Analysis of robot Navigation Among Movable Obstacles in unknown environments and extension to social and dynamic constraints

Benoit RENAULT

Département Informatique
INSA de Lyon
2017/2018

Sous la responsabilité de :
Olivier SIMONIN : INRIA Chroma Team
Christine SOLNON : Département Informatique

Résumé. Nous présentons un état de l'art détaillé du domaine robotique de la NAMO ("Navigation Among Movable Obstacles": Navigation parmi des obstacles mobiles), ainsi qu'un ensemble de critères de comparaison nous permettant de situer les propositions existantes et la notre, après avoir synthétisé les nombreuses hypothèses et approches présentes dans la littérature, et mis en évidence les problématiques encore non explorées. Nous analysons les algorithmes locallement optimaux de NAMO en milieu inconnu, les reformulons en pseudocode et construisons un nouvel ensemble de propositions sur cette fondation. Nos propositions questionnent l'acceptabilité sociale pour un robot de bouger des obstacles et sont les premiers pas requis pour amener des contraintes dynamiques et sociales au domaine. Nous validons partiellement notre proposition avec un simulateur compatible avec les standards ROS (Robot Operating System), et des tests de poussée avec un robot physique Pepper, qui est la plateforme standard pour la compétition Robocup@Home.

Mots-clés : NAMO, Navigation parmi des obstacles mobiles, Optimalité, Navigation Sociale,

Abstract. We present a detailed state of the art of the robotics NAMO (Navigation Among Movable Obstacles) domain, and a set of comparison criterias to sort through existing propositions and situate our own, after synthesizing the many different hypotheses and approaches present in the litterature, and highlighting problematics that have not yet been explored. We then analyze the state of the art algorithms for locally optimal NAMO in unknown environments, reformulate them with pseudocode, and build a new set of propositions on this foundation. These propositions question the social acceptability of moving an obstacle and are the first required steps toward bringing dynamic and social constraints to the domain. We partially validate our propositions with a ROS-compatible (Robot Operating System) simulator and pushing tests with a real Pepper robot, standard platform for the Robocup@Home challenge.

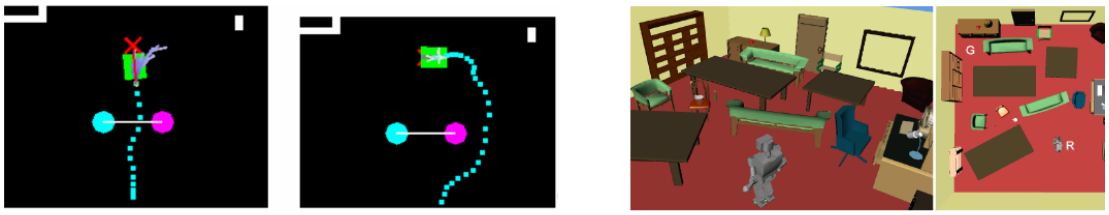
Key-words : NAMO, Navigation Among Movable Obstacles, Social Navigation

1 Introduction

Motivation

Service robotics are an active research field: there is a growing demand for intelligent machines that are meant to be used in human environments for domestic tasks (e.g, maintaining people homes, entertaining them, caring for them; especially old ones or with disabilities, ...). To fulfill such tasks, a service robot obviously needs to be able to autonomously navigate through space, according to the given constraints: these navigation capabilities and constraints are the main focus of the following work.

Human environments represent a very complex challenge, since they are dynamic, alterable and imply social conventions and rules that the robot must also respect in the way it navigates and interacts with the world. For example, in a home setting, humans (or other autonomous actors, such as pets) can be considered as moving obstacles that must be taken into account. Also, for a robot to go from a point A to B, a solution may only be found if it implies moving an obstacle out of the way. And all this must be done in a socially acceptable manner: one would not appreciate a robot moving at high speeds around people or to move obstacles that are not supposed to be moved.



(A) Sample social navigation problem (B) Sample NAMO problem

Illustration 1: Sample problems for social navigation and NAMO. In the sample social navigation problem, the robot (green square) must reconsider its path to avoid penetrating the zone between the two humans (colored dots) interacting with each other (image from [6]). In the NAMO problem, the robot will need to move obstacles in order to access its goal (image from [9]).

The most common constraint for robot navigation is solely to find the shortest collision-free path, and this is an acceptable solution in a static context (nothing moves, at the exception of the robot). However, if we want the robot to navigate a human environment, this is not sufficient.

To our knowledge, the problem of navigation planning in dynamic environments populated with humans [1], [2], and the problem of navigation planning in robot-alterable environments (which is called Navigation Among Movable Obstacles, or NAMO, more precisely defined in [3]) have always been treated separately. The INRIA Chroma Team[1], for one, proposed social and dynamic navigation algorithms that optimize the generation of trajectories by managing the risk of colliding with obstacles [4], [5], respecting social conventions such as avoiding human interaction spaces [6], [7] (Illustration 1(A)), or predicting the trajectories of moving obstacles [8], ... The NAMO problem (Illustration 1(B)) has been dealt with in a great variety of contexts, detailed in the following section. It seems that these two problems have yet to be brought together, and the new problematics that are to arise from this confluence are still to be identified and addressed.

Objective

The long-term objective behind this work is to allow a real service robot to navigate in complex, human, dynamic and robot-alterable environments, preferably in an optimal way (minimize risk of collision, travel distance, time or energy, ...).

More precisely, the context of this long-term objective for formulating, comparing and evaluating hypotheses, is the Robocup@Home[2]. The INRIA Chroma Team participates in the standard league of this

1 Team website: https://team.inria.fr/chroma/

2 Competition website: http://www.robocupathome.org/

international-level competition, that provides service robotics challenges to evaluate solutions proposed by researchers and students. The mandatory robotic platform for the standard league is the Pepper robot[3], piloted through ROS[4] for our team. Notably, this defines the context of our search in that:

- only the onboard sensors of the robot are used to update the robot's **partial environment knowledge** making,
- we thus seek **local optimality**: that is, optimal decision-making given the current belief state of the robot on its environment,
- overall, we will privilegiate solutions that are more easily applicable for Pepper. For example, we will prefer a solution that involves pushing rather than grasping obstacles to move them, as Pepper's limited mechanical features make it difficult to grasp obstacles in a safe way for the robot.

In the end, the specific objectives of this first work are to:

- explore the NAMO problem domain and extract characteristics that allow to sort through existing works,
- identify both new concerns and bridges between this problem and the ones of dynamic and social navigation,
- build upon existing work to propose solutions to the previously identified concerns,
- and finally validate our propositions in simulation, and as much as possible, in a real setting with the Pepper robot.

2 Navigation Among Movable Obstacles: state of the art

As a synthesis of the diversity of NAMO definitions accross publications, we summarize NAMO as *the problem of having an autonomous agent that navigates in an environment, going from its initial pose to a goal pose, while being authorized to move obstacles in order to reach the goal at a lower cost (in terms of time, distance or energy consumption, generally)*. The many approaches to the problem and their contexts highlight a need to characterize them, thus our establishing comparison criteria. In our detailed research report[5], we thoroughly discuss and justify each and every criteria, but because of a lack of space here, we will provide a recapitulative table of them. Globally, keep in mind that these criteria have been established according to the actual content of the studied articles and the goal we have set for ourselves to bring social and dynamic considerations.

In order to be able to properly situate our work in the context of the currently available research in the NAMO domain, we have also made several comparison tables according to the criteria presented beforehand. In the detailed research report, you will find the booleanized comparison tables, where each global criteria is divided into boolean sub-criteria that the paper answers to or not. You will also find the detailed comparison and cross-comparison tables that allowed us to write the following synthesis. The tables were not imported in the current document because of their size, and also because the reference numbers to the articles in the tables are aligned with the ones in the detailed research report.

Hypotheses	Approaches	Performance criteria
Knowledge of the environment	Path Planning Algorithm(s) and heuristics	Evaluation in a simulated/real setting
Obstacle characteristics	Evaluation and evolution of an obstacle's "movable" characteristic and its associated cost	Computation time
Robot characteristics	Object manipulation maneuver planning	Optimality and completeness
Problem class	Planning taking uncertainty into account	Optimality target
		Social acceptability
		Number and Density of obstacles

Table 1: Recapitulative table of comparison criteria, sorted by type.

3 Pepper characteristics are described in section 5 and also on this website: http://doc.aldebaran.com/2-4/family/pepper_technical/index_dev_pepper.html
4 ROS, the Robot Operating System, website: http://www.ros.org/
5 Detailed Research Report available here: https://github.com/Xia0ben/master-report

The first remark we drew from our tables is that while there is a wide diversity of propositions to solve the problem of Navigation Among Movable Obstacles, and each proposition is only applicable in a well-defined context, there are quite a few common points. For example, we learned that as long as the robot is not manipulating an obstacle, its freedom of movement is not limited to specific translation or rotation movements. The main reason why the movement of the robot when manipulating obstacle is often limited from the get-go, reducing the action space, is because it also reduces the search space of the algorithm, thus reducing the computation time complexity [9].

As many of the papers [9]–[15] recall it, the complexity of the NAMO problem would quickly be untractacle if we were to consider any manipulation on every obstacle in the environment to find a path to the goal: it has been shown by Wilfong that even a simplified variant of the domain where the final positions of the obstacles in a polygonal environment are specified by the user (which is closer to the domain of rearrangement planning), finding the obstacle configuration that allows to reach the goal in the best way is a P-Space Hard Problem [16]. The same work shows that if the final positions of the obstacles are not known, then the problem becomes NP-hard. A more recent work by Demaine et. al. showed that if we only considered push actions in a planar grid (problem analogous to the game of Sokoban[6]), the problem is NP-complete. Stilman summarizes this by concluding that "The size of the search space is exponential in the number of movable objects. Furthermore, the branching factor of forward search is linear in the number of all possible world interactions" [15]. Further discussion on the complexity of the domain with illustrations can be found in Stilman's thesis on NAMO [3].This is why the robots are often limited to push or pull actions, following a translation movement in a single direction in the propositions. When it is not so, like in [17], it is because only the nearest obstacle poses to the robot are considered (making the proposition non-optimal), and no real-time constraint is given. Actually, among all the selected papers, this is the only one that does not guarantee or show results of real-time execution.

Another important reason that can motivate the reduction of the action space of the robot is the will to reduce the risk of manipulating obstacles in an unexpected way. Indeed, grasping an obstacle in order to pull or pick&place it augments the number of interactions with the environment, and with that, the chance that something might go wrong: especially, as is mentioned in [13], the robot might lose its balance.

Our tables also showed that a diversity of real and simulated robots have been used for experimenting with NAMO algorithms, but it is to be noted that the ones that were actually used for a real world experiment are very costly robots (PR2[7], HRP2[8] and GOLEM[9]) at the exception of the custom robotic platform built by Clingerman[18]. However, the robot used by Clingerman is arguably not capable of handling problems as complex as the propositions of Stilman and Levihn, since it does not have a manipulation arm.

Our detailed state of the art also showed us that all propositions are built upon a variety of existing path finding algorithms. In most papers, the choice of the path finding algorithm they build upon is not explicitly justified. The only one that justifies the choice of its Path Finding algorithm is Clingerman [19], since it allows him to potentially manage autonomously moving obstacles, since it is based off the D* Lite algorithm (see [20] and [10]), but no experiment with a changing environment is shown. Almost all papers use some sort of heuristic, either for the path finding subroutine or choosing which obstacle to evaluate, and some of these heuristics come at the cost of optimality [9], [10]. Some papers [12]–[14], [18], [19], [21] offer means to take uncertainty into account. All of them have adaptive approach procedures that are executed when nearing the obstacle, but some use much more elaborate approaches to this end, like using probabilistic models [12], [14].

It is noteworthy that many of the algorithms do not focus on guaranteeing the optimality of the plans they produce, or even their completeness. This can easily be explained by the fact that, in robotics, real-

6 Wikipedia page on Sokoban: https://en.wikipedia.org/wiki/Sokoban
7 PR2 Characteristics: http://www.willowgarage.com/pages/pr2/specs
8 HRP2 Characteristics: http://global.kawada.jp/mechatronics/hrp2.html
9 GOLEM Characteristics: http://www.golems.org/projects/krang.html
10 Koenig's other paper on D* Lite: http://idm-lab.org/bib/abstracts/papers/aaai02b.pdf

time performance is paramount, and trading off optimality for performance is often prefered. However, sensible approaches like [9], [10], where the proposition contains an original algorithm that is optimal, but also a modified version that improves performance at the cost of optimality, are more interesting, as it is often easier to improve the computational performance of an algorithm by sacrificing its optimality, than trying to make a non-optimal algorithm optimal.

In the selected papers, about half make propositions to deal with an unkwown or partially known environment (which is definitely correlated with the fact that the robot is considered to have a limited field of vision), and it could give the impression that this is a well-treated subset of NAMO problems. However, this is in fact related to our initial bias that we want to build a solution that can manage a partially known environment. In addition to the selected papers, quite a few of other briefly examined papers strongly related to NAMO [22]–[29] rely on a complete knowledge of the environment, and also, perfect data (that is, they assume that what the robot knows of the environment is always almost perfectly like the real setting). It is the paper of Kakiuchi et.al. [30] that brought the first extension to NAMO in unknown environments [11] but as it was only a local approach (the robot only reacts to a specific movable obstacle, without considering all the others in the computation of the new plan) it had no hopes of optimality. Wu and Levihn were the ones to formulate a locally optimal algorithm for NAMO in completely unknown environments [10], [11]. Levihn and Stilman then worked on another approach for partially known environments that does not guarantee optimality, but improves performance and allows for greater complexity in obstacle configurations and robot action set [21]. Clingerman also proposed later another solution for NAMO in unknown environments [18], [19], however this solution does not consider the manipulation of an obstacle as an action in itself and simply makes the robot try to "pass through" the obstacle, without trying to consider if it is going to collide with its environment.

Finally, what the our tables definitely showed to us is that none of the selected papers (and also the ones that were only briefly examined) consider a case with movable obstacles mixed with humans. Also, if Stilman [9] and Kakiuchi [30] briefly mention the necessity to consider the frailty of a manipulated obstacle, which can be interpreted as taking social conventions into account, neither them or any other paper actually try to adapt their NAMO algorithm to social conventions or norms.

Situating our work in the established context

In the end, we chose to base our following work on the solution proposed by Wu and improved by Levihn [10], [11], because:

- It is a solution designed for unknown environments, thus also adapted to partially known ones,
- One of our initial objectives was for our proposition to make optimal decisions: their solution is the only one allowing that for partially known environments,
- The reduction to push actions is not a problem for us since we did not want to spend time on grasping problematics in the first place (this could have been a lot of time, since grasping problematics have their very own research field [31]), and the grasping capabilities of Pepper are very limited (low applied force in particular).

In our detailed report, the corresponding chapter to the following section thoroughly presented the solution proposed by Wu et. al. [10], and showed in detail how the improvements proposed by [11] should be applied, but we will summarize this by a synthesized summary.

3 Study of Wu et. al.'s algorithm for locally optimal NAMO in unknown environments

In Wu et. al.'s proposition [10], the basic idea is to consider either a plan that doesn't involve interacting with obstacles (which can be achieved with any pre-existing path finding algorithm), or a three-steps plan. As shown in the illustration below, the latter consists in, first, reaching the obstacle (c_1), second, pushing it in a single direction (c_2), and finally reaching the goal from the position we left the obstacle at (c_3).

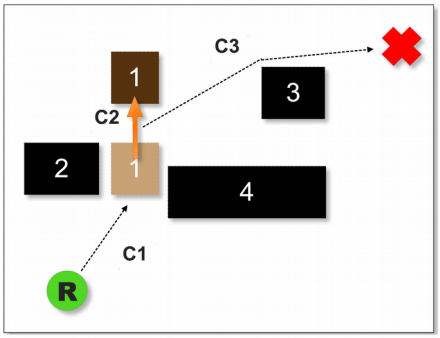


Illustration 2: Figure describing the three plan components when pushing an object, as proposed in [10].

Both algorithms assume that the robot has no prior knowledge of the environment at all, and that new information is gradually stored in a 2D metric map, assuming perfect data, and making the hypothesis that unknown space is free space. The 2D metric map is translated into a binary occupancy grid[11] for A* and the opening detection algorithm (presented later). Whereas the first proposition is limited to rectangular obstacles, the second is extended to any polygonal obstacles, and none take autonomously moving obstacles (humans, animals or animated objects) into account. In the first proposition, obstacles can only be pushed, whereas in the second one, obstacles can be translated in any direction. The considered robot is a simulated differential drive nondescript robot with a limited field of vision, that can only push obstacles in the first proposition, but can also pull them in the second one.

Instead of first showing the original pseudocode that formalizes this algorithm, detailing the ambiguities and notation incohences, and progressively building a new, properly formalized pseudocode as we did in the detailed report, we will directly give (in annex 1) our own reformulation of the pseudocode that answers to the problem of locally optimal navigation in unknown environments, combining the two articles propositions, and explain the algorithm's logic with it.

The algorithm is divided into three main methods : MAKE-AND-EXECUTE-PLAN (Alg.3), MAKE-PLAN (Alg.4) and PLAN-FOR-OBSTACLE (Alg.5). The first one contains the plan invalidation and execution logic, the second, the heuristical process for choosing the order in which the algorithm considers obstacles to evaluate plans involving them, and the third, actual process of computing the currently optimal plan that would imply moving a specific obstacle.

In Alg.3, the robot starts by using an D* Lite path finding subroutine to determine the optimal plan between the current robot's position and the goal, avoiding all known obstacles (none at the beginning). As the robot moves forward (line 29), and therefore gains new information (line 11), it checks whether the current optimal plan is invalidated (causes a collision) by any newly encountered obstacles (lines 17, 18). If the plan is invalidated, then, a new one must be computed : first we compute a plan that implies avoiding all known obstacles (lines 19, 20) and then try to improve it by calling MAKE-PLAN which will evaluate plans that imply moving obstacles. Navigation stop either when the robot reaches its goal (line 10) or when no plan avoiding or moving obstacles is found (lines 24 to 26). If the robot tries to move an obstacle but does not succeed, this obstacle is added to a blacklist (lines 30 to 33)

In Alg.4, the robot first computes for all obstacles the euclidean distance between the goal and the grasping/manipulation pose that is closest to it, giving an underestimate of the cost of the a plan that would imply moving the obstacle (lines 2 to 5). This cost is used as a heuristic to order obstacles in a list,

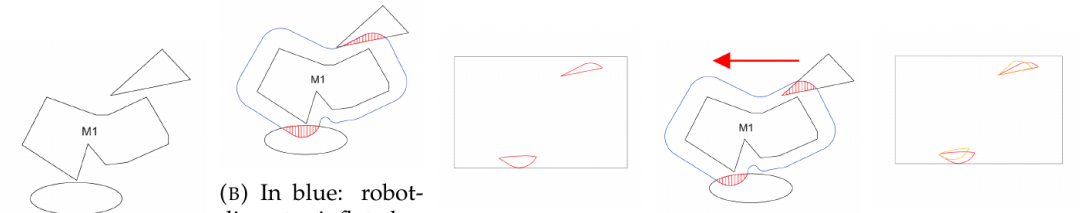
11 **Occupancy grids** are used in robotics to represent the environment as a discrete grid. Binary ones represent occupied space (obstacles) by true values and free space by false ones. More complex modelizations of occupied space are also used, like associating to cells floating-point values of the probability of occupation. A more complete defition can be found here: https://www.mathworks.com/help/robotics/ug/occupancy-grids.html

allowing the algorithm to evaluate the most promising obstacles first. Another, tighter, heuristic cost, is obtained when evaluating an obstacle and is defined as the sum of the costs of plan components c_2 and c_3 (respectively opt. path to push obstacle and opt. path to go from obstacle to goal). This underestimate is only valid if no free space was created (i.e. no obstacle was moved from its place), because then, the moved obstacles could have caused c_2 or c_3 to augment (which explains lines 12 to 14 in Alg. 3). The traversal logic of the two heuristic lists of obstacles euCostL (ordered by euc. dist.) and minCostL (ord. by cost of c_2 and c_3) is that since minCostL is related to a more informed estimate of the actual cost of a plan, it should be used over euCostL to choose the obstacle to evaluate in priority (lines 9 to 35). Overall, this logic allows to terminate evaluation sooner than if we simply evaluated all obstacles, since it is not interesting to continue evaluating plans if the current optimal one already costs less to execute than the next obstacle's cost underestimate (line 8).

Finally, in Alg. 5, the algorithm first computes the path from the robot's current pose to the obstacle (line 6), and if a path has been found it starts simulating manipulations in every allowed direction (line 11). For each direction, it simulates successive unit pushes and verifies for each if the manipulation is still possible (no intersection with other obstacles) and that the currently underestimated cost C_{est} of the simulated manipulation does not exceed the cost of the current optimal plan, in which case, continuing to evaluate plans in this direction is useless (lines 12 to 16 and 31 to 34). Also, in the propositions of Wu and Levihn, the full evaluation of a plan is conditioned (line 17) by the necessity for it to locally create a new opening to the goal in the vicinity of the obstacle, which allows to spare computational resources since this check is less costly than running a graph search algorithm like D* Lite.

We show, however, that conditioning the full evaluation of a plan involving the manip. of an obstacle only by checking local new openings around the obstacle actually hinders the optimality of the algorithm, and even its completeness.

The local opening detection algorithm is described by the authors in a separate technical paper [32]. They explain that a new opening is detected only if the robot-diameter-inflated area of the movable obstacle is no longer intersected by the non-inflated area of another obstacle. If there are no obstacles in the robot-diameter-inflated area, then it means that the robot can pass through it, which can be considered as a local opening in the vicinity of the obstacle. This logic is illustrated in their figures below.



(A) Movable obstacle M1 and other obstacles.

(B) In blue: robot-diameter-inflated movable obstacle M1. In cross-red: intersection areas with other obstacles.

(C) Saved information on blocking areas before manipulation.

(D) State after M1 was translated to the left.

(E) No blocking area has disappeared: no new opening is detected.

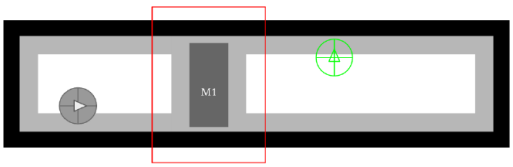
Illustration 3: Opening detection example given in the technical paper

However, conditioning the full evaluation of a plan by the detection of a new opening prevents from finding the solution in cases where the local environment of the obstacle does not contain any other obstacle (Illustration 5), or if for the considered manipulation, the local environment of the obstacle does not change (Illustration 4). Below are examples figures of these cases, assuming that the cost of following a path is the same whether it implies moving an obstacle or not.

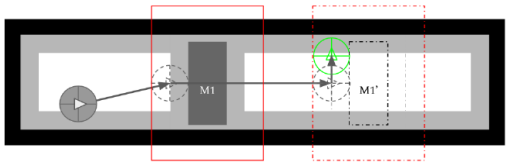
In the corridor case initial situation (Illustration 4(A)), the robot (dark grey disk with a triangle) can not go to its goal (green disk with triangle) without moving the obstacle M1 (dark grey is for a movable obstacle, black is for unmovable obstacle, and light grey is for the inflation of the obstacles by the robot's radius), since the space is divided into two independent free space components (the robot's center cannot penetrate the grey or black zones without entering in collision). Furthermore, the robot can only move the

obstacle by either pushing or pulling it left or right. However far the robot simulates moves in either direction, the robot-diameter-inflated area of M1 (red line) will never lose its two blocking areas (intersections between the red rectangle and the unmovable obstacle in black), therefore, no new opening will ever be detected according to the definition of the algorithm, and no plan moving the obstacle will ever be considered. However, Illustration 4(B) shows clearly that a plan exists that is valid if we simply don't check for new openings (new configuration of M1, noted M1' is represented in dotted lines).

In the open space case initial situation (Illustration 5(A)), no new opening will ever be found however we move the obstacle, since there are no obstacles (represented in black or dark grey) in its robot-diameter-inflated area. Since, this time, there is space for the robot to move around the obstacle instead of pushing it, the algorithm will thus return this suboptimal plan, instead of the optimal one shown in Illustration 5(B) that implies moving M1 to its new configuration M1'.

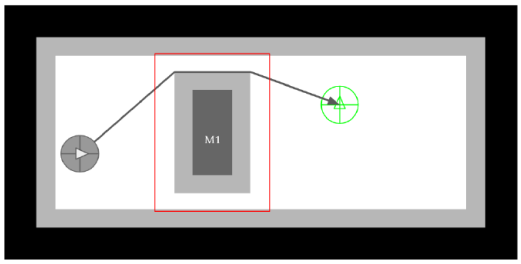


(A) Initial situation where no plan will be found when using new opening checking.

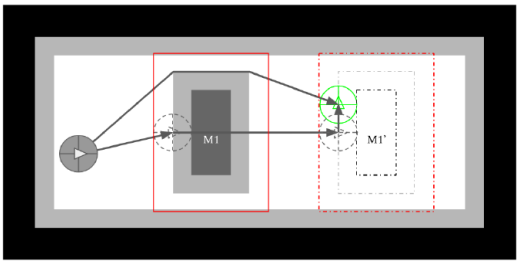


(B) Expected optimal plan.

Illustration 4: "Corridor" case where the original algorithm will not even find a plan when it should.



(A) Initial situation and suboptimal plan avoiding the obstacle that will be returned by the algorithm when using new opening checking.



(B) Expected optimal plan is the one that pushes the obstacle.

Illustration 5: "Open space" case where the original algorithm will only find a suboptimal plan, under the assumption that the cost of following a path is the same whether it implies moving an obstacle or not.

4 Extension of Wu et. al.'s algorithm toward social and dynamic navigation

Discussion on the original hypotheses in the light of constraints brought by the Pepper robot and the Robocup@Home

As our goal experimental platform is the Pepper robot, in the context of the Robocup@Home challenge that emulates a home setting, several hypotheses from the propositions of Wu and Levihn [10], [11] have to be reconsidered:

Initial knowledge of the environment is partial, in that all static obstacles (i.e. objects that are not meant to be moved by any actor, like walls or very heavy furniture) have already been mapped (this is expressed in our pseudocode in Appendix 2, Alg. 6, lines 1 and 3). In the context of the Robocup@Home Challenge, participants are allowed to build such a map prior to the actual trials. This hypothesis is actually quite justified since in a home setting, it is very likely that the robot has undergone a configuration phase prior to its daily use, when it is provided with a manually drawn map of the home, or at least allowed to roam about and map the static obstacles. Having a map of static obstacles is very

important for standard localization algorithms used in ROS, like AMCL[12], since they use this environment knowledge to compensate for odometry[13] error.

Robot manipulation capabilities, for the moment, are to be limited to pushes in a perpendicular direction to the obstacle's side being pushed, as in [10] and not as in [11]. Given the many problematics related to grasping objects (e.g., appropriate positioning of the robot joints, keeping the robot balanced, ...), and the low power that can be delivered by Pepper's arms, it is best for a first iteration not to dwell on these. Also, we will check whether a manipulation is possible or not by verifying whether the area covered by the robot and the obstacle as they move together is in intersection with any other obstacle. Since we limit actions to pushes in a single direction, this area can be defined as the convex hull containing both the robot's and the obstacle's polygonal representation at their initial and final pose. In reference to the existing litterature vocabulary of "safe-swept volume" for 3D environments[3], [33], we will call this the "safe-swept area" if no other obstacle is in intersection with it (see Illustration 6). In our pseudocode, this is done by the "GET-SAFE-SWEPT-AREA" method (Algorithm 7, lines 21 and 36), which returns null if any obstacle is in intersection with the manipulation area.

Push/Manipulation poses are a key concept of manipulating obstacles, as we have shown in the previous chapter, and, contrary to the original algorithms we will explicitly explain our hypotheses as to them. Experimentations with the Pepper Robot (see section 5) and carboard boxes as movable obstacles have shown that a good first approximation that guarantees quasi-systematic push manipulation successes are poses situated at the middle of the object's sides (see Illustration beside). This is, of course, supposing that we are only considering light objects with negligible friction against the ground, and with no other cinematic constraint than a plan-plan link between one of the obstacle's faces and the ground (a perfect plane).

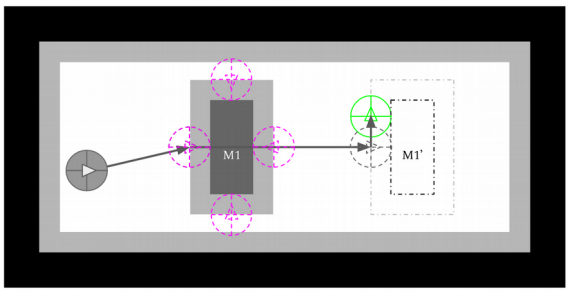


Illustration 6: Illustration of the push poses for obstacle M1 (pink disks) and safe-swept area (blue rectangle)

Also, as the robot approaches obstacles, their geometrical representation is updated according to what the robot's sensors can see. When executing a plan that includes the manipulation of an obstacle, said obstacle knowledge can actually evolve during the execution of the c_1 component, which is problematic for the preservation of optimality if we use the definition of a push/manipulation pose of the previous paragraph, since the obstacle's push poses will evolve with the perceived geometry of the obstacle. Therefore re-evaluation should not only be triggered if a new obstacle intersects with the current optimal plan, but also if the current optimal plan includes the manipulation of an obstacle and if said obstacle has changed in a way that makes the originally targeted pushPose unavailable (Alg. 6, lines 22 to 24).

Propositions on Social awareness and dynamic environments

Since we did not have the time to test in simulation these propositions that are described in our detailed research report, we will not dwell on these in this synthesis. We however provide their final expression in pseudocode in Appendix 3.

12 ROS documentation page on AMCL: \url{http://wiki.ros.org/amcl}

13 **Odometry** is the use of data brought by sensors like rotary encoders or inertial measurement units to deduce an approximatinon of the robot's localization. More on Wikipedia: https://en.wikipedia.org/wiki/Odometry

5 Experimentations & Validation

Pushing tests with Pepper

Before trying to implement the algorithm on the actual Pepper robot, the first thing to do was to check if it actually could push obstacles with certainty as to the result of the manipulation. More precisely, we needed to know what kind of movable obstacles we could move with confidence in the results, and how to position the robot relatively to the obstacle for the push action to succeed. Our hypothesis was that given the round nature of Pepper's base front (see Annex 3), we suppose that light obstacles and obstacles with wheelcasters could be moved, and that placing the robot at the center of the obstacle's side, and pushing in a perpendicular direction to it gives the most repeatable results.

- We start our custom Docker container to connect to the robot, following the instructions in the https://github.com/Xia0ben/rosdocked-kinetic-pepper .
- Position Pepper at the beginning of the test line with an obstacle at its base in a given configuration (for each obstacle, we at least test the middle of the obstacle's side, and if results were encouraging, also positions at the border of the obstacle where the robot would have two, then one contact points). Take photos from the side (wide + close angle), top (only close angle) and front (wide + close angle) (photos in appendix 5).
- Start a video take from side and front wide angles, and send the following command to tell Pepper to go forward for only about 1.5 meters
- Take photos from the side (wide + close angle), top (only close angle) and front (wide + close angle) to notice the drift from the robot's trajectory.
- Reposition Pepper, position the tested obstacle, and restart the experimentation.

After tests conducted on empty cardboard boxes of varied sizes (bigger or smaller than the robot's base), a chair with wheelcasters and an office trash can, we experimentally conclude that:

- Pushing obstacles with wheelcasters without grasping nor adaptive procedures to compensate their drift, results in quick drifts from the robot's trajectory, because each wheelcaster generally has its own friction constraints, and even when all are manually orientated in the same direction, they rotate in an unpredictable way (Appendix 5, Illustrations 12(A) and (B)).
- Light polygonal or round obstacles with an even distribution of mass (boxes and can), drift little if not at all from the robot's trajectory if the robot is placed so that its center faces the center of the side of the obstacle being pushed (the center of the circle for round obstacles). For the cardboard boxes, as long as the robot's span is within the span of the obstacle, little drift is produced from the robot's trajectory, but as soon as the robot's span overflows a little (Appendix 5, Illustrations 12(C) and (D)), there is a significant drift (likely caused by the fact that it makes the robot start with only one contact point with the obstacle at the beginning).

We therefore concluded that it is relatively safe to assume that always placing our robot at the middle of the obstacle's side for a push action will almost always produce the expected translation without significant drift.

Simulation in a ROS-Standards compatible simulator

In order to test our propositions in section 4, and validate our many improvements over the original algorithm formulation in section 3, we developed a custom simulator based on ROS standards. By that, we mean that the simulator uses ROS standard message types[14], to exchange data about the poses, observations of the robot, path to follow, ... This, in itself, reduces the gap to implement our propositions on a real robot (Pepper), since the only missing piece for a first implementation in the real world is an obstacle sensing identification component that would produce the data currently provided by our simulator (pointcloud of the obstacles geometry with each point identified with a specific obstacle id). The vizualition is done through Rviz[15], the standard 3D visualizer for ROS, simply by listening to the

14 See ROS documentation page: http://wiki.ros.org/common_msgs
15 See ROS documentation page: http://wiki.ros.org/rviz

messages exchanged by the program and the simulator and displaying them with its default visual components. It is to be noted that the optimality target for this implementation is at the moment only expressed in distance, but it is relatively trivial to express it in time or energy, by using the moveCost and pushCost constants currently present in the algorithm and passing them to the A* path finding subroutine[16].

At the time of the writing of this report, we were only able to implement the algorithm and the necessary simulation code for the base solution presented in section 4.1 (and corresponding algorithms 6, 4 and 7 in the detailed research report), but not the three propositions that build upon it. This still allowed us to validate all comments and improvements made in Chapter 3 and in the first section of Chapter 4. In appendix 6 are a few illustrations showing a test case in a corridor with two obstacles.

6 Conclusion

We presented a detailed state of the art of the NAMO domain, and provided a set of criterias that allowed us to compare the many propositions that exist in this domain, and situate our overall proposition in contrast to them. We concluded this analysis with a synthesis on the many different hypotheses and approaches present in the litterature, and highlighted problematics that have not yet been explored (dynamic and social navigation in particular).

We then analyzed the state of the art algorithms for locally optimal NAMO in unknown environments [10], [11], fixed and reformulated them properly with pseudocode, and finally, built a new set of propositions on this foundation, that constitute the first steps toward bringing dynamic and social constraints to the domain. A first proposition modified the obtained foundation to make it fit with our hypotheses in regard to the use of the Pepper Robot in the context of the Robocup@Home challenge. A second proposition consisted in allowing the robot to take the action of identifying into consideration in its optimal plan computation, in order to allow the robot to then use this new information to know whether or not it is authorized to move the obstacle. A third proposition questioned the social acceptability of placing an obstacle in a specific spot and brought a basic way to integrate this new constraint. Finally, a fourth and last proposition exposes the minimal necessary changes to allow the algorithm to still make optimal plans in a dynamic environment.

While we were not able to test all of them in simulation or reality at the time of this writing, we were able to validate our first proposition and the possibility for the Pepper robot current actuators to be used in the real application.

7 Perspectives

The first perspective to our work is of course to validate then build upon our existing propositions, in order to manage more complex dynamic and social navigation models, but also to interact with other agents, be them humans or other robots. One can easily imagine robot interactions for NAMO where the robot kindly asks another agent to move out of its way, or ask help in moving obstacles that it is not capable of moving alone (which is a problem that has already been addressed in existing research, but only for the purpose of moving a user-specified object [22]). Another interesting perspective is that we could also explore about the environment observation problem surrounding NAMO, and find an optimal way for a robot to use data provided by other agents (robots or IoT sensors), and ask for other agents to check the surroundings of an obstacle the robot is currently considering for movement. As the need for better performance may rise, we can also study ways of sacrificing the local optimality of our proposition in a controlled manner in order to improve performance as in [12]. In the same way, as we head toward real-world experimentations, we will maybe need to use approaches that better take uncertainty into account, as in [12]–[14], [21]. Finally, evaluating this work in the actual context of the next Robocup@Home challenges would be a great way to validate it.

16 All the code and its execution instructions are available on the following repository: https://github.com/Xia0ben/namo_navigation

References

[1] T. Kruse, A. K. Pandey, R. Alami, and A. Kirsch, "Human-aware robot navigation: A survey," *Robot. Auton. Syst.*, vol. 61, no. 12, pp. 1726–1743, Dec. 2013.

[2] J. Rios-Martinez, A. Spalanzani, and C. Laugier, "From Proxemics Theory to Socially-Aware Navigation: A Survey," *Int. J. Soc. Robot.*, vol. 7, no. 2, pp. 137–153, Apr. 2015.

[3] M. Stilman, "Navigation Among Movable Obstacles," Carnegie Mellon University, Pittsburgh, PA, 2007.

[4] C. Fulgenzi, "Autonomous navigation in dynamic uncertain environment using probabilistic models of perception and collision risk prediction.," phdthesis, Institut National Polytechnique de Grenoble - INPG, 2009.

[5] J. Rios-Martinez, "Socially-Aware Robot Navigation : combining Risk Assessment and Social Conventions," Jan. 2013.

[6] J. Rios-Martinez, A. Spalanzani, and C. Laugier, "Understanding human interaction for probabilistic autonomous navigation using Risk-RRT approach," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 2014–2019.

[7] P. Papadakis, P. Rives, and A. Spalanzani, "Adaptive spacing in human-robot interactions," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 2627–2632.

[8] F. Jumel, J. Saraydaryan, and O. Simonin, "Mapping likelihood of encountering humans: application to path planning in crowded environment," in *The European Conference on Mobile Robotics (ECMR)*, Paris, France, 2017.

[9] M. Stilman and J. J. Kuffner, "Navigation among movable obstacles: real-time reasoning in complex environments," *Int. J. Humanoid Robot.*, vol. 02, no. 04, pp. 479–503, Dec. 2005.

[10] H.-N. Wu, M. Levihn, and M. Stilman, "Navigation Among Movable Obstacles in unknown environments," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 1433–1438.

[11] M. Levihn, M. Stilman, and H. Christensen, "Locally optimal navigation among movable obstacles in unknown environments," in *2014 IEEE-RAS International Conference on Humanoid Robots*, 2014, pp. 86–91.

[12] M. Levihn, J. Scholz, and M. Stilman, "Planning with movable obstacles in continuous environments with uncertain dynamics," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 3832–3838.

[13] M. Stilman, K. Nishiwaki, S. Kagami, and J. J. Kuffner, "Planning and executing navigation among movable obstacles," *Adv. Robot.*, vol. 21, no. 14, pp. 1617–1634, Jan. 2007.

[14] J. Scholz, N. Jindal, M. Levihn, C. L. Isbell, and H. I. Christensen, "Navigation Among Movable Obstacles with learned dynamic constraints," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 3706–3713.

[15] M. Stilman and J. Kuffner, "Planning Among Movable Obstacles with Artificial Constraints," *Int. J. Robot. Res.*, vol. 27, no. 11–12, pp. 1295–1307, Nov. 2008.

[16] G. Wilfong, "Motion planning in the presence of movable obstacles," *Ann. Math. Artif. Intell.*, vol. 3, no. 1, pp. 131–150, Mar. 1991.

[17] K. Okada, A. Haneda, H. Nakai, M. Inaba, and H. Inoue, "Environment manipulation planner for humanoid robots using task graph that generates action sequence," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, 2004, vol. 2, pp. 1174–1179 vol.2.

[18] C. Clingerman and D. D. Lee, "Estimating manipulability of unknown obstacles for navigation in indoor environments," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 2771–2778.

[19] C. Clingerman, P. J. Wei, and D. D. Lee, "Dynamic and probabilistic estimation of manipulable obstacles for indoor navigation," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 6121–6128.

[20] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Trans. Robot.*, vol. 21, no. 3, pp. 354–363, Jun. 2005.

[21] M. Levihn, L. P. Kaelbling, T. Lozano-Pérez, and M. Stilman, "Foresight and reconsideration in hierarchical planning and execution," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 224–231.

[22] C. Amato, G. Konidaris, G. Cruz, C. A. Maynor, J. P. How, and L. P. Kaelbling, "Planning for decentralized control of multiple robots under uncertainty," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 1241–1248.

[23] P. C. Chen and Y. K. Hwang, "Practical path planning among movable obstacles," in *1991 IEEE International Conference on Robotics and Automation Proceedings*, 1991, pp. 444–449 vol.1.

[24] K. Okada, T. Ogura, A. Haneda, J. Fujimoto, F. Gravot, and M. Inaba, "Humanoid motion generation system on HRP2-JSK for daily life environment," in *IEEE International Conference Mechatronics and Automation, 2005*, 2005, vol. 4, pp. 1772-1777 Vol. 4.

[25] D. Nieuwenhuisen, A. F. van der Stappen, and M. H. Overmars, "An Effective Framework for Path Planning Amidst Movable Obstacles," in *Algorithmic Foundation of Robotics VII*, Springer, Berlin, Heidelberg, 2008, pp. 87–102.

[26] J. van den Berg, M. Stilman, J. Kuffner, M. Lin, and D. Manocha, "Path Planning among Movable Obstacles: A Probabilistically Complete Approach," in *Algorithmic Foundation of Robotics VIII*, Springer, Berlin, Heidelberg, 2009, pp. 599–614.

[27] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1470–1477.

[28] M. Levihn, J. Scholz, and M. Stilman, "Hierarchical Decision Theoretic Planning for Navigation Among Movable Obstacles," in *Algorithmic Foundations of Robotics X*, Springer, Berlin, Heidelberg, 2013, pp. 19–35.

[29] M. Levihn, K. Nishiwaki, S. Kagami, and M. Stilman, "Autonomous environment manipulation to assist humanoid locomotion," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 4633–4638.

[30] Y. Kakiuchi, R. Ueda, K. Kobayashi, K. Okada, and M. Inaba, "Working with movable obstacles using on-line environment perception reconstruction using active sensing and color range sensor," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 1696–1701.

[31] A. Sahbani, S. El-Khoury, and P. Bidaud, "An overview of 3D object grasp synthesis algorithms," *Robot. Auton. Syst.*, vol. 60, no. 3, pp. 326–336, Mar. 2012.

[32] M. Levihn and M. Stilman, "Efficient Opening Detection," Georgia Institute of Technology, Technical Report, 2011.

[33] A. Foisy and V. Hayward, "A safe swept volume method for collision detection," *Int. J. Robot. Res. - IJRR*, Jan. 1994.

Annex

1. Pseudocode interpretation of the combined proposition of Wu and Levihn

Algorithm 3 Optimized algorithm for NAMO in unknown environments of Wu et. al. adapted according to M.Levihn et. al.'s (2014) recommandations - EXECUTION LOOP

procedure MAKE-AND-EXECUTE-PLAN(R_{init}, R_{goal})
  $R \leftarrow R_{init}$
  $\mathcal{O}_{new} \leftarrow \emptyset$
  $isManipSuccess \leftarrow True$
  $blockedObsL \leftarrow \emptyset$
  $euCostL, minCostL \leftarrow \emptyset, \emptyset$
  $I \leftarrow empty\_occupation\_grid$
  $p_{opt}.components \leftarrow [D^*\text{Lite}(R_{init}, R_{goal}, I.occGrid)]$
  $p_{opt}.cost \leftarrow |p_{opt}| * moveCost$
  while $R \neq R_{goal}$ **do**
    UPDATE-FROM-NEW-INFORMATION(I)
    if I.freeSpaceCreated() **then**
      $minCostL \leftarrow \emptyset$
    end if
    $\mathcal{O}_{new} \leftarrow \mathcal{O}_{new} \bigcup I.newObstacles$
    $\mathcal{O} \leftarrow I.allObstacles$
    $isPlanNotColliding \leftarrow p_{opt} \bigcap \mathcal{O}_{new} \neq \emptyset$
    if not($isPlanNotColliding$ AND $isManipSuccess$) **then**
      $p_{opt}.components \leftarrow [D^*\text{Lite}(R, R_{goal}, I.occGrid)]$
      $p_{opt}.cost \leftarrow |p_{opt}| * moveCost$
      MAKE-PLAN(R, R_{goal}, I, $\mathcal{O}$, $blockedObsL$, p_{opt}, $euCostL$, $minCostL$)
      $\mathcal{O}_{new} \leftarrow \emptyset$
    end if
    if $p_{opt}.components = \emptyset$ **then**
      return *False*
    end if
    $isManipSuccess \leftarrow True$
    $R_{next} \leftarrow p_{opt}$.getNextStep()
    $R_{real} \leftarrow$ ROBOT-GOTO(R_{next})
    if $p_{opt}.nextStepComponent$ is c_2 AND $R_{real} \neq R_{next}$ **then**
      $isManipSuccess \leftarrow False$
      $blockedObsL \leftarrow blockedObsL \bigcup p_{opt}.o$
    end if
    $R \leftarrow R_{real}$
  end while
  return *True*
end procedure

Algorithm 4 Optimized algorithm for NAMO in unknown environments of Wu et. al. adapted according to M.Levihn et. al.'s (2014) recommandations - PLAN COMPUTATION

procedure MAKE-PLAN(R, R_{goal}, I, $\mathcal{O}$, $blockedObsL$, p_{opt}, $euCostL$, $minCostL$)
  for each $o \in \mathcal{O}$ **do**
    $C_{3_{(Est)}} \leftarrow \min(\{\forall graspPoint \in o.graspPoints \;||\; |\{graspPoint, R_{goal}\}|\})$
    $euCostL$.insertOrUpdate($\{o, C_{3_{(Est)}}\}$)
  end for
  $i_e, i_m \leftarrow 0, 0$
  $evaluatedObstacles \leftarrow \emptyset$
  while min($minCostL[i_m]$.$minCost$, $euCostL[i_e]$.$c3_{est}$) $< p_{opt}.cost$ **do**
    if $minCostL[i_m].minCost < euCostL[i_e].c3_{est}$ **then**
      $o \leftarrow minCostL[i_m].obstacle$
      if $o \notin evaluatedObstacles$ **then**
        $p \leftarrow$ PLAN-FOR-OBSTACLE(o, p_{opt}, I, R, R_{goal}, $blockedObsL$)
        if $p \neq$ null **then**
          $minCostL$.insertOrUpdate($\{o, p.minCost\}$)
        else
          $minCostL$.insertOrUpdate($\{o, +\infty\}$)
        end if
        $evaluatedObstacles$.insert(o)
      end if
      $i_m \leftarrow i_m + 1$
    else
      if not $minCostL$.contains($euCostL[i_e]$.$obstacle$) **then**
        $o \leftarrow euCostL[i_e].obstacle$
        if $o \notin evaluatedObstacles$ **then**
          $p \leftarrow$ PLAN-FOR-OBSTACLE(o, p_{opt}, I, R, R_{goal}, $blockedObsL$)
          if $p \neq$ null **then**
            $minCostL$.insertOrUpdate($\{o, p.minCost\}$)
          else
            $minCostL$.insertOrUpdate($\{o, +\infty\}$)
          end if
          $evaluatedObstacles$.insert(o)
        end if
        $i_e \leftarrow i_e + 1$
      end if
    end if
  end while
end procedure

Algorithm 5 Optimized algorithm for NAMO in unknown environments of Wu et. al. adapted according to M.Levihn et. al.'s (2014) recommandations - PLAN EVALUATION FOR A SINGLE OBSTACLE

procedure PLAN-FOR-OBSTACLE(o, p_{opt}, I, R, R_{goal}, $blockedObsL$)
  if $o \in blockedObsL$ **then**
    return null
  end if
  $P_{o,d} \leftarrow \emptyset$
  $c_1 \leftarrow D^*\text{Lite}(R, o.init, I)$
  if $c_1 = \emptyset$ **then**
    return null
  end if
  $BA \leftarrow null$
  for each possible manipulation direction d on o **do**
    $seq \leftarrow 1$
    $oSimPose \leftarrow o.init + one\_translation\_in\_d$
    $c_{3_{(Est)}} \leftarrow \{oSimPose, R_{goal}\}$
    $C_{est} \leftarrow (|c_1| + |c_{3_{(Est)}}|) * moveCost + seq * |one\_translation\_in\_d| * pushCost$
    while $C_{est} \leq p_{opt}.cost$ AND manipulation on o possible **do**
      if CHECK-NEW-OPENING($I.occGrid$, o, $seq * one\_translation\_in\_d$, BA) **then**
        $c_2 \leftarrow \{o.init, oSimPose\}$
        $c_3 \leftarrow D^*\text{Lite}(oSimPose, R_{goal}, I.withSimulatedObstacleMove)$
        if $c_3 \neq \emptyset$ **then**
          $p.components \leftarrow [c_1, c_2, c_3]$
          $p.cost \leftarrow (|c_1| + |c_3|) * moveCost + |c_2| * pushCost$
          $p.minCost \leftarrow |c_2| * pushCost + |c_3| * moveCost$
          $p.o, p.d \leftarrow o, d$
          $P_{o,d} \leftarrow P_{o,d} \bigcup \{p\}$
          if $p.cost < p_{opt}.cost$ **then**
            $p_{opt} \leftarrow p$
          end if
        end if
      end if
      $seq \leftarrow seq + 1$
      $oSimPose \leftarrow oSimPose + one\_translation\_in\_d$
      $c_{3_{(Est)}} \leftarrow \{oSimPose, R_{goal}\}$
      $C_{est} \leftarrow (|c_1| + |c_{3_{(Est)}}|) * moveCost + seq * |one\_translation\_in\_d| * pushCost$
    end while
  end for
  return $p \in P_{o,d}$ with minimal $p.cost$ or null if $P_{o,d} = \emptyset$
end procedure

Algorithm 6 Execution loop taking our hypotheses into account.

```
1: procedure MAKE-AND-EXECUTE-PLAN(R_init, R_goal, I_init)
2:     ...                                   ▷ Initialization (lines 2 to 6 in Algorithm 3)
3:     I ← I_init
4:     p_opt.components ← [A*(R_init, R_goal, I.occGrid)]
5:     p_opt.cost ← |p_opt| ∗ moveCost
6:     while R ≠ R_goal do
7:         UPDATE-FROM-NEW-INFORMATION(I)
8:         if I.freeSpaceCreated then
9:             minCostL ← ∅
10:        end if
11:        O_new ← O_new ⋃ I.newObstacles
12:        O ← I.allObstacles
13:        isPlanNotColliding ← p_opt ⋂ O_new ≠ ∅
14:        isPushPoseValid ← True
15:        isManipSafe ← True
16:        isObstacleSame ← True
17:        if p_opt.o exists then            ▷ If p_opt includes the manipulation of an obstacle.
18:            isObstacleSame ← O[p_opt.o] = p_opt.o
19:            if not isObstacleSame then
20:                p_opt.o ← O[p_opt.o.id]                          ▷ Update the copy.
21:                p_opt.safeSweptArea ← GET-SAFE-SWEPT-AREA(p_opt.o, p_opt.translation, I)
22:                if p_opt.nextStepComponent is c_1 AND p_opt.pushPose ∉ p_opt.o.pushPoses then
23:                    isPushPoseValid ← False
24:                end if
25:            end if
26:            if p_opt.safeSweptArea ⋂ O ≠ ∅ then
27:                isManipSafe ← False
28:            end if
29:        end if
30:        if not(isPlanNotColliding AND isManipSuccess AND isManipSafe AND
   isPushPoseValid) then
31:            p_opt.components ← [A*(R, R_goal, I.occGrid)]
32:            p_opt.cost ← |p_opt| ∗ moveCost
33:            MAKE-PLAN(R, R_goal, I, O, blockedObsL, p_opt, euCostL, minCostL)
34:            O_new ← ∅
35:        end if
36:        ...                                   ▷ Execution (lines 24 to 34 in Algorithm 3)
37:    end while
38:    return True
39: end procedure
```

Algorithm 7 Obstacle evaluation subroutine taking our hypotheses into account.

```
1: procedure PLAN-FOR-OBSTACLE(o, p_opt, I, R, R_goal, blockedObsL)
2:     if o ∈ blockedObsL then
3:         return null
4:     end if
5:     P_o,d ← ∅
6:     for each pushPose in o.pushPoses do
7:         pushUnit ← (cos(pushPose.yaw), sin(pushPose.yaw))   ▷ Unit vector for push direction
8:         c_1 ← A*(R, pushPose, I.occGrid)                    ▷ c_1 is computed for each push pose
9:         if c_1 = ∅ then
10:            continue
11:        end if
12:        seq ← 1
13:        translation ← pushUnit ∗ onePushDist ∗ seq          ▷ onePushDist is a distance constant
14:        safeSweptArea ←GET-SAFE-SWEPT-AREA(o, translation, I.occGrid)
15:        oSimPose ←pushPose + translation
16:        c_3(Est) ← {oSimPose, R_goal}
17:        C_est ← (|c_1| + |c_3(Est)|) ∗ moveCost+|translation| ∗ o.pushCost
18:        while C_est ≤ p_opt.cost AND safeSweptArea ≠ null do
19:            c_2 ← {pushPose, oSimPose}
20:            c_3 ← A*(oSimPose, R_goal, I.withSimulatedObstacleMove)
21:            if c_3 ≠ ∅ then
22:                p.components ← [c_1, c_2, c_3]
23:                p.cost ← (|c_1| + |c_3|) ∗ moveCost + |c_2|∗o.pushCost
24:                p.minCost ← |c_2|∗o.pushCost+|c_3| ∗ moveCost
25:                p.o ← COPY(o)
26:                p.translation ← translation
27:                p.safeSweptArea ← safeSweptArea
28:                p.pushPose ← pushPose
29:                P_o,d ← P_o,d ⋃{p}
30:                if p.cost < p_opt.cost then
31:                    p_opt ← p
32:                end if
33:            end if
34:            seq ← seq + 1
35:            translation ← pushUnit ∗ onePushDist ∗ seq
36:            safeSweptArea ←GET-SAFE-SWEPT-AREA(o, translation, I)
37:            oSimPose ←pushPose + translation
38:            c_3(Est) ← {oSimPose, R_goal}
39:            C_est ← (|c_1| + |c_3(Est)|) ∗ moveCost+|translation| ∗ o.pushCost
40:        end while
41:    end for
42:    return p ∈ P_o,d with minimal p.cost or null if P_o,d = ∅
43: end procedure
```

3. Final pseudocode formulation of our propositions

Algorithm 16 Merged execution loop.

procedure MAKE-AND-EXECUTE-PLAN(R_{init}, R_{goal}, I_{init})
  $R \leftarrow R_{init}$
  $\mathcal{O}_{new} \leftarrow \emptyset$
  $isManipSuccess \leftarrow True$
  $blockedObsL \leftarrow \emptyset$
  $euCostL, minCostL \leftarrow \emptyset, \emptyset$
  $I \leftarrow I_{init}$
  $p_{opt}.components \leftarrow [A^*(R_{init}, R_{goal}, I.occGrid)]$
  $p_{opt}.cost \leftarrow |p_{opt}| * moveCost$
  $isObservable \leftarrow True$
  while $R \neq R_{goal}$ **do**
    UPDATE-FROM-NEW-INFORMATION(I)
    if I.freeSpaceCreated **then**
      $minCostL \leftarrow \emptyset$
    end if
    $\mathcal{O}_{new} \leftarrow \mathcal{O}_{new} \bigcup I.newObstacles$
    $\mathcal{O} \leftarrow I.allObstacles$
    $isPlanNotColliding \leftarrow p_{opt} \bigcap \mathcal{O}_{new} \neq \emptyset$
    $isBlockingObsMoved \leftarrow I.movedObstacles \neq \emptyset$
    $isPushPoseValid \leftarrow True$
    $isManipSafe \leftarrow True$
    $isObstacleSame \leftarrow True$
    if $p_{opt}.o$ exists **then**
      $isObstacleSame \leftarrow \mathcal{O}[p_{opt}.o] = p_{opt}.o$
      if not $isObstacleSame$ **then**
        $p_{opt}.o \leftarrow \mathcal{O}[p_{opt}.o.id]$
        $p_{opt}.safeSweptArea \leftarrow$ GET-SAFE-SWEPT-AREA($p_{opt}.o$, $p_{opt}.translation$, I)
        if $p_{opt}.nextStepComponent$ is c_1 AND $p_{opt}.pushPose \notin p_{opt}.o.pushPoses$ **then**
          $isPushPoseValid \leftarrow False$
        end if
      end if
      if $p_{opt}.safeSweptArea \bigcap \mathcal{O} \neq \emptyset$ **then**
        $isManipSafe \leftarrow False$
      end if
    end if
    $isPlanValid \leftarrow$ ($isPlanNotColliding$ AND $isManipSuccess$ AND $isManipSafe$ AND $isPushPoseValid$ AND $isObservable$)
    if not $isPlanValid$ **then**
      $p_{opt}.components \leftarrow [A^*(R, R_{goal}, I.occGrid)]$
      $p_{opt}.cost \leftarrow |p_{opt}| * moveCost$
    end if
    if not $isPlanValid$ OR ($isBlockingObsMoved$) **then**
      MAKE-PLAN(R, R_{goal}, I, $\mathcal{O}$, $blockedObsL$, p_{opt}, $euCostL$, $minCostL$)
      $isManipSuccess \leftarrow True$
      continue
    end if
    if $p_{opt}.components = \emptyset$ **then**
      return $False$
    end if
    $isManipSuccess \leftarrow True$
    $isObservable \leftarrow True$
    $R_{next} \leftarrow p_{opt}$.getNextStep()
    if $p_{opt}.o \neq$ null $p_{opt}.o.movableStatus = IS\_MAYBE\_MOVABLE$ AND **not** $isObstacleSame$ AND ($c_{next} = c_0$ OR $c_{next} = c_1$) **then**
      $fObsPose \leftarrow$ GET-FIRST-PATH-OBSPOSE($p_{opt}.o$, p_{opt}.get-c_0() + p_{opt}.get-c_1(), I)
      if $fObsPose =$ null **then**
        $isObservable \leftarrow False$
        continue
      end if
    end if
    $R_{real} \leftarrow$ ROBOT-GOTO(R_{next})
    if $p_{opt}.nextStepComponent$ is c_2 AND $R_{real} \neq R_{next}$ **then**
      $isManipSuccess \leftarrow False$
      $blockedObsL \leftarrow blockedObsL \bigcup p_{opt}.o$
    end if
    $R \leftarrow R_{real}$
  end while
  return $True$
end procedure

Algorithm 17 Merged obstacle evaluation subroutine

procedure PLAN-FOR-OBSTACLE(o, p_{opt}, I, R, R_{goal}, $blockedObsL$)
  if $o \in blockedObsL$ OR $o.movableStatus = IS\_NOT\_MOVABLE$ **then**
    return null
  end if
  $P_{o,d} \leftarrow \emptyset$
  for each $pushPose$ in $o.pushPoses$ **do**
    $pushUnit \leftarrow (cos(pushPose.yaw), sin(pushPose.yaw))$
    $c_1 \leftarrow A^*(R, pushPose, I.occGrid)$
    if $c_1 = \emptyset$ **then**
      continue
    end if
    $c_0 \leftarrow \emptyset$
    if $o.movableStatus = IS\_MAYBE\_MOVABLE$ **then**
      $obsPose \leftarrow$ GET-FIRST-PATH-OBSPOSE(o, c_1, I)
      if $obsPose \neq$ null **then**
        $c_0, c_1 \leftarrow c_1[c_1.firstPose : obsPose], c_1[obsPose : c_1.lastPose]$
      else
        OPT-COMPUTE-C0-C1(o, I, R, $pushPose$, c_0, c_1)
        if $c_0 = \emptyset$ OR $c_1 = \emptyset$ **then**
          continue
        end if
      end if
    end if
    $seq \leftarrow 1$
    $translation \leftarrow pushUnit * onePushDist * seq$
    $safeSweptArea \leftarrow$GET-SAFE-SWEPT-AREA(o, $translation$, I)
    $oSimPose \leftarrow pushPose + translation$
    $suppC_M \leftarrow$GET-OCC-COST(GET-OBS-POINTS(o, $translation$), $occCostGrid$)
    $c_{3_{(Est)}} \leftarrow \{oSimPose, R_{goal}\}$
    $C_{est} \leftarrow ((c_0 \neq \emptyset ? |c_0| : 0) + |c_1| + |c_{3_{(Est)}}|) * moveCost + |translation| * o.pushCost * suppC_M$
    while $C_{est} \leq p_{opt}.cost$ AND $safeSweptArea \neq$ null **do**
      $c_2 \leftarrow \{pushPose, oSimPose\}$
      $c_3 \leftarrow A^*(oSimPose, R_{goal}, I.withSimulatedObstacleMove)$
      if $c_3 \neq \emptyset$ **then**
        $p.components \leftarrow c_0 \neq \emptyset$? $[c_0, c_1, c_2, c_3] : [c_1, c_2, c_3]$
        $p.cost \leftarrow ((c_0 \neq \emptyset ? |c_0| : 0) + |c_1| + |c_3|) * moveCost + |c_2| * o.pushCost * suppC_M$
        $p.minCost \leftarrow |c_2| * o.pushCost * suppC_M + |c_3| * moveCost$
        $p.o \leftarrow$ COPY(o)
        $p.translation \leftarrow translation$
        $p.safeSweptArea \leftarrow safeSweptArea$
        $p.pushPose \leftarrow pushPose$
        $P_{o,d} \leftarrow P_{o,d} \bigcup \{p\}$
        if $p.cost < p_{opt}.cost$ **then**
          $p_{opt} \leftarrow p$
        end if
      end if
      $seq \leftarrow seq + 1$
      $translation \leftarrow pushUnit * onePushDist * seq$
      $safeSweptArea \leftarrow$GET-SAFE-SWEPT-AREA(o, $translation$, I)
      $oSimPose \leftarrow pushPose + translation$
      $suppC_M \leftarrow$GET-OCC-COST(GET-OBS-POINTS(o, $translation$), $occCostGrid$)
      $c_{3_{(Est)}} \leftarrow \{oSimPose, R_{goal}\}$
      $C_{est} \leftarrow ((c_0 \neq \emptyset ? |c_0| : 0) + |c_1| + |c_{3_{(Est)}}|) * moveCost + |translation| * o.pushCost * suppC_M$
    end while
  end for
  return $p \in P_{o,d}$ with minimal $p.cost$ or null if $P_{o,d} = \emptyset$
end procedure

Algorithm 10 Subroutine for getting the first pose in a *path* that allows identification of o, it it exists.

```
1: procedure GET-FIRST-PATH-OBSPOSE(o, path, I)
2:     for each pose in path do
3:         if IS-OBS-IN-FOV-FOR-POSE(o, pose, I) then
4:             return pose
5:         end if
6:     end for
7:     return null
8: end procedure
```

Algorithm 12 Optimized subroutine for computing c_0 and c_1 if c_1 is not already valid.

```
1: procedure OPT-COMPUTE-C0-C1(o, I, R, pushPose, c_0, c_1)
2:     c_1 ← ∅
3:     totalCost ← +∞
4:     euPosesCostL ← ∅                    ▷ Sort observation poses by ascending heuristic cost.
5:     for each obsPose in o.obsPoses do
6:         euPosesCostL.insert({obsPose, |R, obsPose| + |obsPose, pushPose|})
7:     end for
8:     if euPosesCostL ≠ ∅ then
9:         op_next ← euPosesCostL[0]
10:        while totalCost ≥ op_next.cost AND op_next ≠ null do
11:            c_0cur ← A*(R, op_next.obsPose, I.occGrid)
12:            c_1cur ← A*(op_next.obsPose, pushPose, I.occGrid)
13:            newTotalCost = |c_0cur| + |c_1cur|
14:            if newTotalCost < +∞ AND (newTotalCost < totalCost OR (newTotalCost =
    totalCost AND |c_0cur| < |c_0|) then
15:                c_0 = c_0cur
16:                c_1 = c_1cur
17:                totalCost = |c_0| + |c_1|
18:            end if
19:            op_next ← euPosesCostL.getNext()
20:        end while
21:    end if
22: end procedure
```

Algorithm 14 Obstacle evaluation subroutine modified for considering placement.

```
1: procedure GET-OCC-COST(simOccPoints, occCostGrid)
2:     VALUE_RANGE ← (FORBIDDEN_VALUE − ALLOWED_VALUE)
3:     cost ← 1
4:     for each point in simOccPoints do
5:         valueForPoint ← occCostGrid[point]
6:         if valueForPoint = FORBIDDEN_VALUE then
7:             return +∞
8:         else if valueForPoint = ALLOWED_VALUE then
9:             valueForPoint ← 0
10:        end if
11:        cost ← cost + valueForPoint/VALUE_RANGE
12:    end for
13:    return cost
14: end procedure
```

4. Illustrations of Pepper’s characteristics

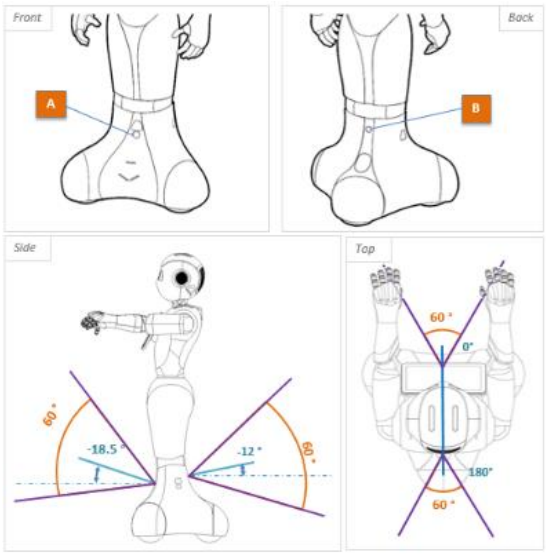


Illustration 7: Position and field of vision of Pepper's sonars

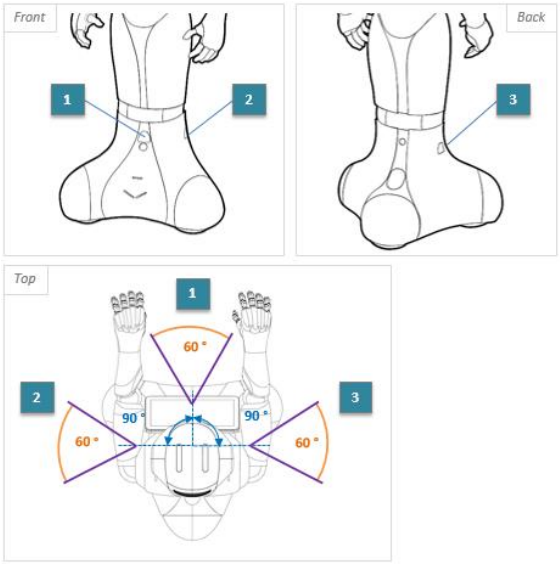


Illustration 8: Position and field of vision of Pepper's lasers

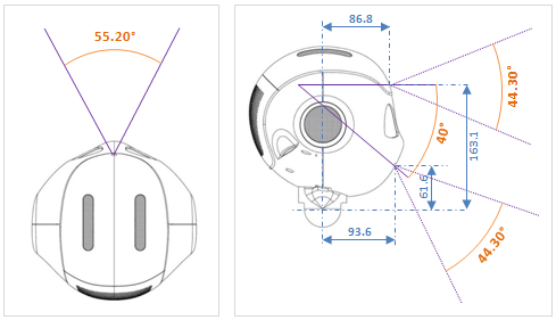


Illustration 9: Position and field of vision of Pepper's 2D camera

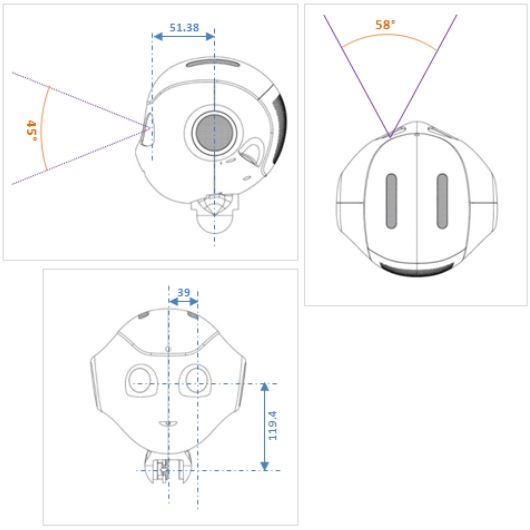
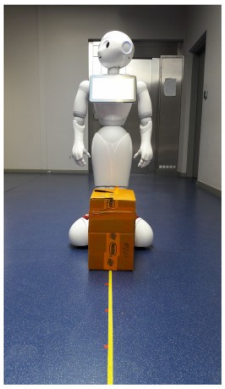


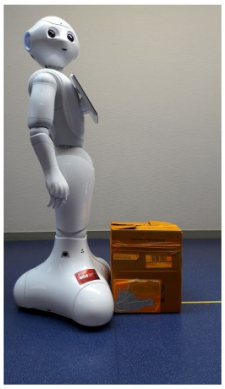
Illustration 10: Position and field of vision of Pepper's 3D camera

5. Photos of the experimentation testing pushes on obstacles

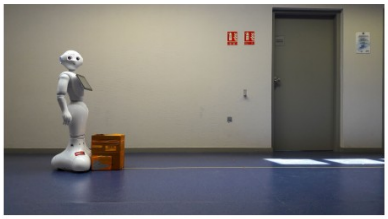
(A) Front close angle



(B) Front wide angle



(C) Side close angle

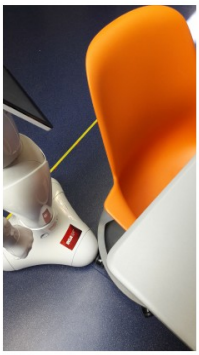


(D) Side wide angle

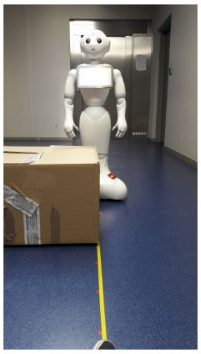
(E) Top close angle

Illustration 11: Experimental setup example with a small cardboard box

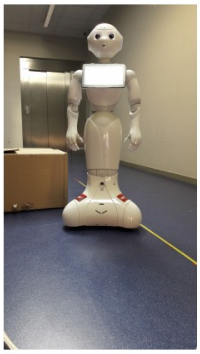
(A) Initial top close angle of well-positioned robot and chair



(B) Final top close angle of the failed chair push attempt



(C) Initial front wide angle of badly-positioned robot and box



(D) Final front wide angle of the failed box push attempt

Illustration 12: Failed manipulation attempts with badly positioned robot and a car-board box, and with well-positioned robot and a chair with wheelcasters.

6. Example simulation results

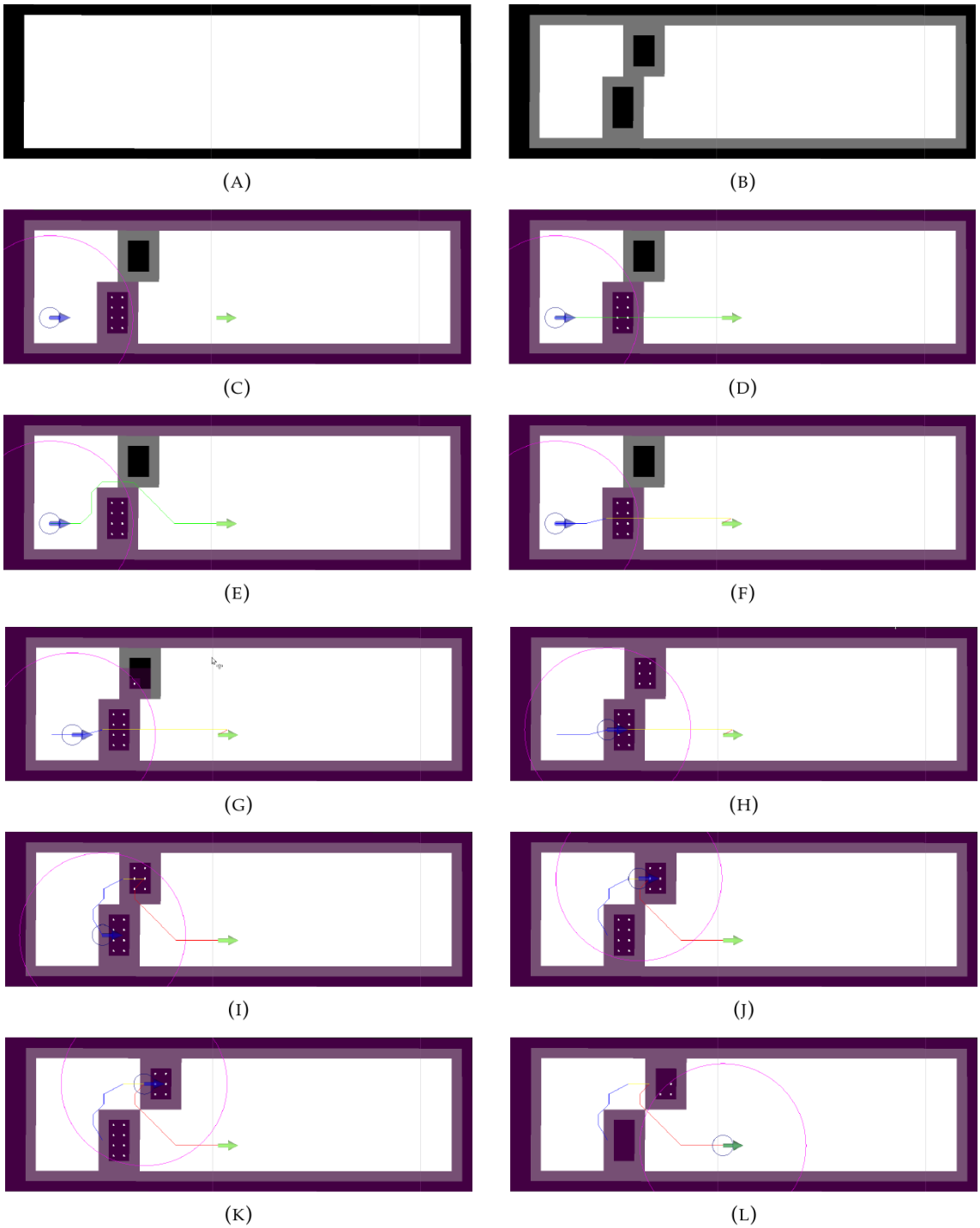


Illustration 13: Simple simulation in a corridor with a movable (top one) and an un-movable (bottom one) obstacles with our algorithm as presented in section 4. (A) represents the static obstacles that never change (walls). (B) also represents the yet unknown obstacles (the bottom one is unmovable, the top one is movable). (C) is our initial robot configuration. The robot is geometrically represented by a little blue circle and a blue arrow for its orientation, and it's field of vision (represented by the pink circle) allows it to detect the point cloud (white point) that represents the bottom obstacle (known environment to the robot is represented by the purple shades). In (D), the robot starts by considering a plan ignoring all obstacles, except the static ones (because we made the hypothesis that it knows about them). Then, in (E), it takes into account its current knowledge of the environment to see that its previous best plan is invalid and sets its new best plan as the one that avoids the obstacle. In (F), the robot evaluates the bottom obstacle and finds a better plan pushing the obstacle than the one avoiding it, and starts executing this plan in (G), where we see that is starts detecting the shape of the top obstacle, only to realize in (H) that the obstacle won't budge. Therefore, it invalidates the current plan, and computes a new one (seen in (I)) that implies moving the top obstacle directly, since no plan avoiding the two obstacles could be found. In Figures (J) and (K), the robot moves the top obstacle, to finally reach its goal in (L).