J Braz Comput Soc (2013) 19:167–179
DOI 10.1007/s13173-012-0091-5

ORIGINAL PAPER

# A set of novel modifications to improve algorithms from the A* family applied in mobile robotics

**Tiago Pereira do Nascimento · Pedro Costa · Paulo G. Costa · António Paulo Moreira · André Gustavo Scolari Conceição**

Received: 8 May 2012 / Accepted: 11 October 2012 / Published online: 1 November 2012


**Abstract** This paper presents a set of novel modifications that can be applied to any grid-based path planning algorithm from the A* family used in mobile robotics. Five modifications are presented regarding the way the robot sees an obstacle and its target to plan the robot's path. The modifications make it possible for the robot to get to the target faster than traditional algorithms, as well as to avoid obstacles that move as fast as (or even faster than) the robot. Some simulations were made using a crowded and highly dynamic environment with twelve randomly moving obstacles. In these first simulations, a middle sized 5DPO robot was used. Also, real experiments were made with a small-sized version of a 5DPO robot to validate the algorithm's effectiveness. In all simulations and real robot experiments the objects are considered to be moving at a constant speed. Finally, we present an overall discussion and conclusion of this paper.

**Keywords** Path planning · Mobile robot · Obstacle avoidance · Dynamic environment

T. P. do Nascimento (✉) · P. Costa · P. G. Costa · A. P. Moreira
INESC TEC, Faculty of Engineering, University of Porto,
Rua Dr. Roberto Frias, s/n 4200-465 Porto, Portugal
e-mail: tpnascimento@gmail.com; tiagopn@ieee.org

P. Costa
e-mail: pedrogc@fe.up.pt

P. G. Costa
e-mail: paco@fe.up.pt

A. P. Moreira
e-mail: amoreira@fe.up.pt

A. G. S. Conceição
Department of Electrical Engineering,
Federal University of Bahia, Salvador, BA, Brazil
e-mail: andre.gustavo@ufba.br

## 1 Introduction

Path-planning algorithms constitute a well-known area of research in mobile robotics. Studies in this area may involve single robot movement, or a group of mobile robots moving in a specific formation. Issues like static obstacle avoidance or mobile obstacle avoidance, known or unknown worlds, structured or unstructured environments and single or multiple robot motion are the main study cases in path planning. In this paper, a set of novel modifications conceived to improve grid-based algorithms from the A* family applied in mobile robotics is presented. For instance, a simple target for the robot to reach was considered.

Motion-planning algorithms are currently widely used. Path-planning algorithms can be the solution for many motion-planning issues, such as UAV path planning [1], in mobile robot outdoor navigation [19], in mobile robot indoor navigation [9] and even in video games [17]. In this work, an indoor environment for mobile robot path planning is used. With a preset target, the robot is supposed to avoid obstacles moving at high velocities. The robots used are the omnidirectional robots used in the Middle Size and Small Size Leagues (see Fig. 1) from the 5DPO Team of robot soccer championships (RoboCup). A good modeling and control for these platforms can be found in [8] and [23], respectively.

Many path planning techniques have emerged over the years. One of the most famous is the artificial potential field approach. This methodology has been widely used, and it states that the collision-free trajectory is generated along the negative gradient of the defined attractive and repulsive potential-field functions. The subsequent studies can be found in [18,25,36]. Nonetheless, the potential-field method is not straightforwardly applicable to mobile vehicles with kinematic constraints since, in the potential-field design, the robot is usually treated as a simple particle. Another major

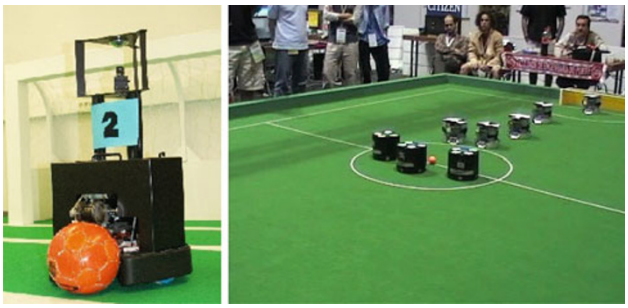


**Fig. 1** The 5DPO middle size robot

problem has to do with the fact that it is essentially a fastest-descent optimization method, and thus can get trapped into local minima of the potential function rather than reach the goal state [20].

Over the years, solutions for motion planning problems were also found in artificial intelligence algorithms, such as neural networks and fuzzy logic. In early years, the use of fuzzy logic was an option for easy-to-control systems [29,32]. Recently, new neural network approaches appeared, showing considerable results. In [21], the authors propose a neural- network-based path planner used in multiple nonholonomic mobile robots with moving obstacles. Other authors have used the neural network approach for non-moving obstacle avoidance [28].

As in artificial intelligence, researchers became aware of some new approaches throughout the years, such as time-optimal approaches [2] and the Dynamic Window Approach [24], which perform well in situations where there are no moving obstacles, despite the computational cost at high velocities. The works of [16] and [35] also applied an optimization method. The first approach is only applied for static obstacles.

Approaches like the Distance-Propagating Dynamic System (DPDS) [34] and the Bug algorithm [15] are recent solutions for the problem of moving-obstacle avoidance. Nevertheless, in [34] the solution causes the robots to move very slowly, while in [15] only a few obstacles are taken into account. In [3] a reactive approach is introduced, while in [4] the reactive method presented is only applied for a straight line. Sonar-based methods can also be seen as reactive methods. In [30], a sonar- based method is well applied, even though it only takes static obstacles into consideration. Completing the set of new approaches that appeared over the last decade, the boundary-following method was introduced by [14] and applied to static obstacles.

Also among the most famous is the Roadmap method. This method can be seen in [5]. Here, a computational geometry data structure was proposed to solve the problem of an optimal path generation between a source and a destination, in the presence of simple disjoint polygonal obstacles. In [27], the Roadmap method is applied successfully using multiple mobile robots in a common environment. Underground mining and the warehouse management problem are considered, even though no randomly moving obstacles are considered. The Roadmap method is successfully applied in low-dimension configuration spaces and sometimes, depending on the approach, it is not easy to implement [20].

Finally, the last method among the most traditional algorithms for path planning is the cell decomposition method [20]. In this category, algorithms such as A*, D*, ARA* and AD* are well known and efficient. The A* algorithm is the oldest, and it has been successfully applied with static [37] and dynamic obstacles [9]. Currently, the main advantage of the Cell Decomposition methods is that, with current technology, they are no longer apply only to indoor environments or small spaces. They can be also applied in UAV obstacle avoidance [1] and in unknown environments [19]. In [7], an approximate cell-decomposition method was developed in which obstacles, targets, sensor platforms and FOV (Field of View) are represented as closed and bounded subsets of a Euclidean workspace. A good overview of the advantages and disadvantages of using these algorithms can be seen in [6,11].

One of the methods that has evolved in recent years is the *Velocity Obstacles* method, first used in [13]. This method defines the set of all the velocities of a robot that will result in a collision at some point in time, assuming that the obstacle maintains the current speed. Therefore, its movement planning aims at finding the speeds that fall outside these groups to ensure that there will not be collisions. This method is widely used in simulations of crowds, having however a minor problem when dealing with static obstacles: the robot circumvents the edges of the obstacles, making the robot slower, as noted in [33].

Therefore, an approach based on algorithms from the A* family for highly dynamic and crowded environments, as well as the modifications for the grid-based path planning algorithm, are presented in the next section. The problem is formulated and results are presented with experiments and simulations in Sect. 3. Finally, conclusions are presented in Sect. 4.

## 2 Path planning algorithms

In robotics, the path-planning task consists of finding a sequence of actions that cause an agent to move from an initial state (position and orientation) to a final state (position and orientation). In path planning, each transition between states represents actions the agent can make, each associated with a cost. A path is said to be *optimal* if the sum of its transition costs is minimal across all possible paths from an initial state $q_{\text{init}}$ to a goal (final) state $q_{\text{goal}}$. A planning

algorithm is said to be *complete* if it always finds a path in a finite amount of time when such a path exists. It can be said that a planning algorithm is optimal if it always finds an optimal path. The proposed modifications can be applied to any of these algorithms (A*, D* and its evolutions, such as D*-Lite and E*, ARA* and AD*) to achieve a faster solution. This affirmation is based on the fact that the differences between these algorithms are in the optimization process, always aiming at a shorter processing time and lower use of resources, such as computational memory. Therefore, in the following subtopics, an overview of grid-based algorithms will be presented.

Furthermore, the cell decomposition algorithms such as D* (and its evolutions such as D*-Lite and E*), ARA* and AD* are based in the A* and were developed to solve problems of computational cost, processing time, or memory expenditure. The modifications proposed here are in the configuration space and not in the algorithm core itself. Therefore, in the matter of configuration space, all the previous algorithms from the A* family should give an equal or similar solution to the A* algorithm. When applying our modification to any algorithm from the A* family, the final solution would be better, as it will be demonstrated with A* in this paper.

Finally, in our approach we base the modifications on the method of cell decomposition, where the modifications are not in the A* algorithm, but in the configuration space to later run an A* algorithm to find the best path. The advantage comes with the fact that, regarding the configuration space, in the cell decomposition there are no local minima, such as in potential functions, while in the VFH or in other similar approaches the local minima can become a problem when avoiding narrow areas. The only exception is when another robot that is trying to block the robot's path is faster than the robot. However, this case would create local minima in any approach.

## 2.1 Grid-based algorithms

### *2.1.1 A* algorithm*

A* is a traditional graph search algorithm that has been developed to calculate least-cost paths on a weighted graph. This algorithm uses a heuristic function,

$$F(n) = g(n) + h(n) \tag{1}$$

which estimates the lowest cost of going from the initial state to the goal state, while going through node $n$. This sets the order in which nodes are sought in order to find the best path as soon as possible. This function is the sum of two other functions:

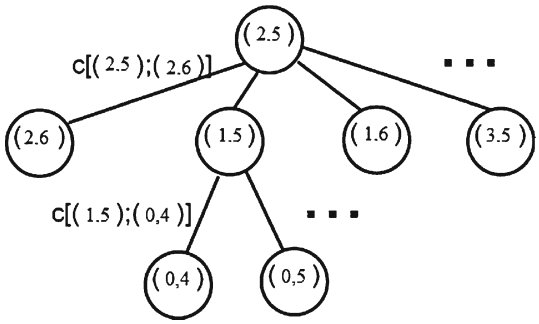


**Fig. 2** Resulting graph from the cell division

1. $g(n) =$ Cost from the initial node to node $n$;
2. $h(n) =$ A heuristic function to estimate the cost of the path from node $n$ to the target node.

There are two lists in this algorithm: the O-list and the C-list. The open list, known as the O-list, contains the nodes that are candidates for exploration. The closed list, known as the C-list, contains the nodes that have already been explored. The nodes from the C-list were previously on the O-list, but as they are explored they are moved to the C-list. The nodes on these lists store the *father* node, which is the node used to optimally reach them. This is the node that lies in the shortest path from the original to the current node. If the heuristic function is *admissible*, then the path cost of $q_{\text{goal}}$ is guaranteed to be optimal.

To use the A* algorithm in the calculation of the robot's path, it is necessary to divide the environment map into cells, as stated in the approximate cell decomposition method. To increase the set of applications of this algorithm, this division could be achieved with a GPS, omnidirectional cameras, global cameras (for outdoor applications when the grid moves with the robot) or previous measurements of the environment, allowing the robotic system to apply this algorithm to almost any kind of situation. Here, each cell represents a node. Each node can be connected to other nodes, and moving from one node to the other has an associated cost (Fig. 2). In this case, the cost is the metric distance between the cell centers. The A* algorithm can calculate the path that minimizes the cost from moving from the initial cell to the target cell. In Fig. 3, the black cells represent the obstacles, the yellow cell represents the initial position (node) and the blue cell represents the destination point (node).

Finally, the robot is represented by the initial node, and it occupies only a single node. This last node is the geometric center of the objects. The destination node is the goal state $q_{\text{goal}}$. All other moving objects are considered to be obstacles. As the robot is represented by a single cell, the obstacles have to be bigger, so as to represent both the obstacle and the robot's body. These obstacles are represented by circles with their radius equal to the sum of the obstacle's radius and the robot's radius. This representation is depicted in the Fig. 4.

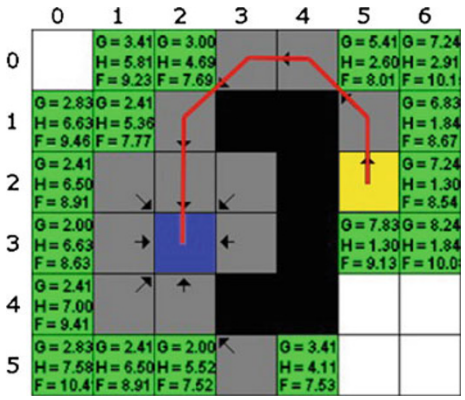


**Fig. 3** Map cell decomposition

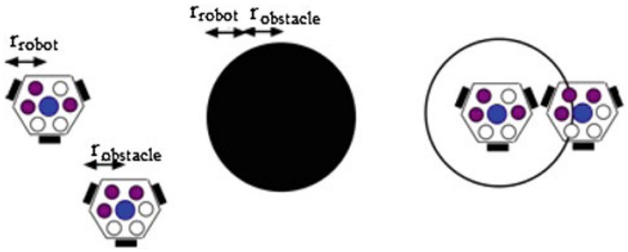


**Fig. 4** Obstacle's total radius

In robotics, it is often important for the agent to keep planning new paths when new information on the environment is received by the sensors. The A* algorithm continuously plans the path from scratch after new information is received. However, it is very computationally expensive to keep planning a path from scratch every time the graph changes. Instead, it may be far more efficient to take the previous solution and repair it.

### *2.1.2 D* algorithm*

The Focused Dynamic A* (also called D*) and D*-Lite have been used for path planning in a large number of robotic systems, including indoor and outdoor platforms. D* and D*-Lite are extensions of A*. Nevertheless, D*-Lite is much simpler and slightly more efficient than D* in some navigation tasks. The D*-Lite proceeds initially similarly to A*, creating an optimal solution path from the initial state to the goal state, in exactly the same manner as A*. The difference is that when the replanning is necessary, the previously planned path is used instead of planning a path from scratch. This saves computational time and can be up to two orders of magnitude more efficient than planning a path from scratch using A* [11].

Generally, D* is very effective for replanning in the context of mobile-robot navigation. In such scenarios, the changes to the graph occur closely to the robot, which means that the effects are usually limited. However, if the areas of the graph being changed are not close to the position of the robot, it is possible that D* is less efficient than A*. This is due to the fact that D* processes every state in the environment twice. The worst-case scenario is when changes are made to the graph in the vicinity of the goal, which happens frequently in a highly complex environment. If the planning problem has changed sufficiently upon the generation of the previous result (a common characteristic of a highly dynamic environment, as in this study case), this result may be a burden rather than a useful starting point. In this case, which is mostly common in real experiments containing uncertainties, A* is much more efficient than D* [11]. Finally, there are some variations of the D* algorithm, such as E*, which makes the path smoother but still suffers the drawbacks of D*, similar to what happens when highly dynamic and complex environments [6] are considered.

### *2.1.3 ARA* algorithm*

In some cases, the reaction of the agent must be quick, and therefore the replanning problem is complex, even in static environments. In such cases, computing optimal paths as described above can be infeasible due to the sheer number of states that need to be processed in order to obtain such paths. Algorithms often construct an initial highly suboptimal solution very quickly, thus improving the quality of the solution afterwards while time permits. One of the most common algorithms is the Anytime Repairing A* (ARA*), which limits the processing performed during each search by considering only those states whose costs at the previous search may not be valid given a new $k$ value (current heuristic parameter of optimality). This improves the efficiency of each state in two ways: by expanding each state at least once when a solution is reached, and by only reconsidering states from the previous search that were inconsistent [11].

However, because ARA* is an anytime algorithm, it is only applicable in static planning domains. If too many changes are being made to the planning graph (which is the biggest characteristic of a highly dynamic environment with moving uncertainties), ARA* is unable to reuse its previous search results and therefore must plan the path from scratch again, which makes A* far more applicable. As a result, it is not appropriate for dynamic planning problems [11]. Therefore, another class of algorithms were created to fix this problem, the Anytime Dynamic A* (also called AD*).

### *2.1.4 AD* algorithms*

Algorithms that plan the path iteratively (A* and D*) have concentrated on finding a single and usually optimal solution, and anytime algorithms (ARA*) have concentrated on static environments. However, some of the most interesting real-world problems are those that are both highly dynamic

(requiring replanning) and highly complex (requiring anytime approaches). The authors in [22] developed the Anytime Dynamic A* (AD*), an algorithm that combines the continuously planning capability of D* Lite with the anytime performance of ARA*. Unfortunately, as the authors put it in [11], this AD* algorithm suffers from the drawbacks of both anytime and replanning algorithms. As with replanning algorithms, AD* can be much more computationally expensive than planning from scratch. The larger the change in the environment, the more time consuming it is to redo planning a path with AD*. This becomes a problem in an environment with many movable uncertainties (moving obstacles). In such cases, A* will also be less time consuming than AD*.

Note here that the following experiments and simulations are highly complex (which becomes an issue for replanning algorithms), highly dynamic (which becomes an issue for anytime repairing algorithms), and full of moving uncertainties, sometimes faster than the robot itself, which makes the AD* computationally expensive. Note also that all these new algorithms only give specific solutions, always with drawbacks, when all problems are considered at the same time, something that is often seen in real-world situations such as in airport daily patrols. To solve these problems, a set of novel modifications based on the A* family algorithms was proposed.

## 2.2 The modifications

As mentioned before, it is known that most environments are highly dynamic, highly complex and contain obstacles moving randomly. The situation studied is often common in the real world considering the dynamic constrains of the robot, which is to find the fastest solution between the initial state $q_{\text{init}}$ and the goal state $q_{\text{target}}$, avoiding as many collisions as possible. Therefore, one of the concepts that it is necessary to highlight is that the best solution, in most cases, is given not by the shortest path (optimal path) and can lead to undesired collisions. In another words, the best solution is not the shortest path (the optimal one), but the fastest path (usually the suboptimal one). That is because the velocity of the robot is not constant (the robot has limited acceleration) and the robot controller has difficulty in following trajectories with abrupt changes in direction.

The first thing to take into consideration when analyzing the proposed modifications is that all contributions should be disregarded and a different angle of analysis should be pursued. The first point of analysis is that the built cell map must have the location of the obstacles in the workspace, in a fixed position. This should be known at the instant the information is captured. This information ignores the velocities of the obstacles. In dynamic environments this can be a big mistake, for it does not allow the robot to avoid obstacles sooner than expected, thus leading to an unwanted collision.

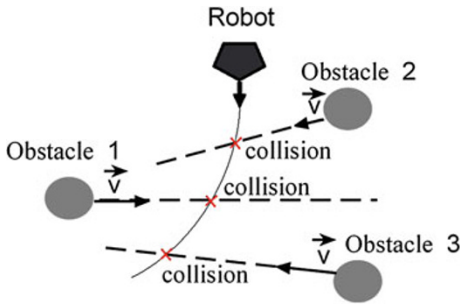


**Fig. 5** Normal collision points

One way to avoid this situation in a cell-based map is to calculate the possible point of collision given the current velocity of the robot and the current velocities of the moving obstacles. In each trajectory calculation the modified algorithm makes the robot assume that the obstacles have constant velocities in the time **t** of data acquisition, and that the robot has a maximum speed and a maximum acceleration. Using this information, the position of new obstacles for path planning calculations is no longer the current position, but the possible collision point, as seen in Fig. 5.

In this case, while the trajectory must be fully planned, only the first steps are taken into account before new information arrives and a new calculation is performed. There are some proposed techniques presented in this paper that can be applied to any A* family algorithm and that are useful when trying to approximate the inherent environment dynamics in a static map. They are called:

1. Obstacle Distance
2. Obstacle Slack
3. Obstacle Direction
4. Processing Time
5. Target Orientation

Finally, it is important to remember that A* family algorithms either find a solution or not. Although the A* and the A* with k can give a sub-optimal solution, there is no mathematical guarantee that there will be no local minima due to the high nonlinearity of the system. Our guarantee is given by the hundreds of hours of using this modified algorithm in the Small Size Robot Soccer League of RoboCup since 2005 with the 5DPO team from University of Porto.

### *2.2.1 Obstacle distance*

This change causes the obstacle to lose relative importance as the distance from the robot increases and the possible collision point is further away from the robot. This can be seen in Figs. 6 and 7. A distant obstacle mostly does not affect the immediate trajectory points. That can speed up the

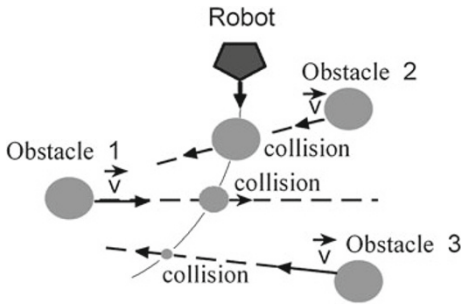


**Fig. 6** Collision points by obstacles awareness

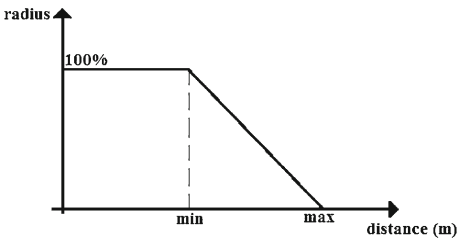


**Fig. 7** Obstacle size versus distance

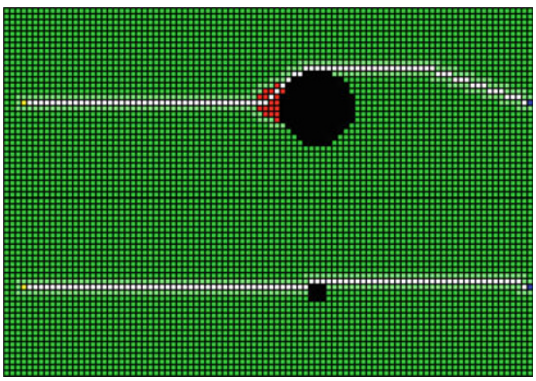

**Fig. 8** Importance modification result

calculation because fewer obstacles will lead to less visited cells and a lower amount of time to find a solution.

Note that in Fig. 7:

1. **min** = Starting distance for decreasing the obstacle's importance
2. **max** = Distance for total loss of obstacle's importance
3. **radius** = here, as the obstacle goes far from the robot, the obstacle's importance decreases and this is measured by the obstacle's radius.

The improvement made by this change is visible in the comparison shown by Fig. 8 in an environment with a static obstacle. The image on the top is the trajectory without the modification, while the image below includes the modification.

### *2.2.2 Obstacle representation (slack)*

This modification changes the way an obstacle is represented in the cells. A security area is created around the obstacle. This area is built by setting the cost for those cells above the free ones, but still allowing the robot to choose a path through those cells if the algorithm finds it optimal (Fig. 9).

This *does not* make the obstacle bigger (*nor does it expand*) but creates a security zone that should be avoided if doing so does not cause any impact on the optimal solution. There are cases where an optimal solution can be found using that zone instead of choosing a longer path. The equation for calculating the can be seen below.

$$C(n_1, n_c) = C(n_1, n_2).\mathbf{Cs} \tag{2}$$

where

1. $C(n_1, n_c)$ = Cost for going from node 1 to node c;
2. $C(n_1, n_2)$ = Cost for going from node 1 to node 2;
3. **Cs** = Cost inside the slack zone.

The **Cs** can be set by the graph in Fig. 10.

The improvement caused by this change is visible in the comparison shown by Fig. 11 in an environment with a static obstacle. The image on the left is the trajectory without

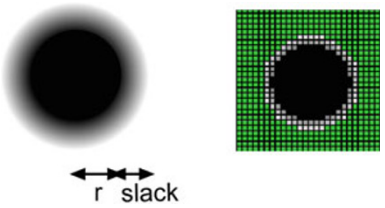


**Fig. 9** Obstacle with a slack zone. The black intensity means a higher cost

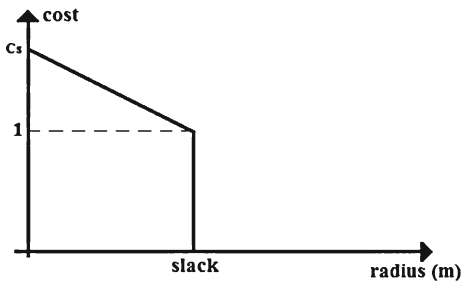


**Fig. 10** Slack zone cost

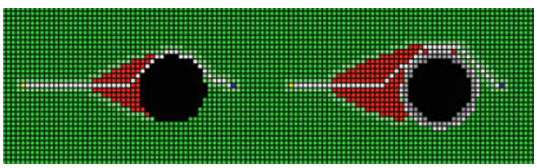

**Fig. 11** Slack modification result

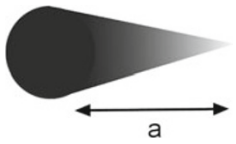


**Fig. 12** Obstacle shape change due to its motion

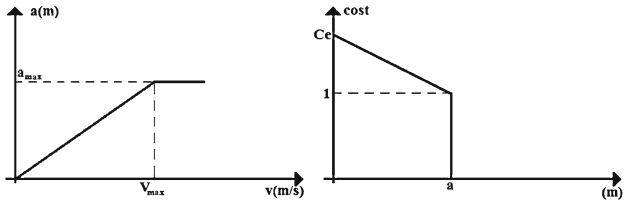


**Fig. 13** Determination of the size and trail cost graphs respectively

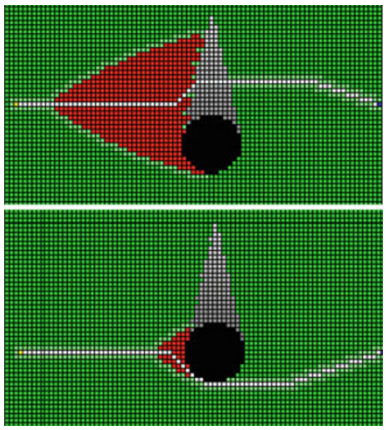

**Fig. 14** Direction modification result

the modification, while the image on the right includes the modification.

### *2.2.3 Obstacle direction*

A moving obstacle can obstruct the robot for a longer period of time if the path to avoid the obstacle ends, moving the robot parallel to the obstacle movement. This change creates a certain dynamic awareness of an otherwise static map. It creates an additional zone for which the cost to travel there is increased. This zone is created around the projected direction of the moving obstacle. The size of this zone depends on the magnitude of the obstacle's speed (Figs. 12, 13).
where:

1. **a** = Magnitude of the direction zone;
2. **Ce** = Cost inside the direction zone.

The improvement caused by this change is visible in the comparison shown by Fig. 14 in an environment with a static obstacle. The image on the top is the trajectory without the modification, while the image below includes the modification.

Finally, it is important to mention that, despite the fact that in real applications the obstacles usually have a non-constant velocity, our algorithm was optimized to be executed in a fast fashion. In each control loop the algorithm is recalculated and the unpredictability of the obstacle detection is smoothened. Usually, the errors in the obstacle's position and velocity estimations decrease abruptly when the obstacle approaches the robot, and therefore for the important obstacles (the ones near the robot) the uncertainty is not high.

### *2.2.4 Processing time*

This change addresses a modified heuristic for the A* search algorithm that reduces the computing time and finds the optimal search effort level, considering the computing time and the optimal path costs. This is the only modification that is not new, as it is well known as weighted A* and is usually used in the anytime algorithms. Also, it was first mentioned by [26]. To do this adjustment, it is necessary to set the correct heuristic parameter **k**. Using Eq. 3 with $\mathbf{k} = 1$ there is a guarantee that the final solution is optimal. Using a higher value for **k**, the search space is reduced and the solution found can be suboptimal. When performing path planning with the original A* method with different **k** values, it is noted that as **k** increases, the region of possible paths decreases. As a result, it is possible to observe that the computing time can be controlled, possibly paying the price of having a suboptimal path where the length of the path found is extended. In fact, **k** affects processing time and path length. While the first increases, the second decreases. Assuming the cost as a weighted sum of both variables, an optimized **k** can be found. However, it will depend on the path type and obstacles.

$$h(x, y) = \mathbf{k}\sqrt{(x - x_t)^2 + (y - y_t)^2} \tag{3}$$

Therefore, it is desirable to find a compromise between cost of computing time and the quality of the result (set by the heuristic parameter **k**). As a result of simulations, the average total cost in computing time can be seen in Fig. 15. It is possible to obtain the minimum cost for a $\mathbf{k} = 1.2$, thus resulting in an acceptable and a much faster suboptimal path. Finally, it is important to notice that this modification can be made in any A*-family algorithm if a suboptimal value is found when studying its time processing.

### *2.2.5 Target orientation*

This change tries to set the required orientation used by the robot as it approaches the target. Without it, the robot will hit the target destination from any direction. There are cases when the approach direction is mandatory. For these

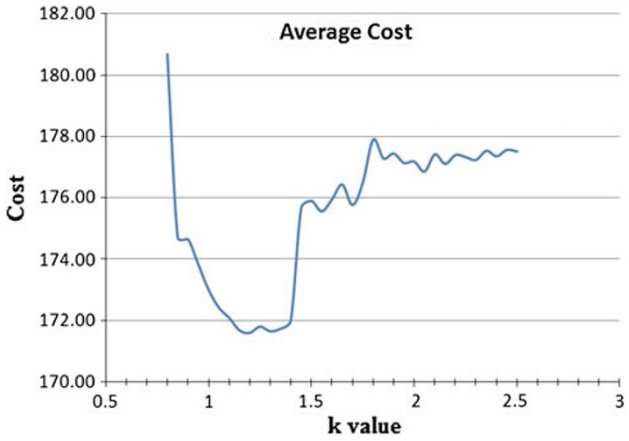


**Fig. 15** Average total cost

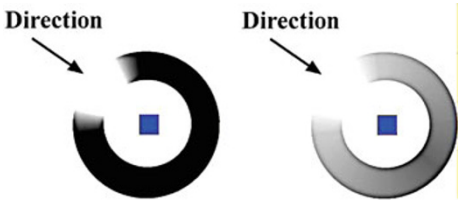


**Fig. 16** Target point with mandatory (*left*) and non-mandatory (*right*) approach direction

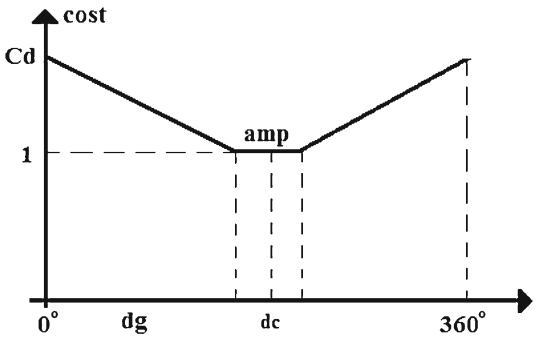


**Fig. 17** Goal approach direction cost

cases, a restriction like the one depicted in Fig. 16 (left) is used.

Sometimes there is a preferred direction, but that restriction is not strict. It can be violated if the gain in the arrival time is significant. To achieve this, a softer version of the extra obstacle is used, as depicted in Fig. 16 (right).

To calculate the approach cost, the image above (Fig. 17) can be used. Where:

1. **Cd** = Approach direction cost;
2. **amp** = Amplitude of the approach direction;
3. **dc** = Center of the amplitude;
4. **dg** = Distance of cost decrement in approaching the amplitude of the goal direction.

**Table 1** Simulation and experiment parameters

| | Middle size | Small size |
|---|---|---|
| *Distance* | | |
| max | 4 m | 2 m |
| min | 2 m | 1 m |
| *Slack* | | |
| Slack | 0.25 m | 0.06 m |
| Cs | 5 | 5 |
| *Direction* | | |
| Ce | 5 | 1.35 |
| a | 0.65 m | 0.6 |
| *Heuristic parameter* | | |
| k | 1.2 | 1.2 |
| *Target orientation* | | |
| amp | 60 | 28.6 |
| Cd | 5 | 1.3 |

## 3 Results

The robots from FEUP's robot soccer competition (5DPO—Small and Middle Size leagues) were used as a test bed to evaluate the produced algorithm. Both the middle size and small size league robots can run up to 2.5 m $s^{-1}$ in a straight line. Two types of evaluation were made. Firstly, a simulation was made using the software called SimTwo, developed by Professor Paulo Costa, PhD, from FEUP, using the middle size league robot. In this simulation, the used field is $8 \times 10$ m. Also, the grid cell used in both A* and its modified version was 0.05 m. Secondly, experiments with real Small Size League robots from FEUP were performed as well. In this case, the grid cell size was 0.03 m. All path-planning algorithms were implemented in a software written in free Pascal Lazarus compiler which communicates with SimTwo or the real robots by UDP protocol. The values used in the constants for each change characterizing the modified algorithm can be seen in Table 1.

The results were divided in two scenarios: simple scenarios demonstrating experimental results with real robots where the modifications are easy to be acknowledged separately and a more complex scenario using twelve random spheres representing mobile obstacles presenting the simulation results. In the complex scenario with high dynamics, the simulations demonstrate the final time of execution and the number of collisions, where in the real robot scenario, each modification can be seen acting in the three performed experiments.

### 3.1 Simulation results

All simulations were made with the SimTwo software. This software uses an Open Dynamic Engine library [10], [31]

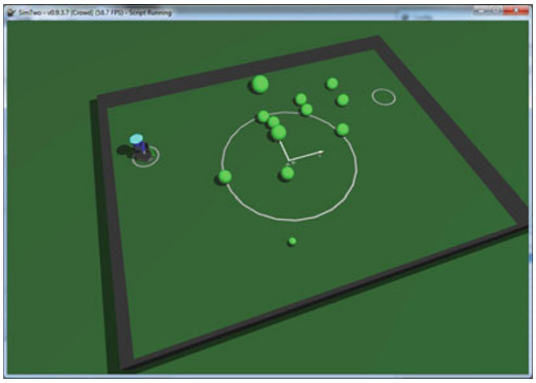

**Fig. 18** The SimTwo simulation environment

which guarantees a perfectly realistic simulation from the dynamics of rigid object, resulting in a realistic object behavior. The robots, nevertheless, were highly and rigorously previously parameterized in SimTwo [12], which makes this platform a realistic and ideal simulation environment for tests with path-planning algorithms. Below is a screenshot of the program (Fig. 18).

It is important to mention that in 2005, before the appearing of simulators such as Gazebo, ROS and so on, the author in [10] had already developed the SimTwo simulator based on the same ODE with similar characteristics (same realism in the dynamics and physical impacts), but with a much simpler installation and use, and the code of which has been mastered by the authors of this paper.

The aim with this set of simulations is to observe the differences between A* and A* with the proposed modifications in a highly dynamic environment with mobile obstacles that move "randomly" at speeds that, for some of the obstacles, can be higher than the speed of the robot itself. Finally, it is important to mention that all simulations were made with all objects (robot and obstacles) in the same position.

The A* algorithm builds a path from the robot's initial state to the goal state that goes around the obstacles; the modified algorithm builds a path and only changes it when the importance of the obstacle increases according to the proximity to the robot.

**Algorithm Set Sphere Speed**

1: **If** ∃ Sphere **then**
2: $NexPosition_x$=$Initial_x$+Radius·cos(Speed·(t+δ))
3: $NexPosition_y$=$Initial_y$+Radius·sin(Speed·(t+δ))
4: **Set** Force of ith-Sphere with
5: $V_x = 100*(NexPosition_x - Position_x)$
6: $V_y = 100*(NexPosition_y - Position_y)$
7: $V_z = 0$
8: **end**
9: **end**

In the simulations, the movement of the obstacles is set to be half- random. This happens because those movements are

**Fig. 19** A* dynamic simulation results with discrete instants $t = 1, 2, 3, 4, 5, 6, 7$ and 8

set by a simple algorithmic procedure in SimTwo. As can be seen in the previous algorithm, the values from each sphere are uploaded and a force is set upon each one, making the movements. With a constant velocity for all the spheres, the movements of each is practically always the same, except when the robot hits one or more spheres. In Figs. 19 and 20, the discrete evolution in the A* and the Modified A* path planners can be seen until the target is reached. An important observation in the last sub-pictures of both figures is that the track became static and the robot chose not to use the A* or its modified version, due to the fact that there were no more obstacles around or near it. Therefore, SimTwo shows the last track calculated by the path planner in case.

It is possible to see now that A* has to go back many times because it was not able to foresee collision points. However, the modified A* algorithm builds a different path, and changing it by predicting the collision points using the calculated

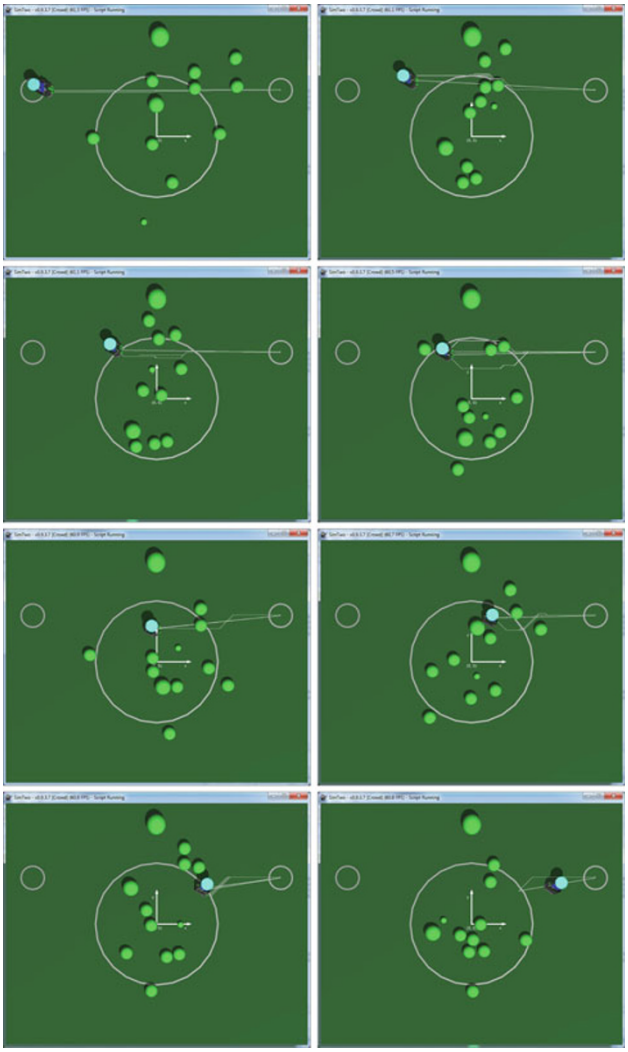

**Fig. 20** Modified A* dynamic simulation results with discrete instants $t = 1, 2, 3, 4, 5, 6, 7$ and 8

spheres' velocity and applying all the mentioned changes. As can also be observed in Table 2, the improvement gained by the modification is much more considerable, not only in terms of getting to the target point sooner, but also when it comes to avoiding more collisions in a crowded environment where obstacles can sometimes be unavoidable, even for humans.

Collisions occur when the robot is blocked by the moving obstacles. Therefore, the collisions cannot be avoided because the obstacles go towards the robot.

## 3.2 Experiment results

Two 5DPO robots from FEUP's Small Size League were used for these experiments. These 5DPO can run up to 1.2 m $s^{-1}$. Therefore, by applying them in real experiments, the same mathematical constrains that were imposed in the simulation

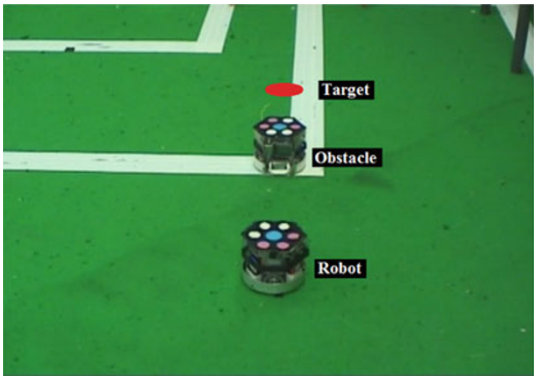


**Fig. 21** Static obstacle representation

**Table 2** Dynamic obstacles result

| Measurements in 30 sim. | A* | Mod. A* |
|---|---|---|
| Average time to target | 29.72 s | 26.77 s |
| Number of collisions | 3.4 | 1.6 |

problems occur. Thus, the experiments can be divided into three cases.

The first case presents a static obstacle located in the robot's path. Then, the robot has to reach the goal state on the other side of the obstacle, avoiding it. In a second case, the obstacle is moving towards the robot. In this case, the robot must also avoid the obstacle to reach the target point. Finally, in the third case, it is stated that the robot, starting from an initial state $q_{\text{init}}$, must reach the goal state $q_{\text{goal}}$ avoiding a moving obstacle.

### *3.2.1 Case 1: static obstacle*

For the first set of tests, the robot departs from the initial state $q_{\text{init}}$ with an initial velocity equal to zero. The accelerations of the robot should be limited to prevent the robot from slipping. Due to the robot's dynamic constrains, it does not succeed in following the planned path, especially if the path is full of abrupt turns.

Figure 21 represents the first experiment with real robots. In Fig. 22 it can be seen that the path resulted from both algorithms, the normal A* algorithm in red and its modified version in blue. The trajectory in Fig. 22 is not the planned path, but the trajectory made by the robot. Note that the modified algorithm in a static environment makes the robot turn sooner than when the normal A* algorithm is used.

On the other hand, the A* algorithm makes the robot turn much closer to the obstacles. In a dynamic environment this could cause an unwanted collision. In this experiment, the robot starts farther away from the obstacle. Therefore, when it reaches the obstacle, the robot is moving at a higher velocity. Here, the effects in the controller made by A* due to the robot's dynamics and constrains are clearly visible. Due to

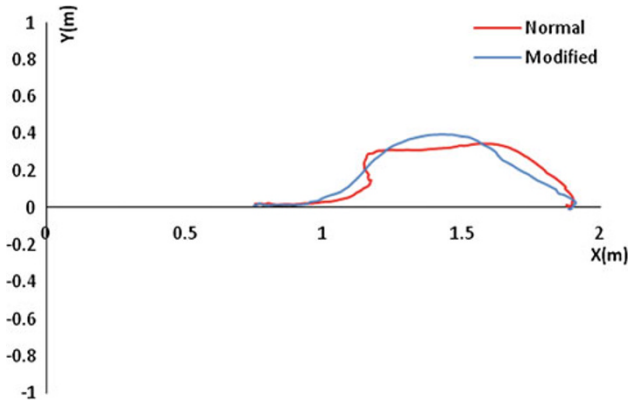


**Fig. 22** Static obstacle avoidance result

**Table 3** Close static obstacle avoidance results

| | Duration(s) | Gain | APT (ms) |
|---|---|---|---|
| Normal | 3.08 | | 0.09 |
| Modified | 2.64 | 14.3 % | 0.12 |

the high velocity and the abrupt turn too near the obstacle the robot controller was not successful in following the path when using the A* algorithm. Therefore, there was a large trajectory tracking error, and A* had to replan the path so that the robot could correct its orientation. This happened because, since A* choose the shortest path, the robot turned closer to the obstacle. At high velocities, this causes the robot controller make bigger efforts so that the robot can follow the desired trajectory. This makes the robot go farther and lose speed. Meanwhile, the modified algorithm chooses a suboptimal solution in a shorter amount of time to reach the target and, along with the cost of abrupt maneuvers, makes the robot run through a much smoother trajectory in a shorter amount of time, allowing the controller to follow the path that was planned. The total time to reach the target can be seen in Table 3, where the modified A* algorithm makes the robot reach the goal sooner than the normal A*, as well as the gain and the average processing time (APT) of both algorithms.

Finally, the modification of slack, direction and orientation increase the processing time, while the modifications of distance and processing time decrease the average processing time of the algorithm. In general there is a small increase in the APT, although it is not large enough to jeopardize the use of this algorithm in real environments and in each control loop.

### *3.2.2 Case 2: moving obstacle towards the robot*

In the second experiment, the robot starts far from the target. In the middle of the trajectory there is an obstacle that is moving towards the robot at 0.4 m $s^{-1}$. The Fig. 23 shows

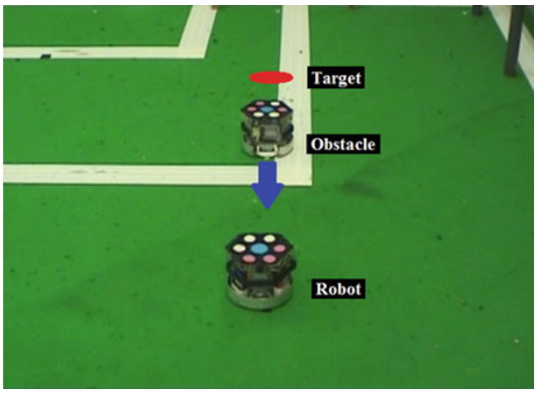


**Fig. 23** Moving obstacle representation—case 2

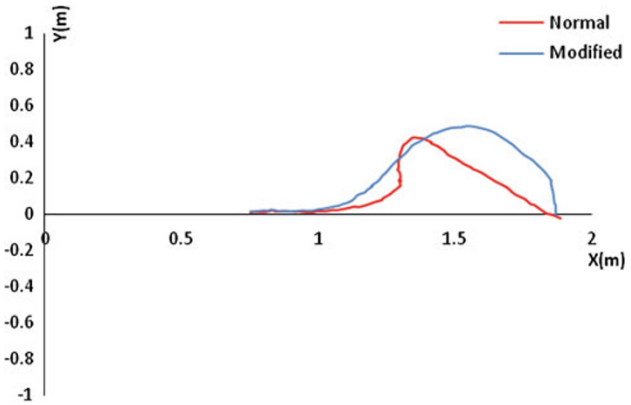


**Fig. 24** Avoidance of the moving obstacle towards the robot

**Table 4** Avoidance of the moving obstacle towards the robot results

| | Duration (s) | Gain | APT (ms) |
|---|---|---|---|
| Normal | 3.284 | | 0.10 |
| Modified | 2.763 | 15.8 % | 0.13 |

the representation of the experiment. It can be verified here also that the robot succeeds in avoiding the obstacle sooner using the modified version of the A* algorithm. Similarly to what happened in the first case, the path built by the modified algorithm caused the robot to turn earlier. This results in a softer path, which does not happen in the normal A* algorithm, as confirmed by Fig. 24. Experiments showed that the robot can avoid obstacles in this situation that move only up to 0.9 m $s^{-1}$.

In this last case, the robot starts at the initial position (far left) at time $\mathbf{t} = 0$, and the aim is for the robot to reach the goal position (far right) with an average velocity of 0.69 m $s^{-1}$. Meanwhile, there is an obstacle with constant velocity of 0.8 m $s^{-1}$ crossing the robot's path, starting at the top of the figure. The total time to reach the target can be seen in Table 4, where the modified A* algorithm makes the robot reach the goal sooner than the normal A*.

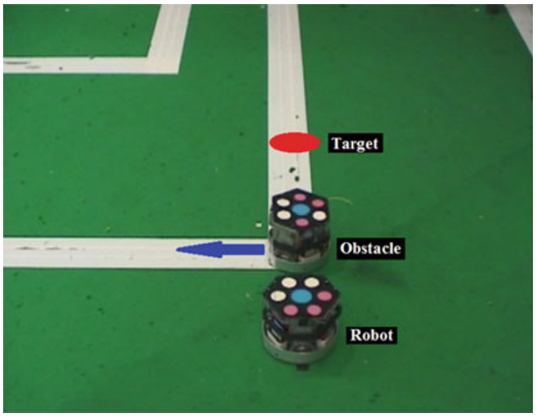


**Fig. 25** Moving obstacle representation—case 3

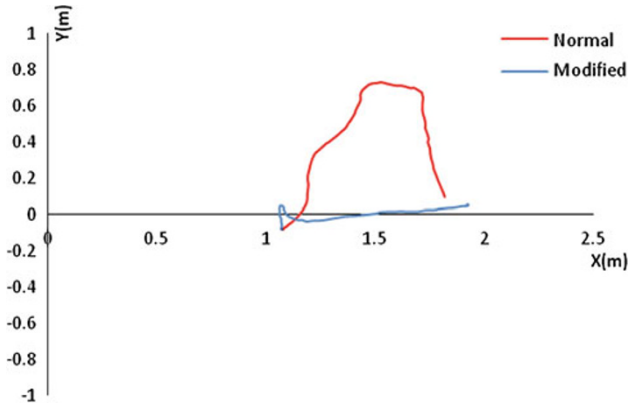


**Fig. 26** Avoidance of the moving obstacle intersecting the robot

**Table 5** Avoidance of the moving obstacle intersecting the robot results

| | Duration (s) | Gain | APT (ms) |
|---|---|---|---|
| Normal | 2.963 | | 0.12 |
| Modified | 2.205 | 25.6 % | 0.21 |

### *3.2.3 Case 3: moving obstacle crossing the robot's path*

The representation of the experimental result can be seen in Fig. 25. In this experiment, it is possible to observe that when A* is used, the robot tries to avoid the obstacle and therefore is taken in the direction of the obstacles' movement as is is shown in Fig. 26. This happens because the optimal path generated by A* makes the robot pass in front of the obstacle. However, when using the modified version of A*, the robot uses the obstacle's velocity and therefore predicts the collision point at a time **t**. Using the direction of the obstacle, the modified algorithm builds an suboptimal solution, making the robot pass behind the obstacle to avoid it. Table 5 shows an even bigger difference in the time it takes for the robot to reach the target. The improvement made by the modified A* is much clearer here.

## 4 Conclusion and future work

This paper presented a set of novel modifications that can be applied to any grid-based path-planning algorithm from the A* family used in mobile robotics. It used five modifications on A* to plan the robot's path: the obstacle distance, slack, direction, processing time and target orientation. Some simulations were made using a crowded and highly dynamic environment with twelve randomly moving obstacles. While the normal A* algorithm built an entire path around the obstacle, the modified A* built a path making changes only when the robot was approaching the obstacle. Here, the normal A* algorithm had to go back many times to succeed in reaching the goal point. The modified algorithm built a different path, changing it by predicting the collision points using the calculated spheres' velocity and applying all the mentioned changes. The improvement achieved by the modified A* algorithm was much more considerable, not only in terms of getting to the goal point sooner, but also in terms of avoiding much more collisions in a crowded environment.

Real experiments were also made. The experiments were divided into three cases: static obstacle, moving obstacle towards the robot, and moving obstacle intersecting the robot's path. For the first set of tests, the modified A* algorithm reached the goal sooner than the normal A*. In the second case, the modifications made much more difference with a moving obstacle. This resulted in a softer path in the both first and second cases. In A* the robot had to make a second turn so that it would not collide with the obstacle. This made the robot go farther and lose speed. In the last case, the robot had to reach the target avoiding a moving obstacle that intersected the robot's path. Here, the experiment showed that when using A*, the robot was taken in the obstacles' movement direction and, while using the modified A*, the robot predicted the collision point and built a suboptimal solution, making the robot pass behind the obstacle to avoid it. This last case showed an even bigger difference in the time it takes for the robot to reach the target.

It is important to mention that the modifications proposed are in the configuration space ($C_{\text{space}}$) and not in the algorithm core itself. Therefore, in the matter of configuration space, all the previous algorithms from the A* family should give an equal or similar solution to the A* algorithm. When applying our modification in any algorithm from the A* family, the final solution would be better, as will be demonstrated with A* in this paper. These modifications aimed to improve the trajectory with respect to the time of execution, and especially in avoiding collisions when used in mobile robotics in highly dynamic environments.

Future works will consider experiments with the uncertainty treatment in the obstacle's velocity measurement, and a model for this uncertainty will be created. This uncertainty estimation will be used to readjust some parameters of the

modified algorithm. The modified algorithm presented in this paper was not configured to all cases (simulation with small size robots, real small size robots, simulation with meddle size robots, or crowded environment) and this future work would bring more robustness to our approach.

**Acknowledgments** The authors would like to thank INESC TEC and FCT for their financial support.

## References


1. Alejo D, Conde R, Cobano J, Ollero A (2009) Multi-UAV collision avoidance with separation assurance under uncertainties. In: 2009 IEEE international conference on mechatronics, pp 1–6
2. Balkcom DJ (2006) Time-optimal trajectories for an omni-directional vehicle. Int J Robot Res 25:985–999
3. Belkhouche F (2009) Reactive path planning in a dynamic environment. IEEE Trans Robot 25:902–911
4. Bernabeu EJ (2009) Fast generation of multiple collision-free and linear trajectories in dynamic environments. IEEE Trans Robot 25:967–975
5. Bhattacharya P, Gavrilova M (2008) Roadmap-based path planning—using the Voronoi diagram for a clearance-based shortest path. IEEE Robot Autom Mag 15:58
6. Bruce J, Veloso M (2006) Safe multirobot navigation within dynamics constraints. In: Proceedings of the IEEE, vol 94, pp 1398–1411
7. Cai C, Ferrari S (2009) Information-driven sensor path planning by approximate cell decomposition. IEEE Trans Syst Man Cybernet B Cybernet 39:672–689
8. Conceicao AS, Moreira A, Costa P (2009) Practical approach of modeling and parameters estimation for omnidirectional mobile robots. In: IEEE/ASME transactions on mechatronics, pp 377–381
9. Costa P, Moreira AP, Costa PJ (2009) Real-time path planning using a modified A* algorithm. In: ROBOTICA 2009—9th conference on mobile robots and competitions, pp 141–146
10. Costa PJ (2012) Simtwo. http://paginas.fe.up.pt/paco/wiki/index.php?n=Main.SimTwo
11. Ferguson D, Likhachev M, Stentz A (2005) A guide to heuristic-based path planning. In: Proceedings of the international workshop on planning under uncertainty for autonomous systems. International conference on automated planning and scheduling (ICAPS), pp 1–10
12. Ferreira JR, Moreira APGM (2010) Non-linear model predictive controller for trajectory tracking of an omni-directional robot using a simplified model. In: 9th Portuguese conference on automatic control
13. Fiorini P, Shiller Z (1998) Motion planning in dynamic environments using velocity obstacles. Int J Robot Res 17(7):760–772
14. Ge SS, Lai X, Mamun AA (2005) Boundary following and globally convergent path planning using instant goals. IEEE Trans Syst Man Cybernet 35(2):240–254
15. Haro F, Torres M (2006) A comparison of path planning algorithms for omni-directional robots in dynamic environments. In: 2006 IEEE 3rd Latin American robotics symposium, pp 18–25
16. Jan G, Parberry I (2008) Optimal path planning for mobile robot navigation. IEEE/ASME Trans Mechatron 13:451–460
17. Khantanapoka K, Chinnasarn K (2009) Pathfinding of 2D & 3D game real-time strategy with depth direction algorithm for multi-layer. In: 2009 Eighth international symposium on natural language processing
18. Kurihara K, Nishiuchi N, Hasegawa J, Masuda K (2005) Mobile robots path planning method with the existence of moving obstacles. In: 2005 IEEE conference on emerging technologies and factory automation, pp 195–202.
19. Lai X-c, Ge SS, Mamun AA (2007) Hierarchical incremental path planning and motion planning considering accelerations. IEEE Trans Syst Man Cybernet 37:1541–1554
20. Latombe J-C (1991) Robot motion planning. Kluwer, Dordrecht
21. Li H, Yang SX, Seto ML (2009) Neural-network-based path planning for a multirobot system with moving obstacles. IEEE Trans Syst Man Cybernet C 39(4):410–419
22. Likhachev M, Ferguson D, Gordon G, Stentz A, Thrun S (2005) Anytime dynamic A*: an anytime replanning algorithm. In: Proceedings of the international conference on automated planning and scheduling (ICAPS)
23. Nascimento TP, Conceição AGS, Moreira APGM (2010) Omnidirectional mobile robot's multivariable trajectory tracking control: a robustness analysis. In: 9th Portuguese conference on automatic control
24. Ögren P, Leonard NE (2005) A convergent dynamic window approach to obstacle avoidance. IEEE Trans Robot 21(2):188–195
25. Pathak K, Agrawal SK (2005) An integrated path-planning and control approach for nonholonimic unycycles using switched local potentials. IEEE Trans Robot 21:1201–1208
26. Pearl J (1984) Heuristics: intelligent search strategies for computer problem solving. Addison-Wesley, New York
27. Peasgood M, Clark CM, McPhee J (2008) A complete and scalable strategy for coordinating multiple robots within roadmaps. IEEE Trans Robot 24:238–292
28. Qu H, Yang SX, Willms AR, Yi Z (2009) Real-time robot path planning based on a modified pulse-coupled neural network model. IEEE Trans Neural Netw 20:1724–1739
29. Rahman N, Jafri A (2005) Two layered behaviour based navigation of a mobile robot in an unstructured environment using fuzzy logic. In: Proceedings of the IEEE symposium on emerging technologies, pp 230–235
30. Ray AK, Behera L, Jamshidi M (2008) Sonar-based rover navigation for single or multiple platforms. Forward safe path and target switching approach. IEEE Syst J 2(2):258–272
31. Smith R (2010) Open dynamics engine. http://www.ode.org
32. Tang P (2001) Dynamic obstacle avoidance based on fuzzy inference and transposition principle for soccer robots. In: 10th IEEE international conference on fuzzy systems (Cat. No.01CH37297), pp 1062–1064
33. Wilkie D, Berg J, Manocha D (2009) Generalized velocity obstacles. In: IEEE/RSJ international conference on intelligent robots and systems, New York
34. Willms AR, Yang SX (2008) Real-time robot path planning via a distance-propagating dynamic system with obstacle clearance. IEEE Trans Syst Man Cybernet B: Cybernet 38(3):884–893
35. Yang J, Qu Z, Wang J, Conrad K (2010) Comparison of optimal solutions to real-time path planning for a mobile vehicle. IEEE Trans Syst Man Cybernet A: Syst Humans 40(4):721–731
36. Yang S (2002) Real-time torque control of nonholonomic mobile robots with obstacle avoidance. In: Proceedings of the IEEE international symposium on intelligent control, pp 81–86
37. Yao J, Lin C, Xie X, Wang AJ, Hung C-C (2010) Path planning for virtual human motion using improved A* star algorithm. In: 2010 Seventh international conference on information technology: new generations