We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

15

Object Path Planner for the Box Pushing Problem

Ezra Federico Parra-Gonzalez and José Gabriel Ramírez-Torres
Cinvestav-Tamaulipas
México

1. Introduction

Researchers generally agree that multi-robot systems have several advantages over single-robot systems (E. Parker & Tang, 2006). The most common motivations for developing multi-robot system solutions are that:

1. The task complexity is too high for a single robot to accomplish.
2. The task is inherently distributed.
3. Building several resource-bounded robots is easier than having a single powerful robot.
4. Multiple robots can solve problems faster using parallelism.
5. The introduction of multiple robots increases robustness through redundancy.

The *box-pushing problem*, as defined in (Gerkey & Mataric, 2002), consists to cooperatively move a box, relatively large when compared to the size of the robots, from an initial position to another goal location using robots that can only make pushing movements.
This problem has many practical applications, among them we can emphasize the next ones:

– Rescue of people.
– Carrying out dangerous works.
– Automatization of industrial processes.
– Carrying out attendance works.
– Movement of heavy objects.

Nevertheless, the most motivating aspect of this problem is that it represents an interesting challenge for the development of coordination schemes in multiagent systems. Indeed, the displacement of the object cannot be realized if cooperation between the different members of the community does not exist.
The *box-pushing problem* is related to the well-known "Piano Mover's Problem"(Schwartz & Sharir 1983) (Gerkey et al., 1995): given an arbitrary rigid polyhedral environment, find a continuous collision-free path taking this object from a source configuration to a desired destination configuration. In the *box-pushing problem*, the object B must be moved by N mobile robots toward the goal position T, with pushing movements only. Then the fundamental research problem is to design the appropriate control laws in order to obtain the desired global behavior by the community of robots (Figure 1). It was shown (Reif, 1979) that this problem is PSPACE-hard, which implies NP-hard.

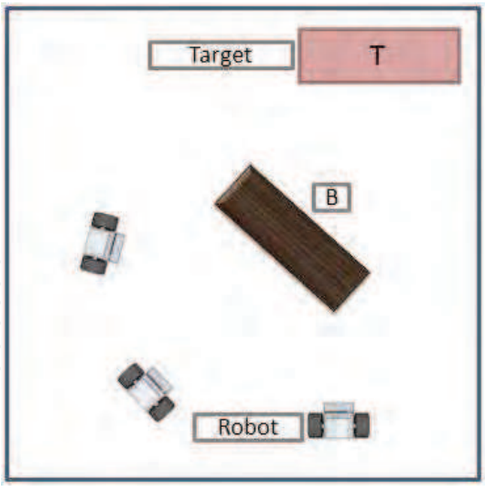


Fig. 1. Basic scheme of the "box-pushing" problem.

The object transportation task made by a cooperative community of several mobile robots involves different problems, for example: collisions avoidance, stable manipulations, path planning for robots and for the object, etc. In the most complex cases robots will have to manipulate the object and to change not only their position, they will have to change also their orientation in order to be able to move the object in the narrower places of the environment.
In this chapter we review some of the outstanding works that has been done in the area and we present a new strategy (collisions avoidance, stable manipulations, object path planning, etc.) to solve this problem. The solution that we design was inspired from the wavefront algorithm but includes certain original considerations which help to obtain trajectories that facilitate the movement of the object by unspecialized robots.

2. Related work

At present, there exist several approaches to the object transportation problem by a community of mobile robots. For example, some works have been developed so all communication needs are omitted and replaced by behavioral mechanisms based on local information (Yamada & Saito, 2001), on the other hand, there are works in which the communication is fundamental to improve the behavior of multi-robot systems (Muñoz Melendez & Drogoul, 2004).
There are some well-known strategies like the "Object closure" (Wang & Kumar, 2002; Guilherme A. S. et al., 2002) where the object position can be controlled by a team of robots that surround the object, so the box position will be controlled by the movement of each robot that surrounds and pushes the object. Another common approach is the "Pusher-Watcher" (Gerkey & Mataric, 2002; Kovac et al., 2004) where a robot (watcher) beholds the movement of the object, while coordinating the operations of the robot team (pushers) that physically manipulate the box.
Some recent works use the "swarm intelligence" model, (Li & Chen, 2004) where self-organized systems of homogeneous robots are built around simple behaviors, obtaining a decentralized and intelligent global behavior. Also, some approaches are based on Artificial Intelligence tools, as the reinforcement learning or "Q-Learning" (Wang & W. de Silva, October 2006). For more complex task of manipulation, some authors propose the use of specialized robots in manipulation tasks (Gupta & Huang, 2003; Inoue et al., 2007).

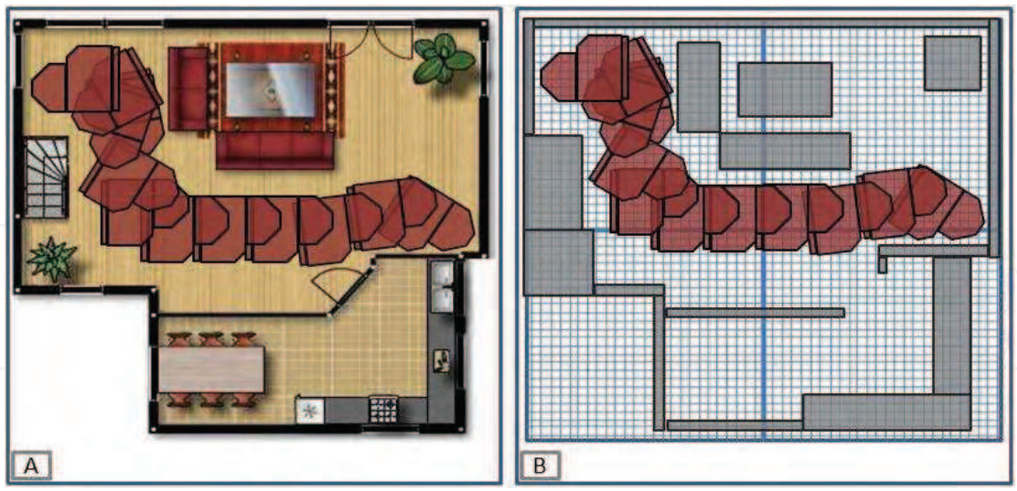


Fig. 2. Path example for moving an object from a starting point to another final.

The proposed strategy in this document obtains a feasible path for the object, computing a discrete representation of the configuration-space (C-space). There exist some works using a similar approach (Gene et al., 2005; Jed et al., 1990; MIYOSHI et al., 2008), but in addition, our strategy focus in obtaining a feasible path for the object, such the displacements (maneuvers or *reconfigurations*) of the pushing robots around it is kept to a minimum.

3. Object path planner

The path planning problem can be defined as follows: given an initial and a final configurations, find a feasible continuous collision free trajectory between them (Figure 2). Path planning algorithms are responsible for maintaining many essential aspects of plausible agent behavior, including collision avoidance and goal satisfaction. Path planning also consumes a significant part of the computation time for many simulations, particularly in highly dynamic environments where most of the agents are moving at the same time.

The main objective of this work is to compute trajectories for objects that will be moved by communities of not specialized mobile robots in cluttered environments[1]. The fundamental contribution is the development of new solving strategies for the collision-free path-planning problem for the object, that take into account some criteria in order to allow the robots to carry out the task precisely and faster.

When using unspecialized robots, each change of direction during the displacement of the box requires that the robots modify their position around the object before starting a new pushing movement (robot reconfiguration). These reconfigurations require certain time to be completed; if the total number of reconfigurations is small, the time required for the displacement is improved.

In this work we will consider that robots are not specialized in manipulation tasks, in other words, they only can push the box, but not to pull it. Therefore, it is important to obtain trajectories for the object that, besides being a solution to the path planning problem, the number of robot reconfigurations during the movement tend to be minimal. This is obtained

[1] A cluttered environment will be considered as a complex work area that contains many obstacles.

Fig. 3. Polygonal description of the object and the obstacles in a same coordinate system.

finding trajectories that reduce the number of changes of object direction, that is to say, trajectories that have a greater "continuity".

4. Proposed method

The proposed approach computes a discrete representation of the configuration space (C-space) for the box in the cluttered environment. In this representation, an heuristical search is performed, in order to obtain a trajectory with the minimal number of changes of direction. The main characteristics of the solution are listed below.

1. Find the shortest trajectories.
2. Look for trajectories with the smaller number of robot reconfigurations.
3. Preference to certain box rotations, in order to improve the pushing points.

The method is realized in four main stages: 1) Obtain the correct space representation from a predefined map; 2) Find intersection between polygons (collision detection); 3) Build the C-space in which valid positions (free of collisions) and invalid positions (collision) are split; 4) Generate the object path trajectory using alternative heuristics which reduces the robot reconfiguration and look for the shortest paths.

4.1 Object and environment representation

The environment and the box are represented by a set of convex polygonal objects, each described as a set of linear constraints (see Equation 1):

$$a_i x + b_i y \leq d_i, i = 1...P_k \tag{1}$$

where P_k is the number of faces of object k and for each $(a_i, b_i); a_i \neq 0$ or $b_i \neq 0$.
A representative example of this polygonal representation is shown in (Figure 3).

4.2 Finding intersections between polygons (collisions)

We need a method to verify whether there is an intersection between 2 given polygons. This can be achieved if we find at least a segment of a line which satisfies the sets of constraints of both polygons.

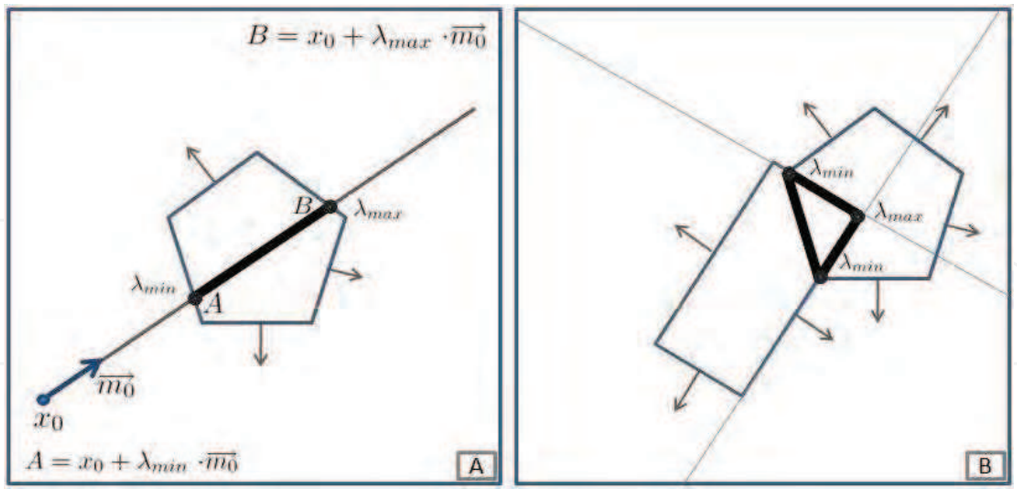


Fig. 4. A)Obtaining a segment of a line that satisfies the polygon's restrictions. B) Example of 3 segments that satisfy both sets of constraints of the polygons.

The general concept to make this evaluation is the following one: Given a line, defined by a point x_0 and a vector $\overrightarrow{m_0}$, we can compute the intersection between this line and a set of linear constraints by finding the intersection of the line with each single constraint and use them to determine the limits of the segment (figure 4A).
To verify the intersection between two polygons we use the previous method as follows. Given 2 polygons verify for each single constraint $n_i x \leq d_i$ if there is a segment on the line $n_i x = d_i$ (edges) which verifies both sets of constraints at the same time.
In figure 4B an example of an intersection between two polygons is shown, as we can see there exist three edges over the constraints that satisfies the restrictions of both polygons.

4.3 C-space and wavefront algorithm

Assuming we have the mathematical representation of all the involved objects and the capacity to identify if an intersection exist between two polygons, the following step is the formal construction of the C-space.
From the polygonal representation of the environment, the free space has to be computed, since the trajectory must belong to it. For this, we use a discrete representation of the configuration space. The C-space is the set of all possible configurations that an object (even articulated) can have in a given space (Lozano-Pérez, 1983).
Computing the precise limits of the free configurations is a difficult problem (Reif, 1979; Reif & Sharir, 1985). A very common representation of the C-space is to consider only the configurations on a grid, obtaining a discrete C-space. The configuration of the object is defined by the 3-tuple (x, y, θ) that represents the degrees of freedom of the box. The pair (x, y) is the position of a fixed point on the object, and θ is the orientation of the box and will take values in interval $[0, 2\pi]$ (Figure 6) The main advantage of the C-space representation, is that the objet is reduced to a single point (the configuration), thus simplifying the motion planning.
Each voxel of the C-space 3D matrix represents a given configuration of the box. This matrix is filled using the collision detection algorithm in order to obtain a binary description of both free and occupied spaces. In the figure 5B, the binary layer corresponding to figure 5A is presented.

The precision of C-space is linked to the amount of configurations that were defined in the work area (discretization resolution), nevertheless, a greater precision will not only affect the computational times, it will also affect the storage requirements.
In order to obtain feasible trajectories, a virtual size expansion is applied to the object. Indeed, since the object must be pushed along the whole trajectory, a minimal distance between the object and the obstacles must be guaranteed, so the robots can always push the box suitably. In Figure 7 an example of the virtual object expansion is shown, as we can see, before start the calculation of the C-space, the object is expanded in such a way that the new size includes the robots size that surround it.

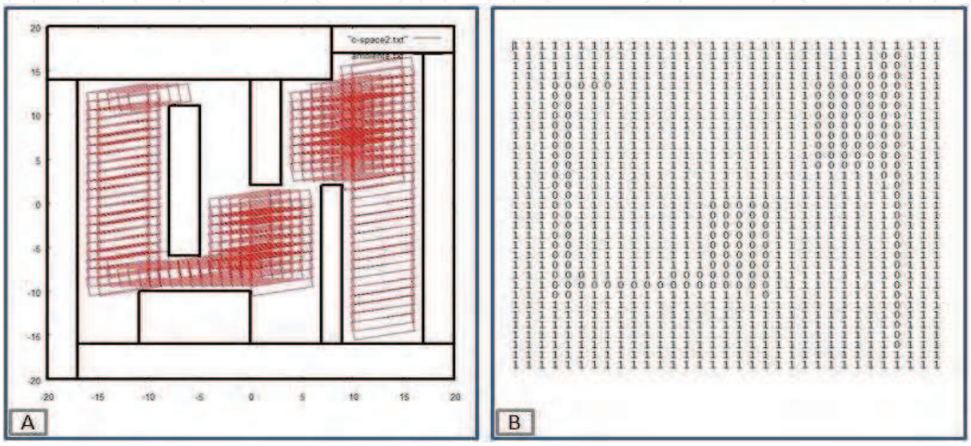


Fig. 5. A) Graphical representation of the C-space when the box is rotated 10 degrees; the positions where the box is not in collision are drawn. B) C-space binary layer that corresponds to figure A.

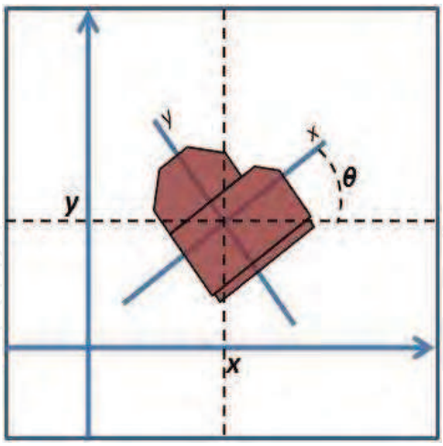


Fig. 6. Configuration of the object.

Once the C-space resolution has been established, the construction is made by verifying in each cell if a collision exist between the object and, at least, one of the obstacles of the environment. This action is made by testing each voxel [x][y][c] of the matrix by moving the object to the (x,y) position and rotate (c * resolution of rotation) degrees. Finally it is necessary to apply the collision detection function in order to verify if the configuration is collision-free ([x][y][c] = 0) or if a collision exists ([x][y][c] = 1) with any obstacle.

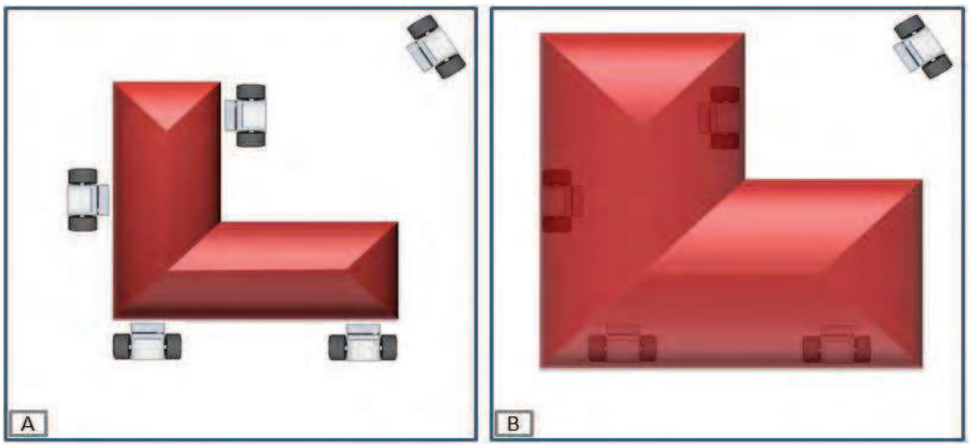


Fig. 7. Sample of the virtual object expansion.

Because there exists a direct relationship between solution quality and the size of the "C-space" (bigger size allows to find more precise paths) it is important define the correct size for the "C-space", however, there exist also a relationship between the size of "C-space" and computing requirements (a bigger size needs a larger number of intersect evaluations) then we need to improve the performance of the "C-space" construction by reducing the number of intersect evaluations in order to have better results in less time.
To reduce the number of intersect evaluations we apply a preprocessing based on the concept of " Bounding boxes" introduced by Cohen (Cohen et al., 1995) which basically works getting the square of minimum area that contains the object (polygon) within it. This action allows determine the possibility of having an intersection between two polygons and decide if is necessary to evaluate intersections with other objects or skip this step. This preprocessing technique has prove to perform well in practice (Suri et al., 1998) and in our tests showed his efficiency by reducing in some cases the 50 percent of the intersections evaluations.
As a result of this step we will have a binary 3D matrix that represents the discrete C-space, where each voxel represents if a given configuration of the object is in collision with the environment or not.
With a 3D-grid representation of the free configurations, is natural to apply an artificial potential field to compute the feasible paths. The wavefront algorithm is widely used for path planning of mobile robots: The algorithm determines if a free collision trajectory exists or not, unlike methods based on classical potential fields, the wavefront algorithm does not have local minima and it guarantees to find the shortest path between the origin and the destiny.

- The algorithm determines if a free collision trajectory exists or not.
- Unlike methods based on classical potential fields, the wavefront algorithm does not have local minima.
- This method guarantees to find the shortest path between the origin and the destiny under the chosen neighborhood criteria.

The construction of a wavefront, according to LaValle (LaValle, 2006), is made in the following way. Suppose that the minimum common cost between each state is i, and having a set of states organized in a wavefront, W_i. The start point of the wave front is W_0 and it represents the objective state X_G. The algorithm will assign the cost of optimal move of 1 for all the states that can be reached from W_0 in a single step. The states that receive cost of 1 can be

organized in a wavefront like W_1. The unexplored neighbors of W_1 will have an assigned cost of 2. This process will be repeated from i to $i+1$ until all the reachable states have been visited. All the unreachable states will be assigned with the value of $\phi(x) = \infty$. The optimal cost to move from a state to another one will be calculated in time $O(n)$, where n is the number of reachable states.
When the wavefront is computed, the robot must follow the descendent gradient selecting the neighboring state with the smaller cost, this process will be repeated until reaching the objective state X_G:

$$u^* = argmin_{u \in U(x)} \left\{ \phi(f(x,u)) \right\}$$

where $U(x)$ is the set of actions (movements) neighboring to the present state x towards which the robot can be moved. u^* represents the following state and $f(x,u)$ is the transition function of action for the action u applied to the state x.

4.4 Path generation

With an artificial potential field, there exist many different solutions to the object path planning problem. In order to find a well suited solution for the box-pushing problem we have developed three different heuristics to find trajectories where the number of robot reconfigurations are minimal. These strategies are based on the concept of "reaching measure". The reaching measure is basically the number of cells (voxels) of the C-space that are valid (decreasing value and free) following a specific direction. The path segment with the greatest reaching measure is used to construct the trajectory.

4.4.1 Trajectory computation with reaching measure (TCMR)

As was stated before, working with non specialized mobile robots (in pushing task) it is necessary, along the path, to modify their position respect to the box (reconfiguration) to be able to obtain a valid pushing position.
The reduction of robots reconfigurations throughout the trajectory, allows in first instance to guarantee an important reduction of time, simply because the change of pushing points is a process that needs some time to be completed. Another aspect that can be improved when the robots reconfigurations are reduced is the increase of the precision of the movements. The continuous movements of the robots around the box can increase the possibilities of having collisions among them, and of course, the increase of the odometry errors by the constant movement of the robots can affect the precision of the pushes.
Our first strategy is a greedy approach, trying to reduce as much as possible the number of robot reconfigurations using the concept of "reaching measure", choosing among the possible directions of movement from a given configuration, the one which allows to move closer to the goal configuration.
This heuristic is based on the conventional "wavefront algorithm" and works in the following way: Once the wavefront matrix is computed, starting from initial configuration, validate all the neighbors (north, south, east, west, up and down). If at least 2 neighbors had an smaller value than the value of the starting point, the next step would be to measure the reach of all the possible routes, following the direction of these neighbors. For instance, if we start from a defined point with a value of 99 and it has only two neighbors with smaller values, west with a value of 98 and up with a value also of 98, the following step is to evaluate and count the strictly decreasing cells in the up and west directions. Finally we compare the number of

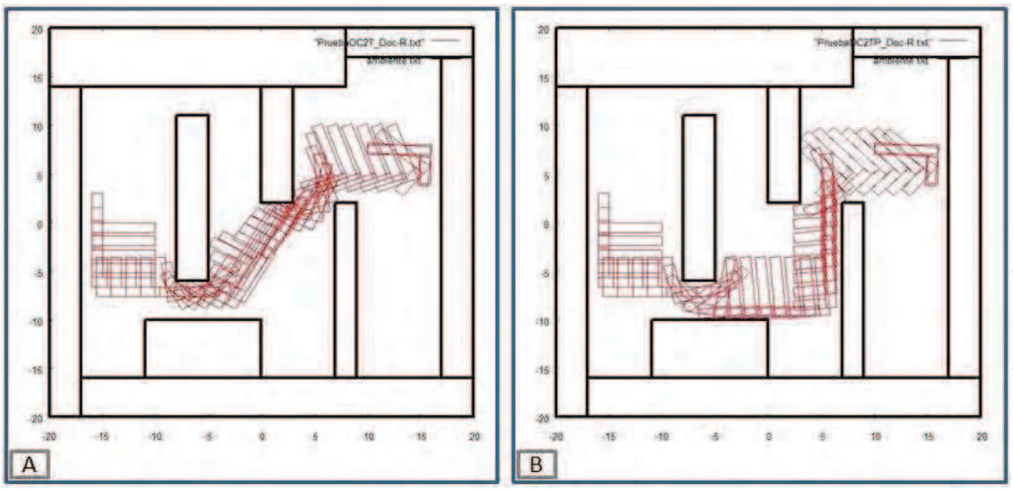


Fig. 8. A) Graphical representation of the box trajectory obtained without any heuristic to select the neighbors. B) Graphical representation of the box trajectory obtained with the proposed heuristic to reduce the reconfigurations.

squares in both directions and follow the one that had the greatest value (moving the robot closer to the goal).
In Figure 8A we can see a sample of a path obtained without using any heuristic, it was calculated by selecting the first cell neighbor that lead us closer to the objective. In Figure 8B we can appreciate another sample path but calculated using the proposed heuristic. If we compare both trajectories we can verify that using the proposed approach we can reduce the changes of the box direction and also reduce the number of required robot reconfigurations.

4.4.2 Trajectory computation with layer validation (TCLV)

Since the obtained trajectories are composed by vertical, horizontal movements and pure rotations, there may be some orientations that are best suitable for pushing than others, given the geometry of the box (e.g. 0, 90, 180 and 270 degrees for a rectangular box). This strategy is similar to the previous one, but the path is generated so these particular orientations of the box are prefered. This improvement will help to obtain a greater precision in the pushing task. Compared with the first heuristic (TCMR) that use the "reach measure" to reduce as most as possible the maneuvers of the object, this alternative uses again the "reach measure" but with some modifications that helps to construct paths having some specific object rotations.
The difference between "TCMR" and this alternative is that "TCLV" will repeat the "reach measure" every time the algorithm detects that it is being evaluated a cell that pass over a privileged layer. This action allows to find new longer path segments on certain layers in spite of following an specified direction. Another modification is to give an extra values to some layers (or even cells), this action helps even more to select some specific "C-space" layers.
Applying these changes helps to find alternative paths that in most cases uses the layers that we privileged, but the fact of place new measurement points and give extra values to some cells can increase (sensibly) the quantity of robot reconfigurations.
For example: If we were paced in a square that has 3 possible directions to follow: north, west and up and after making the neighbor measurement we found that the reach of the routes is: north 21, west 19 and up 11. If we do not privilege any layers clearly the route to follow would be the one that has the north direction. The proposed change is to validate if

the route that is going by the superior layer pass through some of the privileged layers, if it occurs, the value can be increased. In the present implementation the value of the route is duplicated ($11 * 2 = 22$), that is why the route with up direction will be followed to construct the final route. With this, the algorithm can conclude that the trajectory can be pursued with the prefered orientation.

4.4.3 Route graph (RG)

The fist two approaches are greedy so in order to find better solutions, the third strategy uses a graph representation of possible trajectories. This change allows to evaluate more possible trajectories before selecting one. In this *route graph strategy*, each node represents a change of direction of the object. Within this graph, a search is done to identify the path with the minimal number of reconfigurations. In Figure 9, an example of the route graph obtained from a configuration space marked with the wavefront algorithm is shown.
The route graph is constructed as follows (c.f. Algorithm 1):

- Start from the initial configuration and add it as the root node n_i of the route graph.
- Compute the reaching measures for all possible directions of movement.
- The reachable configurations are added as nodes, linked to the root node.
- Steps 2 and 3 are repeated, for each of the new nodes, recursively.
- The construction of the route graph is finished when all the branches converge to the final configuration n_f.

Algorithm 1 "Route graph" strategy

Input: Wave front matrix and origin node n_i
Output: Path from n_i to n_f (if it exist)
$OriginPos \leftarrow$ origin node n_i
Insert $OriginPos$ node on $node[0]$
$x \leftarrow 1$
while $node[x] \neq \emptyset$ **do**
  $OriginPos \leftarrow$ position of $node[x]$
  for All the neighbors of $node[x]$ **do**
    if Neighbor of $node[x]$ is valid **then**
      $dir \leftarrow$ direction to reach the neighbor of $node[x]$
      $NeighborPos \leftarrow$ position of the neighbor of $node[x]$
      while Neighbor of $NeighborPos$ on direction dir is valid **do**
        $NeighborPos \leftarrow$ position of the neighbor of $NeighborPos$
      end while
      Insert node $NeighborPos$ on $node[]$
    end if
  end for
  $x++$
end while
return Route graph path
Generate the path based on the final node of smaller depth

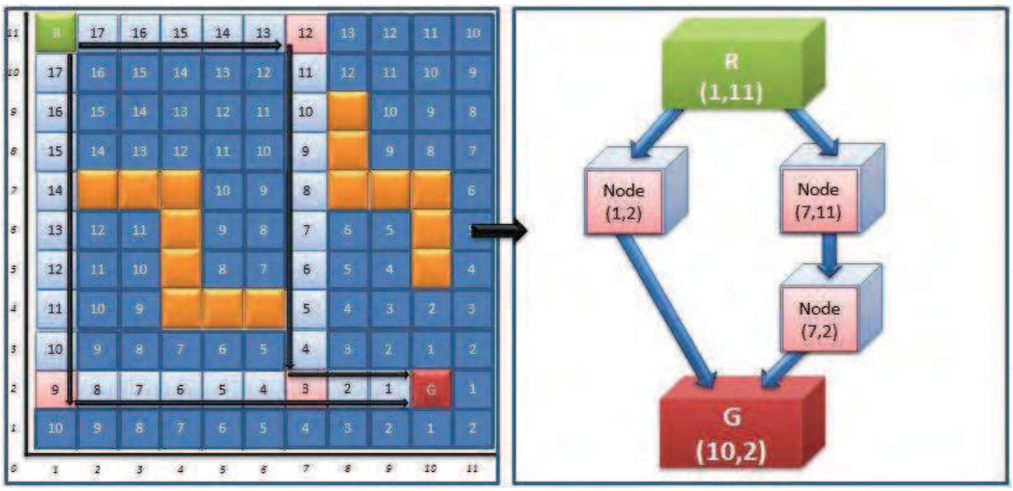


Fig. 9. Example of the route graph construction.

4.4.4 Modified A* algorithm (A*RR)

In order to realize the pertinent comparisons of the proposed heuristics, we implement a modified version of the A-Star algorithm (A*). This algorithm was presented in 1968 by Hart (Hart et al., 1968), and is extensively used for path planning.

The A* algorithm combines the capacities of the Dijkstra algorithm with some characteristics of the breadth-first search (BFS) algorithms, since it looks for the shortest routes like Dijkstra does and in addition it includes an heuristic as in the BFS to quickly converge to the objective. The efficiency secret of the A* algorithm is that it combines pieces of Dijkstra's information (favoring the selection of nodes near the origin) with BFS's information (favoring nodes that are close to their destination). In the A* standard terminology $g(n)$ represents the cumulative cost of the path from the initial position to any node n, and $h(n)$ represents the estimated cost by the heuristics from n to the final position. The algorithm balance the two values as it moves from origin to destination and maintains two lists: one open and one closed where the closed list stores the nodes that have been already evaluated, and the open list contains the nodes that have not been. Each time that the algorithm starts a new cycle, it moves the current position to the node n that has the lower $f(n)$ ($f(n) = g(n) + h(n)$).

The alternative A* heuristic that we propose works by replacing the function $g(n)$, which now will have an additional cost associated to the amount of changes of direction and not to the distance cost from the origin to n (as happen in the classical A* heuristic).

In the Figure 10 we can compare two paths that uses the two different A* heuristics. In Figure 10.A, we shown a route obtained by using the classical A* heuristic. In Figure 10.B we can see the resulting path using the A* alternative cost function. As we can see, the classical heuristic built a shorter trajectory but it requires 18 changes of direction to transport the object, while the modified A* algorithm finds a path with less reconfigurations (only 9), but longer.

5. Tests and results

In this section we will tackle the related aspects to the final implementation of the proposed methods and we will present the obtained results of the testing set.

Fig. 10. A) Path planning with classical A* algorithm. B) Path planning with modified A* algorithm.

5.1 Description of the tests

The total set of tests was constituted by the computation of 2539 random routes divided in 5 different test environments and 2 different resolutions for the C-space. For the tests we implemented 7 heuristics: our 3 proposed heuristics, as well as 4 classical heuristic for comparative effects. The implemented heuristics are the following:

1. Wavefront selecting randomly the neighbor (WRN): Each time it moves from a cell, this algorithm randomly select the neighbor to follow (from all possibles).
2. Wavefront with predefined evaluation of neighbor (WPN): Each time it moves from a cell, this algorithm evaluates the possible neighbors following a predetermined order and select the first valid.
3. Classical A* algorithm (A*C): It uses the classical heuristic to reduce distance.
4. Modified A* algorithm (A*RR): It uses the proposed A* heuristic to reduce maneuvers.
5. Trajectory computation with reaching measure (TCMR)
6. Trajectory computation with layer validation (TCLV)
7. Route graph (RG)

The heuristics 1, 2 and 3 are oriented to reduce distance and the rest to the reduce reconfigurations. To cover a wide range of test cases were developed five different environments that vary in complexity, number of obstacles and in the percentage of free space (Figure 11).

The first test environment (Figure 11(a)) was designed to test that the routes are able to displace the object while the box is going away from the target area. This is useful to evaluate that the heuristics do not fall into local minima.

The second (Figure 11(b)) and third environments (Figure 11(c)) were designed with narrow corridors to test that despite having a smaller number of possible solutions, the heuristics are able to construct valid routes. It is also important to mention that this type of environments are very complex to solve by some techniques, for example those based on Probabilistic Road Maps (PRM).

The 4th environment (Figure 11(d)) is symmetric and with a higher percentage of free space. This environment was created to compare the strategies, those that reduces the distance against those that reduce the maneuvers. The central corridors allow the construction of shorter routes, but more complex.
In the fifth environment (Figure 11(e)) were added small obstacles and placed uniformly on the map, this was designed to evaluate the strategies in environments that have multiple solutions.
The chosen metrics to compare the strategies were:

– Number of reconfigurations: After the paths were computed, the number of changes of direction were counted. This value represents the number of necessary robot reconfigurations for the trajectory.
– Time: Related to the computing time required by each heuristic to find the paths.

The statistic metrics used to analysis the results are:

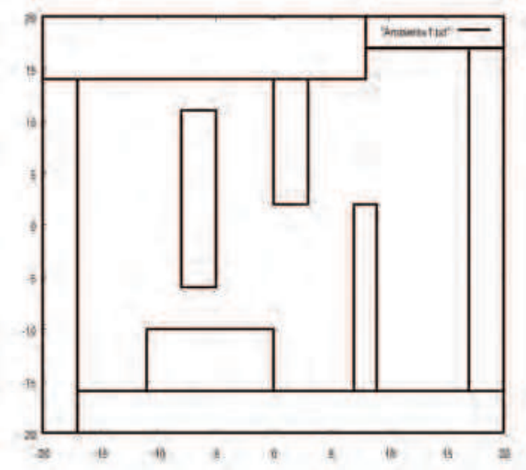
– Average value
– Max value
– Minimum value
– Standard deviation

It is important to state that for our approach it is more attractive obtain trajectories with a smaller number of direction changes. Tables 1 and 2 shows the results for one of the set of tests.

Method	Average	Max val	Min val	Standard dev.
WRN	22.08	51	4	11.02
WPN	12.16	27	2	8.06
A*C	13.1	40	3	8.65
A*RR	6.98	14	3	2.29
TCRM	5.84	12	2	2.39
TCLV	5.94	13	2	2.4
GR	5.2	10	2	1.92

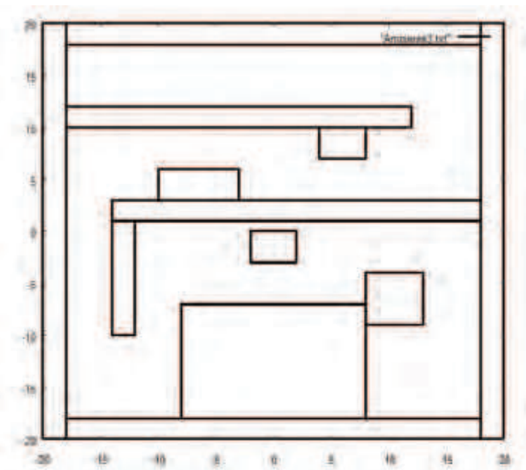
Table 1. Comparison of path planning methods by number of reconfigurations

As we can see (Table 1), the heuristics “WRN”, “WPN” and “A*C” were not competitive compared with the other four strategies. Nevertheless, this situation is congruent because these heuristics do not include any strategy to reduce the number of changes of direction, which is the case of the other 4 strategies.
The results show that the “GR” have the best overall results, however, it was overcome by the heuristic “A*RR” in the situations where to find the paths that have the minimum number or reconfigurations it is necessary to move the object away from the objective (see Figure 10). This situation is consistent because the proposed heuristics use the wavefront matrix to build their solutions, then it is impossible for the heuristics find the trajectories that moves the object away from goal.
In Table 2 we show the computing time needed by the heuristics to build a path. The time values in the table are in milliseconds.
The strategies bases on “A*” algorithms were affected in computing time, this is because they must handled and stored many information to calculate the trajectories. As we can see in

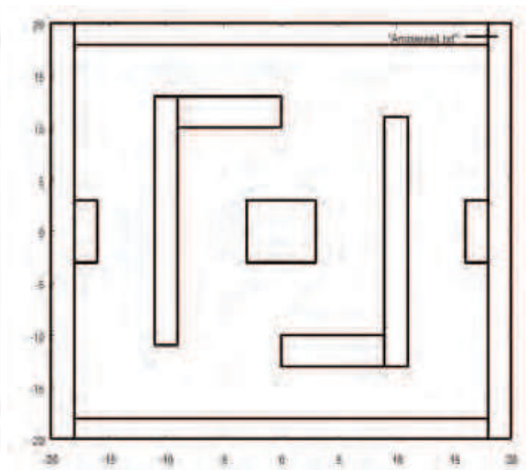


(a) Environment 1 (44.4% free space)

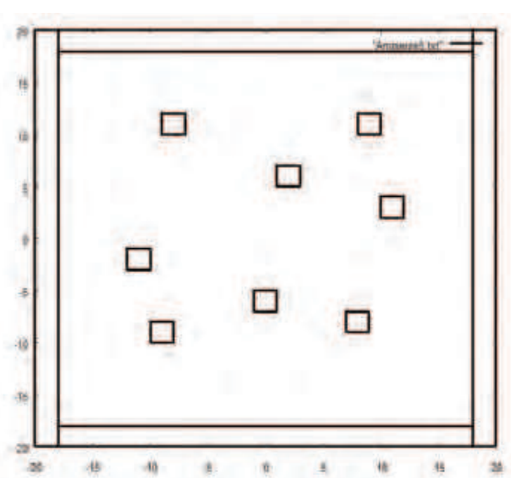
(b) Environment 2 (49% free space)



(c) Environment 3 (56.4% free space)



(d) Environment 4 (67.9% free space)



(e) Environment 5 (79% free space)

Fig. 11. All test environments used in tests.

Method	Average	Max val	Min val	Standard dev.
WRN	375.08	377	375	0.34
WPN	375	375	375	0
A*C	24948.3	272625	1	53531.45
A*RR	1174.34	10062	1	2109.2
TCRM	375.02	376	375	0.14
TCLV	375.02	376	375	0.14
GR	421	438	406	9.78

Table 2. Comparison of path planning methods by computing time (*ms*)

Table 2 the best computing times correspond to the heuristics "WRN", "WPN", "TCRM" and "TCVL", in that order. In this case the results favor the greedy strategies because they do not store any information and they just make desitions based on local information. Nevertheless, the time for the heuristic "GR" was very competitive and give better results for the number of reconfigurations. Thus, the "GR" strategy shows the best compromise between computing time and solution quality.

5.2 Discussion and conclusions

According to the numerical experiments, we can conclude that, although exist situations in which the "A*RR" algorithm can obtain trajectories with smaller number of changes of direction (thus, better results), the proposed heuristics "TCRM" , "TCVL" and particularly the "GR" obtain a better balance between quality of the solution and the required computing effort.

It is important to emphasize that the specialization of heuristic "GR" had a slight increase in the computing time. Nevertheless, we can conclude that the balance between number of reconfigurations, length of trajectories and computing time clearly benefits to heuristic "Route Graph".

6. Future work

In future works the resulting paths will be used to determine (by an optimization strategy) the best combination of robots and pushing points in each sub trajectory to reduce the travel distance for each individual robot during the reconfiguration phase.

Also, we would like to consider the problem of advantageous location of the robots, in order to place the robots which are not participating in a sub trajectory in a favorable position for another sub trajectory. Our main goal is to obtain a coordination strategy able to conclude the task in an efficient way.

7. References

Cohen, J. D., Lin, M. C., Manocha, D. & Ponamgi, M. (1995). I-collide: an interactive and exact collision detection system for large-scale environments, pp. 189–ff.

E. Parker, L. & Tang, F. (2006). Building multirobot coalitions through automated task solution synthesis, *Proceedings of the IEEE* 94(7): 1289–1305.

Gene, E. J., Tong-Ying, J., Jun-Da, H., Chien-Min, S. & Chih-Yung, C. (2005). A fast path planning algorithm for piano mover's problem on raster, *Advanced Intelligent Mechatronics. Proceedings, 2005 IEEE/ASME International Conference on* pp. 522–527.

Gerkey, B. P. & Mataric, M. J. (2002). Pusher-watcher: An approach to fault-tolerant

tightly-coupled robot coordination, *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on* 1: 464–469.

Gerkey, B. P., Mataric, M. J. & Simsarian, K. (1995). Cooperative multi-robot box-pushing, *Proceedings, IROS-95* pp. 556–561.

Guilherme A. S., P., Vijay, K., John, S., Camilo J., T. & Mario F. M., C. (2002). Cooperative transport of planar objects by multiple mobile robots using object closure, *in Experimental Robotics VIII* pp. 275–284.

Gupta, A. & Huang, W. (2003). A carrying task for nonprehensile mobile manipulators, *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on* pp. 2896–2901.

Hart, P., Nilsson, N. & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths, *Systems Science and Cybernetics, IEEE Transactions on* 2: 100–107.

Inoue, Y., Tohge, T. & Iba, H. (2007). Cooperative transportation system for humanoid robots using simulation-based learning, *Appl. Soft Comput.* 7(1): 115–125.

Jed, L., Mark, R., Bruce, R. D. & Donald, P. G. (1990). Real-time robot motion planning using rasterizing computer graphics hardware, *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques* pp. 327–335.

Kovac, K., Zivkovic, I. & Dalbelo Basic, B. (2004). Simulation of multi-robot reinforcement learning for box-pushing problem, *Electrotechnical Conference, 2004. MELECON 2004. Proceedings of the 12th IEEE Mediterranean* 2: 603– 606.

LaValle, S. M. (2006). *Planning Algorithms*, Cambridge. Available on `http://planning.cs.uiuc.edu/`.

Li, Y. & Chen, X. (2004). Modeling and simulation of a swarm of robots for box-pushing task, *12th Mediterranean Conference on Control and Automation, Kusadasi, Aydin, Turkey* .

Lozano-Pérez, T. (1983). Spatial planning: A configuration space approach, *Computers, IEEE Transactions on* pp. 108–120.

MIYOSHI, T., KAWAKAMI, S. & TERASHIMA, K. (2008). Cooperative transportation system for humanoid robots using simulation-based learning, *Journal of Mechanical Systems for Transportation and Logistics* 1(1): 134–145.

Muñoz Melendez, A. & Drogoul, A. (2004). Analyzing multi-robot box-pushing, *Avances en la Ciencia de la Computación. Memoria de los Talleres del Quinto Encuentro Internacional de Computación ENC'04. Universidad de Colima* 1: 530–539.

Reif, J. H. (1979). Complexity of the mover's problem and generalizations, *Annual IEEE Symposium on Foundations of Computer Science* pp. 421–427.

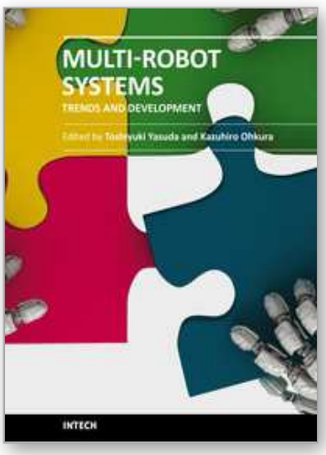
Reif, J. & Sharir, M. (1985). Motion planning in the presence of moving obstacles, *26th Annual IEEE Symposium on Foundations of Computer Science* pp. 144–154.

Suri, S., Hubbard, P. M. & Hughes, J. F. (1998). Analyzing bounding boxes for object intersection.

Wang, Y. & W. de Silva, C. (October 2006). Multi-robot box-pushing single-agent q-learning vs team q-learning, *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on* pp. 3694–3699.

Wang, Z. & Kumar, V. (2002). Object closure and manipulation by multiple cooperating mobile robots, *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on* 1: 394–399.

Yamada, S. & Saito, J. (2001). Adaptive action selection withouth explicit communication for multi-robot box-pushing, *Systems, Man and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 31(3): 398–404.



Multi-Robot Systems, Trends and Development

Edited by Dr Toshiyuki Yasuda

ISBN 978-953-307-425-2
Hard cover, 586 pages
Publisher InTech
Published online 30, January, 2011
Published in print edition January, 2011

This book is a collection of 29 excellent works and comprised of three sections: task oriented approach, bio inspired approach, and modeling/design. In the first section, applications on formation, localization/mapping, and planning are introduced. The second section is on behavior-based approach by means of artificial intelligence techniques. The last section includes research articles on development of architectures and control systems.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

José Ramírez-torres and Ezra Federico Parra González (2011). Object Path Planner for the Box Pushing Problem, Multi-Robot Systems, Trends and Development, Dr Toshiyuki Yasuda (Ed.), ISBN: 978-953-307-425-2, InTech, Available from: http://www.intechopen.com/books/multi-robot-systems-trends-and-development/object-path-planner-for-the-box-pushing-problem

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen