Improving Efficiency in Any-Angle Path-Planning Algorithms

Pablo Muñoz and Maria Rodriguez-Moreno
Universidad de Alcalá
Madrid, Spain
Email: {pmunoz, mdolores}@aut.uah.es

***Abstract*—Searching for optimal paths over grids has been widely discussed using search algorithms such as A*. It is an efficient but restricted to artificial heading changes method. Lately, some algorithms have tried to obtain better paths, such as A* Post Smoothed or Theta*. These two variants of A* get an any-angle path avoiding its main limitation but at the expense of an increment in the computational cost. In this paper we propose two contributions. First, we introduce a new parameter that can help us to compare path-planning algorithms under a different view, not only time and expanded nodes. Second, a new heuristic function that allows us to guide the process towards the objective, improving the computational cost of the search. Results show that our algorithm gets better runtime and memory usage than the others.**

***keywords*—Path-planning; expanded nodes; execution time; heading changes;**

I. Introduction

Path-planning is a widely research problem in mobile robotics. The objective is to get an optimal (or near to it) path, avoiding the known obstacles in the terrain. In this paper we assume that the terrain is fully-observable, so we know about all the obstacles in the terrain. Other algorithms, like Field D* [1], use a replanning scheme in order to fast generation of new paths when an unknow obstacle is detected.

Search algorithms as path-planning, are usually embedded into a robot with a low performance integrated computer. Both processor and memory are a bottleneck for a search algorithm that may grow exponentially. But the increase of the search tree is not the only problem for any-angle algorithm. It also needs to perform a line of sight check for each expanded node. This calculation is a little bit expensive, so the CPU requires a lot of time to achieve a solution. Generally, in an uncertain environment, a robot must respond quickly to the changes. Sometimes its integrity depends on how quickly can take a decision. Thus, spending lot of time searching for a solution may not be desirable. Also, less time spent in searching allows the robot to use the remaining time in other important questions (transmitting data, diagnosis, etc.).

Therefore, we have taken this into consideration to guide the search towards the objective and try to minimize the number of expanded nodes, and thus, the processing time and the memory required during the search. To do this, we consider that the shortest path between two nodes is the straight line if there are no obstacles. So, nodes far from this line are not desirable and thus we do not want to expand them. This affects positively in the runtime (less expanded nodes is equal to less line of sight checks) and the memory usage. Other works have tried to speed up the search algorithm using pruning schemes during the line of sight check in order to reduce the spent time in that calculus, for example, applying it to Theta* [2].

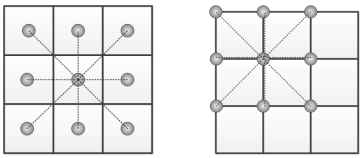
Also, as part of this research in path-planning algorithms, we have tried to caracterize other ways to compare the algorithms. The common parameters are time, expanded nodes and length of the solution. But there is nothing in the literature about how to measure the heading changes. For example, the shortest path may imply long amplitude heading changes what is translated in higher battery consumption. In some cases it can be preferable to have longer paths but with smoother direction changes. In order to measure the "*smoothness*" of the generated path, we introduce a new parameter, called β, that calculates the amplitude of the path's heading changes.

The paper is structured as follows: next, we summarize the terrain discretization and the notation employed. Then, we describe the search algorithms used for the experimental section. Section IV shows how to measure the amplitude of the path's heading changes. In section V, we define a new heuristic function to improve the search in path-planning algorithms. Finally, we present the experimental results and our conclusions.

II. Grid definition and notation

In this paper we have considered the most common terrain discretization in path-planning: a regular grid with blocked and unblocked square cells [3]. For this kind of grids we can find two variants: (i) the center-node (fig. 1(a)) in which the mobile element is in the center of the square; and (ii) corner-node (fig. 1(b)), 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 work can be applied to both.

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 $(p_1, p_2, ..., p_{n-1}, p_n)$ with initial node $p_1 = s$ and goal $p_n = g$. As well, we have defined three functions related to nodes: (i) function $parent(p)$ that indicates who is the parent node of p; (ii) $dist(p, t)$ that represents the straight line distance between nodes p and t



(a) Center-node (b) Corner-node

Fig. 1. Nodes position

(calculated through the eq. 1); and (iii) function $angle(p,t)$ that gives as a result the angle (in degrees) formed by nodes p and t respect to the X axis, in the interval $[0°, 360°)$.

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

III. Search algorithms

In this section, we briefly describe the baseline search algorithms that we have used to improve runtime and memory. The algorithms employed assume fully-observability, but the heuristic that we present in this paper, can be applicable to algorithms that work with partial-observability and replanning schemes.

A. A* and A* Post Smoothed

Some path-planning algorithms are a variation of the A* search algorithm. A* is simple to implement, is very efficient, and has lots of scope for optimization [4]. But it has an important limitation: it typically uses 8 neighbours nodes, so it restricts the path headings to multiples of $\pi/4$, causing that A* generates a sub-optimal path with zig-zag patterns. Other approximations use more adjacent nodes or use framed cells [5] to solve (or relax) this limitation, and thus requiring, in most cases, more computational effort.

For each node, A* maintains three values: (i) the accumulate cost (G(p)), is the length of the shortest path from the start node to p node; (ii) the heuristic value for the node, an estimation of the distance from p to g; and (iii) the parent of the node. The heading changes limitation makes that the best heuristic for A* is the octile distance. The first two values are condensed into F(p) for each node, as shown in eq. 2. Also, A* has two lists to manage the search: (i) the open list, a priority queue of nodes ordered by their F values, and (ii) the closed list, that contains the vertex that A* has already expanded. Algorithm 1 shows the pseudocode of A*. The code for the $UpdateVertex$ function is shown in alg. 2 (only aplies to lines 12-19).

$$\text{F}(p) = \text{H}(p) + \text{G}(p) \quad (2)$$

There are some variations of A* to convert it into an any-angle algorithm [6], [7]. In this paper we use A* Post Smoothed (abbreviated A*PS) algorithm described in [8]. It runs A* and then smoothes the resulting path in a post-processing step. Therefore, the resultant path may be shorter than the original, but it increases the runtime. If A* finds a path $(p_1, p_2, ..., p_n)$, the smooth process checks the line of sight between the first node and the successor node of its successors. For example, taking the initial node $s = p_1$ as the current node, it checks if there is a line of sight between p_1 and p_3. If it is true, the parent of p_3 is now p_1 and thus p_2 is eliminated from the path. The algoritm then takes the next node in the path and checks the line of sight with the current node. If there is not visibility between these two nodes, the last node becomes the current node and the line of sight check continues. The process is repeated until it reaches the goal. So, the resultant path has the same or less nodes than the original one, that is, $n \geq j$, being n the number of nodes in the original path and j the nodes in the PS path.

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
```

B. Theta*

Theta* [9], [10] is a variation of A* for any-angle path-planning on grids. There are two variants for Theta*: Angle-Propagation Theta* [9] and Basic Theta* [10]. We assume that talking about Theta* refers to the last one. Theta* is identical to A* except the $UpdateVertex$ function, so the psedocode for A* shown in alg. 1 applies also to Theta*.

Theta* works like A*PS with an important difference: Theta* does not need a post-processing step, it does the line of sight check during the expansion of nodes. When Theta* expands a node, p, it checks the line of sight between the parent of the node and its eight neighbours. If there is a line of sight between a succesor of p and its parent, then the parent of the succesor is $parent(p)$, not p like A*. When there is an obstacle blocking the line of sight, then Theta* works like A*. For this reason, the parent of a node can be any node, and the path obtained is no restricted to $\pi/4$ headings. The $UpdateVertex$ function pseudocode for Theta* is shown in

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
```

alg. 2. In order to obtain a better result, the heuristic that it employs is the Euclidean distance as seen in eq. 1. As a consequence of the expansion process, Theta* only has heading changes at the corners of the blocked cells.

Paths found by Theta* are shorter than the obtained by A* or A*PS, but is not guaranteed to find true shortest paths. The main problem in Theta* is that the collision test is frequently performed, which degrades significantly its performance.

IV. Measuring the heading changes

In order to compare path-planning algorithms, the length of the resulting path is usually employed as a measure of the optimality of the solution. Besides, there are other parameters such as the expanded nodes or the execution time. However, in the literature we cannot find the number of heading changes (or the associated cost). In the case of a mobile robot, the cost of making a turn can be higher than moving forward half a meter. In order to select a path-planning algorithm for a mobile robot we can take into consideration how this parameter affects the quality of the path. Our initial goal was, precisely, minimize the number of heading changes and its amplitude during the search, but we ended up also minimizing the time and memory thanks to consider this new parameter.

We define the β parameter as the average value of all heading changes (considering that the robot is oriented towards the first node of the resulting path) between the start and goal node. This is formally expresed in eq. 3.

$$\beta = \frac{1}{\text{heading changes}} \cdot \sum_{i=1}^{n-2} \beta_i \tag{3}$$

Each heading change, β_i (eq. 4), is the angle variation produced when the robot goes from node p_i to node p_{i+2} through node p_{i+1}. In other words, β_i is the resultant angle of the intersection of a line that crosses the nodes p_i and p_{i+1}, and the line that crosses the nodes p_{i+1} y p_{i+2}. Also, the involved nodes must have the parent relationship shown in eq. 4. However, we assume that a robot can rotate both to the left and to the right, so, in case of the resultant angle β_i is greater than 180, it must be reduced to obtain an angle in the interval $[0°, 180°]$.

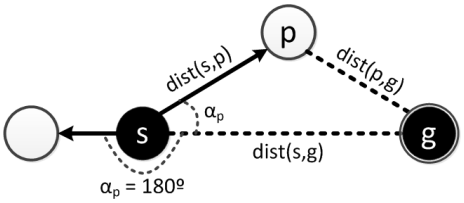


Fig. 2. Graphical representation of α_p

$$\begin{aligned} \beta_i &= |\text{angle}(p_{i+2}, p_{i+1}) - \text{angle}(p_{i+1}, p_i)| \\ \beta_i &= 360 - \beta_i \text{ when } \beta_i > 180 \\ \text{being } p_i &= \text{parent}(p_{i+1}) \text{ and } p_{i+1} = \text{parent}(p_{i+2}) \end{aligned} \tag{4}$$

V. Improving path-planning algorithms

Considering the value of β_i, we propose a new evaluation function, F(p) (eq. 5), instead of eq. 2.

$$\text{F}(p) = \text{H}(p) + \text{G}(p) + \alpha_p(p) \tag{5}$$

α_p aims to expand only the nodes that are near (or are contained) in the straight line that connects the start and the goal nodes. This line is the smallest distance between these two nodes if there are no obstacles blocking the path. For this reason, α_p takes values in the range $[0°, 180°]$, being $0°$ when the node belongs to the line and the search algorithm goes towards the goal node. It takes the middle value, $90°$, when the deviation of the node is perpendicular to the line and, values greater than $90°$ implies that reaching the succesor node increments the distance in straight line to the objective, being the maximun value, $180°$, when the node is in opposite direction to the goal. The formal expression to calculate α_p is shown in eq. 6 and is graphically represented in fig. 2.

$$\alpha_p = \arccos \frac{dist(s,p)^2 + dist(s,g)^2 - dist(p,g)^2}{2 \cdot dist(s,p) \cdot dist(s,g)} \tag{6}$$

Fig. 3 shows the relevant data for two nodes. The α_p values for nodes p and p' are $11.31°$ and $18.43°$ respectively. If we supose that the central cell (corresponds to the square formed by nodes $(2,2)$, $(2,3)$, $(3,2)$ and $(3,3)$) is blocked, A* expands first the nodes located at the top left of the map, expanding the node p', before the node p. But we can see that the predicted any-angle path length (in dot line) has higher β value for node p' than for node p. The difference in the heading change is 17% higher for node p' than node p, in this example. So expanding the node p' is less likely with our

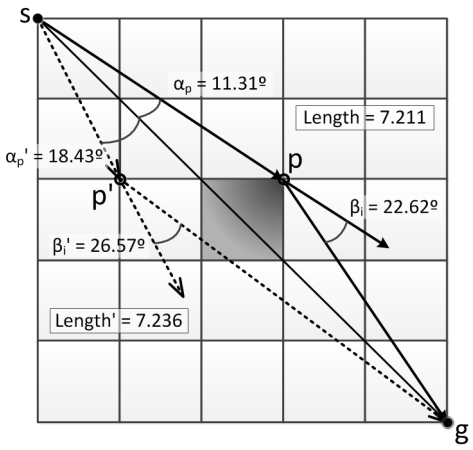


Fig. 3. Example data for two nodes

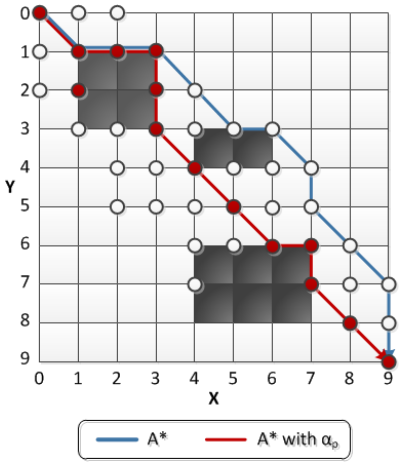


Fig. 4. A* vs A* with α_p

heuristic. Using α_p with the algorithm forces the search to expand first the node p and relegates nodes far to the optimal unblocked path to the back of the open list.

We can get a quick conclusion about α_p if we directly apply it to A*: the search algorithm degenerates into a greedy search algorithm which tries to expand only the nodes that belong to the straight line between the start and goal nodes. In fig. 4 the two obtained routes are represented using A* without α_p (blue line) and with it (red line) starting from the top left node and finishing at the bottom right node. The first one uses the octile distance as a heuristic function, and the last one uses the euclidean distance. Also, the expanded nodes for each search have been marked. Non modified A* expands 41 nodes (empty circles) and A* with α_p expands only 14 nodes (red filled circles). That is, A* with α_p expands near 66% less nodes than the original one. We can also note the tendency of the algorithm to border the obstacles in order to recover the line with $\alpha_p = 0$. Therefore, although is previsible that it reduces the number of vertex expanded (and thus, the runtime), the use of α_p with A* does not imply benefits in terms of path lenght, and, if there is some obstacles blocking the line between the start and the goal node, the number of heading changes could be increased. In the other side, this pseudo-capability to detect obstacles could be usefull for any-angle algorithms, so for these reasons, we will employ this penalization over those kind of path-planning algorithms.

Finally, we must take into consideration that α_p takes values in the interval $[0°, 180°]$. Considering a map with 100x100 nodes, the cost to transverse from one corner to its opposite corner is $100\sqrt{2} \approx 141$, and we can consider that α_p is well sized. However, for smaller or bigger maps this shall not be valid. For example, for 50x50 maps, the relative weight of α_p is double than for a 100x100 map and, for 500x500 nodes maps, is the fifth part. In the first case, the penalization implies an excesive cost to expand nodes that are a little bit far from the line between s and g, whereas in the second case α_p has less effect in the search process, so 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 α_p as show in eq. 7, taking into consideration a map with NxN nodes.

$$\alpha_p = \alpha_p \cdot \frac{\mathrm{N}}{100} \tag{7}$$

A. The c factor

In the previous section we have explained that the relative weight of α_p can modify the behavior of the search algorithm. For this reason, we have considered to multiply its initial value by a factor c, as shown in eq. 8. As this factor increases, the relative weight of α_p over the search algorithm grows up, such that the cost of moving away from the line that connects the start and goal nodes is bigger and force the algorithm to expand less nodes. This means that the c factor is inversely proportional to the number of expanded nodes during the search. It must be taken into consideration that, if $c = 0$, the algorithm does not change its behavior (due to the definition of F(p), eq. 5). So we can modify the behavior of the search algorithm using c as a parameter. In the next section we discuss how the value of α_p affects the different search algorithms employed. To summarize, we can say that a high value of α_p makes the search algorithm greedy, and values near to 0, slightly changes the original behavior.

$$\alpha_p = c \cdot \alpha_p \quad \text{with } c \in (0, 1] \tag{8}$$

VI. Experimental results

In this section we compare A* with A*PS and Theta*, both using and not using α_p. For the experiments with α_p we use 0.25, 0.50 an 1.00 for the c factor. We measure the average of the parameters: path length, search runtime (in miliseconds) and number of expanded nodes. The start node is the south-west corner and the goal node is randomly chosen at the bottom from the column of the east. Due to the algorithm employed in the construction of grids, it is guaranteed that there is at least one valid path from the start node to the goal node. Table I summarizes the results for the execution

TABLE I
EXPERIMENTAL RESULTS FOR 2000 MAPS OF 500X500 NODES

	A*	A*PS	A* PS α_p			Theta*	Theta* α_p		
***c* factor**			**0.25**	**0.50**	**1.00**		**0.25**	**0.50**	**1.00**
Path length									
5 %	690.912	675.789	675.500	**675.425**	675.570	***674.785***	675.007	675.147	675.366
10 %	694.382	677.966	**677.735**	677.859	678.370	***676.560***	677.116	677.494	678.002
20 %	704.985	685.048	**684.983**	685.638	687.233	***682.836***	684.120	685.272	687.007
30 %	722.788	**698.276**	698.395	699.814	703.274	***694.771***	696.874	699.210	703.246
40 %	741.942	**714.814**	715.219	717.882	723.089	***710.213***	713.144	716.988	722.968
Runtime (msec)									
5 %	1301.052	2705.451	1620.204	1179.226	**831.911**	1058.937	650.424	575.510	***526.978***
10 %	1339.988	2771.229	1660.443	1198.165	**851.912**	1182.092	724.100	633.097	***565.276***
20 %	1504.586	2852.795	1859.624	1335.486	**964.053**	1585.026	997.942	831.679	***712.424***
30 %	1539.944	2532.663	1911.298	1389.170	**1029.367**	1863.170	1284.100	1032.206	***849.470***
40 %	1417.973	2051.286	1756.433	1314.728	**979.215**	1760.983	1351.566	1073.788	***855.140***
Expanded nodes									
5 %	22762.980	38468.874	23262.296	17323.768	**11483.968**	13861.442	7778.152	6743.274	***6048.988***
10 %	24448.918	41727.039	24995.514	18541.500	**12632.822**	17331.226	9732.340	8297.594	***7273.318***
20 %	30736.026	51079.580	31404.531	23223.675	**16477.058**	28007.257	16296.966	13330.642	***11104.778***
30 %	38685.568	60854.500	39607.290	29049.040	**21138.685**	41403.018	25242.750	19872.176	***15802.969***
40 %	44400.196	65582.608	45508.286	33101.610	**23778.130**	49242.972	31567.148	24334.391	***18469.732***

of the algorithm over 2000 maps of 500x500 nodes with differents number of blocked cells (5%, 10%, 20%, 30% and 40%). Best value for both algorithms, A*PS and Theta* is highlighted in bold. Also, we use cursive to remark the best value of all algorithms. We perform the same tests for grids of 100x100 and 1000x1000 nodes (results not shown) getting similar results to the exposed here.

The algorithms are implemented in Java and all of them use the same methods and structures to manage the grid information. 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()`.

We are going to compare the results obtained for A*PS and Theta* with and without α_p. First, we take into consideration the path length. So let us start with A*PS.

In terms of path length, the use of α_p has little effect, with less than 20% of blocked cells and with low α_p values (that is, $c = 0.25$). The best cases correspond to few blocked cells and low values, getting little shorter paths (almost imperceptible) than the original algorihtm. The worst case, 40% blocked cells and $c = 1.00$, gets a 1.16% longer paths. The runtime using α_p is always better, decreasing between 1/3 (less blocked cells) and 1/2 (more blocked cells) when $c = 1.00$. This can be explained due to the number of expanded nodes: A*PS with α_p expands less nodes as you increase the c factor. Also, we can observe that A*PS with α_p and $c \geq 0.50$ decreases both the path length and nodes expanded (and thus, the runtime) compared to A*.

For Theta* using α_p the results are similar to the obtained with the modified A*PS, but the degradation of the path length is more notable. For example, the path lenght worst case (40% blocked cells and $c = 1.00$), gets 1.83% longer paths than the original Theta*. The runtime is always lower using α_p, but with $c = 0.25$ it needs near 65% of the original time to get a solution, and with $c = 1.00$ the speedup is near to 50%, so in A*PS using α_p works better in terms of runtime. Same as above, Theta* with α_p and $c = 0.25$ achieves better runtimes than A*. For expanded nodes the behaviour is the same as in A*PS; with $c = 0.50$ it expands near the half of nodes than Theta* and with $c = 1.00$ it decreases to the third part.

Related to the number of heading changes we do not show the average of the β parameter due to the high dispersion of the results. Instead, in table II we show the number of solutions with lower β values for the original path-planning algorithms (A*PS or Theta*), for the algorithm using α_p (with differents c values), and the number of solutions in wich β has the same value for both algorithms (that is, with and without α_p). For A*PS with α_p we obtain good results for maps with less than 30% blocked cells. For example, with 10% of blocked cells and $c = 0.50$ we have the same β value for near half of the maps and better values than the original algorithm in about 1/4 of the maps. That is, in 3/4 of the maps the modification has no effect in this parameter or has positive effects. With more blocked cells, the original algorithm has better β values, but always we can found maps in which α_p has positive impact in the path-planning algorithm. For Theta* we can say that is less likely to find a path with lower β values using α_p, but for less than 20% blocked cells in near half of the cases we can found the same value than Theta*.

TABLE II
SOLUTIONS WITH BETTER β VALUE OVER 2000 MAPS OF 500X500 NODES

5% blocked cells							
c factor	**0.25**	**0.50**	**1.00**	***c factor***	**0.25**	**0.50**	**1.00**
A*PS	164	328	411	**Theta***	312	354	450
A*PS α_p	332	627	663	**Theta* α_p**	114	150	173
Equals	1504	1045	926	**Equals**	1574	1496	1422
10% blocked cells							
c factor	**0.25**	**0.50**	**1.00**	***c factor***	**0.25**	**0.50**	**1.00**
A*PS	301	572	689	**Theta***	518	601	622
A*PS α_p	344	528	529	**Theta* α_p**	156	200	239
Equals	1355	900	782	**Equals**	930	1199	1099
20% blocked cells							
c factor	**0.25**	**0.50**	**1.00**	***c factor***	**0.25**	**0.50**	**1.00**
A*PS	492	881	1073	**Theta***	815	938	1044
A*PS α_p	433	513	427	**Theta* α_p**	255	296	318
Equals	1075	606	500	**Equals**	930	766	638
30% blocked cells							
c factor	**0.25**	**0.50**	**1.00**	***c factor***	**0.25**	**0.50**	**1.00**
A*PS	770	1207	1438	**Theta***	1074	1233	1351
A*PS α_p	556	558	428	**Theta* α_p**	408	428	427
Equals	674	235	134	**Equals**	568	339	222
40% blocked cells							
c factor	**0.25**	**0.50**	**1.00**	***c factor***	**0.25**	**0.50**	**1.00**
A*PS	921	1422	1632	**Theta***	1236	1439	1525
A*PS α_p	629	515	354	**Theta* α_p**	408	421	417
Equals	450	63	14	**Equals**	356	140	58

VII. CONCLUSION

In this paper we have presented an independent domain heuristic that can be applied to any path-planning algorithm, although we have shown the results in A*, A*PS and Theta*. As a conclusion, we can say that using α_p in A*PS achieves better results in terms of number of expanded nodes (and thus, runtime) but degradates the path length proportionally to both c factor and the number of blocked cells. However, for small c values the degradation is not very high and, using $c = 0.50$ the results show that A*PS gets better results in all parameters than A* with less runtime. The same can be applied to Theta*, taking into consideration that the degradation is more remarkable.

We consider that α_p has three advantages: first, it is easy to implement, and is valid for any path-planning algorithm based in A*; second, we can modify the behaviour of the α_p through the c factor to deal between the runtime and the degradation of the other parameters. If the only parameters to consider are the path-length and runtime, using high c values can boost up three times the runtime of A*PS and two times for Theta*, with a little bit longer paths than the original algorithms. Finally, α_p affects to the memory required because higher values cause expanding less nodes, and thus, less memory employed during the search.

Applying these modified algorithms in a robot imply less time spent in searching, and thus, less battery energy required. Also for the same grid size we need less memory. But we need to consider that the best values of these two parameters involve degradation of the other parameters. So we need to deal between the weight of α_p and the specifications of the robot. For example, if heading changes have low cost, we can use A*PS with $c \approx 0.50$ and the search speeds up near two times with a small degradation of the path length (around 1%).

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] I. Millington and J. Funge, *Artificial Intelligence for Games*, 2nd ed. Morgan Kaufmann Publishers, 2009.

[5] 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.

[6] C. E. Thorpe and L. H. Matthies, "Path relaxation: Path planning for a mobile robot," in *OCEANS Conference*, September 1984, pp. 576–581.

[7] M. Kanehara, S. Kagami, J. Kuffner, S. Thompson, and H. Mizoguhi, "Path shortening and smoothing of grid-based path planning with consideration of obstacles," in *IEEE International Conference on Systems, Man and Cybernetics, ISIC.*, October 2007, pp. 991–996.

[8] A. Botea, M. Muller, and J. Schaeffer, "Near optimal hierarchical path-finding," *Journal of Game Development*, vol. 1, pp. 1–22, 2004.

[9] 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.

[10] 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.