SAPIENZA
UNIVERSITÀ DI ROMA

Grid-based path planning for a differential drive robot

Final Project in *Autonomous and Mobile Robotics*

June 2022

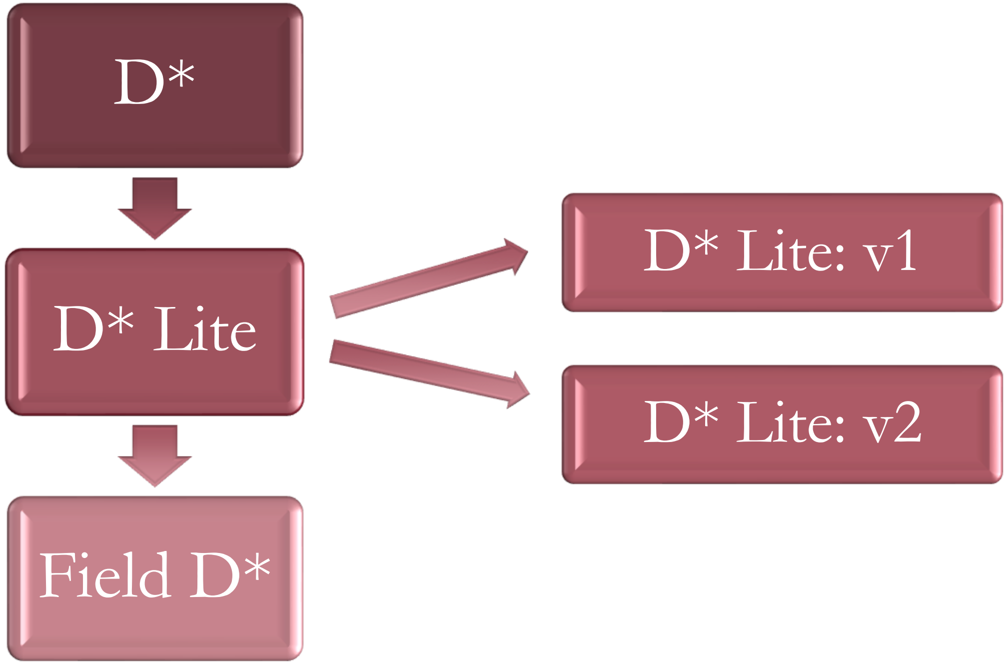
Alessandro Lambertini, Denise Landini, Gianluca Lofrumento

Introduction

Grid-based path planning for a differential drive robot

Goal of the project

Compare different algorithms that use a **grid-based methods**



ITERATIVE ALGORITHMS:

They iterate until the mobile robot reaches the goal position

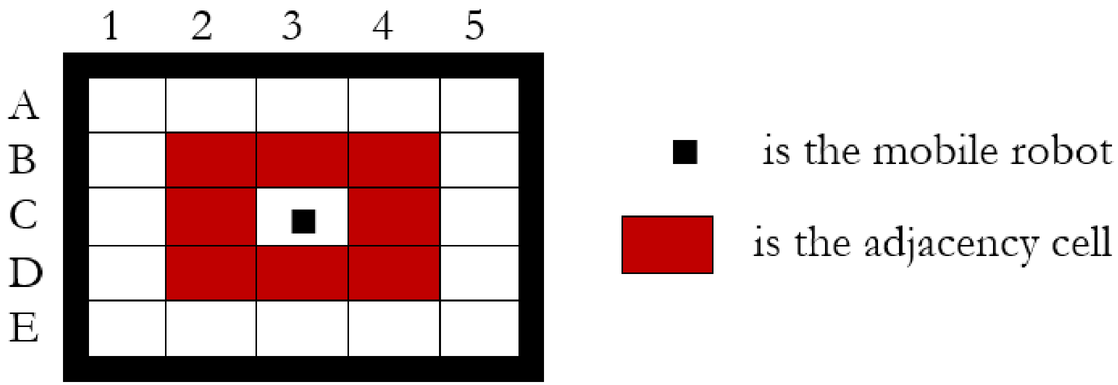
Assumption

Environment:
partially known

- Not known all the terrain in which the robot drives
- Not known all the obstacles that the robot will meet

represented by

- **Discrete grid**: each cell can have different states
- Robot: can move in **8 directions**



Goal: <u>plan a trajectory</u> from start position to a goal position by <u>replanning locally</u> the trajectory according to the environment changes

D* algorithm

Grid-based path planning for a differential drive robot

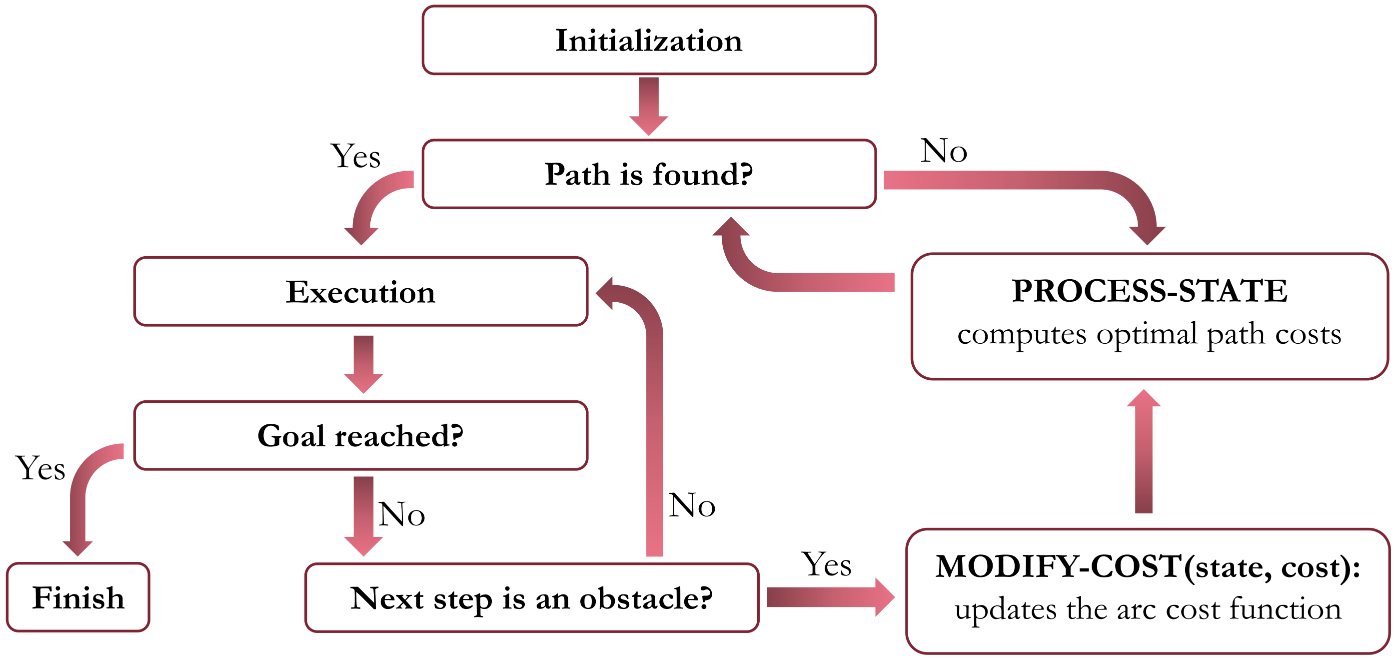
SAPIENZA
UNIVERSITÀ DI ROMA

D*: how does it work?

Goal: generate optimal trajectories and replanning

The **state** is composed by:

- coordinates (x, y) (position on the grid)
- backpointer to another state
- $tag = \{NEW, OPEN, CLOSED\}$
- path cost value: $h(Goal, State)$
- key value: $\mathrm{k}(Goal, State)$

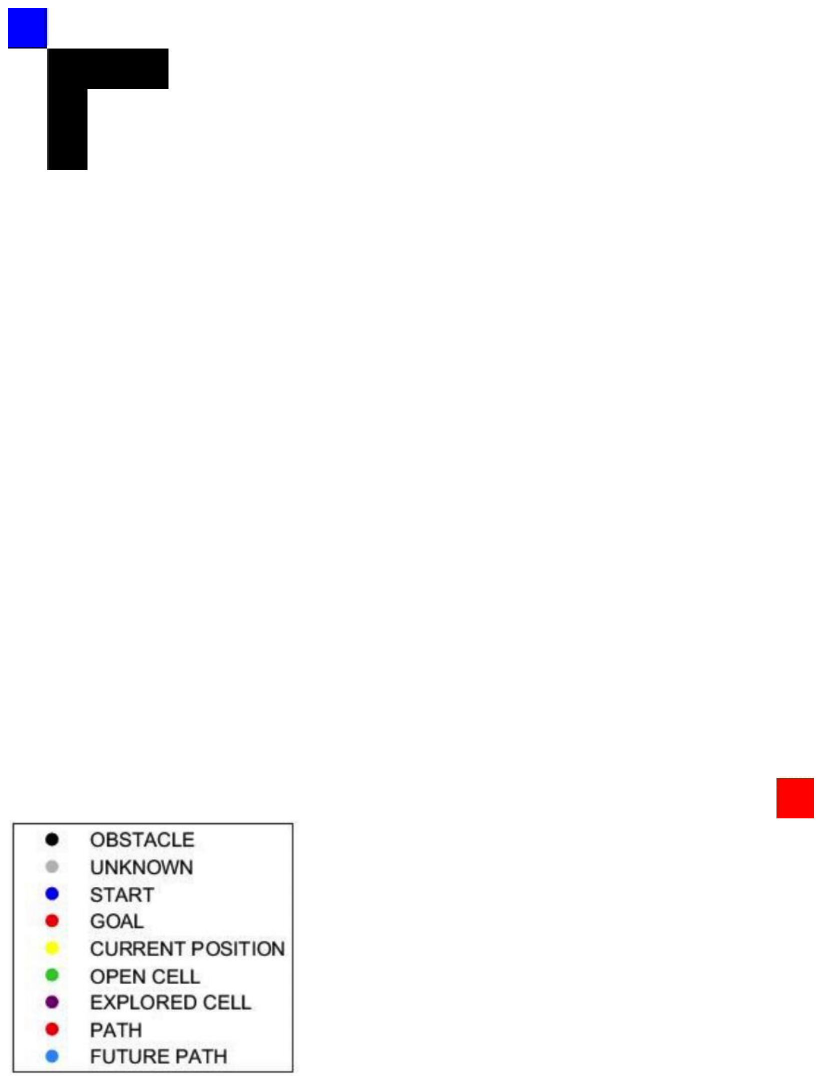


At start – Initialize():

- $tag = \{NEW\}$
- $\mathrm{k}(Goal, State) = 0$
- $h(Goal, State) = 0$

The **path cost function estimation** is the optimal, minimal, cost from the current position to the goal position, assuming the unknown cells as EMPTY

D*: how does it work?



PROCESS STATE FUNCTION:

- The lowest k value is removed from the OPEN list
- If it is a RAISE $k(X) < h(X)$, it is analyzed to reduce the cost
- If it is a LOWER, $k(X) = h(X)$, its cost is optimal, so each neighbor is analyzed to see if its path cost can be lowered.
- If it is not LOWER, then:
  - If it can lower an immediate descendant, it does
  - If it can lower a non immediate descendant, it is placed for further expansions
  - If it can be reduced by a suboptimal neighbor, the neighbor is further analyzed

Function: PROCESS-STATE ()

```
X = MIN-STATE( )
if X = NULL then return -1
k_old = GET-KMIN( ); DELETE(X)
if k_old < h(X) then
  for each neighbor Y of X:
    if h(Y) ≤ k_old and h(X) > h(Y) + c(Y, X) then
      b(X) = Y; h(X) = h(Y) + c(Y, X)
if k_old = h(X) then
  for each neighbor Y of X:
    if t(Y) = NEW or
      (b(Y) = X and h(Y) ≠ h(X) + c(X, Y)) or
      (b(Y) ≠ X and h(Y) > h(X) + c(X, Y)) then
      b(Y) = X; INSERT(Y, h(X) + c(X, Y))
else
  for each neighbor Y of X:
    if t(Y) = NEW or
      (b(Y) = X and h(Y) ≠ h(X) + c(X, Y)) then
      b(Y) = X; INSERT(Y, h(X) + c(X, Y))
    else
      if b(Y) ≠ X and h(Y) > h(X) + c(X, Y) then
        INSERT(X, h(X))
      else
        if b(Y) ≠ X and h(X) > h(Y) + c(Y, X) and
          t(Y) = CLOSED and h(Y) > k_old then
          INSERT(Y, h(Y))
return GET-KMIN( )
```

D*: Properties

Advantages:

- **Soundness**: once a state has been visited, a finite sequence from this state to the goal has been constructed
- **Optimality**: if the value returned by process-state is equal or exceed $h(Goal, State)$, then $h(Goal, State) = minimumCost(Goal,\ State)$ – so the optimal one
- **Completeness**:
  - If a path from start to goal exists, and the search space is finite, the path will be found in a finite amount of time
  - If it does not exist, it will be reported in a finite amount of time too

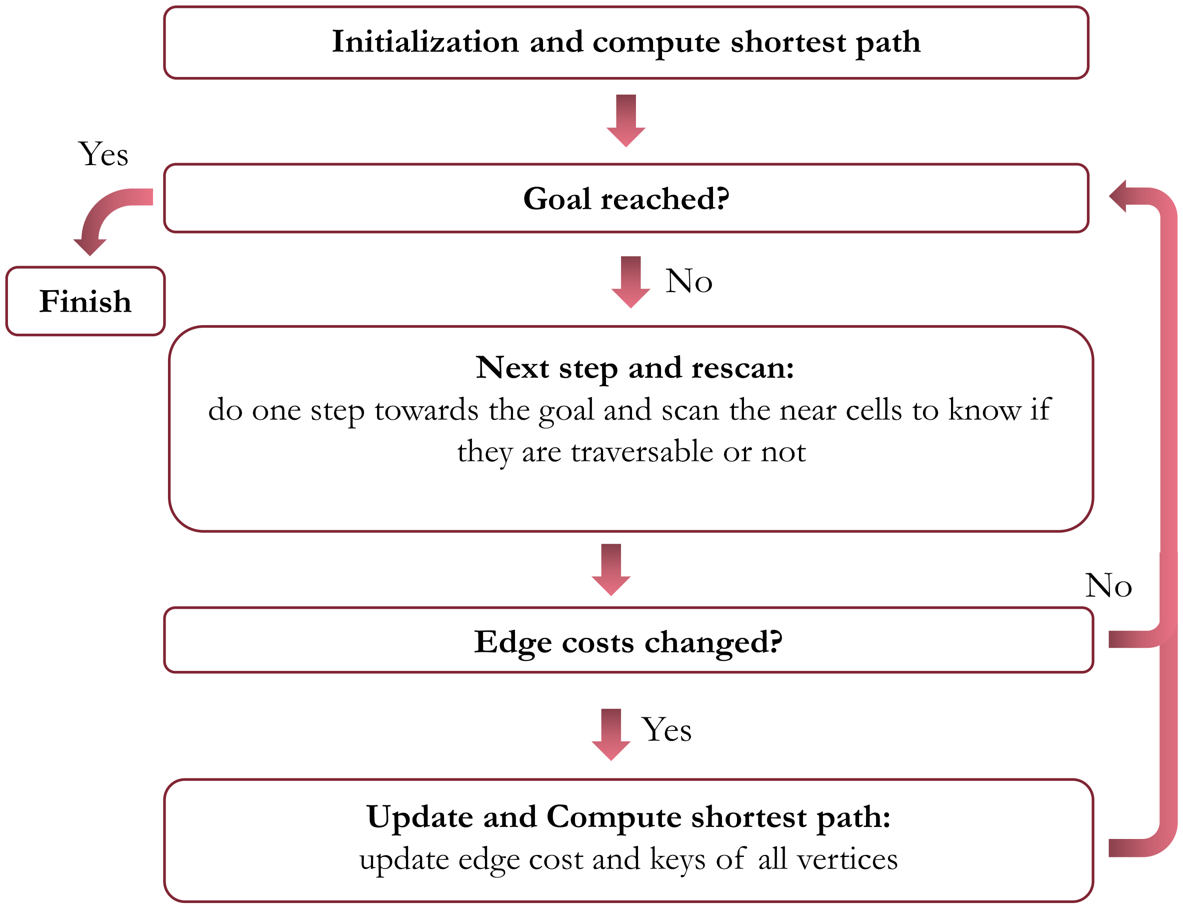
D* Lite algorithms

Grid-based path planning for a differential drive robot

Sapienza
Università di Roma

D* Lite v1: how does it work?

Goal: generate optimal trajectories and **replan in shorter time when a cost changes**



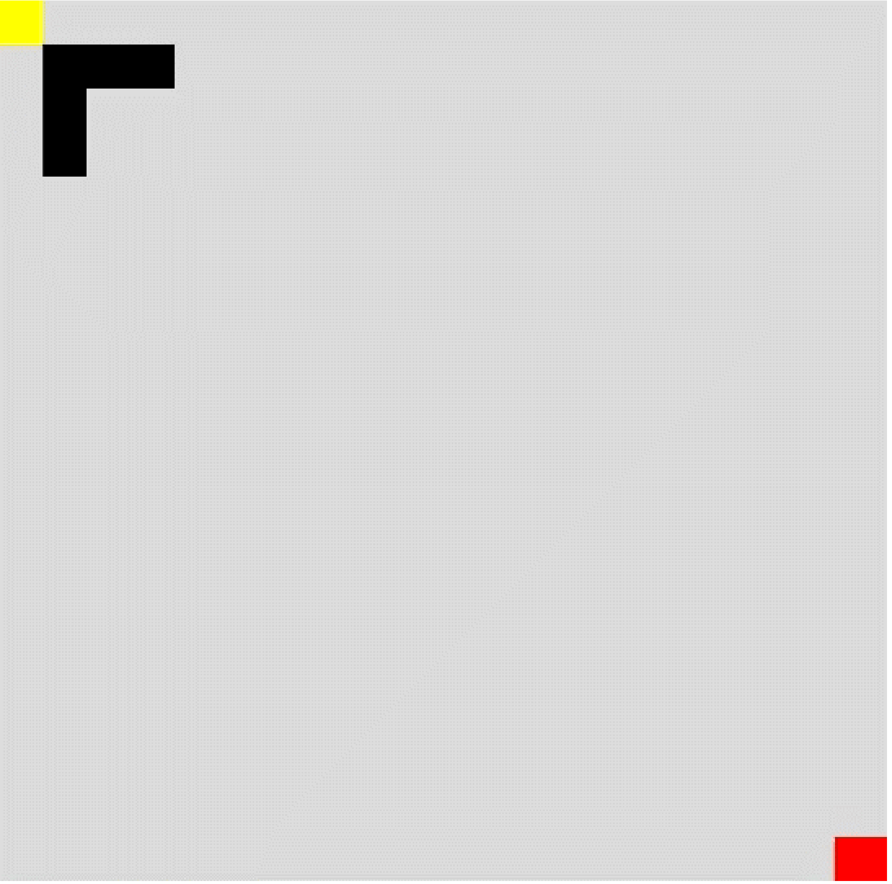
Difference with D*: heuristics change when the robot moves, so key values of the vertices need to be recomputed.

$\boldsymbol{g(s)}$: cost to move from current state to the goal state
$\boldsymbol{rhs(s)}$: righ-hand side value, it is the minimum value among the successors evaluated as $g(s') + c(s, s')$
Priority queue: list of inconsistent ordered nodes
$\boldsymbol{h(s)}$: heuristic function from state s to state s_{start}

At start – Initialize():

- $g(s) = \infty, rhs(s) = \infty, \forall s$, where s is a vertex
- $rhs(s_{goal}) = 0$
- Inserted s_{goal} in the priority queue

D* Lite v1: how does it work?



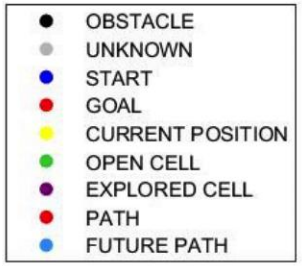
UpdateVertex function:

Update rhs-value of vertices s and, by checking their local consistency, we add/remove them from U (priority queue)

ComputeShortestPath function:

Recalculates g-values for each vertex s under conditions:

- s locally consistent
  → $g = rhs$
- s locally underconsistent vertex
  → $g = \infty$




```
procedure CalcKey(s)
{01'} return [min(g(s), rhs(s)) + h(s_start, s); min(g(s), rhs(s))];

procedure UpdateVertex(u)
{06'} if (u ≠ s_goal) rhs(u) = min_{s'∈Succ(u)}(c(u, s') + g(s'));
{07'} if (u ∈ U) U.Remove(u);
{08'} if (g(u) ≠ rhs(u)) U.Insert(u, CalcKey(u));
procedure ComputeShortestPath()
{09'} while (U.TopKey() <̇ CalcKey(s_start) OR rhs(s_start) ≠ g(s_start))
{10'}    u = U.Pop();
{11'}    if (g(u) > rhs(u))
{12'}       g(u) = rhs(u);
{13'}       for all s ∈ Pred(u) UpdateVertex(s);
{14'}    else
{15'}       g(u) = ∞;
{16'}       for all s ∈ Pred(u) ∪ {u} UpdateVertex(s);
procedure Main()
{17'} Initialize();
{18'} ComputeShortestPath();
{19'} while (s_start ≠ s_goal)
{20'}    /* if (g(s_start) = ∞) then there is no known path */
{21'}    s_start = arg min_{s'∈Succ(s_start)}(c(s_start, s') + g(s'));
{22'}    Move to s_start;
{23'}    Scan graph for changed edge costs;
{24'}    if any edge costs changed
{25'}       for all directed edges (u, v) with changed edge costs
{26'}          Update the edge cost c(u, v);
{27'}          UpdateVertex(u);
{28'}       for all s ∈ U
{29'}          U.Update(s, CalcKey(s));
{30'}       ComputeShortestPath();
```

D* Lite v1: Properties

Sapienza Università di Roma

Advantages:

- **Shorter** than D*: less nested conditional instructions and simplified program flow
- Same or better efficiency than D*

Disdvantage:

- Need to **reorder the priority queue**

D* Lite v2: how does it work?

Novelty: use the search method of D*, to avoid reordering the priority queue, because when robot moves, vertices remain in the correct order

Difference between v1 and v2:

- k_m **variable** when we compute new keys. In the initialization phase $k_m = 0$
- k_m **updated when edge costs change** by adding the heuristic considering the s_{start} and the s_{last}
  s_{last}: position in which there was the last change in the environment
- Computing **the shortest path**:
  - remove from the queue the vertex with the minimum value
  - $CalcKey()$ to compute the keys that it should have and manage the different situations

procedure CalcKey(s)
{01'''} return $[\min(g(s), rhs(s)) + h(s_{start}, s) + k_m; \min(g(s), rhs(s))]$;

Advantage:

No need to reorder the priority queue

Field D* algorithm

Grid-based path planning for a differential drive robot

Sapienza
Università di Roma

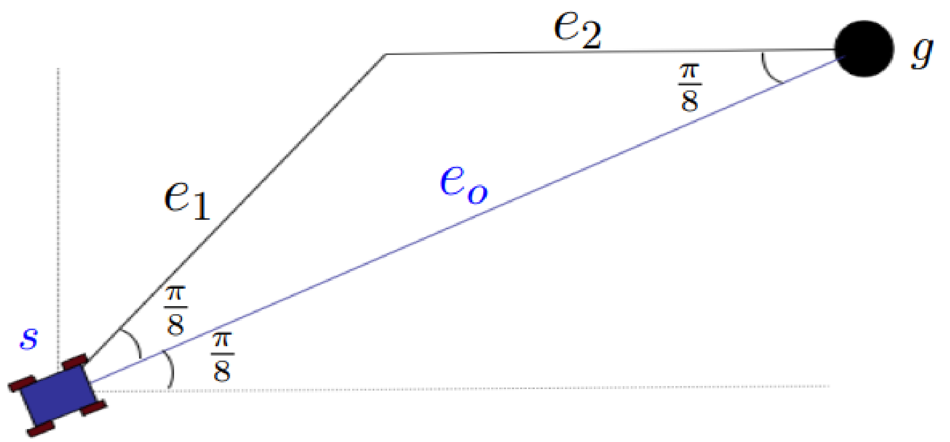
Field D*: novelty

Limitations of classical planning:

Paths produced are restricted to headings of $\frac{\pi}{4}$ increments, they may be suboptimal considering the continuous environment

Solutions:

- Post-process the generated paths (not always works).
- Switch to an interpolation planner.



Field D*: introduction

Problem of D* and D* Lite:

work with a small and discrete set of possible transitions that the mobile robot can perform

Solutions:

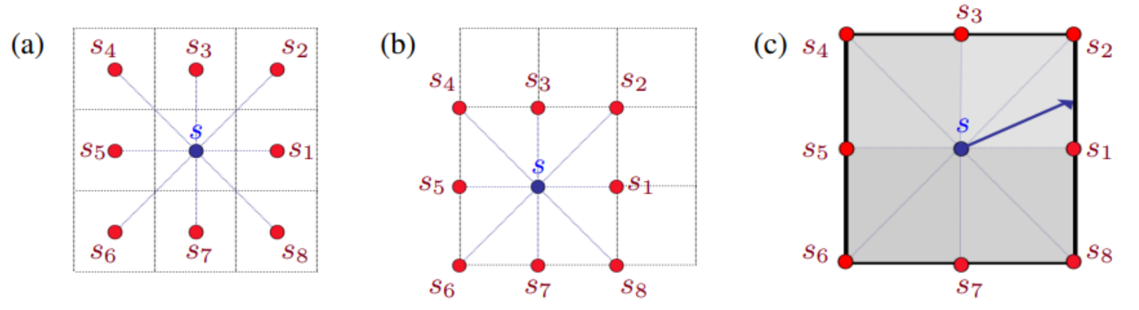
- change the computation of the path cost

Advantage:

- Smoother and low-cost path thanks to the **interpolation-based planning**

Interpolation-based cost calculation: considerations

- compute the cost from each grid node to the goal
- each node resides in the corner of the grid
- cost on the boundary of two cells is the minimum between the traversal costs of both
- manage different conditions by considering the less expensive path depends on the traversal costs



Field D*: interpolation planning

ComputeCost(s, s_a, s_b)

if (s_a is a diagonal neighbor of s)
    $s_1 = s_b;\ s_2 = s_a;$
else
    $s_1 = s_a;\ s_2 = s_b;$
c is traversal cost of cell with corners s, s_1, s_2;
b is traversal cost of cell with corners s, s_1 but not s_2;
if $(\min(c, b) = \infty)$
    $v_s = \infty;$
else if $(g(s_1) \leq g(s_2))$
    $v_s = \min(c, b) + g(s_1);$
else
    $f = g(s_1) - g(s_2);$
    if $(f \leq b)$
        if $(c \leq f)$
            $v_s = c\sqrt{2} + g(s_2);$
        else
            $y = \min(\frac{f}{\sqrt{c^2 - f^2}}, 1);$
            $v_s = c\sqrt{1 + y^2} + f(1 - y) + g(s_2);$
    else
        if $(c \leq b)$
            $v_s = c\sqrt{2} + g(s_2);$
        else
            $x = 1 - \min(\frac{b}{\sqrt{c^2 - b^2}}, 1);$
            $v_s = c\sqrt{1 + (1 - x)^2} + bx + g(s_2);$
return $v_s;$

$x = 1; y = 0$
$f < 0 \rightarrow 1$

$x = 0; y = 1$
$c < f < b \rightarrow 4$

$x = 0; y\ computed$
$f < c, b \rightarrow 3$

$x = 0; y = 1$
$c < f < b \rightarrow 4$

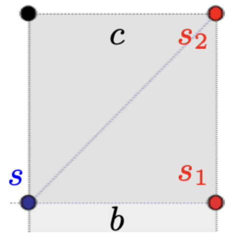
$x\ computed; y = 1$
$b < c, f \rightarrow 2$

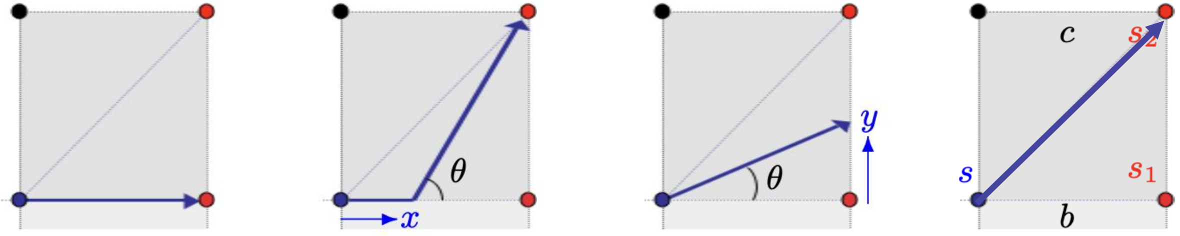
Classical planning:

$$g(s) = min_{s' \in nbrs(s)}(c(s, s') + g(s'))$$

Interpolation planning:

$$g(s) = min_{(x,y)} \left[bx + c\sqrt{(1-x)^2 + y^2} + g(s_2)y + g(s_1)(1-y)\right]$$





1. 2. 3. 4.

Field D*: how does it work?

At start – Initialize():

- $g(s) = \infty, rhs(s) = \infty,$ $\forall s$, where s is a vertex
- $rhs(s_{goal}) = 0$
- Inserted s_{goal} in the priority queue

Initialization

Goal reached?

Yes → **Finish**

No

Next step and rescan: do one step towards the goal and scan the near cells to know if they are traversable or not

Edge costs changed?

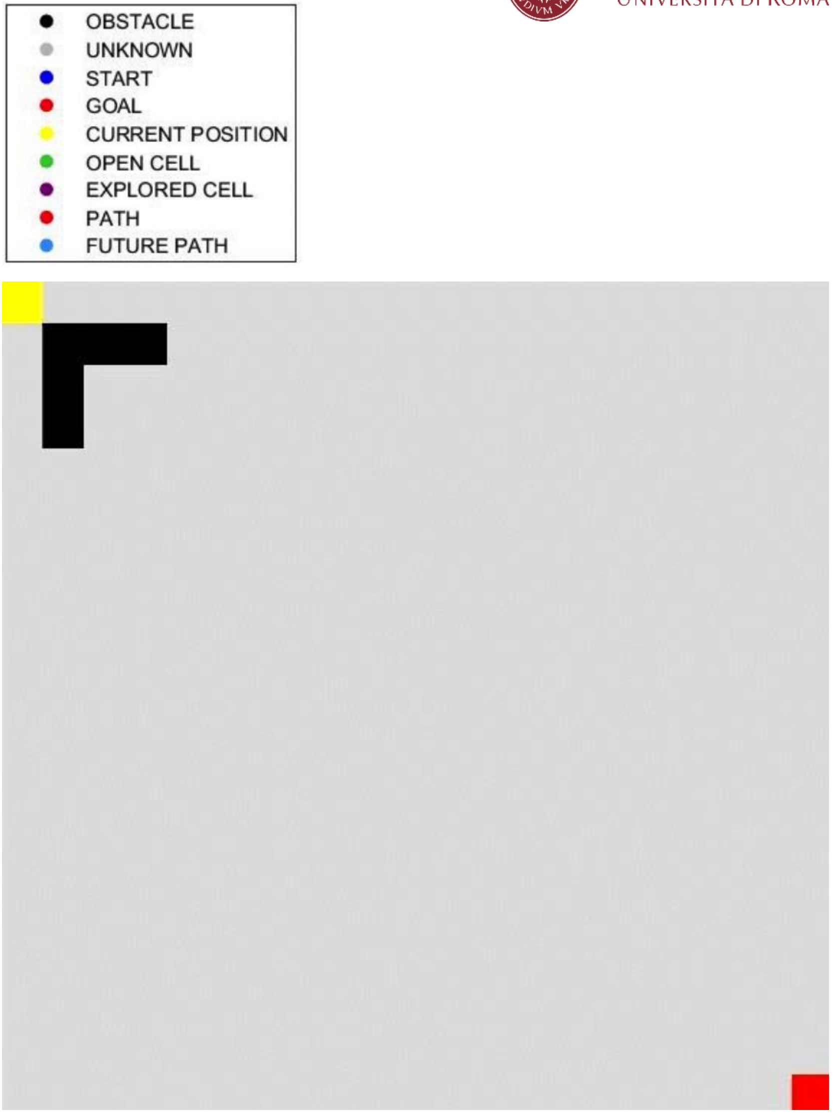
No

Yes

Update: update edge cost and keys of all vertices

Interpolation-based cost calculation:

we use a **different cost calculation**



Results and comparisons

Grid-based path planning for a differential drive robot

Results and comparisons

Algorithms were implemented with a OOP paradigm rather than procedural approach

Parameters used:

Parameters	Values
Scan Radius	3 units (cells)
Costs	1. $Cost = 0.3$ 2. $Cost = 0.6$ 3. $Cost = 1.0$ 4. $Cost = 5.0$
Map size	40×40
Obstacles %	40%
Start and goal positions	Fixed
Epochs	100

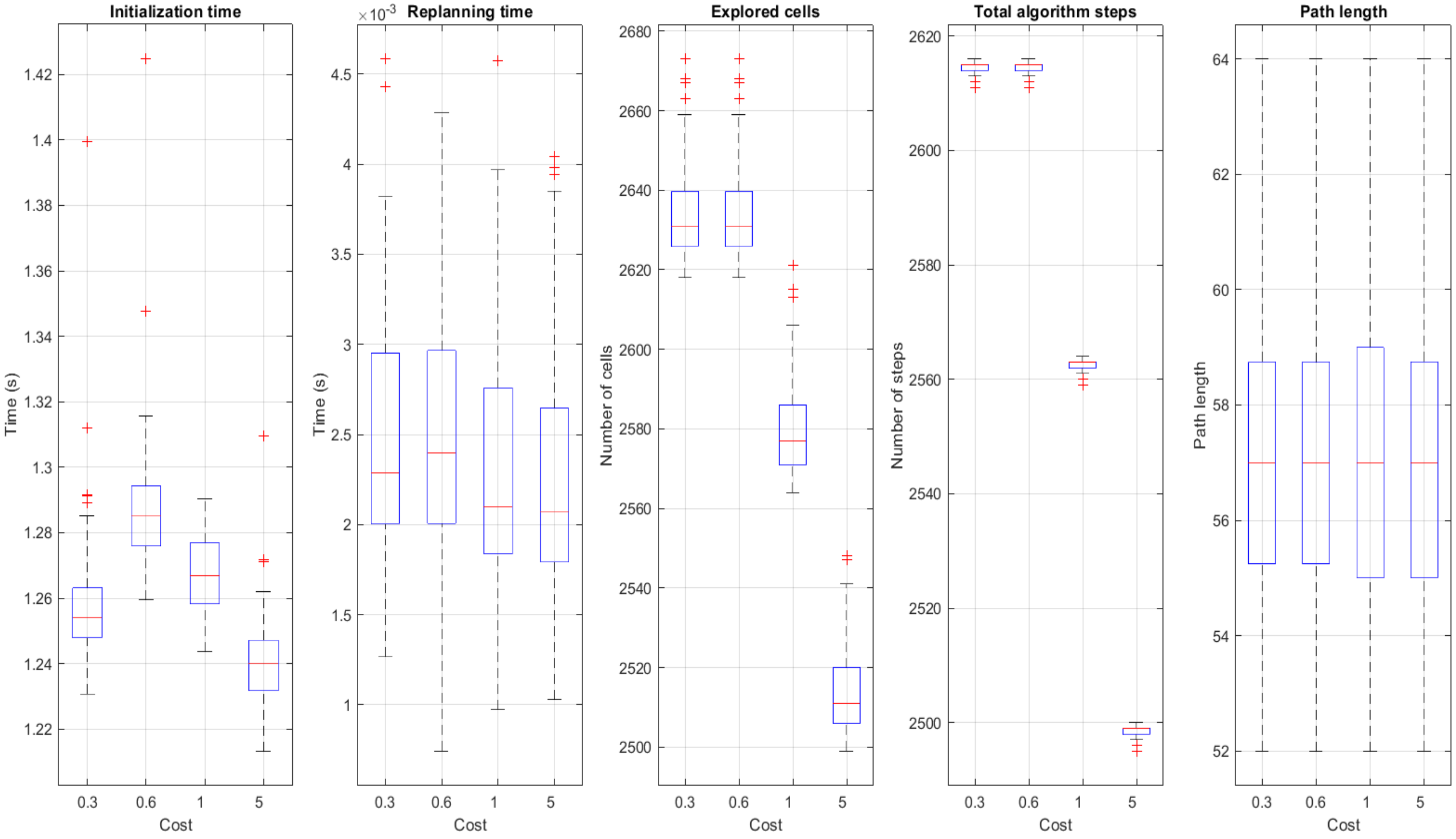
Metrics used:

Metrics	Definition
Initialization time	time needed to create the first path
Replanning time	average time needed for replanning
Number of explored cells	number of the cells the robot explores from the start to the goal
Total algorithm steps	number of times the process function is invoked
Path length	number of cells the robot has traversed to reach the goal

Results: D*

When the **cost increases:**

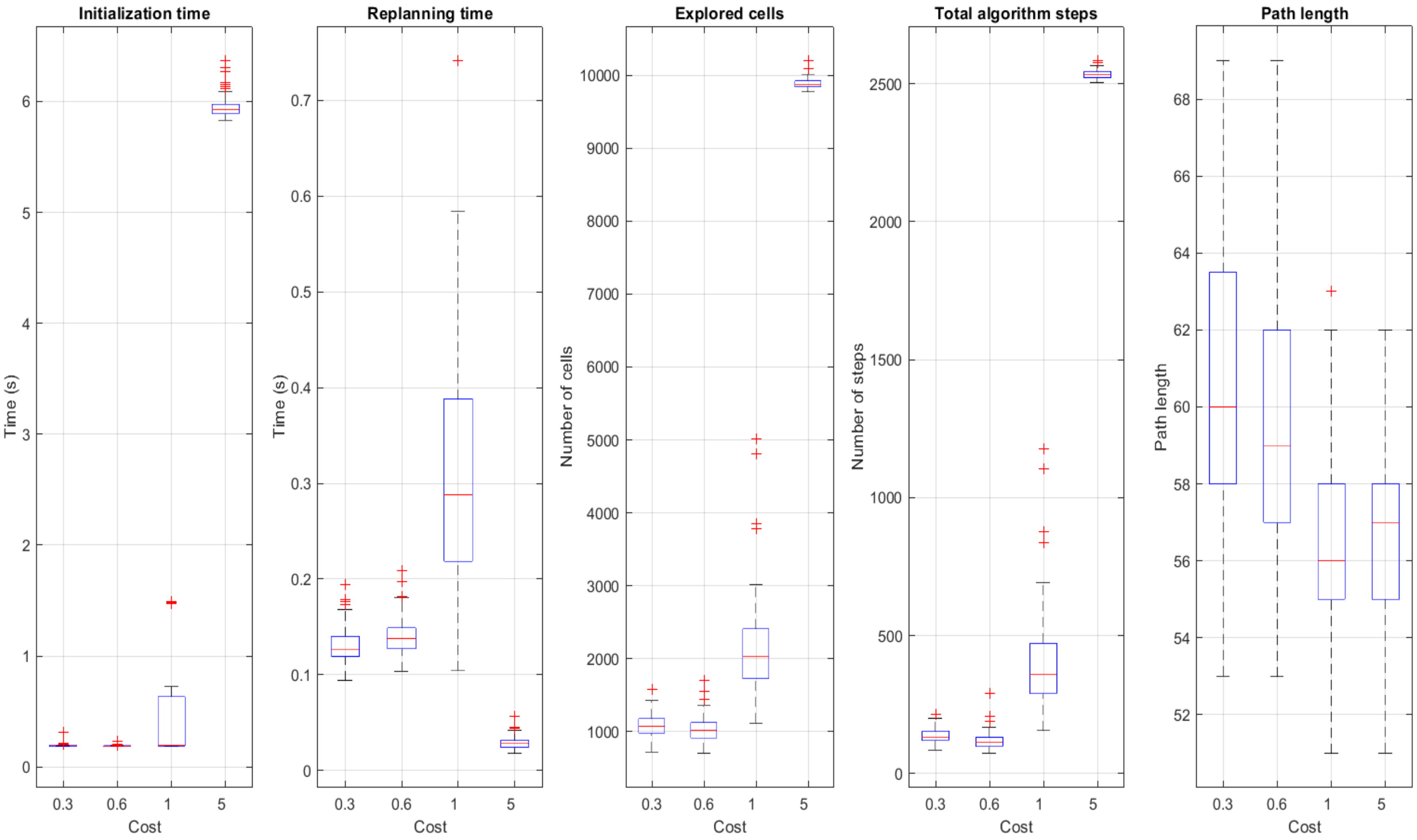
- The **initialization time** decreases starting from $cost = 0.6$
- The **replanning time** on average it is always the same around $0.0022s$
- The **explored cells** and **total algorithm steps** decrease a lot
- The **path length** has not significant changes



Results: D* Lite v1

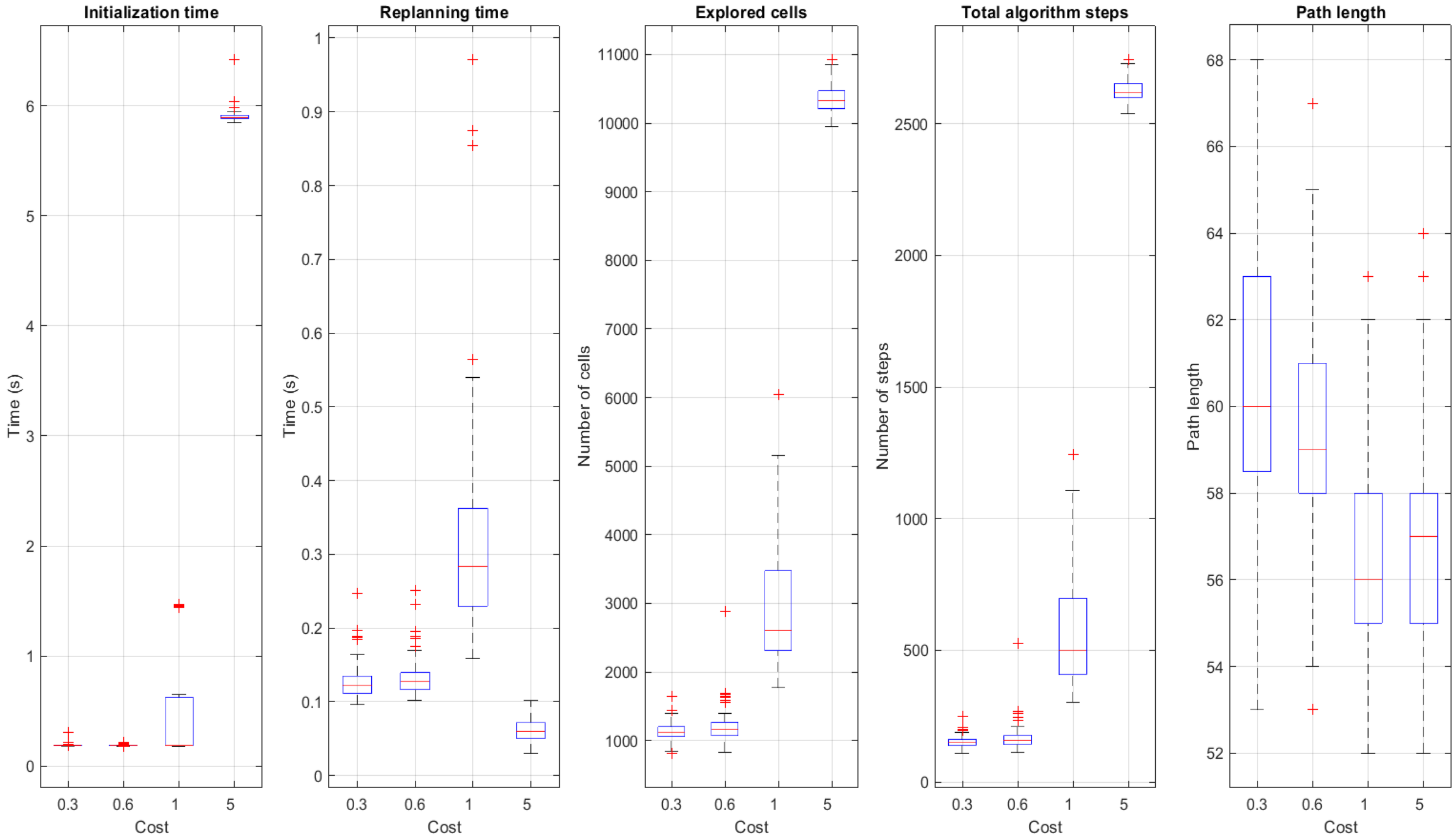
When the **cost increases:**

- **inizialization time** is very low but with $cost = 5$ it increses a lot
- **replanning time** has an incremental behaviour with a maximum when $cost = 1$ when $cost = 5$, replanning time becomes very low
- **number of explored cells** and **total algorithm steps** have the same behaviour of the initialization time
- **path length** slightly decreases



Results: D* Lite v2

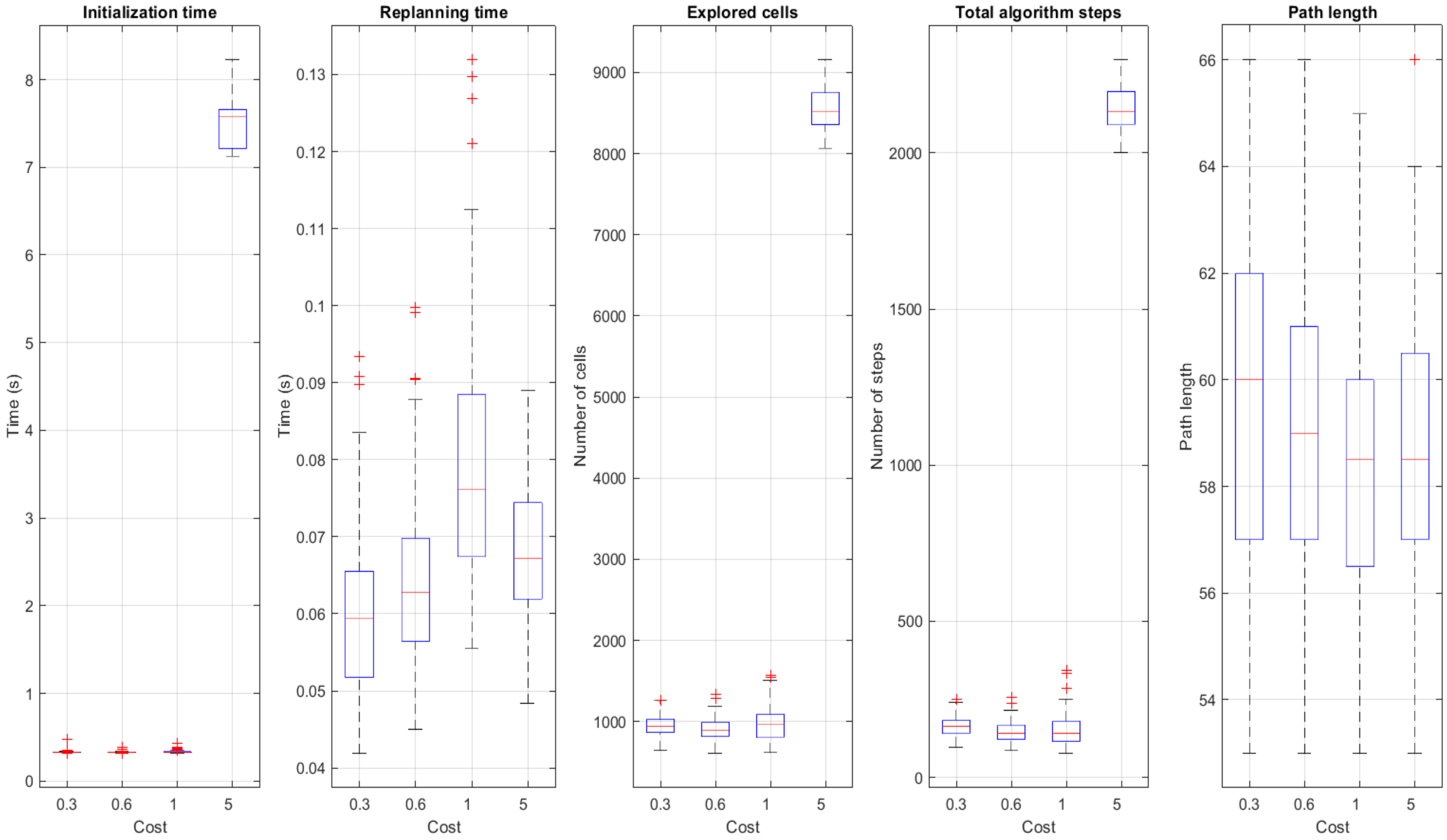
About the same behaviour of D* Lite v1, but the path length has a lower variance



Results: Field D*

When the **cost increases:**

- **inizialization time** is very low but increases a lot when $cost = 5$
- **replanning time** has an incremental behaviour with a maximum when $cost = 1$ and then it starts to decrease
- **number of explored cells** and **total algorithms steps** have the same behaviour of the initialization time
- **path length** decreases



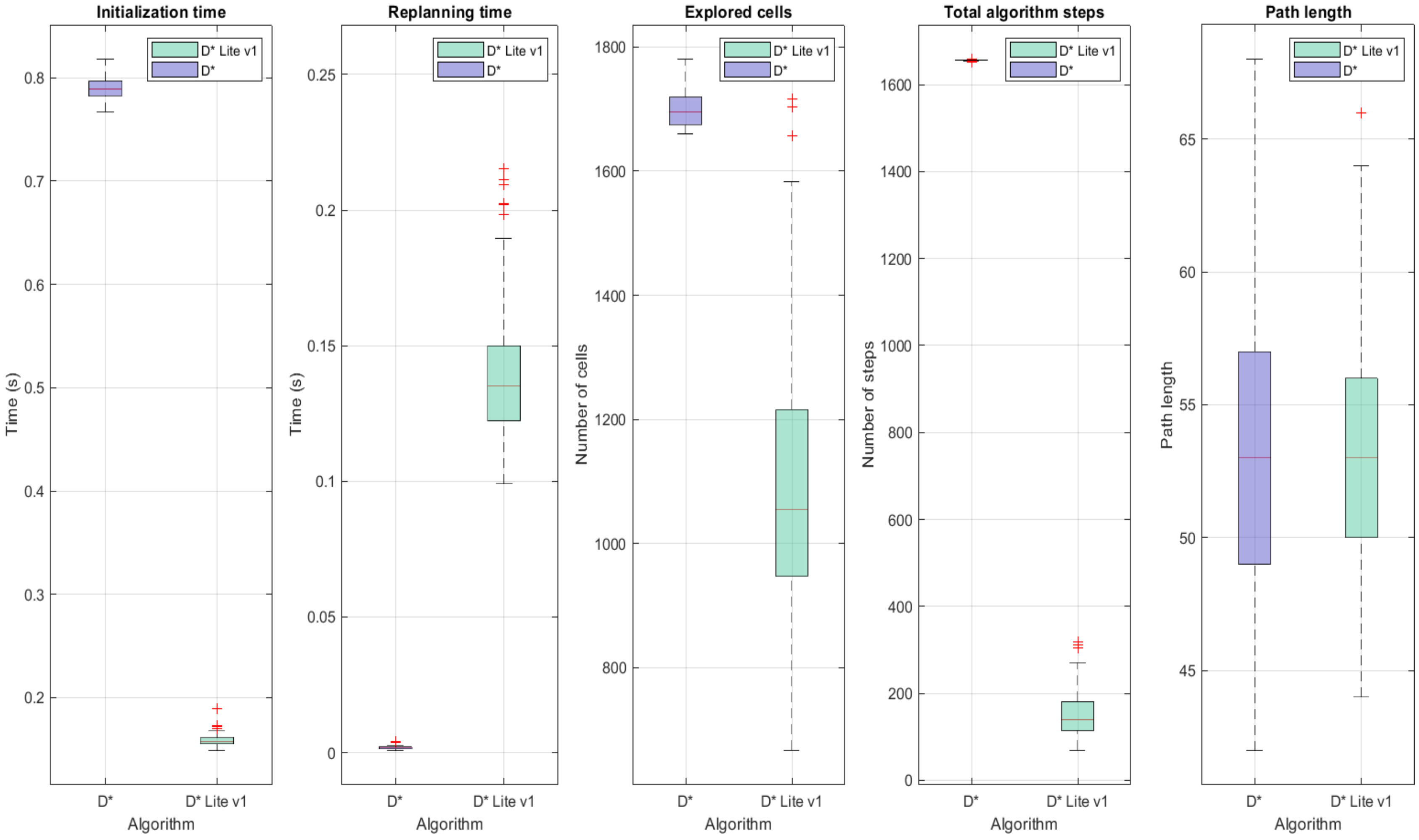
Testing and comparisons

Grid-based path planning for a differential drive robot

Random map comparisons: D* – D* Lite v1

		D*	D* Lite v1
Initialization time	75th Median 25th	0.797 s 0.789 s 0.782 s	0.161 s 0.157 s 0.155 s
Replanning time	75th Median 25th	0.002 s 0.001 s 0.000 s	0.150 s 0.135 s 0.122 s
Explored cells	75th Median 25th	1719 1695 1675	1216 1056 947
Total algorithms steps	75th Median 25th	1656 1655 1655	181 140 115
Path length	75th Median 25th	57 53 49	56 53 50

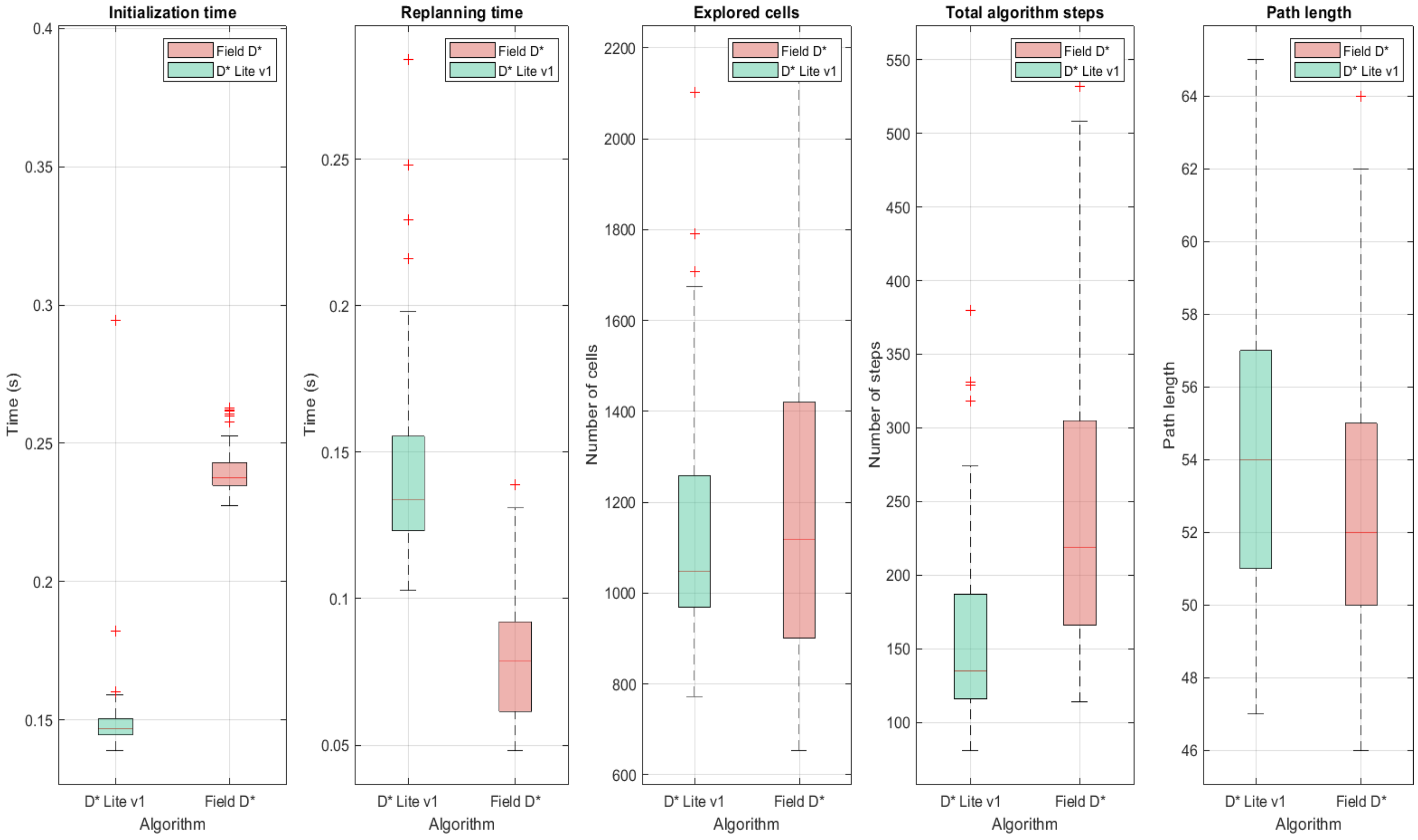
Cost D* = 1
Cost D* Lite v1 = 0.6



Random map comparisons: D* Lite – Field D*

		D* Lite v1	Field D*
Initialization time	75th Median 25th	0.149 s 0.147 s 0.145 s	0.241 s 0.237 s 0.235 s
Replanning time	75th Median 25th	0.157 s 0.134 s 0.123 s	0.092 s 0.079 s 0.062 s
Explored cells	75th Median 25th	1336 1062 969	1441 1130 901
Total algorithms steps	75th Median 25th	205 137 116	307 219 166
Path length	75th Median 25th	58 54 51	56 53 50

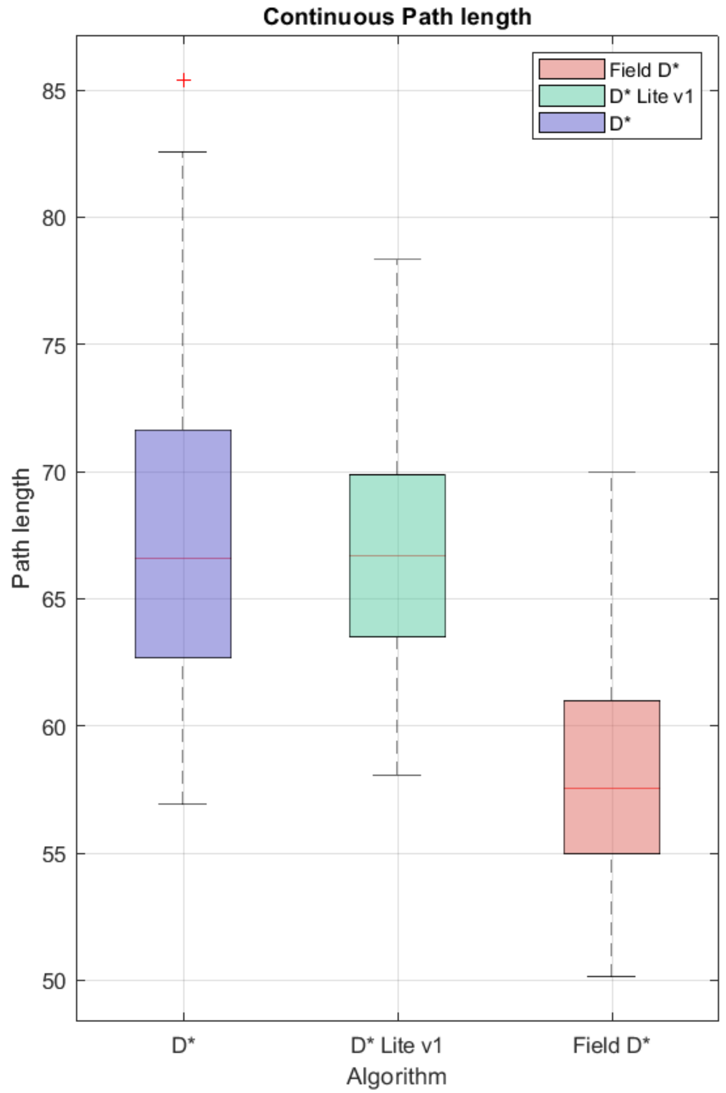
Cost D* Lite v1 = 0.6
Field D* = 0.9



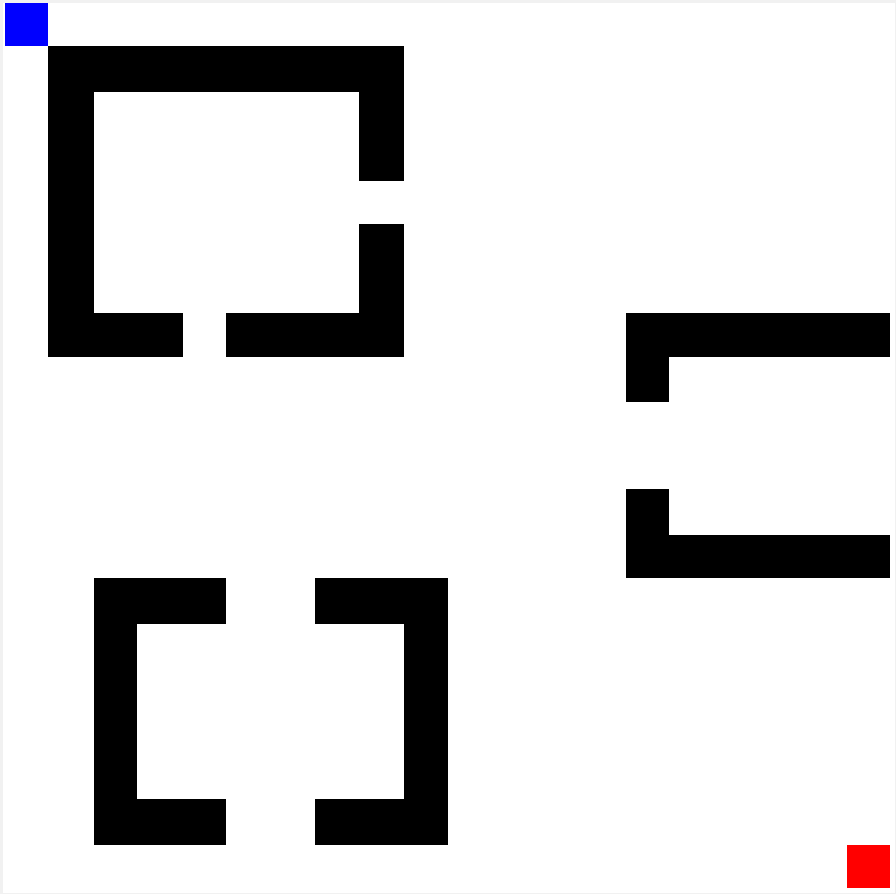
Continuous path length comparisons:
D* – D* Lite v1 – Field D*

		D*	D* Lite v1	Field D*
Continous Path length	75th Median 25th	72.083 66.740 62.912	69.598 66.669 63.498	61.556 57.583 54.971

Field D* produces the shortest path in terms of actual (continuous) distance



Map used to highlight differences

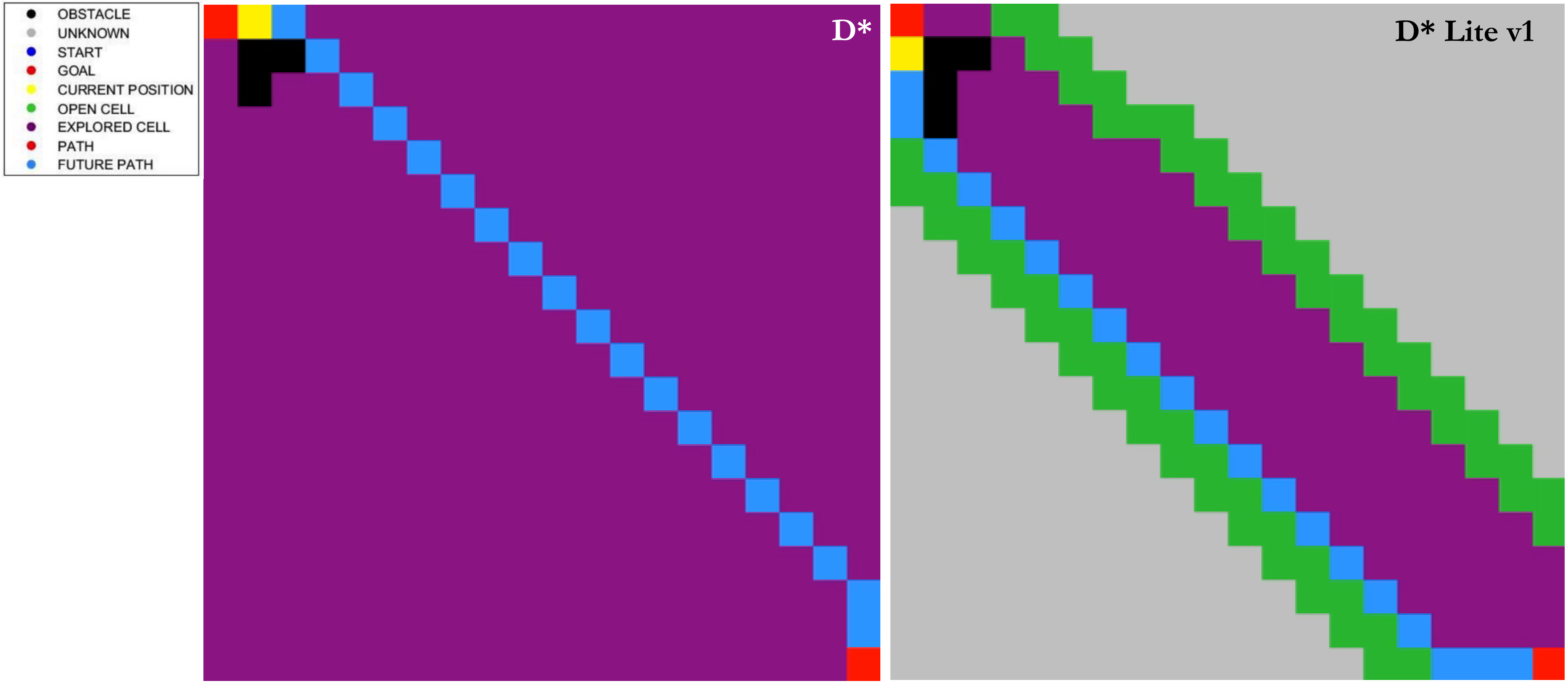


- Start position
- Goal position
- **Obstacles**

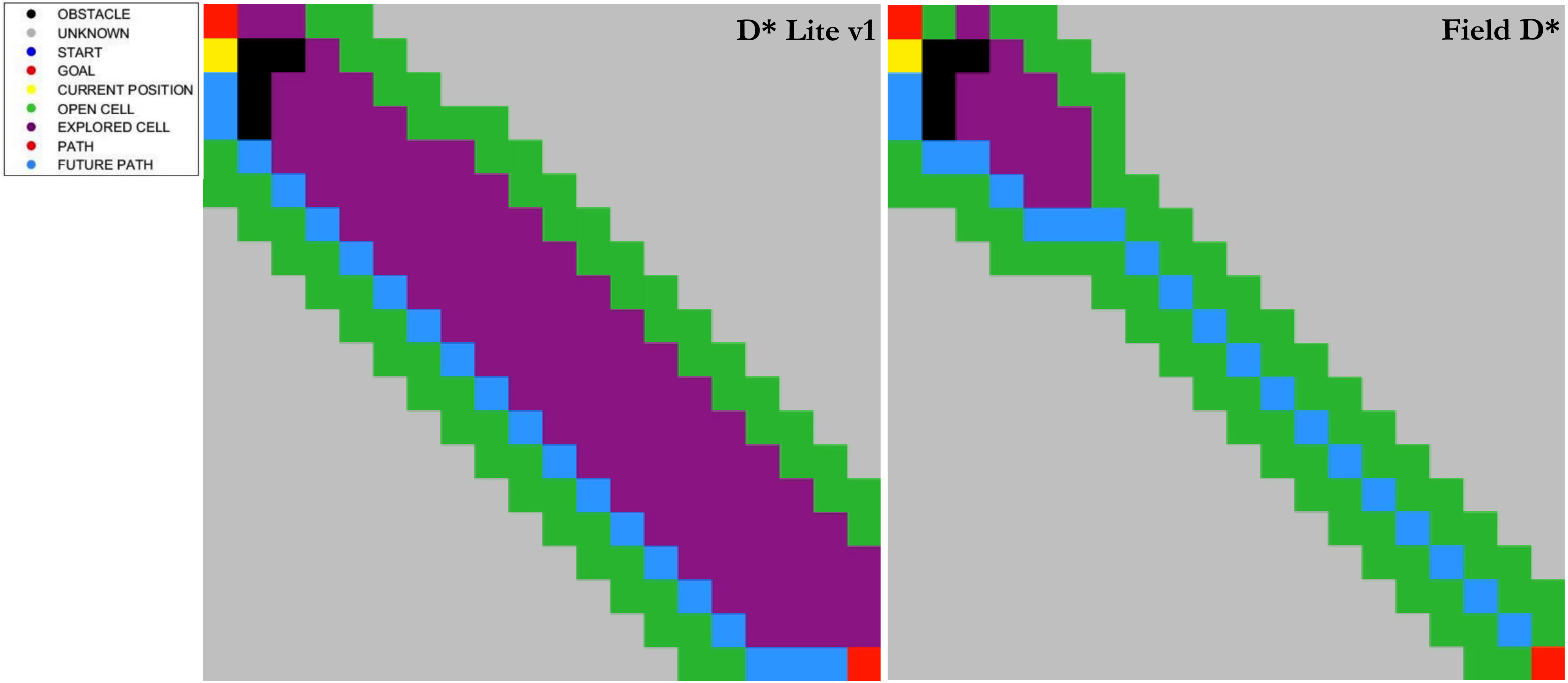
Comparisons done by using this map are:

- D* – D* Lite v1
- D* Lite v1 – Field D*

Video: D* – D* Lite v1



Video: D* Lite v1 – Field D*



3D Simulation with CoppeliaSim

Grid-based path planning for a differential drive robot

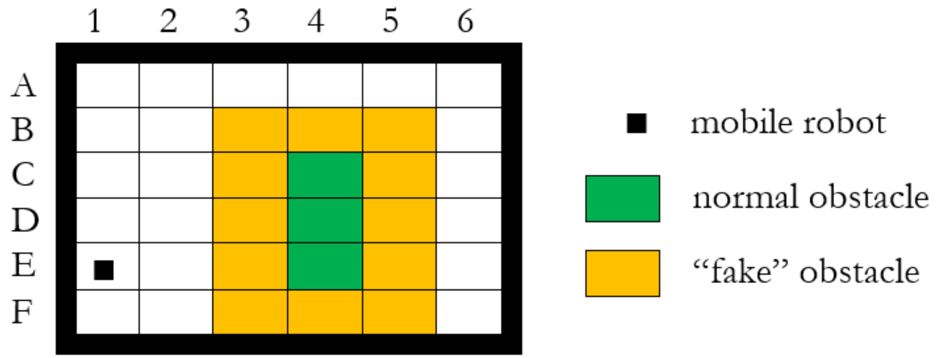
SAPIENZA
UNIVERSITÀ DI ROMA

Assumption for CoppeliaSim

“Actual” obstacles are surrounded by “fake” obstacles

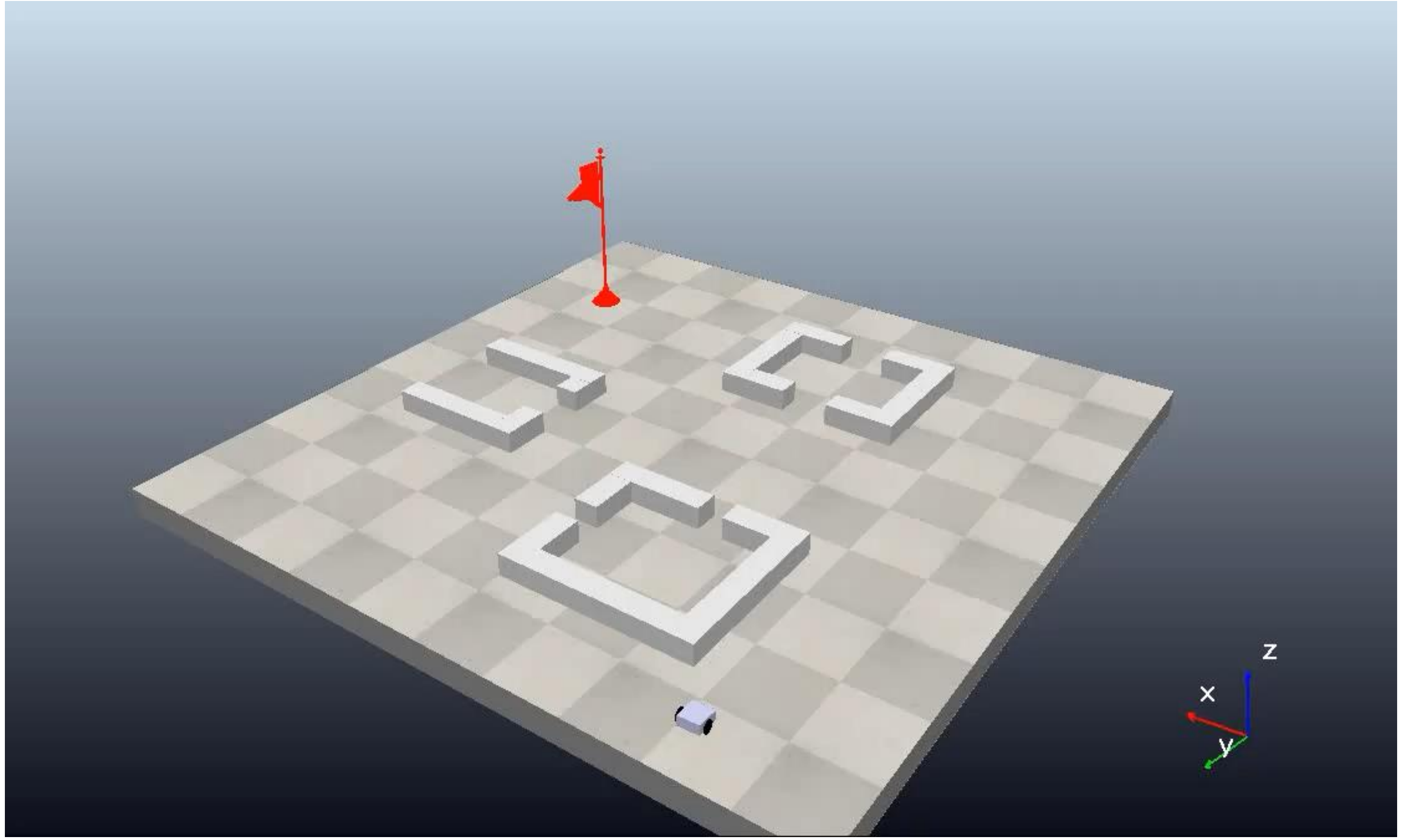
Reasons:

- Safe path
- Avoid collisions



CoppeliaSim environment

- Differential Drive Robot commanded by Matlab through CoppeliaSim RemoteAPI
- Input-Output Linearization controller for inputs
- Map imported by Floor Plan Importer



Conclusion

Grid-based path planning for a differential drive robot

Final considerations

D* produces good solutions but requires an **higher initialization time** (and a **lower replanning time**).
On scale, it could be quite slow.

D* lite reaches about the same solutions of D* but with considerably **less time**.
On scale, it could be a valid solution.

Field D* is a bit worse in time than **D* lite**, but it keeps the promise of making **shorter path in real applications.**

References

- [1] A. Stents, "Optimal and efficient path planning for partially-known environments", ICRA 1994
- [2] S. Koenig and M. Likhachev, " Fast replanning for navigation in unknown terrain", IEEE Transactions on Robotics 2005
- [3] D. Ferguson and A. Stents, "Field D*: an interpolation-based path planner and replanner", Carnegie Mellon Univeristy
- [4] S. Koenig *at al.*, " Lifelong planning A*, " *Artificial Intell.* J., vol 155, 2004

SAPIENZA
UNIVERSITÀ DI ROMA

Thanks for your attention!

Final Project in *Autonomous and Mobile Robotics*
June 2022

Alessandro Lambertini, Denise Landini, Gianluca Lofrumento