Universidad de Alcalá

Path Planning

Intelligent Systems Group

Outline

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

Introduction

- 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

- 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

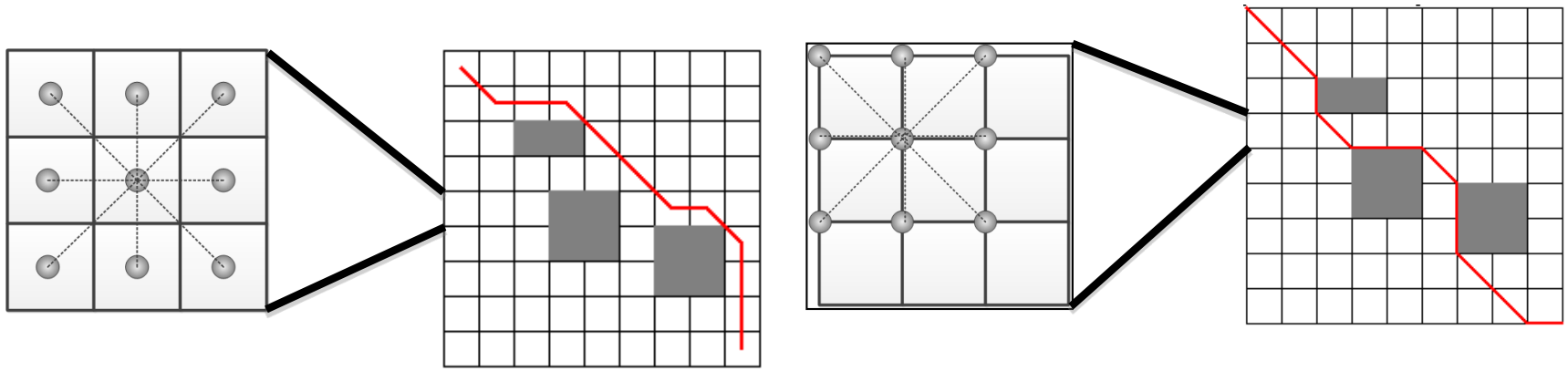
- Long term path-planning is more related to AI
- Easier with fully observable environment
- High effort trying to obtain optimal paths
- In some domains (such as planetary exploration) path-planning and task-planning are highly coupled

Introduction

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

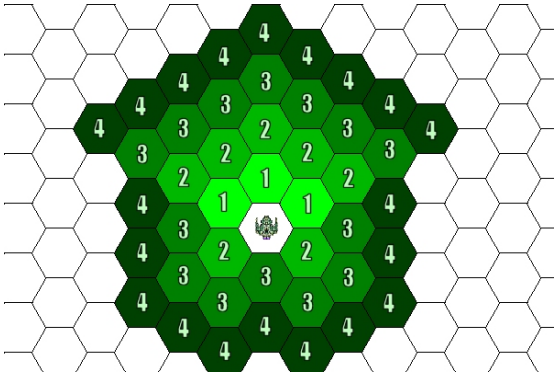
Center node

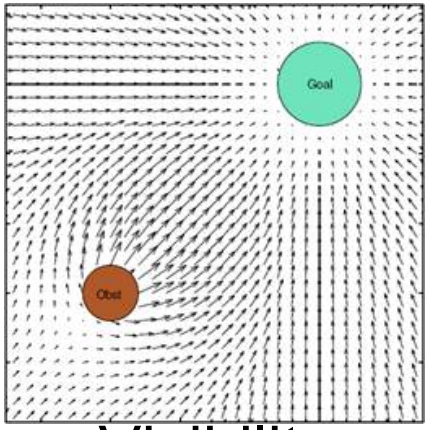
Corner node



Introduction

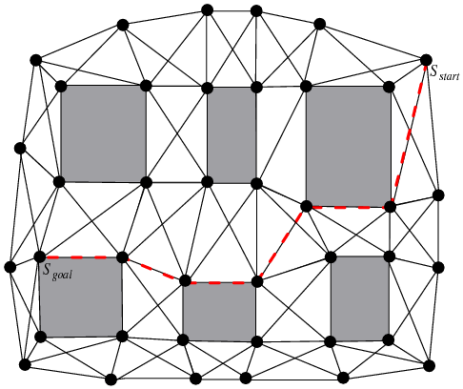
- Other representations:
  - Hexagonal regions

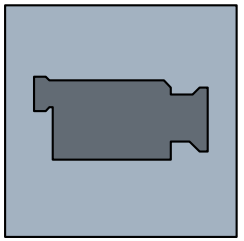




♦Potential fields

  - Visibility graphs

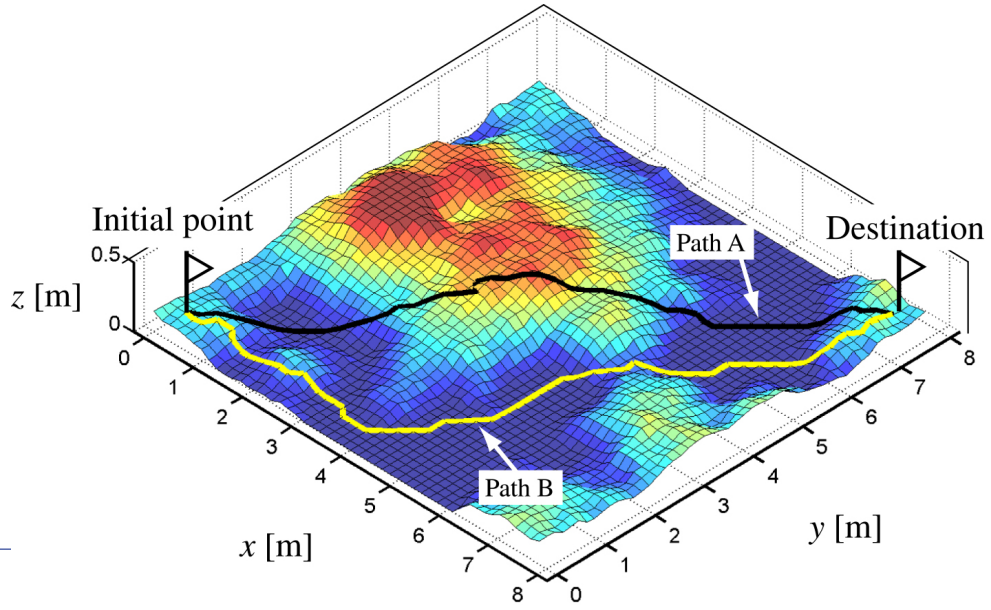




Introduction

- For dynamic environments usually a replanning strategy is followed
- Optimize the replanning process?

- What can be taken into consideration into the terrain?
  - Altitude → DEM
  - Hazardous areas

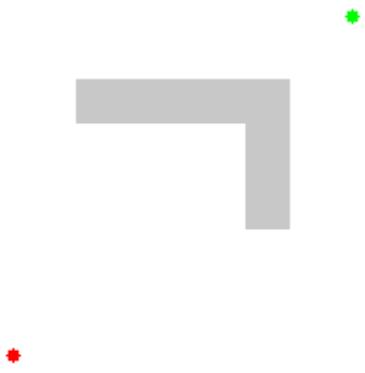


Outline

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

Dijkstra

- 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

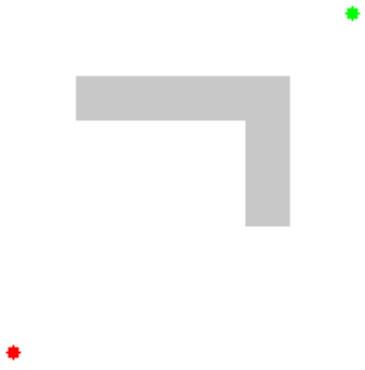
```
1  function Dijkstra(Graph, source):
2      dist[source] := 0                          // Initializations
3      for each vertex v in Graph:
4          if v ≠ source
5              dist[v] := infinity                // Unknown distance from source to v
6              previous[v] := undefined           // Predecessor of v
7          end if
8          PQ.add_with_priority(v,dist[v])
9      end for
10
11
12     while PQ is not empty:                     // The main loop
13         u := PQ.extract_min()                  // Remove and return best vertex
14         for each neighbor v of u:              // where v has not yet been removed from PQ.
15             alt = dist[u] + length(u, v)
16             if alt < dist[v]                   // Relax the edge (u,v)
17                 dist[v] := alt
18                 previous[v] := u
19                 PQ.decrease_priority(v,alt)
20             end if
21         end for
22     end while
23     return previous[]
```

Outline

- Introduction
- Dijkstra
- **A***
- A*PS
- Theta*
- S-Theta*
- 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 45º headings



A*

Algorithm 1 A* search

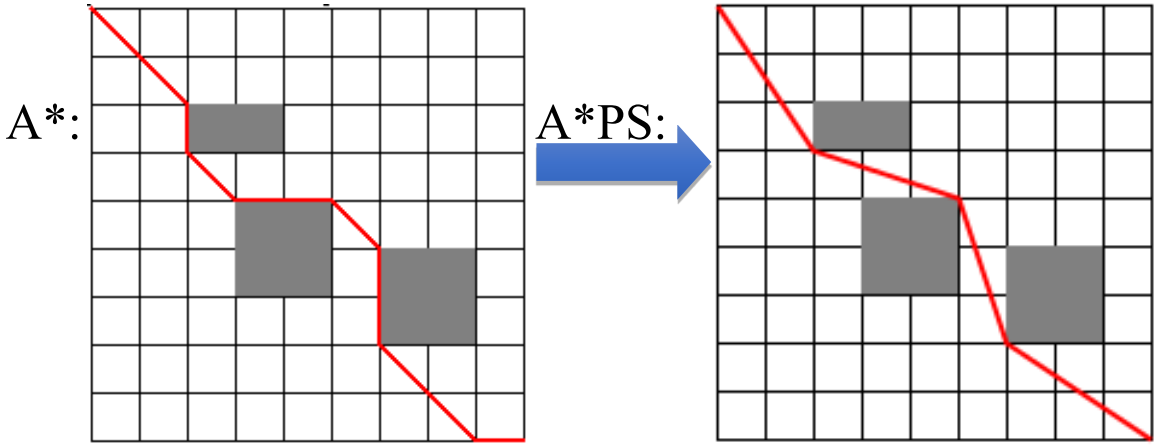
```
1  G(s) ← 0
2  parent(s) ← s
3  open ← ∅
4  open.insert(s, G(s), H(s))
5  closed ← ∅
6  while open ≠ ∅ do
7      p ← open.pop()
8      if p = g then
9          return path
10     end if
11     closed.insert(p)
12     for t ∈ neighbours(p) do
13         if t ∉ closed then
14             if t ∉ open then
15                 G(t) ← ∞
16                 parent(t) ← null
17             end if
18             UpdateVertex(p, t)
19         end if
20     end for
21 end while
22 return fail
```

Outline

- Introduction
- Dijkstra
- A*
- **A*PS**
- Theta*
- S-Theta*
- 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



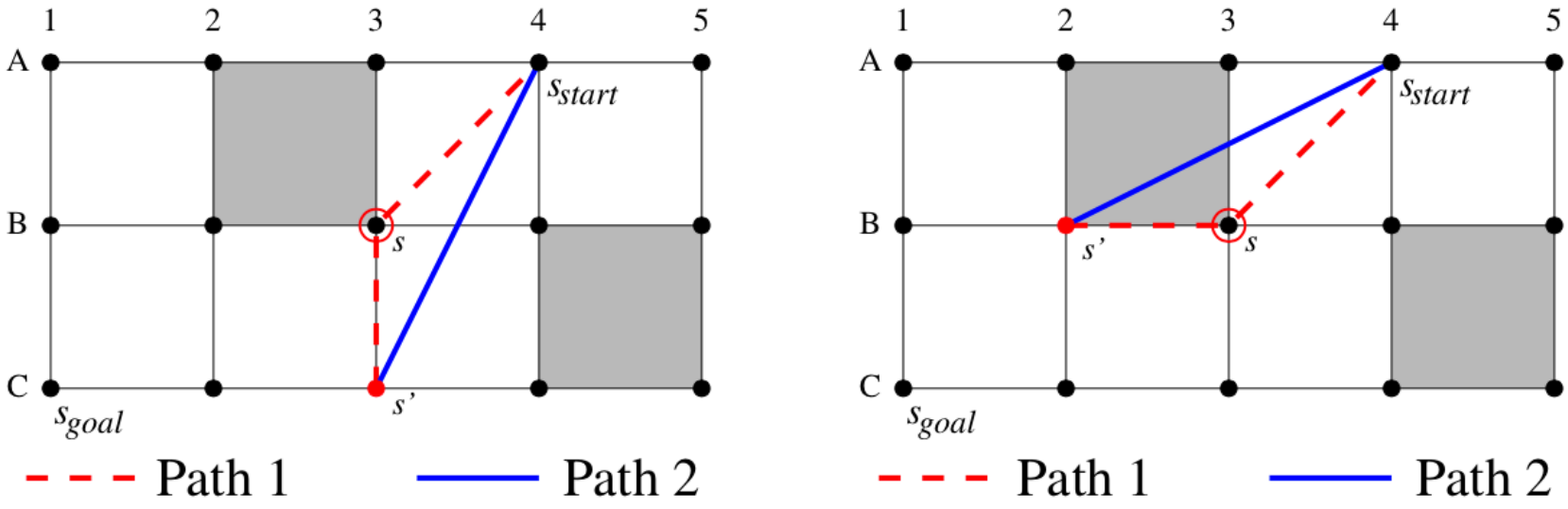
Outline

- Introduction
- Dijkstra
- A*
- A*PS
- **Theta***
- S-Theta*
- Conclusions
- Practice

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

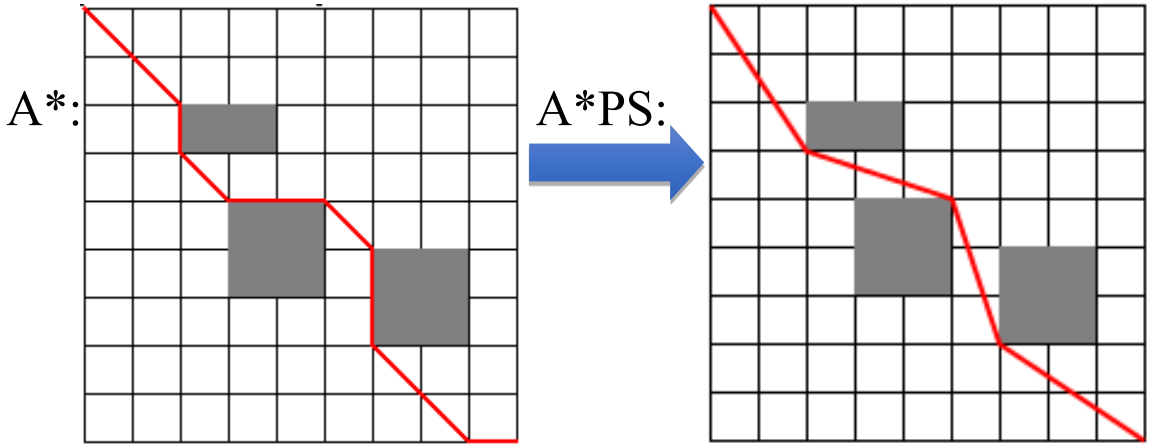
Theta*

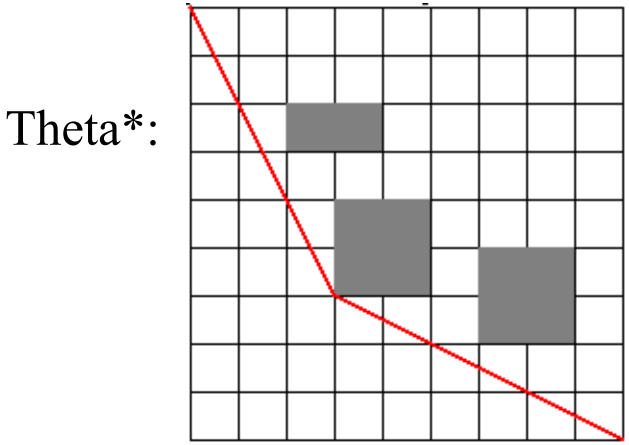
Algorithm 2 Update vertex function for Basic Theta*

```
1  UpdateVertex(p, t)
2  if LineOfSight(parent(p), t) then
3      if G(parent(p)) + dist(parent(p), t) < G(t) then
4          G(t) ← G(parent(p)) + dist(parent(p), t)
5          parent(t) ← parent(p)
6          if t ∈ open then
7              open.remove(t)
8          end if
9          open.insert(t, G(t), H(t))
10     end if
11 else
12     if G(p) + dist(p, t) < G(t) then
13         G(t) ← G(p) + dist(p, t)
14         parent(t) ← p
15         if t ∈ open then
16             open.remove(t)
17         end if
18         open.insert(t, G(t), H(t))
19     end if
20 end if
```

Theta*

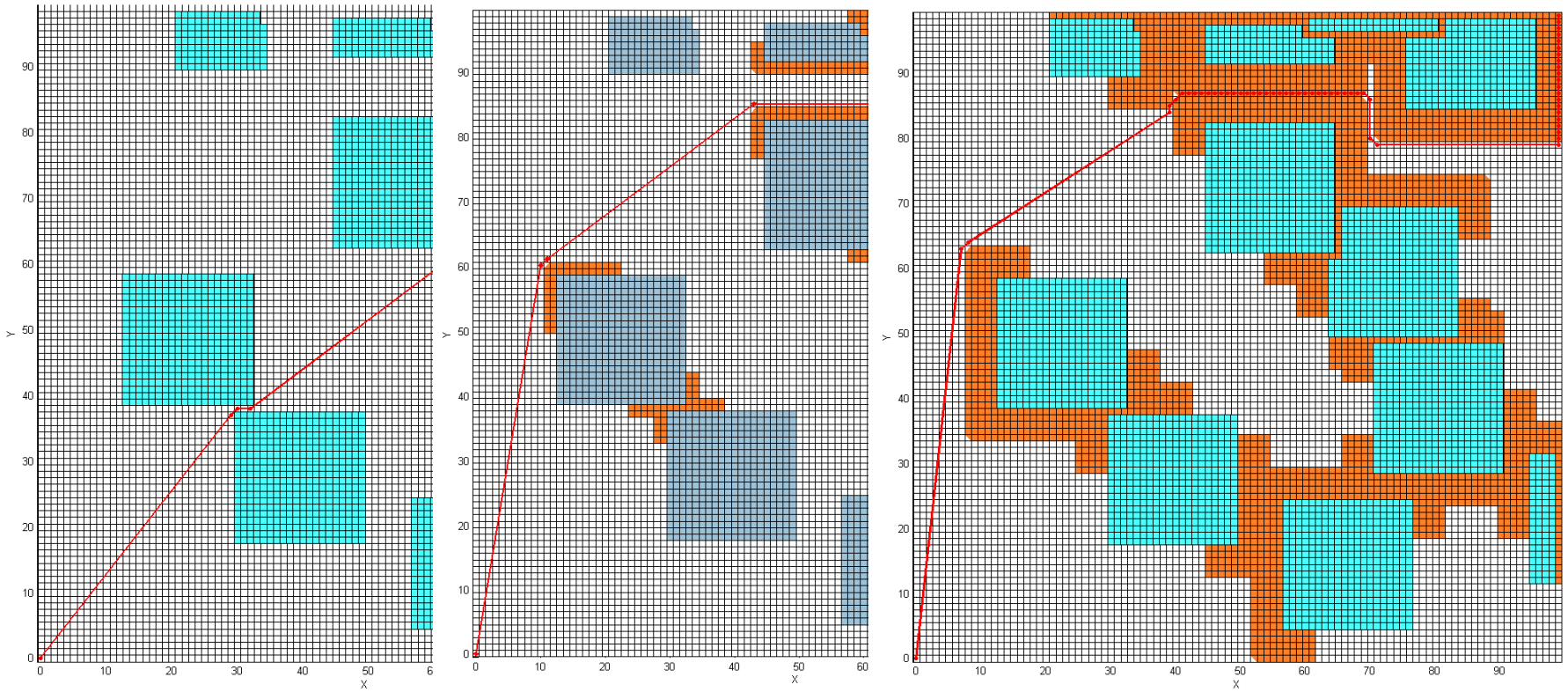
- A*PS and Theta* are any-angle algorithms: not restricted to 45º headings





Theta*

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



Outline

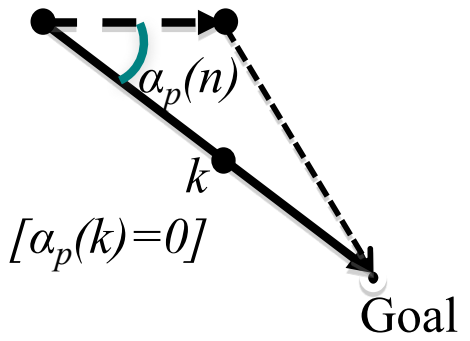
- Introduction
- Dijkstra
- A*
- A*PS
- Theta*
- **S-Theta***
- Conclusions

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

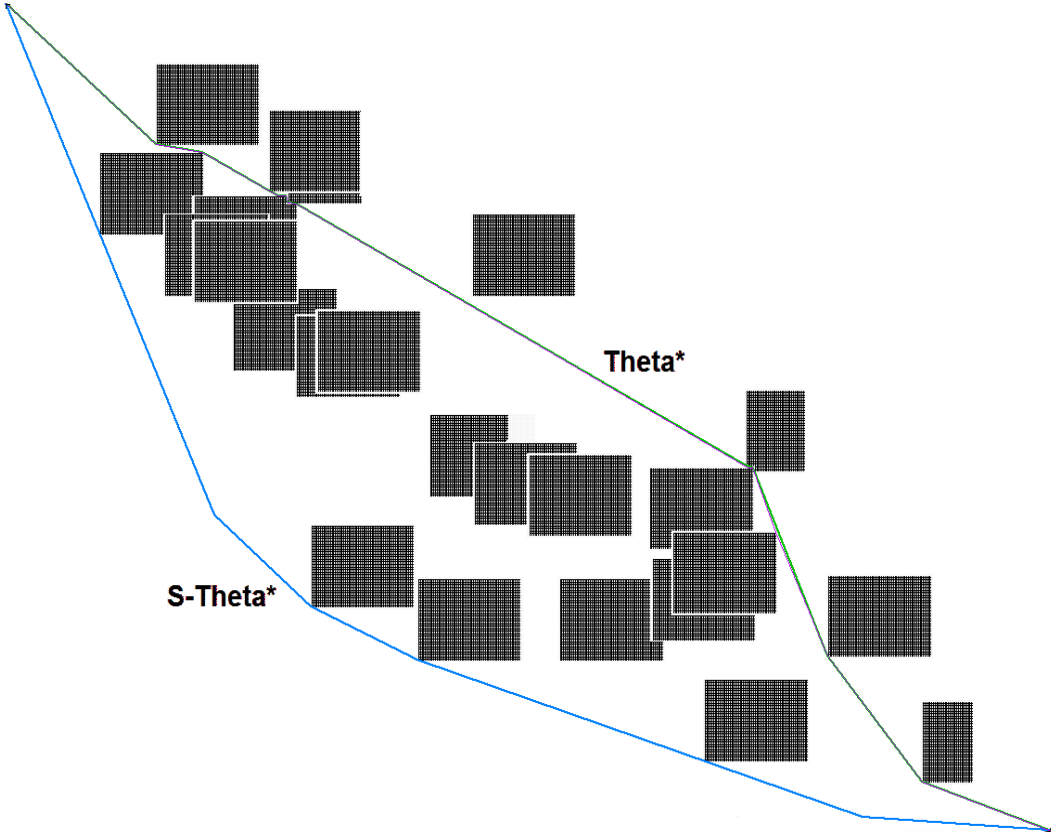
S-Theta*

- Include a new term in the cost function: α(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)+α(n)



S-Theta*

- α(n) tries to surround obstacles and return the best path
- Does not expand nodes far from the line
- Need to weight
  - α(n) = α (n) x N
  - α(n) is [0º, 180º]



Outline

- Introduction
- Dijkstra
- A*
- A*PS
- Theta*
- S-Theta*
- **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*…