Robots móviles

Tema 6. Navegación y planificación de trayectorias

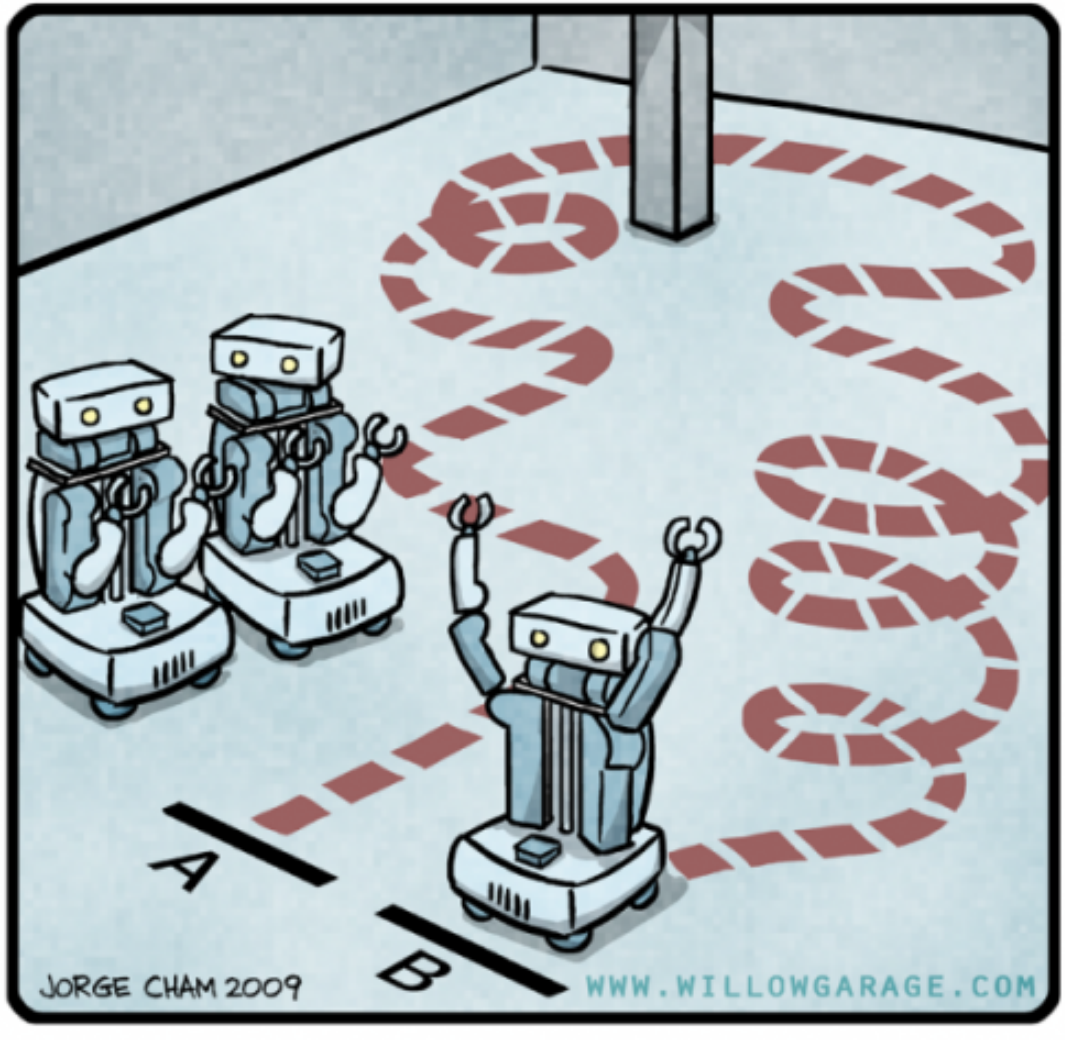
Navegación

Introducción: ¿qué es navegar?

Navegación

Conjunto de técnicas y algoritmos necesarios para que un robot móvil pueda llegar hasta su destino por el **camino más corto** posible y **sin chocar con los obstáculos**

R.O.B.O.T. Comics



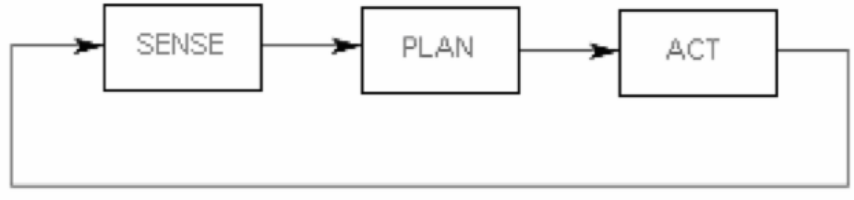
"HIS PATH-PLANNING MAY BE SUB-OPTIMAL, BUT IT'S GOT FLAIR."

Global vs local

- **Navegación global**: encontrar el camino óptimo (más corto o más adecuado) Necesitaremos:
  - Un **mapa**
  - Un algoritmo para el **cálculo del camino** óptimo según el mapa
- **Navegación local**: no chocar con obstáculos no reflejados en el mapa. Necesitaremos:
  - Información de los **sensores de rango**
  - Un algoritmo de **evitación de obstáculos**, que calcule la mejor dirección de movimiento para evitarlos (sin alejarnos demasiado del camino óptimo)

Robótica clásica (años 70)

Paradigma *deliberativo* o *jerárquico* : énfasis sobre todo en la **planificación global**

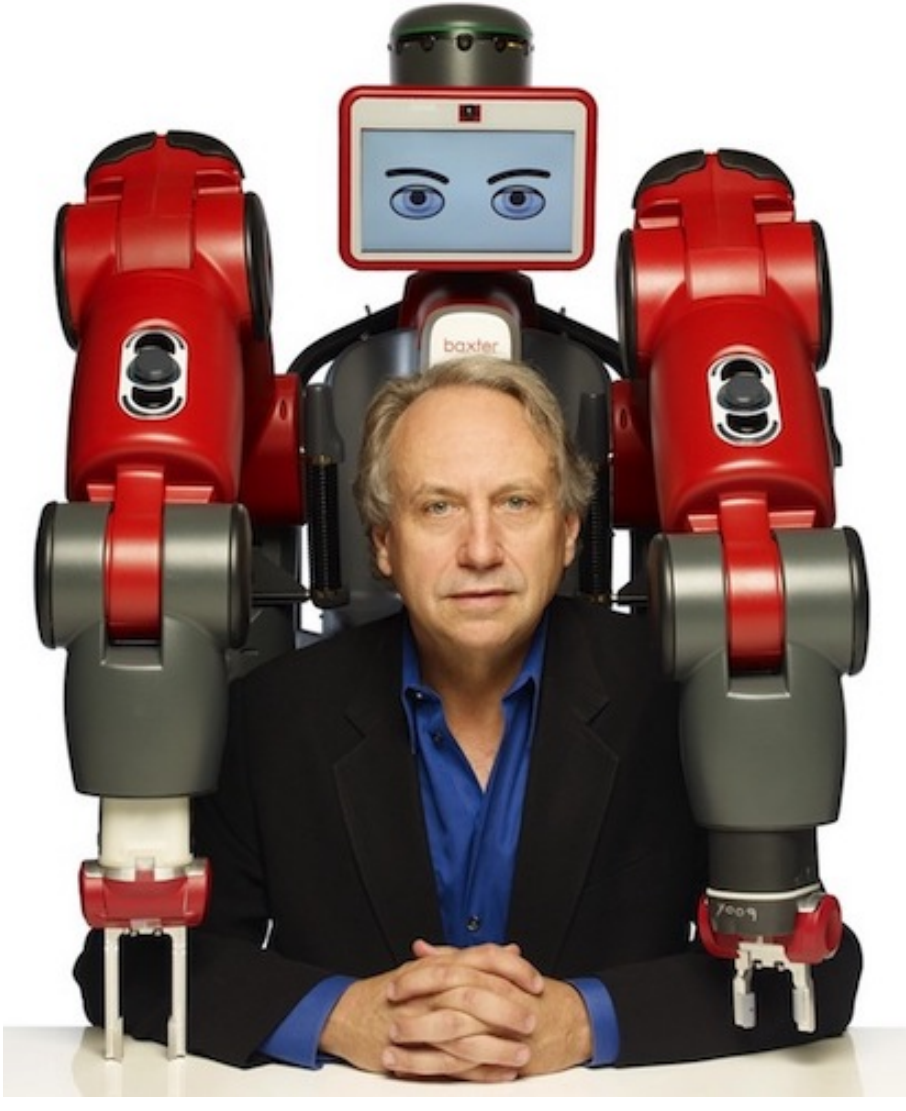


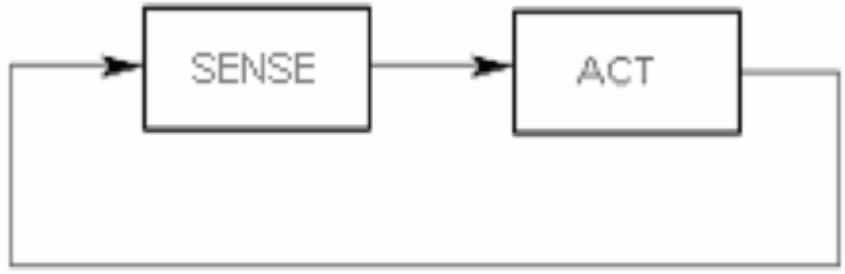
Izquierda: El robot *shakey* (1966-1972), primer robot móvil capaz de planificar sus tareas. Derecha: el ciclo de procesamiento típico del paradigma jerárquico. La información de los sensores se usa sobre todo para elaborar un **plan**

Robótica reactiva (años 80)

"Reacción" a la robótica clásica. **Énfasis en los sensores**, se elimina la planificación global.

Izquierda: Rodney Brooks, uno de los "popes" de la robótica reactiva. A la derecha, el bucle de control clásico en este paradigma: nótese que no hay planificación


baxter

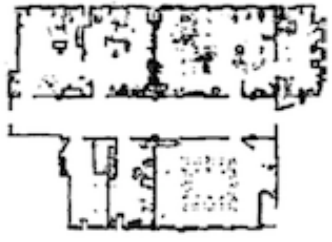

SENSE
ACT

Robótica en la actualidad

Paradigma híbrido: se combina la planificación global (proceso de "baja frecuencia") con la local (se ejecuta continuamente)

Tomada del curso Introduction to Mobile Robotics" de Wolfram Burgard

Classic Two-layered Architecture



map

sensor data

Planning

low frequency

sub-goal

Collision Avoidance

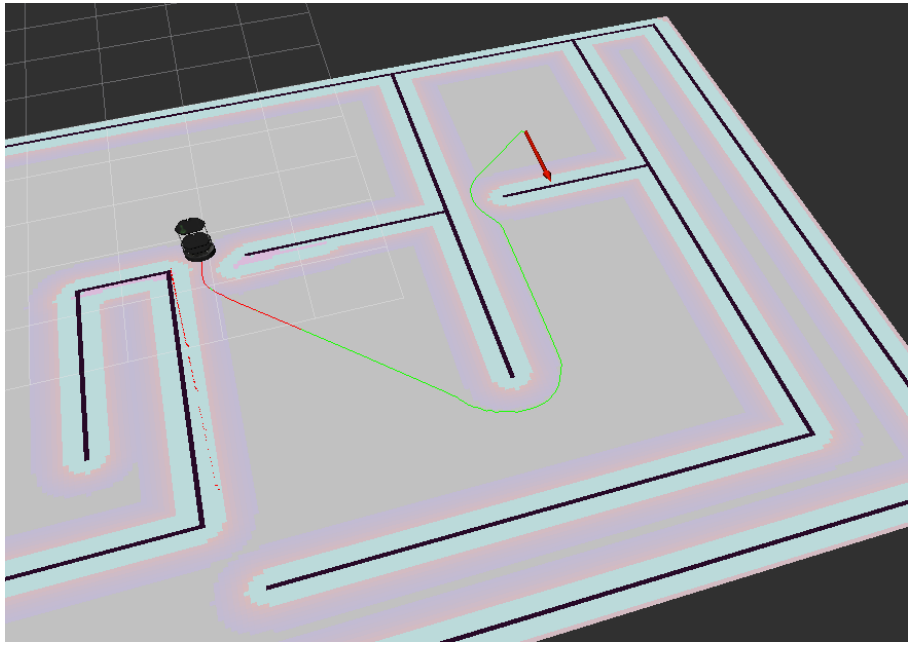
high frequency

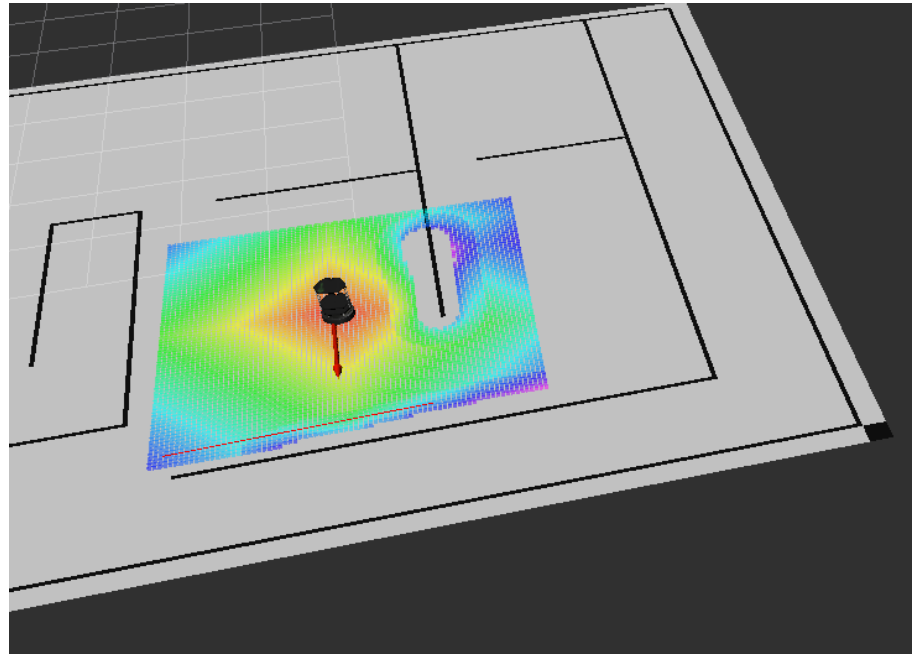
motion command

robot

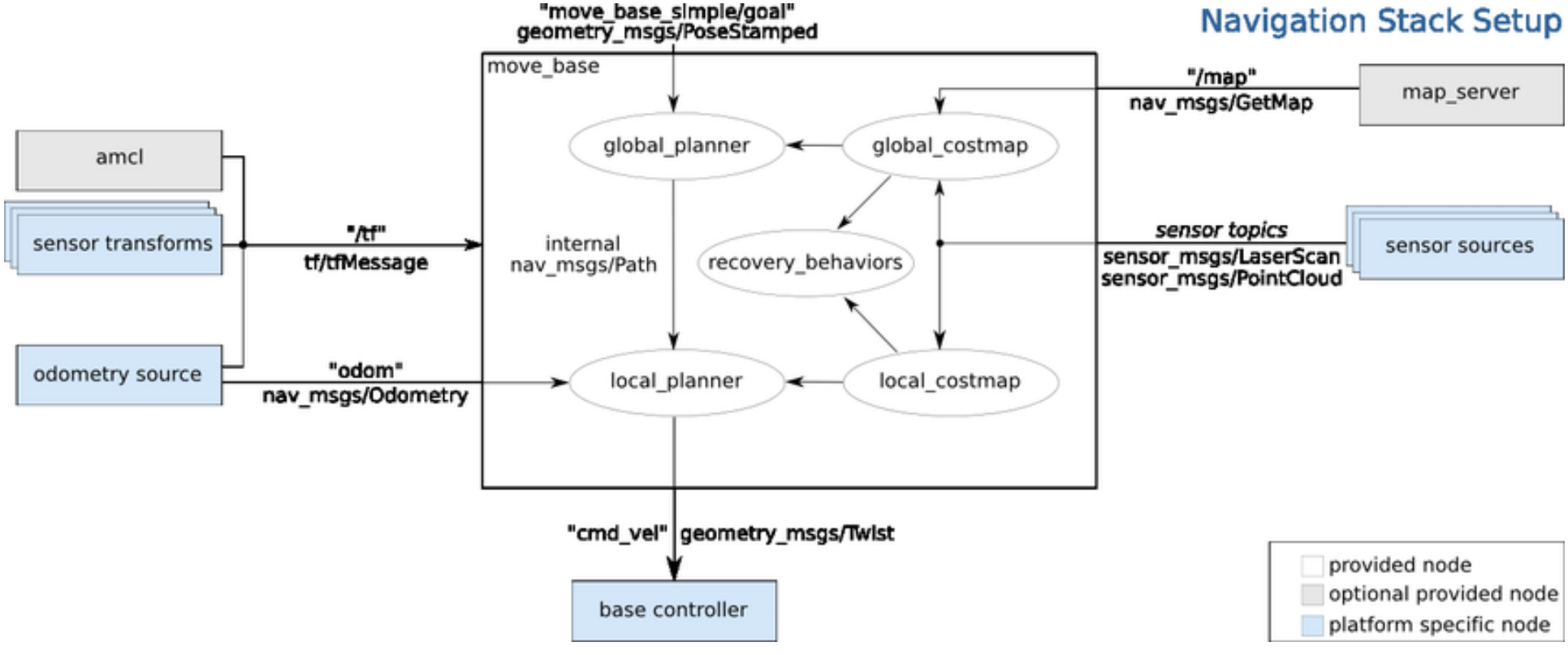
ROS: navegación global (izq.) y local (der.)

Se muestra el *costmap* global y el cálculo del camino, y el *costmap* local





El *stack* de navegación en ROS



Planificación global

Cálculo del camino más "corto"

La gran mayoría de algoritmos de cálculo de rutas trabajan en el **espacio de configuraciones** o CSpace: el espacio formado por las posibles poses del robot

- Tantas dimensiones como grados de libertad tenga el robot
- Buscamos una ruta en este espacio que solo pase por espacio libre

Es un concepto habitual en planificación de movimiento de brazos robot (Demo)

CSpace para robots móviles

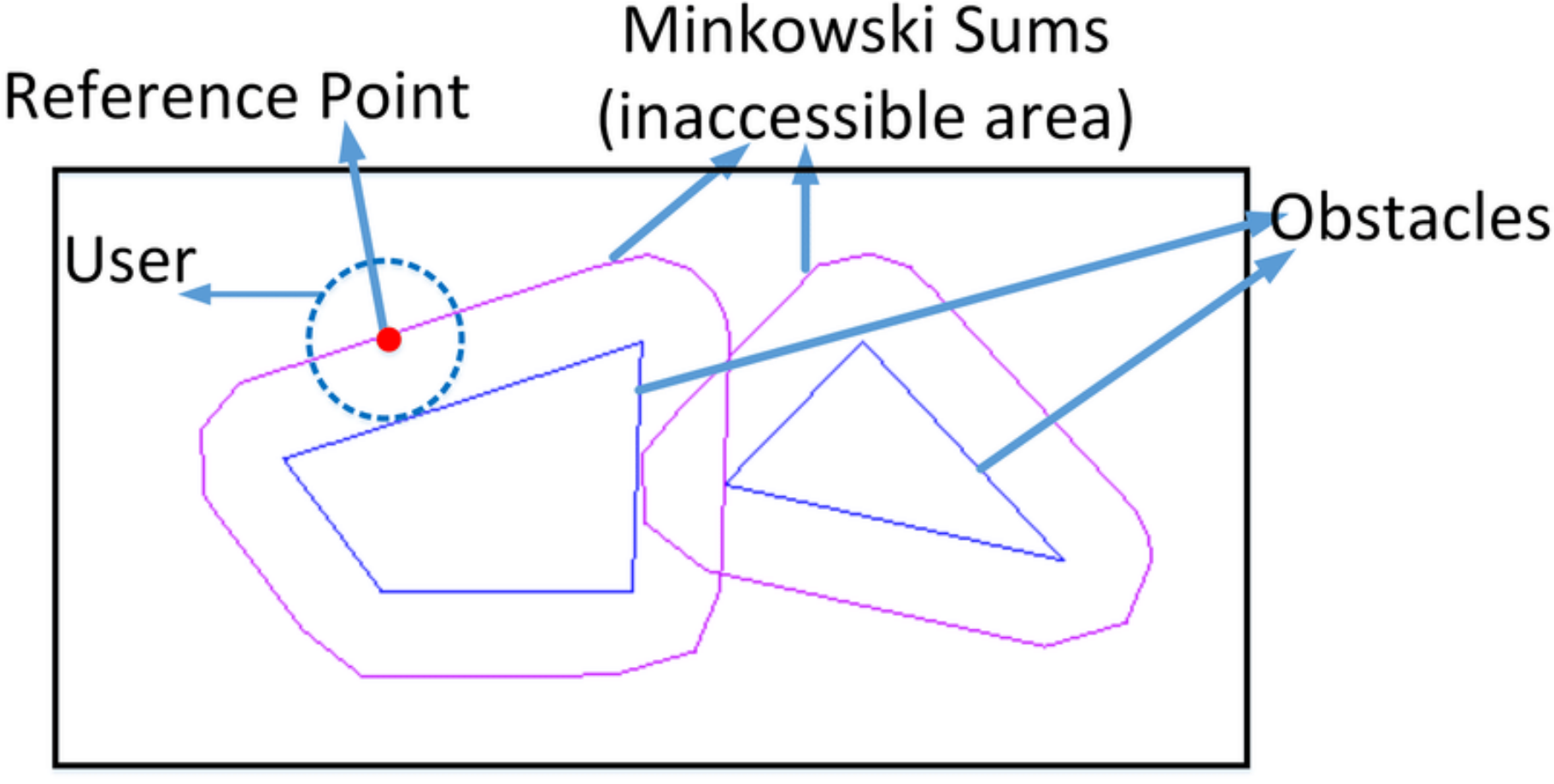
Típicamente la *pose* se define con (x, y, θ), lo que daría un CSpace 3D.

No obstante, vamos a hacer simplificaciones:

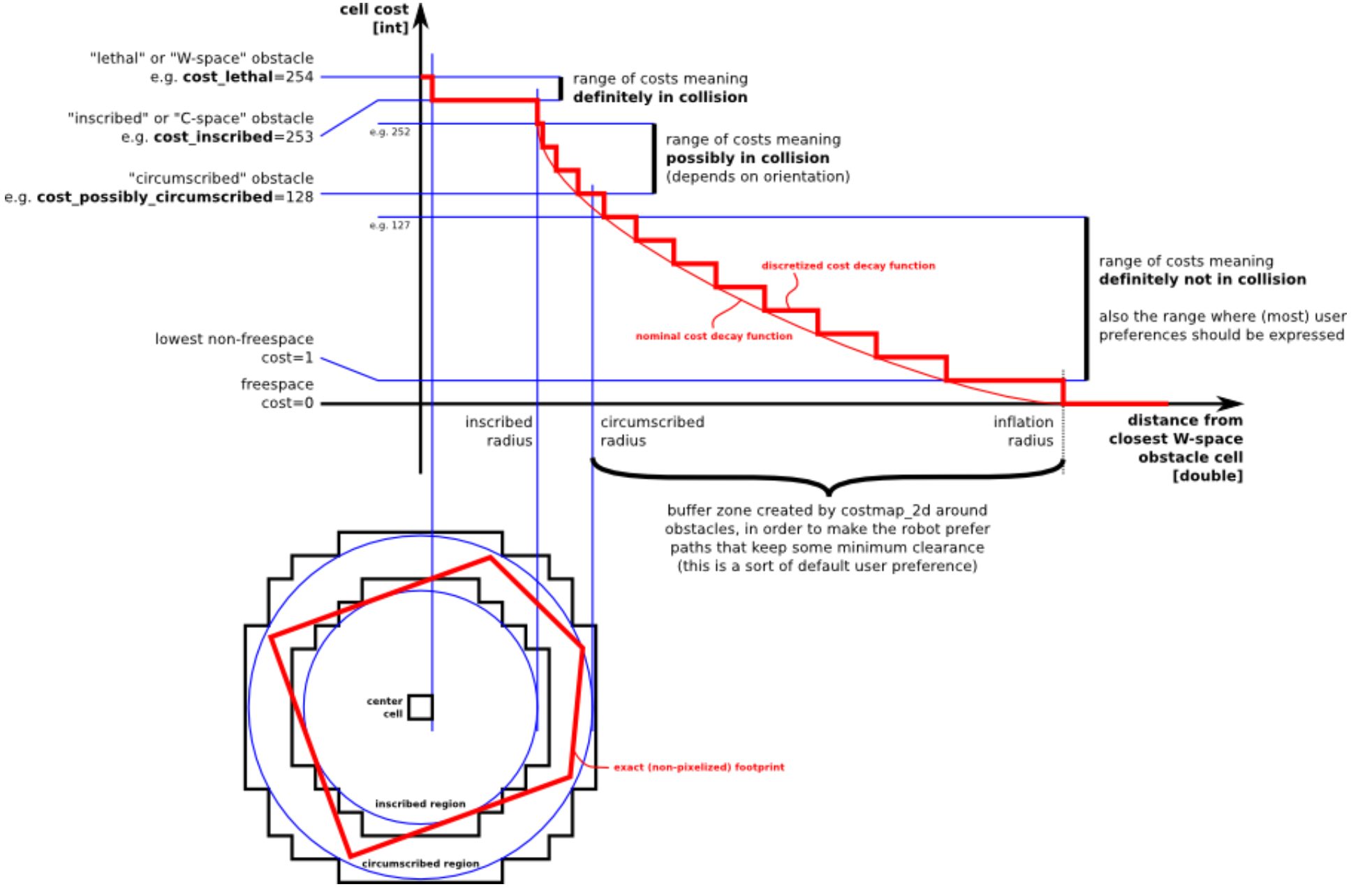
- **Ignoraremos la θ** (el CSpace se queda en 2D). Esto podemos hacerlo si el robot es *holonómico*, es decir puede seguir cualquier trayectoria en el CSpace.
- Podemos suponer que **el robot es un punto**, si **"dilatamos" los obstáculos** al menos en un tamaño igual al radio del robot

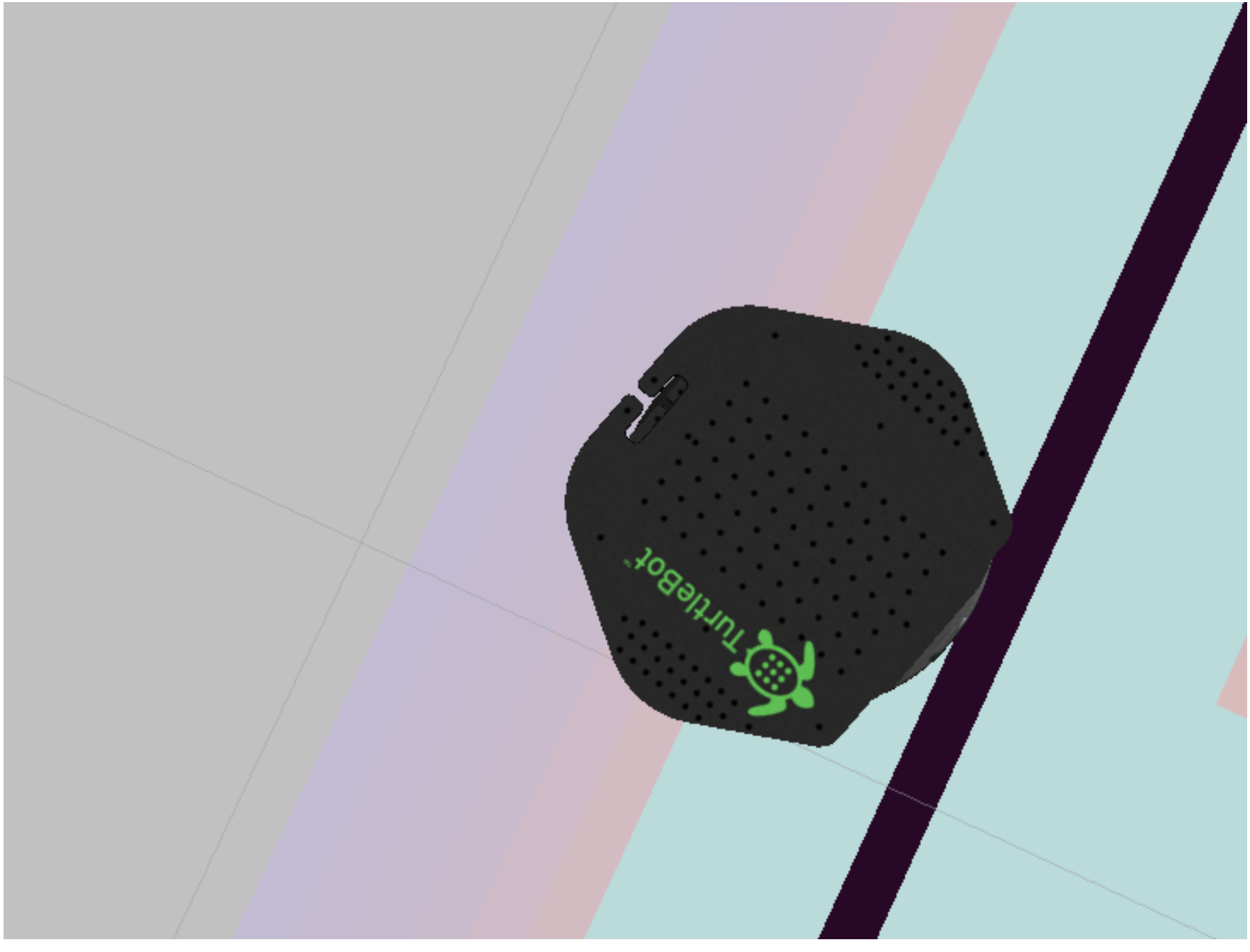
La operación de dilatación que necesitamos ha sido formalizada en diferentes campos de las matemáticas:

- *Suma de Minkowski* de la forma del robot y los obstáculos
- En *morfología matemática* la operación se denomina también *dilatación*


Minkowski Sums
(inaccessible area)
Reference Point
User
Obstacles

En ROS se aplica la misma idea, generando un *costmap* llamado *inflation costmap*


cell cost
[int]
"lethal" or "W-space" obstacle
e.g. cost_lethal=254
range of costs meaning
definitely in collision
"inscribed" or "C-space" obstacle
e.g. cost_inscribed=253
e.g. 252
range of costs meaning
possibly in collision
(depends on orientation)
"circumscribed" obstacle
e.g. cost_possibly_circumscribed=128
e.g. 127
discretized cost decay function
nominal cost decay function
range of costs meaning
definitely not in collision
also the range where (most) user
preferences should be expressed
lowest non-freespace
cost=1
freespace
cost=0
inscribed
radius
circumscribed
radius
inflation
radius
distance from
closest W-space
obstacle cell
[double]
buffer zone created by costmap_2d around
obstacles, in order to make the robot prefer
paths that keep some minimum clearance
(this is a sort of default user preference)
center
cell
exact (non-pixelized) footprint
inscribed region
circumscribed region


TurtleBot

Búsqueda del camino más corto

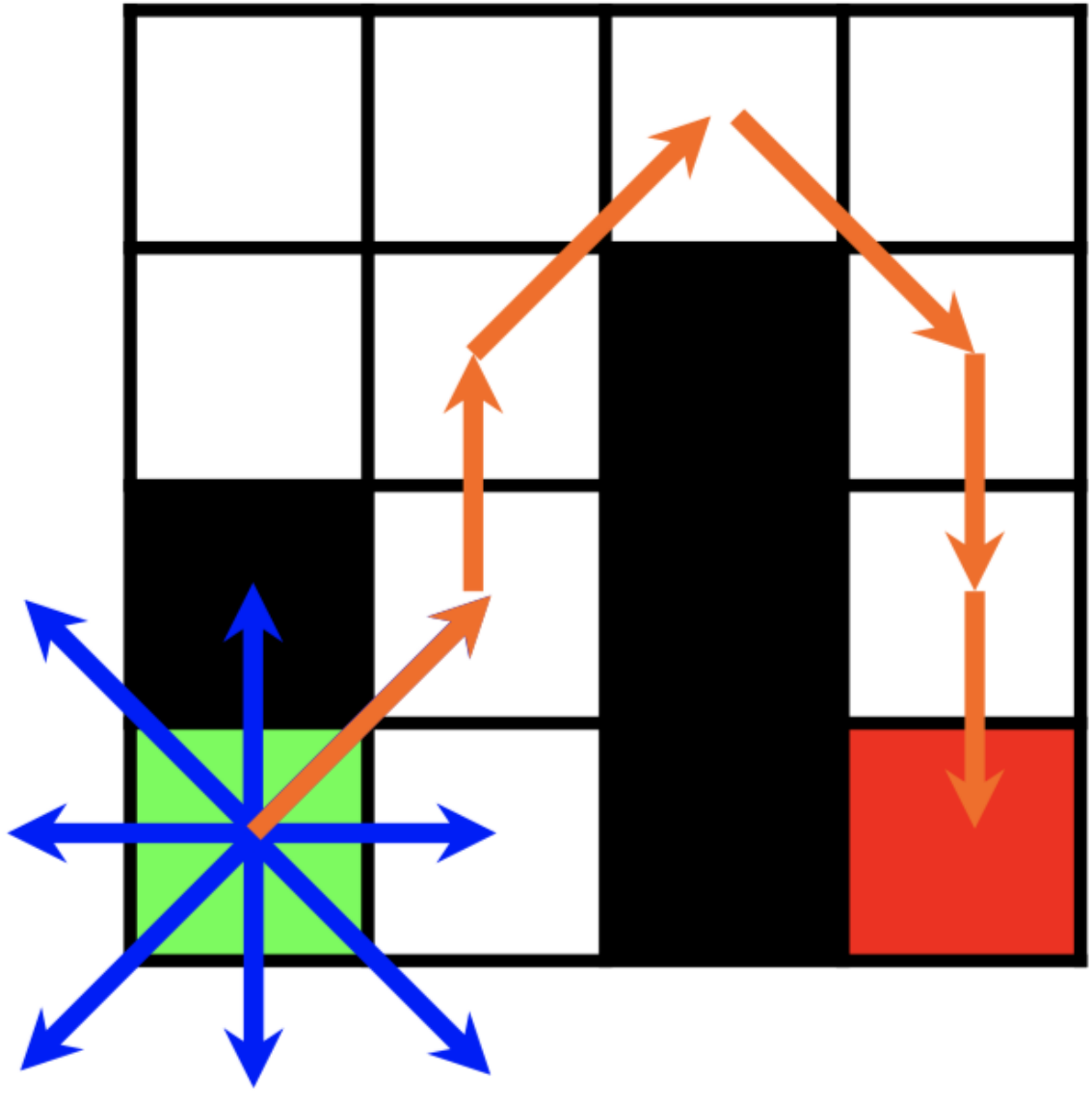
Hay infinitos caminos posibles entre un origen y un destino, pero el espacio de búsqueda no puede ser infinito, hay que *restringirlo*.
Normalmente:

1. Transformar el CSpace en un grafo que contenga todos los caminos a considerar.
2. Aplicar algún algoritmo de búsqueda de camino más corto en grafos.

Conversión del CSpace en un grafo

Dependerá de la representación del mapa:

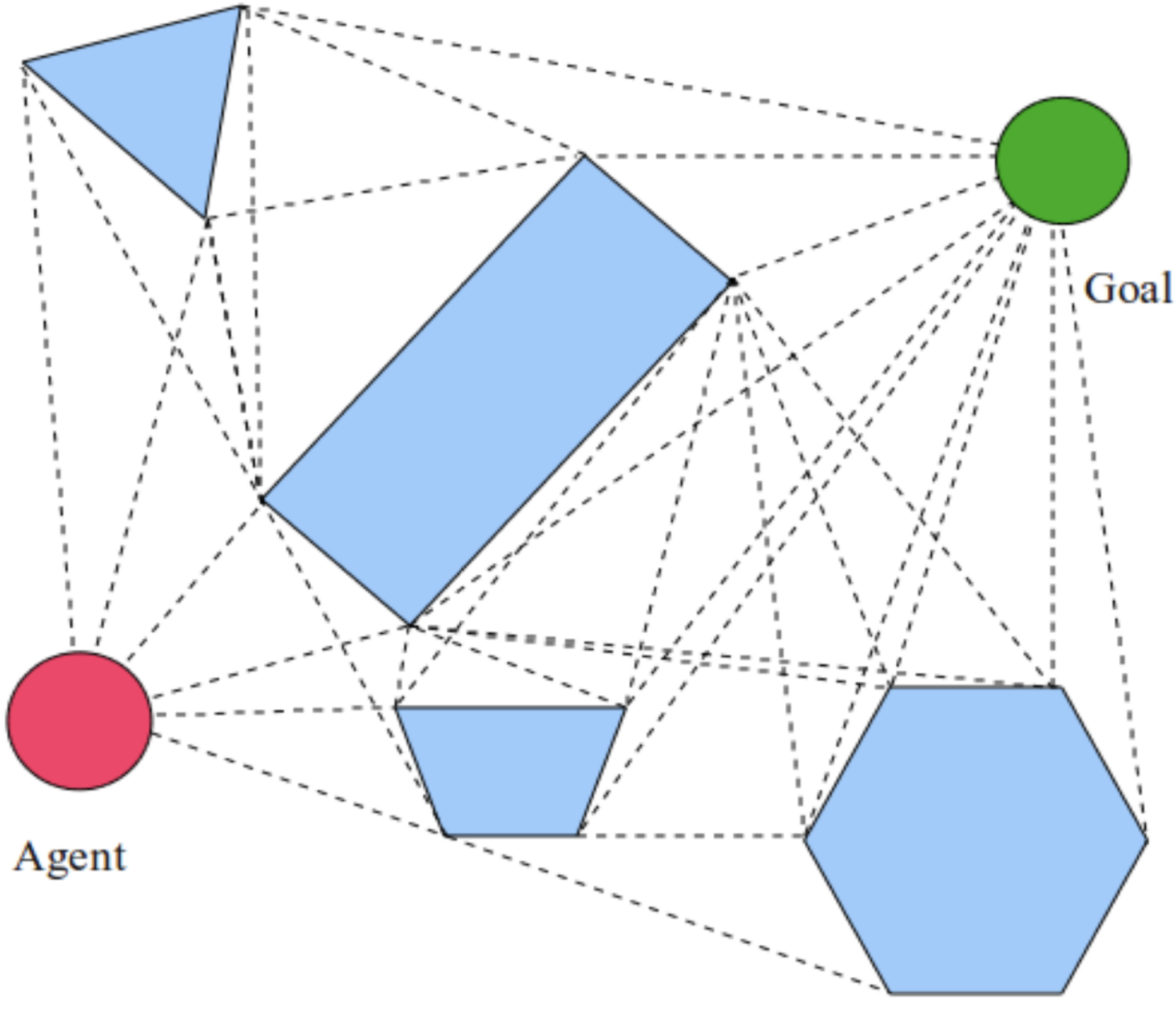
- *Mapas poligonales*: varias formas:
  - Grafo de visibilidad
  - Grafo de voronoi
- *Rejillas de ocupación*: cada celda será un nodo, conectado con sus 8 vecinos más inmediatos.



Grafo de visibilidad

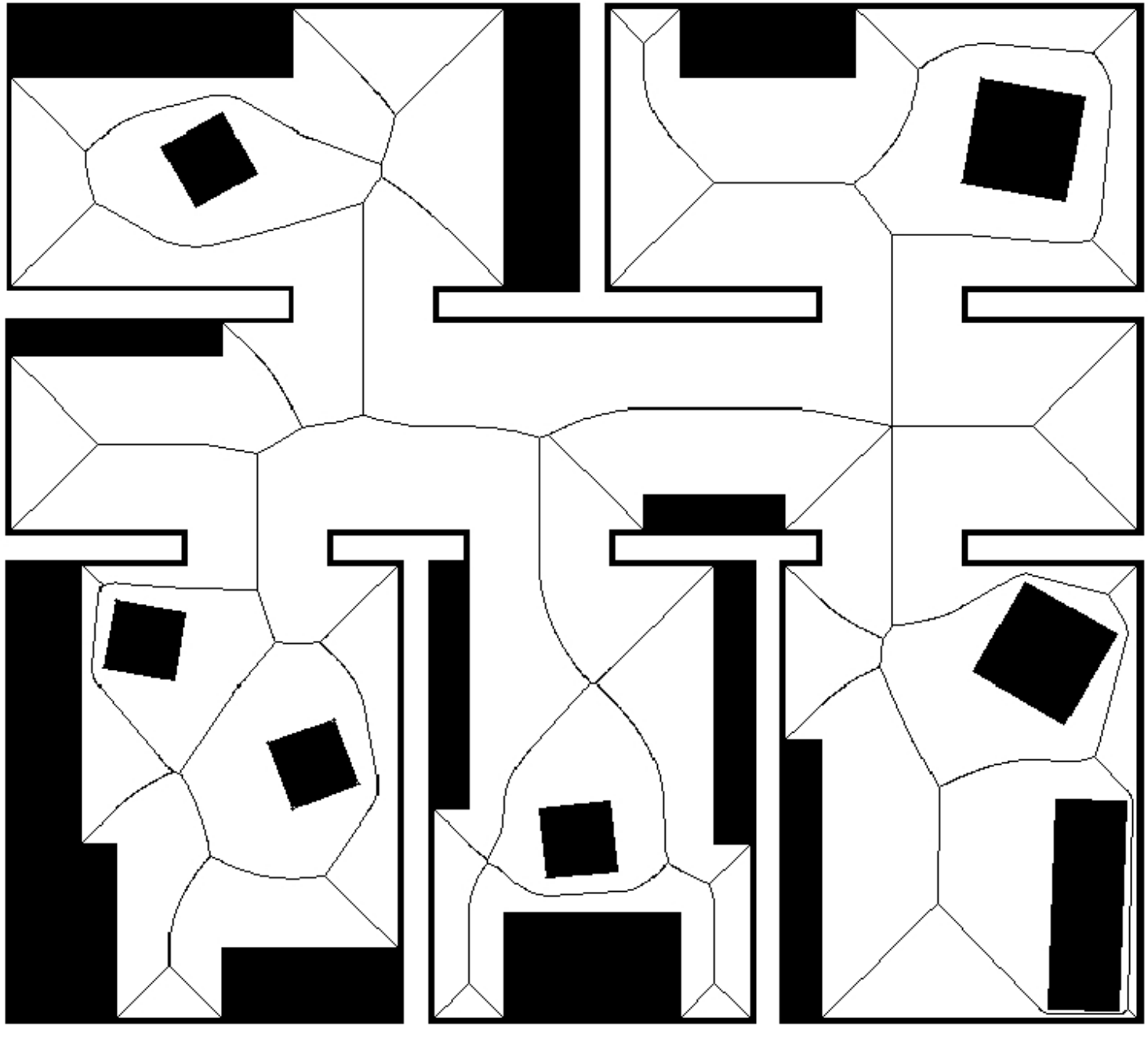
Grafo cuyos nodos son los vértices de los polígonos, y los arcos las conexiones entre ellos que no intersectan ningún obstáculo.

Tomado de https://www.slideshare.net/GauravGupta527/visibility-graphs


Goal
Agent

Diagramas de Voronoi

Formado maximizando la distancia mínima a los obstáculos (todos los puntos con la misma distancia mínima a dos o más objetos)



Algoritmos de búsqueda de camino más corto

- Algoritmos clásicos de búsqueda en grafos: el más típico es **Dijkstra**
- Guiados por heurísticas: el más usado es `A*`, aunque hay otros similares, como `D*`

A*

- La búsqueda se guía por un $f(n)$ para cada nodo: $f(n) = g(n) + h(n)$, siempre expandimos por el nodo de menor $f(n)$
  - $g(n)$: coste del camino ya recorrido
  - $h(n)$ una heurística *admisible* (== una estimación "optimista") para el camino que queda

Demo: http://qiao.github.io/PathFinding.js/visual/

Pseudocódigo A*

```
      // A*
1:    initialize the open list
2:    initialize the closed list
3:    put the starting node on the open list (you can leave its f at zero)
-
4:    while the open list is not empty
5:        find the node with the least f on the open list, call it "q"
6:        pop q off the open list
7:        generate q's 8 successors and set their parents to q
8:        for each successor
9:            if successor is the goal, stop the search
10:           successor.g = q.g + distance between successor and q
11:           successor.h = distance from goal to successor
12:           successor.f = successor.g + successor.h
-
13:           if a node with the same position as successor is in the OPEN list \
-                 which has a lower f than successor, skip this successor
14:           if a node with the same position as successor is in the CLOSED list \
-                 which has a lower f than successor, skip this successor
15:           otherwise, add the node to the open list
16:       end
17:       push q on the closed list
18:   end
```

Otros algoritmos: D* y D* lite

- Similares a A*, aunque parten del destino en lugar del origen
- Pueden *replanificar* trayectorias: pueden "reparar" la trayectoria de modo incremental si hay cambios en el grafo

- Los *rover* de Marte necesitan autonomía dado el retardo de la señal Tierra-Marte (entre 3-22 min)
- Usan un mapa de costes de rejilla y una versión modificada del algoritmo `D*` para calcular el camino más corto.

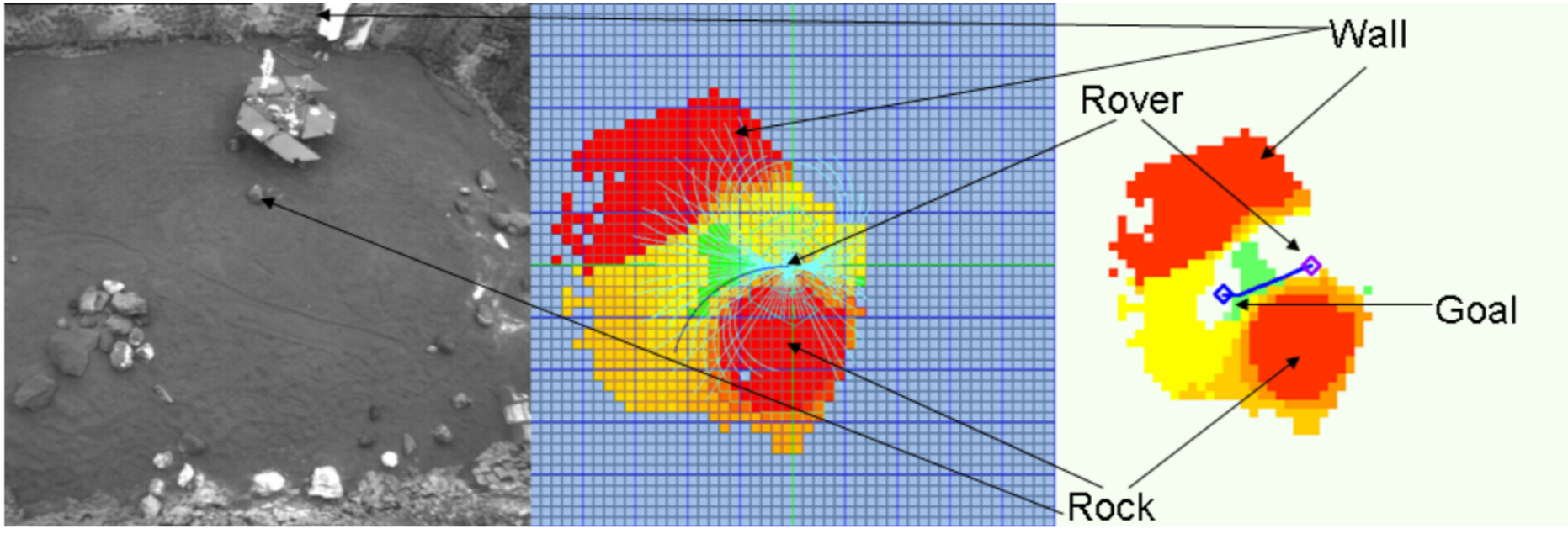


Figure 5. The left image is an overhead view of the rover. The middle image is the corresponding goodness map, and the Field D* cost map is shown in the right image. Blue cells have unknown traversability. All other cells are colored based on a gradient between green (high goodness/low cost) and red (low goodness/high cost). Note that the entire goodness map is presented, but only a small portion of the cost map is shown in here.

Búsqueda del camino más corto en ROS

Paquete **`global_planner`**, implementa Dijkstra y A*, seleccionables cambiando el parámetro **`use_dijkstra`**