Introducción a la Robótica Móvil

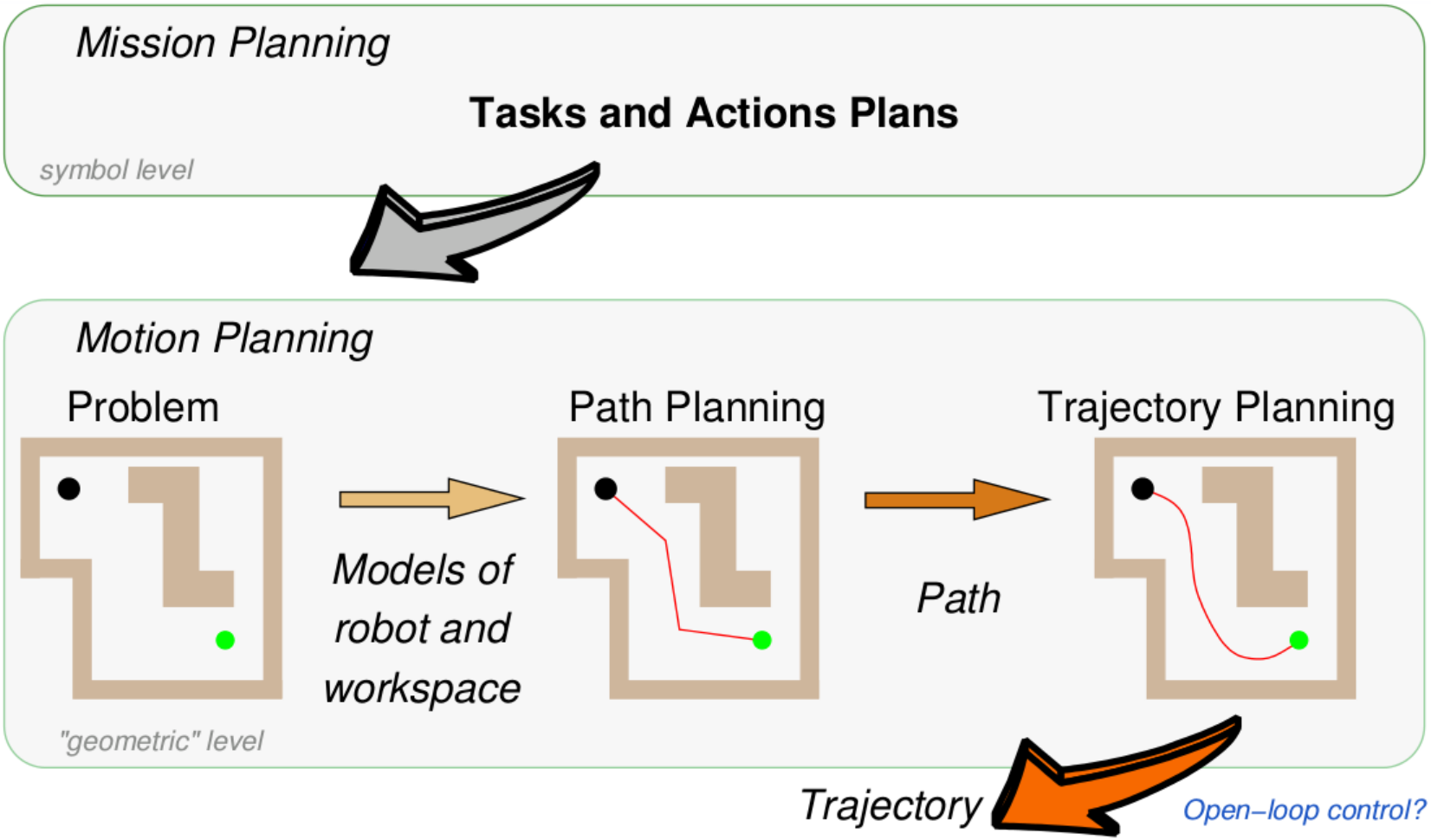
Primer cuatrimestre de 2018

Departamento de Computación - FCEyN - UBA

Planificación de caminos - clase 15

Algoritmo A* sobre celdas de ocupación (*occupancy grids*)

Contexto

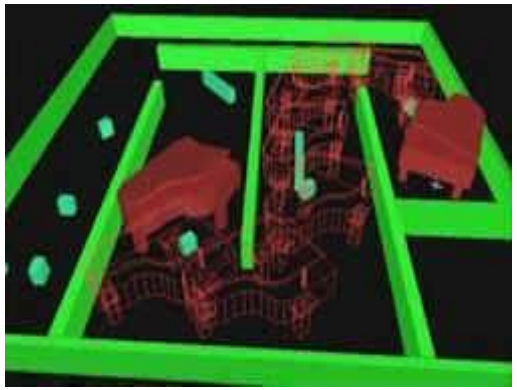

Mission Planning
Tasks and Actions Plans
symbol level
Motion Planning
Problem
Path Planning
Trajectory Planning
Models of robot and workspace
Path
"geometric" level
Trajectory
Open–loop control?
Robot Control
Sensing and Acting
feedback control
controller – drives (motors) – sensors
"physical" level

Motivación

- Queremos traducir una tarea de alto nivel especificada en términos humanos en una descripción de bajo nivel que sea apropiada para el controlador de los actuadores del robot.
- La planificación de movimiento (*motion planning*) nos proveen esas transformaciones para poder mover el robot considerando todas las restricciones que hay en el sistema.
- La planificación de movimientos se puede dividir a su vez distintos niveles: la planificación de caminos (*path planning*), la planificación de trayectorias (*trajectory planning*), el seguimiento de la trayectoria (*trajectory following*) y el control de los actuadores (*motion control*).
- La planificación de caminos consiste en encontrar un camino que sea factible (lista de poses que el robot pueda alcanzar) y seguro (libre de colisiones) entre un punto inicial (*start*) y un punto final (*goal*).

Motivación

Un problema icónico de planificación de caminos es el problema de mover un piano: teniendo un modelo 3D de un piano y un modelo 3D del entorno, tenemos que resolver cómo mover el piano desde un lugar hasta otro sin golpear nada.

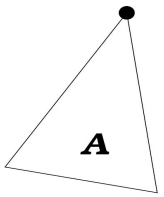

- Necesitamos una representación del modelo y una definición formal del problema.
- También necesitamos una serie de hipótesis para simplificar el problema.

Definición del problema

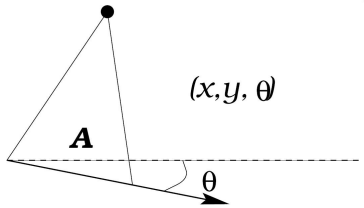
- El problema consiste en computar un camino libre de coliciones para un objeto rígido en movimiento (el robot) a través de obstáculos estáticos.
- Input:
  - Modelo geométrico del robot y de los obstáculos.
  - Modelo de movimiento del robot (modelo cinemático)
  - Configuración inicial y final del robot (posición y orientación)
- Output: secuencia continua de configuraciones del robot libres de colisiones que conectan la configuración inicial con la configuración final.
- Un algoritmo que resuelva completamente este problema (encuentre un camino si existe y reporte que no existe camino en caso contrario) puede tomar tiempo exponencial.
- Por lo tanto, no podemos aplicar este tipo de algoritmo exacto en robótica móvil por restricciones de tiempo real.

Espacio de configuración

- Un concepto clave para la planificación de caminos es la configuración del sistema: una completa especificación de la pose de cada punto del sistema.
- Por ejemplo: para un robot terrestre que se traslada pero no rota en un plano: ¿qué representación resulta suficiente para su configuración?

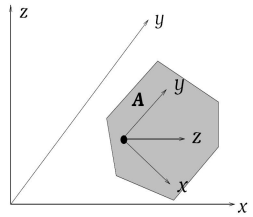


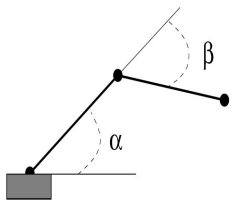
(x,y)



(x,y,θ)

$(x,y,z,\alpha,\beta,\gamma)$





(α,β)

Definiciones

- El espacio de todas las posibles configuraciones del robot es el espacio de configuración o C-space ($\mathcal{C}$).
- La dimensión de C-space es igual al número de variables que tengo para representar una configuración (los grados de libertad!)
- El espacio de trabajo (*workspace*) es el entorno donde opera el robot, i.e. el conjunto de puntos del espacio alcanzables.
- El modelo del mundo ($\mathcal{W}$) describe el espacio de trabajo del robot y sus bordes determinan los obstáculos ($\mathcal{O}_i$).
- Por ejemplo, en un mundo 2D, $\mathcal{W} = \mathbb{R}^2$.
- Si llamamos $\mathcal{R}$ al subconjunto de $\mathcal{W}$ ocupado por el robot, $\mathcal{R} = \mathcal{R}(q)$ con $q \in \mathcal{C}$.
- Si llamamos $\mathcal{C}_{obs}$ al subconjunto de $\mathcal{W}$ ocupado por los obstáculos, $\mathcal{C}_{obs} = \{q \in \mathcal{C} : \mathcal{R}(q) \cap \mathcal{O}_i \neq \emptyset, \forall i\}$
- La configuración libre de colisiones es $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$
- $\mathcal{C}_{free}$ es generalmente un conjunto abierto.

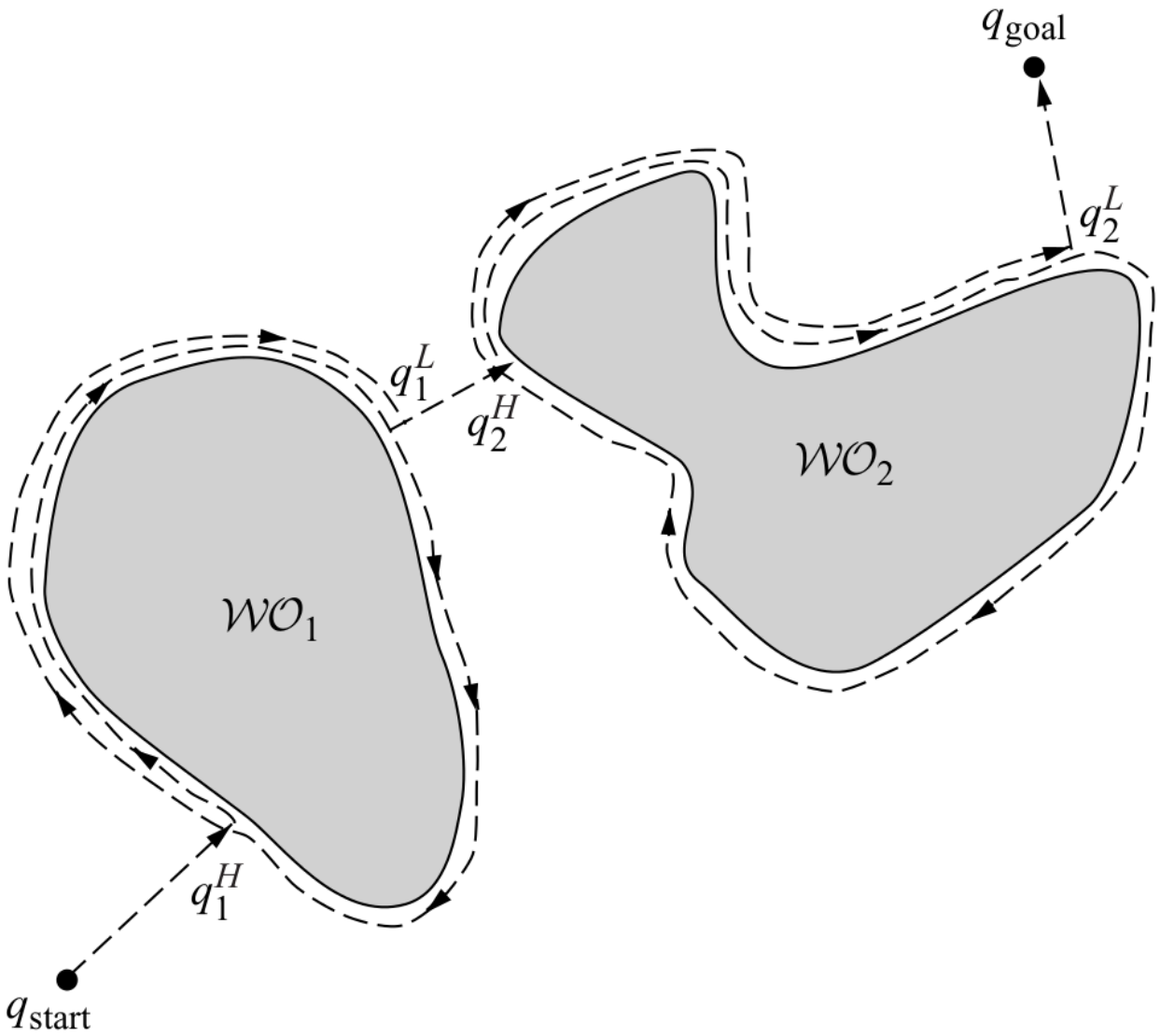
¿Todos los caminos conducen al goal?

- Podemos definir como camino a un mapeo continuo en el espacio de configuración tal que:

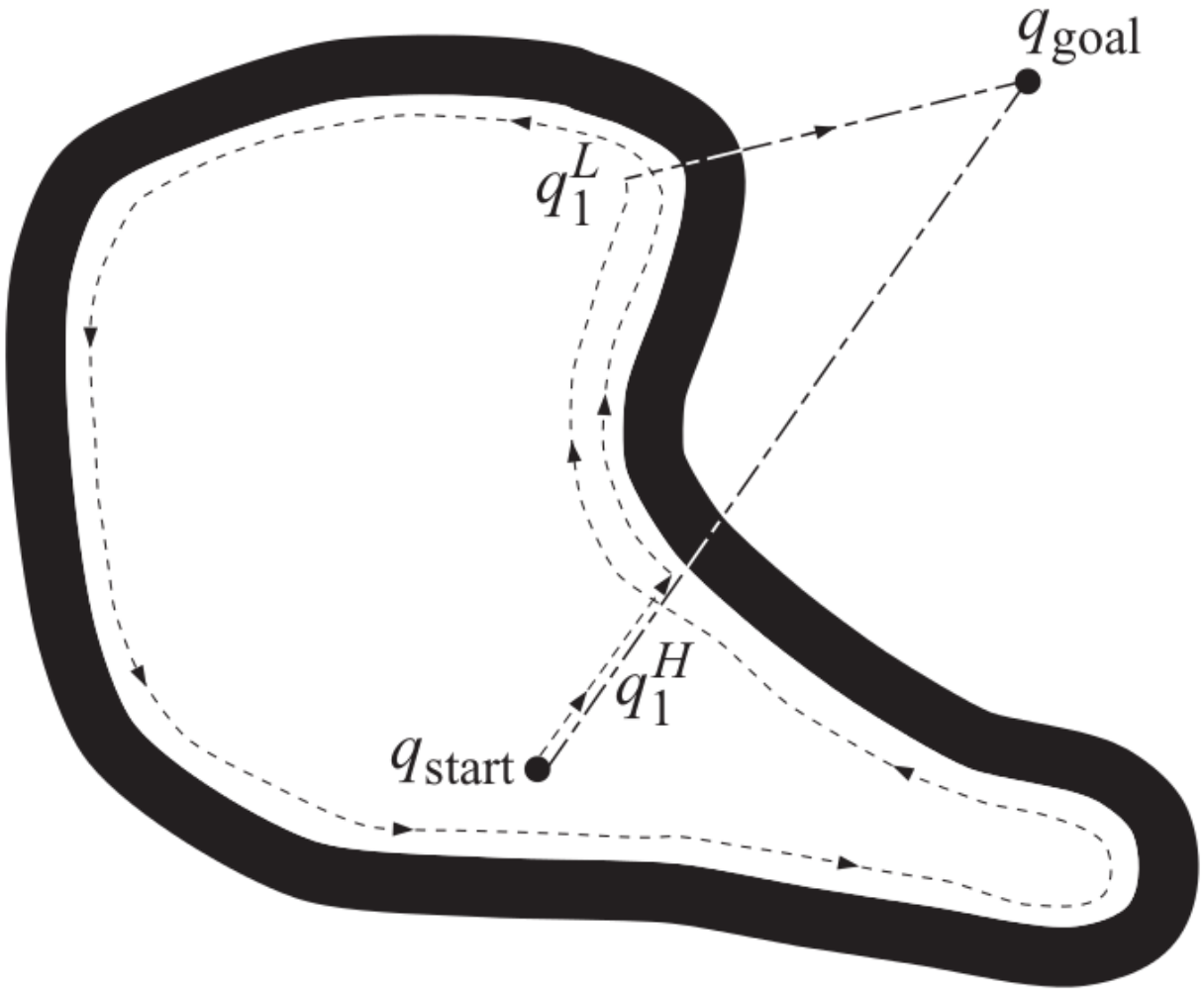
$$\pi : [0, 1] \rightarrow \mathcal{C}_{free}, \text{ con } \pi(0) = q_0 \text{ y } \pi(1) = q_f,$$

  donde q_0 es la configuración inicial (start) y q_f es la final (goal).
- El problema de la planificación de caminos entonces consiste en encontrar la función $\pi(\cdot)$.

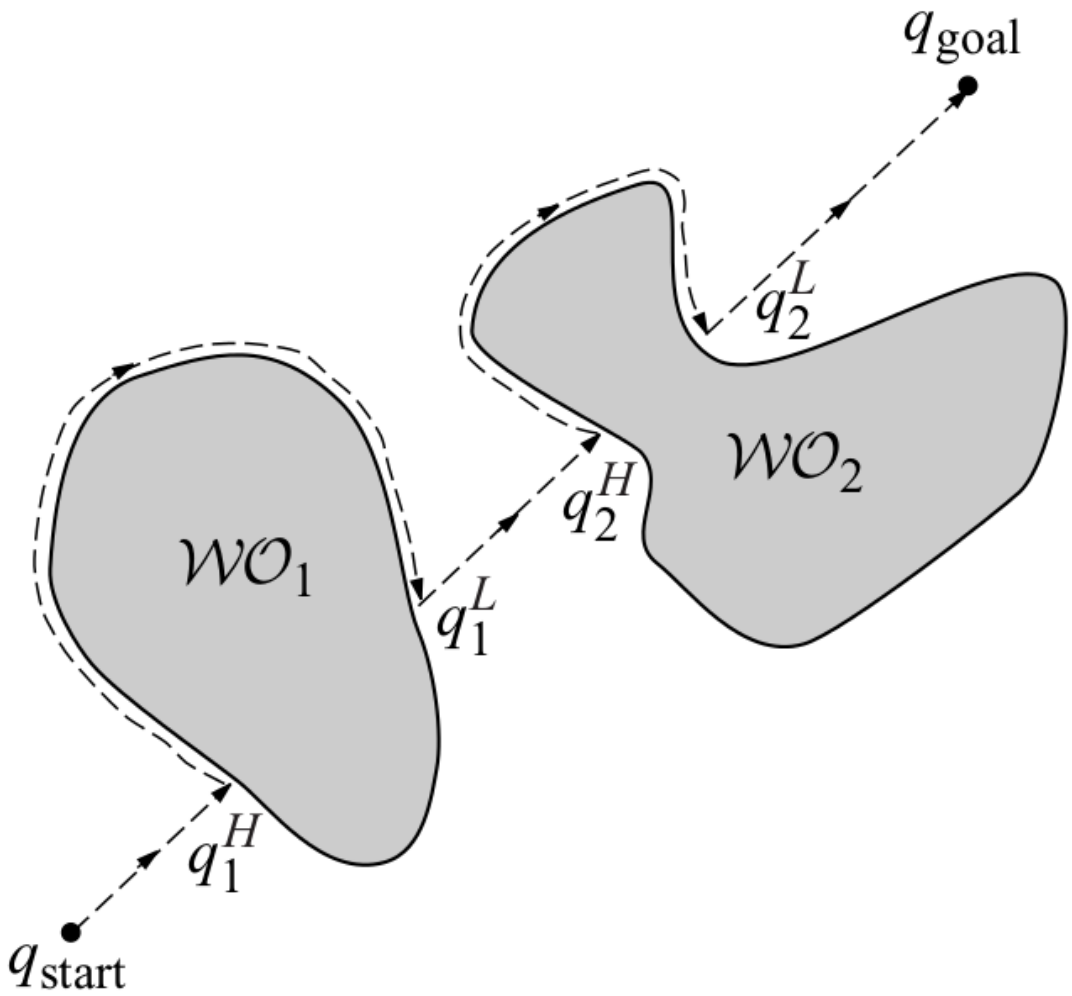
Caso sencillo: Algoritmo Bug1



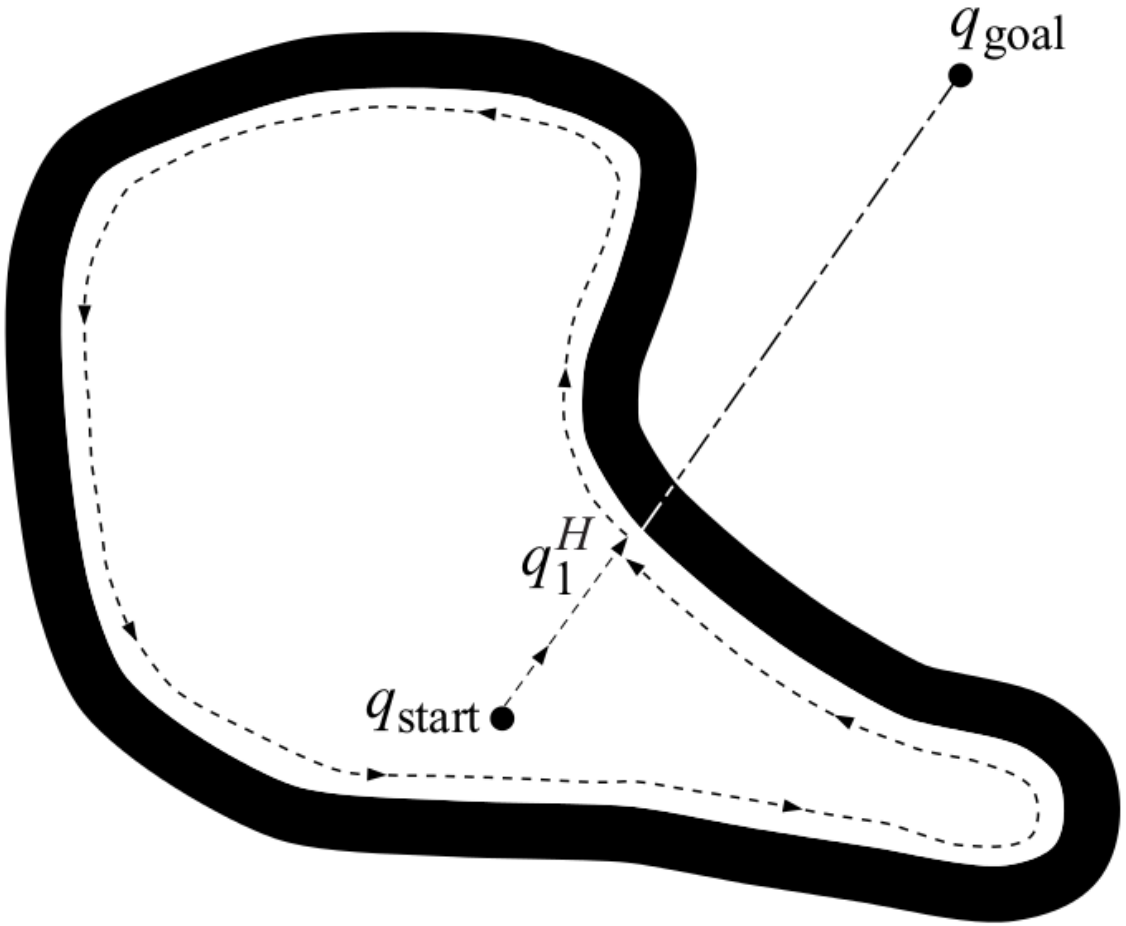
Caso sencillo: Algoritmo Bug1



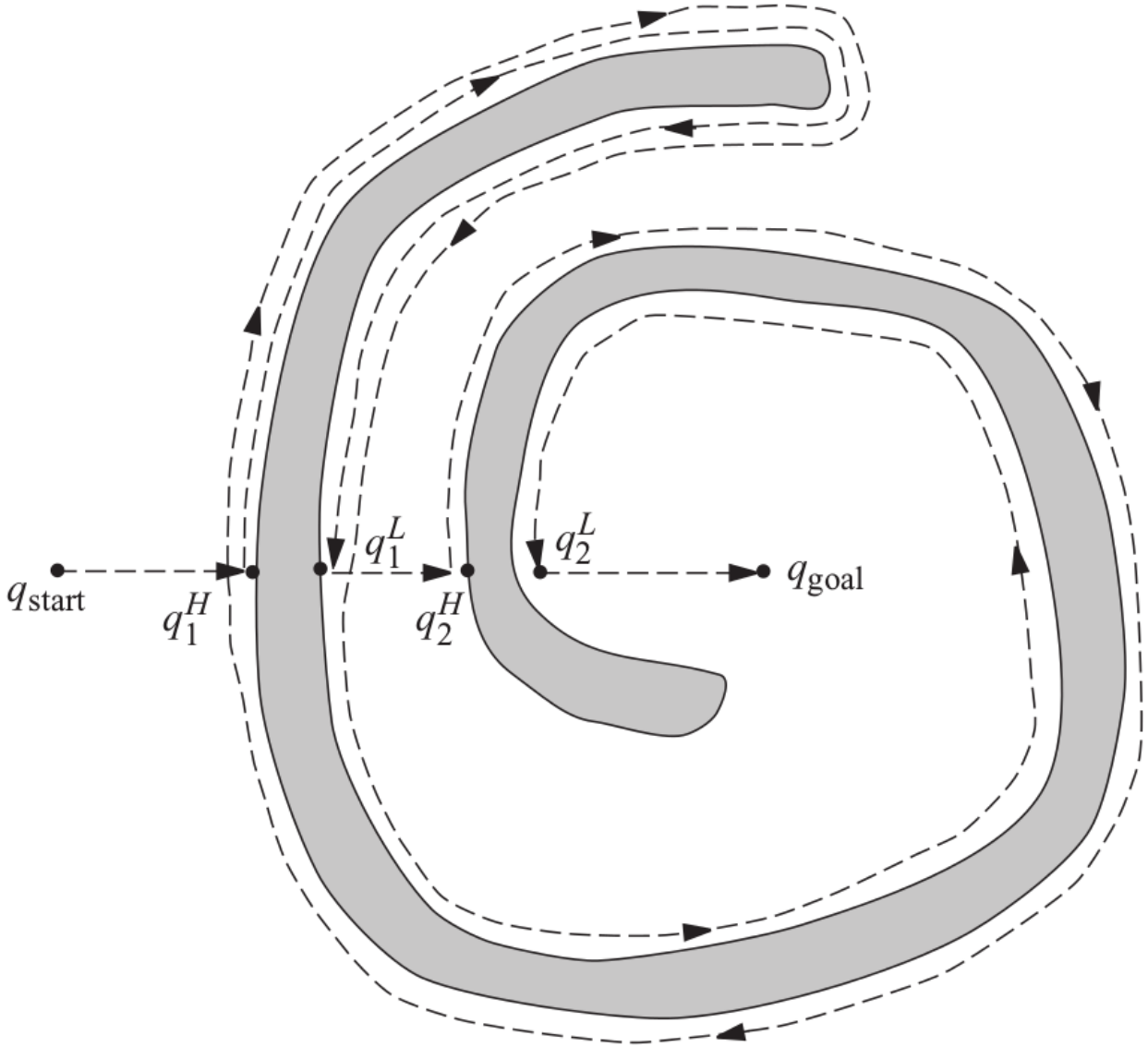
Caso sencillo: Algoritmo Bug2



Caso sencillo: Algoritmo Bug2



¿Cuál es mejor? Bug1 vs Bug2



Problemas

Los algoritmos intuitivos que vimos recién tienen varios problemas:

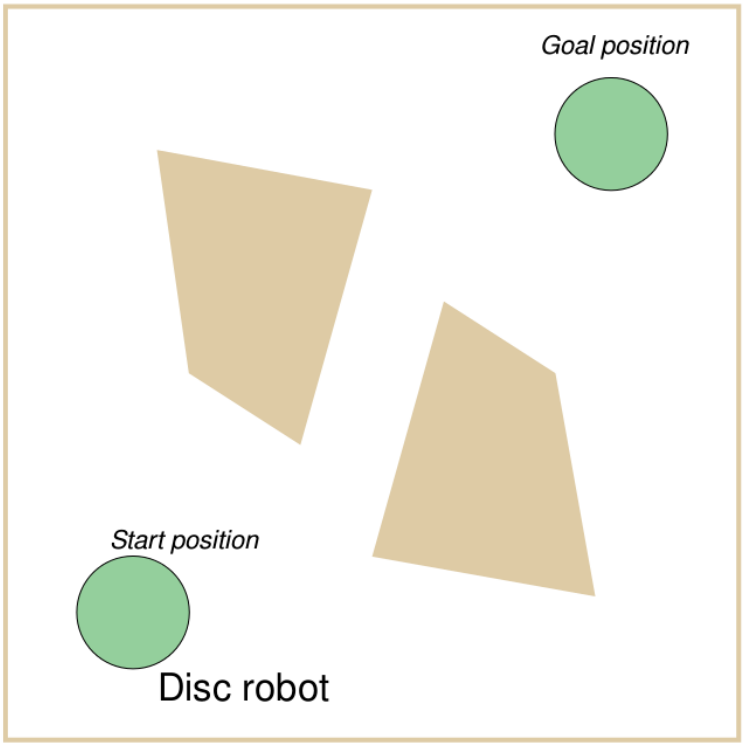
- El robot no puedo considerarlo puntual, tiene un cuerpo.
- No puedo recorrer cada obstáculo cada vez que me encuentro con uno.
- Tenemos que volver al problema original: el problema de mover el piano.

Enfoques para abordar el problema

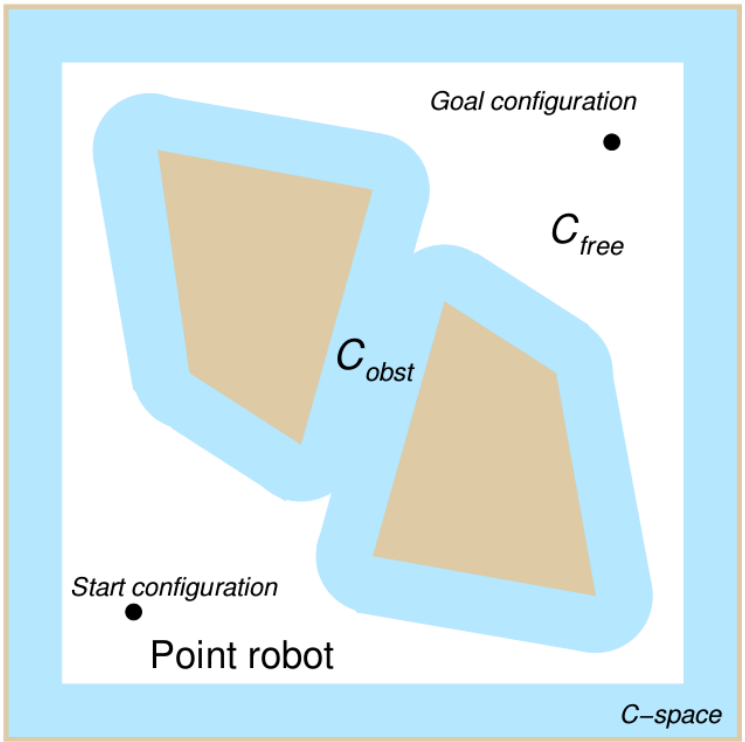
- El problema más general conocido como el problema de mover el piano es PSPACE-hard.
- Algoritmos exactos existen, pero son muy complejos para utilizarse en la práctica en robótica.
- La investigación en planificación de caminos se focalizó en proponer algoritmos aproximados.
- Los enfoques más exitosos son:
  - Basados en descomposición de celdas y grafos: Breadth First Search (BFS), Dijstra, A*.
  - Basados en muestreo aleatorio: Probabilistic RoadMap (PRM), Rapidly-exploring Random Tree (RRT)
  - Basados en muestreo probabilístico óptimo: Rapidly-exploring Random Graph (RRG)

Ejemplo de espacio de configuración $\mathcal{C}-$space

Consideremos la planificación de camino para un robot circular de radio ρ:



Problema de planificación de caminos en la representación geométrica de $\mathcal{W}$.

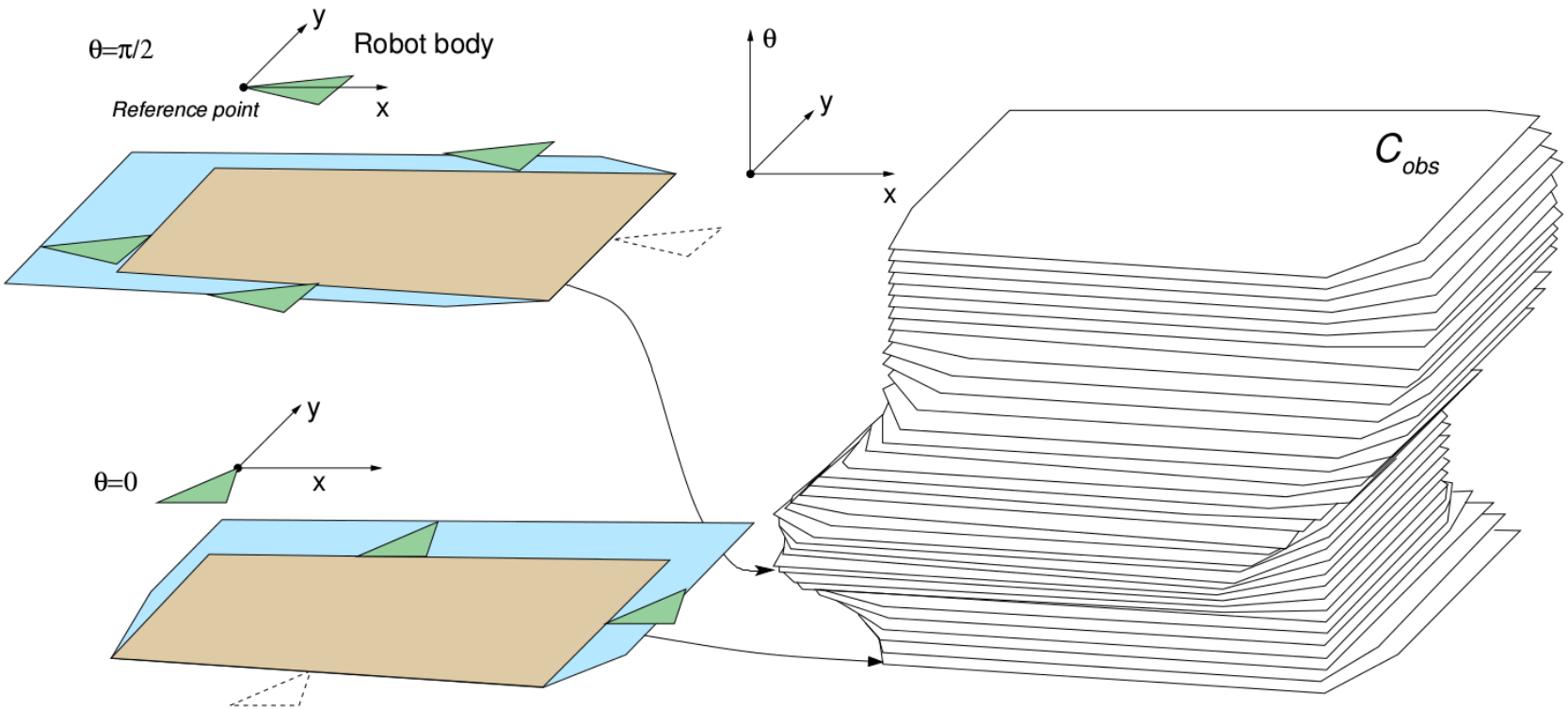


Problema de planificación de caminos en la representación $\mathcal{C}$.

El espacio de configuración ($\mathcal{C}$) se puede obtener agregando a los obstáculos el cuerpo del robot $\mathcal{R}$ de radio ρ:
$\mathcal{C} = \mathcal{O} \oplus \mathcal{R} = \{x + y | x \in \mathcal{O}, y \in \mathcal{R}\}$ donde $\oplus$ es la suma de Minkowski.

Ejemplo de $\mathcal{C}_{obs}$

Consideremos el espacio de los obstáculos de un robot con forma de triángulo donde consideramos la orientación θ:



- Un simple obstáculo en 2D deviene en un complicado $\mathcal{C}_{obs}$.
- Existen algoritmos determinísticos para computar $\mathcal{C}_{obs}$ pero requieren tiempo exponencial respecto de la dimensión de $\mathcal{C}$
- Una representación explícita de $\mathcal{C}_{free}$ es impráctica de computar.

Representación del espacio de configuración $\mathcal{C}-$space

¿Cómo lidiar con la representación del espacio de configuración?

Representación continua de $\mathcal{C}-$space, resulta intratable.

↓

Discretización

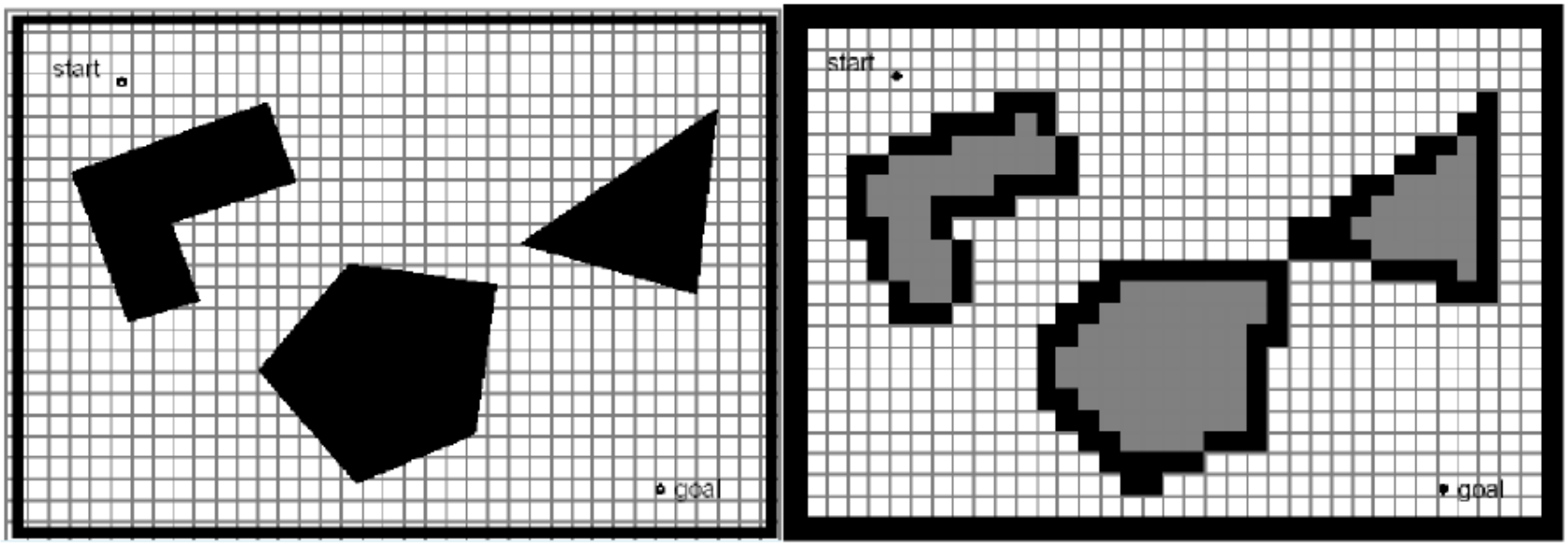
Procesando geométricamente el espacio, haciendo muestreo aleatorio del espacio, con una descomposición en celdas, con campos potenciales, etc.

↓

Técnicas de búsqueda en grafos

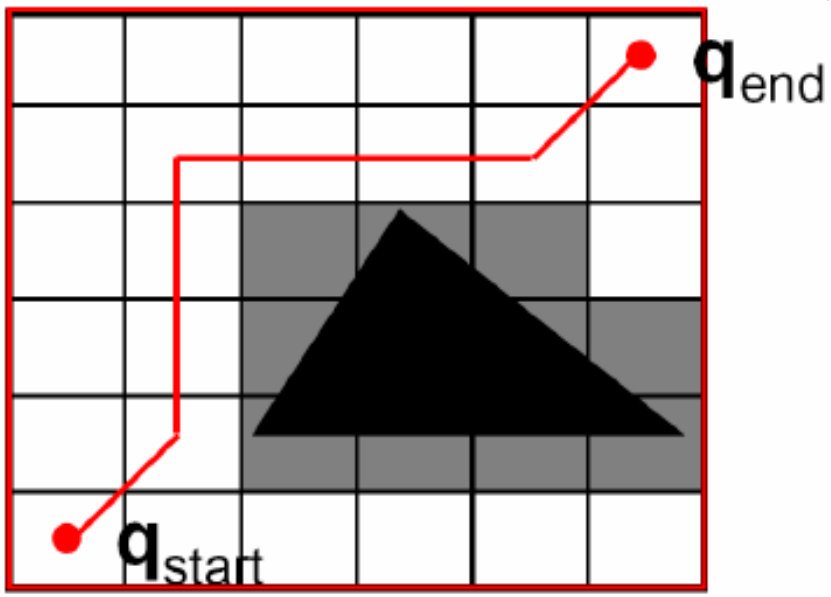
BFS, Búsqueda por gradiente, A*

Descomposición en celdas de ocupación (*occupancy grids*)



- Se define una grilla discreta en el espacio de configuración
- Marcar celdas que intersecan algun objeto como ocupada

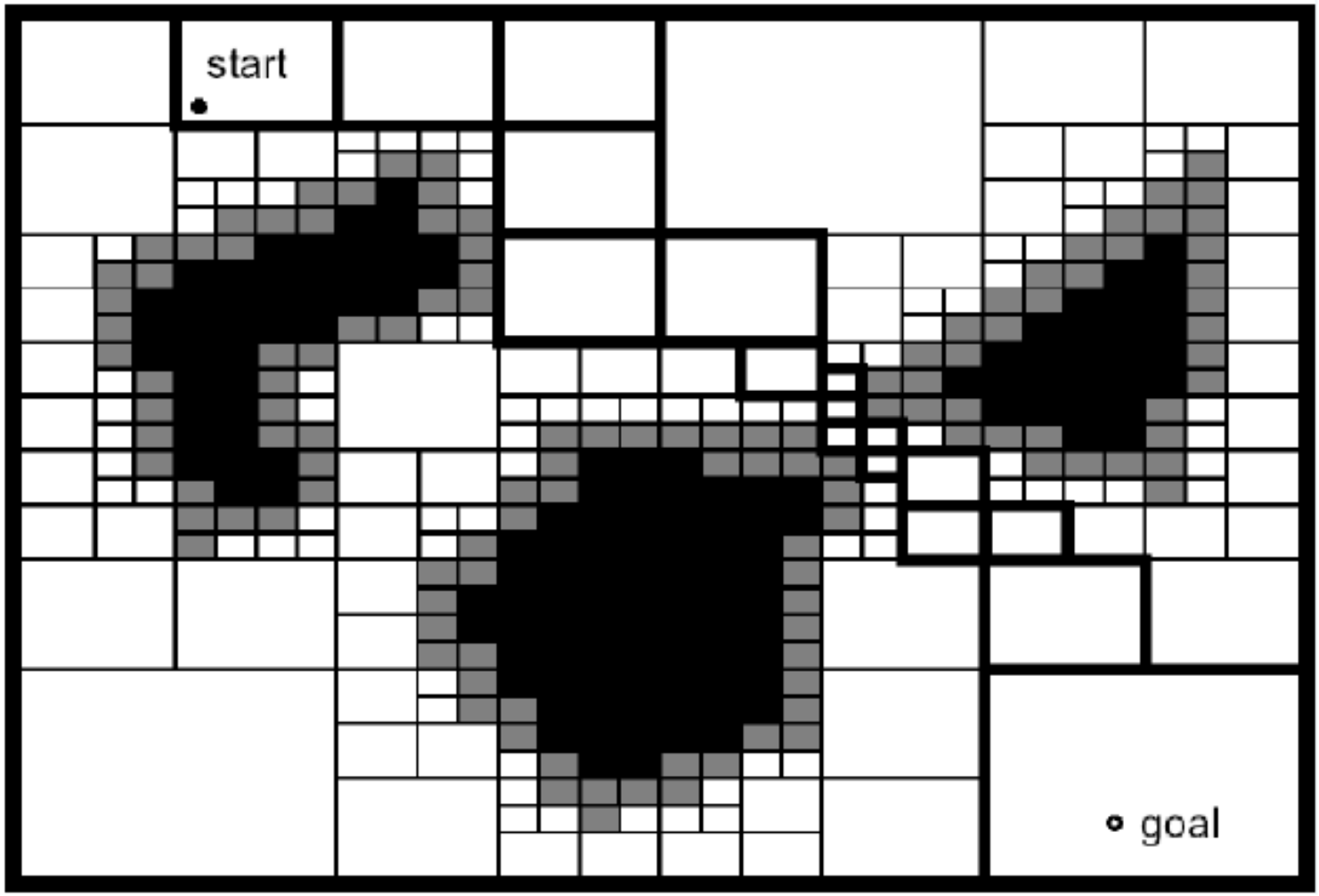
Descomposición en celdas aproximadas



- Se encuentra un camino sobre las celdas libres que llevan al objetivo
- Algoritmos: Dijkstra, A*, etc.

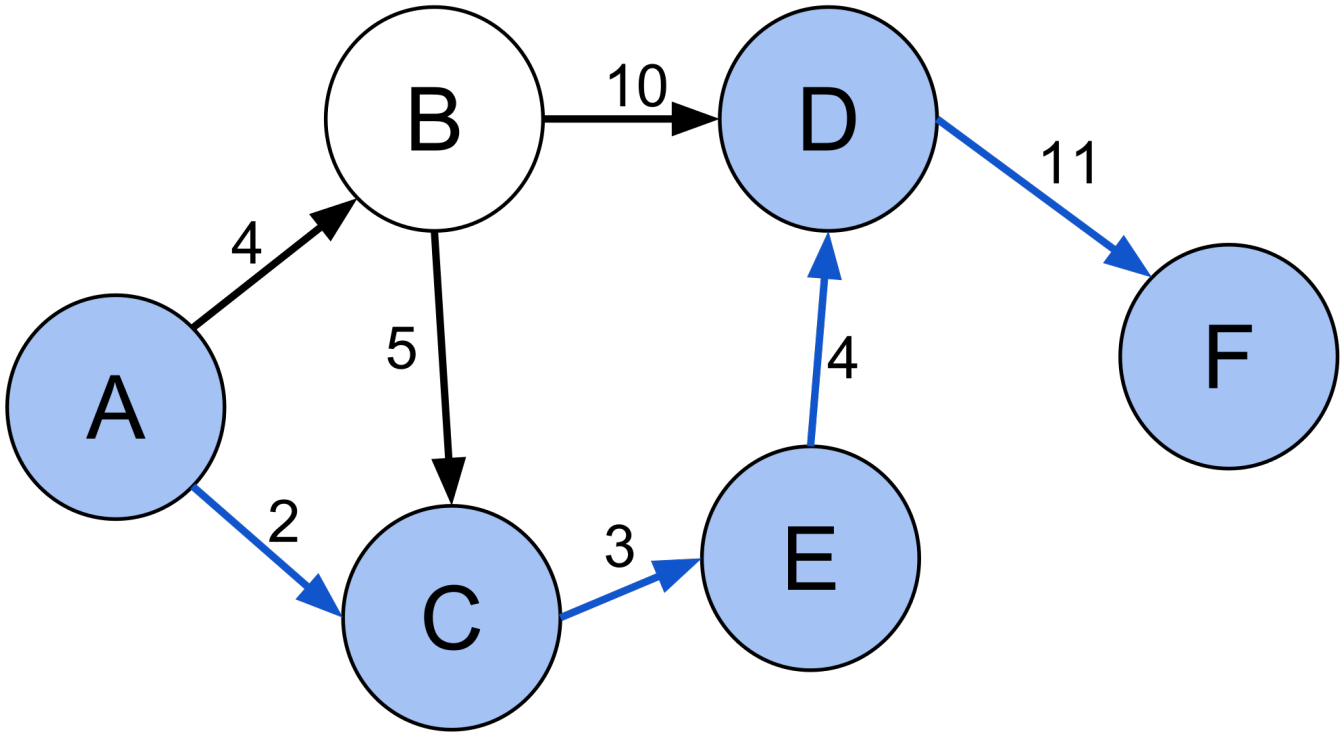
¿Qué problema tiene esta descomposición?

Descomposición en celdas de forma adaptativa



¿Cómo encontrar el camino más corto?

- El problema a resolver (búsqueda de camino mínimo) es ampliamente estudiado en la Teoría de Grafos
- Planificación sobre grillas = búsqueda camino mínimo en grafo



Camino mínimo y planificación de caminos

Sea un grafo $G = (V, E)$, queremos un camino $(v_1 \rightarrow v_2, \ldots, v_{n-1} \rightarrow v_n)$ tal que la suma de todas las distancias desde una celda c_i a una c_{i+1} sea mínima.
Bajo el problema de planificación tenemos una grilla con celdas c_i que representan las configuraciones posibles. Podemos hacer la siguiente equivalencia:

- $c_i \leftrightarrow v_i$
- Por cada c_j alcanzable desde una celda c_i, tendremos una arista $e_{i,j}$ $\rightarrow$ concepto de "vecindad"
- Las aristas tendran asociadas un peso $f(c_i, c_j)$ igual a la distancia entre c_i y c_j.
- Según el caso, puedo utilizar distintas definiciones de distancia: norma Euclídea, Manhattan, Diagonal, Chebyshov, etc.

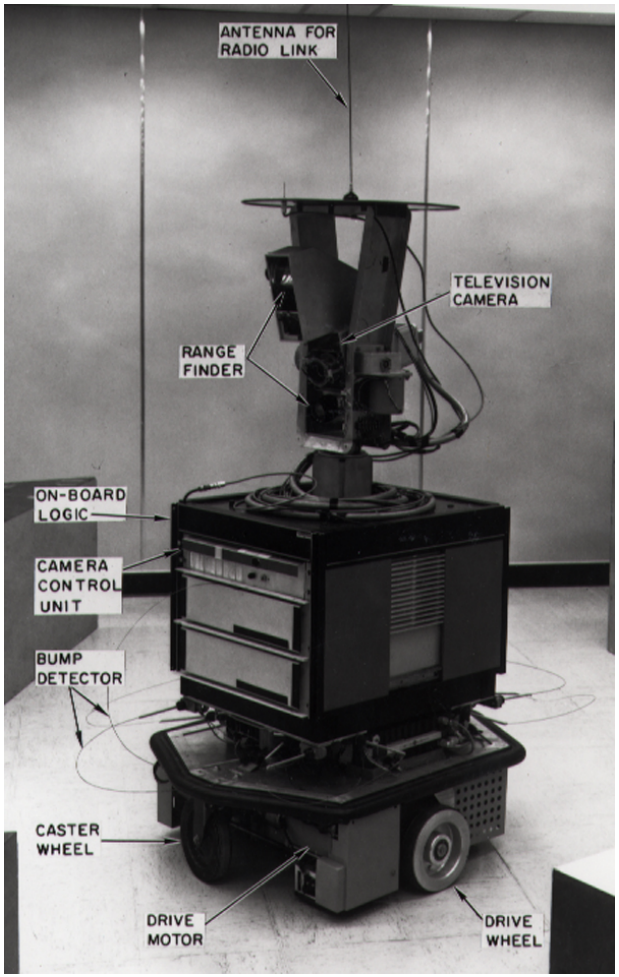
Algoritmo de Dijkstra

Busca en forma exhaustiva el camíno mínimo entre un c_i inicial y un c_f final (objetivo).

Algorithm 1 Dijkstra(*Graph*, *start*)

```
1: create vertex set OPEN
2: for each vertex v in Graph do
3:     g[v] ← INFINITY
4:     prev[v] ← UNDEFINED
5: end for
6: add start to OPEN
7: g[start] ← 0
8: while OPEN is not empty do
9:     current ← vertex in OPEN with min g[]
10:    remove current from OPEN
11:    for each neighbor of current do
12:        add neighbor to OPEN
13:        cost = g[current] + movement_cost(current, neighbor)
14:        if cost < g[neighbor] then
15:            g[neighbor] ← cost
16:            prev[neighbor] = current
17:        end if
18:    end for
19: end while
20: return g[], prev[]
```

Algoritmo A*



- Algoritmo para busqueda de caminos en grafos
- Creado en 1968 por Nils Nilsson para mejorar el path planning de "Shakey the Robot" (un prototipo que navegaba en una habitación con obstaculos)
- Basado en el algoritmo de Dijkstra.
- Hace uso de una heurística para estimar la distancia al goal, es específica del problema a solucionar
- Comienza por los nodos (vértices) que *parecen* llevar más rápido a una solución
- Logra un buen balance entre performance y calidad.

Algoritmo A*: idea

Hacer que Dijkstra sea más eficiente dirigiendo la búsqueda hacia celdas que parezcan ser óptimas. Para ello usamos una heurística de distancia al objetivo $h()$ que le sumamos a la distancia $g()$ que ya calculamos.

$$f(c_i) = g(c_i) + h(c_i)$$

Algoritmo A*

Algorithm 2 Algoritmo A*(*Graph*, *start*, *goal*)

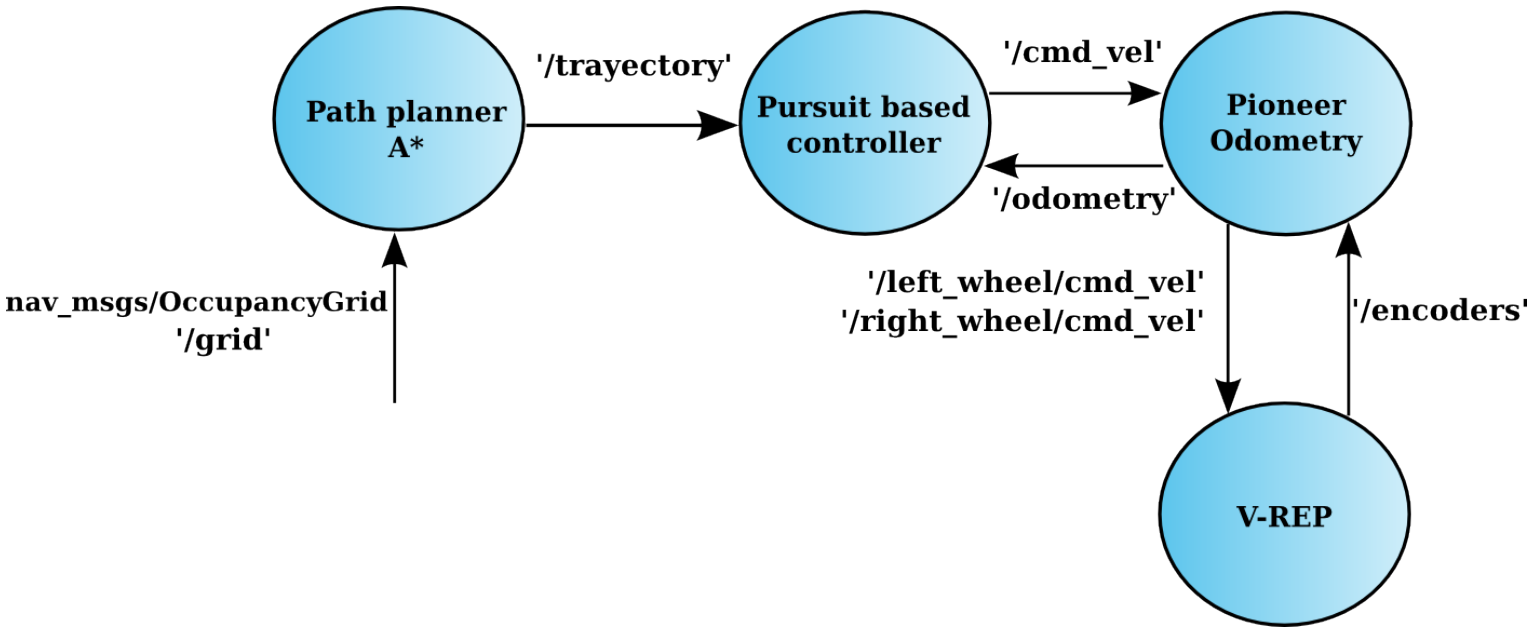
```
1: create vertex set OPEN
2: create vertex set CLOSED
3: add start to OPEN
4: for each vertex v in Graph do
5:    g[v] ← INFINITY
6:    f[v] ← INFINITY
7: end for
8: g[start] ← 0
9: f[start] ← h(goal)
10: while OPEN is not empty do
11:    current ← vertex in OPEN with min f[]
12:    if current = goal then
13:       return  reconstruct_path(prev, current)
14:    end if
15:    remove current from OPEN
16:    add current to CLOSED
17:    for each neighbor of current do
18:       if neighbor in CLOSED then
19:          continue
20:       end if
21:       cost = g[current] + movement_cost(current, neighbor)
22:       if neighbor not in OPEN then
23:          add neighbor to OPEN
24:       else if cost >= g[neighbor] then
25:          continue {Este no es un mejor camino}
26:       end if
27:       prev[neighbor] ← current
28:       g[neighbor] ← cost
29:       f[neighbor] = g[neighbor] + h(neighbor)
30:    end for
31: end while
```

Algoritmo A*: más sobre la heurística

La heurística definida en $h()$ controla el comportamiento del algoritmo

- En un extremo, si $h(c_i)$ es 0, solo $g(c_i)$ juega un papel en $f(c_i)$, entonces A* se vuelve el algoritmo de Dijkstra, que garantiza (por demostración) encontrar el camino mínimo.
- Si $h(c_i)$ es siempre menor o igual al costo de llegar desde c_i al *goal*, A* garantiza encontrar el camino mínimo. Mientras menor sea $h(c_i)$, más nodos expande A*, haciendolo más lento.
- Si $h(c_i)$ es igual al costo de llegar desde c_i al *goal*, entonces A* va a seguir el mejor camino y no va a expandir ningún nodo adicional, haciendolo un algoritmo óptimo.
- Si $h(c_i)$ supera el costo de llegar desde c_i al *goal*, A* no garantiza encontrar el camino mínimo (por eso decimos que se basa en una heurística), pero de todas formas mejora en performance a Dijkstra.

Para el taller: ecosistema ROS



- Hipótesis: conocemos la grilla de ocupación (mapa con obstáculos)
- Debemos planear como llegar al goal.
- El algoritmo A* nos da una solución aplicable a problemas en 2D.
- Queremos hallar una trayectoria para que el robot la siga (de alguna de las formas vistas en la materia)

Para el taller: celdas de ocupación (occupancy grid)

nav_msgs/OccupancyGrid

std_msgs/Header header
nav_msgs/MapMetaData info
int8[] data

- info: Metadata del mapa
- data: La información del mapa, ordenado incrementalmente por filas y empezando en (0,0). La probabilidad de que una celda esté ocupada está en el rango [0,100]. Si se desconoce es -1.

Para el taller: celdas de ocupación (occupancy grid)

nav_msgs/MapMetaData

time map_load_time
float32 resolution
uint32 width
uint32 height
geometry_msgs/Pose origin

- resolution: Resolución del mapa [m/cell]
- width: Ancho del mapa [cells]
- height: Alto del mapa [cells]
- origin: El origen del mapa [m, m, rad], La pose real de la celda (0,0) en el mapa

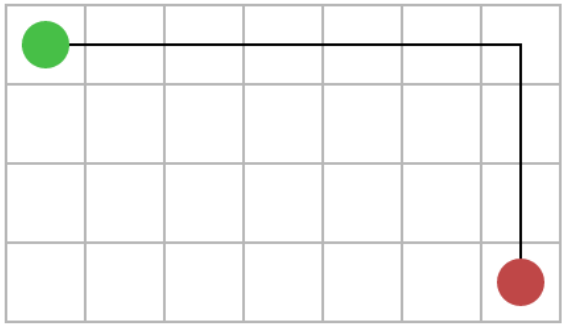
Para el taller: analizando el problema

- ¿Que resolución de mapa usamos? ¿Baja o alta?
- Muchas resolución mejora la calidad de la solución pero empeora el tiempo de cómputo.
- Poca resolución optimiza la ejecución pero se pueden perder soluciones posibles.
- ¿Que movimientos permitimos?¿Cuánto cuesta cada movimiento?
- Mas celdas conectadas nos mejora el camino obtenido.
- De nuevo tenemos más ejes para calcular, ergo más tiempo de cómputo.
- Podemos asumir que todos los movimientos cuestan los mismo.

Para el taller: ¿qué heurística usamos?

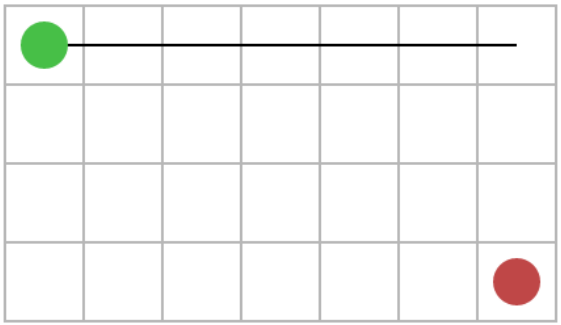
Si sólo permitimos grillas con movimientos en 4 direcciones usaremos la distancia de Manhattan (o norma 1 para los amigos)

$$h(n) = |n.x - goal.x| + |n.y - goal.y|$$



En cambio, si permitimos movimientos en 8 direcciones (diagonales) lo mejor será usar la distancia diagonal (o norma infinito)

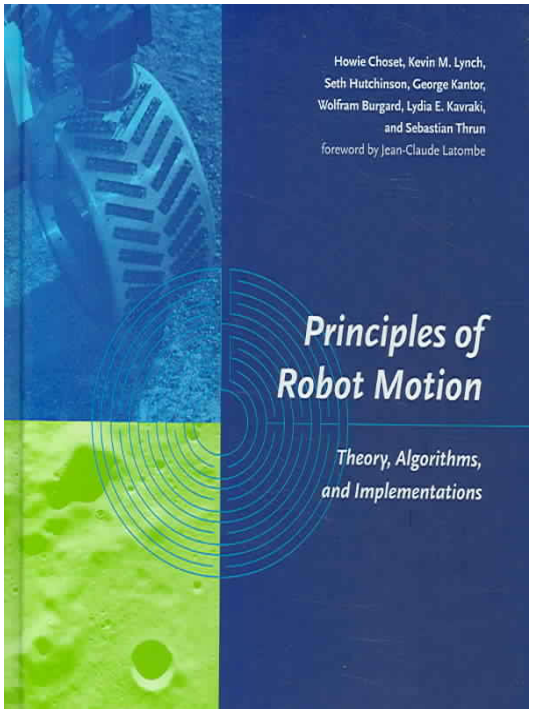
$$h(n) = max(|n.x - goal.x|, |n.y - goal.y|)$$



Para el taller: ¿dónde quedo mi espacio continuo?

- Debemos lograr un correlato entre las celdas de la grilla y las posiciones en nuestro espacio de configuraciones (un plano en este caso).
- Vamos a tener que des-discretizar el camino. Una opción es usar el centro dado por la celda como posición real a devolver como solución.
- Podemos armar el camino usando el criterio anterior, generando una lista de puntos de todas las celdas atravezadas por A*
- ¿Cómo se modelan las orientaciones de la trayectoria?

Más sobre Planificación de caminos y A*



"Principles of robot motion: theory, algorithms, and implementation", Howie Choset, Kevin Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia Kavraki, Sebastian Thrun, MIT press, 2005

http://theory.stanford.edu/~amitp/GameProgramming/
https://www.redblobgames.com/pathfinding/a-star/introduction.html
https://www.redblobgames.com/pathfinding/a-star/implementation.html