemplos:
  - Coordenadas respecto a un **punto de partida**
    - La (0, 0)
  - Coordenadas **X e Y actuales** respecto a algo
    - (X, Y)
  - Otros: pasos, metros, distancia, grados... Con un **encoder**

# Localización III

- o **Tipos de localización**

- o **Seguimiento relativo**

- o Hace un **seguimiento de la localización a través de la posición actual**
    - o **Realmente no maneja** una posición, sino **una diferencia respecto a la posición inicial**
      - o ¿Dónde estoy? Inicio → Derecha – Izquierda – De frente.
      - o ¿Dónde estoy? Inicio (0, 0) → (+3x, -5y).
      - o ¿Dónde estoy? Inicio (0, 0) → Ruedas adelante 2m, giro derecha 30ms, ruedas adelante 1m.

- o **Localización global**

- o Determina su localización con un medio
      - o GPS, mapa, etc.

- o Es necesario para la **capacidad de secuestro** (problema del robot secuestrado)

- o El robot **se coloca en una ubicación sin que sea consciente de que ha sido movido**
    - o Movimiento del robot por una persona, errores, pérdida de datos, fallos de localización, etc.
  - o Debe recolocarse a si mismo en su nueva posición
    - o Problema muy complejo
    - o ¿Continúa trabajando como siempre o cambia?
      - o Robot tractor con coordenadas prefijadas vs aspiradora

# Localización IV

- Localización **Activa y Pasiva**

- Activa**

- Busca activamente puntos de referencia** en su entorno para localizarse
    - La localización es una tarea propia del robot**
    - Trata de** localizarse o **mejorar la estimación de su localización**
      - Disminuir el error en la estimación de la localización
    - Ejemplo: salida directa del laberinto, aspiradora, coche autónomo

- Pasiva**

- El robot no se centra en la localización**
      - Se centra en su tarea**
      - La localización se estima en segundo plano
      - Se «reaprovechan» los datos obtenidos para la tarea
      - Solo se tienen en cuenta los movimientos del robot
    - Ejemplo: resolución del laberinto, aspiradora

# Localización V

- o Tipo de **entorno**

- o **Estático**

- o El entorno apenas, o incluso no varía
    - o La postura del robot o el robot es lo que se mueve
    - o Casas, jardines, etc.

- o **Dinámico**

- o Tanto el robot como su entorno se mueven
    - o El entorno es muy cambiante
    - o Calles, entornos con gente moviéndose, carretera, etc.

# Localización VI

- o **Número de robots** controlados

- o **1** robot

- o Todos los datos son recogidos por el robot que los necesita
    - o No necesita datos de otros robots
    - o Ejemplo: robots de clase

- o **Varios** robots

- o Los robots pueden comunicarse entre ellos
    - o Necesitan información sobre o de los otros robots
      - o Situación, quién hace qué tarea, quién entra primero al cruce, ...
    - o Ejemplo: robots Kiva de Amazon, robots granjeros



# Técnicas y algoritmos de localización

# Técnicas y algoritmos para la localización

- Odometría
- Triangulación y trilateración
- Balizas
- Probabilística
- Markov
- Otros

# Odometría I

- **Técnica utilizada** por algunos controles robóticos **para estimar su posición relativa a un punto de partida**
- **Utiliza la rotación de las ruedas** para estimar los cambios de posiciones
  - Necesitamos las especificaciones del robot
    - Diámetro de ruedas, velocidad, deslizamiento, error, etc.
- Suele ser **muy sensible a errores**
  - **Precisión cuestionable en muchos entornos**
    - Problemas de deslizamiento, baches, ruedas en el aire, etc.

# Odometría II – Tipos de errores

- **Errores sistemáticos**

- **Se realizan en todas las mediciones** de una magnitud, **por defectos de los instrumentos, por el proceso de medición, etc.**
- El error está implícito en el estudio y **no tiende a 0 al aumentar la muestra**
- Depende las características del robot y sus sensores
- Son importantes puesto que **se acumulan**
  - Mala alineación de las ruedas, diferencia de diámetro entre ruedas, mala adherencia de las ruedas, desajuste en la calibración de sus motores, fallos en los sensores de recogida de datos (encoders), etc.

- **Errores no sistemáticos/aleatorio/accidental**

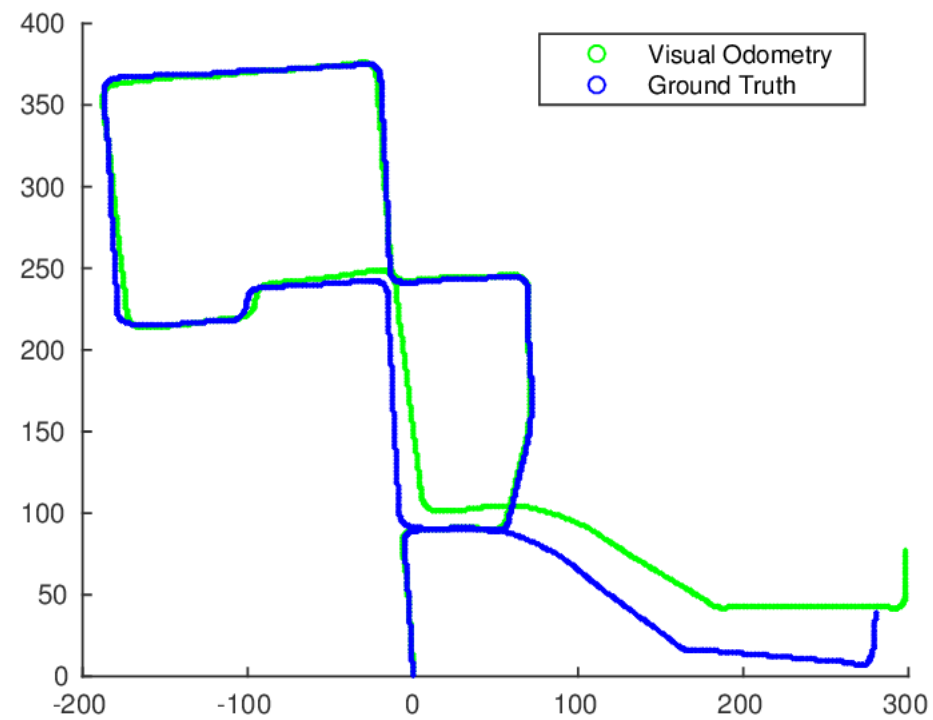
- **Errores ajenos/aleatorios/no controlables** al estudio y que **tienden a 0 al aumentar el estudio**
- Son errores inesperados y difíciles de estimar
  - Desnivel del suelo, suelos resbaladizos, fuerzas externas (viento), sobre/subaceleración debido a pendientes, ruedas en el aire, baches, etc.

- **Existen varios algoritmos** para estimar posiciones basados en la odometría

- Normalmente la posición esta «rodeada» de una **elipse de error**
  - Indica la región de incertidumbre para el robot
  - Cuanto más recorrido ande, mayor será la elipse
- **Físicos:** basados en los sensores/actuadores
- **Software:** basados en visión por computador

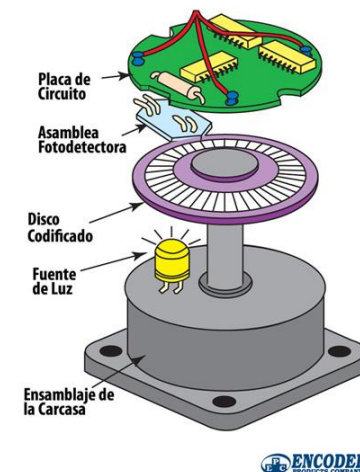
# Odometría III

- [https://youtu.be/homos4vd\\_Zs](https://youtu.be/homos4vd_Zs)
- <https://youtu.be/C6-xwSOOdqQ>



## Odometría IV

- Requiere recoger los datos del robot rápidamente y de forma precisa
- Algunos sensores pueden ayudarnos
  - **Codificador rotatorio/Encoder:** sensor que permite calcular velocidad, distancia y ángulo de giro usando luz y ranuras para obtener la posición angular de un eje
  - Existen «**encoders**» con distinta calidad, tasa de muestreo y funcionamiento (ópticos, magnético, mecánico, etc.)
  - Ejemplo: ratón de bola, ...



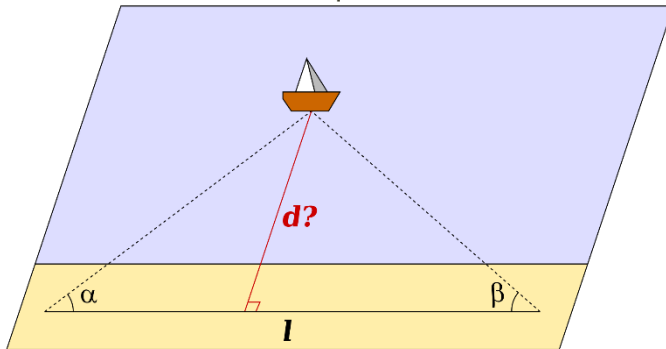
ENCODER  
PRODUCTS COMPANY

<http://encoder.com/blog/encoder-basics/que-es-un-encoder/>

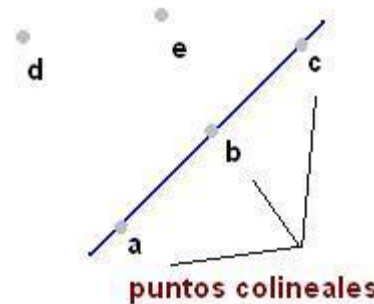
# Triangulación y trilateración

## Triangulación

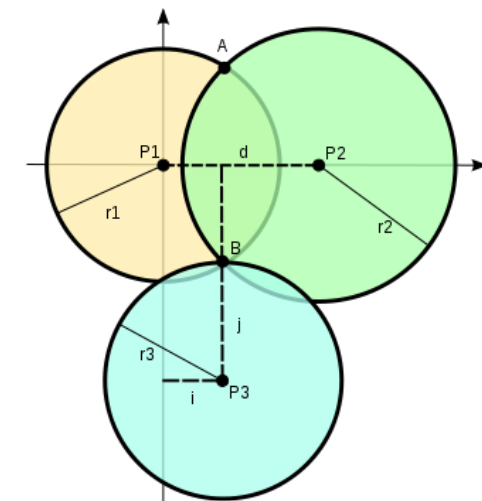
- Método de localización simple:** medidas de ángulos + al menos 1 distancia
- Usa las propiedades geométricas de los triángulos (trigonometría)** sobre las distancias entre el robot y los puntos de referencia
- Para GPS se utiliza **trilateración** (análogo): 3 distancias
  - No usa ángulos
  - Se necesitan tres puntos no colineales
  - Colineal:** que se encuentran en la misma recta



[https://es.wikipedia.org/wiki/Triangulaci%C3%B3n#/media/Archivo:Distance\\_by\\_triangulation.svg](https://es.wikipedia.org/wiki/Triangulaci%C3%B3n#/media/Archivo:Distance_by_triangulation.svg)



<https://definicion.de/puntos-colineales/>



<https://commons.wikimedia.org/wiki/File:Trilateration.svg>

# Balizas

- Consiste en **utilizar balizas que pueda leer el robot**
- Se aplica en **entornos restringidos y cerrados**
- **Pueden ser**
  - Códigos de barras, infrarrojos, Bluetooth, QR, GPS, etc.
- **Se utiliza triangulación o trilateración para calcular su situación**
- Ejemplos:
  - Aspiradoras: barreras infrarrojas
    - Impedir el paso
  - Cortacéspedes: cables metálicos bajo la tierra, código QR, balizas inalámbricas, GPS, ...
    - Restringir el área necesita de media entre 4 y 9 balizas
- Noticia:
  - Las radiofrecuencias pueden dar problemas de interferencia con la radioastronomía: cortacésped vs astrónomos
    - <https://www.microsiervos.com/archivo/gadgets/robot-segadora-indigna-astronomos.html>



## Localización probabilística

- **La localización** del robot estimada **se puede representar de forma probabilística**
  - Sobre todas las posibles localizaciones del robot
- Inicialmente **el robot no sabe en que posición está**
  - **Al observar un punto de referencia aumenta la probabilidad** de que esté en ciertas posiciones
    - Una puerta, una baliza, etc.
  - **Cuando se mueve, observa otros puntos de referencia**
    - Las probabilidades van cambiando

# Markov I

- **Requisitos**

- El robot conoce el mapa
- Sabe a donde tiene que ir
- No sabe donde está, ni como llegar al objetivo (¿secuestro?)

- Utiliza un **modelo de decisión para ver a que punto le conviene moverse**


- **En cada movimiento** que realiza el robot **las probabilidades cambian**
- **Se sigue moviendo para saber donde está y poder situarse** en el mapa

- **Cuando se sitúa en un punto con probabilidad muy alta**

- Asume que se encuentra en ese punto del mapa
- **Se localiza así mismo en el mapa**
- **Traza la trayectoria de salida**

## Markov II – Ejemplo simple

- <https://www.youtube.com/watch?v=2W1ApfksqM>

$P(x)$	$P(x)$	$P(x)$	$P(x)$
$P(x)$		$P(x)$	$P(x)$
$P(x)$	$P(x)$	$P(x)$	$P(x)$
$P(x)$	$P(x)$	$P(x)$	$P(x)$

1	0	0	1
0	0	0	0
1	0	1	0
0		0	0

Preguntas tema

○ <https://forms.office.com/r/nSQdjTzG06>



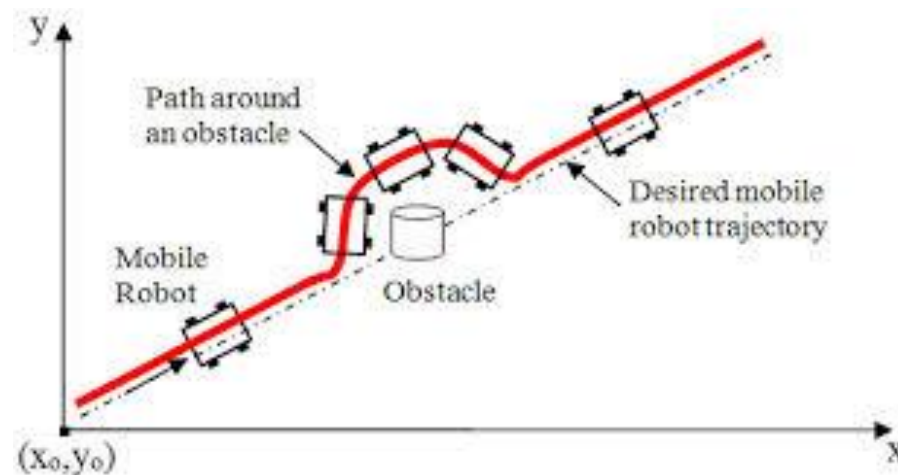
# Trayectorias

# Trayectoria I

- Es **la manera de llegar a un punto**
  - Ya sea **de alguna manera**
  - **O de la mejor manera posible**, la más óptima en base a lo requerido
- Secuencia de **configuraciones que el robot debe hacer**
  - Inicio – N Configuraciones – Fin
- **La trayectoria es continua**
- **El orden de las configuraciones suele ser muy importante**
- Igual se requiere conocer el mapa previamente o haberlo reconocido
  - Ejercicios del laberinto

## Trayectoria II

- **Se busca la mejor trayectoria, dependiendo de los criterios**
  - Camino libre de obstáculos
  - Minimizar el coste de distancia, tiempo, etc.
  - Zonas más seguras o con menos posibilidades de obstáculos
- Esto sirve para cualquiera: coches, aspiradoras, cortacésped, ...
- **Es un problema de búsqueda y optimización**



# Trayectoria III – Algoritmos

- o **Algoritmos simples**

- o *Algoritmos BUG (pulga)*
- o *Métodos Wavefront*

- o **Planes de trabajo (modelos)**

- o *Mapas de Meadow*
- o *Gráficos de visibilidad*
- o *Descomposición en celdas*
- o *Gráficos generalizados de Voronoi*

- o **Algoritmos basados en muestreo**

- o *Planes de trabajo probabilísticos*
- o *Exploración de árbol aleatorio*
- o *Variantes*
  - o *Estrategias de muestreo*
  - o *Estrategias de conexión*
  - o *Algoritmos perezosos*
  - o *Postprocesamiento*

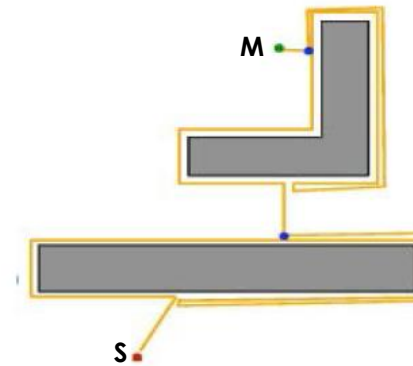


# Algoritmos BUG I

- **Asumen solo el conocimiento local y un objetivo global**
  - El robot **utiliza sensores** para detectar obstáculos
- **Comportamientos simples**
  - Seguir muro (derecha o izquierda)
  - Moverse en línea recta hasta la meta
- **Suposiciones**
  - Número finito de obstáculos
    - Dirección conocida del objetivo
  - Una línea cruza un obstáculo un número finito de veces
  - El espacio de trabajo (mapa) es finito

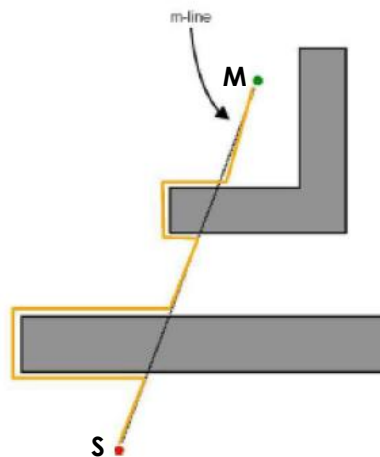
# Algoritmo BUG 1

- **Dirigirse hasta la meta**
  - Si hay un obstáculo, lo circunvala entero
    - Búsqueda exhaustiva
  - Por cada punto de la circunvalación del obstáculo
    - Calcula la distancia con la meta
    - Recuerda el punto más cercano a la meta
- Al acabar de **circunvalar el objeto**
  - Se vuelve al punto que estaba más cerca de la meta
    - Vuelve por la pared
- A partir de ahí **continúa hasta encontrar otro obstáculo**
  - Realiza los pasos anteriores



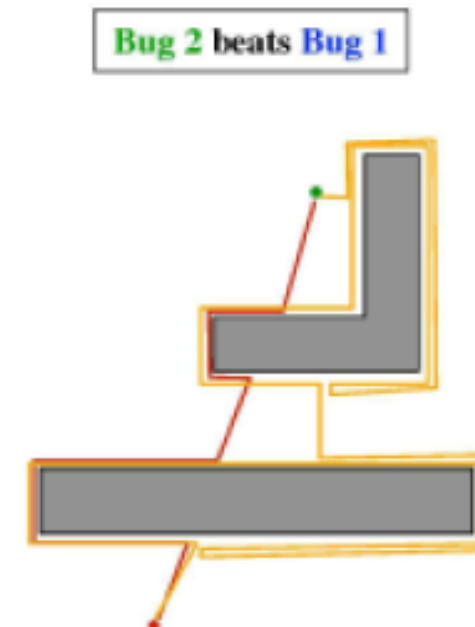
## Algoritmo BUG 2 (m-line)

- **Trazo una línea m-line**, desde el punto de inicio a la meta
- Si hay un **obstáculo en el camino**
  - Lo circunvala hasta encontrar un punto de la m-line
  - Deja de circunvalar el objeto
  - Sigue la m-line hasta encontrar otro obstáculo o la meta
    - Si es un obstáculo repite los pasos anteriores
- **Continúa hacia la meta**



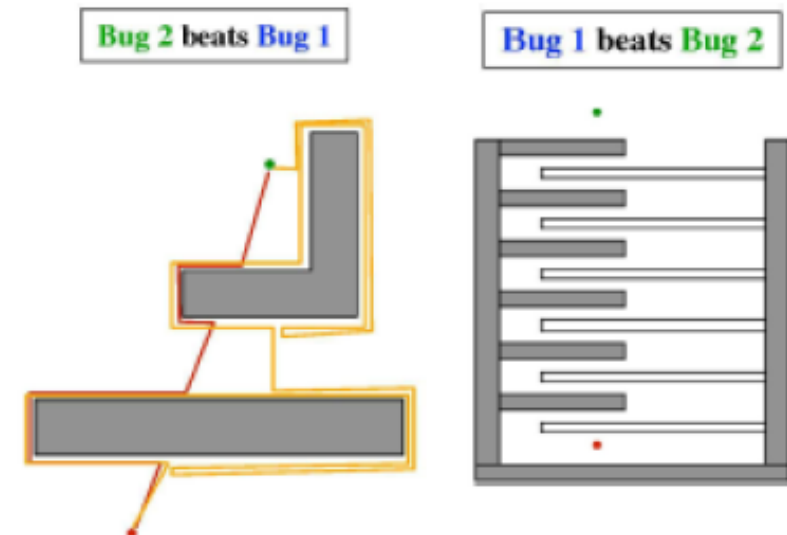
# Algoritmos BUG II

- Bug 1
  - **Búsqueda exhaustiva**
    - Examina todas las opciones antes de comprometerse
- Bug 2
  - **Algoritmo voraz**
    - Toma la primera decisión que ve mejor
    - Puede ser que no tome la adecuada o más óptima
- Bug 2 supera a Bug 1 en muchos casos
- Bug 1 tiene un rendimiento más predecible



# Algoritmos BUG II

- Bug 1
  - **Búsqueda exhaustiva**
    - Examina todas las opciones antes de comprometerse
- Bug 2
  - **Algoritmo voraz**
    - Toma la primera decisión que ve mejor
    - Puede ser que no tome la adecuada o más óptima
- Bug 2 supera a Bug 1 en muchos casos
- Bug 1 tiene un rendimiento más predecible
  - En este caso
    - Bug 2 puede que haga un bucle en la zona exterior y nunca salga
      - Si va siempre a la izquierda o derecha
    - Bug 1 recorrerá todo, pero llegará



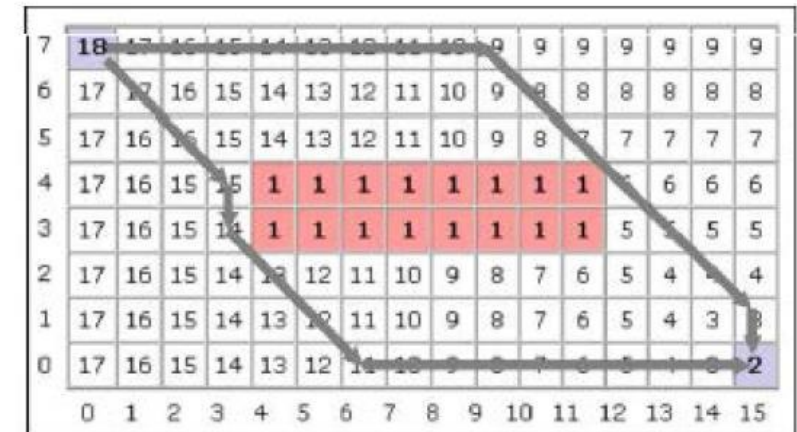
# Ideas básicas

- Se debe **ejecutar**
  - El comportamiento de **movimiento hacia el objetivo** cuando el camino esta «espejado»
  - **Si se encuentra un obstáculo** hay que **encontrar un «punto limite»** alrededor del obstáculo
    - **Punto limite**
      - La meta es visible
      - Debe ser el punto con menor distancia hacia el objetivo
- **El objetivo** puede coincidir con **la meta final o ser una submeta**
- Tener en cuenta
  - ¿Importan el número de giros?
    - Puede hacer que sea más lento o gaste más

# Wavefront I

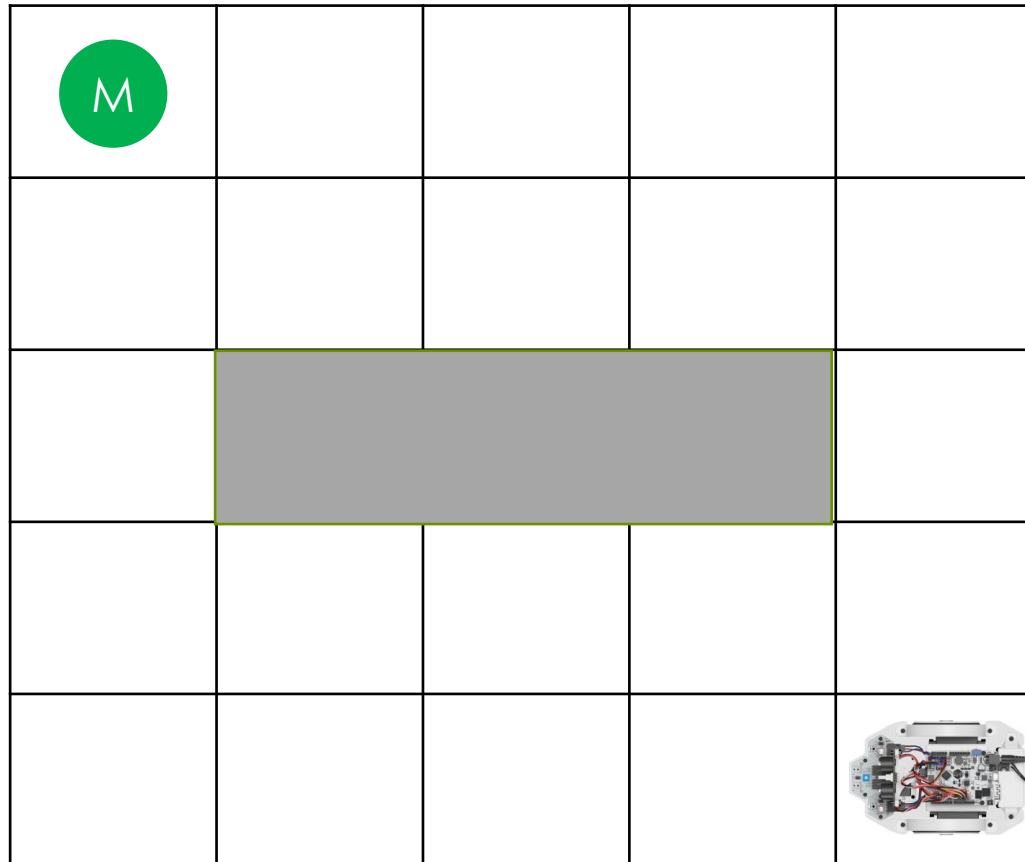
## Uso

- Resolver trayectorias cuando contamos con todo el mapa
  - ¿Permitiremos diagonales? Depende del robot e implementación
- Representa internamente el mapa como una matriz
- Da un código a **cada celda**: **distancia con el objetivo**
  - Objetivo (número menor)
  - Robot (número mayor)
- Se representan los **obstáculos** con un **1**, **meta** el **2**
- La trayectoria **sigue los números en orden decreciente**



## Wavefront II – Ejemplo I


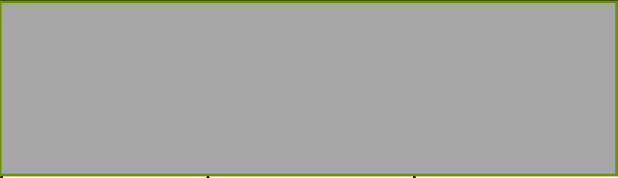

- Representa internamente el mapa como una **matriz**





## Wavefront II – Ejemplo II

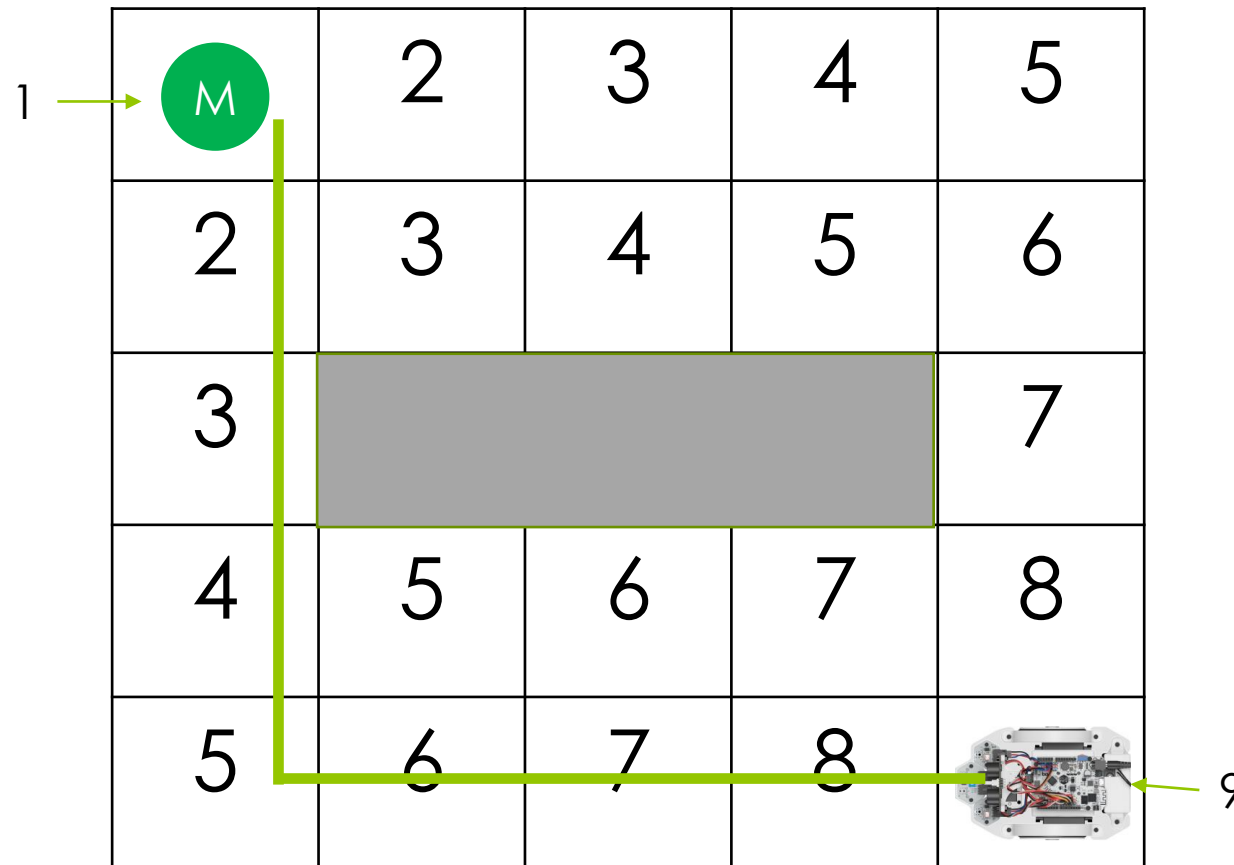
- Numeramos sin diagonales

1 →		2	3	4	5
	2	3	4	5	6
	3				7
	4	5	6	7	8
	5	6	7	8	

9 →

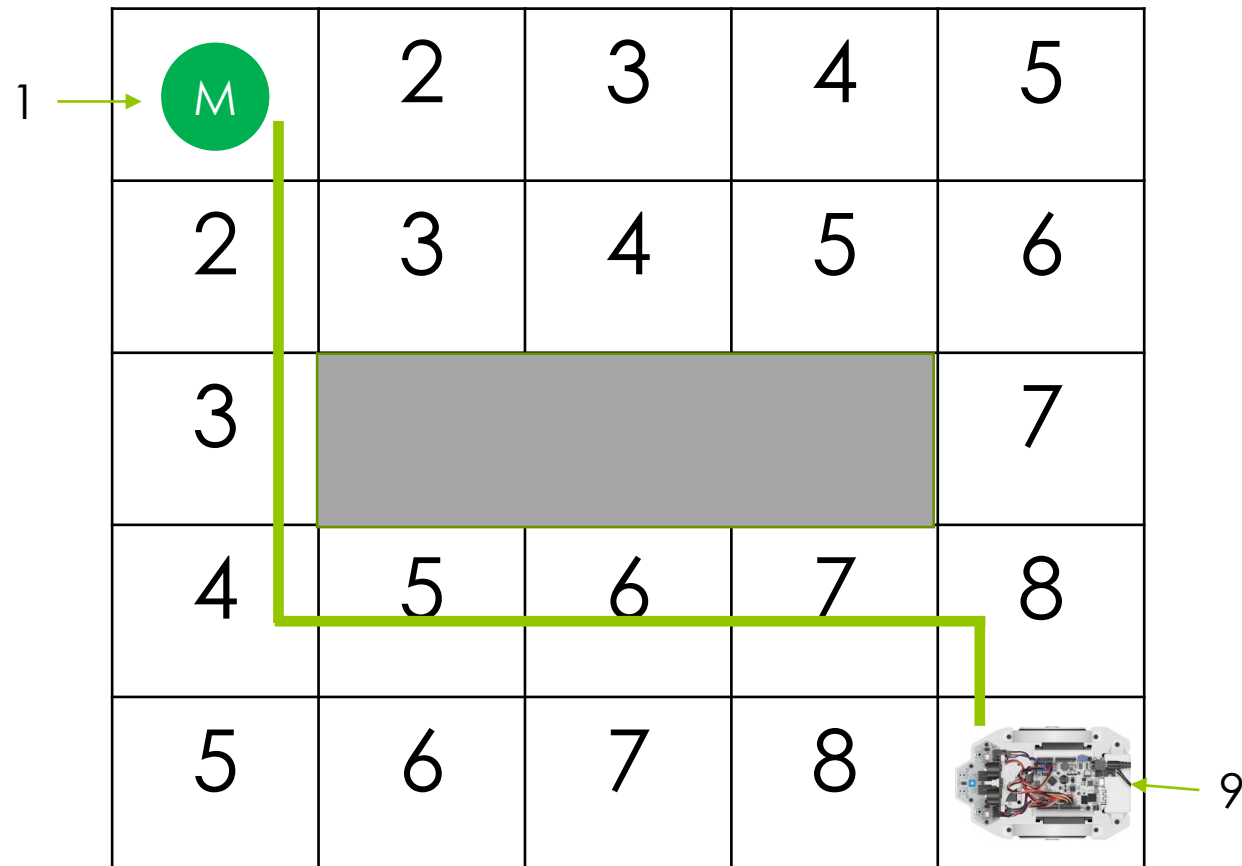
## Wavefront II – Ejemplo III

- Curvas rectas: opción 1



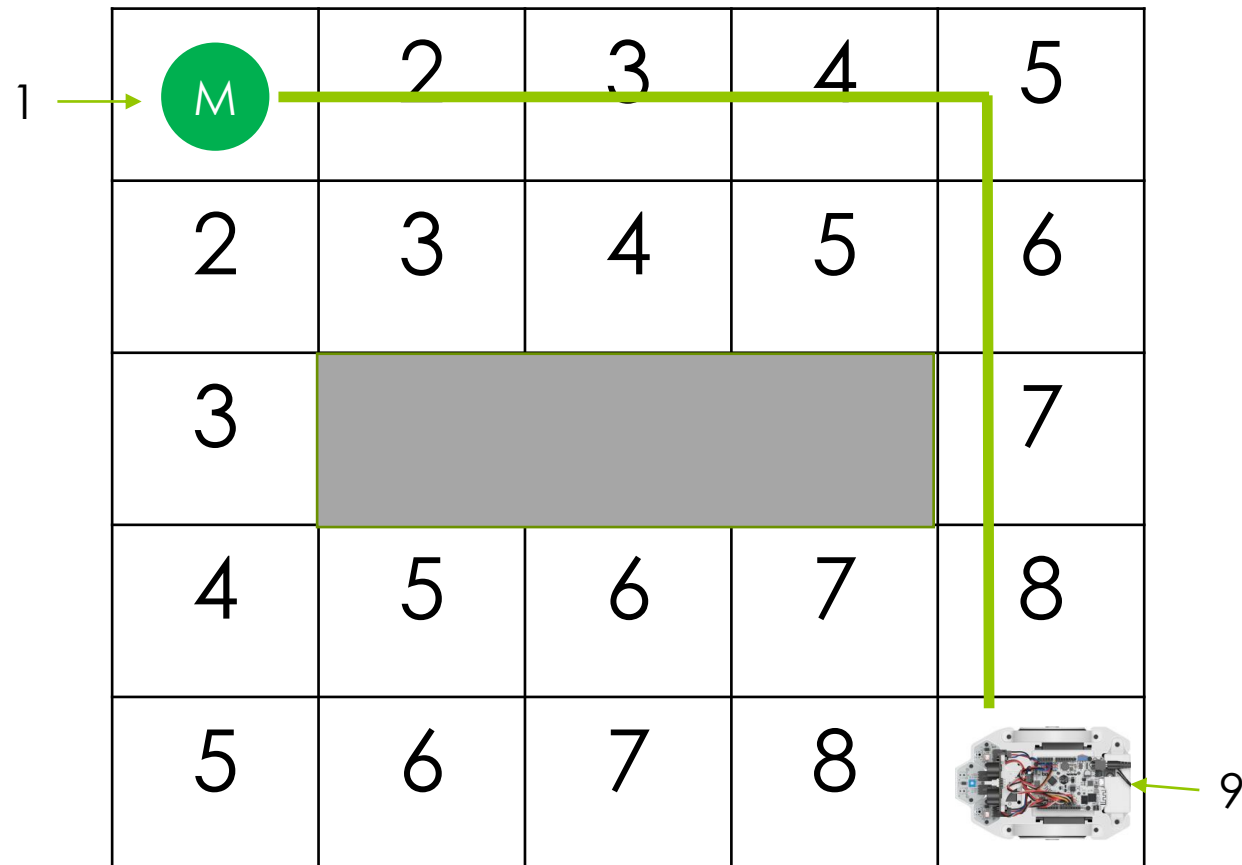
## Wavefront II – Ejemplo IV

- Curvas rectas: opción 2



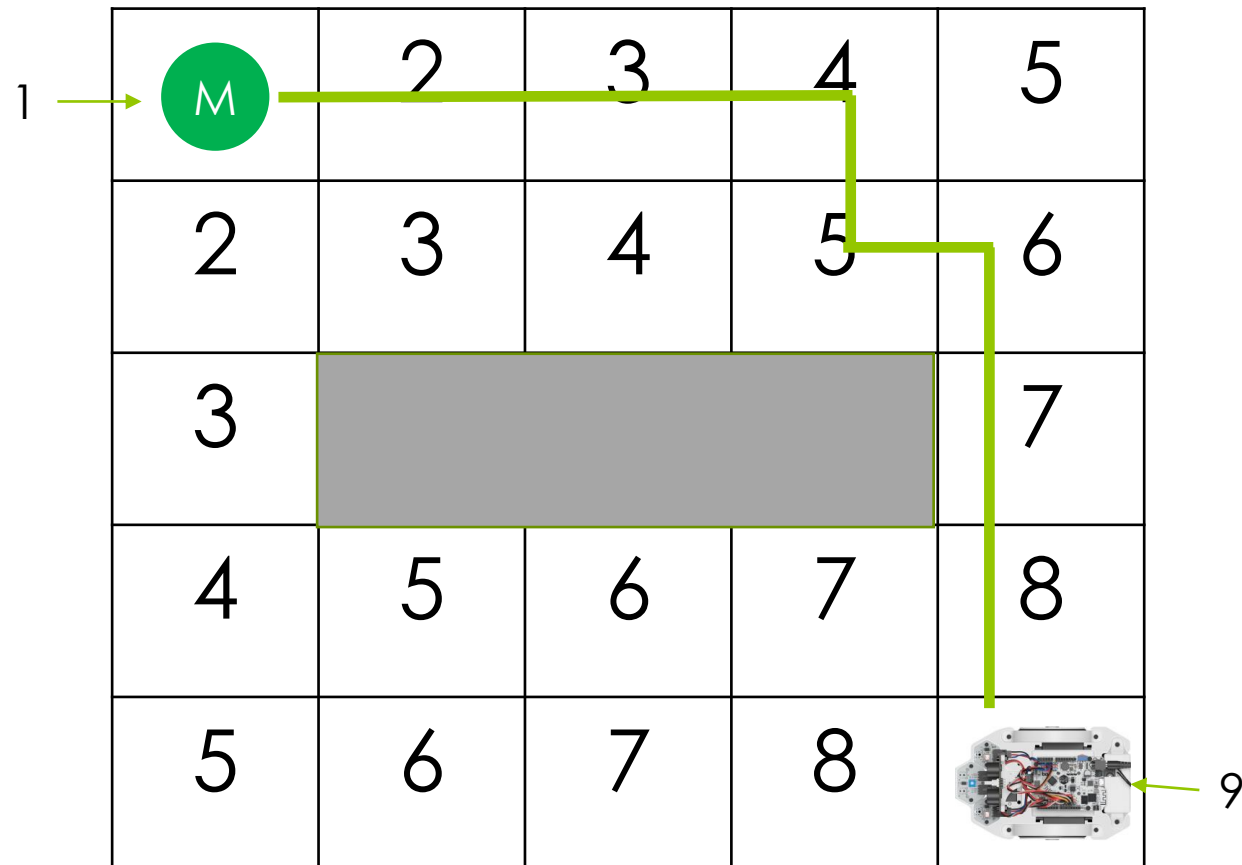
## Wavefront II – Ejemplo V

- Curvas rectas: opción 3



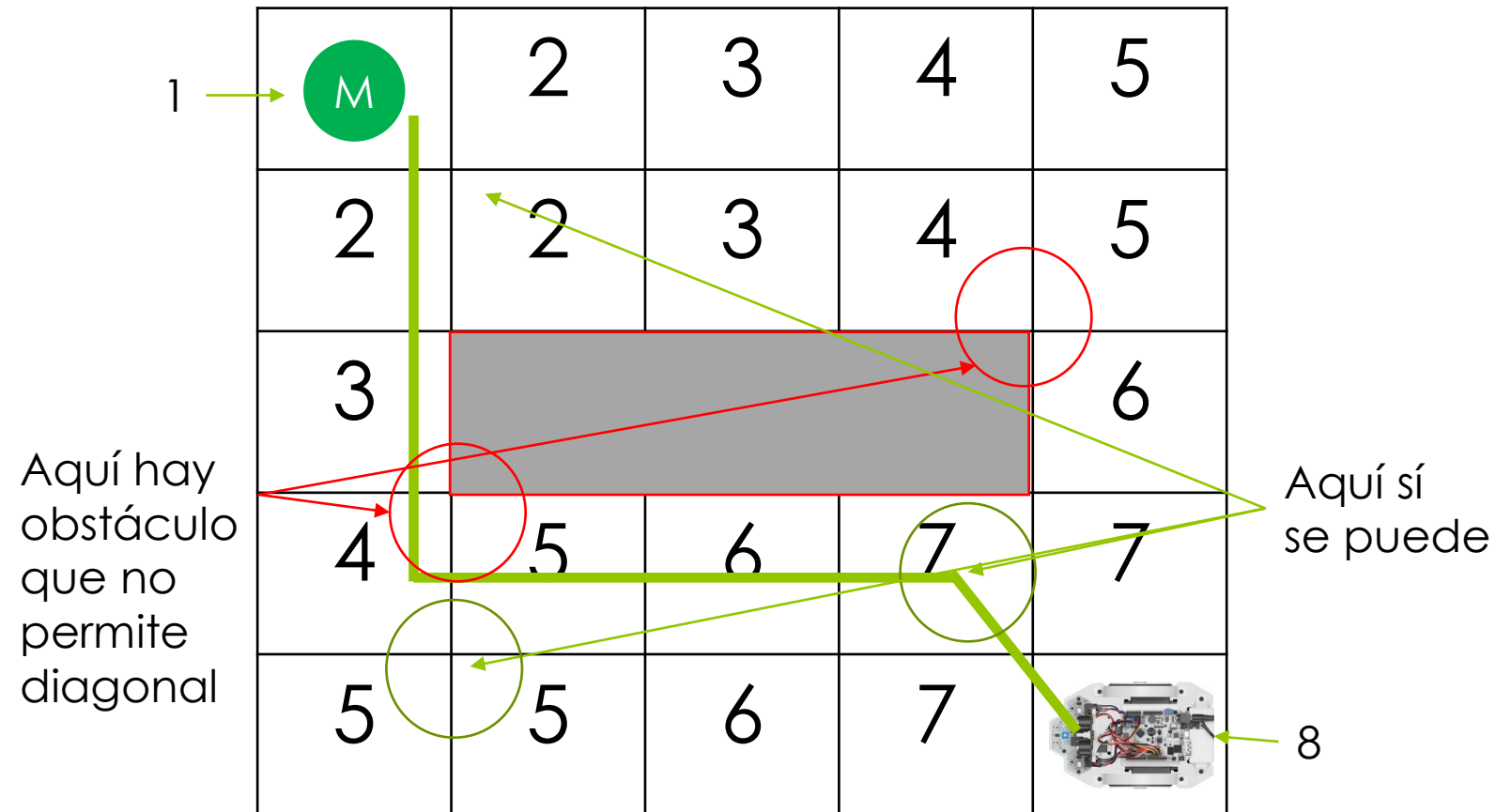
## Wavefront II – Ejemplo VI

- Curvas rectas: opción 4



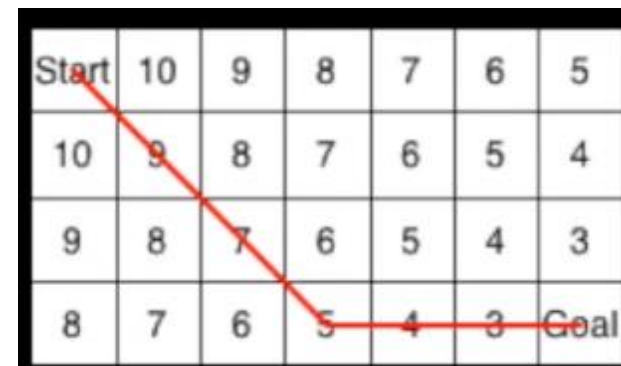
## Wavefront II – Ejemplo VII

- Curvas en diagonal cuando pueda (cambia numeración): reduce 1 paso



## Wavefront III

- Si salen varios caminos, se toma el más eficiente
  - **Voraz**: la primera solución útil y razonablemente adecuada
  - Búsqueda en base a diferentes criterios
    - Operaciones más costosas que otras
      - Cruzar en diagonal más costoso que moverse en línea recta debido a como gira el robot
      - O igual hay diagonales imposibles de hacer por el obstáculo
    - Posibles obstáculos dinámicos
    - Excesivos giros
    - Etc.
  - **Asignación de pesos a las operaciones**



# Wavefront IV

- **Uso**

- Resolver trayectorias cuando contamos con todo el mapa

- También **se podría ir actualizando dinámicamente**

- **Al detectar obstáculos** que no aparecían en el mapa inicial
    - Recalculamos, elegimos y seguimos
  - **Conocemos el mapa, pero no los elementos que se «mueven» por él**
    - Recalculamos, elegimos y seguimos

- Ejemplos

- Dinámico
    - <https://youtu.be/wilBZb6MpHI>
  - Optimización (no numera toda la matriz)
    - <https://youtu.be/yeL7ICc8g4A>





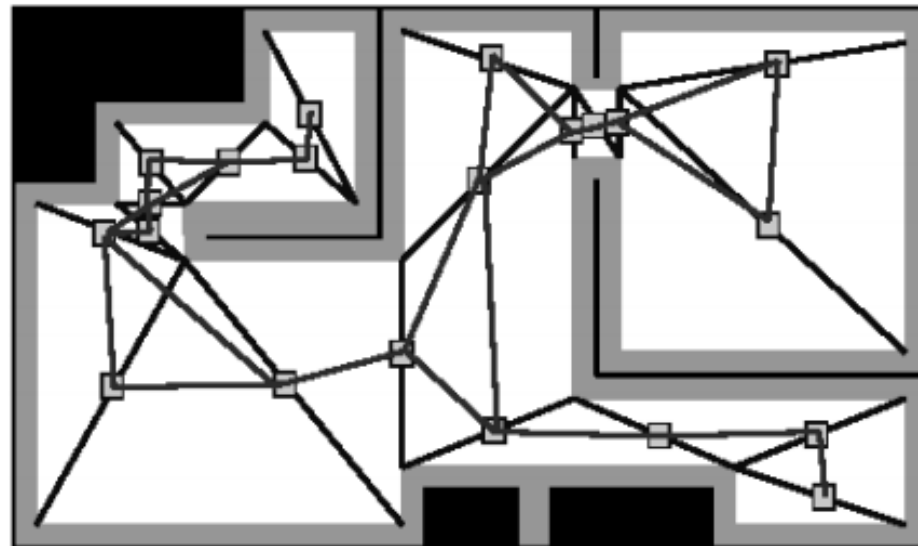
## Planes de trabajo (modelos)

# Modelos

- Son **representaciones de objetos del entorno**
  - Cuadrículas (Grid), líneas, polígonos, etc.
- Se utilizan en labores de localización
- Principal uso
  - Localización y planificación de caminos
    - En Wavefront sabríamos si podemos o no dar una diagonal
  - En base al modelo se ejecutan reglas lógicas y algoritmos

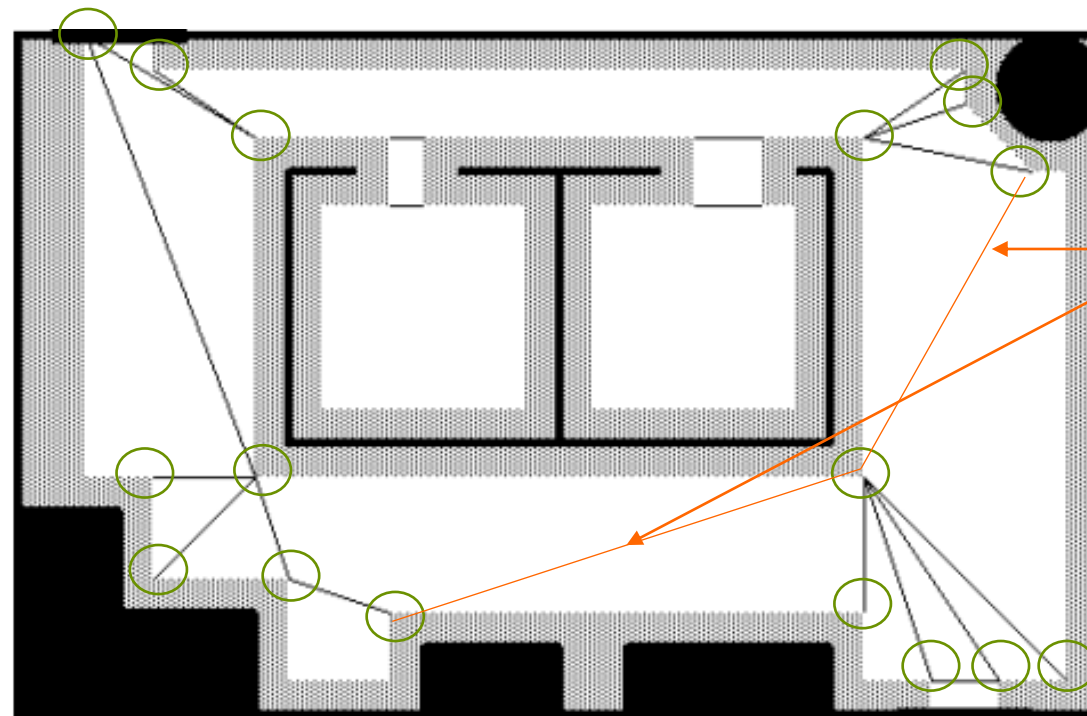
## Mapa de Meadow I

- ◉ **Combina vértices y representaciones de espacio libre**
  - ◉ Encuentra las esquinas de los objetos y conectarlas
  - ◉ Usa el centro de estos bordes como puntos de referencia
- ◉ **La representación se basa en polígonos**
  - ◉ Puede ser computacionalmente **costoso**
  - ◉ El camino ideal es por el «medio» y **no es el más óptimo**



# Mapa de Meadow II – Pasos I

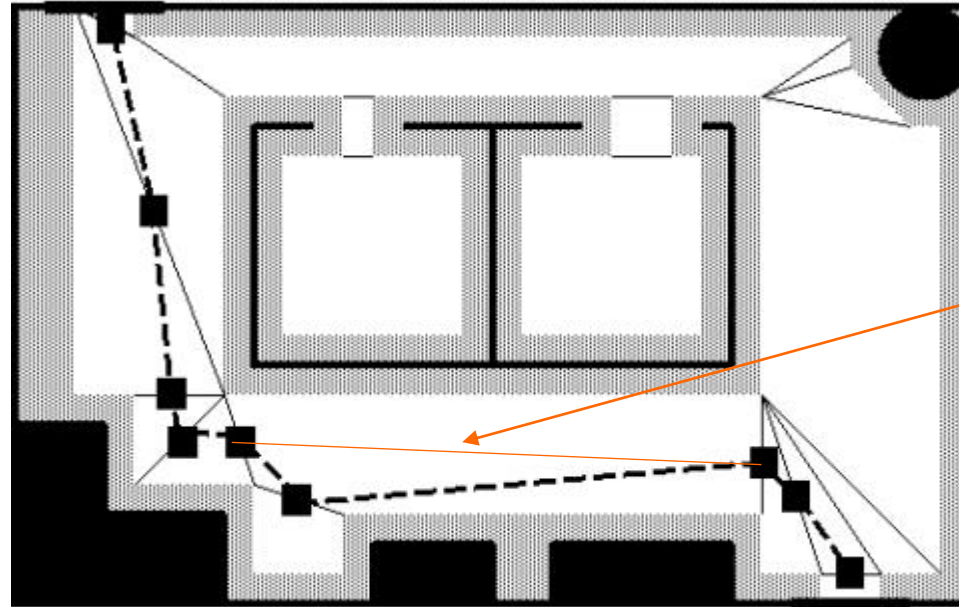
1. **Identificar segmentos de línea entre** pares de **esquinas**, puertas, vértices de obstáculos, ...
  1. Todos o la mayoría



Otras posibilidades

## Mapa de Meadow II – Pasos II

2. **Convertirlo en un grafo**
3. **Conectar los puntos medios** de las líneas
4. **Aplicar un algoritmo de búsqueda en el grafo**
  - Camino más corto, mayores probabilidades de llegar, etc.

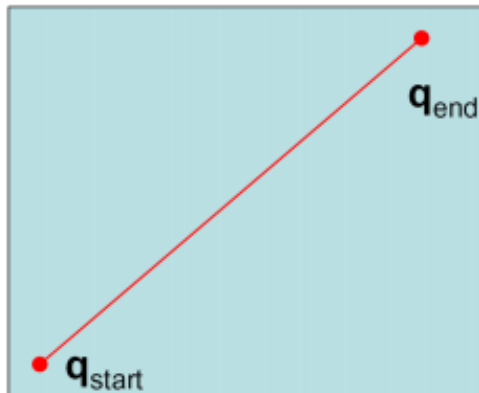


Esta es más óptima

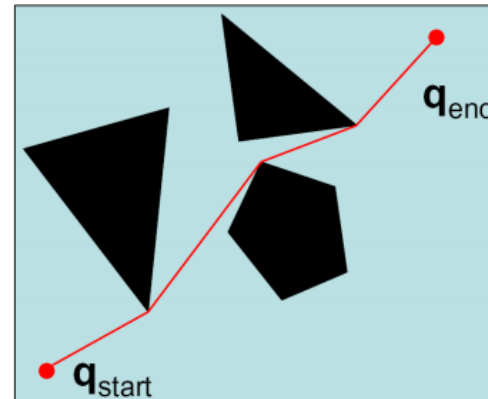
# Gráficos de Visibilidad I

## Conecta

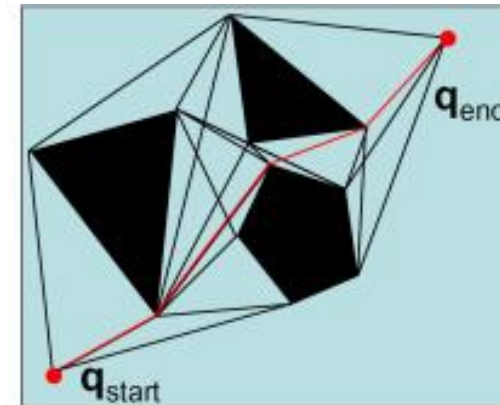
- Cada vértice de los obstáculos a los vértices visibles desde ese punto
  - Algunas conexiones son altamente improbables (heurístico)
- El punto inicial y la meta a los vértices visibles
  - El punto inicial y la meta se procesan como vértices



Resultado:  
mapa sin obstáculos



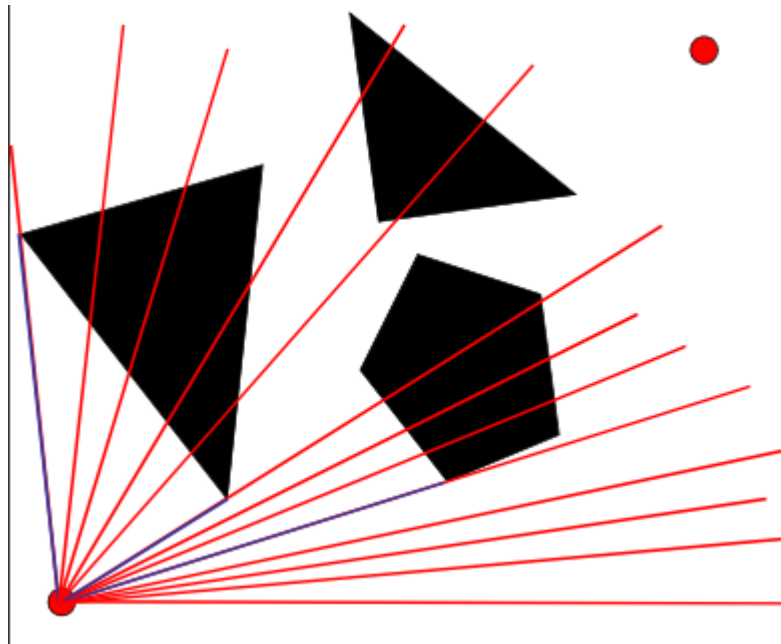
Resultado:  
con obstáculos



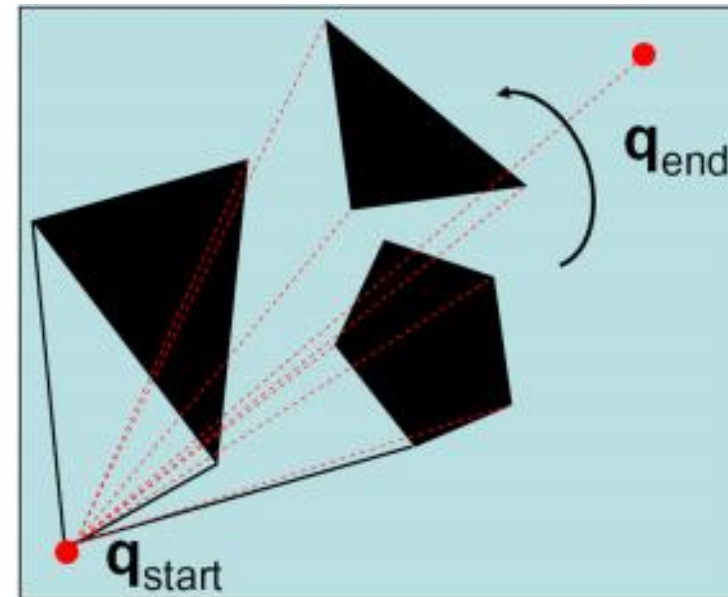
Resultado:  
camino elegido

## Gráficos de Visibilidad II

- Ejemplo: visibilidad y detección de vértices (3 vértices azul)



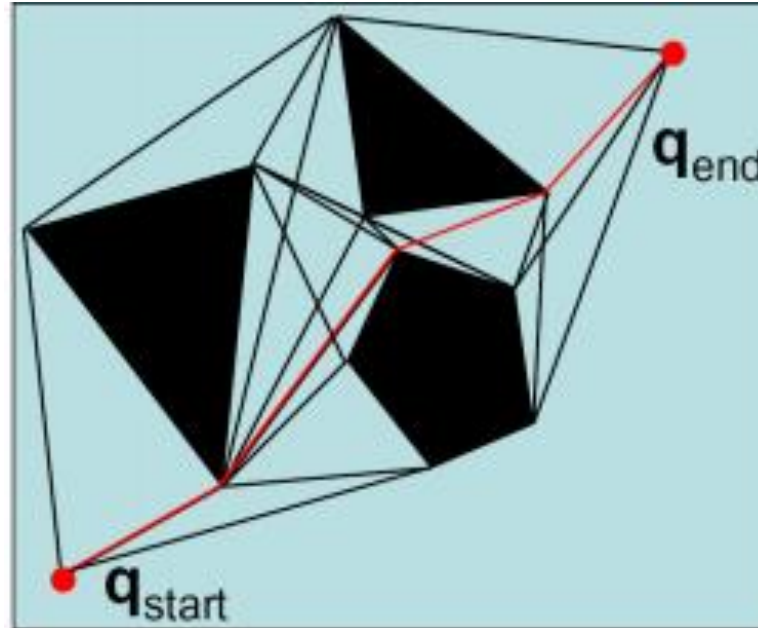
Visibilidad desde la salida



Detección de 3 vértices

## Gráficos de Visibilidad III

- Repetir el proceso desde cada vértice
- Cuando los tenemos todos
  - Exploramos posibles caminos desde el inicio hasta la meta
  - Búsqueda de caminos o algoritmo voraz

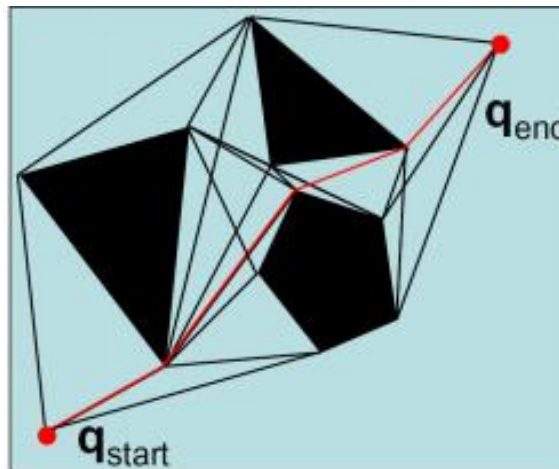




## Gráficos de Visibilidad IV

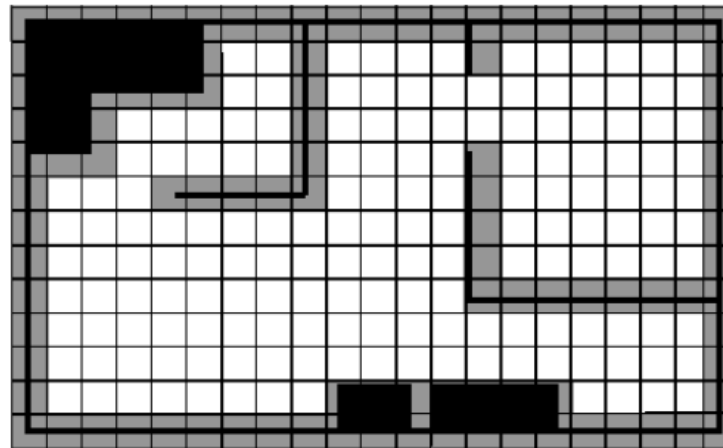
- **Debilidades**

- **Intenta ir lo más cerca posible de los obstáculos**
  - Cualquier error de cálculo puede llevar a una colisión
- **Se complica mucho si se superan las 2D**
  - 3D: altura del robot para evitar colisiones con salientes
- Es **más importante buscar un camino seguro** a uno más rápido/corto
  - Riesgo de colisiones/giros muy cerrados, etc.



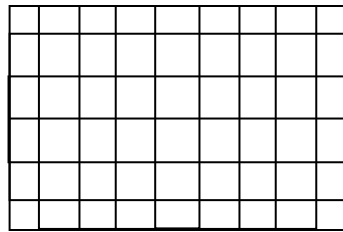
## Mapa de Grid I

- **La información se representa en un Grid** (matriz o otras estructuras)
  - Cada celda indica si esta libre o contiene algún tipo de **objeto** / obstáculo
  - Se puede expresar con 0/1 o con porcentajes de **ocupación** (no hay certeza de si hay o no hay)
  - **Puede ser inexacto**
    - Ej: celda con obstáculos parciales/pequeños o con sitio libre para pasar
    - Depende de la resolución de las celdas

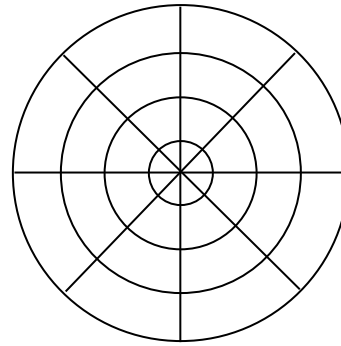


# Mapa de Grid II

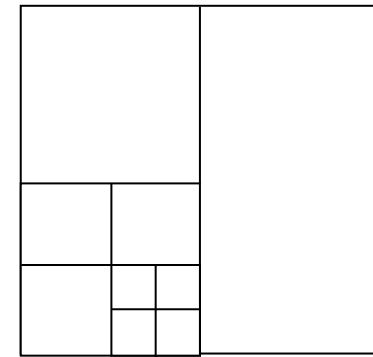
- Posibles tipos de Grids



Regular grid



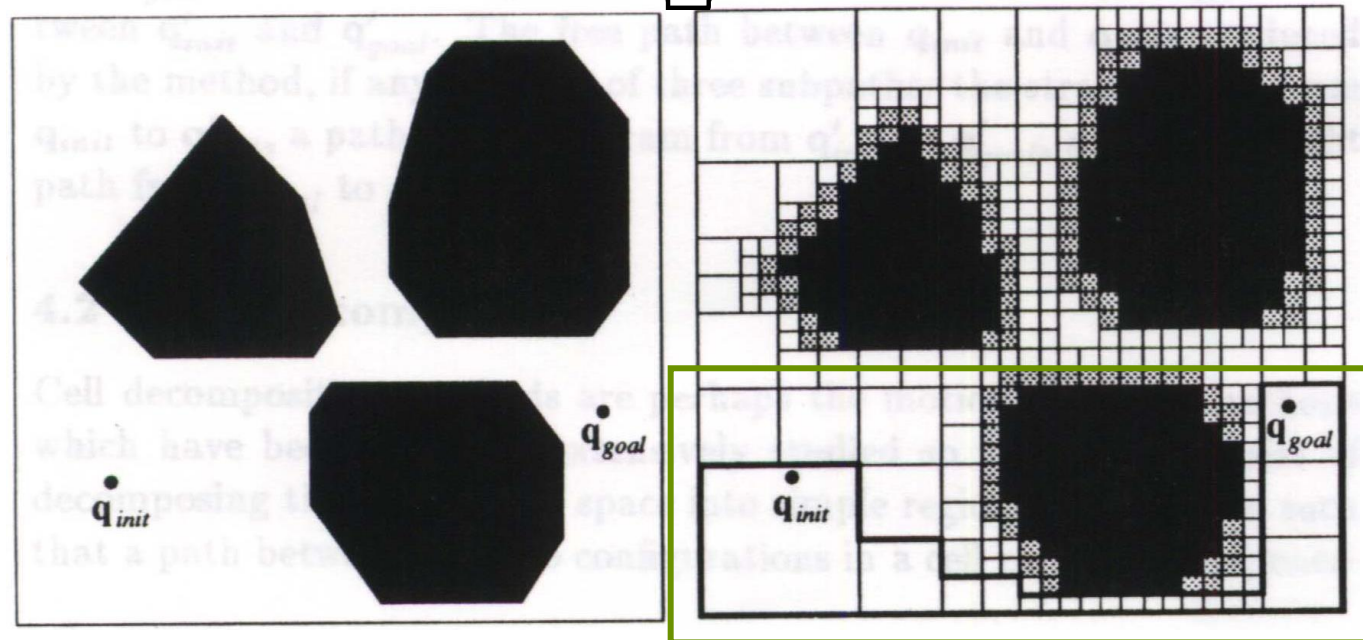
Sector grid



Quadtree

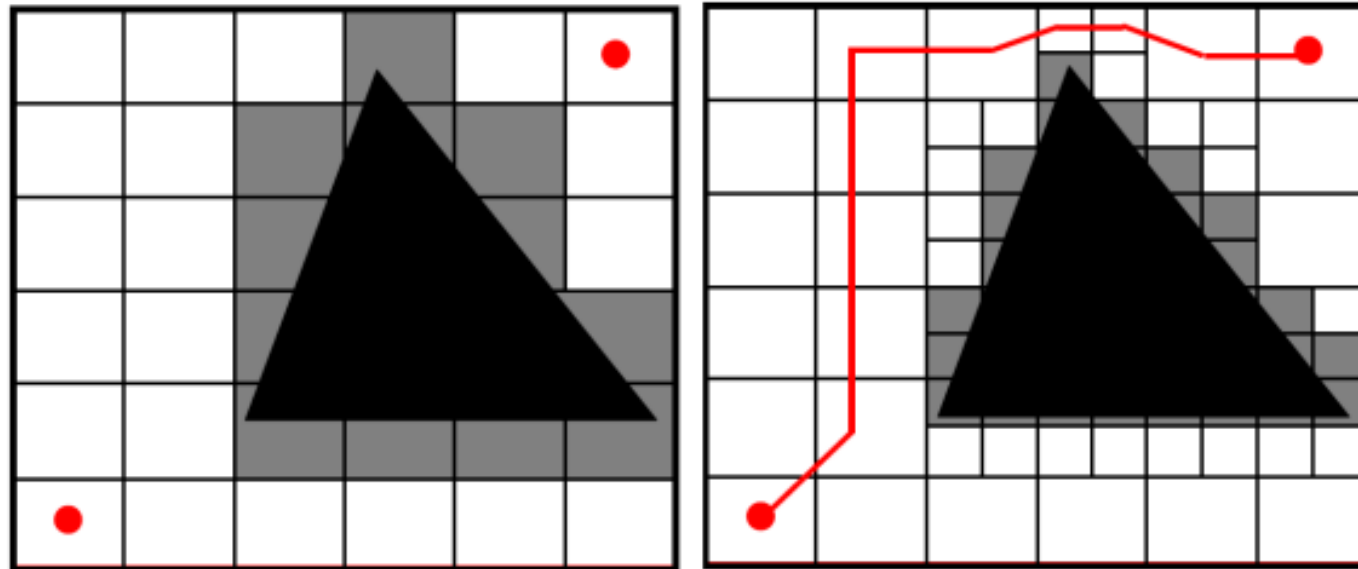
## Mapa de Grid III – Quadtree

- Ejemplo: Quadtree con pérdida de precisión
  - **Aumenta la precisión donde hay obstáculos para tener más información**
    - Celda pequeña: tamaño robot, precisión máxima de los sensores, etc.
    - Ejemplo: <https://youtu.be/XfIBhg6tq5c?t=106>



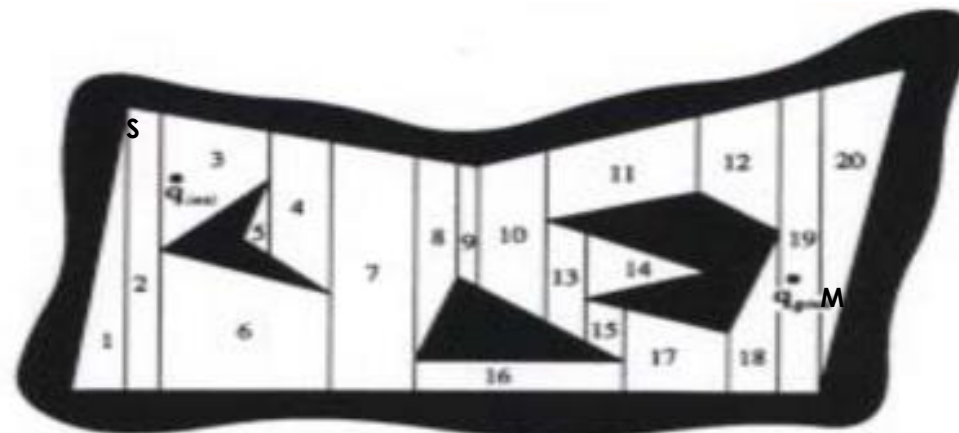
## Mapa de Grid IV

- Búsqueda de camino sobre el Grid
- **Cuanto más preciso sea el Grid**
- ✓ ○ Mejor será el camino
- ✗ ○ Más costoso de calcular y almacenar



# Descomposición Trapezoidal I

- **Descompone el espacio en celdas trapezoidales**
  - Se **parte de las esquinas** de los objetos/obstáculos
  - **Cada celda se numera**
  - El espacio queda dividido en trapecios (o triángulos)
- **El mapa representa un grafo de conexiones**
  - Sobre el que realizar operaciones y desplazarse entre ellos
- Patente: <https://patents.google.com/patent/US20120221237>



<https://patents.google.com/patent/US20120221237>

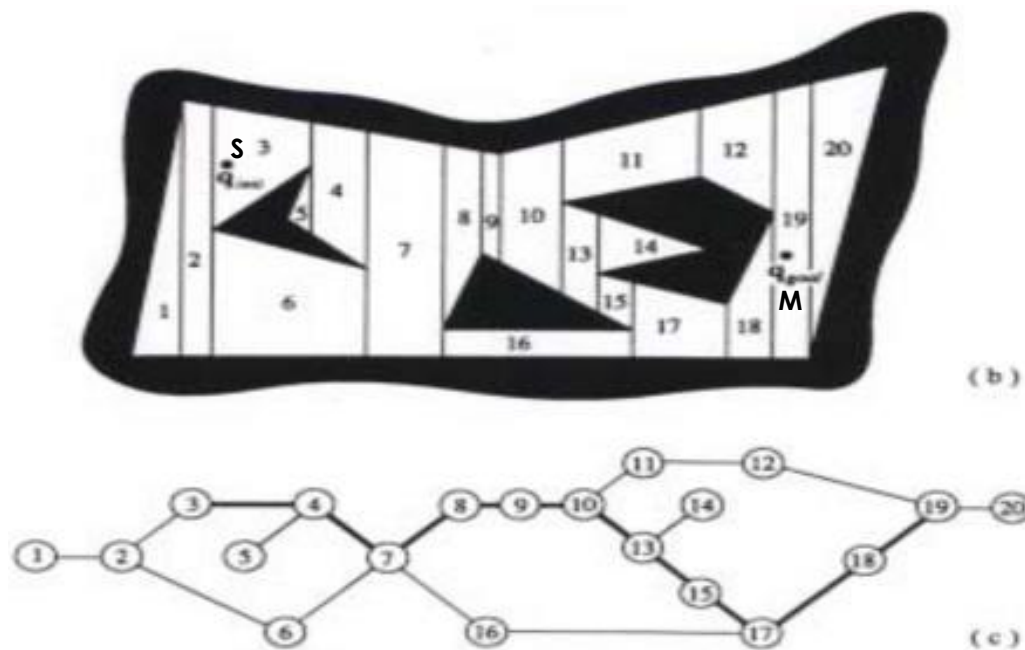
## Descomposición Trapezoidal II – Pasos para generar el mapa I

1. **Líneas verticales**
2. **Trazo de líneas verticales**, de abajo hacia arriba



## Descomposición Trapezoidal II – Pasos para generar el mapa II

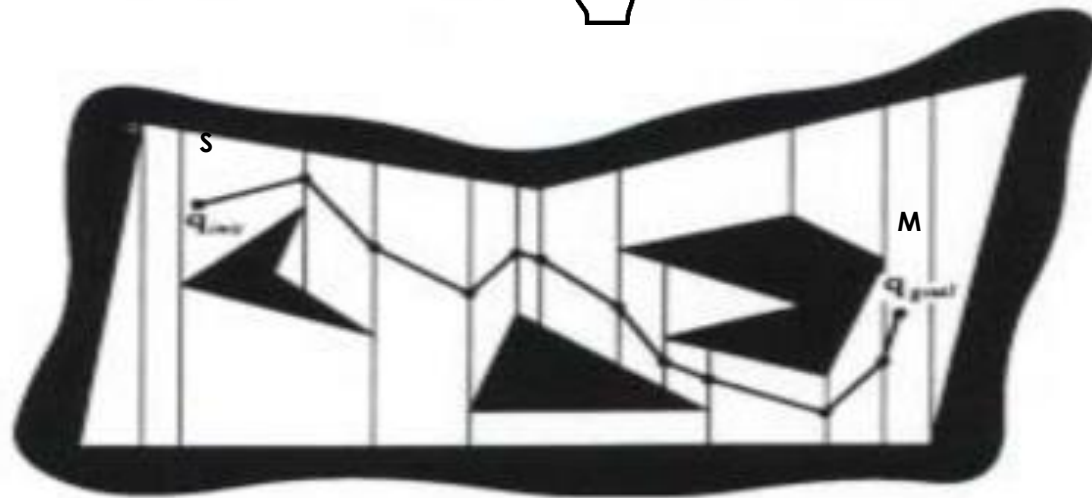
3. **Numeración** de celdas
4. **Generación del grafo** equivalente
5. **Búsqueda del camino** sobre el grafo, de la celda 3 a 19
  - No pasa por todos los nodos intermedios: 5, 11, 12





## Descomposición Trapezoidal II – Pasos para generar el mapa III

6. **Navegación** por las celdas indicadas
  6. Se pueden tener en cuenta distancias, giros de las esquinas, etc.
7. 3 – 4 – 7 – 8 – 9 – 10 – 13 – 15 – 17 – 18 – 19
- Ejemplo: <https://youtu.be/gZfw4BvbOzg>



Preguntas tema

○ <https://forms.office.com/r/3Y8dP64JYC>



# Localización y trayectorias

Cristian González García  
gonzalezcristian@uniovi.es

Material original de Jordán Pascual  
Espada

v 1.2.1 Noviembre 2022