Robótica Móvil
un enfoque probabilístico

Planeamiento de trayectorias

Ignacio Mas

Planeamiento de trayectorias

Latombe (1991):

"… fundamental ya que, por definición, un robot cumple tareas moviéndose en el mundo real."

Objetivos:

- Trayectorias sin colisiones.
- El robot debe llegar a destino lo más rápido posible.

… en entornos dinámicos

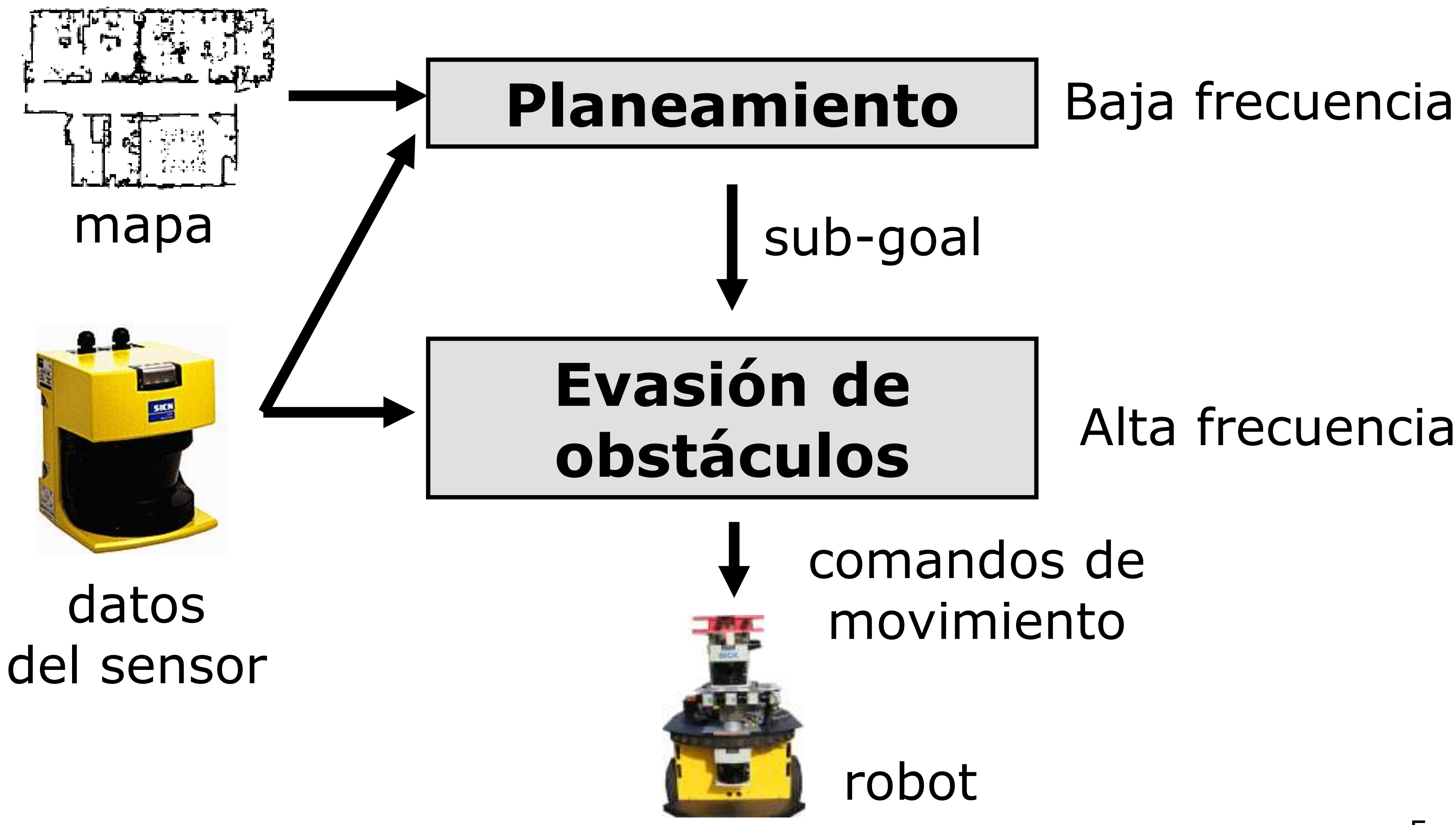
- Cómo reaccionar ante obstáculos inesperados?
  - eficiencia
  - confiabilidad
  - Método de ventana dinámica
    [Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]
  - Planeamiento basado en mapas de grilla
    [Konolige, 00]
  - Navegación por diagrama de cercanía
    [Minguez at al., 2001, 2002]
  - Histograma de campo vectorial+
    [Ulrich & Borenstein, 98]
  - A*, D*, D* Lite, ARA*, …

Dos desafíos

- Calcular el camino óptimo tomando en cuenta incertezas en las acciones

- Generar acciones rápidamente en caso de encontrar objetos inesperados

Arquitectura clásica de dos capas



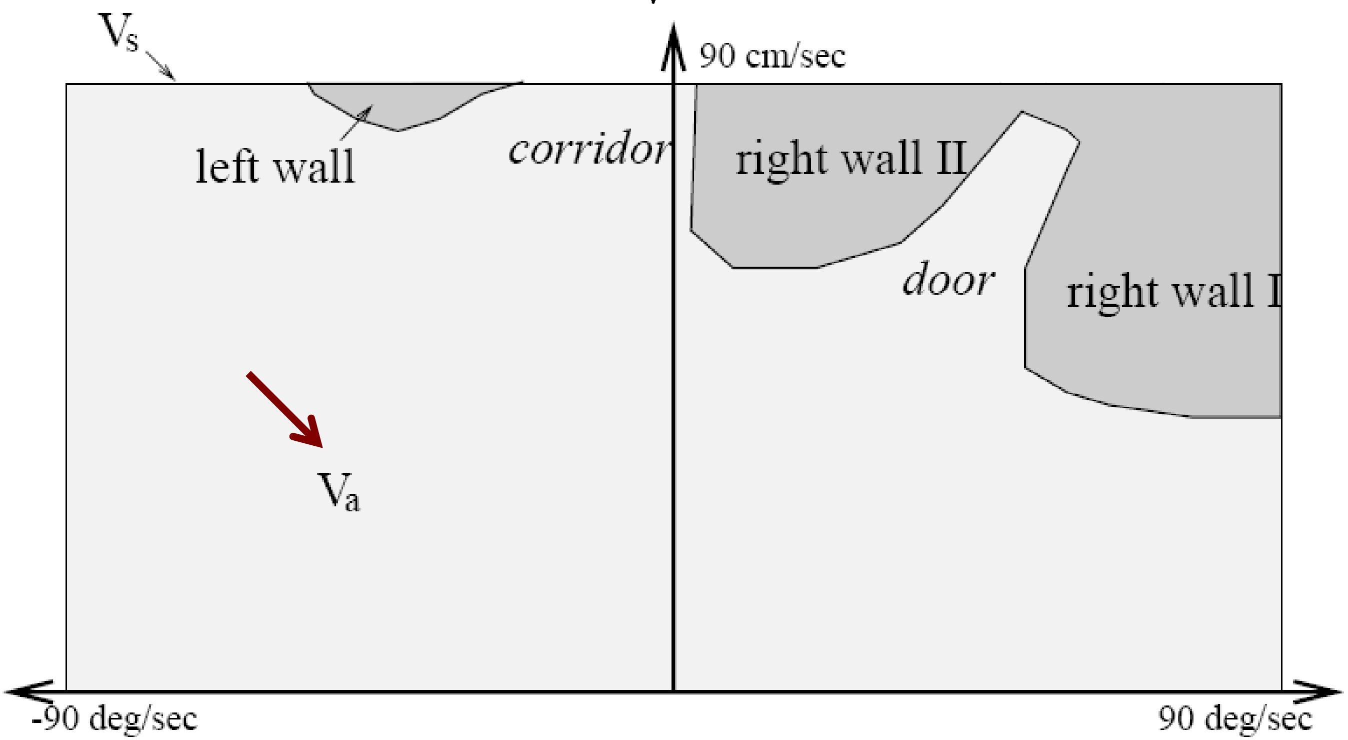
Método de ventana dinámica (DWA)

- **Evasión de obstáculos:** Determinar trayectorias libres de colisiones usando operaciones geométricas
- Ejemplo: el robot se mueve en arcos circulares
- Comandos de movimiento (v,ω)
- ¿Qué (v,ω) son admisibles y alcanzables?

Velocidades admisibles

- Una velocidad es admisible si el robot puede parar antes de llegar al obstáculo

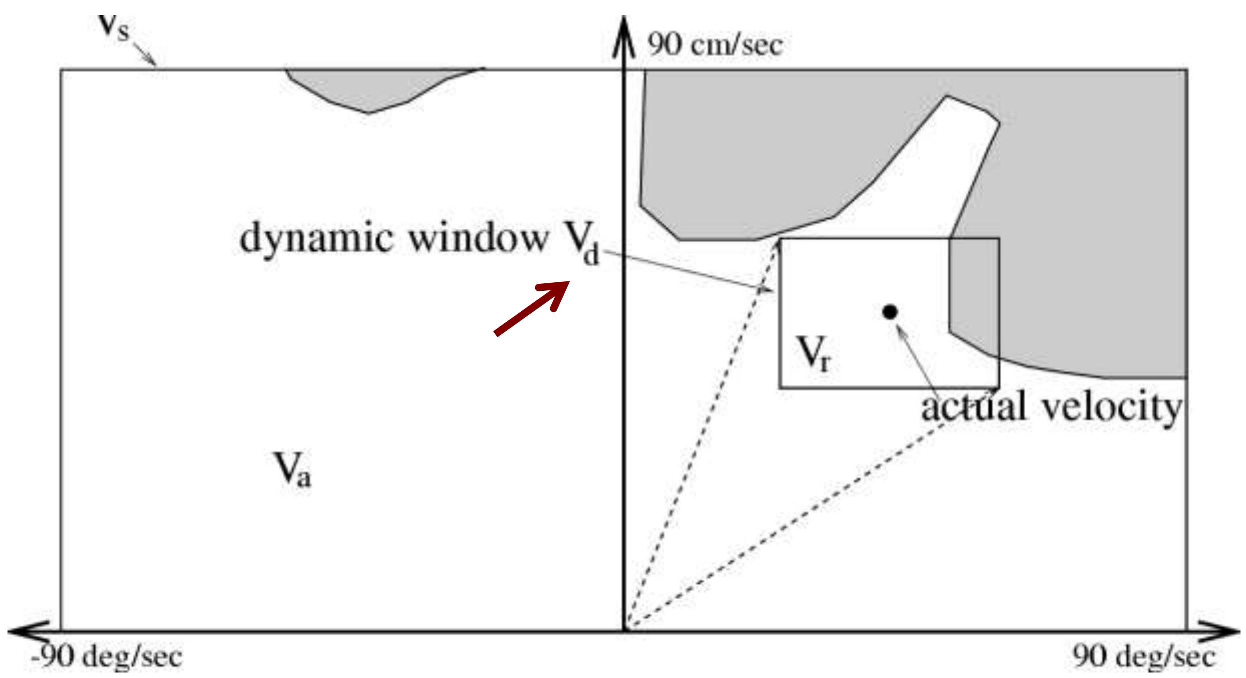
$$V_a = \{(v,\omega) \ \mid \ v \leq \sqrt{2\text{dist}(v,\omega)a_{trans}} \ \ \wedge \\ \omega \leq \sqrt{2\text{dist}(v,\omega)a_{rot}}\}$$



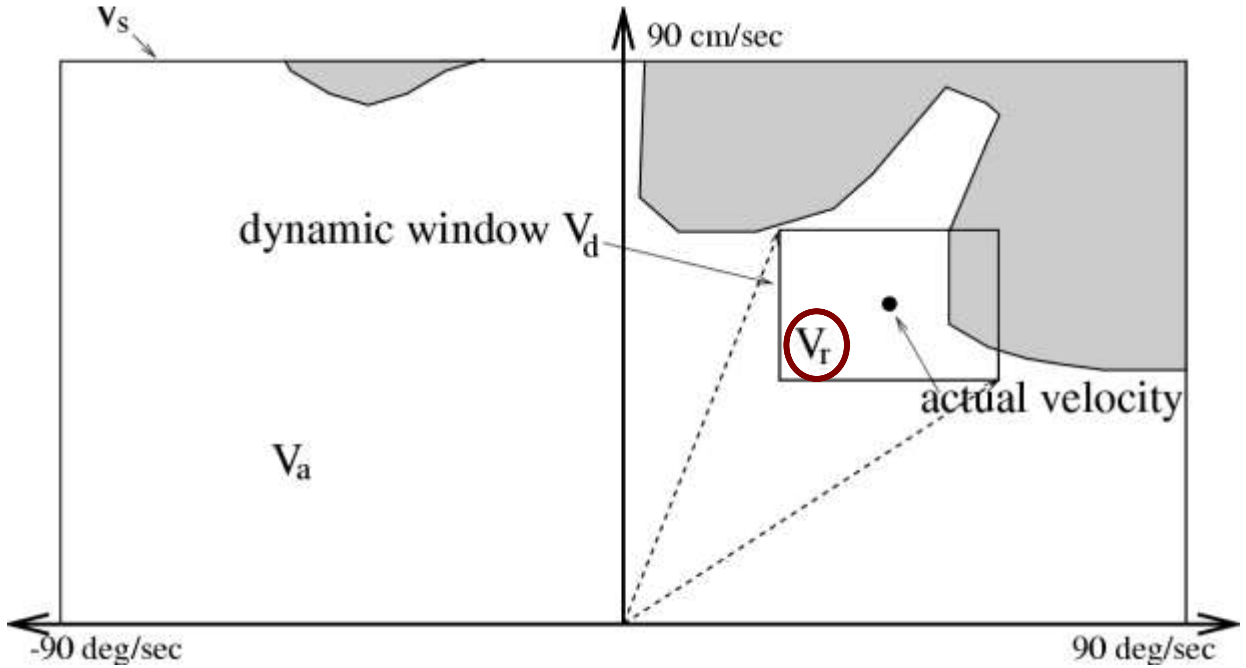
Velocidades alcanzables

- Velocidades que son alcanzables por aceleración

$$V_d = \{(v, \omega) \ \mid \ v \in [v - a_{trans}t, v + a_{trans}t] \ \wedge \\ \omega \in [\omega - a_{rot}t, \omega + a_{rot}t]\}$$



Espacio de búsqueda DWA



- V_s = todas las posibles velocidades del robot.
- V_a = área libre de obstáculos.
- V_d = velocidades alcanzables dentro de cierto tiempo según las aceleraciones posibles.

$$V_r = V_s \cap V_a \cap V_d$$

Método de ventana dinámica

- ¿Cómo elegir <v,ω>?
- Los comandos se eligen según una función heurística de navegación.
- Esta función trata de minimizar el tiempo de viaje "**yendo rápido** en la **dirección correcta**".

Método de ventana dinámica

- Función de navegación Heurística.
- Planeamiento restringido al espacio <x,y>.
- No en el espacio de velocidades.

Función de navegación *NF*: [Brock & Khatib, 99]

$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Método de ventana dinámica

- Función de navegación Heurística.
- Planeamiento restringido al espacio <x,y>.
- No en el espacio de velocidades.

Función de navegación *NF*: [Brock & Khatib, 99]

$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Maximiza la velocidad

Método de ventana dinámica

- Función de navegación Heurística.
- Planeamiento restringido al espacio <x,y>.
- No en el espacio de velocidades.

Función de navegación *NF*: [Brock & Khatib, 99]

$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Maximiza la velocidad

Considera costo de alcanzar objetivo

Método de ventana dinámica

- Función de navegación Heurística.
- Planeamiento restringido al espacio <x,y>.
- No en el espacio de velocidades.

Función de navegación *NF*: [Brock & Khatib, 99]

$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Maximiza la velocidad

Considera costo de alcanzar objetivo

Sigue camino basado en grilla calculado por A*.

Método de ventana dinámica

- Función de navegación Heurística.
- Planeamiento restringido al espacio <x,y>.
- No en el espacio de velocidades.

Función de navegación

Cercanía al objetivo

$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Maximiza la velocidad

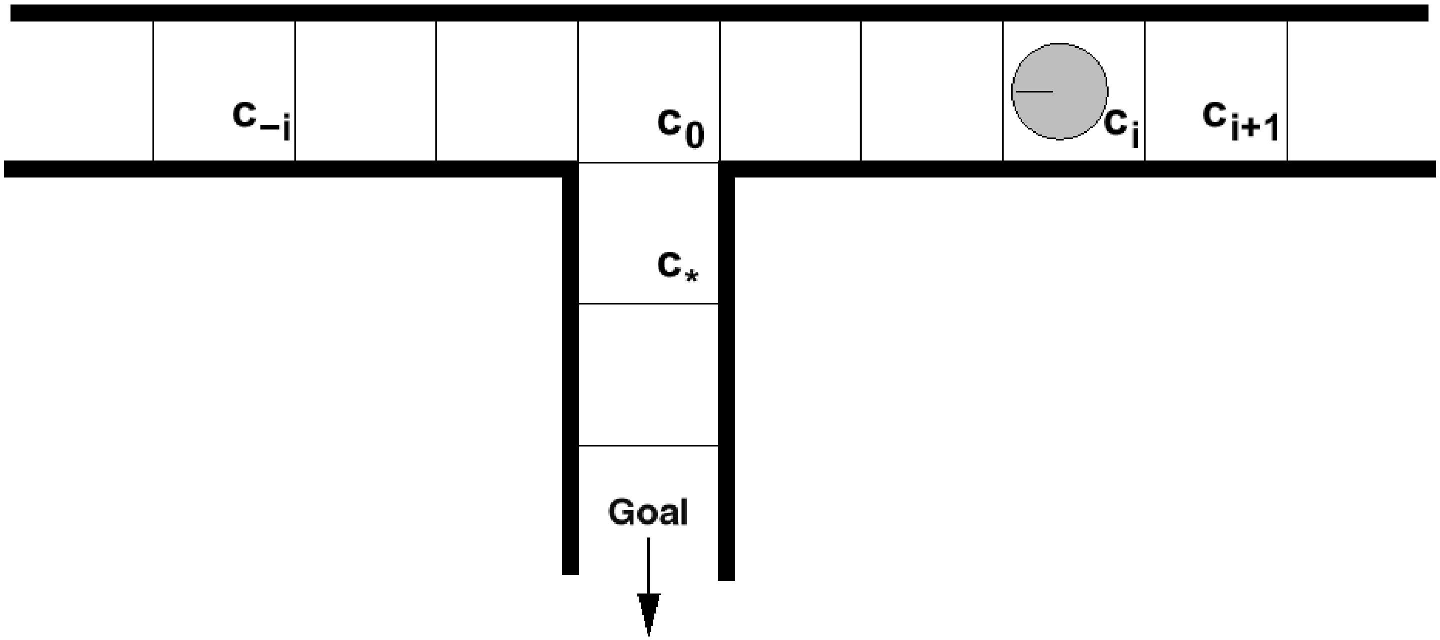
Considera costo de alcanzar objetivo

Sigue camino basado en grilla calculado por A*.

Método de ventana dinámica

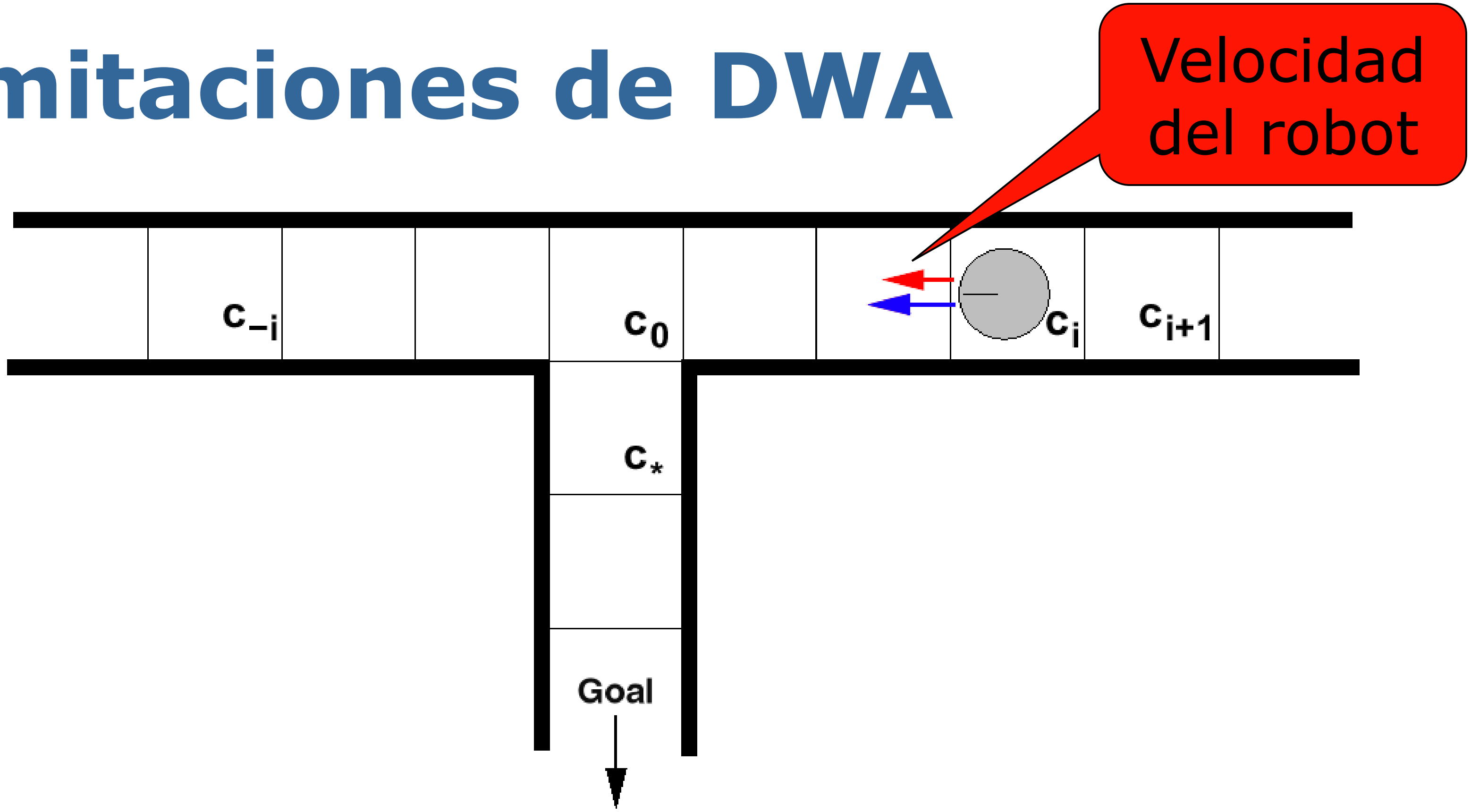
- Reacción rápida.
- Bajos requerimientos computacionales.
- Produce trayectorias libres de colisiones.
- Usado exitosamente en muchos escenarios reales.
- Las trayectorias resultantes son en general sub-óptimas.
- Puede no llegarse al objetivo debido a mínimos locales.

Limitaciones de DWA



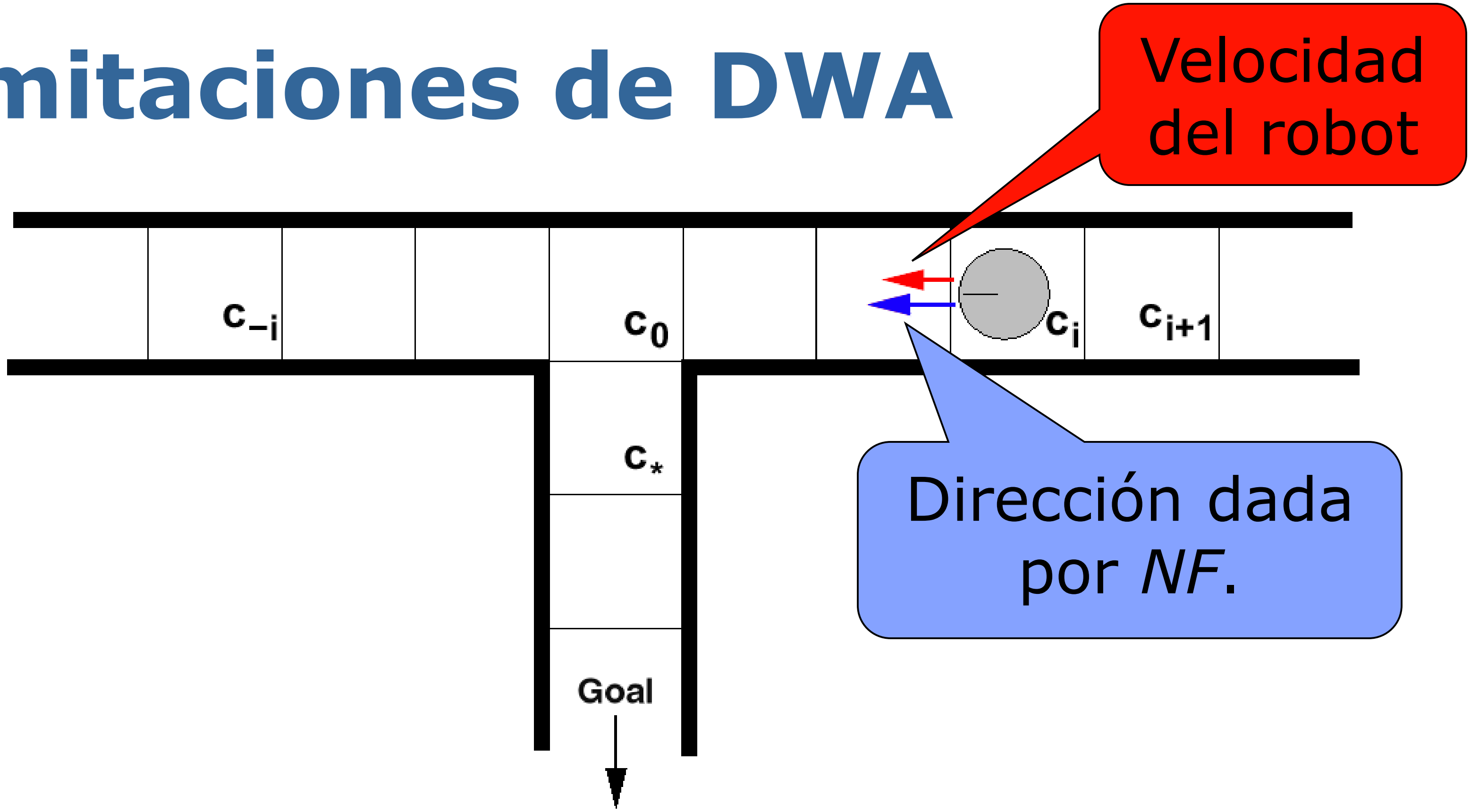
$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Limitaciones de DWA



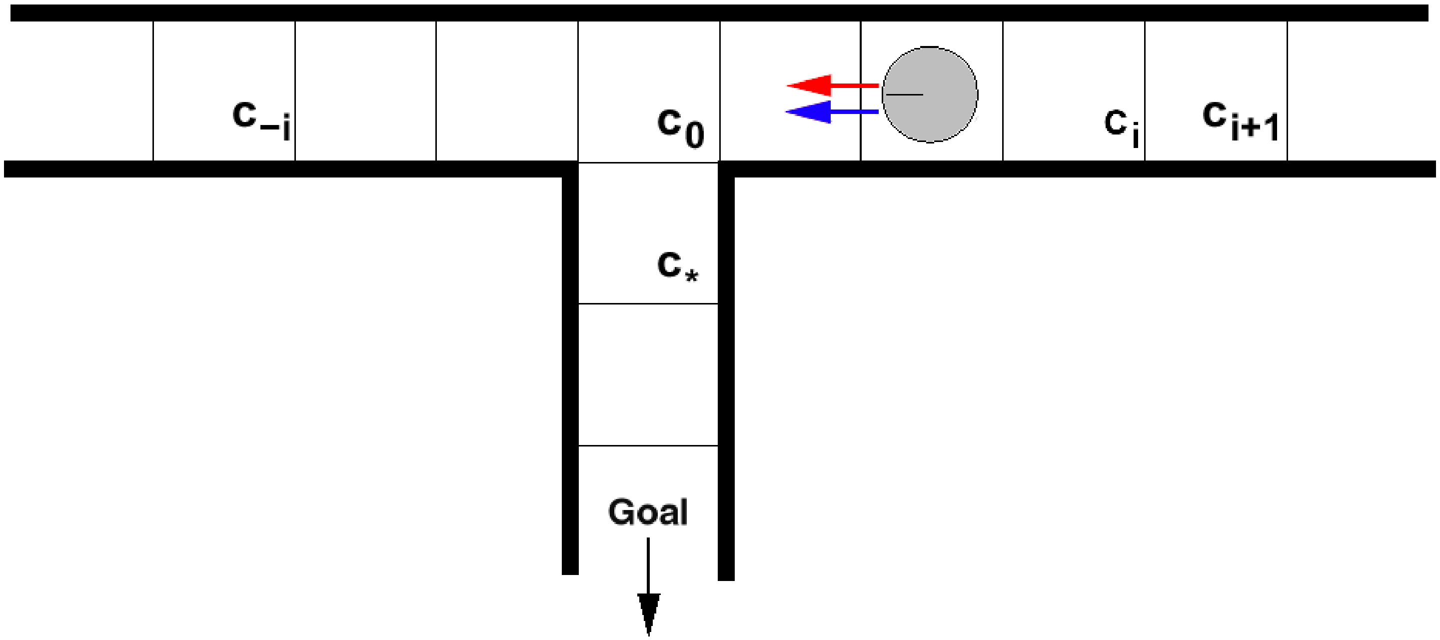
$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Limitaciones de DWA



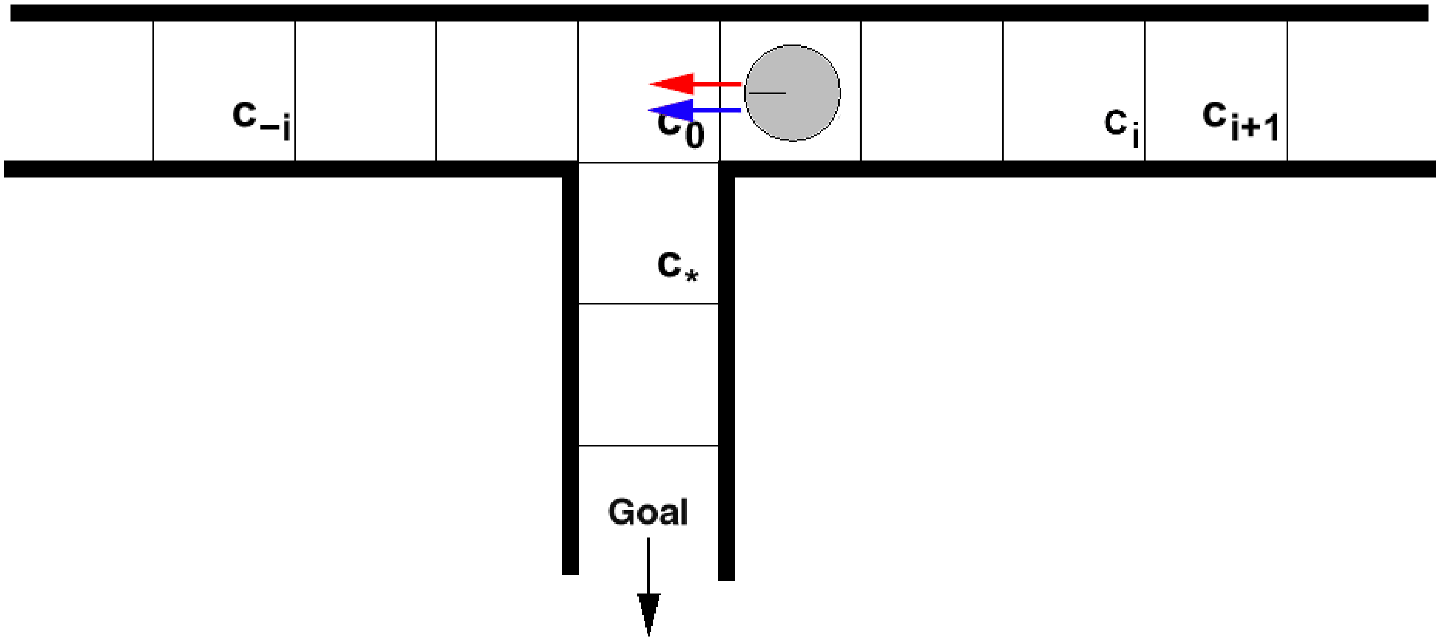
$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Limitaciones de DWA



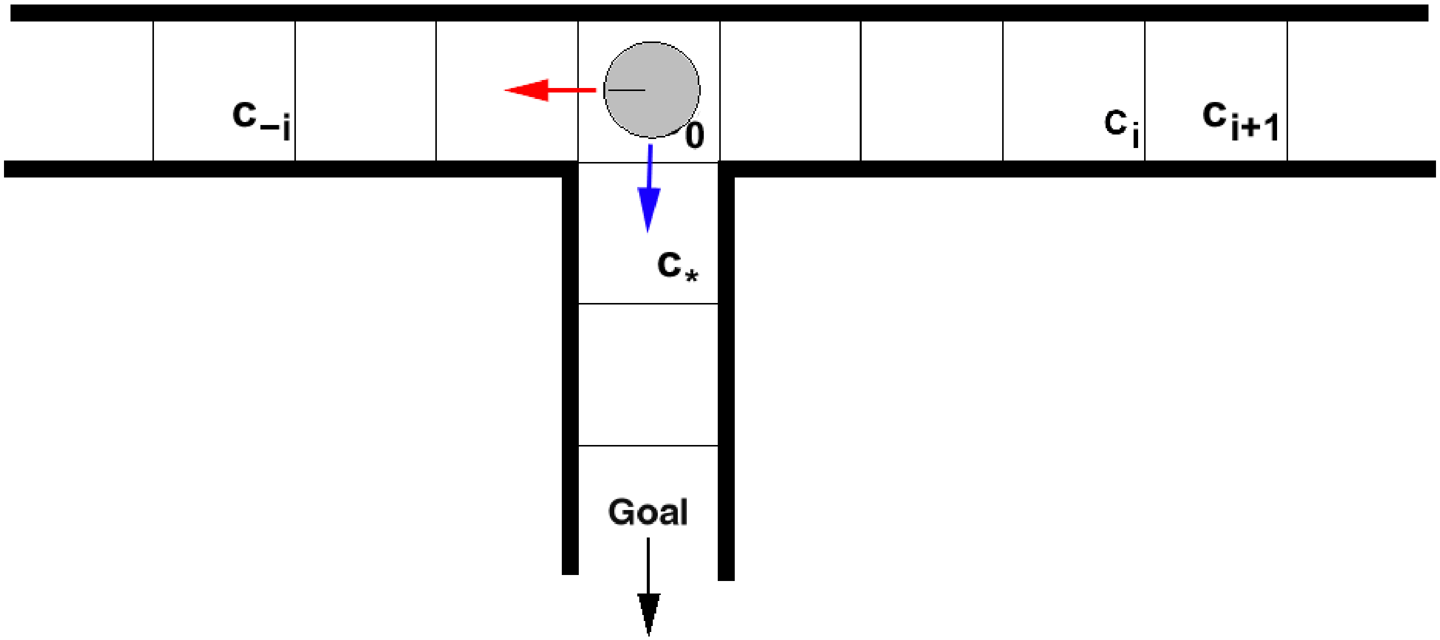
$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Limitaciones de DWA



$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

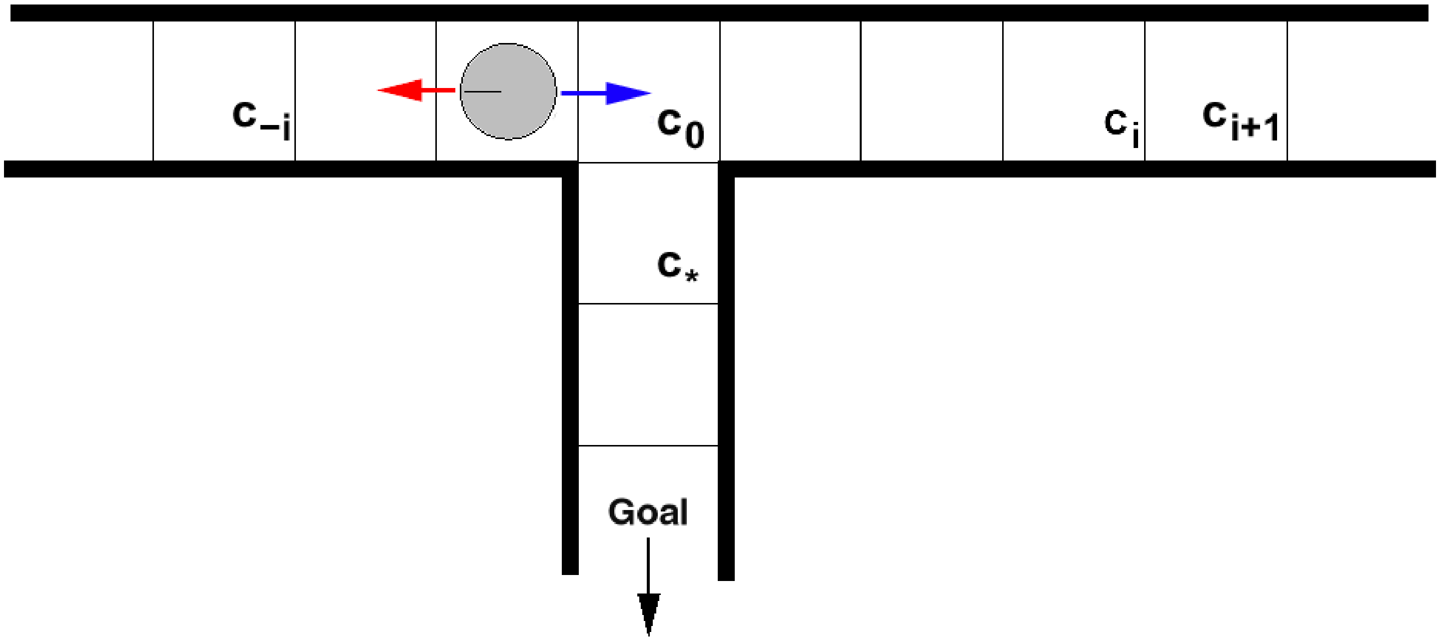
Limitaciones de DWA



$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

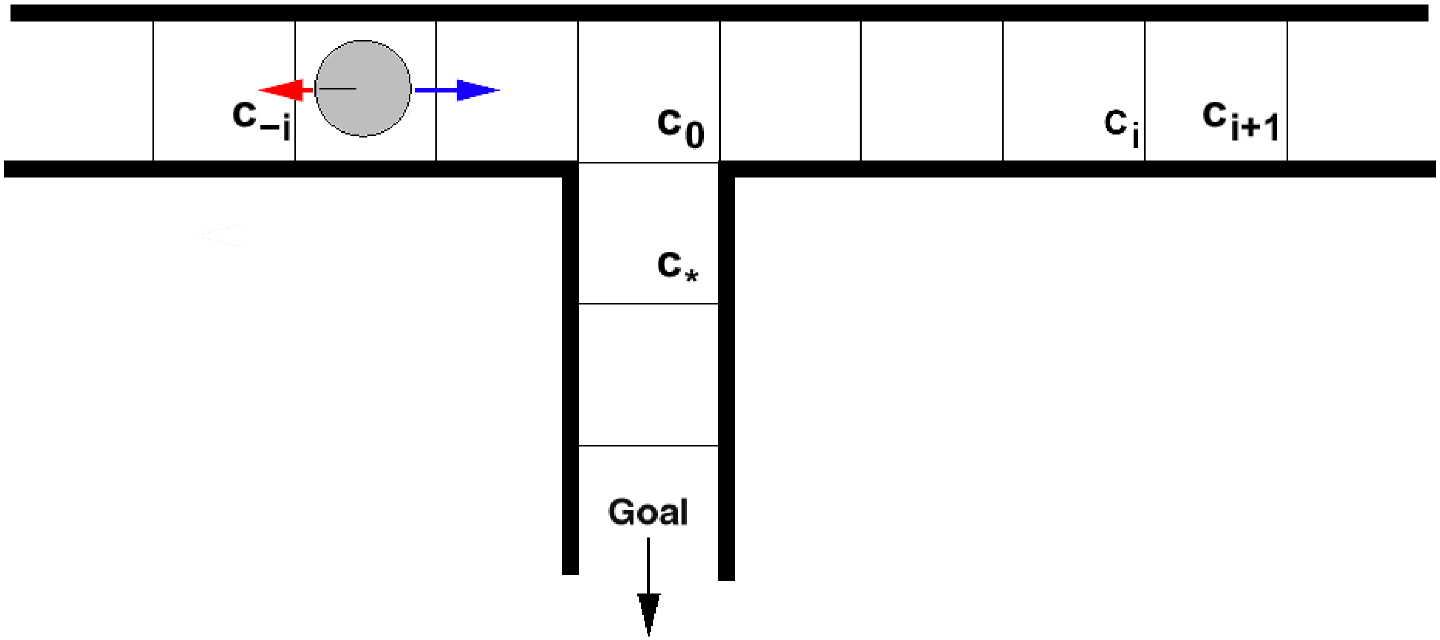
El robot tiene una velocidad demasiado alta en c_0 para poder ir hacia abajo.

Limitaciones de DWA



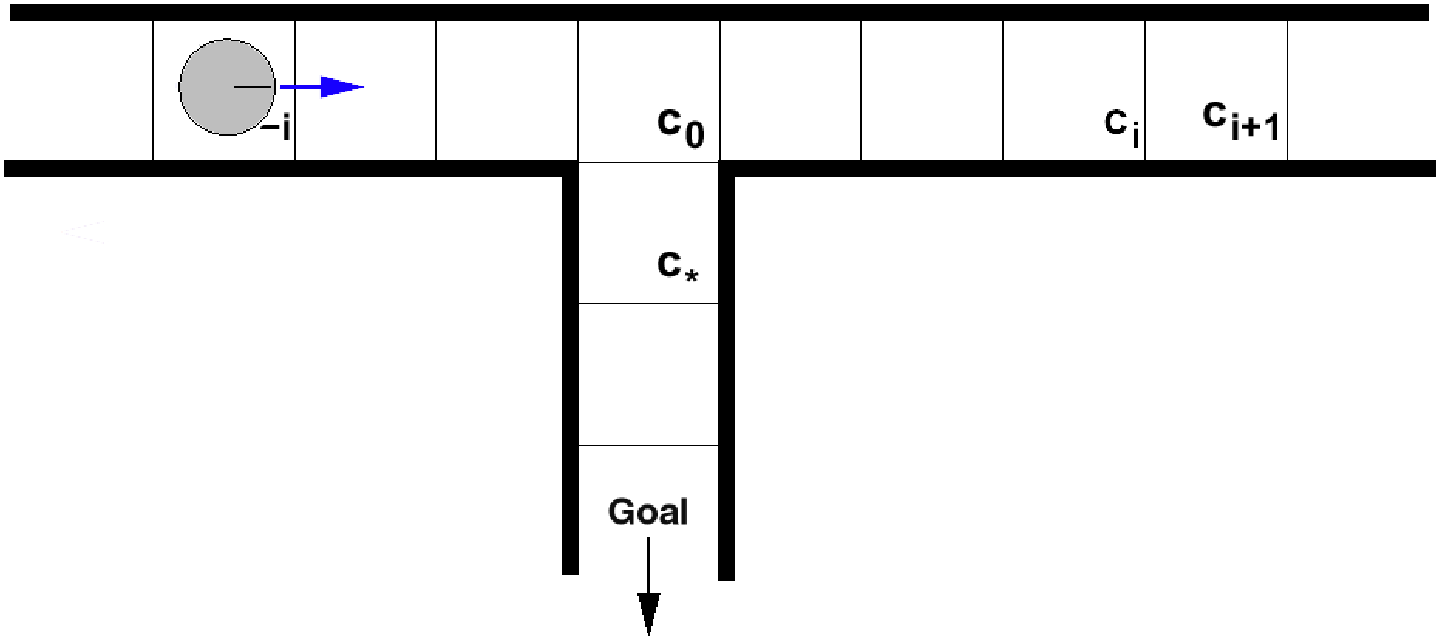
$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Limitaciones de DWA



$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Limitaciones de DWA



Situación similar a la inicial!

➔ DWA tiene problemas para llegar al objetivo.

Limitaciones de DWA

- Problema típico de una situación real:

- El robot no frena a tiempo para entrar por la puerta de manera suave.

Formulación del problema de planeamiento

- El problema de **planeamiento** se puede definir así: Dados
  - Una pose **inicial** del robot
  - Una pose **objetivo** deseada
  - Una descripción geométrica del **robot**
  - Una representación geométrica del **entorno**

- Encontrar un camino que lleva gradualmente al robot desde la pose **inicial** hasta la pose **objetivo** deseada **sin tocar ningún obstáculo**

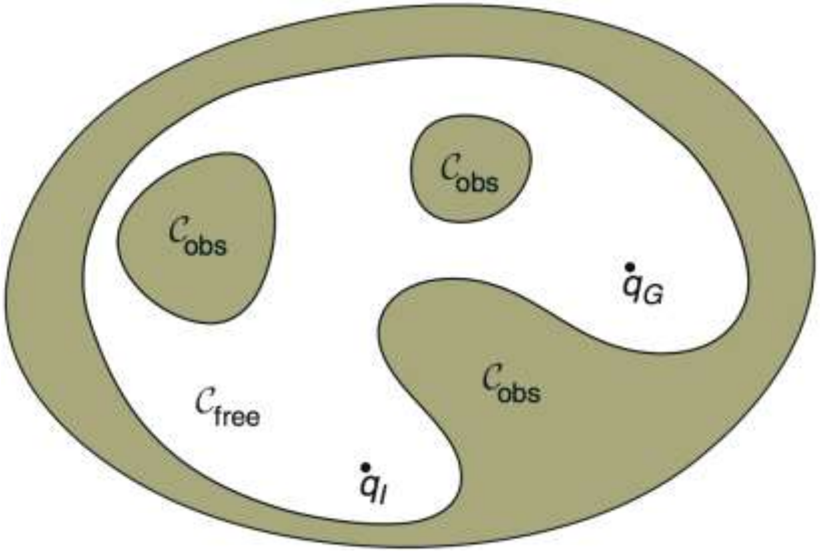
Espacio de configuración

- Aunque el problema de planeamiento se define en el mundo regular, se desarrolla en otro espacio: el **espacio de configuración**
- La configuración del robot q es una especificación de las posiciones de todos los puntos del robot relativos a un sistema de coordenadas fijo
- En general, una configuración se expresa como un **vector de posiciones** y **orientaciones**

Espacio de configuración

- **Espacio libre** y **región de obstáculo**
- Siendo $\mathcal{W} = \mathbb{R}^m$ el espacio de trabajo, $\mathcal{O} \in \mathcal{W}$ el conjunto de obstáculos, $\mathcal{A}(q)$ el robot en configuración $q \in \mathcal{C}$

$$
\begin{aligned}
\mathcal{C}_{free} &= \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} = \emptyset\} \\
\mathcal{C}_{obs} &= \mathcal{C} / \mathcal{C}_{free}
\end{aligned}
$$



- Además, definimos

  q_I : configuración inicial

  q_G : configuración final

Espacio de configuración

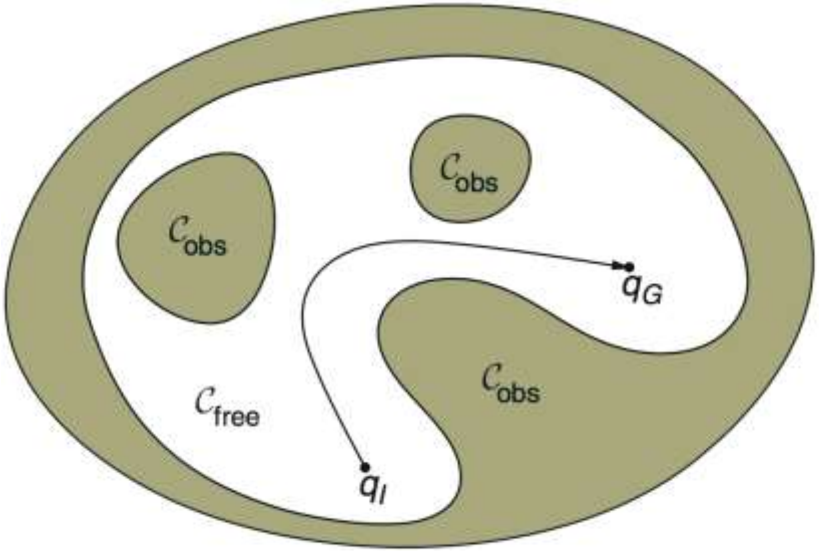
Entonces, planeamiento implica

- Encontrar un camino continuo

  $$\tau : [0,1] \rightarrow \mathcal{C}_{free}$$

  con $\tau(0) = q_I \,,\, \tau(1) = q_G$

- Dado esto, se hace planeamiento con el robot como un punto en el **espacio C**



Discretización del espacio C

- El terreno continuo debe **discretizarse** para hacer planeamiento
- Hay **dos métodos** generales para espacios C discretizados:
  - **Planeamiento combinatorio**
    Caracteriza C_{free} explícitamente capturando la conectividad de C_{free} en un grafo y usa técnicas de búsqueda
  - **Planeamiento basado en muestras**
    Usa detección de colisión para probar y buscar incrementalmente en el espacio C una solución

Búsqueda

El problema de **búsqueda:** encontrar una secuencia de acciones (un camino) que lleva al estado deseado (objetivo)

- **Búsqueda No-Informada:** no hay información más allá de la definición del problema ("búsqueda ciega")
- Las variantes están dadas por cómo se expanden los nodos
- Ejemplos de algoritmos: en anchura, en profundidad, en costo uniforme, bidireccional, etc.

Búsqueda

El problema de **búsqueda:** encontrar una secuencia de acciones (un camino) que lleva al estado deseado (objetivo)

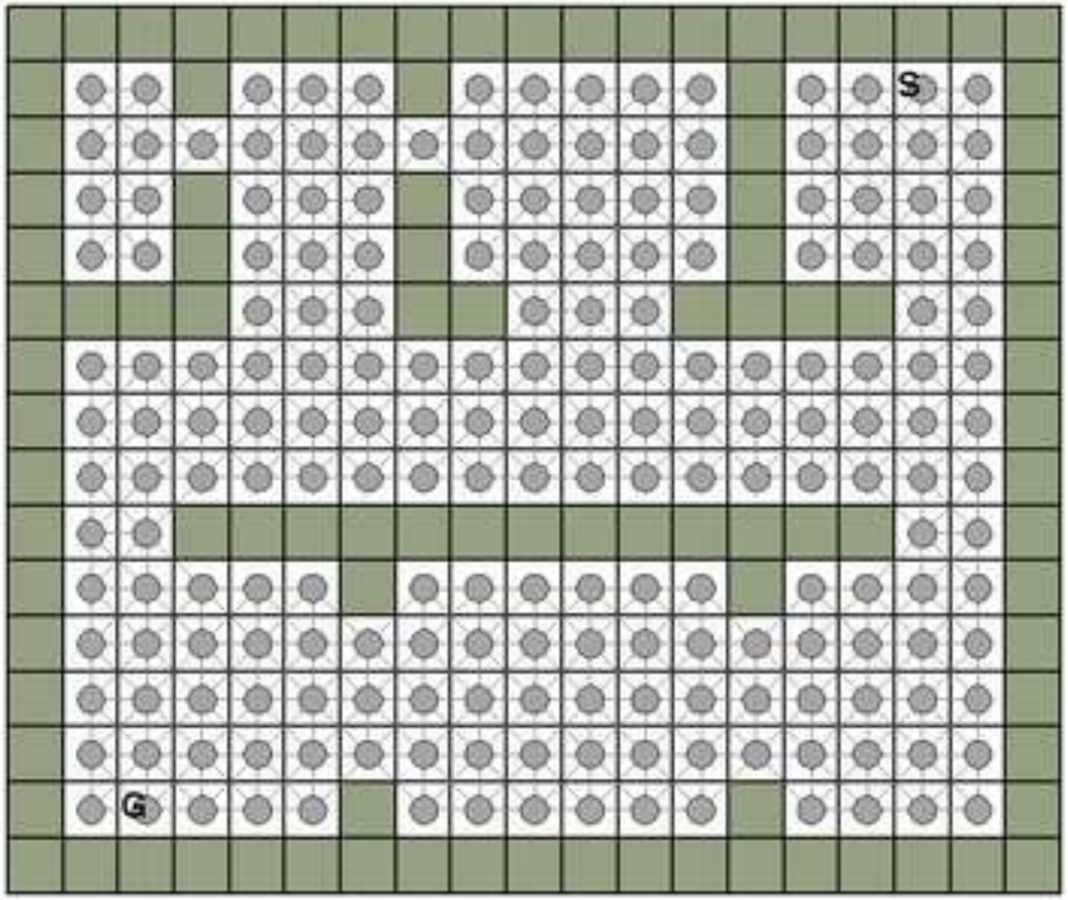
- **Búsqueda Informada:** información adicional a través de **heurísticas**
- Capacidad de decir si un nodo "es mejor candidato" que otro nodo
- Ejemplos de algoritmos: búsqueda voraz primero el mejor, **A***, variantes de A*, D*, etc.

Búsqueda

El desempeño de un algoritmo de búsqueda se mide en 4 dimensiones:

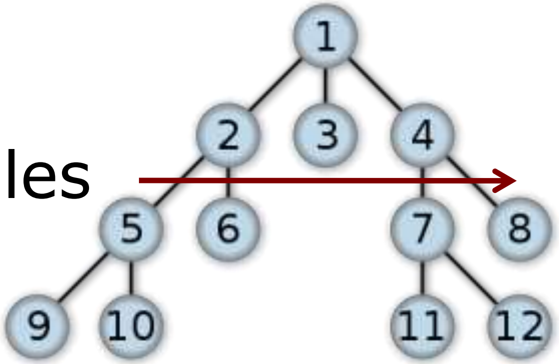
- **Completitud:** ¿Encuentra una solución si existe?
- **Optimalidad:** ¿Es la solución la mejor de todas las posibles soluciones en términos de costo?
- **Complejidad temporal:** ¿Cuanto se tarda en encontrar la solución?
- **Complejidad de memoria:** ¿Cuanta memoria se necesita para hacer la búsqueda?

Discretización del espacio de configuración

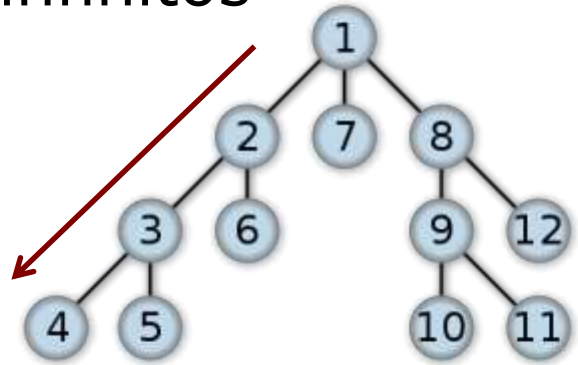


Búsqueda No Informada

- **En anchura**
  - Completo
  - Óptimo si los costos son iguales
  - Tiempo y mem.: $O(b^d)$



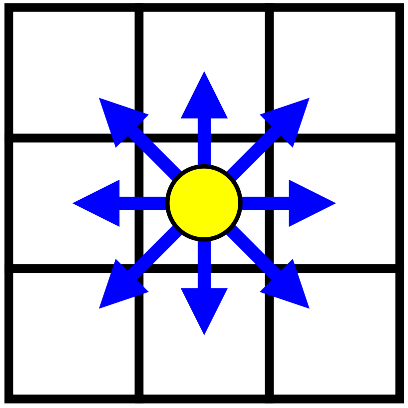
- **En profundidad**
  - No es completo en espacios infinitos
  - No es óptimo
  - Tiempo: $O(b^m)$
  - memoria: $O(bm)$ (olvido de subárboles explorados)



(b: factor de ramificación, d: profundidad del objetivo, m: profundidad *máxima)*

Búsqueda informada: A*

- Encuentra el camino más corto (de menor costo)
- Requiere una estructura de grafo
- Número limitado de arcos
- en robótica: planeamiento en un mapa de grilla de ocupación 2D



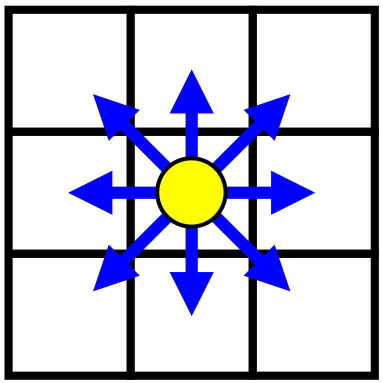
A^*: Minimiza el costo estimado del camino

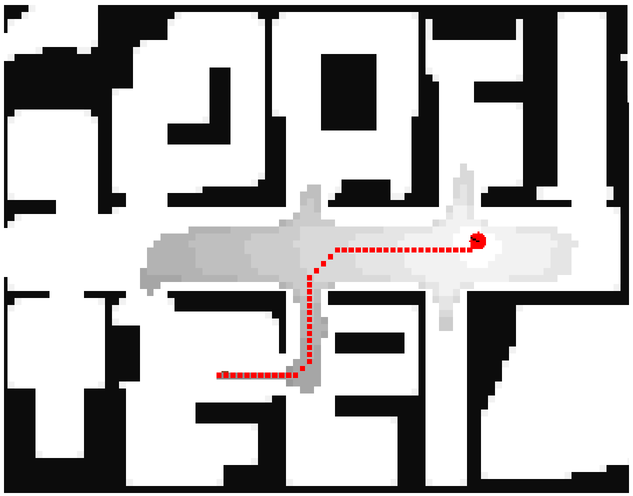
- $g(n)$ = costo del estado inicial hasta n.
- $h(n)$ = costo estimado desde n hasta el objetivo.
- $f(n) = g(n) + h(n)$, costo estimado de la solución de menor costo a través de n.
- Sea $h^*(n)$ el costo del camino óptimo desde n hasta el objetivo.
- h es admisible si se cumple para todo n que:

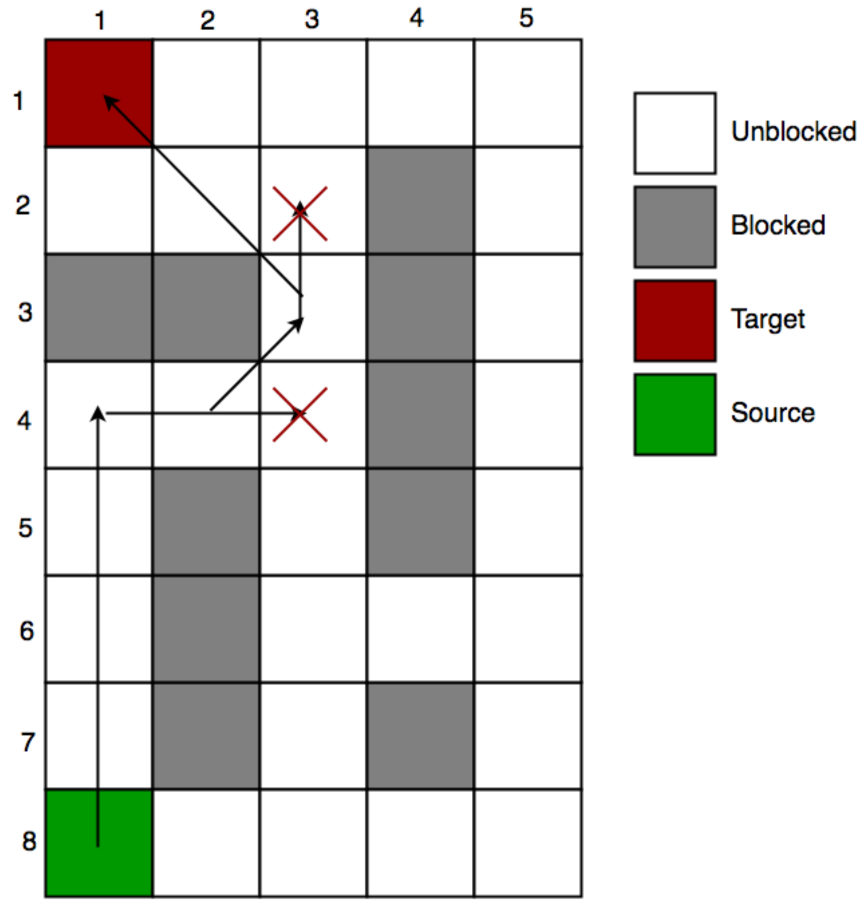
$$h(n) \leq h^*(n)$$

- Se necesita que para A^*, h sea admisible (la distancia recta es admisible en el espacio euclídeo).

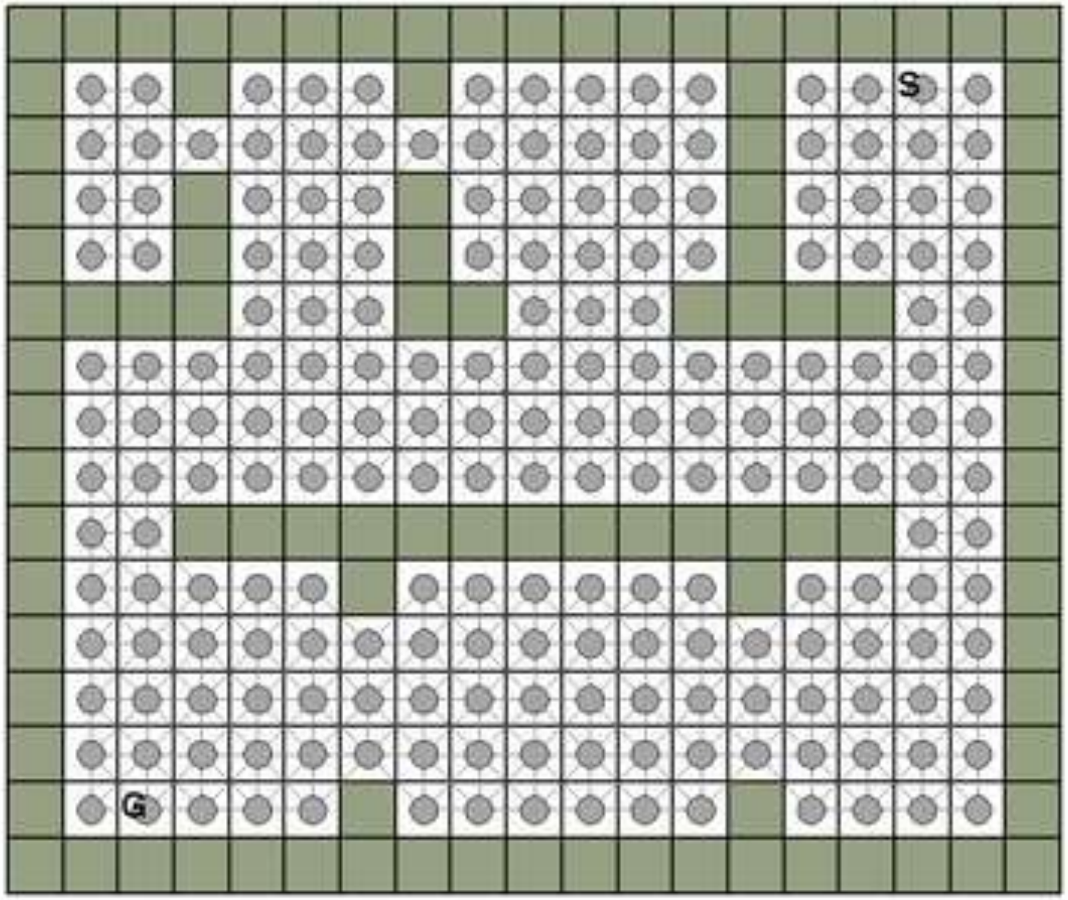
Ejemplo: planeamiento en un mapa de grilla







Ejemplo



Iteración determinística de valores

- Para calcular el camino más corto desde todos los estados a un estado objetivo, usar iteración (determinística) de valores.
- Muy similar al algoritmo de Dijkstra.
- Esta distribución de costo es la heurística óptima para A^*.

Suposiciones típicas en robótica para planeamiento A^*

1. El robot se supone localizado.
2. El robot calcula su camino basado en una grilla de ocupación.
3. Los comandos de movimiento correctos son ejecutados.

¿Son 1. y 3. siempre ciertos?

Problemas

- ¿Qué pasa si el robot está (levemente) deslocalizado?
- Al moverse por el camino más corto, el robot puede pasar muy cerca de obstáculos.
- La trayectoria está alineada con la estructura de grilla.

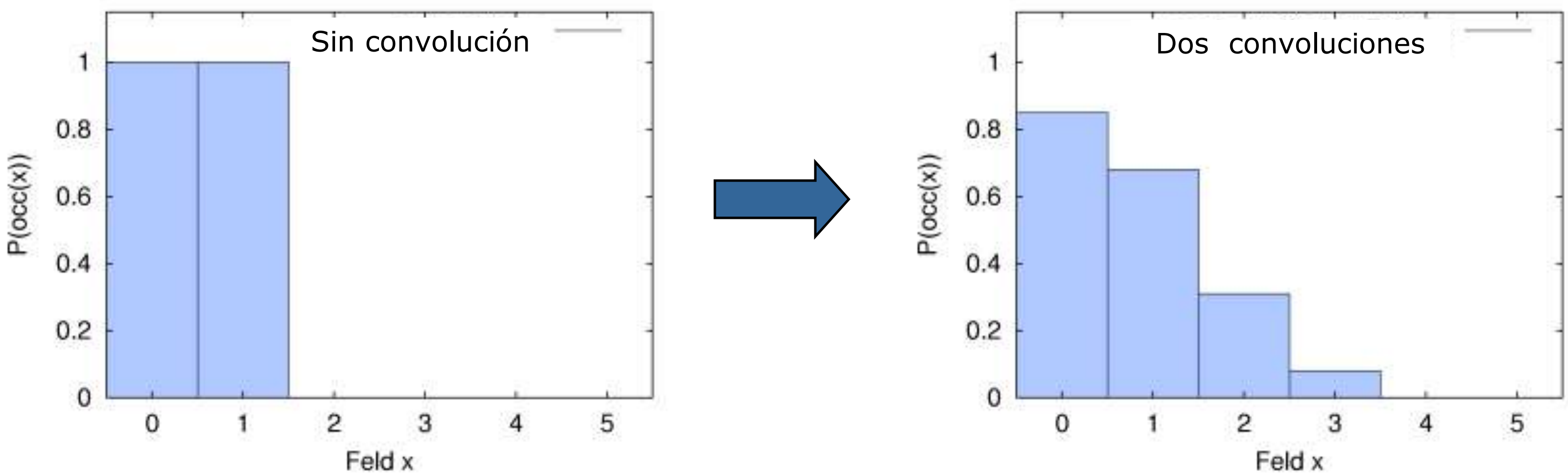
Convolución del mapa de grilla

- La convolución suaviza el mapa.
- Los obstáculos se asumen más grandes que lo que en realidad son.

- Hacer una búsqueda A^* en el mapa convolucionado (usando ocupación como costo de recorrido).
- El robot aumenta su distancia a los obstáculos y hace un camino corto!

Ejemplo: Convolución de mapa

- Entorno uni-dimensional, celdas $c_0, \ldots, c_5$



Celdas antes y después de dos convoluciones.

Convolución

- Dado un mapa de ocupación, la convolución se define como:

$$P(occ_{x_i,y}) = \frac{1}{4} \cdot P(occ_{x_{i-1},y}) + \frac{1}{2} \cdot P(occ_{x_i,y}) + \frac{1}{4} \cdot P(occ_{x_{i+1},y})$$

$$P(occ_{x_0,y}) = \frac{2}{3} \cdot P(occ_{x_0,y}) + \frac{1}{3} \cdot P(occ_{x_1,y})$$

$$P(occ_{x_{n-1},y}) = \frac{1}{3} \cdot P(occ_{x_{n-2},y}) + \frac{2}{3} \cdot P(occ_{x_{n-1},y})$$

- Esto se hace para cada fila y cada columna del mapa.
- “Gaussian blur”

A* en mapas convolucionados

- Los costos son el producto de la longitud del camino y la probabilidad de ocupación de las celdas.
- Las celdas con mayor probabilidad (ej: debido a la convolución) son evitadas por el robot.
- Se mantiene la distancia a los obstáculos.
- Este método es rápido y confiable.

Planeamiento 5D

- Alternativa a la arquitectura de dos capas
- Planea en el espacio de configuración completo $\langle x, y, \theta, v, \omega \rangle$ usando A^*.
  - ➔ Considera las restricciones cinemáticas del robot.

El espacio de búsqueda (1)

- Cúal es el estado en este espacio?
  $<x,y,\theta,v,\omega>$ = posición y velocidad actual del robot

- ¿Cómo es la transición de estados?
  $<x1,y1,\theta 1,v1,\omega 1> \rightarrow <x2,y2,\theta 2,v2,\omega 2>$

  - con el comando de movimiento $(v2,\omega 2)$ y $|v1-v2| < a_{v_{max}}t$, $|\omega 1-\omega 2| < a_{\omega_{max}}t$.

  - La nueva pose del robot $<x2,y2,\theta 2>$ es el resultado de las ecuaciones de movimiento.

El espacio de búsqueda (2)

Idea: Buscar en el espacio discretizado $<x,y,\theta,v,\omega>$.

Problema: el espacio de búsqueda es demasiado grande para ser explorado dentro de las restricciones de tiempo (5+ Hz para planeamiento online).

Solución: restringir el espacio de búsqueda.

Pasos principales del algoritmo

1. Actualizar el mapa de grilla (estático) según la información de los sensores.

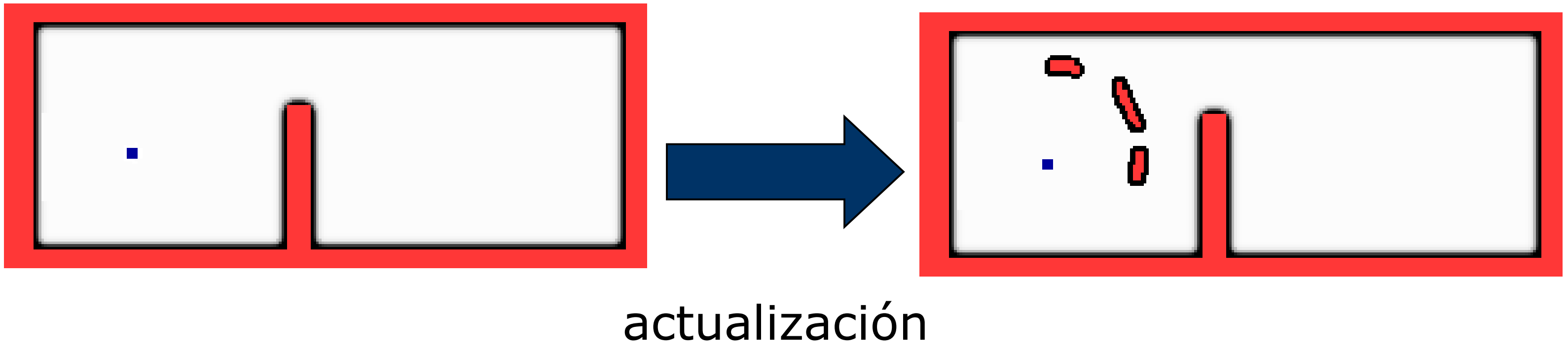
2. Usar A* para encontrar la trayectoria en el espacio <x,y> usando el mapa de grilla actualizado.

3. Determinar un espacio de configuración restringido en 5D basado en el paso 2.

4. Encontrar la trayectoria planeando en el espacio restringido $<x,y,\theta,v,\omega>$.

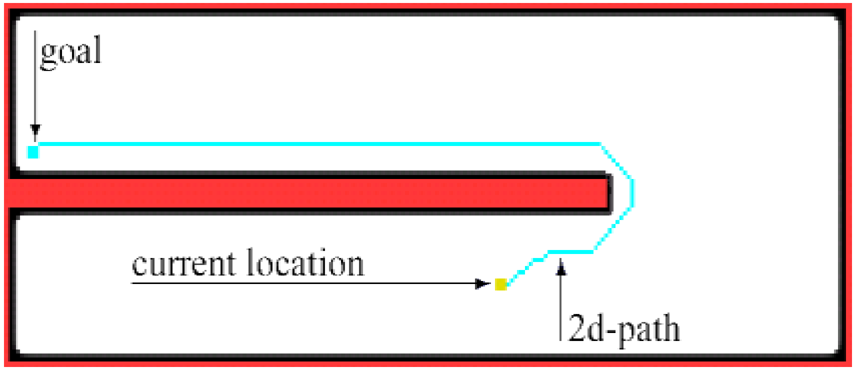
Actualizando el mapa de grilla

- Grilla de ocupación 2D.
- Convolución.
- Se agregan obstáculos detectados.
- Liberar celdas no ocupadas.



Encontrar trayectoria en el espacio 2D

- Usar A^* para buscar el camino óptimo en la grilla 2D.
- Usar heurísticas basadas en la iteración de valor determinístico dentro del mapa estático.



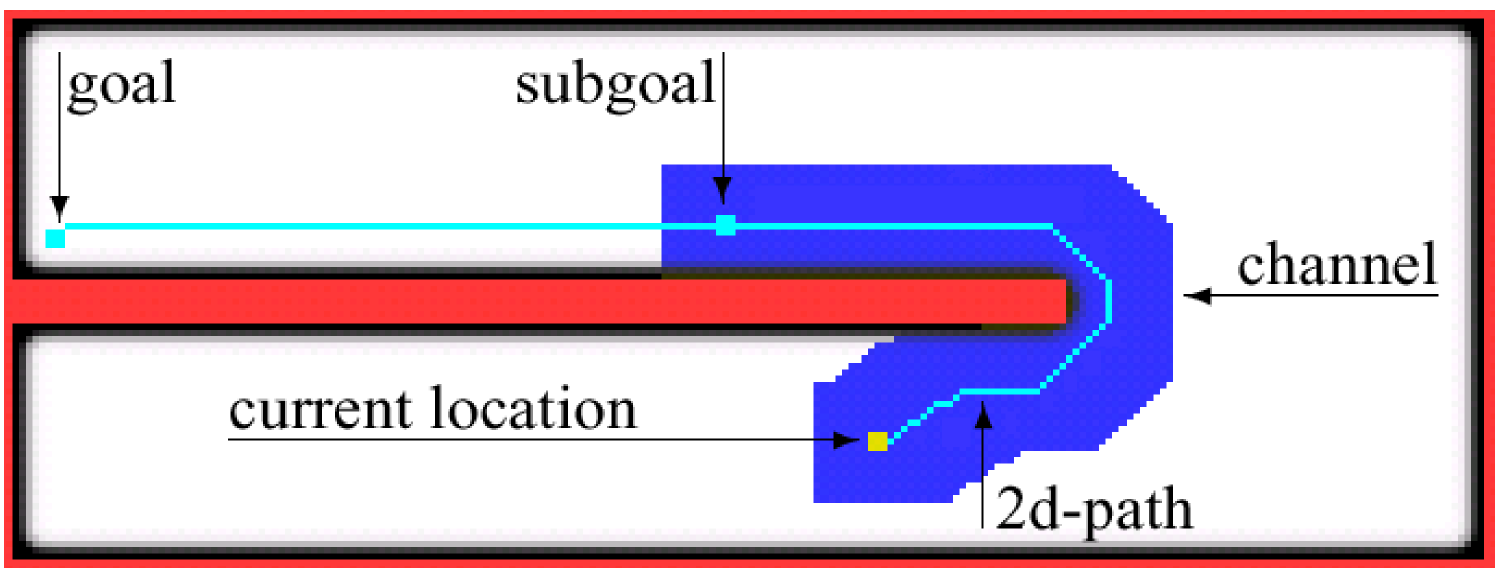
Restringir el espacio de búsqueda

Suposición: la proyección del camino en 5D sobre el espacio <x,y> está cerca del camino óptimo en 2D.

Entonces: construir un espacio de búsqueda restringido (canal) basado en el camino en 2D.

Restricción espacial

- Espacio de búsqueda resultante :
  <x, y, θ, v, ω> con (x,y) ϵ canal.
- Elegir un objetivo intermedio (subgoal) en el camino 2D dentro del canal.

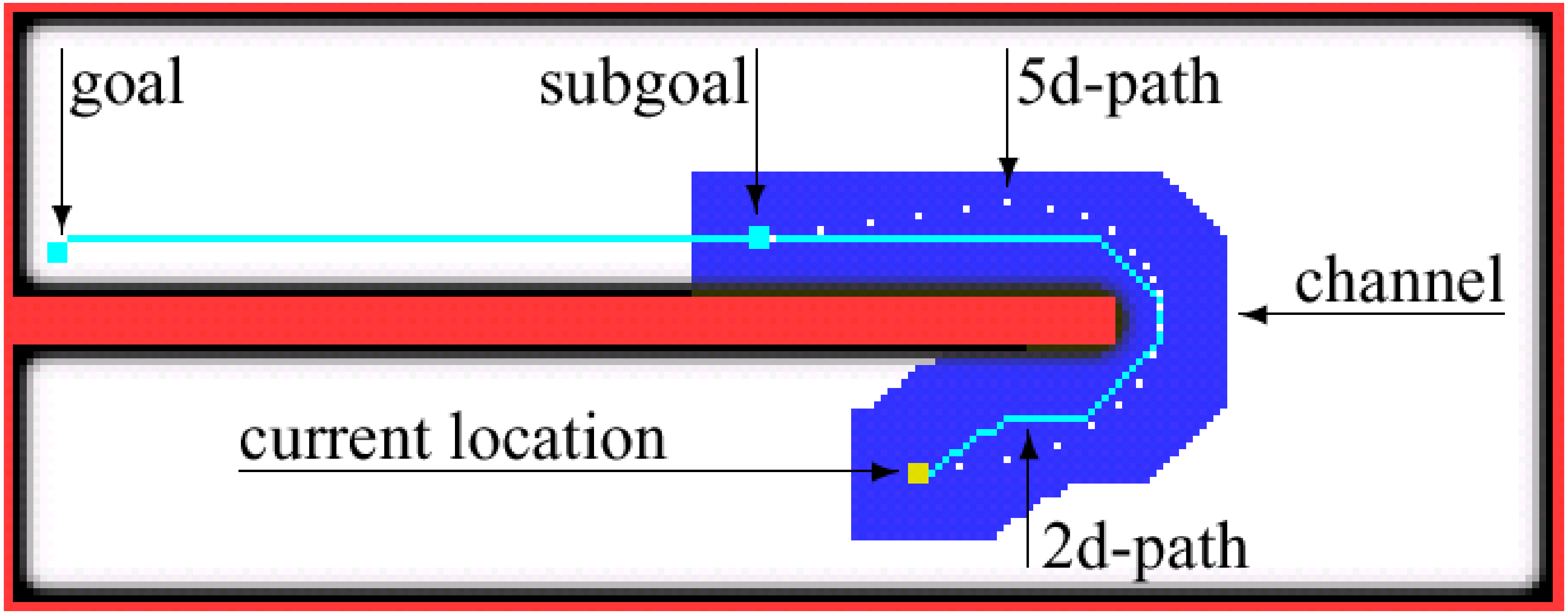


Encontrar trayectoria en el espacio 5D

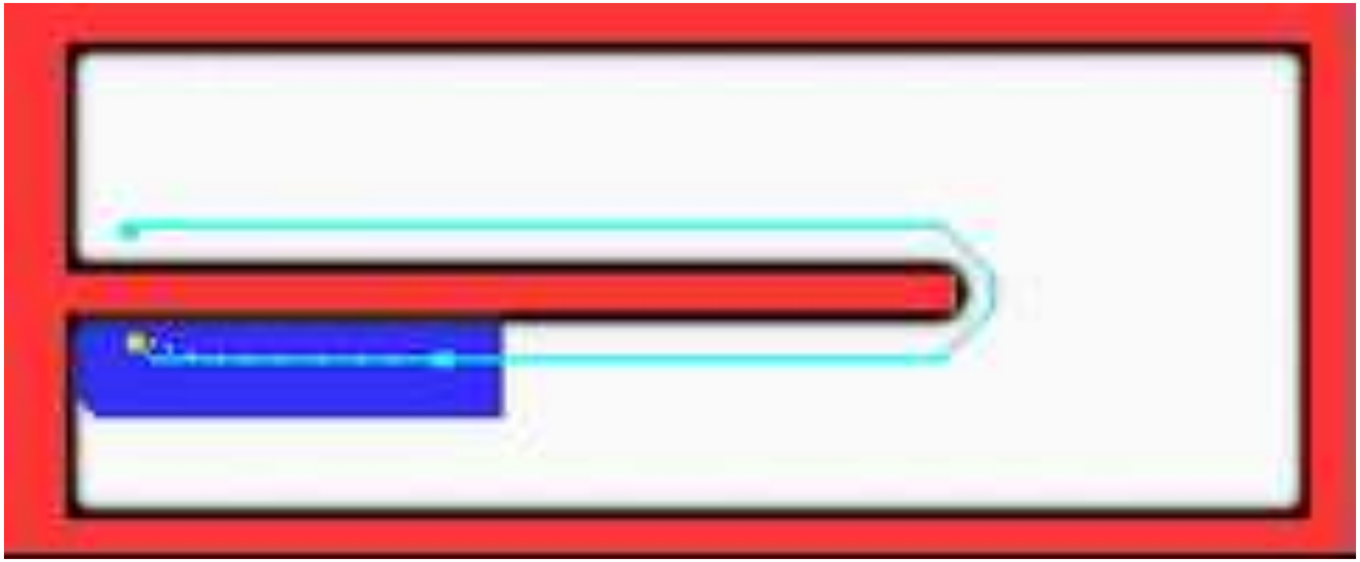
- Usar **A^* en el espacio restringido 5D** para encontrar la secuencia de comandos para llegar al objetivo intermedio.

- Para estimar los costos de las celdas: hacer una iteración determinística de valores 2D dentro del canal.

Ejemplos



Ejemplos



Límites de tiempo (timeouts)

- Mover un robot en tiempo real requiere nuevos comandos a tasa alta. Ej: cada 0.2 segundos.

- ¿Que pasa si no encuentro una solución?

→ Abortar la búsqueda después de 0.2 segundos.

¿Cómo encontrar un comando admisible para el robot?

Comandos de acción alternativos

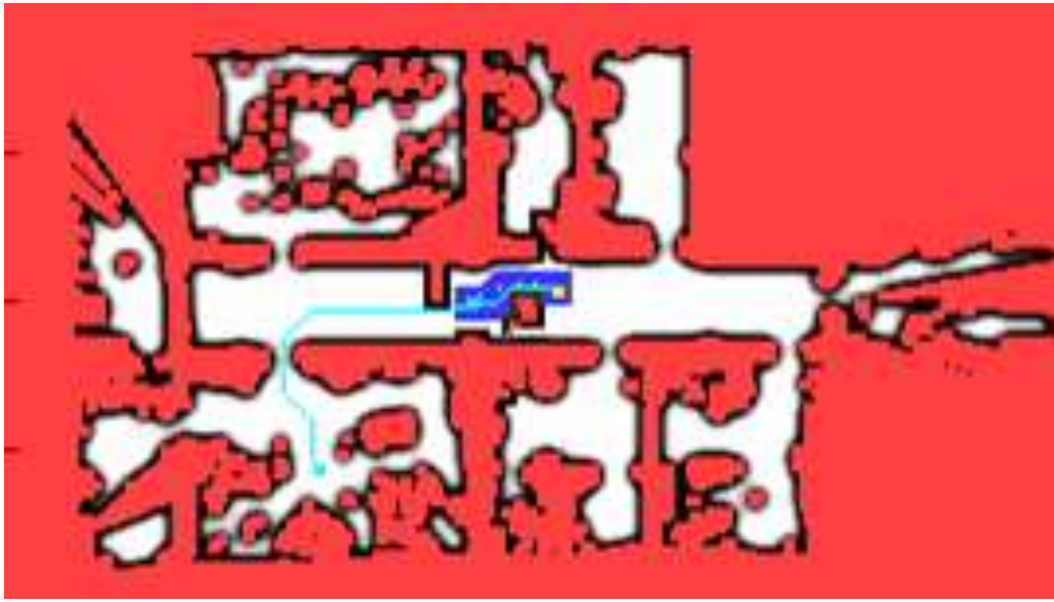
- ¿La trayectoria anterior es todavía admisible?

  → **OK, la uso.**

- Si no, seguir el camino 2D o usar DWA para encontrar un nuevo comando.

Ejemplo

Robot XR-4000

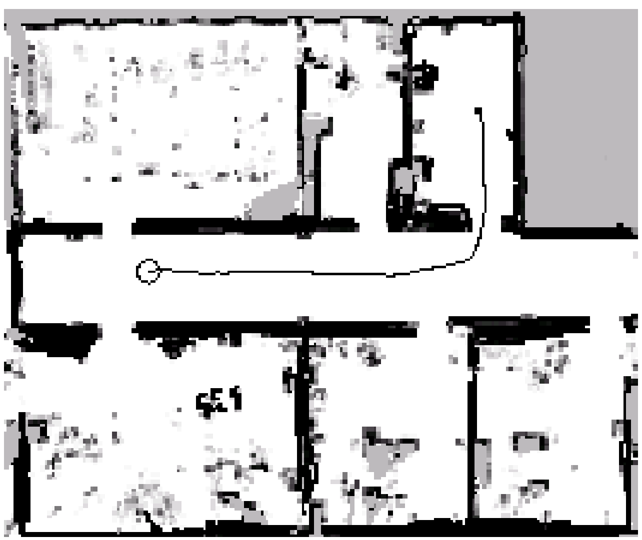


Planeamiento

Comparación con DWA (1)

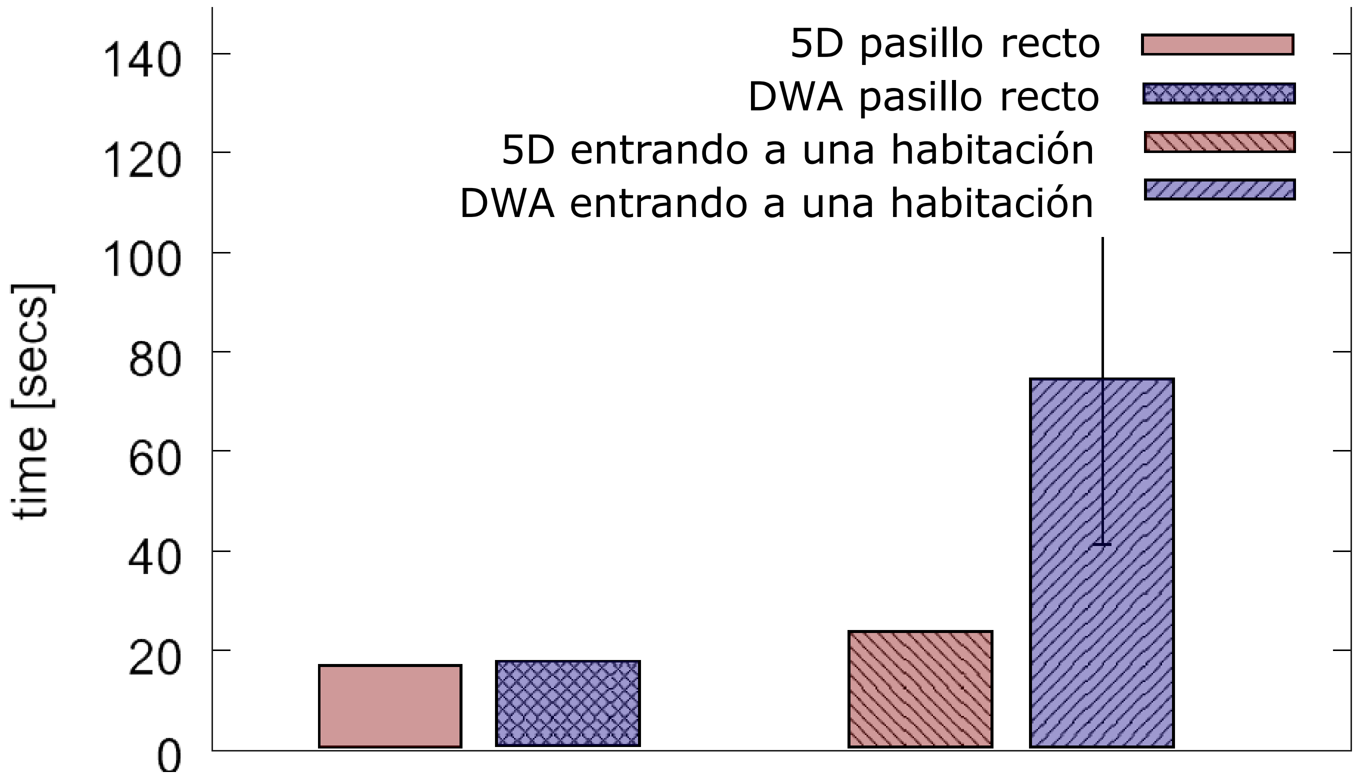
- DWA muchas veces tiene problemas para entrar por lugares estrechos.

Planeamiento DWA



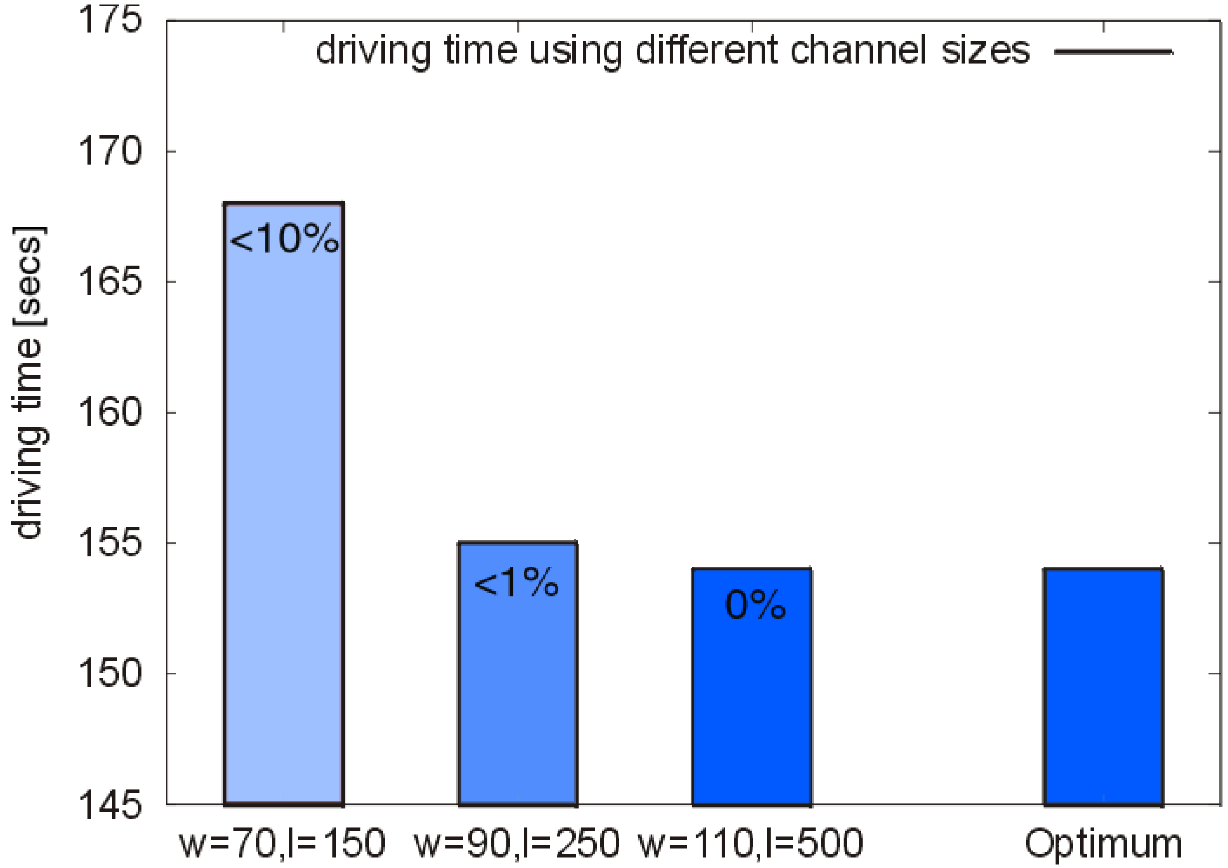
Método 5D.

Comparación con DWA (2)



El método 5D resulta en movimientos mucho más rápidos al pasar por lugares estrechos!

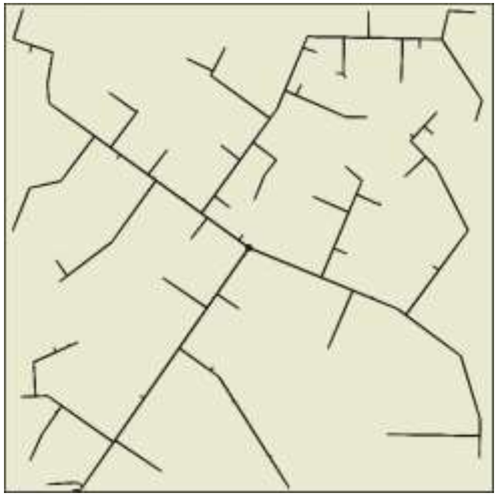
Comparación con el óptimo



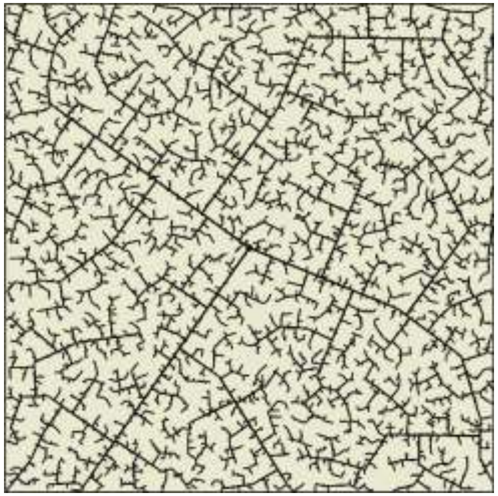
Para un canal mayor, las acciones resultantes están cerca de la solución óptima.

Rapidly Exploring Random Trees

- **Idea:** explorar el espacio C agresivamente **expandiendo incrementalmente** desde una configuración inicial q_0
- El territorio explorado queda definido por un **árbol** que nace en $\boldsymbol{q_0}$



45 iteraciones



2,345 iteraciones

RRTs

Algoritmo: Dado C y q_0

Algorithm 1: RRT

G.init(q_0)
repeat
    $q_{rand} \rightarrow$ RANDOM_CONFIG($\mathcal{C}$)
    $q_{near} \leftarrow$ NEAREST(G, q_{rand})
    G.add_edge(q_{near}, q_{rand})
until *condition*

Muestrea de una **región acotada** centrada en q_0

Ejemplo: traslación o rotación aleatoria

RRTs

- Algoritmo

Algorithm 1: RRT

G.init(q_0)
repeat
  $q_{rand} \rightarrow$ RANDOM_CONFIG($\mathcal{C}$)
  $q_{near} \leftarrow$ NEAREST(G, q_{rand})
  G.add_edge(q_{near}, q_{rand})
until *condition*

Encontrar el nodo más cercano en G usando una **función de distancia**

$$\rho \ : \ \mathcal{C} \times \mathcal{C} \rightarrow [0, \infty)$$

Formalmente, una ***métrica*** definida en C

RRTs

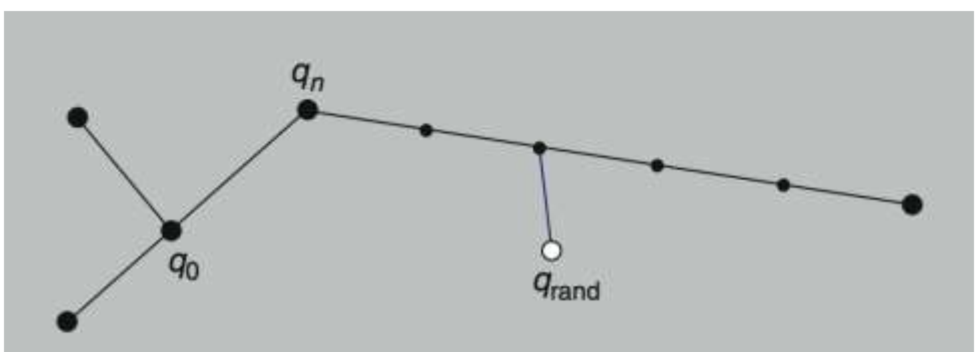
- Algoritmo

Algorithm 1: RRT

$G.\text{init}(q_0)$
repeat
  $q_{rand} \rightarrow \text{RANDOM\_CONFIG}(\mathcal{C})$
  $q_{near} \leftarrow \text{NEAREST}(G, q_{rand})$
  $G.\text{add\_edge}(q_{near}, q_{rand})$
until *condition*

Muchas estrategias para encontrar q_{near} dado el nodo más cercano en G:

- Tomar el nodo más cercano
- Chequear puntos intermedios a intervalos regulares y dividir arcos en q_{near}



RRTs

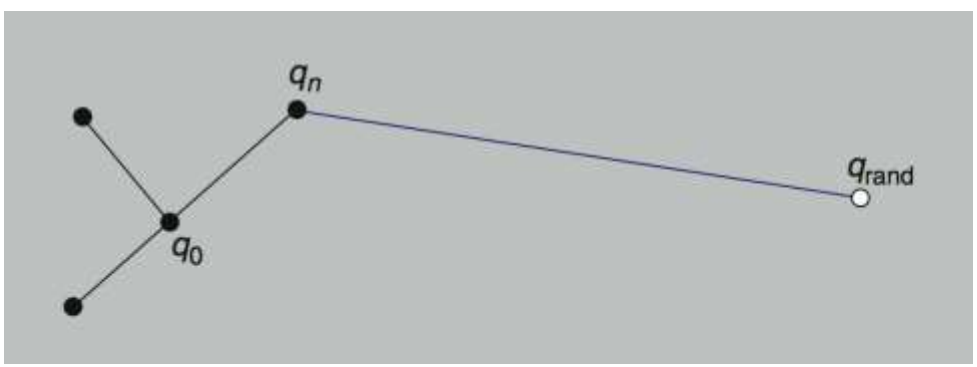
- Algoritmo

Algorithm 1: RRT

G.init(q_0)
repeat
  $q_{rand} \rightarrow$ RANDOM_CONFIG($\mathcal{C}$)
  $q_{near} \leftarrow$ NEAREST(G, q_{rand})
  G.add_edge(q_{near}, q_{rand})
until *condition*

Conecta el punto más cercano con el punto aleatorio usando un **planeamiento local** de q_{near} a q_{rand}

- No hay colisión: agregar arco

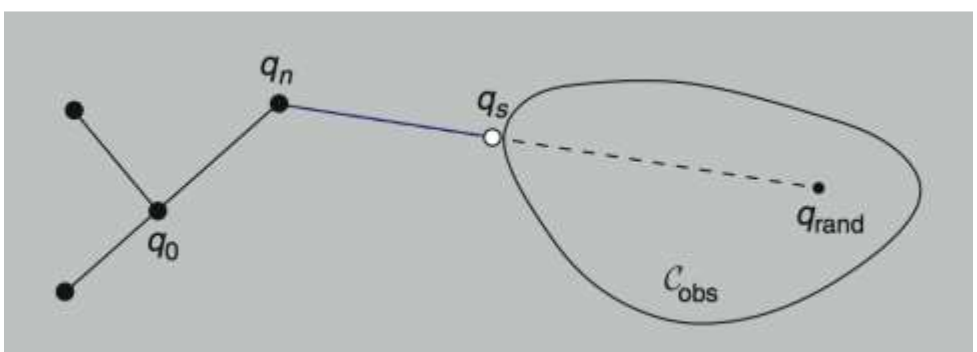


RRTs

- Algoritmo

Algorithm 1: RRT

$G.\text{init}(q_0)$
repeat
    $q_{rand} \rightarrow \text{RANDOM\_CONFIG}(\mathcal{C})$
    $q_{near} \leftarrow \text{NEAREST}(G, q_{rand})$
    $G.\text{add\_edge}(q_{near}, q_{rand})$
until *condition*



Conecta el punto más cercano con un punto aleatorio usando un **planeamiento local** de q_{near} a q_{rand}

- No hay colisión: agregar arco
- Hay colisión: nuevo nodo es q_s, lo más cercano posible a C_{obs}

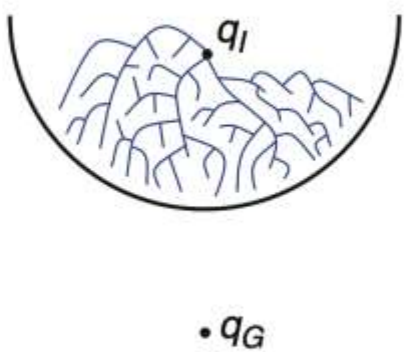
RRTs

- Cómo hacer planeamiento de trayectorias con RRTs?
  1. Comenzar RRT en q_I
  2. Cada, supongamos, 100 iteraciones, forzar $q_{rand} = q_G$
  3. Si se llega a q_G, el problema está resulto

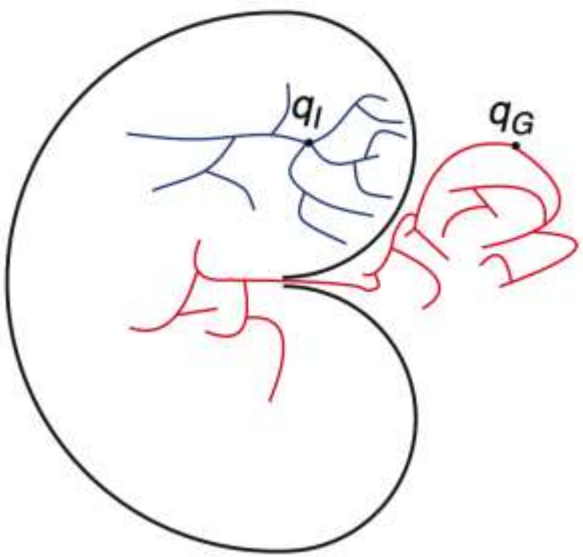
- ¿Por qué no elegir q_G en cada iteración?
  - Porque se van a gastar recursos terminando en C_{Obs} en vez de explorar el espacio

RRTs

- Algunos problemas requieren métodos más efectivos: **búsqueda bidireccional**
- generar **dos** RRTs: uno desde q_I, otro desde q_G
- Cada cierto # de pasos, tratar de extender cada árbol hasta el nodo más nuevo del otro



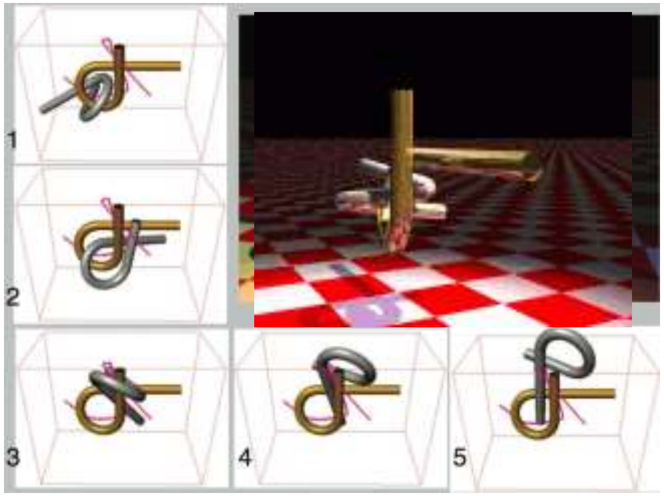
Llenar un pozo



Trampa de insectos

RRTs

- RRTs son muy populares, existen muchas extensiones: RRTs en tiempo real, anytime RRTs, para entornos dinámicos, etc.
- **Pro:**
  - Balance entre búsqueda voraz y exploración
  - Fácil de implementar
- **Contra:**
  - Sensible a la métrica
  - Velocidad de convergencia desconocida



Puzzle Alpha 1.0.
Resuelto con RRT bidireccional

Planeamiento de Mapa de Ruta (Road Map)

- Un **mapa de ruta** es un **grafo en** $\boldsymbol{C_{free}}$ en el cual cada nodo es una configuración en C_{free} y cada arco es un camino libre de colisiones entre nodos a través de C_{free}

- Varias **técnicas de planeamiento**
  - Grafos de visibilidad
  - Diagramas de Voronoi
  - Descomposición exacta de celdas
  - Descomposición aproximada de celdas
  - Road maps aleatorios

Planeamiento de Mapa de Ruta (Road Map)

- Un **mapa de ruta** es un **grafo en C_{free}** en el cual cada nodo es una configuración en C_{free} y cada arco es un camino libre de colisiones entre nodos a través de C_{free}

- Varias **técnicas de planeamiento**
  - Grafos de visibilidad
  - **Diagramas de Voronoi**
  - Descomposición exacta de celdas
  - Descomposición aproximada de celdas
  - **Road maps aleatorios**

Diagrama de Voronoi

- Definir regiones que contienen los puntos más cercanos a un obstáculo.

- **Las fronteras** son los lugares de **máxima distancia** a todos los obstáculos cercanos

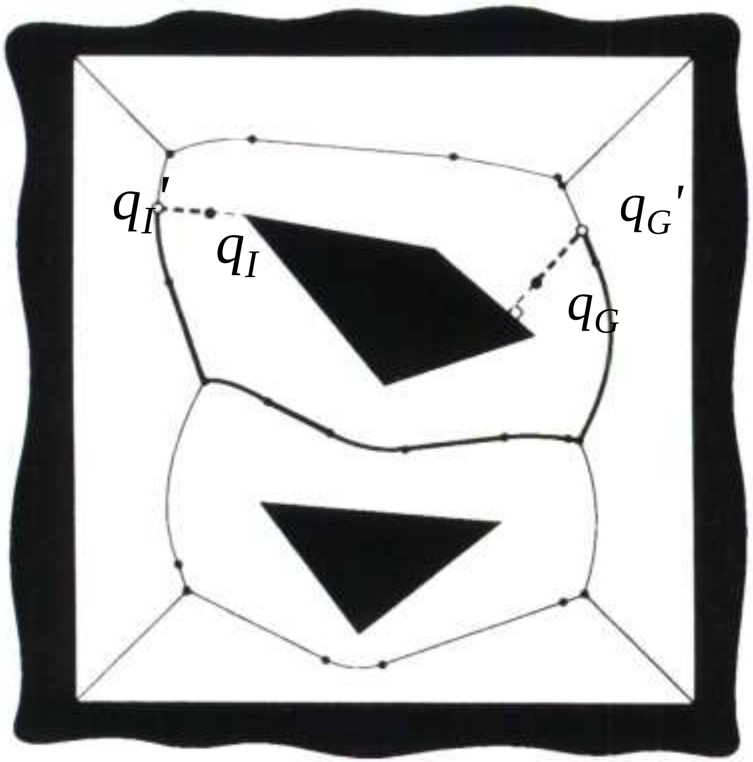


Diagrama de Voronoi

- **Formalmente:**

  Sea $\beta = \partial \mathcal{C}_{free}$ la frontera de C_{free} y $d(p,q)$ la distancia euclídea entre p y q. Entonces, para todo q en C_{free} sea

  $$clearance(q) = \min_{p \in \beta} d(p, q)$$

  La distancia mínima de q, y

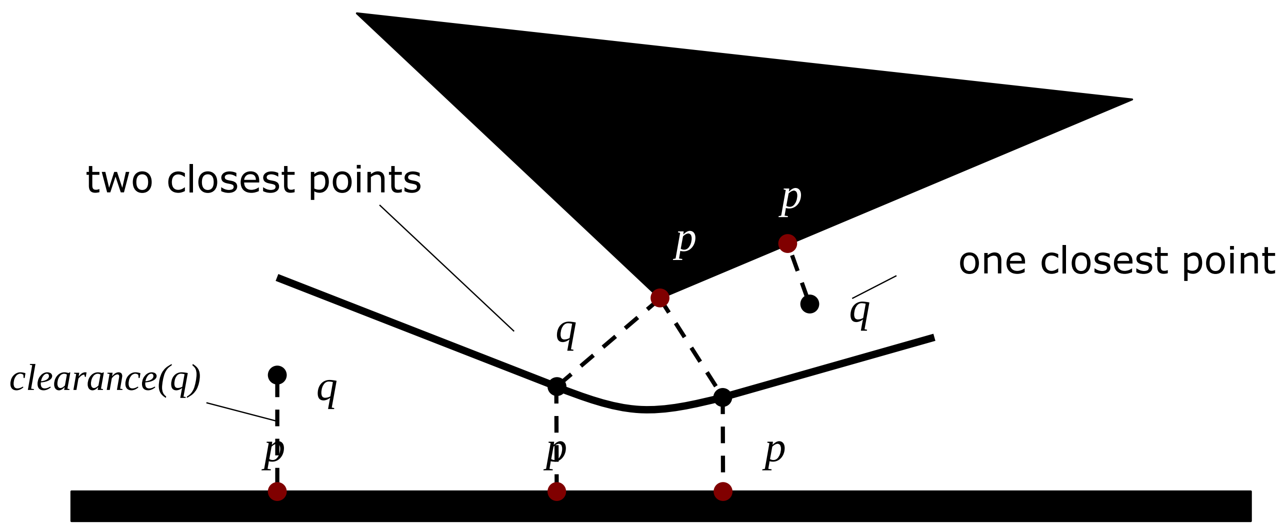
  $$near(q) = \{p \in \beta \mid d(p, q) = clearance(q)\}$$

  el conjunto de puntos de "base" en β con la misma distancia a q. El **Diagrama de Voronoi** es el conjunto de q's con más de un punto de base p

  $$V(\mathcal{C}_{free}) = \{q \in \mathcal{C}_{free} \mid |near(q)| > 1\}$$

Diagrama de Voronoi

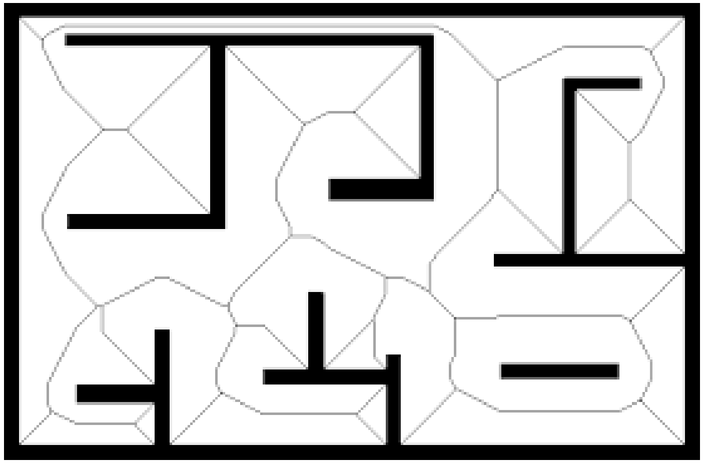
- **Geométricamente:**



- Para un C_{obs} poligonal, el diagrama de Voronoi consta de (n) líneas y segmentos parabólicos
- Algoritmo más simple: $O(n^4)$, mejor: $O(n \log n)$

Diagrama de Voronoi

- Han sido muy estudiados para planeamiento (reactivo) de trayectorias de robots móviles
- Existen métodos rápidos para calcular y actualizar el diagrama en tiempo real para pocas dimensiones de C
  - **Pros:** maximiza distancia a obstáculos (bueno por incertezas)
  - **Contras:** atracción a espacios abiertos, caminos subóptimos



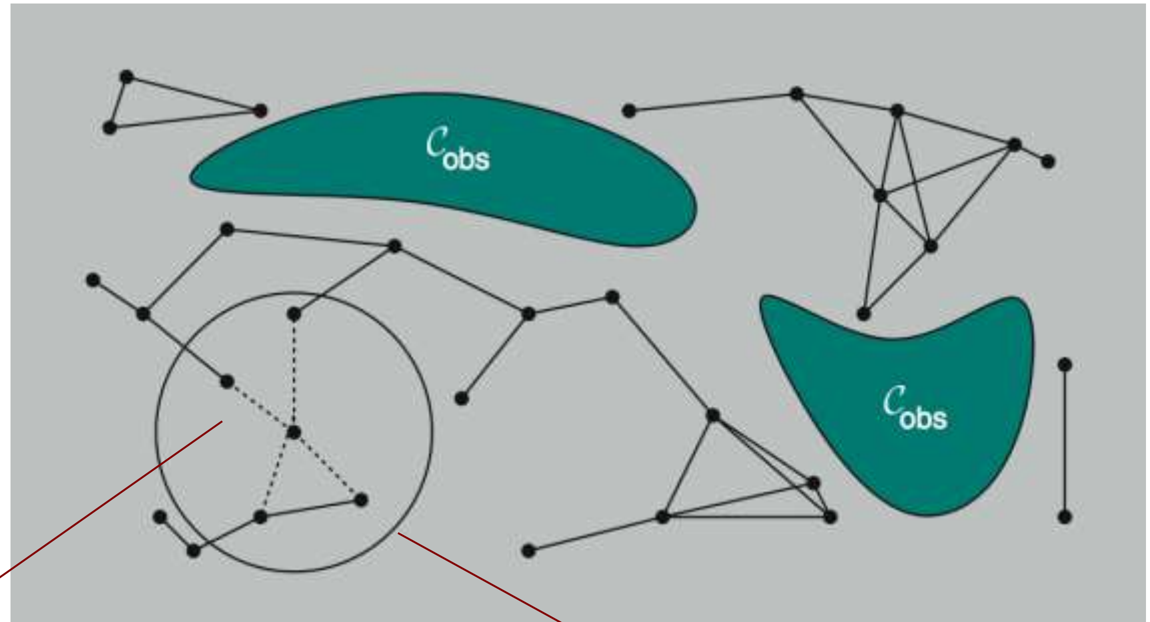
Mapas de ruta probabilísticos

También llamados mapas de ruta aleatorios

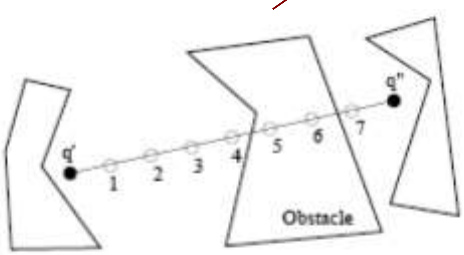
- **Idea:** tomar muestras aleatorias de C, declararlos nodos si están en C_{free}, tratar de conectar nodos cercanos con planeamiento local
- El **planeamiento local** chequea si las conexiones rectas están libres de colisiones
- Se agregan configuraciones y conexiones hasta que el mapa de ruta es **suficientemente denso**

Mapas de ruta probabilísticos

- Ejemplo



Nodos cercanos según un radio especificado

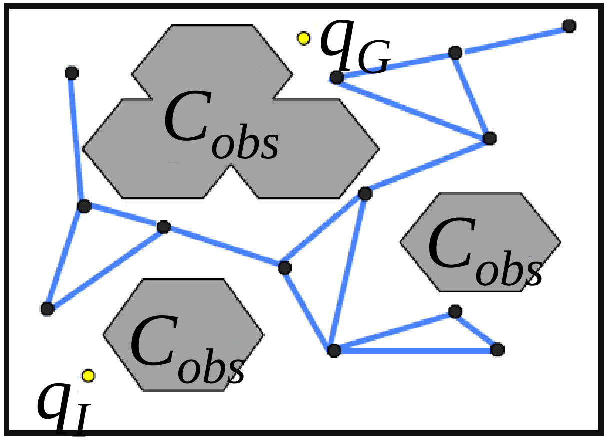


Ejemplo planeamiento local

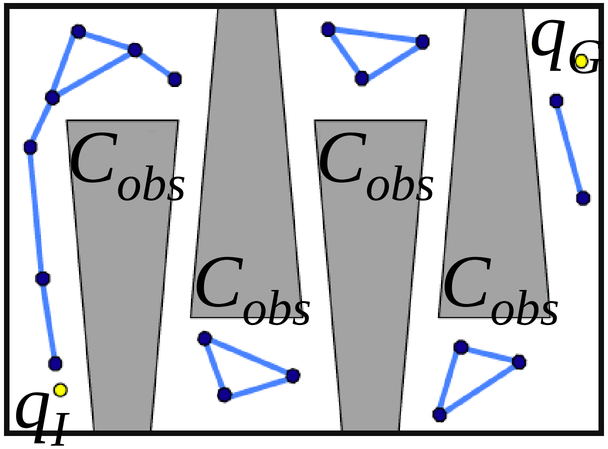
¿Qué significa "cercano" en un manifold? Definir una buena métrica en C es crucial

Mapas de ruta probabilísticos

- **Pros:**
  - *Probabilísticamente completo*
  - No desarrolla todo el espacio C
  - Fácil de aplicar para muchas dimensiones del espacio C
  - Resuelve problemas que otros métodos no pueden resolver



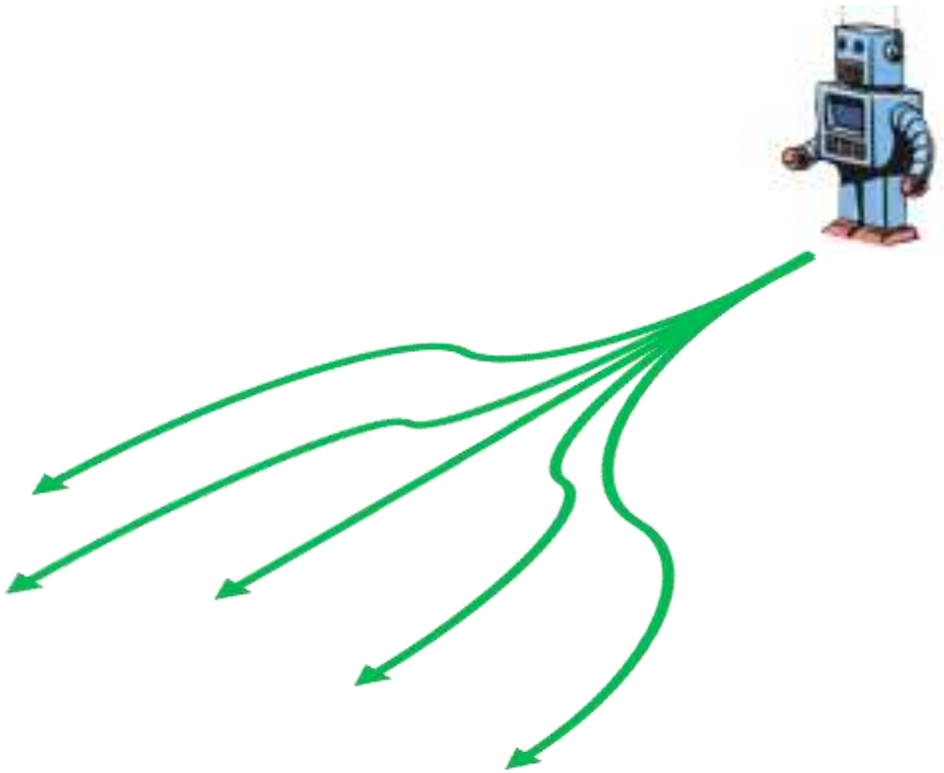
- **Contras:**
  - No funciona bien para algunos problemas: ej: pasajes angostos
  - No es óptimo, no es completo



Mapas de ruta probabilísticos

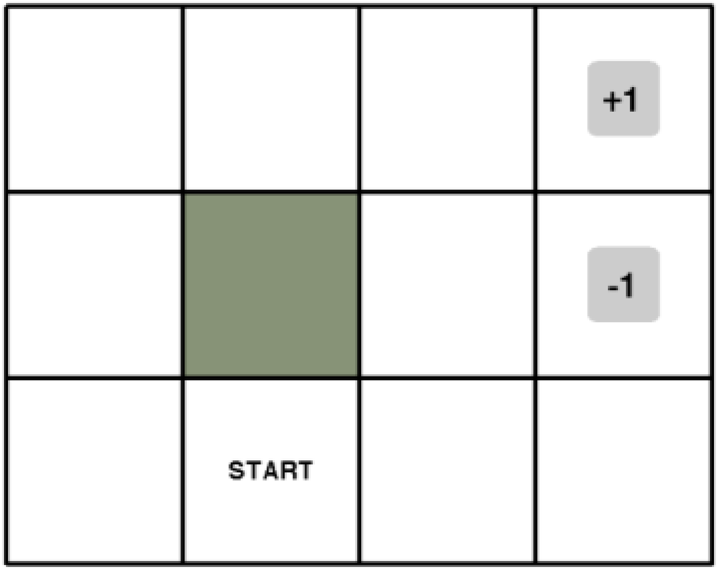
- Los métodos considerados hasta ahora **construyen un mapa de ruta** (sin considerar el par de interés q_I y q_G)
- Una vez que está generado, el **mismo mapa de ruta** se puede reusar para cualquier par de interés (si el mundo y el robot no cambian)
- Trayectoria:
  1. **Encontrar** la celda/nodo que contiene (o está cerca de) q_I y q_G
  2. **Conectar** q_I y q_G al mapa de ruta
  3. **Buscar** en el mapa de ruta un camino de q_I a q_G

Planeamiento e incerteza



Procesos de decisión de Markov (MDP)

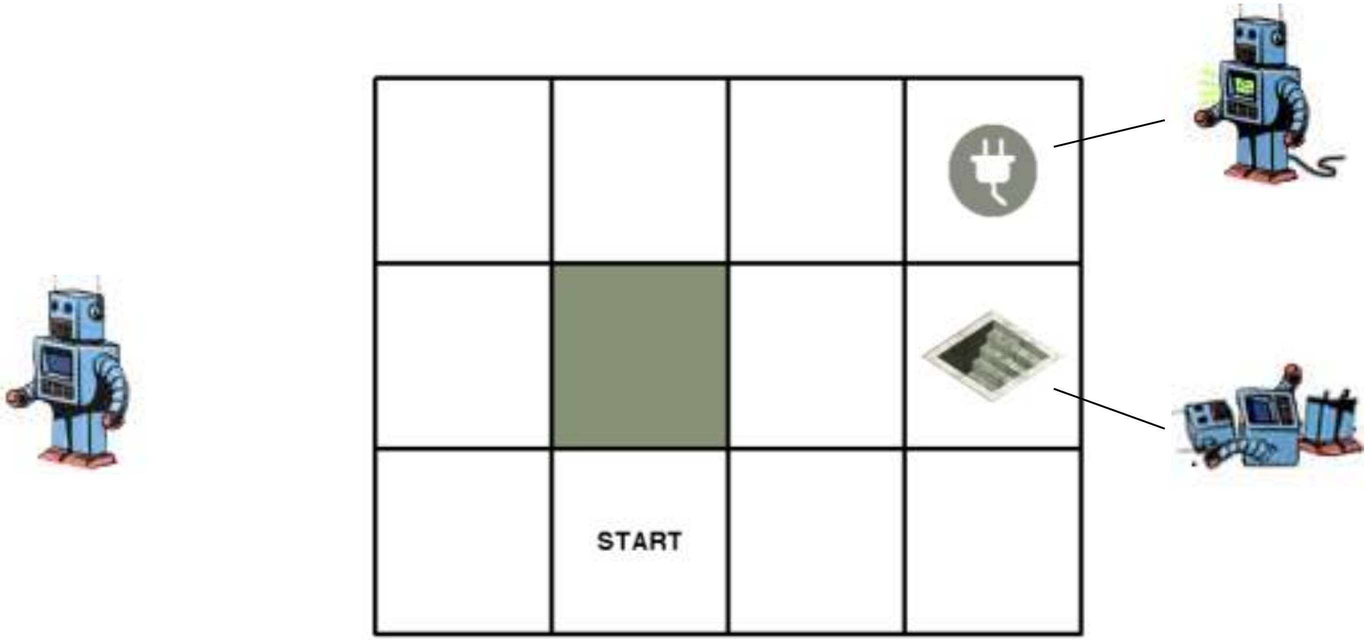
- Considerar un agente en este entorno



- Su misión es alcanzar el objetivo marcado como +1 evitando la celda marcada como -1

Procesos de decisión de Markov

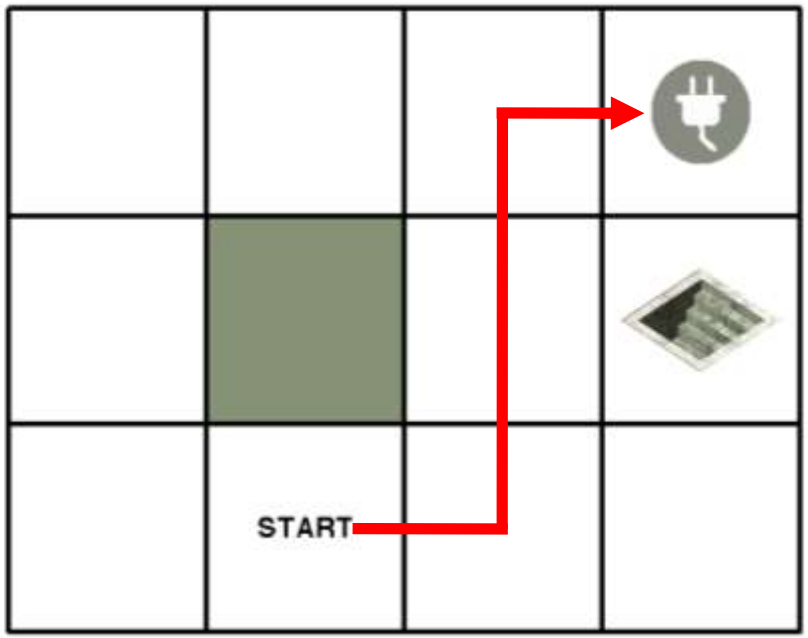
- Considerar un agente en este entorno



- Su misión es alcanzar el objetivo marcado como +1 evitando la celda marcada como -1

Procesos de decisión de Markov

- Fácil! Usar un algoritmo de búsqueda, por ejemplo: A*



- Solución (camino más corto):
  - Secuencia de acciones *[derecha, arriba, arriba, derecha]*

¿Cuál es el problema?

- Considerar un sistema que no es perfecto en el que las acciones se ejecutan con **probabilidad menor a 1**
- ¿Cuáles son las mejores acciones para un agente en estas circunstancias?

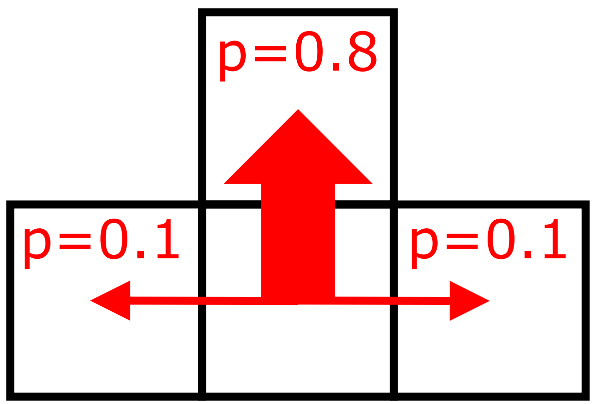
- Ejemplo: un robot móvil no hace exactamente lo que se desea
- Ejemplo: navegación humana

→ Incerteza en la ejecución de acciones!

Ejemplo MDP

- Considerar el modelo de transición no-determinístico (N / E / S / O):

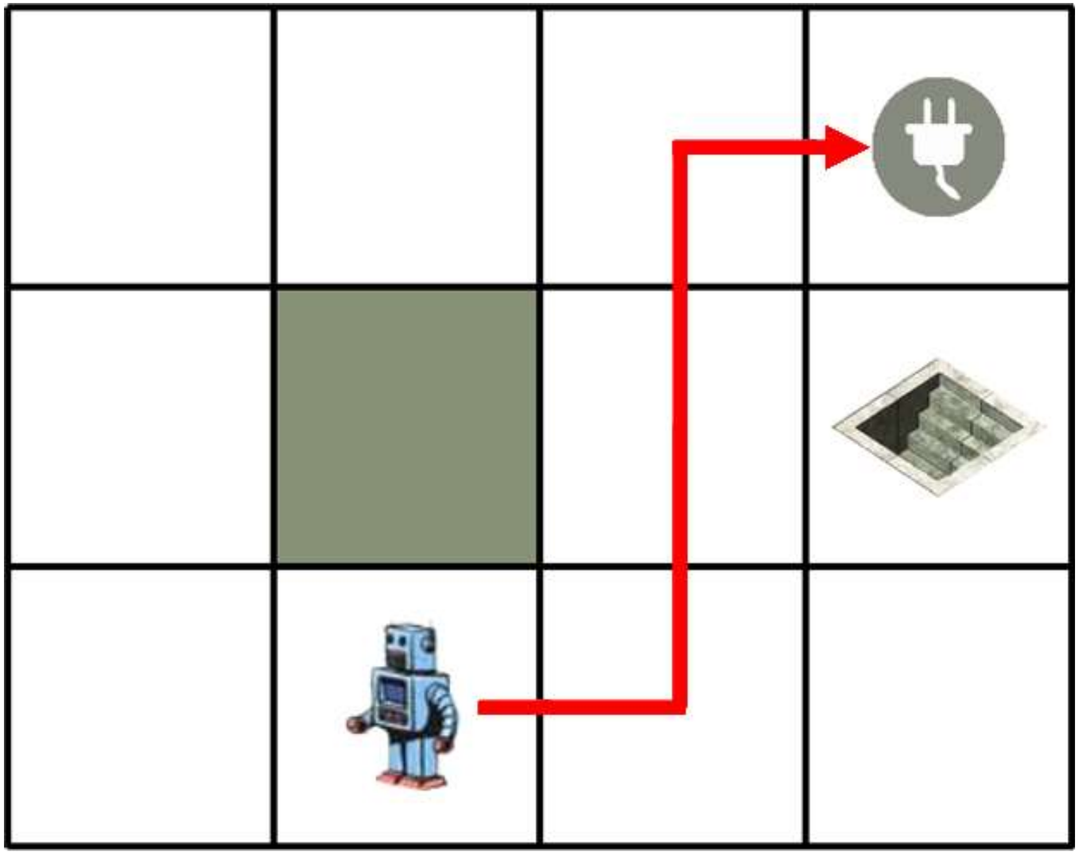
Acción deseada: N



- La acción deseada se ejecuta con p=0.8
- Con p=0.1, el agente se mueve a la izq. o derecha
- El agente "rebota" en las paredes

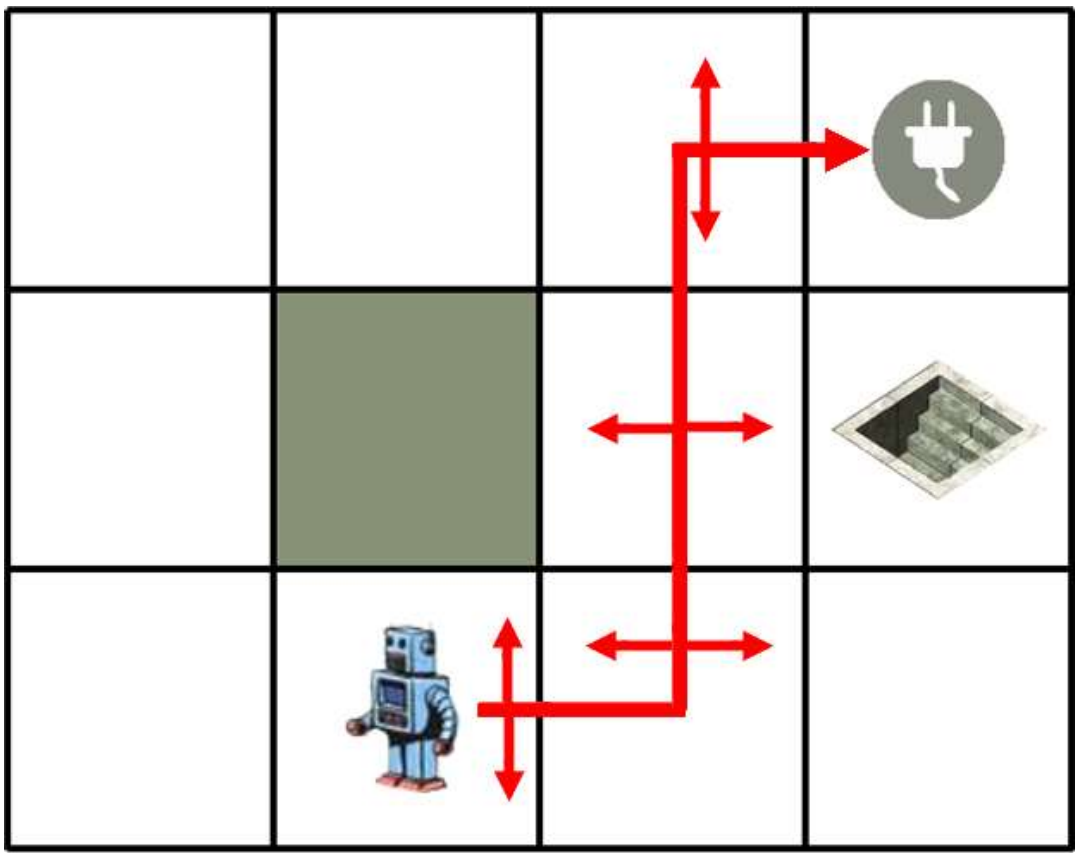
Ejemplo MDP

- Ejecutando un **plan A***



Ejemplo MDP

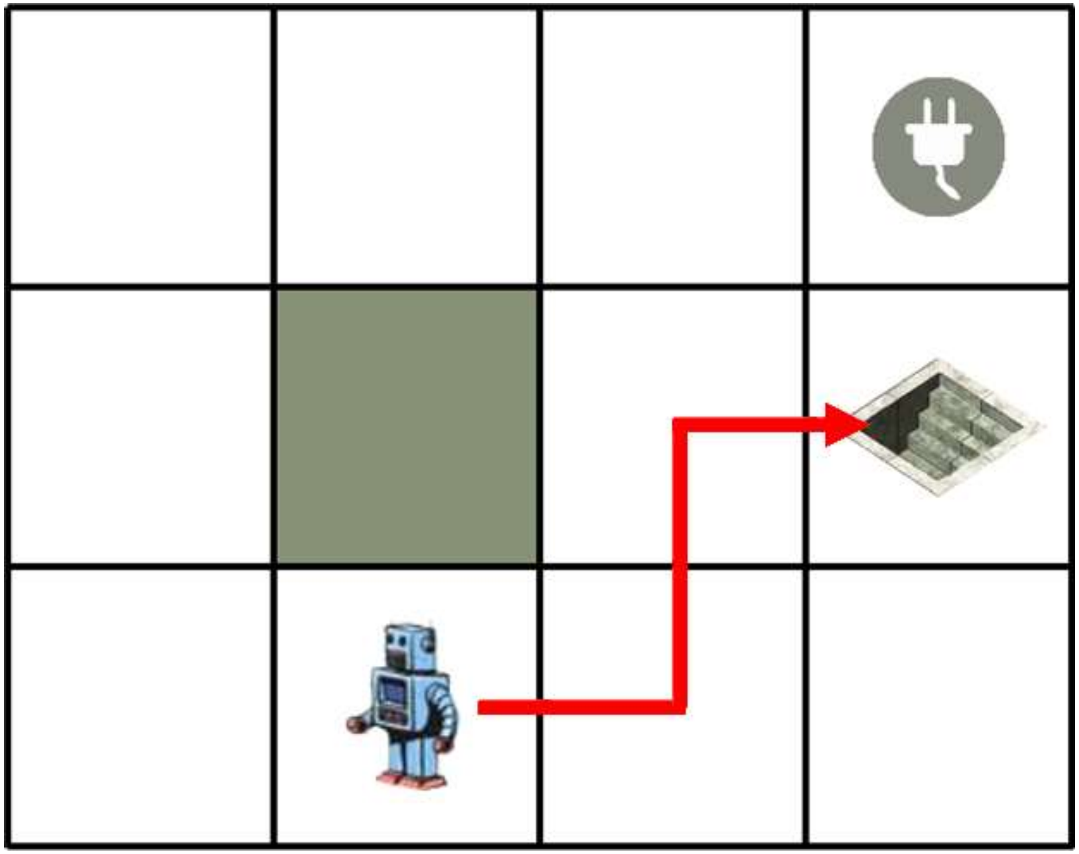
- Ejecutando un **plan A***



las transiciones no son determinísticas!

Ejemplo MDP

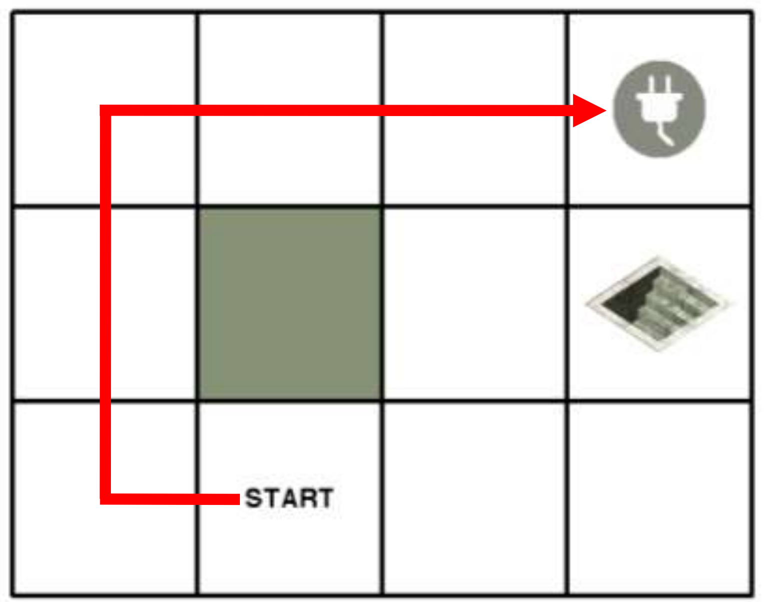
- Ejecutando un **plan A***



En algún momento va a pasar esto…

Ejemplo MDP

- Usar un camino **más largo** con **menor** probabilidad de terminar en la celda **-1**



- Este camino tiene la **utilidad total más alta**
- Probabilidad $0.8^6 = 0.2621$

Modelo de transición

- La probabilidad de llegar al próximo estado s' desde s ejecutando la acción a

  $$T(s, a, s')$$

  se llama **modelo de transición**

Propiedad de Markov:

La probabilidad de transición de s a s' **depende sólo del estado actual** s y no de los estados anteriores

Recompensa

- En cada estado s, el agente recibe una **recompensa** $R(s)$

- La recompensa puede ser positiva o negativa, pero debe estar **acotada**

- Se puede generalizar a una función $R(s,a,s')$.

  Nota: considerando solo $R(s)$ no cambia el problema

Recompensa

- En nuestro ejemplo, la recompensa es **-0.04** en todos los estados (costo de movimiento) excepto los estados terminales que tienen recompensas **+1/-1**

- Una recompensa negativa es un **incentivo** para llegar al objetivo **rápido**

- concepto: “estar en este entorno no es agradable”

-0.04	-0.04	-0.04	+1
-0.04		-0.04	-1
-0.04	-0.04	-0.04	-0.04

Definición de MDP

- Dado un **problema de decisión secuencial** en un entorno observable estocástico con un modelo de transición de Markov

- Entonces un **Proceso de Decisión de Markov** se define por los componentes
  - *Conjunto de estados:* S
  - *Conjunto de acciones:* A
  - *Estado inicial:* s_0
  - *Modelo de transición:* $T(s, a, s')$
  - *Función de recompensa:* $R(s)$

Política

- Una solución de MDP se denomina **política** π
- Una política es un mapeo de estados a acciones

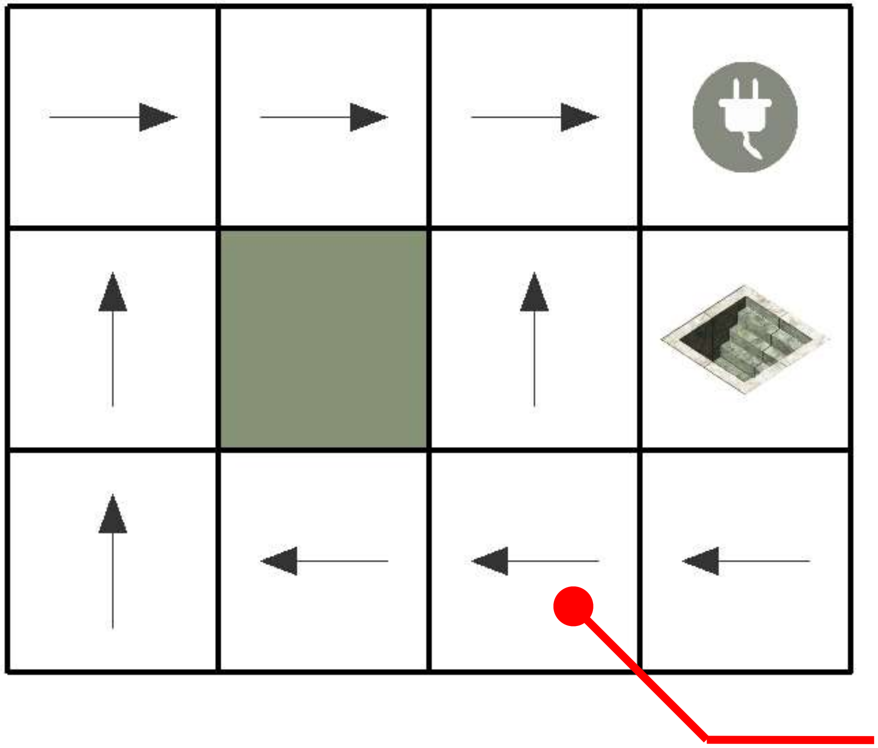
$$\textit{Política} : \textit{Estados} \longrightarrow \textit{Acciones}$$

- En cada estado, una política le dice al agente **que hacer** en el próximo paso
- Sea $\pi(s)$ la acción que π especifica para s
- De todas las políticas que resuelven un MDP, nos interesa la **política óptima** π^*.

  Definiremos luego qué entendemos por óptima

Política

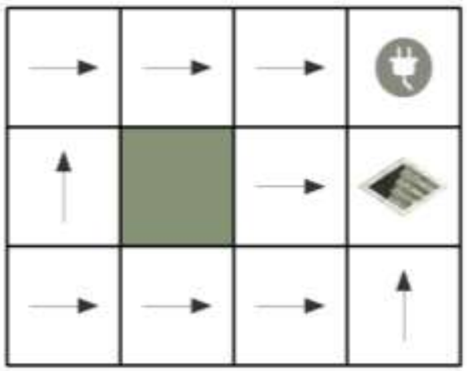
- La política óptima para nuestro ejemplo



Elección conservadora: dar toda la vuelta, ya que el costo por paso de -0.04 es chico comparado con el costo de caer por la escalera y recibir una recompensa de **-1**

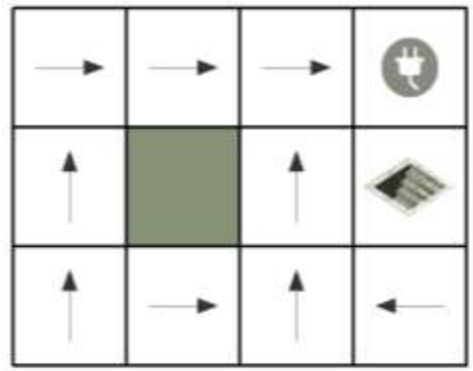
Política

- Cuando el balance entre riesgo y recompensa cambia, **otras políticas son óptimas**



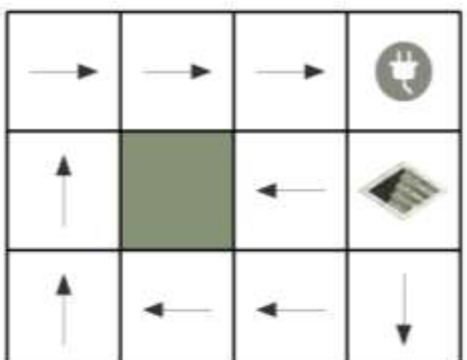
R = -2

Irse lo antes posible



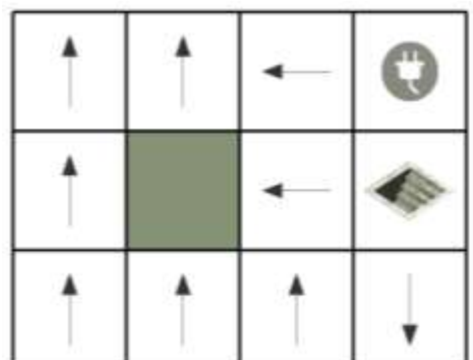
R = -0.2

Tomar atajo, poco riesgo



R = -0.01

No se toman riegos



R > 0

No irse nunca

Utilidad de un estado

- La **utilidad de un estado** $U(s)$ cuantifica el **beneficio** de un estado para la **tarea general**
- Definimos $U^{\pi}(s)$ como la **utilidad esperada de todas las secuencias de estados que empiezan en** s dado π

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} R(s_t) \mid \pi, s_0 = s\right]$$

- $U(s)$ evalúa (y encapsula) todos los posibles futuros desde s **en adelante**

Utilidad de un estado

- Con esta definición, podemos expresar $U^{\pi}(s)$ como una **función de su próximo estado** s'

$$\begin{aligned}
U^{\pi}(s) &= E\left[\sum_{t=0}^{\infty} R(s_t) \mid \pi, s_0 = s\right] \\
&= E\left[R(s_0) + R(s_1) + R(s_2) + \ldots \mid \pi, s_0 = s\right] \\
&= E\left[R(s_0) \mid s_0 = s\right] + E\left[R(s_1) + R(s_2) + \ldots \mid \pi\right] \\
&= R(s) + E\left[\sum_{t=0}^{\infty} R(s_t) \mid \pi, s_0 = s'\right] \\
&= R(s) + U^{\pi}(s')
\end{aligned}$$

Política óptima

- La utilidad de un estado nos permite aplicar el **Principio de Máxima Utilidad Esperada** para definir la política óptima π^*

- La **política óptima** π^* en s elige la acción a que maximiza la utilidad esperada de s (y de s')

$$\pi^*(s) \quad = \quad \underset{a}{\operatorname{argmax}} \; E\left[U^{\pi}(s)\right]$$

U esperada tomada sobre todas las políticas

Política óptima

- Sustituyendo $U^{\pi}(s)$

$$
\begin{aligned}
\pi^*(s) &= \underset{a}{\operatorname{argmax}}\ E\left[U^{\pi}(s)\right] \\
&= \underset{a}{\operatorname{argmax}}\ E\left[R(s) + U^{\pi}(s')\right] \\
&= \underset{a}{\operatorname{argmax}}\ E\left[R(s)\right] + E\left[U^{\pi}(s')\right] \\
&= \underset{a}{\operatorname{argmax}}\ E\left[U(s')\right] \\
&= \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')\,U(s')
\end{aligned}
$$

- Recordar que $E[X]$ es el promedio pesado de los posibles valores que X puede tomar

Utilidad de un estado

- La **utilidad verdadera de un estado** *U(s)* se obtiene aplicando la política óptima
  $U^{\pi^*}(s) = U(s)$

$$
\begin{aligned}
U(s) &= \max_a E\left[U^{\pi}(s)\right] \\
&= \max_a E\left[R(s) + U^{\pi}(s')\right] \\
&= \max_a E\left[R(s)\right] + E\left[U^{\pi}(s')\right] \\
&= R(s) + \max_a E\left[U(s')\right] \\
&= R(s) + \max_a \sum_{s'} T(s,a,s')\,U(s')
\end{aligned}
$$

Utilidad de un estado

- Este resultado es importante:

$$U(s) = R(s) + \max_a \sum_{s'} T(s, a, s') U(s')$$

  Es una relación directa entre la **utilidad de un estado** y la **utilidad de sus vecinos**

- La utilidad de un estado es la recompensa inmediata de ese estado más la utilidad esperada del próximo estado, **en el caso en que** el agente elige la acción **óptima**

Ecuación de Bellman

$$U(s) = R(s) + \max_a \sum_{s'} T(s,a,s')\,U(s')$$

- Para cada estado hay una ecuación de Bellman para calcular su utilidad
- Hay ***n* estados** y ***n* incógnitas**
- ¿Resolver usando álgebra lineal?
- ¡No! El operador *max* que elige la acción óptima hace que el sistema sea no-lineal
- Hay que usar un método **iterativo**

Utilidad de un estado

- Las **utilidades de estado** para nuestro ejemplo

0.812	0.868	0.918	+1
0.762		0.66	-1
0.705	0.655	0.611	0.388

- Notar que las utilidades son más altas cercanas al objetivo, ya que se necesitan menos pasos para alcanzarlo

Cálculo Iterativo

Idea:

- La utilidad se calcula iterativamente:

$$U_{i+1}(s) \leftarrow R(s) + \max_a \sum_{s'} T(s,a,s')\,U_i(s')$$

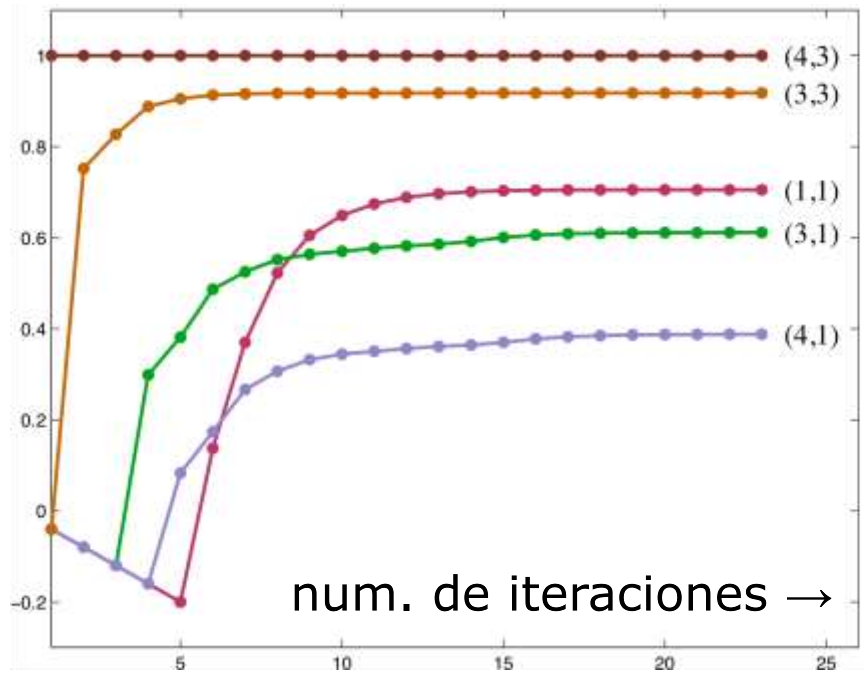
- Utilidad óptima $U^* = \lim_{t \to \infty} U_t$
- Parar si la variación de utilidad está por debajo de un umbral

Ejemplo de iteración de valores

- En nuestro ejemplo

0.812	0.868	0.918	+1
0.762		0.66	-1
0.705	0.655	0.611	0.388

(1,1)



- Los estados lejanos al objetivo primero acumulan recompensas negativas hasta que un camino hasta el objetivo es descubierto

Política óptima

- ¿Cómo calcular la **política óptima**? Se extrae a lo largo del camino con

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s, a, s') U(s')$$

- **Notar:** *U(s)* y *R(s)* son valores muy diferentes. *R(s)* es la recompensa de **corto plazo** por estar en *s*, y *U(s)* es la recompensa de **largo plazo** desde *s* **en adelante**

Resumen

- Una navegación robusta requiere una combinación de planeamiento y evasión de obstáculos.
- Se debe considerar las restricciones cinemáticas del robot, o sea, planear en el espacio de velocidades.
- Una combinación de técnicas de búsqueda y reactivas logran mejores resultados que un método DWA puro en muchas situaciones.
- Usando el método 5D, se contemplan las características del hardware utilizado.
- El método 5D resulta en caminos cercanos al óptimo.

Resumen

- El planeamiento es un problema complejo.
- Se hace foco en un subconjunto del espacio de configuración:
  - mapas de ruta,
  - grillas.
- Los algortimos de muestreo son más rápidos y conllevan una solución de compromiso entre optimalidad y velocidad.
- La incerteza en el movimiento lleva a la necesidad de implementar Problemas de Decisión de Markov.

¿Qué faltaría?

- Vehículos más complejos (ej: autos, robots caminadores -con patas-, manipuladores, …).
- Obstáculos móviles, predicción de movimiento.
- Espacios de dimensión alta.
- Heurísticas para mejorar la performance.
- Aprendizaje automático.