Path Planning

Dra. Mª Dolores Rodríguez Moreno

Objectives

Specific Objectives
• Role of path planning in robotics • Main representation • Main techniques
Source
• Maxim Likhachev. Planning Techniques for Robotics. CMU. http://www.cs.cmu.edu/~maxim/classes/robotplanning/ • P. Muñoz, B. Castaño and M.D. R-Moreno. 3Dana: A Path Planning Algorithm for Surface Robotics (2017). Int. Scientific Journal Engineering Applications of AI. Vol. 60, pp:175-192

Outline

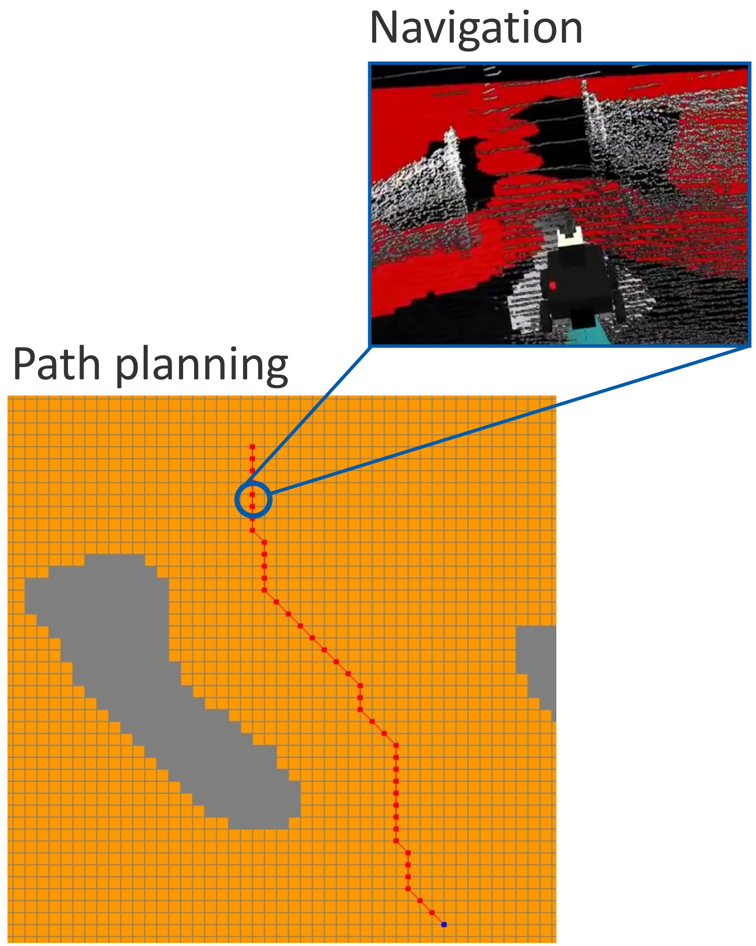
- **Introduction**
- Representation
- Dijkstra
- A*
- A*PS
- Theta*
- S-Theta*
- Heuristics
- Conclusions

Introduction (I)

- Path-planning problem aims to obtain feasible and optimal (or near to it) routes between two or more points
  - Find optimal (or near to it) paths is not trivial
  - Feasible implies not transverse over obstacles or overcome system limitations
  - Usually parameters: path length, run-time, expanded nodes and number of heading changes
- It is a fundamental task in mobile robots and video games

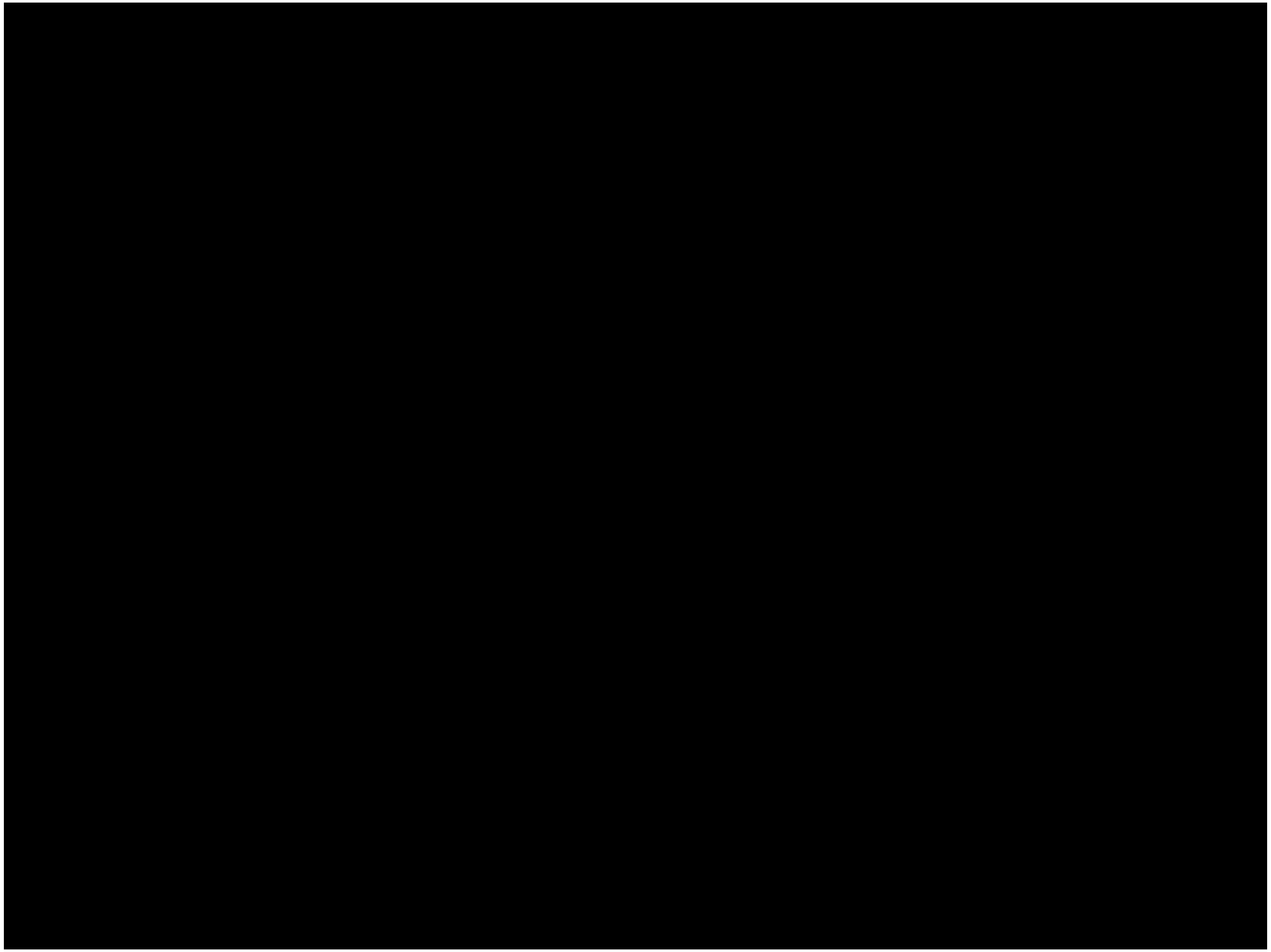
Introduction (II)

- Navigation
  - Local paths
  - Guided by sensors
- Path Planning
  - Global paths
  - Known terrain
  - Related to AI



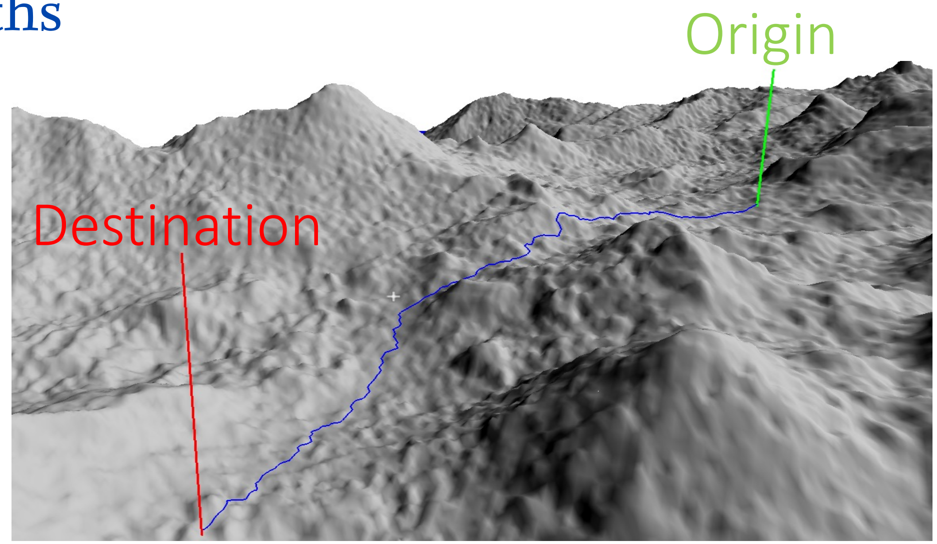
Introduction (III)

- First point to address: the environment
  - Local planning vs Long term planning
  - Totally observable vs Partially observable
  - Discrete vs Continuous
  - Dynamic environment? Maybe
  - Extra information on the terrain? Maybe



Introduction (IV)

- Long term path-planning is more related to AI
- Easier with fully observable environment
- High effort trying to obtain optimal paths
- Several possibilities on the map
- In some domains (such as planetary exploration) path-planning and task-planning are highly coupled



Outline

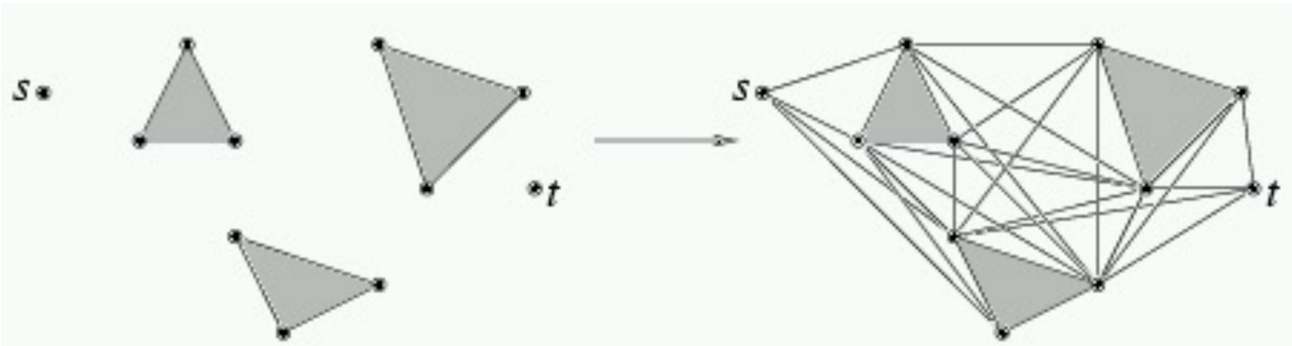
- Introduction
- **Representation**
- Dijkstra
- A*
- A*PS
- Theta*
- S-Theta*
- Heuristics
- Conclusions

Representation

- Skeletonization
  - Visibility graphs
  - Voronoi diagrams
  - …
- Cell decomposition
  - n-connected grids
  - …

Visibility graphs (I)

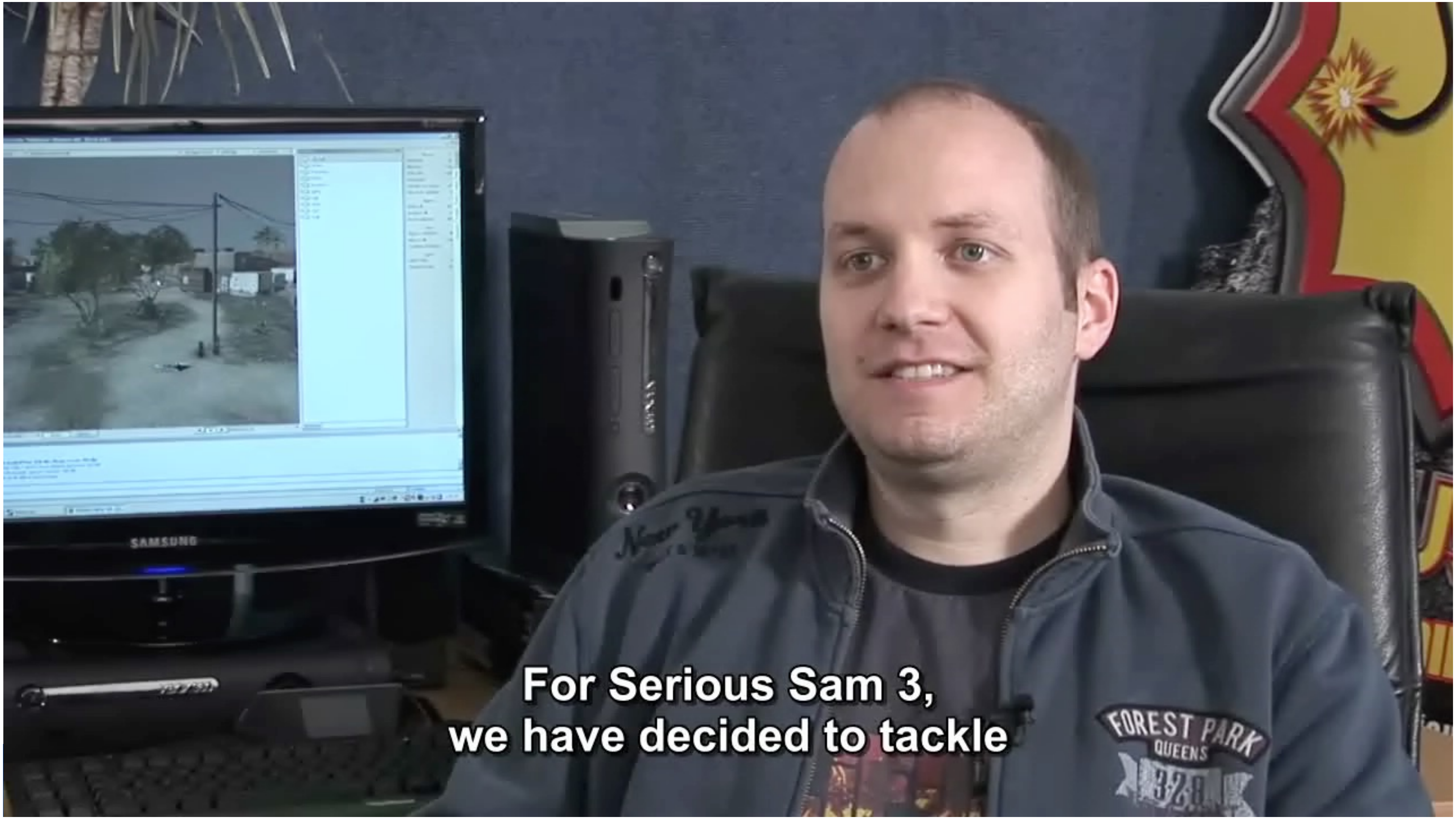
- Based on idea that the shortest path consists of obstacle-free straight line segments connecting all obstacle vertices and start and goal
- Construct a graph by connecting all vertices, start and goal by obstacle-free straight line segments



Visibility graphs (II)

- Independent of the size of the environment (+)

- Path is too close to obstacles (-)
- Hard to deal with the cost function that is not distance (-)
- Hard to deal with non-polygonal obstacles (-)
- Hard to maintain the polygonal representation of obstacles (-)
- Can be expensive in spaces higher than 2D (-)

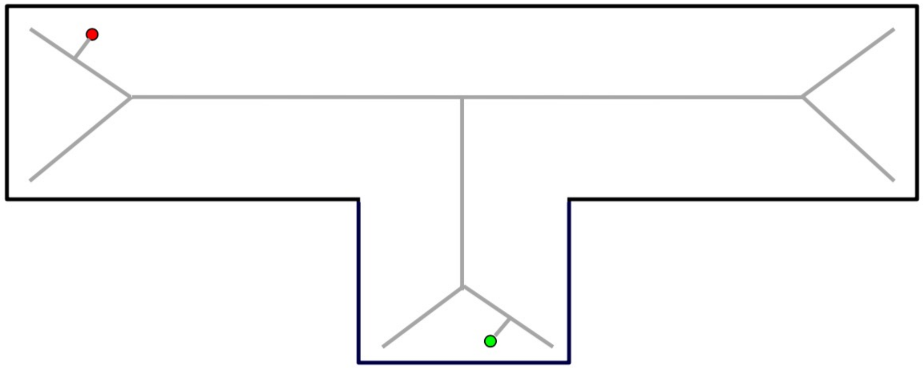

For Serious Sam 3,
we have decided to tackle
SAMSUNG
FOREST PARK
QUEENS

Representation

- Skeletonization
  - Visibility graphs
  - **Voronoi diagrams**
  - ...
- Cell decomposition
  - n-connected grids
  - ...

Voronoi diagram-based graph (I)

- Edges: Boundaries in Voronoi diagram
- Vertices: Intersection of boundaries
- Add start and goal vertices
- Add edges that correspond to:
  - Shortest path segment from start to the nearest segment on the Voronoi diagram
  - Shortest path segment from goal to the nearest segment on the Voronoi diagram



Source

Voronoi diagram-based graph (II)

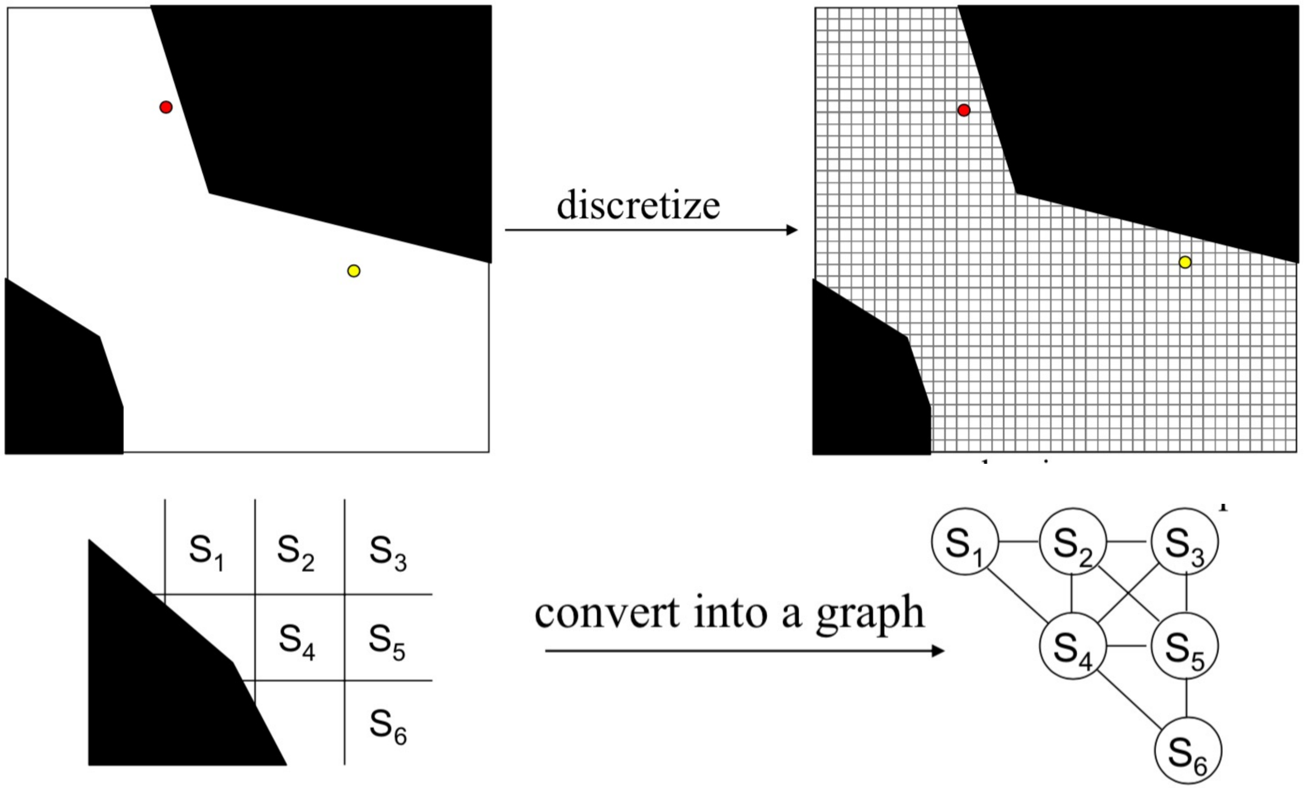
- Tends to stay away from obstacles (+)
- Independent of the size of the environment (+)
- Can work with any obstacles represented as set of points (+)

- Can result in highly suboptimal paths (-)
- Hard to deal with the cost function that is not distance (-)
- Hard to use/maintain beyond 2D (-)

Representation

- Skeletonization
  - Visibility graphs
  - Voronoi diagrams
  - ...
- Cell decomposition
  - **n-connected grids**
  - ...

Cell descomposition (I)

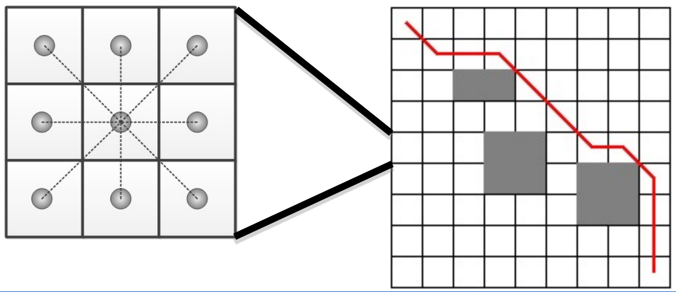


Source

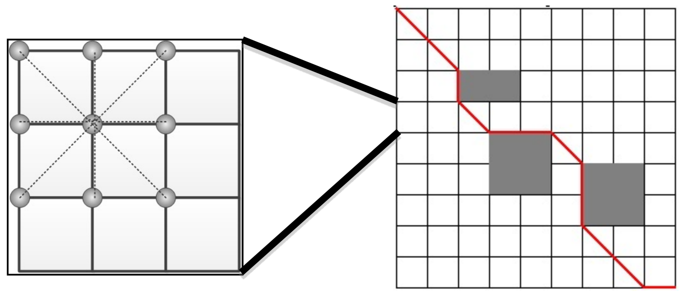
Cell descomposition (II)

- Classical path-planning algorithms are based on A* algorithm
- Works over 2D grids with blocked and unblocked cells
- Nodes are (*usually*) 8-connected with its neighbors
- Two representations:

Center node



Corner node



Cell descomposition (III)

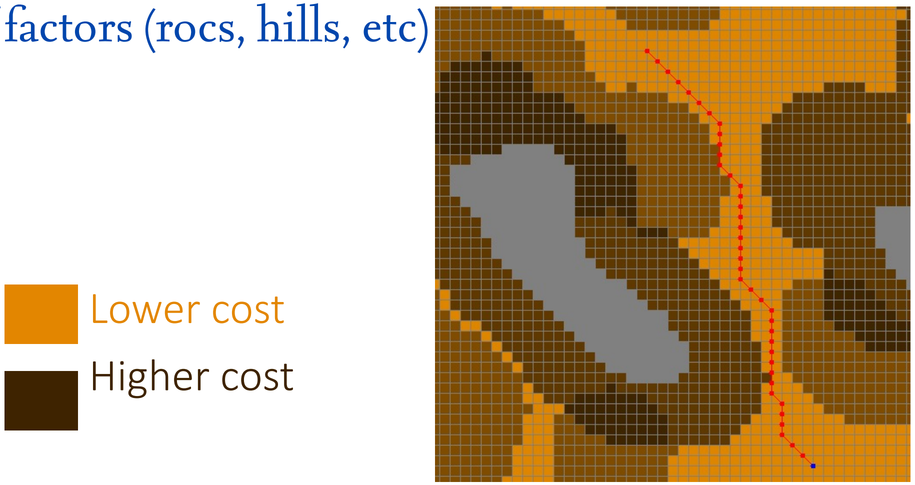
- Very simple to implement (super popular) (+)
- Can represent any dimensional space (+)
- Works well with obstacles represented as set of points (+)
- Works with any cost function (+)

- Size does depend on the size of the environment (-)
- Expensive to maintain/compute grids of dimensions > 3 (-)

Cell descomposition (IV)

- Costs maps
  - Extension of 2D maps
  - Add information
  - Tipically, lineal combination of factors (rocs, hills, etc)

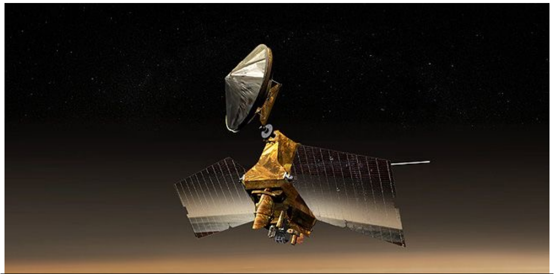
- Goal: avoid hazardous areas

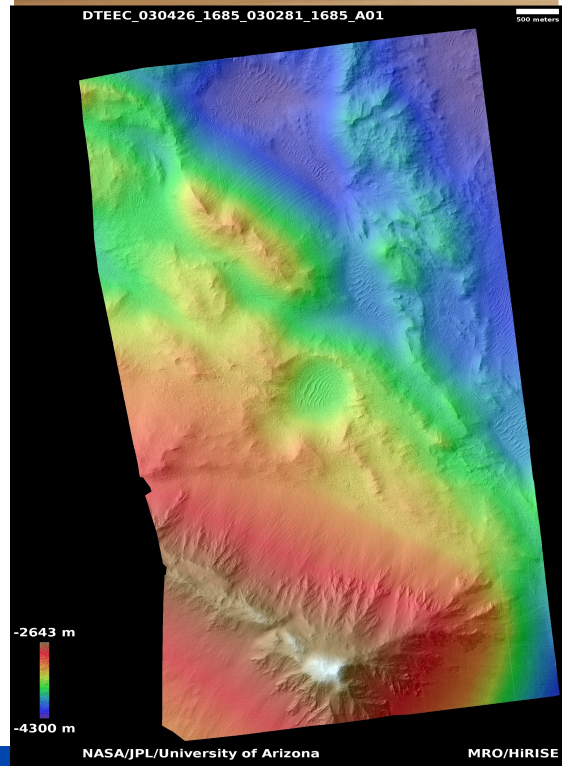


Cell descomposition (V)

- Digital terrain models (DTM)
- Mars on high resolution
  - From 2m of horizontal resolution
  - Up to 25cm!
  - Vertical resolution in dc
- Used for planning (MER/MSL)
- Free download

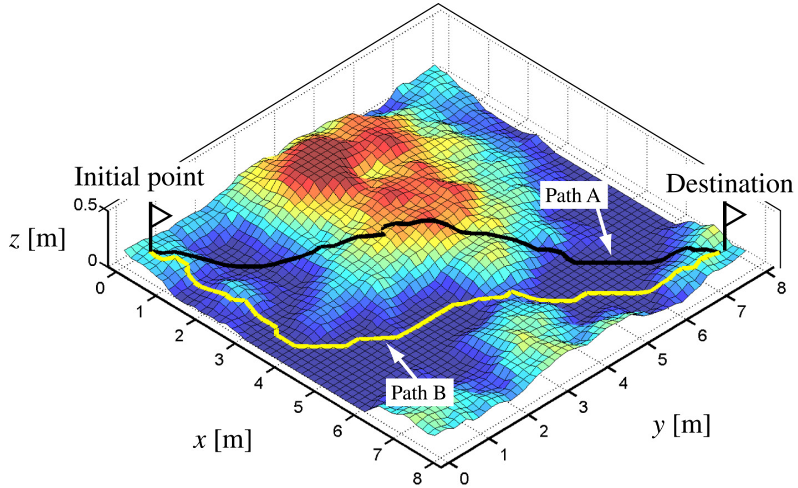
www.uahirise.org/dtm





Cell descomposition (VI)

- For dynamic environments usually a re-planning strategy is followed
- Optimize the re-planning process?
- What can be taken into consideration into the terrain?
  - Altitude → DEM
  - Hazardous areas

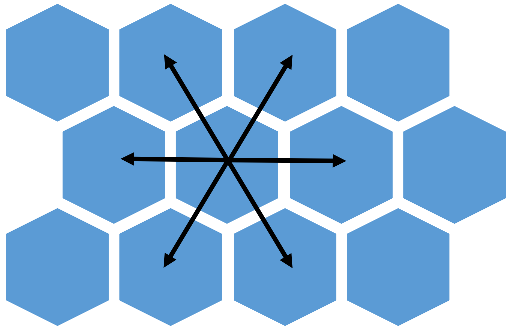


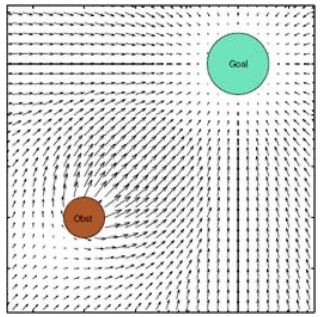
Cell descomposition (VII)

- Deterministic algorithms
  - Non Informed search
  - Heuristic search
- Stocastic algorithms
  - Tree search
  - Genetic algorithms
  - Ant colony

Cell descomposition: Others

- Hexagonal cells
  - 6 neighbours
  - More complex!
- Potential fields
  - Obeys Laplace's equation
  - Uses potential field to regulate a robot on a space
  - Difficult to implement for a real-world application
  - Poor performance in narrow passages
  - Poor performance in a dynamic environment
  - Prone to get stuck in local minima situations



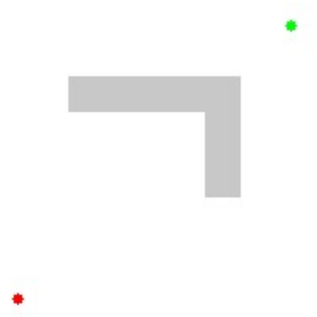


Outline

- Introduction
- Representation
- **Dijkstra**
- A*
- A*PS
- Theta*
- S-Theta*
- Heuristics
- Conclusions

Dijkstra

- Non informed search
- Pick the unvisited vertex with the lowest-distance
- Calculate the distance through it to each unvisited neighbor
- Update the neighbor's distance if smaller
- Mark visited when done with neighbors



Dijkstra

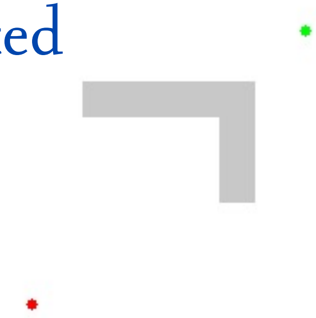
```
function Dijkstra(Graph, source):
    dist[source] := 0                        // Initializations
    for each vertex v in Graph:
        if v ≠ source
            dist[v] := infinity              // Unknown distance from source to v
            previous[v] := undefined         // Predecessor of v
        end if
        PQ.add_with_priority(v,dist[v])
    end for
    while PQ is not empty:                   // The main loop
        u := PQ.extract_min()                // Remove and return best vertex
        for each neighbor v of u:            // where v has not yet been removed from PQ.
            alt = dist[u] + length(u, v)
            if alt < dist[v]                 // Relax the edge (u,v)
                dist[v] := alt
                previous[v] := u
                PQ.decrease_priority(v,alt)
            end if
        end for
    end while
    return previous[]
```

Outline

- Introduction
- Representation
- Dijkstra
- **A***
- A*PS
- Theta*
- S-Theta*
- Heuristics
- Conclusions

A*

- A* makes guided search using two values:
  - Accumulate cost (G(t)): cost to reach a node
  - Heuristic (H(t)): predicted cost to achieve goal from a node (Euclidian distance, Octal distance)
- Node Evaluation: F(t) = G(t)+H(t)
- A* is simple, fast and guarantee optimal paths in eight-connected grids
- Artificially restricted to 45° headings



A*

Algorithm 1 A* search

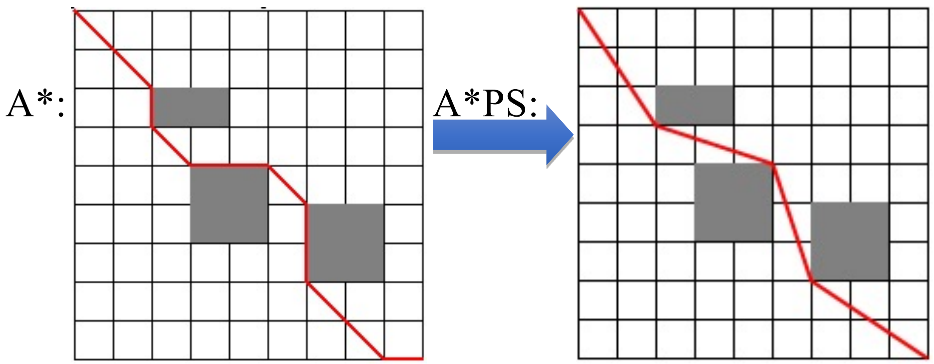
```
G(s) ← 0
parent(s) ← s
open ← ∅
open.insert(s, G(s), H(s))
closed ← ∅
while open ≠ ∅ do
    p ← open.pop()
    if p = g then
        return path
    end if
    closed.insert(p)
    for t ∈ neighbours(p) do
        if t ∉ closed then
            if t ∉ open then
                G(t) ← ∞
                parent(t) ← null
            end if
            UpdateVertex(p, t)
        end if
    end for
end while
return fail
```

Outline

- Introduction
- Representation
- Dijkstra
- A*
- **A*PS**
- Theta*
- S-Theta*
- Heuristics
- Conclusions

A* Post Processed

- A* Post Processed (A*PS) tries to smooth A* routes by removing intermediate nodes when there is line of sight between 2 no neighbors nodes, that is, there are no obstacles

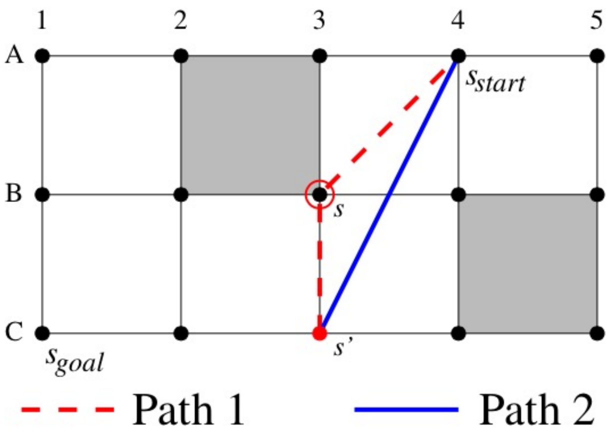


Outline

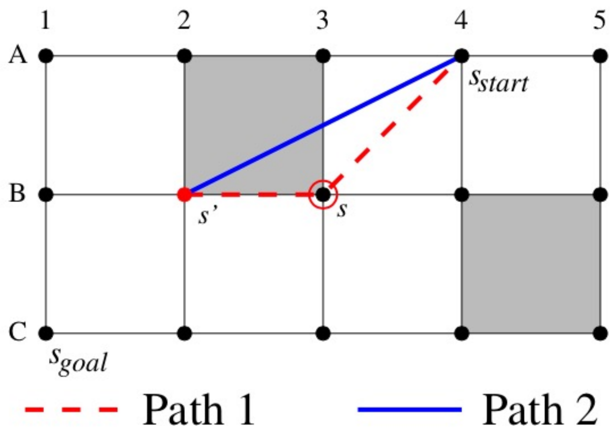
Theta*

- Theta* is a variation of A* that integrates the line of sight check during search
- For this reason Theta* is not restricted to 45º headings, and gets more realistic paths than A* without post processing
- Theta* is slower than A* due to line of sight calculation, but paths are shorter and smoothest

Theta*



(a) Path 2 is unblocked



(b) Path 2 is blocked

Source: Daniel et al. 2010

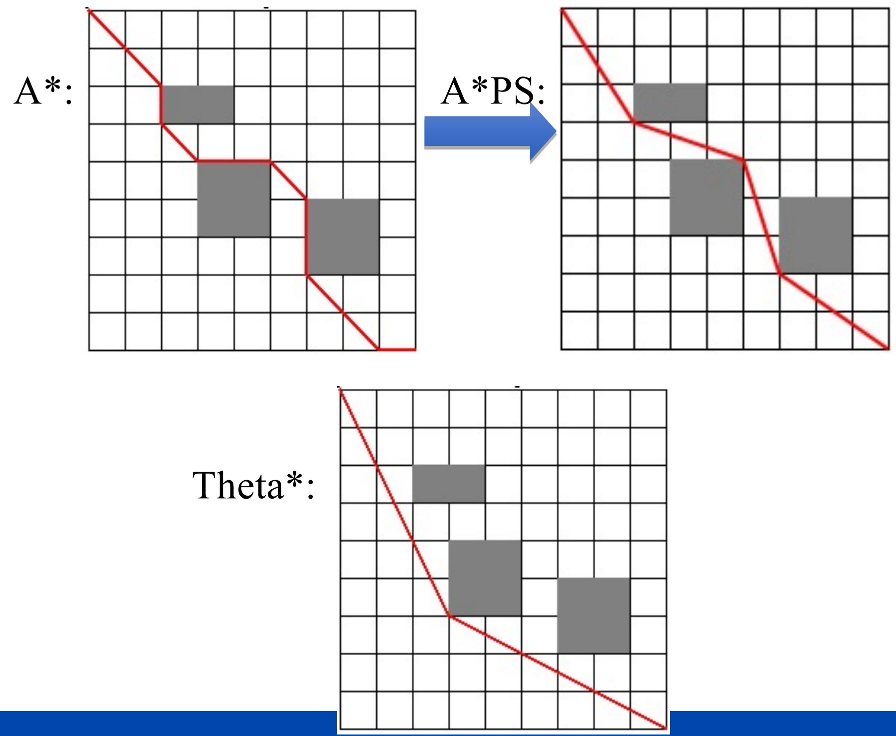
Theta*

Algorithm 2 Update vertex function for Basic Theta*

```
UpdateVertex(p, t)
if LineOfSight(parent(p), t) then
    if G(parent(p)) + dist(parent(p), t) < G(t) then
        G(t) ← G(parent(p)) + dist(parent(p), t)
        parent(t) ← parent(p)
        if t ∈ open then
            open.remove(t)
        end if
        open.insert(t, G(t), H(t))
    end if
else
    if G(p) + dist(p, t) < G(t) then
        G(t) ← G(p) + dist(p, t)
        parent(t) ← p
        if t ∈ open then
            open.remove(t)
        end if
        open.insert(t, G(t), H(t))
    end if
end if
```

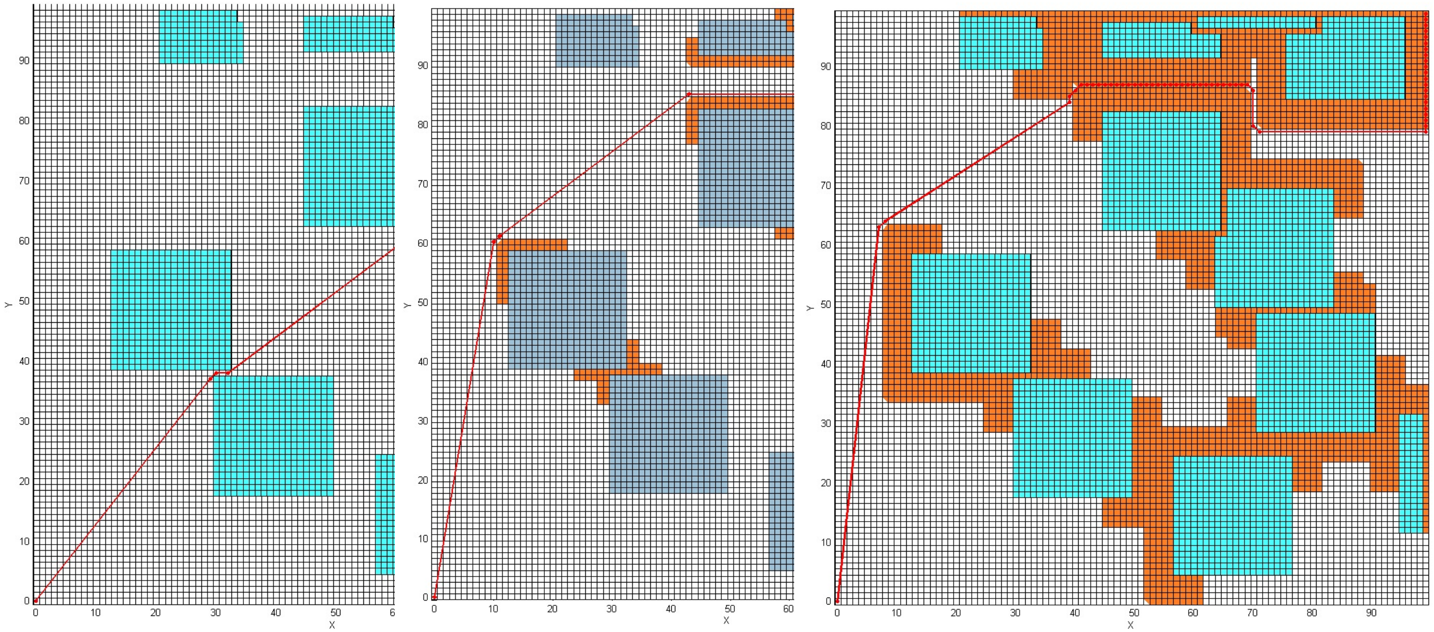
Theta*

- A*PS and Theta* are any-angle algorithms: not restricted to 45º headings



Theta*

- What about crossing the obstacles at the corner?
  → Safety margin



Outline

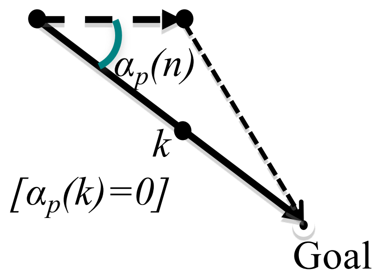
- Introduction
- Representation
- Dijkstra
- A*
- A*PS
- Theta*
- **S-Theta***
- Heuristics
- Conclusions

S-Theta*

- Theta* updates a node depending on the distance to reach it, regardless of its orientation
- Adapt Theta* to take into consideration heading changes during the search process
  - Robotics hardware is usually very limited
  - Rotation cost is greater than the movement straight
- Best path between two points in a free obstacle grid is straight line
- Achieve less heading changes in exchange of a slight degradation of the path length

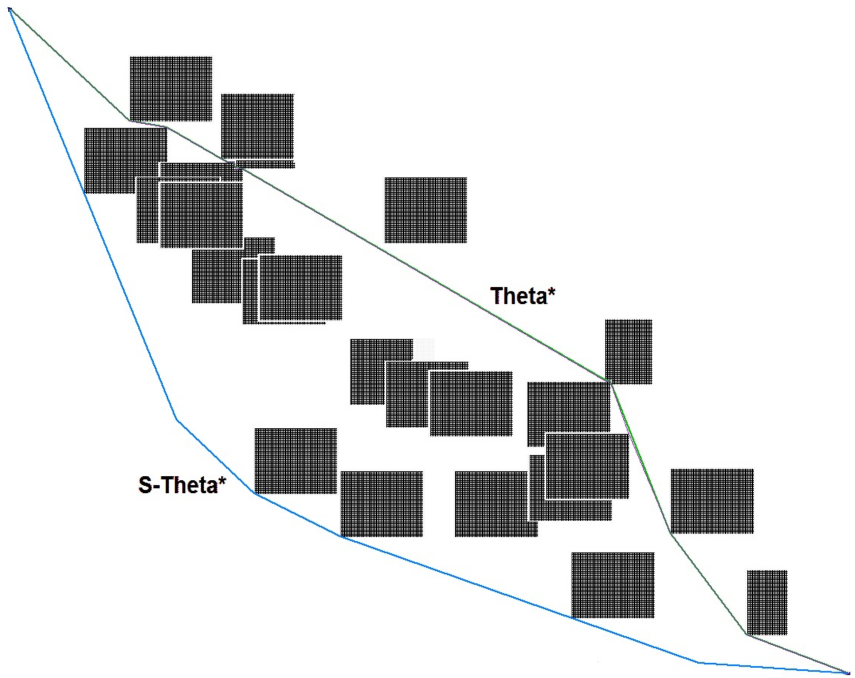
S-Theta*

- Include a new term in the cost function: $\alpha(n)$ that represents the heading change variation to reach a node n in relationship with the objective and previous nodes
- This term guides the search process to:
  - Smooth heading changes
  - Reduce number of heading changes
- $F(t) = G(t)+H(t)+\alpha(n)$



S-Theta*

- $\alpha(n)$ tries to surround obstacles and return the best path
- Does not expand nodes far from the line
- Need to weight
  - $\alpha(n) = \alpha(n) \times N$
  - $\alpha(n)$ is $[0^{\circ}, 180^{\circ}]$



Outline

- Introduction
- Representation
- Dijkstra
- A*
- A*PS
- Theta*
- S-Theta*
- **Heuristics**
- Conclusions

Heuristics

- We consider a plane with p_1 at (x_1, y_1) and p_2 at (x_2, y_2)
- Euclidean
  - $E = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
- Manhattan
  - $M = |x_1 - x_2| + |y_1 - y_2|$
- Octile
  - $\nabla x = |x_1 - x_2|$ $\nabla y = |y_1 - y_2|$
  - $O = \sqrt{1^2 + 1^2}\ .\min(\nabla x, \nabla y) + |\nabla x - \nabla y|$

Outline

- Introduction
- Representation
- Dijkstra
- A*
- A*PS
- Theta*
- S-Theta*
- Heuristics
- **Conclusions**

Conclusions

- Classical path-planning based on informed search methods provides a good approximation for big observable areas
- Heuristics are very relevant
- Representation of the environment is an important point to consider
- Large number of works, but still possibility to improve
  - Field D*, Block A*, Lazy Theta*…

Conclusions

- Planning is a repeated process
  - Partially-known environments
  - Dynamic environments
  - Imperfect execution of plans
  - Imprecise localization
- Need to be able to plan during execution (re-plan fast!)
- Several methodologies 4 replanning:
  - Anytime heuristic search: return the best plan possible within T msecs
  - Incremental heuristic search: speed up search by reusing previous efforts
  - Real-time heuristic search: plan few steps towards the goal and re-plan later