View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Biblioteca Digital do IPB

Towards an Orientation Enhanced Astar Algorithm for Robotic Navigation

Elisabete Fernandes*, Pedro Costa*†, José Lima*‡, Germano Veiga*
*INESC TEC - Tecnologia e Ciência (formely INESC Porto)
†FEUP - Faculdade de Engenharia da Universidade do Porto
‡Instituto Politécnico de Bragança
Email: {elisabete.s.fernandes, germano.veiga}@inesctec.pt , pedrogc@fe.up.pt , jllima@ipb.pt

***Abstract*—This paper presents an algorithm capable of generating smooth, feasible paths for an any-shape non-holonomic mobile robot, taking into account orientation restrictions, with the aim of navigating close to obstacles. Our contribution consists in an extension of the A* algorithm in a cell decomposition, where besides its position, the orientation of the platform is also considered when searching for a path. This is achieved by constructing 16 layers of orientations and only visiting neighbor layers when searching for the lowest cost. To simplify collision checking, the robot's footprint is used to inflate obstacles, yet, to allow the robot to find paths close to obstacles, the actual footprint of the robot must used. By discretizing the orientation space into layers and computing an oriented footprint for each layer, the actual footprint of the robot is used, increasing the configuration space without becoming computationally expensive. The path planning algorithm was developed under the EU-funded project CARLoS[1] and was implemented in a stud welding robot simulated within a naval industry environment, validating our approach.**

I. Introduction

There are several approaches that address the automation process in naval industry. The MINOAS project [1] [2] is an example of such an effort, by projecting a climbing robot to perform vessel inspection. In [3] a robotic crawler for ship hull inspection is also described.

Among many others, the pre-outfitting, that include the stud welding of pins to support the insulation and the marking operations for other components such as extinguishers or pipes, represents a significant value, estimated in 10% of the overall cost of the ship [4]. These tasks are still a manual labor. In order to automate this process, a robotic mobile manipulator is being developed in the scope of the CARLoS project.

Mobile manipulators have an extended workspace, but not without some challenges. For a mobile robot it is often enough to compute a path that minimizes some cost, such as distance or computation time but in the context of mobile manipulators, this may not be the case. On the one hand manipulators usually require contact with objects and have a limited workspace, so the platform's position is conditioned by the range of the manipulator. On the other hand, mobile robots have the ability to move in the environment. Combining both characteristics allows robotic applications to go beyond new approaches.

In the context of stud welding the robot needs to work close to walls. As the robot is non-holonomic and its dimensions do not allow an in-place rotation due to its proximity to the wall, the robot should be able to arrive at the desired position with the correct orientation, as it is not capable of rotating at the goal. Most path planning techniques would fail to compute a path because they use an inflated footprint of the robot, often circular, not being able to generate a path so close to an obstacle, or do not take into account the orientation restrictions when approaching an obstacle. Because the robot is a tracked vehicle, pure rotations constitute the most demanding case, with more power consumption and more slippage on the ground. As the localization system also has the most error in such cases, in-place rotations were eliminated from the search space.

Most solutions found in the literature that cope with orientation are based on a set of primitives that construct a lattice graph, that is then searched upon [5] [6] [7]. Our approach combines the A* path planning algorithm [8] with the orientation of the robot. This way, it is possible to create a smooth path, that minimizes the distance while taking into account the orientation constraints in the map.

To find collision-free paths the planner must take the footprint of the robot into consideration. By using the actual footprint, the planner can plan as close to the wall as the heading of the robot allows it. However, full collision checking is expensive, so an expansion of the obstacles is often employed for efficiency. This way the robot can be assumed a single point and a full body pose collision check can be avoided [7].

This paper begins with a review of related work, followed by a description of the platform used in section III. The developed algorithm is stated in section IV and the results are presented in V.

II. Related Work

A path planning task finds the navigation path between two specified locations, the initial and the goal state with an associated cost. The path is possible and optimal if the total cost is minimal across all feasible paths leading from the initial position (start state) to the goal position (goal state). There are many solutions available to plan a path for mobile robots movement. Approaches such as Potential field planners, Probabilistic Roadmap (PRM), Tree-based planner (RRT) and Non-holonomic and Kinodynamic planning, among others, are topics addressed by researchers to solve the path planning problem.

[1] http://carlosproject.eu

978-1-4799-7800-7/15/$31.00 ©2015 IEEE

A common technique for robotic path planning consists of representing the environment (or configuration space) in a discrete set of states. Planning a path can therefore be seen as a search problem on this set (or graph). Classical graph search algorithms have been developed for calculating least-cost paths on a weighted graph; the most popular one is the Dijkstra's algorithm [8] whereas A* [8] is basically an informed variation of Dijkstra. A* operates essentially in the same way as Dijkstra's algorithm except that it searches towards the most promising states using an heuristic function, saving time and computational resources.

Search based methods can be divided into two categories: *grid-based* and *lattice-based* [9]. Grid-based planners decompose the continuous space into a fixed cell grid. Typically nodes represent states and edges represent the cost of traversing trough states. On the other hand, lattice-based planners depend on state lattices that discretize the space into a set of reachable configurations. Nodes represent configurations and edges represent feasible paths. [10]

Both lattice-based and grid-based approaches allow any search method, for instance A*, to be applied in order to search the state space. The disadvantage of lattice is that it requires a set of motion primitives to be constructed beforehand [10] and therefore a trajectory generator, such as [11] is needed to determine all feasible paths in the free space.

One of the limitations of the original A* is that it's algorithm is static. This means applications such as mobile robots with real time constraints are not suitable to use the original A*. Several variants of the A* were developed, such as replanning algorithms (e.g. D*), anytime algorithms (e.g. ARA*), and anytime replanning algorithms (e.g. AD*). Replanning every time the scene changes seems to be a waste of computation. Instead, it may be more efficient to take the previous solution and repair it taking into account the changes to the graph. Anytime algorithms address the high dimensional search problem by first computing a feasible path and improving it towards an optimal solution during execution.

In order to cope with the high dimensional space, several approaches are used. Sampling based algorithms, such as PRM and RRT are frequently used because they are able to find smooth paths in high dimensional space [9]. However, they often produce highly suboptimal paths and require a post-processing step to find a feasible path [7] .

Another approach is to rely on a discretization of the configuration space. In [12] the state space is reduced by dividing the orientation space into discrete intervals, representing the free space. By representing the collision-free configuration space in a more compact manner, it is suitable for an incremental planner.

III. Mobile Platform

The platform used in this implementation was Guardian[2]. Guardian is a differential mobile manipulator, with both tracks and wheels and with skid steered locomotion. This platform was selected because of the unstructured aspect of ship building. The platform can be used with or without the wheels, which rotate together with the tracks. The tracks allow the robot to pass over objects and doors within the ship.

The platform is roughly 1.1x0.5m with tracks, and 1.1x0.75m with wheels. It is equipped with 2 Hokuyo laser range finders, a URG-04LX in the front and a URG-04LX-UG01 in the back, for a 360 degrees view of the world. This allows the platform to drive both forward and backwards.

The software was implemented under ROS (Robot Operating System) [13]. This architecture allows a modular approach and the interface between different tasks (ROS nodes) becomes easier.

Guardian is also one of the robots available in ROS[3] and is simulated under Gazebo[4]. Fig. 1 represents such simulation. To avoid a more complex computation, the arm was not simulated in Gazebo.

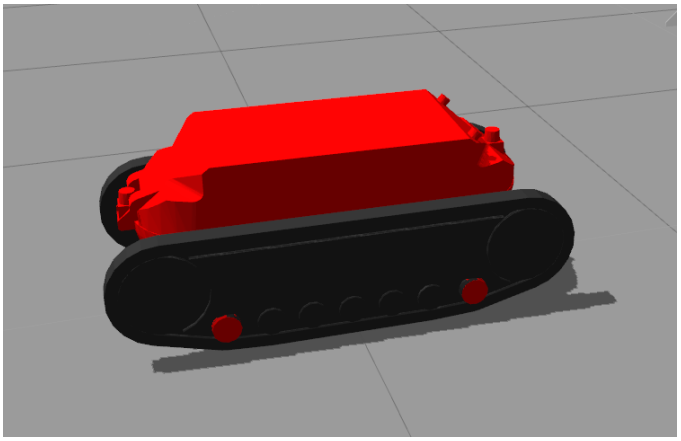


Fig. 1. Guardian simulation in Gazebo

IV. Implementation

A. Path planning with Orientation Restrictions

A* is a graph search algorithm that finds the lowest cost path from a given initial position to a goal position. It can work with a cell based map. Each cell (position XY) represents a node. A* explores the environment by computing a cost function for each possible cell to search and then selects the lowest cost position to add to the search space. In this approach, the differences relative to the traditional A* by cell decomposition are:

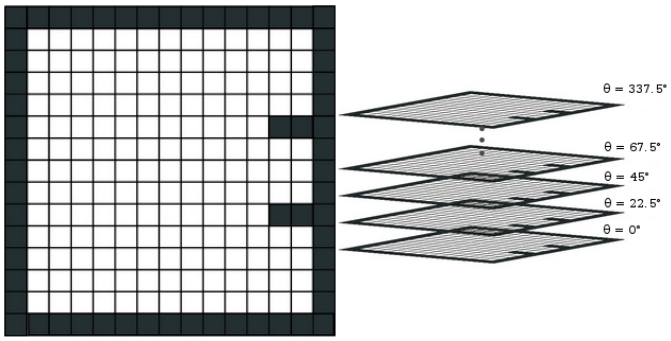
1) Map: From an input 2D map, Fig. 2a, a 3D representation of the map will be constructed, the third dimension being the orientation, as shown in Fig. 2b. There are 16 layers, each layer representing a range of orientations. With this discretization each layer represents 22.5°.

2) Neighbors: Traditionally, the A* in a cell decomposition has connectivity 8, where the 8 neighboring cells are the cells surrounding it, as illustrated in Fig. 3a. In this case the connectivity is 16, as shown in Fig. 3b. The aim is to have neighbors that can represent all possible directions. In this case, as we have 16 layers representing 16 angles, it is appropriate to have a neighbor for each angle.

[2]http://www.robotnik.eu/mobile-robots/guardian/

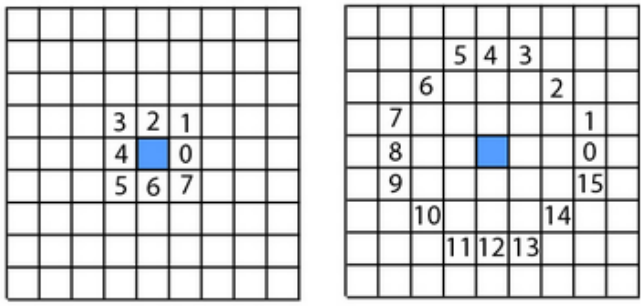
[3]http://wiki.ros.org/Robots/Guardian

[4]http://gazebosim.org/



(a) Input 2D map. (b) 16 layered map

Fig. 2. Map representation



(a) 8-connected neighbors. (b) 16-connected neighbors

Fig. 3. Neighbors representation

But in this case, in each iteration, only six of the sixteen neighbors are visited. Because we are trying to find a smooth path, it is unreasonable to visit orientations in a complete different direction than its current orientation, θ. So, for a node in layer n the six neighbors to visit are:

- Two neighbors in layer n, representing movement in the same direction of the layer (i.e. keep the orientation at that moment).
- Two neighbors in the preceding layer, $n-1$, representing movement in the same direction of the layer (i.e. rotate 22.5° clockwise).
- Two neighbors in the next layer, $n+1$, representing movement in the same direction of the layer (i.e. rotate 22.5° counter-clockwise).

For each layer, the two nodes represent moving forward and backwards in that same layer. So, for layer n, the nodes visited are n and $n+8$, representing movement in θ and $\theta + 180°$.

For instance, if the current orientation of the robot is $\theta = 0°$, then its current layer is $n = 0$ and:

- for layer 0, the nodes visited are 0 and 8.
- for layer 15, the nodes visited are 15 and 7.
- for layer 1, the nodes visited are 1 and 9.

This implies that the possible movements are forward and backwards motion with orientation 0°(layer 0), 22.5°(layer 1) and 337.5°(layer 15).

Fig. 4 illustrates such case. The blue cell represents the current node, the green cells represent the visited neighbors.

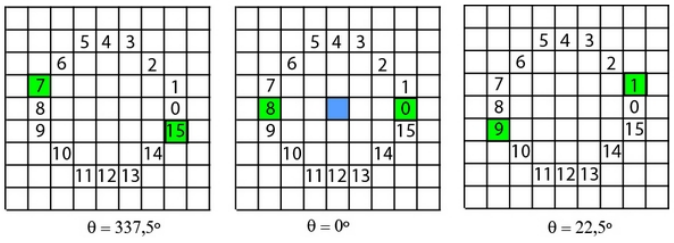


Fig. 4. Visited neighbors for a node in layer 0

B. Collision checking/ Layered obstacle expansion

Prior to the search the known obstacles are inflated by the footprint of the robot, serving two purposes. First, this eliminates the need for a full collision check, as the robot can now be expressed as a single cell and collision checking becomes a simple check of whether the robot position falls within a free or occupied cell in the map. Second, this also removes untraversable cells from the search, reducing the configuration space and optimizing the computation.

A common practice is to inflate the obstacles by the inner or outer radius of the robot [7], resulting in an optimistic or in a conservative approach. This approximation is adequate for circular or nearly squared robots, but fails for those rectangular or unsymmetrical. If the robot's distance to an obstacle is less than the inner radius, then the robot would be certainly in collision with the obstacle. For a distance farther than the inner radius, than a collision can happen, depending on the current heading of the robot.

Fig. 5 demonstrates the importance of the robot's orientation relative to the obstacles. In this case, it is clear that for the same configuration ($\theta = 0°$) the distance to the center of the robot is different considering a lateral or frontal wall, so using the inner or outer radius of the robot would not respond to both cases.

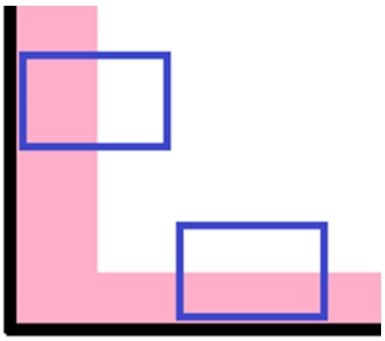


Fig. 5. Obstacle inflation depending on orientation

Using the inner radius will allow the planner to compute infeasible paths for some configurations while using the outer radius may cause the planner to discard feasible paths in low mobility zones, such as in narrow spaces or close to obstacles. As the purpose of the algorithm is to compute paths close to obstacles while also being feasible, the outer or inner radius

are not viable options. Our approach uses the actual footprint of the robot and works for any robot shape.

By using an oriented footprint of the robot, the configuration space is augmented relative to an outer radius inflation yet by using an orientation discretization the configuration space is compact enough for an efficient collision checking.

The orientation space is divided into 16 layers, and for each one, the oriented footprint of the robot is computed and the obstacles convoluted with this footprint. Each layer will have the same obstacle map, but will have different obstacle inflation, depending on the robot's orientation. This way, when searching, the planner may find a path for a specific orientation, close to an obstacle, that wouldn't be possible to obtain if the outer radius was employed.

C. Control

Although the work focus is in global path planning, a PD (proportional-derivative) controller was implemented to follow the computed paths.

V. Results

Tests were conducted on two different scenarios. One represents a ship interior scenario, 8.4x12.4m and the other represents some corridors, 24x12.4m. Fig. 6 illustrates such scenarios.

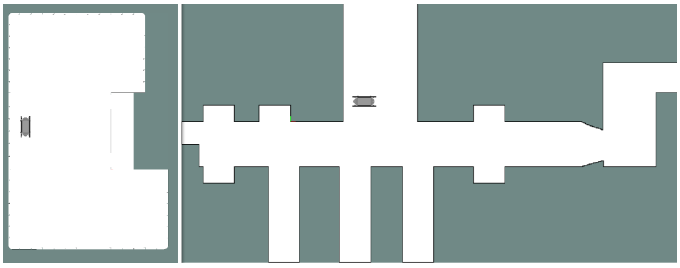


Fig. 6. Tested scenarios

The same map was tried with several resolutions ranging from 0.01m to 0.1m.

A. Orientation Restricted Paths

Fig. 7 illustrates the need for the orientation in the search space. The red path represents the path returned by ROS' implementation of Dijkstra and the blue one the path returned by our planner. Dijkstra (an uninformed version of A*) fails under our purposes because it requires turning in-place. This implementation only searches in (x,y) whereas our planner searches in (x,y,θ). The blue arrows represent the orientation of the robot in each point. In this particular example, the robot should move backwards until the second-last point and then forward until the goal point.

Fig. 8 illustrates the case in which the goal point is parallel to the initial point, with the same orientation. As the previous example, the original A* would compute a straight line, rotating both at the initial and the goal point. Our algorithm generates a smooth path that rotates only 22.5°in each iteration, avoiding in-place rotations.

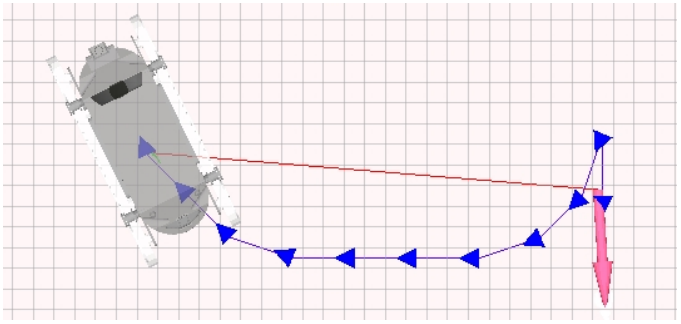


Fig. 7. Dijkstra vs orientation restricted path

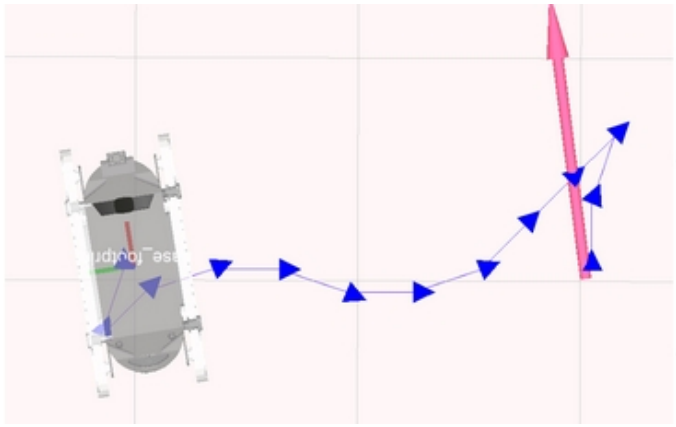


Fig. 8. Path generated for a goal parallel to the initial point - 0.1m resolution

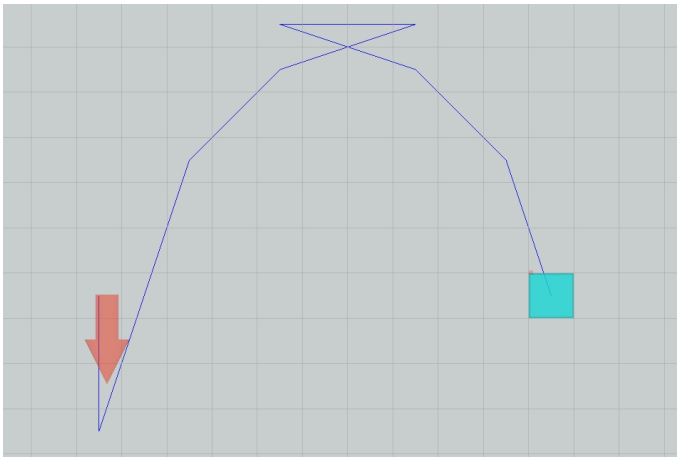


Fig. 9. Path generated for a goal parallel to the initial point - 0.01m resolution

Fig. 9 also exposes a parallel configuration. In this case the map resolution is higher (0.01m instead of 0.1m) and the goal (red arrow) is closer to the initial point (cyan square), 0.1m instead of 2m apart. The resolution makes the orientation arrows much closer to each other, making it to hard to comprehend, therefore, no arrows are showed in the picture.

If the robot is at a wall A and it receives a goal point in a close range but in a wall B, perpendicular to A, then we are in the presence of a corner. The narrow space will not allow the robot to turn in-place, so the planner will first distance itself from the wall, performing then a maneuver to move towards

the goal. This maneuver is illustrated in Fig. 10.

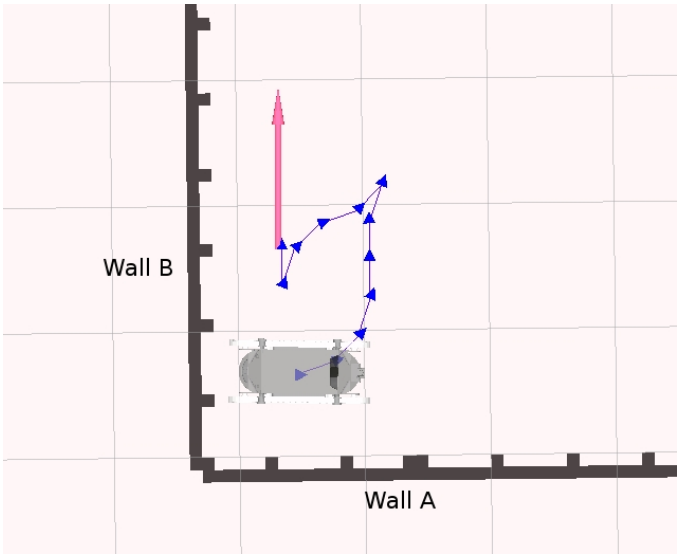


Fig. 10. Example of a corner maneuver

B. Layered Obstacle Expansion

In Fig. 11, at the top, layers 0 (0°, cyan) and 1 (22.5°, pink) are represented. At the bottom, layers 3 (67.5°, green) and 4 (90°, yellow). It can be verified how in layer 0 (if robot's heading is 0°) than it is able to enter the doors, while in layer 4 (robot's heading is 90°) it is incapable of such.

If the outer radius was used to inflate the walls, the robot would never been able to enter the doors, while if the inner radius was used it would always assume free space and would cause collisions for certain configurations.

Fig. 12 illustrates a path for a goal in layer 0 (orientation 0°) as well as the obstacle inflation for layers 0 (cyan), 4 (yellow), 14 (orange) and 15 (red). The planner is able to find a path because the goal point is free in layer 0 (point is outside cyan area). If the point were to be in layers 4, 14 or 15, the planner would fail to compute a path because the point is inflated in such layers and therefore is untraversable. From the image can be concluded that the planner respects the restrictions from the obstacle expansion, not entering areas that its heading does not allow.

The layered obstacle inflation allows the planner to construct paths close to obstacles for certain configurations while keeping the robot away from them in configurations that would lead to a collision. These layers also allow an efficient collision checking, eliminating the need for a full collision checking.

C. Path Following

In Fig. 13, a representation of the path followed by the robot (green line) over the planned path (blue line) can be viewed in a 0.025m resolution map. The path was followed backwards and the tolerance was 0.05m for distance error and 5° for angle error. The controller was able to follow closely the planned path, stopping within tolerance, at 0.0474m and $-0.134°$ from the goal.

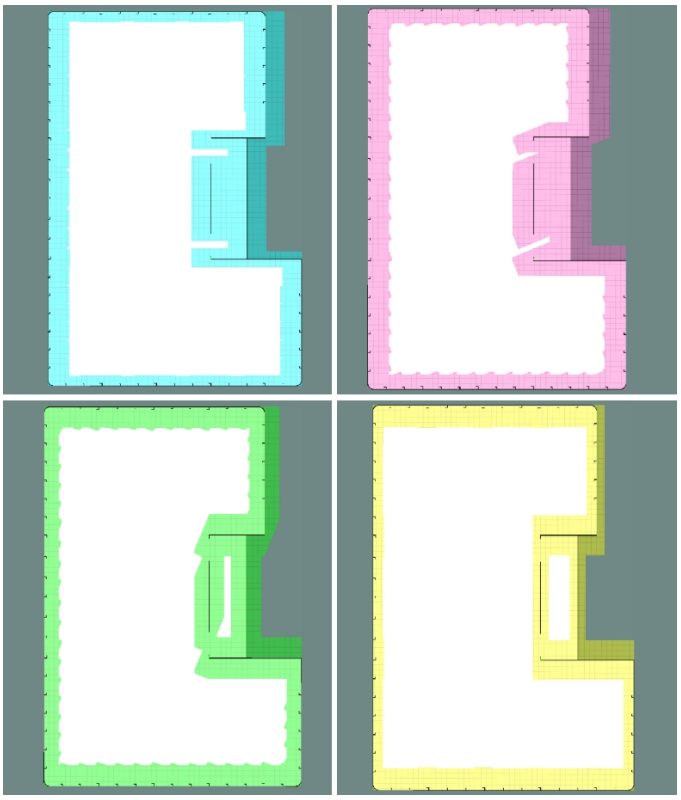


Fig. 11. Obstacle inflation for different orientations

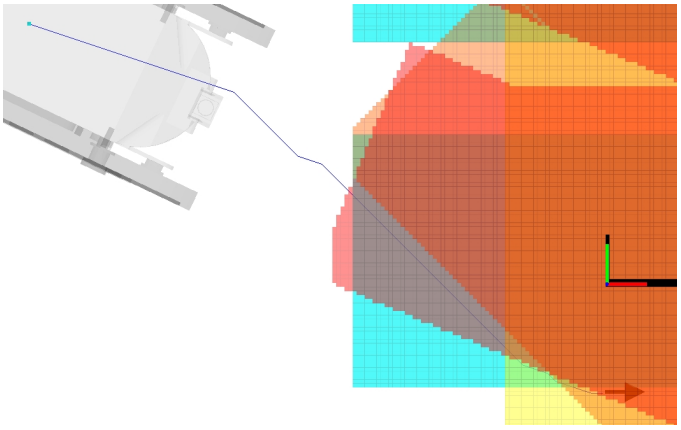


Fig. 12. Paths close to obstacles

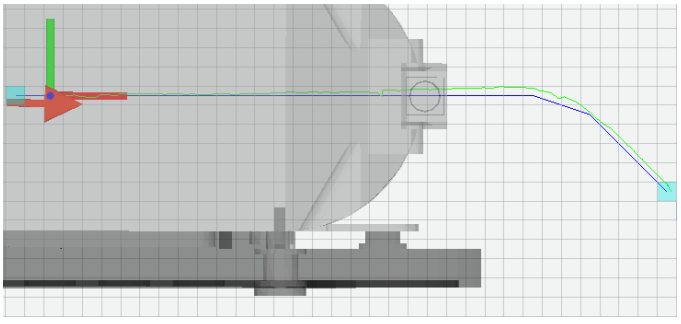


Fig. 13. Path followed by the robot

VI. Conclusions

This paper presented the first developments on an extension over the A* algorithm, where besides the 2D position of the platform, its orientation was also considered, resulting in a smooth, feasible path. The actual footprint of the robot was considered in order to avoid being too conservative or generating infeasible paths, working for any-shape robots. In order to improve collision checking and reduce the state space, the oriented footprint of the robot was used to inflate the obstacles only in the orientation of the motion, allowing the use of the actual footprint of the robot without increasing the complexity of the collision checking, enabling the planner to compute paths close to obstacles. Simulation tests demonstrate that the planner is capable of creating paths even in narrow spaces, such as close to walls and inside corners.

Acknowledgment

The authors would like to thank everyone involved in the CARLoS Project. This project has received funding from the European Commission Seventh Framework Programme for Research and Technological Development under the grant agreement number 606363 and from the national project "NORTE-07-0124-FEDER-000060".

References

[1] M. Bibuli, G. Bruzzone, M. Caccia, A. Ortiz, T. Vogele, M. Eich, L. Drikos, Y. Koveos, E. Kolyvas, F. Spadoni, A. Vergine, K. Tanneberger, A. Todorova, I. Gaviotis, and V. Apostolopoulou, "The minoas project: Marine inspection robotic assistant system," in *Control Automation (MED), 2011 19th Mediterranean Conference on*, June 2011, pp. 1188–1193.

[2] M. Eich and T. Vogele, "Design and control of a lightweight magnetic climbing robot for vessel inspection," in *Control Automation (MED), 2011 19th Mediterranean Conference on*, June 2011, pp. 1200–1205.

[3] L. Menegaldo, M. Santos, G. Ferreira, R. Siqueira, and L. Moscato, "Sirus: A mobile robot for floating production storage and offloading (fpso) ship hull inspection," in *Advanced Motion Control, 2008. AMC '08. 10th IEEE International Workshop on*, March 2008, pp. 27–32.

[4] L. Deschamps and C. Greenwell, *Integrating Cost Estimating with the Ship Design Process*. SPAR Associates, Inc.

[5] A. Hornung, M. Phillips, E. Jones, M. Bennewitz, M. Likhachev, and S. Chitta, "Navigation in three-dimensional cluttered environments for mobile manipulation," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, May 2012, pp. 423–429.

[6] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *I. J. Robotic Res*, 2009.

[7] B. MacAllister, J. Butzke, A. Kushleyev, H. Pandey, and M. Likhachev, "Path planning for non-circular micro aerial vehicles in constrained environments," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, May 2013, pp. 3933–3940.

[8] D. Ferguson , M. Likhachev, and A. T. Stentz, "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)*, June 2005.

[9] J. P. Gonzalez and M. Likhachev, "Search-based planning with provable suboptimality bounds for continuous state spaces," in *SOCS'11*, 2011, pp. –1–1.

[10] M. Pivtoraiko and A. Kelly , "Efficient constrained path planning via search in state lattices," in *The 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, September 2005.

[11] A. K. andBryan Nagy, "Reactive nonholonomic trajectory generation via parametric optimal control," *The International Journal of Robotics Research*, vol. 22, no. 7-8, pp. 583–601, 2003.

[12] M. Dakulovic, C. Sprunk, L. Spinello, I. Petrovic, and W. Burgard, "Efficient navigation for anyshape holonomic mobile robots in dynamic environments," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, Nov 2013, pp. 2644–2649.

[13] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," *ICRA workshop on open source software*, vol. 3, no. 3.2, p. 5, 2009.