S-Theta*: low steering path-planning algorithm

Pablo Muñoz and María D. R-Moreno

Abstract The path-planning problem for autonomous mobile robots has been addressed by classical search techniques such as A* or, more recently, Theta*. However, research usually focuses on reducing the length of the path or the processing time. Applying these advances to autonomous robots may result in the obtained "short" routes being less suitable for the robot locomotion subsystem. That is, in some types of exploration robots, the heading changes can be very costly (i.e. consume a lot of battery) and therefore may be beneficial to slightly increase the length of the path and decrease the number of turns (and thus reduce the battery consumption). In this paper we present a path-planning algorithm called S-Theta* that *smoothes* the turns of the path. This algorithm significantly reduces the heading changes, in both, indoors and outdoors problems as results show, making the algorithm especially suitable for robots whose ability to turn is limited or the associated cost is high.

1 Introduction

Path-planning is focused on finding an obstacle-free path between an initial position and a goal, trying as far as possible that this path is optimal. The path-planning problem representation as a search tree over discretized environments with blocked or unblocked squares has been widely discussed. Algorithms such as A* [1] allow one to quickly find routes at the expense of an artificial restriction of heading changes of $\pi/4$. However, there have been many improvements such as its application to non-uniform costs maps in Field D* [2] or, more recently, Theta* [3] which aims

Pablo Muñoz
Departamento de Automática, Universidad de Alcalá, e-mail: pmunoz@aut.uah.es

María D. R-Moreno
Departamento de Automática, Universidad de Alcaá, e-mail: mdolores@aut.uah.es

to remove the restriction on heading changes that generates A* and gets better path lengths. The main difference between A* and Theta* is that the former only allows that the parent of a node is its predecessor, while in the last, the parent of a node can be any node. This property allows Theta* to find shorter paths with fewer turns compared to A*. However, this improvement implies a higher computational cost due to additional operations to be performed in the expansion nodes process as it will be explained in section 3. Other approximations want to reduce the processing time via heuristics [4] or improving the efficiency of the algorithms [5]. It is worth mentioning that these algorithms work on fully observable environments except Field D*, that can face partially observable environments applying a replanning scheme.

When designing the robot control system, we must take into consideration the morphology of the robot and the application environment. In this paper we focus on the path-planning problem for autonomous mobile robots in open environments with random obstacles as well as indoor areas with corridors and interconnected rooms.

In [6], an extensive comparison between Theta*, A*, Field D* and A*Post Smoothed (A*PS) [7] is performed. A*PS is a variant of A* that is based on smoothing the path obtained by A* at a later stage. As a result we can see that Theta* is the one that usually gets the shortest routes. However, depending on the features of the robot locomotion subsystem, it may be preferable solutions with longer distances but with lower number of heading changes.

Therefore, in this paper we propose the S-Theta* algorithm with an evaluation function that optimizes the heading changes of the route. The field of application is straight forward: robots whose rotation cost is greater than the movement in straight line (what is a vast majority). The paper is organized as follows. Next section shows the notation and the modelization of the terrain used. Then, section 3 describes the Theta* algorithm. The S-Theta* algorithm definition is given in section 4. The results obtained when comparing both algorithms will be shown in section 5. Finally, the conclusions will be outlined.

2 Grid Definition and Notation

The most common terrain discretization in path-planning is a regular grid with blocked and unblocked square cells [8]. For this kind of grids we can find two variants: (i) the center-node (fig. 1 left) in which the mobile element is in the center of the square; and (ii) corner-node (fig. 1 right), where nodes are the vertex of the square. For both cases, a valid path is that starting from the initial node reaches the goal node without crossing a blocked cell. In our experiments we have employed the corner-node approximation, but our algorithm also works with the center-node representation.

A node is represented as a lowercase letter, assuming p a random node and, s and g the start and goal nodes respectively. Each node is defined by its coordinate pair (x,y), being x_p and y_p for the p node. A solution has the form $(p_1, p_2, ..., p_{n-1}, p_n)$

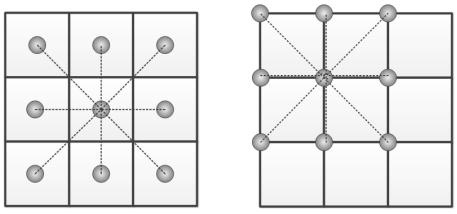

Fig. 1 Possible node representations in grids. Left: center-node; right: corner-node.

with initial node $p_1 = s$ and goal $p_n = g$. As well, we have defined four functions related to nodes: (i) function $succ(p)$ that returns a subset with the visible neighbours of p; (ii) function $parent(p)$ that indicates who is the parent node of p; (iii) $dist(p,t)$ that represents the straight line distance between nodes p and t (calculated through the eq. 1); and (iv) function $angle(p,q,t)$ that gives as a result the angle (in degrees) formed by segments $\overline{pq}$ and $\overline{pt}$ (that is, the angle in opposition of the $\overline{qt}$ segment) of the triangle formed by $\triangle pqt$, in the interval $[0^\circ, 180^\circ)$.

$$\text{dist}(p,t) = \sqrt{(x_t - x_p)^2 + (y_t - y_p)^2} \tag{1}$$

3 Theta* Algorithm

Theta* [3, 6] is a variation of A* for any-angle path-planning on grids. It has been adapted to allow any-angle route, i.e. it is not restricted to artificial headings of $\pi/4$ as is the case of A* with 8 neighbors. Although there are works like [9] that subdivide cells easing this restriction, or others that use more neighbours, but they are ad-hoc solutions. There are two variants for Theta*: Angle-Propagation Theta* [3] and Basic Theta* [6]. We assume that talking about Theta* refers to the last one. The difference between these two versions is how the calculation of the line of sight between pairs of nodes is performed, being in Basic Theta* simpler but with a higher computational complexity in the worst case than the variant Angle-Propagation. Moreover, the latter sometimes indicates that two nodes do not have line of sight when in fact they do, slightly increasing the length of the resulting path.

Theta* shares most of the code of the A* algorithm, being the main block identical (alg. 1) and the only variation is in the `UpdateVertex` function (alg. 2). Thus, all the nodes will store the following values:

- $G(t)$: the cumulative cost to reach the node t from the initial node.
- $H(t)$: the heuristic value, i.e, the estimated distance to the goal node. In the case of A* uses the Octile heuristic, while Theta* uses the Euclidean distance as shown in eq. 1.
- $F(t)$: the node evaluation function expressed by eq. 2.

$$\text{F}(t) = \text{G}(t) + \text{H}(t) \tag{2}$$

- $parent(t)$: the reference to the parent node. For A* we must have $parent(t) = p \Rightarrow t \in succ(p)$, however, Theta* eliminates this restriction and allows that the parent node of t is any node q accessible whenever there is a line of sight between t and q.

Theta* will expand the most promising nodes in the order established in the open list, i.e. it first expands the nodes with lower values for $F(t)$ as shown in alg. 1. In the case that the expanded node is the goal, the algorithm will return the path by traversing the parents pointers backwards from the goal to the start node. If instead the open list is empty, it means that it is impossible to reach the goal node from the initial node and the algorithm will return a failure.

Algorithm 1 Theta* algorithm

```
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 ∈ succ(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
```

When dealing with the successor nodes, the first thing that Theta* does is to check if among the successors of the current position p, being $t \in succ(p)$, and the parent of the current node $q = parent(p)$, there is a line of sight. The high computational cost associated to this checking process is due to the algorithm used (a variant for drawing lines [10] that only uses integer operations for verification) which linearly grows as a function of the distance between the nodes to evaluate. In case of no line of sight, the algorithm behaves as A* and the parent node of t is p. However, if the node t can be reached from q, the algorithm will check whether the node t has already been reached from another node, and then, only update the node t if the cost of reaching it from q is less than the previous cost. In this case, the parent of t will be q and the node is inserted into the open list with the corresponding values

of $G(t)$ and $H(t)$ as shown in alg. 2. Following the described expansion process, Theta* only has heading changes at the edges of the blocked cells.

Algorithm 2 Update vertex function for 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
```

4 S-Theta* Algorithm

The Smooth Theta* (S-Theta*) algorithm that we have developed from Theta*, aims to reduce the amount of heading changes that the robot should perform to reach the goal. To do this, we have based on a modified cost function $F(t)$, as shown in eq. 3.

$$\mathrm{F}(t) = \mathrm{G}(t) + \mathrm{H}(t) + \alpha(t) \tag{3}$$

The new term $\alpha(t)$ gives us a measure of the deviation from the optimal trajectory to achieve the goal as a function of the direction to follow, conditional to traversing a node t. Considering an environment without obstacles, the optimal path between two points is the straight line. Therefore, applying the triangle inequality, any node that does not belong to that line will involve both, a change in the direction and a longer distance. Therefore, this term causes that nodes far away from that line will not be expanded during the search. The definition of $\alpha(t)$ is given in def. 1 and is represented graphically in fig. 2.

Definition 1. $\alpha(t)$ represents the deviation in the trajectory to reach the goal node g through the node t in relation to the straight-line distance between the parent of its predecessor ($t \in succ(p)$ and $parent(p) = q$) and the goal node.

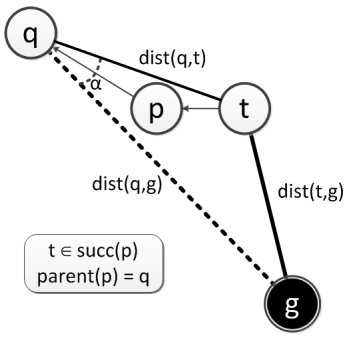

Fig. 2 Graphical representation of α. Actual position is p with $parent(p) = q$. The successor considered is t.

To compute $\alpha(t)$ we have used eq. 4. A priori, it could seem interesting to use the triangle formed by the initial, goal and actual nodes. However, this line stays unchanged during the search, what implies that the algorithm would tend to follow the line connecting the initial node to the goal node. This would cause undesirable heading changes since one could rarely follow that route.

Then, we use the triangle $\triangle qtg$ for $\alpha(t)$ computation, where $q = parent(p)$ and $t \in succ(p)$. The result is that once the initial direction has changed, the algorithm tries to find the new shortest route between the successor to the current position, t, and the goal node. The shortest route will be, if there are no obstacles, the one with $\alpha(t) = 0$, i.e., the route in which the successor of the current node belongs to the line connecting the parent node of the current position and the goal node. Figure 3 shows how the value of $\alpha(t)$ evolves as the search progresses.

$$\alpha(t) = \arccos \frac{dist(q,t)^2 + dist(q,g)^2 - dist(t,g)^2}{2 \cdot dist(q,t) \cdot dist(q,g)} \quad \text{with } t \in succ(p) \text{ and } parent(p) = q \tag{4}$$

The alg. 3 shows the pseudocode of the function `UpdateVertex` for S-Theta*. For computation purposes, $\alpha(t)$ will be included as a cost in the evaluation function of the nodes, so the algorithm will also discriminate the nodes in the open list depending on the orientation of the search. Thus, a node in the open list may be replaced (which means that its parent will be changed) due to a lower value of $\alpha(t)$.

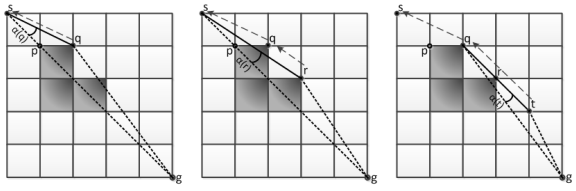

Fig. 3 Representation of the evolution of $\alpha(t)$. Arrows are pointed to the parent of the node after expansion.

In contrast, Theta* updates a node depending on the distance to reach it, regardless of its orientation. As a result, the main difference with respect to Theta* is that S-Theta* can produce heading changes at any point, not only at the vertex of the obstacles as seen in fig. 4.

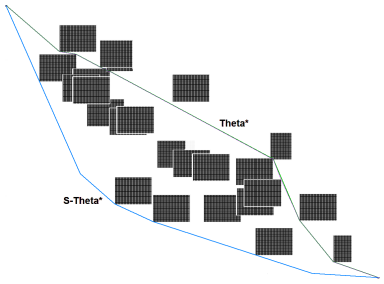

Fig. 4 Solution paths for Theta* and S-Theta* in a random map. Theta* only has heading changes at vertices of blocked cells, while S-Theta* not.

Algorithm 3 Update vertex function for S-Theta*

```
1  UpdateVertex(p, t)
2  α(t) ← angle(parent(p),t,g)
3  if α = 0 or LineOfSight(parent(p),t) then
4      G_aux ← G(parent(p)) + dist(parent(p),t) + α(t)
5      if G_aux < G(t) then
6          G(t) ← G_aux
7          parent(t) ← parent(p)
8          if t ∈ open then
9              open.remove(t)
10         end if
11         open.insert(t,G(t),H(t))
12     end if
13 else
14     G_aux ← G(p) + dist(parent(p),t) + α(t)
15     if G_aux < G(t) then
16         G(t) ← G_aux
17         parent(t) ← p
18         if t ∈ open then
19             open.remove(t)
20         end if
21         open.insert(t,G(t),H(t))
22     end if
23 end if
```

$\alpha(t)$ affects the order of the open list, and thus, how the nodes are expanded. So we need to take into consideration the weight of this term in the evaluation function. If the relative weight in the evaluation function is to small, the algorithm works like the original Theta*, but if it is excessive, it possible that the deal between path-length and heading changes are not fine. $\alpha(t)$ takes values in the interval $[0°, 180°]$. Considering a map with 100x100 nodes, the cost of transversing from one corner to

its opposite corner is $100\sqrt{2} \approx 141$, so the cost added by $\alpha(t)$ is usually less than 90°, a little bit more than the half the cost for traversing among opposite corners of the map. This implies that the angle can grows near the 50% weight in the evaluation function, making that nodes with good $G+H$ and higher $\alpha(t)$ values go back into the open list due to nodes with worse $G+H$ values but with better $\alpha(t)$, and this is that we want to minimize. For this reason we consider that $\alpha(t)$ is well sized in relationship with the $G(t)+H(t)$ values. However, for smaller or bigger maps this shall not be valid. For example, for 50x50 nodes maps, the relative weight of $\alpha(t)$ is double than for a 100x100 nodes map and, for 500x500 nodes maps, is the fifth part. In the first case, the penalization implies an excessive cost to expand nodes that are a little bit far away from the line between s and g, whereas in the second case $\alpha(t)$ has less effect in the search process. Then, the algorithm tends to behave like the original one, that is, both expand a similar number of nodes. In order to compensate this fact, we redefine the value of $\alpha(t)$ as shown in eq. 5, taking into consideration a map with NxN nodes. In the experiments this estimation works fine, so we consider that it is valid.

$$\alpha(t) = \alpha(t) \cdot \frac{\mathrm{N}}{100} \tag{5}$$

4.1 Implementation issues

We suggest two procedures to improve the implementation of the S-Theta* algorithm. The $\alpha(t)$ computation degrades the S-Theta* performance respect Theta* because of both, the cost of floating point operations and the increase of the cost of checking the lines of sight (to make less heading changes, the algorithm needs to check the line of sight for bigger map sections). However, the degradation is not significant in terms of CPU time as we will see in the experimental section. This is thanks to the optimization procedures that will be outlined here and that S-Theta* expands less nodes than the original Theta*.

Procedure 1 The distance between the node t and the goal node ($dist(t,g)$) is static and we can save many operations if once it has been calculated is stored as a property of the node. Before computing the distance between a node and the goal node we will check if this operation has already been performed, so we can save this computation. We only need to initialize this data to a negative value in the instantiation of the nodes prior to the search.

Procedure 2 If $\alpha(t)$ is 0, it means that the node and the predecessor of its parent are in the same line, i.e. the nodes $parent(p)$, p and t are in the same line. Therefore, and given that $t \in succ(p)$ and being the set $succ(p)$ the neighbours reachable from p, it follows that there is a line of sight between t and $parent(p)$. This saves the line of sight checking.

5 Experimental Results

In this section we show the results obtained by comparing the algorithms Theta* and S-Theta*. We also include as a reference the base of both algorithms, A*. The following subsections show the results obtained by running the algorithms on outdoors (random obstacles) maps and indoor maps with interconnected rooms and corridors. For both cases, we have taken into consideration the average values for the following parameters: (i) the length of the path, (ii) the total number of accumulated degrees by the heading changes, (iii) the number of expanded nodes during the search, (iv) the CPU time or search runtime, and, (v) the number of heading changes.

The algorithms are implemented in Java and all of them use the same methods and structures to manage the grid information. All non-integer operations use floating precision. The execution is done on a 2 GHz Intel Core i7 with 4 GB of RAM under Ubuntu 10.10 (64 bits). To measure the runtime we have employed `System.currentTimeMillis()`.

5.1 Outdoors maps

In the outdoors maps we have not exploited the Digital Elevation Model (DEM) to model the terrain. For classical algorithms such as A*, including the height of the points does not make any significant difference when working with the DEM, since we can know the height of all points since the movement is restricted to the vertices. However, in any-angle algorithms, we can traverse a cell at any point. Then, how to calculate or approximate that value is important. In this paper we have not considered the height, although extending the Theta algorithms it is not a difficult task but out of the scope of this paper.

Table 1 shows the results obtained from the generation of 10000 random maps of 500x500 nodes, gradually increasing the percentages of blocked cells to 5%, 10%, 20%, 30% and 40% (each obstacle group has 2000 problems). The way to generate the maps guarantee that there will be at least a valid path from any starting point to the goal. To do that, each time an obstacle is randomly introduced, we force that around the obstacle there are free cells and these free cells cannot overlap with another obstacle. Bold data represent the best values for each parameter measured. In all cases the initial position corresponds to the coordinates (0, 0) and the objective is to reach a node in the last column randomly chosen from the bottom fifth (499, 400-499). Figure 5 shows the last parameter considered to compare the algorithms (not shown in the table) that is the heading changes.

As we can see, the algorithm that obtains the shorter routes is Theta*, however, for the accumulated cost of the heading changes (total spin), the best values are obtained by S-Theta*. The path degradation that occurs in S-Theta* respect Theta* is 3.6% higher for the worse case (40% of blocked cells), while the improvement in the number of heading changes is around 29% in S-Theta*. For the same percentage of obstacles, Theta* performs, in average, 9.555 turns while the number of turns in

Table 1 Experimental results for groups of 2000 random maps of 500x500 nodes

Blocked	**A***	**Theta***	**S-Theta***	**A***	**Theta***	**S-Theta***
	Path length			**Total spin (degrees)**		
5 %	690.942	**674.530**	676.984	340.785	**15.231**	19.878
10 %	697.538	**677.510**	683.368	457.605	38.887	**38.776**
20 %	712.171	**686.965**	699.650	682.290	104.723	**82.101**
30 %	727.090	**697.661**	717.739	898.875	204.670	**148.129**
40 %	744.137	**711.997**	737.732	1151.685	350.847	**249.910**
	Expanded nodes			**Runtime (msec)**		
5 %	18822.062	13193.082	**9257.260**	953.566	1074.314	**829.230**
10 %	23613.764	22844.558	**16265.886**	**1175.865**	1595.634	1271.088
20 %	31897.962	35410.890	**25873.328**	**1439.859**	2027.954	1803.116
30 %	39250.270	44620.867	**33491.780**	**1516.181**	2084.500	2111.310
40 %	43999.836	50526.354	**41109.494**	**1390.862**	1871.207	2375.552

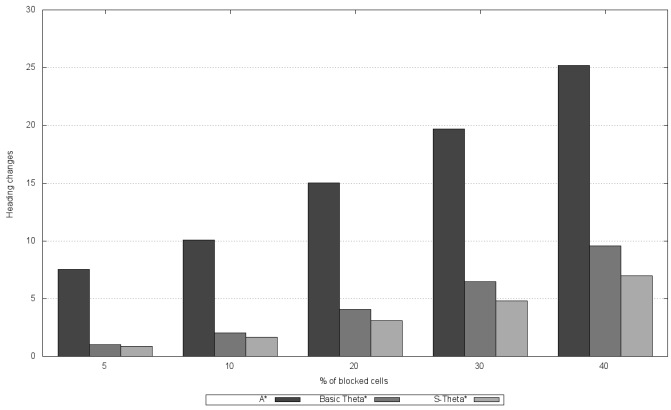

Fig. 5 Average heading changes for groups of 2000 random maps of 500x500 nodes with different number of obstacles.

S-Theta* is reduced to 6.994 (26.8% less). Both algorithms always improve A* in all the parameters except the runtime. For the number of expanded nodes S-Theta* expands near 28% fewer nodes that Theta* except in the case of 40% of blocked cells where the improvement drops to 19%. In the case of time spent in searching the best results are for A*, except for the case of 5% of obstacles, in which S-Theta* gets the best result. However, S-Theta* shows being slightly faster than Theta* with less obstacles, only being surpassed by the original with 30% of obstacles (S-Theta* is only 0.013% slower) and with 40% of obstacles the runtime degradation is remarkable (21% slower). This is due to the increment in the number of expanded nodes: with 40% of blocked cells Theta* expands 5600 nodes more than his execution over maps with 30% of blocked cells, on the other hand, S-Theta* expands close to 7600 nodes more for the same conditions.

5.2 Indoor maps

For indoor maps, we have run the algorithms over 1800 maps with different sizes, 150x150, 300x300 and 450x450 nodes (600 maps per size), always starting from the upper left corner (0, 0) and reaching the target in the opposite corner bottom. The indoor maps are generated from the random combination of 30x30 nodes square patterns that represent different configurations of corridors and rooms. These patterns are designed in a way that we can access to the next pattern through doors on each side, symmetrically placed. Table 2 shows the results for the four comparison criteria, where the best results are highlighted in bold. Figure 6 shows the heading changes using a plot representation.

Table 2 Experimental results for groups of 600 indoor maps with different sizes

Nodes	**A***	**Theta***	**S-Theta***	**A***	**Theta***	**S-Theta***
	Path length			**Total spin (degrees)**		
150x150	247.680	**238.130**	248.425	1151.025	650.224	**584.231**
300x300	497.740	**480.626**	513.876	2074.650	1313.495	**1098.458**
450x450	746.997	**722.412**	780.907	3025.950	1918.551	**1590.245**
	Expanded nodes			**Runtime (msec)**		
150x150	8260.947	8943.435	**4363.208**	312.105	301.440	**270.228**
300x300	36379.942	41096.000	**18203.616**	**1101.347**	1281.430	1191.517
450x450	83993.192	96933.753	**40562.192**	**3666.833**	4530.158	5230.250

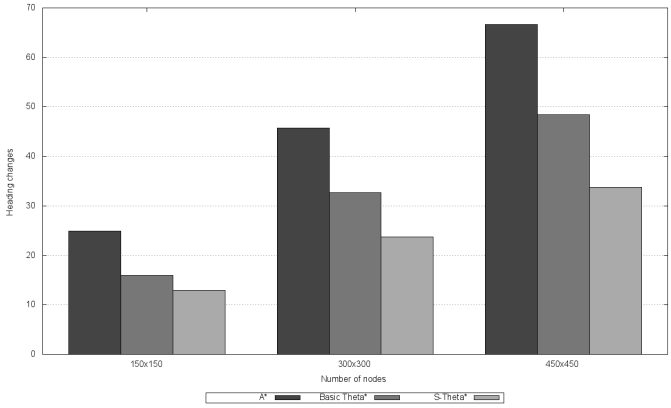

Fig. 6 Heading changes for groups of 600 indoor maps with different sizes.

The data obtained for indoor maps are similar, in general terms, to those obtained in outdoor maps. In all cases the path length is shorter in Theta* than the rest of algo-

rithms, although in this case A* obtains better results than S-Theta*. Furthermore, the path degradation in S-Theta* respect Theta* is 4% higher for the smallest size maps, rising up to 8% in the 450x450 nodes maps. However, for the accumulated cost of the heading changes, the difference is more significant than outdoors maps, getting S-Theta* the best results. For 150x150 nodes maps, S-Theta* uses 10% less turns, whereas in larger size maps this percentage is increased to 17%. In the heading changes there is a clear advantage of S-Theta* above the rest. In bigger maps we can appreciate the difference: Theta* performs on average 48.433 heading changes while this value in S-Theta* drops to 33.736 (about 30% lower). Also, it is even more drastic the reduction in the number of expanded nodes, Theta* expands more than double of nodes that S-Theta*, being S-Theta* who obtains the best results. Finally, the best execution time for small maps is obtained by S-Theta*, but is A* the winner for bigger maps. Comparing S-Theta* and Theta* for 300x300 nodes maps, S-Theta* is 7% faster than S-Theta*, while for 450x450 nodes maps, Theta is 13% faster than S-Theta*.

Finally, fig. 7 shows the sum of the length of the paths, and the total cumulative degree of heading changes for each algorithm. The data are normalized for comparison with Theta*, showing that for the A* algorithm the results are worse than for Theta*, while in the case of S-Theta* always keeps the values below, starting from 0.93 in 150x150 nodes maps and it gets stabilized around 0.89 from 300x300 nodes maps.

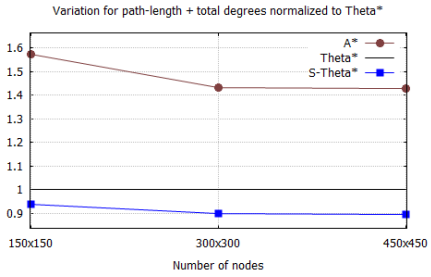

Fig. 7 Normalized values for path length + total spin degrees for indoor maps (normalized to Theta*).

6 Conclusion

In this paper we have presented S-Theta*, an algorithm that is based on Theta*. The main motivation in this work was inspired by the idea that the cost of performing heading changes in real robots is high and it can be desirable to have longer paths with less turns (or at least with smoother turns) than shorter paths with abrupt direction changes. High degree heading changes in real robots are translated into high battery consumption and a lot of time in turning. Based on this idea, we have modified Theta* to compute the path deviation from the optimal path. For that, we have

introduced a new parameter in the evaluation function called $\alpha(t)$, that effectively reduces the number of heading changes required by the search algorithm to achieve the objective, as well as the total cost associated with these turns.

As the experimental results show, the S-Theta* algorithm improves the original algorithm Theta* on the number of heading changes and its accumulate cost, in exchange for a slight degradation on the length of the path. This is true for both outdoors and indoor maps without considering the Digital Elevation Model (DEM) to model rough outdoor terrains. Taking into consideration that an optimal solution is the one that includes both the length of the path and the cost associated for turning, S-Theta* gets better results than Theta*, being small improvements for outdoor maps and more remarkable for indoor maps. In the case of outdoor maps, the best results are obtained with 40% of obstacles, in which S-Theta* gets 8% of improvement. In indoor with maps of 450x450 nodes, the improvement reaches the 10%.

In addition, since S-Theta* expands fewer nodes than the original Theta*, it will require less memory, which is always a plus in embedded systems, typically limited by memory and computation.

Acknowledgements Pablo Muñoz is supported by the European Space Agency (ESA) under the Networking and Partnering Initiative (NPI) *Cooperative systems for autonomous exploration missions*.

References

1. I. Millington and J. Funge, *Artificial Intelligence for Games*, 2nd ed. Morgan Kaufmann Publishers, 2009.
2. D. Ferguson and A. Stentz, "Field D*: An interpolation-based path planner and replanner," in *Proceedings of the International Symposium on Robotics Research (ISRR)*, October 2005.
3. A. Nash, K. Daniel, S. Koenig, and A. Felner, "Theta*: Any-angle path planning on grids," in *In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2007, pp. 1177–1183.
4. P. M. noz and M. D. R-Moreno, "Improving efficiency in any-angle path-planning algorithms," in *6th IEEE International Conference on Intelligent Systems IS'12*, Sofia, Bulgaria, September 2012.
5. S. Choi, J. Y. Lee, and W. Yu, "Fast any-angle path planning on grid maps with non-collision pruning," in *IEEE International Conference on Robotics and Biomimetics*, Tianjin, China, December 2010, pp. 1051–1056.
6. K. Daniel, A. Nash, S. Koenig, and A. Felner, "Theta*: Any-angle path planning on grids," *Journal of Artificial Intelligence Research*, vol. 39, pp. 533–579, 2010.
7. A. Botea, M. Muller, and J. Schaeffer, "Near optimal hierarchical path-finding," *Journal of Game Development*, vol. 1, pp. 1–22, 2004.
8. P. Yap, "Grid-based path-finding," in *Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, vol. 2338. Springer Berlin / Heidelberg, 2002, pp. 44–55.
9. G. Ayorkor, A. Stentz, and M. B. Dias, "Continuous-field path planning with constrained path-dependent state variables," in *ICRA 2008 Workshop on Path Planning on Costmaps*, May 2008.
10. J. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, pp. 25–30, 1965.